

Creación de aplicaciones con JBuilder®

JBuilder® X

Borland Software Corporation
100 Enterprise Way
Scotts Valley, California 95066-3249
www.borland.com

Consulte el archivo `deploy.html` ubicado en el directorio `redist` de su producto JBuilder para acceder a una lista completa de archivos que puede distribuir de acuerdo con su Declaración de licencia y Garantía limitada de JBuilder.

Borland Software Corporation puede tener patentes y/o solicitudes de patente pendientes en lo relativo a los asuntos tratados en este documento. Consulte el CD del producto o el cuadro de diálogo Acerca de para ver la lista de patentes aplicables. La entrega de este documento no supone la cesión de ninguna licencia sobre estas patentes.

Copyright © 1997–2004 Borland Software Corporation. Reservados todos los derechos. Todas las marcas y nombres de productos Borland son marcas comerciales o registradas de Borland Software Corporation en los Estados Unidos y en otros países. Las demás marcas pertenecen a sus respectivos propietarios.

Para conocer las condiciones y limitaciones de responsabilidad de terceros, consulte las Notas de la versión en el CD de JBuilder.

Impreso en los EEUU.

JXE0010WW21005bajb 9E12R1103
0304050607-9 8 7 6 5 4 3 2 1
PDF

Índice de materias

Capítulo 1

Introducción

1-1

Tema	1-2
Tutoriales:	1-3
Apéndices	1-4
Convenciones de la documentación	1-5
Asistencia y recursos para desarrolladores	1-7
El servicio de asistencia técnica de	
Borland	1-7
Recursos en línea	1-8
World Wide Web	1-8
Grupos de noticias de Borland	1-8
Usenet, grupos de noticias	1-8
Información sobre errores	1-9

Capítulo 2

Creación y gestión del proyecto

2-1

Creación de proyectos	2-2
Creación de proyectos con ayuda del	
Asistente para proyectos	2-2
Selección del nombre del proyecto y la	
plantilla	2-3
Configuración de las vías de acceso	
del proyecto	2-4
Configuración de las opciones	
generales del proyecto	2-5
Creación de proyectos a partir de archivos	
anteriores	2-7
Selección del directorio fuente y el nombre	
del nuevo proyecto de JBuilder.	2-8
Presentación de los archivos	2-9
Cambio entre archivos	2-10
Visualización de recopilatorios desde el	
panel de proyecto	2-10
Almacenamiento de proyectos	2-11
Apertura de proyectos y archivos	2-11
Abrir archivos.	2-11
Cómo abrir de nuevo proyectos y	
archivos.	2-12
El Visualizador de archivos	2-13
Favoritos	2-13
Gestión de proyectos	2-14
Cómo añadir elementos a un proyecto.	2-14
Más información sobre la adición de	
carpetas	2-15

Importación de código fuente a un

proyecto.	2-15
Adición de archivos y paquetes con	
detección automática de la vía de	
acceso a archivos fuente	2-17
Adición de archivos y paquetes sin	
detección automática de la vía de	
acceso a archivos fuente	2-17
Creación de un archivo fuente Java	2-17
Eliminación de carpetas, archivos, clases	
y paquetes del proyecto	2-19
Eliminación de archivos	2-20
Apertura de archivos fuera de un	
proyecto	2-20
Cambio de nombre	2-20
Adición de una vista de directorio	2-21
Personalización de las vistas de	
directorio	2-22
Definición de las propiedades del proyecto.	2-23
Configuración del JDK	2-24
Modificación del JDK	2-24
Depuración con -classic	2-25
Cambiar el JDK.	2-25
Configuración de los JDK	2-27
Configuración de vías de acceso a	
bibliotecas necesarias	2-27
Trabajo con varios proyectos	2-28
Cambio de un proyecto a otro	2-29
Guardar varios proyectos	2-29
Visualización de recopilatorios desde el	
panel de proyecto	2-29
Información adicional acerca de los	
proyectos	2-29

Capítulo 3

Los grupos de proyectos

3-1

Creación de grupos de proyectos.	3-1
Adición y eliminación de proyectos de los	
grupos de proyectos.	3-3
Cambio del orden de un grupo de proyectos	3-4
Desplazamiento por los grupos de proyectos	3-4
Adición de proyectos como bibliotecas	
necesarias.	3-4

Capítulo 4	
Gestión de las vías de acceso	4-1
Las bibliotecas	4-1
Adición y configuración de bibliotecas	4-2
Modificación de bibliotecas	4-4
Adición de proyectos como bibliotecas	
necesarias	4-5
Presentación de la lista de bibliotecas	4-5
Paquetes	4-6
Ubicación del archivo .java = vía de acceso	
a archivos fuente + vía de acceso a	
paquetes	4-7
Ubicación del archivo .class = vía de	
salida + vía de acceso a paquetes	4-7
Utilización de paquetes en JBuilder	4-8
Directrices de nomenclatura de paquetes	4-9
Cómo construye JBuilder las vías de acceso	4-9
Vía de acceso a archivos fuente	4-9
Vía de salida	4-10
Vía de acceso a clases	4-10
Vía de búsqueda	4-11
Vía de acceso a documentos	4-11
Vía de acceso a las copias de seguridad	4-12
Directorio de trabajo	4-12
Localización de los archivos	4-12
Cómo encuentra JBuilder los archivos al	
profundizar	4-13
Cómo encuentra JBuilder los archivos al	
compilar	4-13
Cómo encuentra JBuilder los archivos de	
clase al ejecutar o depurar	4-13
Capítulo 5	
Compilación de programas en Java	5-1
Comprobación inteligente de dependencias	5-2
Compilación de un programa	5-3
Menús de generación de JBuilder	5-4
Generación de proyectos con el	
comando Ejecutar	5-5
Mensajes de error	5-5
Problemas de compilación al abrir	
proyectos	5-6
Comprobación de correspondencia entre	
paquetes y directorios	5-7
Definición de opciones del compilador	5-7
Definición de un compilador	5-9
Configuración de las opciones adicionales	
de compilación y generación	5-9
Configuración de la vía de salida	5-10

Compilación de proyectos en un grupo de	
proyectos	5-11
Compilación desde la línea de comandos	5-11
bmj (Make de Borland para Java)	5-11
bcj (Compilador de Borland para Java)	5-12
Creación de proyectos desde la línea de	
comandos	5-12
Cambio entre la línea de comandos y	
el IDE	5-12

Capítulo 6

Generación de programas en Java 6-1

El sistema de generación de JBuilder	6-2
Términos del sistema de generación	6-2
Fases de generación	6-3
El comando Ejecutar Make comandos	6-4
El comando Generar de nuevo	6-5
El comando Limpiar	6-5
Utilización del panel de mensajes	6-6
Generación en segundo plano	6-7
Asignación de valores a propiedades de	
generación	6-7
Configuración de las preferencias de	
generación	6-8
Generación de grupos de proyectos	6-9
Definición del orden de generación de un	
grupo de proyectos	6-9
Generación de un grupo de proyectos	6-10
Adición de tipos de generación de	
grupos de proyectos al menú Proyecto	6-11
Generación con archivos Ant	6-12
Adición de archivos de generación Ant	
a los proyectos	6-13
Adición de archivos Ant con el Asistente	
para Ant	6-13
Adición manual de archivos Ant	6-14
Creación y modificación de archivos de	
generación Ant	6-15
Especificación de las vías de acceso en	
los archivos de generación Ant	6-16
Desplazamiento por archivos de	
generación Ant	6-18
Generación de proyectos en Ant	6-19
Definición del JDK	6-20
Adición de bibliotecas	6-20
Generación de proyectos Ant con el	
comando Ejecutar	6-22
Configuración de las propiedades de Ant	6-23
Opciones Ant	6-25

Importación de proyectos Ant	6-25	Gestión de las configuraciones de ejecución	7-11
Exportación de proyectos de JBuilder a Ant	6-25	Creación y modificación de configuraciones de ejecución	7-12
Exportación de proyectos con el Asistente Exportar a Ant	6-26	Selección de tipos de generación	7-12
Tareas de generación de JBuilder no admitidas	6-28	Tipos de configuraciones de ejecución	7-14
Exportación de EJB a Ant	6-28	Creación de configuraciones de ejecución	7-15
Configuración de las propiedades de Exportar a Ant	6-29	Modificación de configuraciones de ejecución	7-17
Creación de tareas externas de generación	6-30	Ejecución de programas desde la línea de comandos	7-17
Asistente para tareas externas de generación	6-30	Ejecución de programas distribuidos desde la línea de comandos	7-18
Generación de tareas externas	6-31		
Configuración de propiedades de las tareas externas de generación	6-32		
Generación de archivos SQLJ	6-32		
Configuración del menú Proyecto	6-34		
Configuración del menú Proyecto para proyectos	6-34		
Configuración del menú Proyecto para grupos de proyectos	6-36		
Recopilación automática de paquetes fuente	6-38		
Filtrado de paquetes	6-39		
Exclusión de paquetes	6-40		
Inclusión de paquetes	6-40		
Generar de nuevo sin filtros	6-41		
Copia selectiva de los recursos	6-42		
Propiedades de recursos individuales	6-42		
Opciones específicas de archivos	6-43		
Opciones específicas del proyecto	6-44		
Adición de tipos de archivos no reconocidos como archivos de recursos genéricos	6-44		
La ficha Recursos del cuadro de diálogo Propiedades de proyecto	6-45		
Capítulo 7		Capítulo 8	
Ejecución de programas en JBuilder	7-1	Depuración de programas en Java	8-1
Ejecución de proyectos	7-3	Tipos de errores	8-2
Ejecución de proyectos individuales	7-3	Errores de ejecución	8-2
Ejecución de proyectos agrupados	7-4	Errores lógicos	8-3
Ejecución de OpenTools	7-5	Descripción general del proceso de depuración	8-3
Ejecución de archivos	7-6	Creación de una configuración de ejecución	8-4
Ejecución de archivos .java	7-6	Compilación del proyecto con información simbólica de depuración	8-4
Ejecución de archivos Web	7-7	Inicio del Depurador	8-6
Definición de las configuraciones de ejecución	7-8	Inicio del depurador con la opción -classic	8-7
		Ejecución bajo el control del depurador	8-7
		Pausar la ejecución del programa	8-8
		Finalización de una sesión de depuración	8-8
		Interfaz del depurador	8-9
		Sesiones de depuración	8-9
		Vistas del depurador	8-10
		Vista Salida, entrada y errores de consola	8-11
		Vista Clases con inspección desactivada	8-12
		Vista Puntos de interrupción de datos y código	8-14
		Vista Hilos, pilas de llamada y datos	8-16
		Vista Puntos de observación de datos	8-21
		Vista Clases cargadas y datos estáticos	8-24
		Vista Monitores de sincronización	8-27
		Vista Personalizar	8-28
		Barra de herramientas del depurador	8-29
		Métodos abreviados para el depurador	8-30

ExpressionInsight	8-31	Definición de las propiedades de los puntos de interrupción	8-56
Ayuda inmediata	8-32	Definición de las acciones de puntos de interrupción.	8-57
Depuración de código fuente ajeno a Java	8-32	Detención de la ejecución del programa	8-57
Control de la ejecución del programa	8-33	Registro de mensajes	8-58
Ejecución e interrupción del programa.	8-33	Creación de puntos de interrupción condicionales.	8-59
Reinicio del programa	8-34	Definición de la condición de punto de interrupción	8-59
El punto de ejecución	8-34	Utilización de puntos de interrupción por número de pasadas.	8-60
Definición del punto de ejecución.	8-35	Desactivación y activación de puntos de interrupción.	8-60
Gestión de hilos	8-36	Eliminación de puntos de interrupción.	8-61
Utilización del panel dividido	8-36	Ubicación de los puntos de interrupción de línea.	8-61
Presentación del hilo actual.	8-36	Examen de los valores de datos del programa	8-62
Presentación del marco de pila superior	8-37	Presentación de las variables en el depurador	8-62
Elección de un hilo para inspeccionarlo	8-37	Cambio de valores de datos.	8-64
Interrupción prolongada de un hilo	8-37	Presentación de objetos como una Cadena.	8-66
Detección de conflictos	8-38	Utilización de un visualizador personalizado en un objeto	8-67
Desplazamiento a través del código	8-38	Observación de expresiones	8-70
Inspección de código de llamadas a métodos	8-39	Puntos de observación de variables	8-71
Omisión de inspección en las llamadas a métodos	8-39	Puntos de observación de objetos	8-73
Salida de métodos	8-39	Edición de un punto de observación	8-74
Paso inteligente	8-40	Eliminación de puntos de observación	8-74
Ejecución hasta un punto de interrupción	8-41	Evaluación y modificación de expresiones	8-74
Ejecución hasta el final de un método	8-41	Evaluación de expresiones.	8-74
Ejecución hasta la posición del cursor	8-41	Evaluación de las llamadas a métodos.	8-75
Visualización de llamadas a métodos	8-42	Modificación de los valores de las variables	8-75
Localización de una llamada a un método	8-42	Modificación del código durante la depuración	8-76
Determinación de las clases que se han de inspeccionar	8-43	Actualización de todos los archivos de clase	8-76
Inspección de clases cuando el archivo fuente no está disponible	8-45	Actualización de archivos de clase individuales.	8-78
Puntos de interrupción y configuración de Inspección desactivada	8-46	Restablecimiento del punto de ejecución	8-78
Puntos de interrupción	8-46	Opciones para modificar el código	8-78
Definición de puntos de interrupción	8-47	Personalización del depurador	8-80
Definición de puntos de interrupción de línea	8-48	Personalización de la presentación del depurador	8-80
Definición de puntos de interrupción por excepción	8-49		
Definición de puntos de interrupción de clase.	8-51		
Definición de puntos de interrupción de método	8-52		
Definición de puntos de interrupción de campo	8-53		
Configuración de un punto de i nterrupción interprocesal	8-54		

Configuración de las opciones de depuración	8-81	Comprobación de la validez de un JavaBean	10-22
Definición de los intervalos de actualización	8-82	Instalación de beans en la paleta de componentes	10-23
Capítulo 9		Capítulo 11	
Depuración remota	9-1	Presentación de código con UML	11-1
Apertura y depuración de programas en un equipo remoto	9-2	Java y UML	11-2
Depuración de un programa que se ejecuta en un ordenador remoto	9-6	Términos de Java y UML	11-2
Depuración del código local que se ejecuta en un proceso independiente	9-9	JBuilder y UML	11-4
Depuración con puntos de interrupción interprocesales.	9-10	Diagramas limitados a las dependencias de paquetes	11-5
		Diagramas combinados de clases.	11-6
		Glosario de los diagramas UML de JBuilder	11-9
		Iconos de accesibilidad.	11-11
		Presentación de los diagramas UML	11-13
		El visualizador UML de JBuilder.	11-14
		Presentación de diagramas de paquetes	11-15
		Presentación de diagramas de clases.	11-15
		Presentación de las clases internas.	11-15
		Presentación del código fuente	11-16
		Visualización de Javadoc	11-16
		Uso del menú contextual	11-17
		Desplazamiento de la vista	11-17
		Actualización de la vista	11-18
		Desplazamiento por diagramas.	11-18
		UML y el panel de estructura	11-18
		Diagramas de paquetes	11-19
		Diagramas de clases	11-19
		Personalización de diagramas UML	11-20
		Definición de las propiedades del proyecto	11-20
		Filtrado de paquetes y clases	11-20
		Inclusión de referencias de bibliotecas de proyecto	11-21
		Inclusión de referencias del código generado	11-22
		Configuración de preferencias UML	11-22
		Creación de imágenes de diagramas UML	11-23
		Impresión de diagramas UML	11-23
		Perfeccionamiento y Buscar referencias	11-24
Capítulo 10		Capítulo 12	
Creación de JavaBeans con BeansExpress	10-1	Comparación de archivos y versiones	12-1
Definición de JavaBean	10-1	Glosario de gestión de versiones	12-1
¿Por qué desarrollar JavaBeans?.	10-2	Comparación de dos archivos	12-2
Generación de clases bean	10-2		
Diseño de la interfaz de usuario de un bean	10-4		
Adición de propiedades a un bean	10-4		
Modificación de propiedades.	10-7		
Eliminación de propiedades	10-8		
Adición de propiedades monitorizables y restringidas	10-8		
Creación de una clase BeanInfo	10-9		
Especificación de datos BeanInfo de una propiedad	10-10		
Utilización del diseñador de BeanInfo	10-10		
Modificación de clases BeanInfo.	10-11		
Adición de sucesos a un bean	10-12		
Activación de sucesos	10-12		
Monitorización de sucesos	10-15		
Creación de un conjunto de sucesos personalizado.	10-16		
Creación de editores de propiedades	10-18		
Creación de un editor de lista de cadenas	10-18		
Creación de un editor de listas de etiquetas de cadena	10-19		
Creación de un editor de listas de etiquetas de enteros	10-20		
Creación de editores de propiedades basados en componentes personalizados	10-21		
Admisión de serialización	10-22		

Utilización de etiquetas locales en la gestión de revisiones de archivos locales	12-4
La vista Histórico	12-6
ficha Contenido	12-7
Ficha Diferencias.	12-9
Ficha Información	12-10
Ficha Conflictos en la fusión	12-11

Capítulo 13

Perfeccionamiento de código 13-1

Detección de diferencias antes del perfeccionamiento	13-2
Búsqueda de definiciones	13-2
Búsqueda de métodos redefinidos	13-3
Búsqueda de referencias locales	13-3
Búsqueda de referencias	13-4
Perfeccionamiento en JBuilder	13-6
Perfeccionamiento de EJB	13-7
Presentación previa de los cambios antes del perfeccionamiento	13-7
Finalización del perfeccionamiento	13-9
Cómo deshacer un perfeccionamiento.	13-10
Almacenamiento de perfeccionamientos.	13-11
Ejecución de perfeccionamiento	13-11
Optimizar importaciones	13-11
Optimización de las importaciones	13-14
Perfeccionamientos por cambio de nombre	13-14
Cambio de nombre de paquetes	13-14
Cambio de nombre de clases, clases internas o interfaces.	13-15
Cambio de nombre de métodos	13-16
Cambio de nombre de variables locales.	13-17
Cambio de nombre de campos	13-18
Cambio de nombre de propiedades	13-19
Desplazamiento de una clase a un paquete diferente	13-19
Cambio de parámetros de métodos	13-20
Extracción de métodos.	13-22
Introducción de variables.	13-24
Introducción de campos	13-25
Perfeccionamiento con sentencia try/catch	13-26
Subida de métodos	13-27
Bajada de métodos	13-28
Subida de campos	13-29
Bajada de campos	13-30
Extracción de interfaces	13-31

Introducción de superclases.	13-32
--------------------------------------	-------

Capítulo 14

Test de módulos 14-1

JUnit	14-1
Cactus	14-2
Funciones de test de módulos de JBuilder	14-3
Detección de tests	14-3
Recopilador de tests de JUnit	14-4
Creación de tests y conjuntos de tests JUnit	14-5
El Asistente para tests.	14-6
Adición de código a los tests	14-6
El Asistente para conjuntos de tests.	14-7
El Asistente para clientes de prueba EJB	14-8
Montajes para tests predefinidos	14-8
Montaje JDBC	14-9
Montaje JNDI	14-10
Montaje para comparación	14-10
Creación de un montaje para tests personalizado	14-11
Utilización de Cactus	14-11
Asistente para la configuración de Cactus	14-12
Creación de un test Cactus para los Enterprise JavaBean	14-13
Ejecución de test Cactus	14-13
Ejecución de tests	14-14
JBTestRunner	14-15
Jerarquía del test	14-16
Fallos del test.	14-16
Resultados del test	14-16
JUnit TextUI	14-16
JUnit SwingUI	14-16
Configuraciones de ejecución para pruebas	14-17
Definición del filtro de seguimiento de la pila de tests	14-17
Depuración de tests.	14-18

Capítulo 15

Creación de Javadoc a partir de archivos fuente 15-1

Adición de comentarios Javadoc a los archivos fuente	15-2
Colocación de los comentarios Javadoc	15-3
Etiquetas Javadoc	15-4
Generación automática de comentarios y etiquetas Javadoc	15-7

El cuadro de diálogo Javadoc en la generación y modificación de comentarios y etiquetas	15-7	¿El programa depende de funciones JDK 1.1 o Java 2 (JDK 1.2 y superior)?	16-6
Las plantillas de código en la generación de comentarios y etiquetas.	15-9	¿Tiene el usuario bibliotecas Java instaladas en su ordenador?	16-6
JavadocInsight	15-10	¿Se trata de un applet o de una aplicación?	16-7
Etiquetas @todo Javadoc	15-13	Tiempo de descarga de archivos	16-8
Visualización de etiquetas @todo	15-13	Distribución rápida	16-8
Creación de etiquetas Javadoc personalizadas	15-14	Aplicaciones	16-9
Conflictos en comentarios Javadoc	15-17	Applets.	16-10
Generación del nodo de documentación	15-17	JavaBeans.	16-11
Selección del formato de la documentación	15-18	Sugerencias de distribución	16-12
Selección de opciones para generar documentación	15-19	Configuración del entorno de trabajo	16-12
Selección de paquetes que se han de documentar.	15-21	Distribución en Internet	16-13
Especificación de opciones de línea de comandos para el doclet	15-23	Distribución de aplicaciones distribuidas.	16-13
Creación de archivos de salida	15-26	Redistribución de las clases que se suministran con JBuilder	16-13
Creación de archivos adicionales	15-28	Información adicional de distribución	16-15
Archivos de paquetes.	15-28	Utilización del Creador de compiladores	16-16
Archivos de comentarios de aspectos generales	15-30	El Creador de compiladores y los recursos	16-16
Visualización de Javadoc	15-30	Seleccione un tipo de compilador	16-16
Apariencia de Javadoc en JBuilder	15-32	Indique el archivo que va a crear el proceso de compilación	16-17
Mantenimiento de Javadoc	15-32	Indique los contenidos del compilador	16-19
Cambio de propiedades para el nodo de documentación	15-33	Añadir filtros	16-19
Cambio de las propiedades del nodo	15-33	Modificación y eliminación de filtros	16-21
Modificación de propiedades Javadoc	15-34	Adición de archivos.	16-21
Modificación de propiedades doclet	15-34	Seleccione los archivos descriptores de la distribución	16-22
Creación de un archivo compilador de documentación	15-35	Especificación del contenido de un compilador de adaptador de recursos	16-22
Creación de un doclet personalizado	15-37	Indique cómo tratar las dependencias de las bibliotecas.	16-23
Capítulo 16		Configurar opciones del descriptor del compilador	16-24
Distribución de programas en Java		Seleccione un método para indicar la clase principal de la aplicación.	16-25
16-1		Determinar qué ejecutables se van a generar.	16-26
Distribución de archivos compiladores de Java (JAR)	16-2	Ejecución de ejecutables.	16-26
Conceptos básicos acerca del archivo descriptor	16-3	Configuración de las opciones de depuración	16-27
Estrategias de distribución	16-4	Generación de archivos compiladores	16-28
Aspectos relativos a la distribución	16-5	Los nodos de compiladores.	16-29
¿Está todo lo necesario en la vía de acceso a clases?	16-6	Presentación del archivo compilador y del archivo descriptor	16-29

Modificación de las propiedades de los nodos de compiladores	16-30
Eliminación, borrado y asignación de nombres a compiladores	16-30
Creación de ejecutables con el Creador de ejecutables nativos	16-31
Personalización de archivos de configuración ejecutables	16-32
Inicio de la MV	16-34
Requisitos del archivo de configuración	16-34
Directivas	16-34

Capítulo 17

Internacionalización de programas con JBuilder 17-1

Términos y definiciones de internacionalización	17-2
Funciones de internacionalización de JBuilder	17-3
Aplicación de ejemplo multilingüe	17-3
Eliminación de cadenas no modificables (hard-coded) incluidas en el código	17-5
Utilización del Asistente para extracción de recursos	17-5
Utilización del cuadro de diálogo Propiedad localizable	17-8
Funciones de internacionalización de dbSwing	17-9
Componentes que identifican la versión localizada	17-10
Los componentes de JBuilder muestran todos los caracteres Unicode	17-10
Funciones de internacionalización del diseñador de interfaz de usuario	17-11
Unicode en el Depurador IDE	17-12
Elección de una codificación nativa para el compilador	17-13
Definición de la opción de codificación	17-13
Adición y redefinición de codificaciones	17-14
Otros asuntos relacionados con las codificaciones nativas	17-14
Formato Unicode de 16 bits	17-15
Utilización de Unicode mediante ASCII y '\u'	17-15
JBuilder en el mundo	17-16
Asistencia internacional en línea	17-16

Capítulo 18

Tutorial: Creación de un proyecto con un archivo de generación Ant 18-1

Paso 1: Crear un proyecto y una aplicación	18-2
Paso 2: Crear el archivo de generación Ant	18-2
Paso 3: Ejecutar tipos de generación individuales	18-4
Paso 4: Ejecutar el objetivo por defecto	18-6
Paso 5: Tratamiento de errores con Ant	18-7
Paso 6: Añadir un tipo de generación al menú Proyecto	18-7
Paso 7: Configuración de las propiedades de Ant	18-8
Cómo añadir tareas Ant personalizadas a su proyecto	18-10

Capítulo 19

Tutorial: La vista Histórico 19-1

Paso 1: Generación de varias versiones de un archivo	19-2
Paso 2: La ficha Contenido	19-4
Paso 3: La ficha Diferencias	19-7
Ordenación de las revisiones	19-7
Cómo deshacer cambios	19-8
Comparación inteligente	19-9
Paso 4: La ficha Información	19-10

Capítulo 20

Tutorial: Compilación, ejecución y depuración 20-1

Paso 1: Abrir el proyecto de ejemplo	20-2
Paso 2: Solucionar errores de sintaxis	20-3
Paso 3: Solucionar errores de compilación	20-4
Paso 4: Ejecutar el programa	20-8
Ejecución del programa	20-10
Paso 5: Corregir el método subtractValues()	20-10
Almacenamiento de archivos y ejecución del programa	20-17
Paso 6: Corregir el método divideValues()	20-18
Almacenamiento de archivos y ejecución del programa	20-21
Paso 7: Corregir el método oddEven()	20-21
Paso 8: Buscar excepciones producidas durante la ejecución	20-24

Capítulo 21	
Tutorial: Depuración remota	21-1
Paso 1: Abrir el proyecto de ejemplo	21-2
Paso 2: Definir las configuraciones de ejecución y depuración	21-3
Paso 3: Definir los puntos de interrupción.	21-6
Paso 4: Compilar el servidor y copiar los archivos de clase del servidor al equipo remoto	21-9
Paso 5: Iniciar el registro de RMI y el servidor en el equipo remoto	21-10
Paso 6: Iniciar el proceso de servidor remoto, el cliente en modo depuración e inspeccionar el punto de interrupción interprocesal	21-12

Capítulo 22	
Tutorial: Visualización de código con el visualizador UML	22-1
Paso 1: Compilación del ejemplo	22-2
Paso 2: Visualización de un diagrama de paquete UML	22-3
Paso 3: Visualización de un diagrama de clase UML	22-6
Paso 4: Añadir referencias de bibliotecas.	22-9
Paso 5: Filtrado de diagramas UML.	22-12

Capítulo 23	
Tutorial: Creación y ejecución de tests y de conjuntos de tests	23-1
Paso 1: Apertura de proyectos	23-2
Paso 2: Creación de montajes para tests	23-2
Paso 3: Implementación de un método de test que lanza una excepción esperada	23-3
Visualización de la salida de fallo del test	23-4
Corrección del test para que pase	23-5
Paso 4: Creación de un segundo método de test	23-5
Paso 5: Creación de un conjunto de tests.	23-5
Paso 6: Ejecución de tests	23-7

Capítulo 24	
Tutorial: Utilización de montajes para tests	24-1
Paso 1: Creación de proyectos	24-2
Paso 2: Creación de un módulo de datos.	24-2

Paso 3: Creación de un montaje de comparación.	24-3
Paso 4: Creación de un montaje JDBC.	24-4
Paso 5: Modificación del montaje JDBC para ejecutar scripts SQL	24-5
Paso 6: Creación de un test utilizando montajes para tests	24-6
Paso 7: Implementación del test	24-7
Paso 8: Adición de una biblioteca necesaria	24-8
Paso 9: Ejecución del test	24-8

Apéndice A

Las herramientas de línea de comandos

25-1

Utilización de macros de la línea de comandos	25-2
Definición de la vía de acceso a clases para herramientas de línea de comandos	25-3
Opción -classpath	25-3
Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos.	25-4
Unix: variable de entorno CLASSPATH	25-4
Windows: variable de entorno CLASSPATH	25-5
Interfaz de la línea de comandos de JBuilder	25-6
Acceso a una lista de opciones	25-6
Sintaxis	25-7
Opciones.	25-7
Make de Borland para Java (bmj)	25-9
Sintaxis	25-9
Descripción	25-9
Opciones.	25-10
Opciones de compilación cruzada.	25-14
Especificadores para las clases raíz	25-14
Opciones de MV	25-15
El compilador de Borland para Java (bcj).	25-16
Sintaxis	25-16
Descripción	25-16
Opciones.	25-17
Opciones de compilación cruzada.	25-20
Opciones de MV	25-20

Índice

I-1

Tablas

1.1	Convenciones tipográficas y de símbolos	1-5	8.14	Iconos en la vista Puntos de observación de datos	8-21
1.2	Convenciones de plataformas:	1-6	8.15	Menú contextual con punto de observación seleccionado en la vista Puntos de observación de datos	8-22
4.1	Colores de las listas de bibliotecas:	4-5	8.16	Menú contextual sin selección en la vista Puntos de observación de datos	8-24
5.1	Compiladores disponibles	5-9	8.17	Iconos en la vista Clases cargadas y datos estáticos	8-24
6.1	Términos del sistema de generación	6-2	8.18	Menú contextual con selección de la vista Clases cargadas y datos estáticos	8-25
6.2	Etapas del sistema de generación independiente	6-3	8.19	Menú contextual sin selección de la vista Clases cargadas y datos estáticos	8-27
6.3	Etapas del sistema de generación que establecen dependencias	6-4	8.20	Iconos en la vista Monitores de sincronización	8-27
6.4	Iconos del filtrado de paquetes	6-41	8.21	Menú contextual de la vista Monitores de sincronización	8-28
7.1	Resumen de las condiciones que determinan el uso de las configuraciones de ejecución a cargo de JBuilder	7-9	8.22	Menú contextual con visualizador personalizado seleccionado	8-28
8.1	Comandos de menú para iniciar el depurador	8-6	8.23	Menú contextual sin selección en la vista Puntos de observación de datos	8-28
8.2	Vistas del depurador	8-10	8.24	Botones de la barra de herramientas	8-29
8.3	Iconos de la vista Consola	8-12	8.25	Métodos abreviados para el depurador	8-30
8.4	Menú contextual de la vista Consola	8-12	8.26	Características del depurador para ver el estado del programa	8-62
8.5	Iconos en la vista Clases con inspección desactivada	8-13	8.27	Menú contextual del elemento de la matriz	8-66
8.6	Menú contextual con clase o paquete seleccionado en la vista Clases con inspección desactivada	8-13	8.28	Tipos de variables de puntos de observación en ámbito	8-72
8.7	Menú contextual sin selección en la vista Clases con inspección desactivada	8-13	11.1	Términos de Java y UML	11-2
8.8	Iconos de la vista Puntos de interrupción por datos y código	8-14	11.2	Definiciones de los diagramas UML	11-9
8.9	Menú contextual con punto de interrupción seleccionado en la vista Puntos de interrupción de datos y código	8-15	11.3	Iconos de accesibilidad	11-12
8.10	Menú contextual sin selección en la vista Puntos de interrupción de datos y código	8-15	13.1	Detalles de Buscar referencias	13-4
8.11	Iconos en la vista Hilos, pilas de llamadas y datos	8-17	13.2	Detalles de Buscar referencias	13-5
8.12	Menú contextual con selección en la vista Hilos, pilas de llamadas y datos	8-18	13.3	Detalles de perfeccionamiento	13-9
8.13	Menú contextual sin selección en la vista Hilos, pilas de llamadas y datos	8-20	15.1	Etiquetas Javadoc	15-5
			15.2	Opciones que no se configuran en el asistente de Javadoc	15-25
			21.1	Fichas del cuadro de diálogo para definir las configuraciones de depuración y ejecución servidor y cliente	21-3

21.2	Argumentos de línea de comandos para RMI y el depurador	21-11
21.1	Mensajes de error de RMI cliente/servidor	21-14

Figuras

6.1	Ficha Recursos del cuadro de diálogo	11.8	Panel de estructura de los
	Propiedades de proyecto		diagramas UML
8.1	Interfaz del depurador	11-19	
8.2	Barra de herramientas del depurador .	13.1	Presentación de referencias locales .
8.3	Ventana ExpressionInsight	13-4	
8.4	Ventana Ayuda inmediata	13.2	Presentación de las referencias a
8.5	El punto de ejecución		clases
8.6	Panel dividido de la vista Hilos,	13-5	
	pilas de llamadas y datos	13.3	Presentación de las referencias a
8.7	Vista Monitores de sincronización. .		métodos.
8.8	Ejemplo de archivo fuente stub	13-6	
8.9	Cuadro de diálogo Detenido en	13.4	Presentación de referencias a
	clase con inspección desactivada. . .		campos y variables locales
8.10	Vista Puntos de interrupción de datos	13.5	La opción Vista previa
	y código	13-8	
8.11	Acciones de punto de interrupción . .	13.6	La pestaña Perfeccionamiento
8.12	Mensaje de la barra de estado de		antes de la acción.
	punto de interrupción	13-8	
8.13	Puntos de interrupción condicionales .	13.7	La pestaña Perfeccionamiento
8.14	Vista Clases cargadas y datos		después de la acción
	estáticos	13-10	
8.15	Vista Hilos, pilas de llamada y datos .	13.8	El archivo fuente y la pestaña
8.16	Vista Puntos de observación de		Perfeccionamiento después de la
	datos		acción
8.17	Evaluación de expresiones en el	13-10	
	cuadro de diálogo Evaluar/Modificar .	15.1	Ventana JavadocInsight
8.18	Evaluación de métodos en el cuadro	15-11	
	de diálogo Evaluar/Modificar	15.2	Carpeta Por hacer del panel de
8.19	Nodo Depuración del cuadro de		estructura
	diálogo Modificar configuración de	15.3	Conflictos Javadoc en el panel de
	ejecución		estructura
11.1	Diagrama de paquetes	15-17	
11.2	Diagrama combinado de clases.	15.4	Paso 1 de selección de doclet
11.3	Diagrama de clases con las	15-18	
	propiedades por separado	15.5	Paso 2 de selección de opciones
11.4	Diagrama de clases sin las		de generación y del proyecto
	propiedades por separado	15-20	
11.5	Iconos de visibilidad de JBuilder . .	15.6	Paso 3 de selección de paquetes y
11.6	Visualizador UML		nivel de visibilidad.
11.7	Presentación de las clases internas.	15-22	
		15.7	Paso 4 de especificación de
			opciones de línea de comandos
			para el doclet
		15.8	Nodo de documentación en el
			panel de proyecto.
		15.9	Nodo de documentación ampliados .
		15-30	
		15.10	Salida de archivos de índices
			desde el Doclet estándar
		15-31	
		15.11	Salidas de archivos de índices
			desde el Doclet JDK 1.1
		15-31	
		15.12	Información Javadoc "sobre la
			marcha"
		15-32	

Tutoriales

Creación de un proyecto con un archivo de generación Ant.	18-1	Visualización de código con el visualizador UML	22-1
La vista Histórico	19-1	Creación y ejecución de tests y de conjuntos de tests	23-1
Compilación, ejecución y depuración	20-1	Utilización de montajes para tests	24-1
Depuración remota	21-1		

Capítulo 1

Introducción

Creación de aplicaciones con JBuilder explica cómo utilizar el IDE de JBuilder para gestionar proyectos y para compilar, ejecutar y depurar programas en Java. También explica la forma de utilizar BeansExpress para crear JavaBeans y describe técnicas avanzadas como la distribución de aplicaciones y la internacionalización para diferentes países, la presentación del código, el perfeccionamiento (refactoring) y la comprobación de módulos.

Este manual se ha organizado según el ciclo de vida normal de las aplicaciones, como se muestra a continuación:



La creación de aplicaciones se desarrolla a lo largo de seis etapas distintas:

- 1 Definición de lo que hará la aplicación.
- 2 Diseño de la aplicación y la interfaz de usuario.
- 3 Desarrollo de la aplicación.
- 4 Comprobación de la aplicación.

- 5 Distribución de la aplicación.
- 6 Gestión de todo el proceso mediante el trabajo en equipo y el control de versiones.

Al principio de cada capítulo de *Creación de aplicaciones con JBuilder* se resalta la sección adecuada de la imagen ALM de arriba, mostrando a qué fase del ciclo de vida de la aplicación corresponde el capítulo.

Para ver la definición de términos sobre Java desconocidos, consulte los "Glosarios en línea" en *Introducción a Java*.

Tema

Creación de aplicaciones con JBuilder incluye estos capítulos:

- [Capítulo 2, "Creación y gestión del proyecto"](#)
Explica cómo trabajar con los proyectos de JBuilder y cómo establecer las propiedades de los proyectos.
- [Capítulo 3, "Los grupos de proyectos"](#)
Describe cómo colocar proyectos relacionados en grupos de proyectos y cómo utilizarlos.
- [Capítulo 4, "Gestión de las vías de acceso"](#)
Se trata de un capítulo que acompaña al [Capítulo 2, "Creación y gestión del proyecto"](#), y en el que se describe cómo se utilizan las vías de acceso en JBuilder. Describe cómo se trabaja con bibliotecas y paquetes.
- [Capítulo 5, "Compilación de programas en Java"](#)
Explica cómo compilar el proyecto y cómo definir las opciones del compilador. También explica cómo compilar desde la línea de comandos.
- [Capítulo 6, "Generación de programas en Java"](#)
Explica el proceso de generación de JBuilder y discute las diferencias entre Ejecutar Make y Generar de nuevo. Describe cómo generar archivos Ant externos y grupos de proyectos, así como importar proyectos Ant y exportar proyectos JBuilder a Ant. Entre otras funciones de generación adicionales, se incluyen la generación de archivos SQLJ, la creación de tareas externas de generación, la configuración del menú Proyecto para proyectos y grupos de proyectos, la recopilación automática de paquetes fuente, filtros de paquetes y copia de recursos.
- [Capítulo 7, "Ejecución de programas en JBuilder"](#)
Explica como utilizar el IDE de JBuilder para ejecutar aplicaciones y applets. También explica cómo gestionar las configuraciones de ejecución y utilizar los parámetros y comandos de JBuilder en la línea de comandos.
- [Capítulo 8, "Depuración de programas en Java"](#)
Explica cómo utilizar el Depurador integrado de JBuilder con el fin de localizar y solucionar los errores del programa. Describe la totalidad del

proceso de depuración de applets y aplicaciones, explica los tipos de errores que pueden producirse y la forma de examinar los valores de las variables de los programas para descubrir los errores.

- [Capítulo 9, “Depuración remota”](#)

Describe la forma de depurar un programa que se ejecuta en un ordenador remoto.

- [Capítulo 10, “Creación de JavaBeans con BeansExpress”](#)

Describe la forma de crear JavaBeans y convertir las clases en JavaBean.

- [Capítulo 11, “Presentación de código con UML”](#)

Describe la forma de utilizar las funciones de presentación de código de JBuilder para examinar el código, desplazarse por él e interpretar su significado.

- [Capítulo 12, “Comparación de archivos y versiones”](#)

Aquí se describen las fichas y las funciones disponibles en la ficha Histórico del panel de contenido. Se enumeran las funciones de las distintas versiones de JBuilder y se explica la forma de trabajar con sistemas de control de versiones y sin ellos.

- [Capítulo 13, “Perfeccionamiento de código”](#)

Explica cómo utilizar las funciones de perfeccionamiento de JBuilder.

- [Capítulo 14, “Test de módulos”](#)

Describe las funciones de comprobación de módulos de que dispone JBuilder.

- [Capítulo 15, “Creación de Javadoc a partir de archivos fuente”](#)

Describe la forma de utilizar las funciones relacionadas con Javadoc de JBuilder con el fin de generar archivos de salida HTML a partir de comentarios en código fuente.

- [Capítulo 16, “Distribución de programas en Java”](#)

Ofrece una descripción general de todo lo relacionado con la distribución, explica cómo se utiliza el Creador de recopilatorios para distribuir programas Java y cómo se utiliza el Creador de ejecutables nativos para crear ejecutables nativos para programas Java distribuidos.

- [Capítulo 17, “Internacionalización de programas con JBuilder”](#)

Explica la forma de internacionalizar aplicaciones y applets Java con JBuilder.

Tutoriales:

Estos tutoriales ofrecen la oportunidad de practicar con las funciones de desarrollo de aplicaciones de JBuilder.

Disponible en todas las ediciones de JBuilder

- [Capítulo 18, “Tutorial: Creación de un proyecto con un archivo de generación Ant”](#) - Se utiliza un archivo Ant para crear un proyecto.
- [Capítulo 19, “Tutorial: La vista Histórico”](#) - Gestión de versiones de archivos mediante las fichas Histórico.

Disponible en JBuilder Foundation

- “Tutorial de compilación, ejecución y depuración”: encuentre y arregle los errores de sintaxis, de compilación, y de lógica. (Esta versión ha sido diseñada específicamente para JBuilder Foundation y está disponible en la ayuda en línea.)

Disponibles en las versiones Developer y Enterprise de JBuilder

- [Capítulo 20, “Tutorial: Compilación, ejecución y depuración”](#) - Encuentre y arregle los errores de sintaxis, de compilación, y de lógica. (Esta versión ha sido diseñada específicamente para JBuilder Developer y Enterprise.)
- [Capítulo 21, “Tutorial: Depuración remota”](#) - Se utilizan las funciones de depuración remota para vincularse con un programa que ya se está ejecutando en un ordenador remoto y depurarlo por medio de la inspección interprocesal.

Disponible en JBuilder Enterprise

- [Capítulo 22, “Tutorial: Visualización de código con el visualizador UML”](#) - Se utilizan las funciones de UML de JBuilder para presentar, analizar y resolver los problemas del código.
- [Capítulo 23, “Tutorial: Creación y ejecución de tests y de conjuntos de tests”](#) - Uso de la comprobación de módulos de JBuilder en la creación y ejecución de comprobaciones de módulos con JUnit.
- [Capítulo 24, “Tutorial: Utilización de montajes para tests”](#) - Creación de un montaje para JDBC y para comparaciones y utilización en un test.

Apéndices

Los siguientes apéndices pertenecen a *Creación de aplicaciones con JBuilder*:

- [Apéndice A, “Las herramientas de línea de comandos”](#)

Explica la forma de utilizar los compiladores de línea de comandos de JBuilder, los argumentos de línea de comandos y las herramientas de JDK. También trata de la configuración de la vía de acceso a clases.

Convenciones de la documentación

En la documentación de Borland para JBuilder, el texto con significado especial se identifica mediante la tipografía y los símbolos descritos en la siguiente tabla.

Tabla 1.1 Convenciones tipográficas y de símbolos

Tipografía	Significado
Negrita	La negrita se utiliza para las herramientas java, bmj (Borland Make for Java - Make de Borland para Java), bcj (Borland Compiler for Java - Compilador de Borland para Java) y opciones del compilador. Por ejemplo: javac , bmj , -classpath .
Cursiva	Las palabras en cursiva indican los términos nuevos que se definen y los títulos de libros; ocasionalmente se usan para indicar énfasis.
Teclas	Las teclas, como "Pulse <i>Esc</i> para salir de un menú".
Letra monoespaciada	<div>El tipo monoespaciado representa lo siguiente:</div> <ul style="list-style-type: none">■ texto tal y como aparece en la pantalla■ texto que el usuario debe introducir, como en "Escriba <i>Hola a todos</i> en el campo Título del Asistente para aplicaciones"■ nombres de archivos■ nombres de vías de acceso■ nombres de directorios y carpetas■ comandos, como <code>SET PATH</code>■ código Java■ tipos de datos de Java, como <code>boolean</code>, <code>int</code> y <code>long</code>■ identificadores de Java, como nombres de variables, clases, nombres de paquetes, interfaces, componentes, propiedades, métodos y sucesos■ nombres de argumentos■ nombres de campos■ palabras clave de Java, como <code>void</code> y <code>static</code>
[]	Los corchetes, en las listas de texto o sintaxis, encierran elementos optativos. En estos casos no se deben escribir los corchetes.

Tabla 1.1 Convenciones tipográficas y de símbolos (continuación)

Tipografía	Significado
< >	<p>Los corchetes angulares se utilizan para indicar las variables en vías de acceso a directorios, opciones de comandos y ejemplos de código.</p> <p>Por ejemplo, <nombredearchivo> se puede utilizar para indicar que es necesario especificar un nombre de archivo (incluida su extensión); <nombredeusuario> indica que se debe escribir un nombre de usuario.</p> <p>Cuando reemplace las variables en vías de acceso a directorios, opciones de comandos y ejemplos de código, sustituya la variable entera, incluidos los corchetes angulares (< >). Por ejemplo, sustituya <nombredearchivo> por el nombre de un archivo, como por ejemplo empleado.jds, y quite los corchetes angulares.</p> <p>Nota: los archivos HTML, XML, JSP y de otros formatos basados en etiquetas utilizan también corchetes angulares para delimitar elementos del documento, como por ejemplo: o <ejb-jar>. La siguiente convención describe la forma de especificar cadenas de variables dentro de los ejemplos de código en cuya sintaxis se utilizan corchetes angulares como delimitadores.</p>
<i>Cursiva, serif</i>	<p>Este formato de texto se utiliza para indicar cadenas de variables dentro de los ejemplos de código que ya utilizan corchetes angulares como delimitadores. Por ejemplo, <url="jdbc:borland:jbuilder\samples\guestbook.jds"></p>
...	<p>En los ejemplos de código, los puntos suspensivos (...) indican código que se ha omitido para ahorrar espacio y mejorar la claridad. Si están en un botón, los puntos suspensivos indican que este conduce a un cuadro de diálogo de selección.</p>

JBuilder se puede utilizar con diversas plataformas. En la tabla siguiente se proporciona una descripción de las convenciones de plataformas utilizadas en la documentación.

Tabla 1.2 Convenciones de plataformas:

Item	Significado
Vías de acceso	<p>En las vías de acceso de la documentación se utiliza la barra normal (/).</p> <p>En la plataforma Windows se utiliza la barra invertida (\).</p>

Tabla 1.2 Convenciones de plataformas:

Item	Significado
Directorio inicial	<p>La ubicación del directorio inicial varía según la plataforma y se indica con la variable <home>.</p> <ul style="list-style-type: none"> ■ En UNIX y Linux, el directorio inicial puede variar. Por ejemplo, puede ser /user/<nombre de usuario> o /home <nombre de usuario> ■ En Windows NT, el directorio inicial es C:\Winnt\Profiles\<nombre de usuario> ■ En Windows 2000 y XP, el directorio inicial es C:\Documents and Settings\<nombre de usuario>
Imágenes de pantalla	Las imágenes o capturas de pantalla utilizan el aspecto Metal en diversas plataformas.

Asistencia y recursos para desarrolladores

Borland proporciona una serie de opciones de asistencia y recursos de información para ayudar a los desarrolladores a obtener el máximo rendimiento de los productos Borland. Entre estas opciones se incluye una gama de programas de asistencia técnica de Borland, así como servicios gratuitos en Internet, los cuales permiten consultar una amplia base de información y ponerse en contacto con otros usuarios de productos Borland.

El servicio de asistencia técnica de Borland

Borland ofrece varios programas de asistencia para clientes actuales y potenciales. Se puede elegir entre varios tipos de asistencia, que van desde la ayuda en la instalación de los productos Borland hasta el asesoramiento de expertos y la asistencia pormenorizada.

Si desea más información sobre el servicio al desarrollador de borland.com, visite nuestra página web, en <http://www.borland.com/devsupport> o póngase en contacto con nuestro departamento de ventas.

Cuando se ponga en contacto con el servicio técnico tenga a mano la información completa sobre el entorno, la versión del producto utilizada y una descripción detallada del problema.

Si necesita más información sobre las herramientas o la documentación de otros proveedores, póngase en contacto con ellos.

Recursos en línea

También puede obtener información de los siguientes recursos en línea:

World Wide Web	http://www.borland.com/ http://www.borland.com/techpubs/jbuilder/
Newsletters	Para suscribirse a las newsletters, rellene el formulario en línea que aparece en: http://www.borland.com/products/newsletters/index.html

World Wide Web

Visite periódicamente www.borland.com/jbuilder. El equipo de desarrollo de productos Java publica en esta página documentación técnica, análisis de competitividad, respuestas a preguntas frecuentes, aplicaciones de ejemplo, software actualizado e información sobre productos nuevos y antiguos.

En particular, pueden resultar interesantes las siguientes direcciones:

- <http://www.borland.com/jbuilder/> (actualizaciones de software y otros archivos)
- <http://www.borland.com/techpubs/jbuilder/> (actualizaciones de documentación y otros archivos)
- <http://community.borland.com/> (contiene nuestra revista de noticias para desarrolladores en formato web)

Grupos de noticias de Borland

Puede registrar JBuilder y participar en los grupos de debate sobre JBuilder, estructurados en hilos. Los grupos de noticias de Borland ofrecen un medio para que la comunidad internacional de clientes de Borland pueda intercambiar sugerencias y técnicas sobre los productos de Borland, así como las herramientas y tecnologías relacionadas.

Puede encontrar grupos de noticias, moderados por los usuarios, sobre JBuilder y otros productos de Borland, en <http://www.borland.com/newsgroups>.

Usenet, grupos de noticias

En Usenet existen los siguientes grupos dedicados a Java y temas relacionados:

- [news:comp.lang.java.advocacy](#)
- [news:comp.lang.java.announce](#)
- [news:comp.lang.java.beans](#)
- [news:comp.lang.java.databases](#)

- `news:comp.lang.java.gui`
- `news:comp.lang.java.help`
- `news:comp.lang.java.machine`
- `news:comp.lang.java.programmer`
- `news:comp.lang.java.security`
- `news:comp.lang.java.softwaretools`

Nota Se trata de grupos moderados por usuarios; no son páginas oficiales de Borland.

Información sobre errores

Si cree que ha encontrado un error en el software, por favor informe a Borland en alguno de los siguientes sitios:

- La página Support Programs en <http://www.borland.com/devsupport/america/>. Pulse el enlace "Reporting Defects" para llegar al formulario Entry.
- Quality Central en <http://qc.borland.com>. Siga las instrucciones de la sección "Bugs Reports" de la página Quality Central.
- Comando de menú Quality Central del menú Herramientas de JBuilder (Herramientas\Quality Central). Siga las instrucciones para crear una cuenta de usuario de QC e informar del fallo. Si desea más información, consulte la documentación de Borland Quality Central.

Cuando informe sobre un fallo, incluya todos los pasos necesarios para llegar a él, así como toda la información posible sobre la configuración, el entorno y las aplicaciones que se estaban utilizando junto con JBuilder. Intente explicar con la mayor claridad posible las diferencias entre el comportamiento esperado y el obtenido.

Si desea enviar felicitaciones, sugerencias o quejas al equipo de documentación de JBuilder, envíe un mensaje a jpgpubs@borland.com. Envíe únicamente comentarios sobre la documentación. Tenga en cuenta que los asuntos relacionados con el servicio técnico se deben enviar al departamento de asistencia técnica para programadores.

JBuilder es una herramienta creada por desarrolladores y para desarrolladores. Valoramos intensamente sus aportaciones.



Creación y gestión del proyecto

JBUILDER realiza todas las tareas dentro del contexto de un *proyecto*. En este manual, el término "proyecto" incluye todos los archivos que constituyen un trabajo definido por un usuario, la estructura de directorios en la que residen esos archivos y las vías de acceso, opciones y recursos necesarios.

El proyecto es una herramienta de organización, no un lugar de almacenamiento. Esto significa que los archivos de un proyecto pueden estar en cualquier carpeta. La reestructuración de un árbol de proyecto no afecta al árbol de directorios. Así, se ofrece un control independiente de los proyectos y de la estructura de directorios.

Cada proyecto se administra mediante un archivo de proyecto. El nombre del archivo del proyecto es el nombre del proyecto con la extensión `jpx`. El archivo de proyecto contiene una lista de los archivos del proyecto y mantiene las propiedades del proyecto; estas incluyen una plantilla de proyecto, las vías de acceso por defecto, las bibliotecas de clase y las configuraciones de conexión. JBUILDER utiliza esta información cuando carga, guarda, genera y ejecuta proyectos. Los archivos de proyecto se modifican cuando se utiliza el entorno de desarrollo de JBUILDER para añadir o eliminar archivos, o para definir o cambiar las propiedades del proyecto. El archivo de proyecto se ve como un nodo en el panel de proyecto. A continuación se enumeran todos los paquetes y archivos del proyecto.

Nota Si la recopilación automática de fuentes está activada, en el panel de proyecto aparecen también los nodos de los paquetes de código fuente. Estos muestran los archivos y paquetes que aparecen en la vía de acceso a fuentes del proyecto. Consulte ["Recopilación automática de paquetes fuente" en la página 6-38](#).

A pesar de que pueden incluir archivos de cualquier tipo en los proyectos de JBuilder, hay ciertos tipos que JBuilder reconoce automáticamente y para los que dispone de visualizadores. Para añadir archivos de tipos binarios, personalice la gestión de los tipos de archivos mediante la asociación de otros tipos de archivo a los que ya reconoce JBuilder y vea los iconos asociados con tipos de archivos seleccionando Herramientas|Preferencias y eligiendo Navegador|Tipos de archivos para mostrar la ficha Tipos de archivos.

La primera vez que se inicia JBuilder se solicita que se configuren las asociaciones de archivos. A continuación, se solicita que se asocien a JBuilder los archivos `.class` y `.java`, así como los archivos de proyecto y de grupo de proyectos. De este modo, JBuilder se convierte en el programa por defecto para abrir y ver estos archivos. Para modificar esta configuración, seleccione Herramientas|Configurar asociaciones de archivos. Se abre el cuadro de diálogo del mismo nombre.

Creación de proyectos

Para crear un proyecto, utilice el Asistente para proyectos de JBuilder para generar automáticamente la estructura básica de archivos, directorios, vías de acceso y preferencias. El Asistente para proyectos crea automáticamente el archivo de notas de proyecto, que recoge las notas y comentarios. Cuando se utilizan los asistentes de JBuilder para crear archivos Java, los campos de la clase Javadoc que se rellenan en el Asistente para proyectos se utilizan en el archivo de notas del proyecto en forma de comentarios de encabezado Javadoc y, por tanto, se incluyen en la documentación generada por Javadoc. Estos comentarios se pueden modificar en la ficha General de Propiedades de proyecto.

En muchos casos, cuando se abre un asistente de JBuilder sin que haya un proyecto abierto, aparece el Asistente para proyectos, que ofrece la posibilidad de crear un proyecto.

Creación de proyectos con ayuda del Asistente para proyectos

Para crear un proyecto con el Asistente para proyectos, seleccione Archivo|Nuevo proyecto. También se puede seleccionar Archivo|Nuevo (o pulsar el botón Nuevo de la barra de herramientas de JBuilder), seleccionar Proyecto en el árbol y hacer doble clic en el icono Proyecto.

Sugerencia Otro modo de iniciar el Asistente para proyectos es hacer clic en la flecha abajo que se encuentra junto al botón Nuevo de la barra de herramientas, que abre un menú desplegable, y seleccionar ahí Proyecto|Proyecto. También se puede hacer clic con el botón derecho en un directorio en el panel del proyecto y seleccionar Nuevo|Proyecto.

Se abre el asistente para proyectos.

Selección del nombre del proyecto y la plantilla

Utilice el Paso 1 para definir el nombre, el tipo, el directorio raíz y la plantilla del proyecto.

1 Escriba un nombre para el nuevo proyecto.

JBuilder utiliza el nombre del proyecto como nombre de paquete por defecto.

Cualquier nombre de archivo válido para el sistema de archivos se puede utilizar como nombre del proyecto. Sin embargo, hay otros nombres que se derivan de este nombre de archivo y tienen ciertas restricciones que hay que cumplir:

- a El nombre del directorio del proyecto puede aparecer en una vía de acceso a clases Java. Dado que los espacios incrustados pueden ocasionar problemas, si hay espacios se sustituyen por guiones bajos.
- b El asistente utiliza el nombre del proyecto como nombre de paquete por defecto. Por tanto, debe ser un nombre de paquete Java válido. Esto significa que los números se eliminan del nombre del archivo, los espacios se sustituyen por guiones bajos, las mayúsculas se cambian por minúsculas y el nombre puede cortarse si es demasiado largo.

2 Seleccione el directorio del proyecto. El directorio del proyecto es el que contiene el archivo del proyecto. Casi todas las demás vías de acceso del proyecto, como las vías de acceso a fuentes y a copias de seguridad, se derivan de este directorio.

- Pulse la tecla de flecha abajo para seleccionar un directorio superior utilizado previamente o para elegir uno del mismo árbol y modificarlo.
- Puede escribir en el campo directamente o pulsar el botón de puntos suspensivos para buscar un directorio ya creado.

Nota

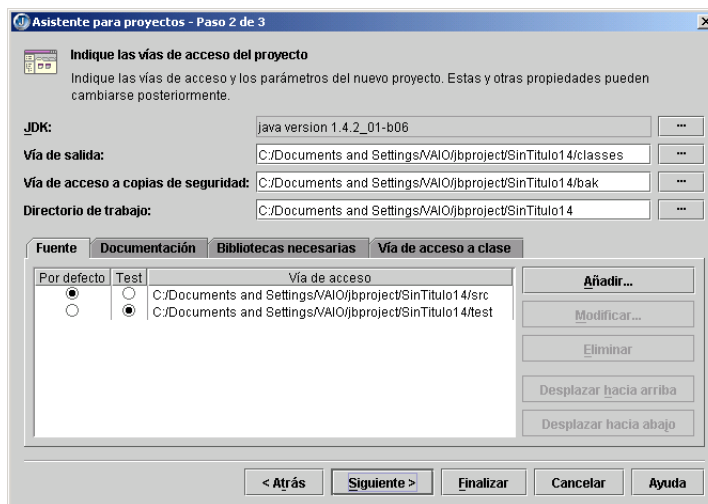
Si la vía de acceso tiene errores de sintaxis, no podrá continuar.

- 3 Acepte Proyecto por defecto como el valor del campo Plantilla. (Puede pulsar el botón de ayuda si desea leer más acerca de las plantillas de proyectos.)
- 4 Para añadir el proyecto que está creando a un grupo de proyectos ya creado (el grupo de proyectos debe estar activo o esta opción es desactivada), marque la casilla de selección Añadir proyecto al grupo activo. Solo se puede activar esta casilla si tiene abierto y activo un grupo de proyectos. Los grupos de proyectos solamente están disponibles en JBuilder Developer y Enterprise. Para obtener más información sobre grupos de proyectos, consulte el [Capítulo 3, “Los grupos de proyectos”](#).
- 5 Para generar un archivo de notas de un proyecto HTML, marque la casilla de selección Generar archivo de notas del proyecto. Este archivo es opcional.
- 6 Pulse Siguiente para pasar al Paso 2.

Si está habilitado el botón Finalizar, cuando se pulsa se aceptan las opciones por defecto de JBuilder para el resto del asistente y el proyecto se crea inmediatamente.

Configuración de las vías de acceso del proyecto

En el Paso 2 se establecen todas las vías de acceso del proyecto, incluida la versión del JDK que se utiliza para la compilación y ejecución. Si lo cree necesario, puede cambiar esta configuración más adelante mediante la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto (Proyecto/Propiedades de proyecto)



JBuilder sugiere que el directorio del proyecto configurado en el Paso 1 sea el directorio de trabajo. El directorio de trabajo es el directorio inicial que JBuilder proporciona a los programas cuando se inician. Se puede configurar cualquier directorio como directorio de trabajo.

Si desea modificar alguna de las vías de acceso de esta ficha, puede escribir la nueva o desplazarse hasta ella mediante el botón de puntos suspensivos contiguo al campo apropiado.

Nota Si acaba de empezar a utilizar JBuilder, solo tiene que aceptar los valores por defecto de esta ficha. Si se escribe una vía de acceso con errores de sintaxis no se puede continuar. Los usuarios con más experiencia quizás prefieran cambiar estos directorios por los de su elección. Si desea obtener información más completa acerca del uso de esta ficha y sobre los directorios, pulse el botón de ayuda del Asistente para proyectos.

Pulse Siguiente para continuar con el Paso 3 o, bien, pulse Finalizar para crear su proyecto. Los usuarios con más experiencia quizás prefieran continuar hasta el Paso 3.

Configuración de las opciones generales del proyecto

El Paso 3 del Asistente para proyectos incluye las opciones generales del proyecto, como la codificación, la recopilación automática de paquetes fuente, los campos de clases Javadoc y las referencias de las bibliotecas del proyecto.

Esta información se puede modificar más adelante en la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto|Propiedades de proyecto).

Asistente para proyectos - Paso 3 de 3

Especificar configuración general del proyecto

Los siguientes parámetros servirán para definir su nuevo proyecto. Estas y otras propiedades pueden cambiarse después de crear el proyecto.

Codificación Por defecto

Recopilación automática de paquetes fuente

☒ Activar la localización y compilación de paquetes fuente

Niveles jerárquicos mostrados: 3

Archivos Javadoc de clases:

Etiqueta	Texto
Título:	
Descripción:	
Copyright:	Copyright (c) 2003
Empresa:	
@author	
@version	1.0

☐ Incluir referencias de archivos de clase de las bibliotecas del proyecto

☐ Mostrar diagrama con referencias del código

< Atrás Siguiente > Finalizar Cancelar Ayuda

- 1 Seleccione una codificación o acepte la que se ofrece por defecto. La codificación determina cómo gestiona JBuilder los caracteres que no pertenecen al conjunto de caracteres ASCII. La opción por defecto es la página de códigos del sistema operativo.

Consulte

- “Elección de una codificación nativa para el compilador” en la [página 17-13](#)
- “Internationalization Tools: native2ascii” (en inglés) en <http://java.sun.com/products/jdk/1,4/docs/tooldocs/tools.html#intl>

2 Seleccione Opciones de recopilación automática de paquetes fuente.

- a** La opción Activar la localización y compilación de paquetes fuente está activada por defecto. Si está activada tienen lugar varias acciones:

- Todos los paquetes de la vía de acceso a fuentes del proyecto aparecen en el panel de proyecto (panel superior izquierdo) del IDE.
- Los paquetes que contienen archivos Java se compilan de forma automática en tiempo de compilación.

No se muestran todos los paquetes como descendientes del proyecto, solo un subconjunto lógico determinado por el nivel de profundidad con el que JBuilder expone los paquetes. Se debe ampliar el árbol para ver los subpaquetes restantes.

- b** Indique hasta qué nivel desea que se expongan los paquetes.

JBuilder expone los paquetes hasta el nivel que se defina aquí, excepto si la adición de niveles a un paquete no modifica la longitud de la lista de paquetes. Por ejemplo, si cuenta con estos tres paquetes:

```
uno.dos.tres.cuatro
uno.dos.tres.cinco
uno.dos.cuatro.seis
```

y define el nivel de paquetes en tres, esto es lo que aparece en el panel de proyecto:

```
uno.dos.tres
uno.dos.cuatro.seis
```

Los dos paquetes `uno.dos.tres.cuatro` y `uno.dos.tres.cinco` están en `uno.dos.tres`, por lo que JBuilder sólo representa el paquete superior y permite ampliar el nodo para tener acceso a los paquetes y archivos que contiene.

El paquete `uno.dos.cuatro.seis` se expone en el cuarto nivel, ya que si se acorta la representación no se acorta la lista de paquetes, por lo que se presenta de todas formas.

Consulte

- “Paquetes” en la [página 4-6](#)
- 3** Especifique los campos de la clase Javadoc. Se pueden utilizar en el archivo de notas del proyecto e insertar como comentarios de cabecera Javadoc en los archivos generados por asistentes creados para el proyecto.

Seleccione el campo que desea modificar y escriba el texto deseado en la columna Texto.

- 4 Active la opción Incluir referencias de archivos de clase de las bibliotecas del proyecto si desea que sea posible buscar referencias a todas las bibliotecas del proyecto. El comando Buscar referencias del menú contextual del editor permite detectar todos los archivos fuente en los que se utiliza un símbolo determinado. Actívela también si desea que los diagramas UML del proyecto muestren las referencias de las bibliotecas del proyecto. Buscar referencias es una característica de JBuilder Developer; UML es una característica de JBuilder Enterprise.

Consulte

- [“Búsqueda de referencias” en la página 13-4](#)
 - [Capítulo 11, “Presentación de código con UML”](#)
- 5 Si tiene JBuilder Enterprise y desea incluir referencias tales como archivos IOP o stubs EJB en los diagramas UMK del proyecto, seleccione la opción Mostrar diagrama con referencias del código.

Consulte

- [Capítulo 11, “Presentación de código con UML”](#)
- 6 Pulse el botón Finalizar.

Creación de proyectos a partir de archivos anteriores

Es una función de JBuilder Developer y Enterprise

El asistente Proyecto para código existente permite crear un proyecto en JBuilder a partir del trabajo ya realizado. Cuando se utiliza este asistente, JBuilder examina el directorio actual y crea vías de acceso para compilar, buscar, depurar y realizar otros procesos. Todos los archivos JAR y ZIP que no se encuentran en bibliotecas se colocan en una nueva biblioteca y se añaden al proyecto. Las bibliotecas del proyecto se recogen en la pestaña Bibliotecas necesarias de la ficha Vías de acceso de Propiedades de proyecto (Proyecto|Propiedades de proyecto).

Para abrir el asistente Proyecto para código existente:

- 1 Seleccione Archivo|Nuevo (o pulse el botón Nuevo de la barra de herramientas de JBuilder). Se abre la galería de objetos.
- 2 Seleccione Proyecto en el árbol para que se abra la ficha Proyecto.
- 3 Haga doble clic sobre el icono Proyecto para código existente.

Sugerencia

Otro modo de iniciar el Asistente para Proyecto para código existente consiste en hacer clic en la flecha abajo que se encuentra junto al botón Nuevo de la barra de herramientas de JBuilder, que abre un menú, y seleccionar ahí Proyecto|Proyecto para código existente.

Selección del directorio fuente y el nombre del nuevo proyecto de JBuilder

En el Paso 1 se configuran el directorio, el nombre, el tipo y la plantilla del proyecto.

- 1 Seleccione el directorio donde se encuentra el proyecto o el árbol de fuente. Pulse el botón de puntos suspensivos para buscarlo. JBuilder examina el directorio seleccionado en busca de archivos de clase, de fuente, JAR y ZIP, y los coloca en los subdirectorios adecuados de ese directorio.
- 2 Escriba un nombre para el nuevo proyecto.
- 3 Seleccione la plantilla del proyecto si elige:
 - Pulse la tecla de flecha abajo para seleccionar un proyecto que se haya utilizado anteriormente como plantilla.
 - Pulse el botón de puntos suspensivos para utilizar como plantilla un proyecto diferente en el cuadro de diálogo Abrir proyecto.

Las plantillas de proyectos ofrecen valores por defecto para las opciones descritas en el cuadro de diálogo Propiedades de proyecto (Proyecto|Propiedades de proyecto). Si ya cuenta con un proyecto de JBuilder de propiedades parecidas a las que se necesitan en el nuevo, selecciónelo aquí. Así se reducen al mínimo las tareas repetitivas que conlleva la configuración de un nuevo proyecto en un entorno ya establecido.

- 4 Indique si desea crear un archivo HTML de notas del proyecto. La información inicial de este archivo, como el título, el autor y la descripción, se genera desde los campos de la clase Javadoc configurados en el Paso 3 del asistente Proyecto para el código existente. En caso necesario también es posible añadir notas y otra información en este archivo.
- 5 Pulse Siguiente para pasar al Paso 2.

Los pasos 2 y 3 del asistente Proyecto para el código existente son los mismos que en el Asistente para proyectos. Estos pasos también coinciden con la ficha Vías de acceso y la ficha General de Propiedades de proyecto. Consulte [“Creación de proyectos con ayuda del Asistente para proyectos” en la página 2-2](#).

Si el proyecto necesita alguna biblioteca determinada, puede añadirla en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto. Para configurar la clase principal para ejecutar el proyecto, seleccione la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto.

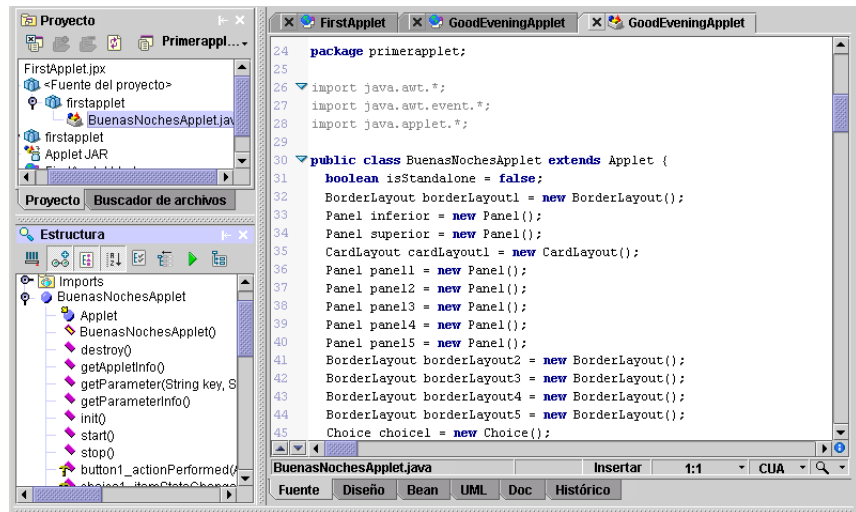
Consulte

- [“Configuración de vías de acceso a bibliotecas necesarias” en la página 2-27](#)
- [“Ejecución de proyectos” en la página 7-3](#)

Presentación de los archivos

JBuilder muestra todos los archivos abiertos de un proyecto en el panel de contenido del Visualizador. Haga doble clic en un archivo para abrirlo en el panel de contenido. Para abrir un archivo también se puede arrastrar del panel del proyecto al panel de contenido. En la parte superior aparece una pestaña con el nombre del archivo.

La siguiente figura muestra en el panel de proyecto un archivo de proyecto, Primerapplet.jpx, con los archivos fuente enumerados debajo. Este proyecto incluye un paquete, un fichero de archivos y tres archivos fuente. En el panel de contenido hay dos archivos abiertos y el seleccionado, BuenasNochesApplet.java, se muestra en el panel de código fuente.



Sugerencia Si lo prefiere, puede tener pestañas de los archivos abiertas verticalmente a la derecha del panel de contenido en lugar de en la parte superior. Seleccione Herramientas|Preferencias|Visualizador y en las opciones de la pestaña Panel de contenido, seleccione Vertical en la lista desplegable Orientación. Las opciones de la ficha Panel de contenido también ofrecen otras opciones para mostrar las pestañas.

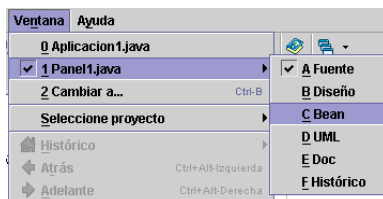
Algunos nodos del panel del proyecto se pueden ampliar mediante el icono de ampliación que tienen a la izquierda. De esta forma, se pueden ver los archivos que contiene el nodo. Por ejemplo, si se amplía un nodo de paquete, los archivos del paquete aparecen en el panel del proyecto.

Haga clic con el botón derecho en el panel del proyecto para mostrar un menú con opciones como Nuevo, Abrir, Añadir archivos/Paquetes/Clases, Eliminar del proyecto, Cerrar proyecto, Ejecutar Make, Generar de nuevo, Formato, Exportar como un servicio web, Exportar como servicio web asíncrono y Propiedades. Las opciones de menú exactas disponibles dependen del nodo seleccionado y de que se seleccionen uno o más nodos. También se puede acceder a muchas de estas opciones de menú desde el menú Proyecto.

Cambio entre archivos

Cuando hay muchos archivos abiertos, no siempre es fácil examinar todas las pestañas de archivos abiertos en busca de la que desea utilizar. Existen dos modos de cambiar rápidamente entre los archivos abiertos:

- Seleccionar Ventana/Cambiar a o pulsar **Ctrl+B** para abrir el cuadro de diálogo, que enumera todos los archivos abiertos del proyecto. Desplácese para seleccionar el archivo que desea. O bien, comience a escribir el nombre del archivo, con lo que se selecciona la primera coincidencia de la lista; siga escribiendo hasta que se seleccione el archivo que desea. Una vez seleccionado, pulse Aceptar.
- Seleccione Ventana para abrir el menú Ventana y seleccione el archivo que desee en la lista de archivos abiertos enumerados en el menú. El archivo activo se identifica mediante una marca. El archivo activo también cuenta con una flecha a la derecha del menú que indica que tiene un submenú disponible. Si se abre el submenú, se ven las opciones para presentar diferentes vistas del archivo:



El menú Ventana también permite cambiar entre *proyectos* abiertos mediante Selección de proyecto.

Visualización de recopilatorios desde el panel de proyecto

Si se amplía un nodo de archivo recopilatorio (JAR o EAR) en el panel del proyecto, se pueden ver los archivos que contiene. Cuando se hace doble clic en un archivo, este se abre en el panel de contenido. JBuilder utiliza el visualizador adecuado según el tipo de archivo. Por ejemplo, si el recopilatorio contiene un archivo HTML y un archivo GIF, la ficha Ver usa el navegador de JBuilder para mostrar el archivo HTML, y el visualizador de imágenes para mostrar la imagen.

Almacenamiento de proyectos

Cuando se trabaja con un proyecto, se puede guardar en la ubicación propuesta o en un directorio diferente. Por defecto, JBuilder guarda los proyectos en el directorio `jbpproject` del directorio inicial, aunque esto depende de cómo esté configurado el sistema. Cada proyecto se guarda en su propio directorio dentro de `jbpproject`. Los directorios de proyecto incluyen un archivo del proyecto, un archivo `.html` optativo para las notas del proyecto, un subdirectorio `classes` para los archivos generados (como los archivos `.class`), un subdirectorio `src` para los archivos de código fuente, un subdirectorio `bak` para los archivos de copia de seguridad y un subdirectorio `doc` para la documentación.

Almacenamiento y cierre de proyectos



Para guardar un proyecto, seleccione **Archivo|Guardar todo** o **Archivo|Guardar proyecto**, o pulse el botón **Guardar todo** de la barra de herramientas principal.



Para cerrar un proyecto, seleccione **Archivo|Cerrar proyectos** o pulse el botón **Cerrar proyecto** de la barra de herramientas del proyecto.

Consulte

- [“Cómo construye JBuilder las vías de acceso” en la página 4-9](#)
- [“Localización de los archivos” en la página 4-12](#)

Apertura de proyectos y archivos

Para abrir por primera vez un proyecto ya creado, utilice **Archivo|Abrir proyecto**. Para abrir un proyecto que ya se ha abierto antes, utilice el comando **Archivo|Abrir proyecto** o el comando **Archivo|Abrir de nuevo**.

Para abrir un proyecto mediante el comando **Archivo|Abrir proyecto**:

- 1 Seleccione **Archivo|Abrir proyecto**. Se muestra el cuadro de diálogo **Abrir archivo**.
- 2 Desplácese hasta el directorio que contenga el archivo de proyecto que desea abrir.
- 3 Seleccione el archivo de proyecto y pulse **Aceptar** o *Intro*. También puede hacer doble clic en el archivo de proyecto para abrirlo.

Abrir archivos

Para abrir un archivo en el panel de contenido, puede seguir uno de estos cuatro métodos:

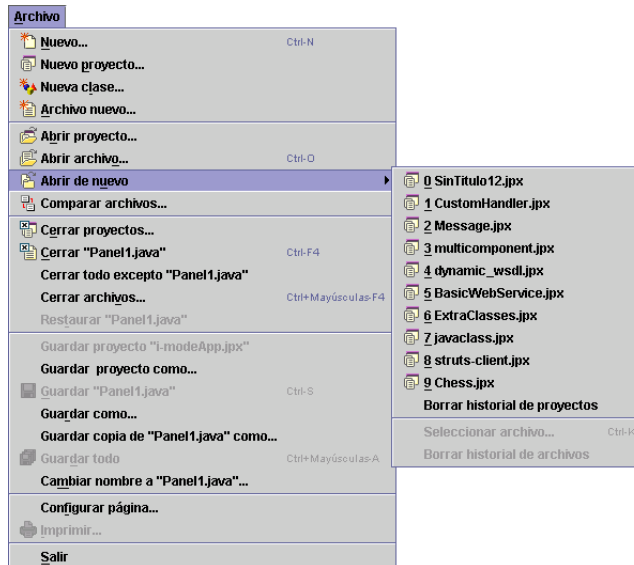
- Hacer doble clic en el archivo, en el panel de proyecto.
- Seleccionar el archivo en el panel de proyecto y pulsar *Intro*.

- Hacer clic con el botón derecho del ratón en el archivo, en el panel de proyecto, y seleccionar Abrir.
- Arrastrar un archivo del panel del proyecto al panel de contenido.

Cómo abrir de nuevo proyectos y archivos

Se pueden abrir de nuevo proyectos y archivos abiertos anteriormente mediante el comando ArchivoAbrir de nuevo:

- 1 Elija ArchivoAbrir de nuevo. Aparece el menú Abrir de nuevo:



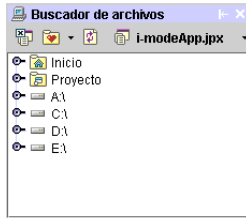
- 2 Seleccione el proyecto que desee abrir en la lista de proyectos abiertos previamente de la parte superior del menú o, bien, seleccione el archivo que desee abrir en la lista de archivos abiertos anteriormente de la parte inferior del menú. También se puede ver una lista completa de archivos abiertos anteriormente entre la que elegir si se selecciona Seleccionar archivo en el menú.

Para eliminar la lista de proyectos del menú Abrir de nuevo, seleccione ArchivoAbrir de nuevoBorrar historial de proyectos. Para eliminar la lista de archivos del menú Abrir de nuevo, seleccione ArchivoAbrir de nuevoBorrar historial de archivos.

El Visualizador de archivos

El panel del proyecto puede contar con un Buscador de archivos que permite buscar en todos los archivos del sistema.

Para abrirlo, haga clic en la pestaña Buscador de archivos. El panel del proyecto tiene una apariencia parecida a la siguiente, según el número de unidades a que tenga acceso el sistema:



El Buscador de archivos proporciona rápido acceso al directorio del proyecto actual, al directorio inicial de JBuilder y a todas las unidades disponibles. Haga doble clic en cualquier archivo que desee abrir. Para ver el contenido de una carpeta o unidad, amplíe el nodo correspondiente en el panel del proyecto.

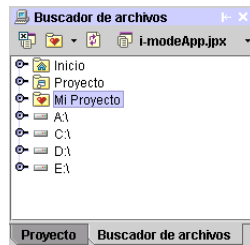
Favoritos

También se puede marcar como favorito un proyecto, un directorio o una unidad, para que aparezca en el Buscador de archivos, lo que permite acceder rápidamente a un proyecto, directorio o unidad desde el Buscador de archivos:

- 1 Utilice el Buscador de archivos para seleccionar el proyecto, directorio o unidad que desea especificar como favorito.
- 2 Haga clic en el icono Favoritos de la barra de herramientas del Buscador de archivos y seleccione Añadir a Favoritos.
- 3 Acepte el nombre por defecto o especifique uno nuevo para el elemento. El nombre que se especifica es el nombre que aparece en el Buscador de archivos con un icono de corazón junto a él, que indica que es el que se ha especificado como favorito.
- 4 Pulse Aceptar.

El directorio, el proyecto o la unidad que se ha marcado como favorito se añade a la lista de favoritos que aparece bajo el directorio inicial y del

proyecto en el Buscador de archivos. Todos los favoritos tienen un icono de corazón junto a ellos.



Para organizar la lista de favoritos, haga clic en el icono Favoritos de la barra de herramientas y seleccione Organizar Favoritos. Aparece un cuadro de diálogo con la lista de favoritos. Utilice el cuadro de diálogo para cambiar los elementos de la lista al orden que desee.

La misma función Favoritos se puede encontrar en los cuadros de diálogo que abren los comandos de menú ArchivosAbrir proyecto y ArchivosAbrir archivo.

Gestión de proyectos

JBUILDER tiene como finalidad asistir a los desarrolladores en la realización de la mayor cantidad posible de tareas de desarrollo. Las herramientas de proyectos de JBUILDER, el completo IDE y las amplias funciones del editor automatizan y simplifican la tarea del desarrollo, y permiten concentrarse en el código.

Cómo añadir elementos a un proyecto

En el Visualizador se pueden añadir al proyecto archivos nuevos y antiguos, paquetes, directorios y carpetas, y con la edición JBUILDER Developer, vistas de directorio. JBUILDER proporciona varias formas de añadir elementos a los proyectos.

- Haga clic con el botón derecho del ratón en un nodo del panel del proyecto, elija Nuevo en el menú contextual y examine las opciones disponibles. Según el nodo elegido, estas son las opciones disponibles:
 - Clase: abre el Asistente para clases.
 - Interfaz: abre el Asistente para interfaces.
 - Paquete: abre el cuadro de diálogo Crear paquete.
 - Archivo: abre el cuadro de diálogo Crear archivo.
 - Directorio: abre el cuadro de diálogo Crear directorio.
 - Carpeta: abre el cuadro de diálogo Crear una carpeta.
 - Vista de directorio: abre el cuadro de diálogo Seleccionar un directorio.



- Pulse el icono Añadir al proyecto de la barra de herramientas del panel del proyecto para mostrar el cuadro de diálogo Añadir al <proyecto>, en el que se pueden añadir al proyecto archivos y paquetes ya creados.
- Elija Archivos\Archivo nuevo para abrir el cuadro de diálogo del mismo nombre, que permite añadir archivos vacíos al proyecto.
- Seleccione Archivos\Nueva clase para abrir el Asistente para clases, que permite añadir una clase al proyecto.

Después de añadir los archivos al proyecto se pueden arrastrar de unos nodos antecesores del proyecto a otros.

En el caso de proyectos más grandes se pueden utilizar carpetas de proyecto para organizar la jerarquía del proyecto.

Nota Las carpetas de proyecto sólo se utilizan para fines organizativos y no se corresponden con los directorios del disco.

Más información sobre la adición de carpetas

Las carpetas de proyecto no afectan al árbol de directorios. Son herramientas de organización que permiten ordenar los elementos de un proyecto de una forma útil sin afectar a la estructura de directorios.

Para añadir una carpeta de proyecto:

- 1 Haga clic con el botón derecho del ratón en el nodo de proyecto del panel de proyecto.
 - 2 Elija Nuevo\Carpeta.
- Sugerencia** Para anidar la nueva carpeta dentro de otra ya creada, seleccione esa carpeta antes de elegir Nuevo\Nueva carpeta.
- 3 Escriba el nombre de la carpeta en el cuadro de diálogo que aparece.
 - 4 Pulse Aceptar o *Intro*.

Para añadir un archivo a una carpeta:



- 1 Haga clic con el botón derecho del ratón en la carpeta y pulse Añadir archivos/paquetes/clases para abrir el cuadro de diálogo de adición, o seleccione la carpeta y pulse el icono Añadir al proyecto de la barra de herramientas del panel de proyecto.
- 2 Desplácese por la ficha Explorador del cuadro de diálogo Añadir archivo al proyecto, para acceder al directorio que contiene el archivo que desea añadir.
- 3 Seleccione el archivo y pulse Abrir.

Importación de código fuente a un proyecto

Se puede importar código fuente o recursos a un proyecto mediante el Asistente para importar código fuente. Los archivos seleccionados para importar se copian en la vía de fuentes del proyecto.

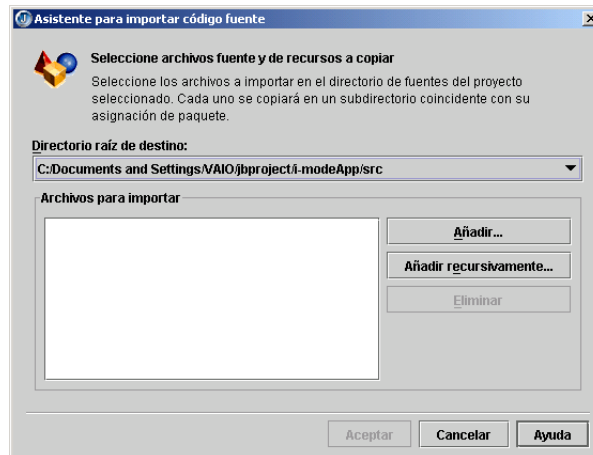
Para iniciar el Asistente para importar código fuente:

- 1 Pulse el botón Nuevo de la barra de herramientas de JBuilder para abrir la galería de objetos.
- 2 Seleccione General en el árbol.
- 3 Haga doble clic en el icono Importar código fuente de la ficha General.

Sugerencia

Otro modo de iniciar el Asistente para importar código fuente consiste en hacer clic en la flecha abajo que se encuentra junto el botón Nuevo de la barra de herramientas, que abre un menú desplegable y, a continuación, seleccionar Importar código fuente. O bien, también se puede seleccionar Asistentes\Importar código fuente.

Aparece el Asistente para importar código fuente:



Se puede elegir especificar todos los archivos que se añaden al proyecto, o añadir todos los archivos en un directorio, incluidos los de todos los subdirectorios. Los tipos de archivos de recursos que se encuentran en un directorio con al menos un archivo Java (o en el directorio superior de archivos de recursos) se supone que están en el mismo paquete que ese archivo Java.

Para especificar todos los archivos que desee añadir, pulse el botón Añadir y desplácese a la ubicación del archivo.

Para añadir todos los archivos de un directorio y de sus subdirectorios, pulse el botón Añadir recursivamente y desplácese hasta el principio de la estructura de directorios que desee añadir.

Mientras se seleccionan archivos y directorios, los nombres de los archivos seleccionados aparecen en la lista Archivos para importar. Siga añadiendo archivos a la lista, mediante el botón Añadir o el botón Añadir recursivamente, hasta que haya seleccionado todos los archivos que desea copiar en el proyecto actual. Cuando haya seleccionado todos los archivos que desea, seleccione Aceptar, y el asistente empieza a copiar los archivos seleccionados en la vía de fuentes del proyecto.

Adición de archivos y paquetes con detección automática de la vía de acceso a archivos fuente

La detección automática de la vía de acceso a archivos fuente añade automáticamente todos los archivos y paquetes de la vía de acceso a archivos fuente del proyecto. Cuando se realiza la generación con esta opción activada, todos los paquetes que contienen archivos generables se generan y se copian automáticamente, con todos los recursos, en la vía de acceso de archivos generados de proyecto. Puede configurar esta opción en la ficha General del cuadro de diálogo Propiedades de proyecto. Está activada por defecto.

Adición de archivos y paquetes sin detección automática de la vía de acceso a archivos fuente

Si no se utiliza la detección automática de la vía de acceso, para que JBuilder considere que los archivos y paquetes pertenecen al proyecto es necesario añadirlos expresamente. Puede añadir archivos y paquetes al proyecto actual mediante el cuadro de diálogo Añadir archivos o paquetes al proyecto.

Hay dos modos de acceder a este cuadro de diálogo. Utilice cualquiera de los dos:



- Pulse el botón Añadir archivos/paquetes/clases de la barra de herramientas del panel de proyecto.
- Haga clic con el botón derecho del ratón en cualquier nodo del panel de proyecto y elija Añadir archivos/paquetes/clases en el menú contextual.

Cuando aparezca el cuadro de diálogo, siga estos pasos:

- 1 Seleccione la ficha Explorador para añadir un archivo, la ficha Paquetes si desea añadir un paquete o la ficha Clases para añadir una clase. Todas estas fichas admiten la selección múltiple.
- 2 Desplácese hasta el archivo, clase o paquete que desea importar.
- 3 Selecciónelo. Cuando se encuentre en el directorio superior de un archivo puede escribir su nombre en lugar de seleccionarlo.
- 4 Haga doble clic en la selección, pulse el botón Aceptar o pulse la tecla *Intro*.
El nuevo nodo aparece en el directorio del proyecto, dentro del panel de proyecto.

Creación de un archivo fuente Java

Hay varias formas de crear archivos fuente de Java con JBuilder. La mayoría de los asistentes pueden crear Archivos. Se accede a algunos de ellos mediante la galería de objetos (ArchivolNuevo), y a otros, desde el menú Asistentes. Más concretamente, el Asistente para clases genera el marco de una nueva clase Java.

Para crear una nueva clase:

- 1 Haga clic con el botón derecho del ratón en un paquete, en el panel de proyecto.
- 2 Seleccione NuevoClase.
- 3 En el cuadro de diálogo que aparece, escriba un nombre para la nueva clase y, en la lista desplegable, especifique el paquete en el que desea colocarla.
- 4 Pulse Aceptar.

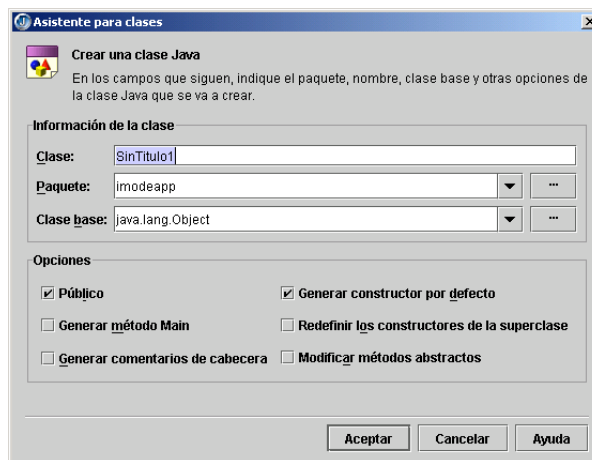
Para crear un archivo fuente Java vacío:

- 1 Seleccione Archivo|Archivo nuevo para abrir el cuadro de diálogo Crear archivo.
- 2 Escriba el nombre del archivo en el campo Nombre.
- 3 Seleccione el tipo de archivo `.java` del menú desplegable o incluya la extensión cuando escriba el nombre.
- 4 Si desea modificar el directorio en el que se guardan los archivos, escriba el nuevo directorio en el campo Directorio o pulse el botón de puntos suspensivos (...) para seleccionar el directorio que desee.
- 5 Pulse Aceptar.

JBuilder crea el archivo y lo abre en el panel de contenido.

Para crear un archivo fuente Java con el Asistente para clases:

- 1 Cree un proyecto nuevo como se indica en [“Creación de proyectos” en la página 2-2](#).
- 2 Para iniciar el Asistente para clases, seleccione Archivo|Nueva clase o haga clic con el botón derecho del ratón en el nodo del proyecto, en el panel del proyecto, y elija NuevoClase.



- 3 Escriba los nombres de paquete, clase y clase base en el Asistente para clases.
- 4 Configure las opciones de exposición, gestión de métodos y comentarios de cabecera.
- 5 Pulse Aceptar.

El archivo `.java` se crea y se añade al proyecto (su nodo aparece en el panel del proyecto). El archivo se abre en el panel de contenido del editor.

Consulte

- [“Cómo añadir elementos a un proyecto” en la página 2-14](#)
- [“Paquetes” en la página 4-6](#)
- [“Configuración de las opciones generales del proyecto” en la página 2-5](#)



Para guardar un archivo tras haberlo modificado, seleccione ArchivarGuardar o pulse el icono Guardar archivo. La vía de acceso y el directorio superior del archivo se muestran en la parte superior de la ventana del Visualizador cuando se selecciona y se abre el archivo. También puede guardar los archivos del proyecto seleccionando ArchivarGuardar todo o pulsando el icono Guardar todo.



Sugerencia

Si se hace clic en la flecha abajo que se encuentra junto al botón Guardar todo, se pueden ver los archivos que guarda el comando Guardar todo. Si lo desea, puede seleccionar en esta lista un solo archivo para guardar.

Eliminación de carpetas, archivos, clases y paquetes del proyecto

Es posible eliminar carpetas, archivos, clases y paquetes del proyecto sin borrarlos de la unidad de almacenamiento mediante el cuadro de diálogo Eliminar del proyecto.

Para eliminar un nodo de su proyecto, seleccione una de las siguientes opciones:

- Pulse con el botón derecho del ratón sobre el nodo que desee eliminar, seleccione Eliminar del proyecto y, a continuación, pulse Aceptar.
- Seleccione el nodo que desee eliminar, pulse el botón Eliminar del proyecto de la barra de herramientas del panel de proyecto y, a continuación, pulse Aceptar.



Nota Los archivos y paquetes creados mediante la localización automática de la vía de acceso a código fuente no se pueden eliminar con ninguna de estas opciones.

Nota Si una carpeta contiene archivos, estos también se eliminan del proyecto. También es posible seleccionar varios nodos y eliminarlos del proyecto.

Eliminación de archivos

Si no se utiliza la función de localización automática de la vía de acceso a código fuente, los archivos no deseados se pueden eliminar del disco si se hace clic con el botón derecho del ratón en el elemento en el panel del proyecto y se selecciona Eliminar.

Precaución Este comando elimina permanentemente los archivos del proyecto y del disco duro del ordenador.

Apertura de archivos fuera de un proyecto

Utilice el comando Archivo|Abrir archivo para abrir un archivo en el Visualizador de aplicaciones sin añadirlo al proyecto abierto.

Para abrir un archivo sin añadirlo al proyecto:

- 1 Seleccione Archivo|Abrir archivo. Se muestra el cuadro de diálogo Abrir archivo.
- 2 Seleccione el archivo que desea abrir.
- 3 Pulse Aceptar. En el Visualizador aparece el contenido del archivo.

Otra opción consiste en utilizar el Buscador de archivos para desplazarse hasta el archivo y hacer doble clic para volver a abrirlo.

Cambio de nombre

Para cambiar el nombre de un proyecto o un archivo:

- 1 Seleccione el proyecto o el archivo en el panel de proyecto.
- 2 Seleccione Proyecto|Renombrar o Archivo|Renombrar, o haga clic con el botón derecho en el panel de proyecto y seleccione Renombrar.
- 3 Escriba el nuevo nombre en el cuadro de diálogo Cambiar nombre y pulse Aceptar.

También se puede cambiar el nombre de un archivo abierto por medio de su pestaña, en la parte superior del panel de contenido:

- 1 Haga clic con el botón derecho en la pestaña del archivo, situada en la parte superior del panel de contenido.
- 2 Seleccione Cambiar nombre.
- 3 Escriba el nuevo nombre en el cuadro de diálogo Cambiar nombre y pulse Aceptar.

Nota El cambio de nombre de un archivo no modifica su tipo. Si desea cambiar la extensión del archivo, utilice Archivo|Guardar como.

Nota No se puede cambiar el nombre a los nodos de paquetes creados por la función de localización automática de la vía de acceso a código fuente.

Precaución Cuando se cambia el nombre de un archivo no se modifican los nombres de paquete y archivo a los que se hace referencia en el código. JBuilder Developer y Enterprise ofrecen la función de perfeccionamiento por cambio de nombre; esto sustituye el nombre antiguo por el nuevo en todos los lugares donde aparece.

Consulte

- [Capítulo 13, “Perfeccionamiento de código”](#)

Adición de una vista de directorio

**Es una función de
JBuilder Developer y
Enterprise**

Se puede añadir un nuevo nodo al panel de proyecto que apunte al directorio de su elección. Por ejemplo, puede tener código que se encuentre enterrado en el fondo de la estructura de su proyecto. Solo tiene que añadir el directorio que contenga ese código como un nodo de proyectos del panel de proyectos. Cuando abra el nodo, tendrá acceso inmediato al código, en lugar de tener que navegar por una complicada estructura de directorios.

Un directorio de navegación es una vista de su directorio o árbol de directorios, que muestra todos los tipos de archivos. Una vez creada una vista de directorios y cuando los cambios se han producido en el contenido actual del directorio o sus subdirectorios, la vista de directorio se actualiza en el panel de proyecto (es posible que se necesite pulsar el botón Actualizar de la barra de proyectos para actualizar el proyecto).

Si se extrae un proyecto mediante CVS, se crea automáticamente un nodo de directorios de navegación que muestra el árbol de directorios completo del proyecto con los subdirectorios de CVS ocultos. Esto proporciona la posibilidad de encontrar en el proyecto todo tipo de archivos, sin tener en cuenta el tipo, y seleccionarlos para realizar operaciones en CVS.

Para añadir un directorio de navegación a un proyecto:

- 1** Seleccione Proyecto|Nuevo directorio de navegación, o bien, pulse con el botón derecho del ratón sobre el nodo de proyectos (el archivo .jpx) en el panel de proyecto y, a continuación, seleccione Nuevo directorio de navegación.
- 2** Desplácese hasta el directorio que desee para que aparezca directamente en el panel de proyecto.
- 3** Pulse Aceptar, y el directorio que haya seleccionado se añade como un nodo al panel de proyecto.
- 4** Abra el nodo para ver los archivos que se incluyen en el directorio.

Cuando se tiene una vista de directorio, se le pueden añadir archivos rápidamente, arrastrándolos desde fuera de JBuilder hasta la vista de directorio, en el panel del proyecto. Desde dentro de JBuilder, se pueden arrastrar los archivos a la vista de directorio para añadir copias.

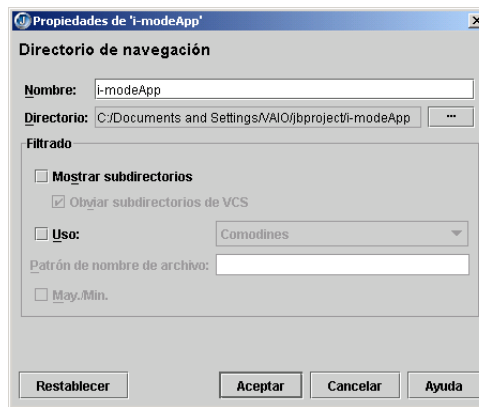
Personalización de las vistas de directorio

Puede personalizar los archivos y subdirectorios que aparecen cuando se abre el directorio de navegación del nodo de proyectos. Por ejemplo, puede elegir que aparezcan solo los archivos `.java` o los archivos `.html`. También puede tener varios directorios de navegación en un mismo proyecto, para que uno de ellos muestre los archivos `.java`, otro los archivos `.html` y otro todos los archivos que empiecen con la letra 'a'.

Para personalizar el directorio de navegación:

- 1 Pulse con el botón derecho del ratón sobre el directorio de navegación del nodo de proyectos en el panel de proyecto.
- 2 Seleccione Propiedades.

Aparece el cuadro de diálogo Propiedades:



- 3 Escriba un nombre descriptivo para el directorio de navegación del nodo de proyectos o mantenga el nombre por defecto, que es el nombre del directorio.
- 4 Utilice las opciones de filtrado para filtrar los archivos y los directorios que deben aparecer al abrir el directorio de navegación del nodo de proyectos. Estos son ejemplos de los modelos de archivos:

```
*.java
myfile?.java
file??.*
```

Nota No puede encadenar varios modelos de archivos juntos, como `*.java;*.cpp`.

Si desea obtener información más completa sobre el uso de estas opciones, pulse el botón de ayuda de este cuadro de diálogo.

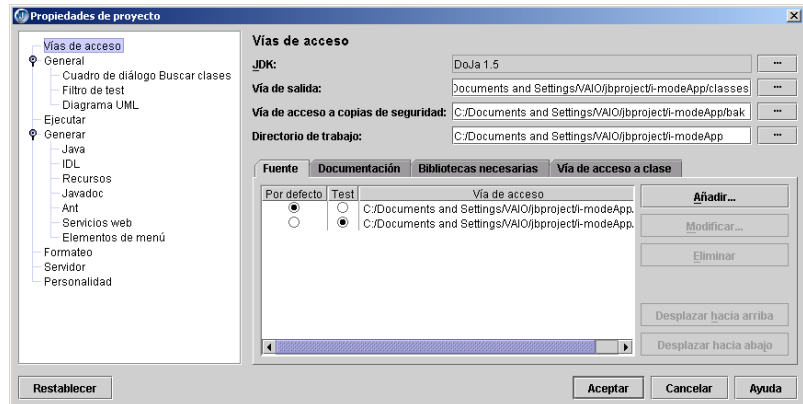
- 5 Pulse Aceptar.

Definición de las propiedades del proyecto

Las propiedades del proyecto determinan la forma en que se compila. Mediante el cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto), puede definir la configuración de las vías de acceso, establecer una configuración de ejecución para el proyecto, definir el modo en que se ha de crear el proyecto, personalizar la apariencia de los diagramas UML, especificar las opciones del servidor y mucho más.

Para establecer las propiedades del proyecto:

- 1 Haga clic con el botón derecho en el nombre de archivo con extensión .jpx, dentro del panel de proyecto, y seleccione Propiedades para abrir el cuadro de diálogo Propiedades de proyecto. También puede seleccionar el proyecto y elegir Proyecto | Propiedades de proyecto. Aparece el cuadro de diálogo Propiedades de proyecto.
- 2 En el árbol de la izquierda, seleccione el nodo de las opciones que desea configurar. En esta imagen, se ha seleccionado la ficha Vías de acceso:



A continuación, se incluye una breve descripción de lo que se puede hacer en cada una de las fichas del cuadro de diálogo Propiedades de proyecto:

- **Ficha Vías de acceso:** configuración de las vías de acceso del proyecto a la versión del JDK, vía de salida, vías a copia de seguridad, directorio de trabajo, archivos fuente, comprobación, documentación y bibliotecas necesarias.
- **Ficha General:** establece opciones para la codificación, la activación de la recopilación automática de paquetes fuente y la modificación los campos de los campos de clase Javadoc que generan los asistentes, así como la configuración de la opción de incluir referencias de las bibliotecas del proyecto.
- **Ficha Ejecutar:** selecciona o crea la configuración que se utiliza durante la ejecución o la depuración.

- **Ficha Generar:** define las opciones del compilador para la creación de un proyecto. Esto incluye la información de depuración y la copia selectiva de recursos.
- **Ficha Formato:** establece las opciones de formato del código. JBuilder puede acelerar la codificación formateando el código automáticamente de acuerdo con sus indicaciones. Es una función de JBuilder Enterprise.
- **Ficha Servidor:** configura las opciones del servidor. Es una función de JBuilder Enterprise.
- **Ficha Personalidad:** determina los grupos de características (*personalidades*) que aparecen en JBuilder. La configuración de las personalidades de JBuilder permite personalizar JBuilder para que sólo se vean las características necesarias. Es una función de JBuilder Enterprise.

Nota

También es posible definir estas opciones para los proyectos que se creen más adelante. Para ello, utilice el cuadro de diálogo Propiedades por defecto para proyectos (Proyecto1Propiedades por defecto para proyectos) cuando selecciona el proyecto por defecto como plantilla de documento en el Asistente para proyectos.

Configuración del JDK

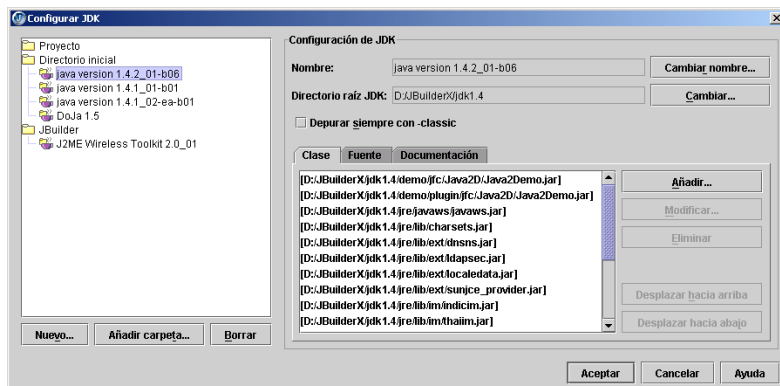
En la ficha Vías de acceso se pueden configurar la versión del JDK, distintas vías de acceso para el proyecto y las vías de acceso a bibliotecas necesarias.

Puede configurar la versión JDK para el proyecto en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto y añadir, modificar y eliminar los JDK en el cuadro de diálogo Configurar JDK. Consulte [“Cambiar el JDK” en la página 2-25](#).

Modificación del JDK

La versión actual del JDK se puede modificar de la siguiente manera:

- 1 Seleccione Herramientas|Configurar JDK para abrir el cuadro de diálogo Configurar JDK.



- 2 Pulse el botón Cambiar, que se encuentra a la derecha del campo Directorio del JDK. Aparece el cuadro de diálogo Seleccione un directorio.
- 3 Desplácese hasta el JDK de destino.
- 4 Pulse Aceptar para cambiar el JDK.
Tome nota del nuevo nombre del JDK y de la vía de acceso de raíz que aparecen en el cuadro de diálogo Configurar JDK.
- 5 Pulse Aceptar para cerrar el cuadro de diálogo Configurar JDK.

Depuración con -classic

La opción Depurar siempre con -classic del cuadro de diálogo Configurar JDK proporciona un mejor rendimiento a los usuarios con versiones de la MVJ anteriores a la 1.3.1. JBuilder comprueba automáticamente si esta opción va a mejorar el rendimiento, y activa o desactiva la casilla según el resultado. Esta función está disponible en todas las ediciones de JBuilder.

Cuando se efectúa la evaluación, JBuilder lleva a cabo dos comprobaciones:

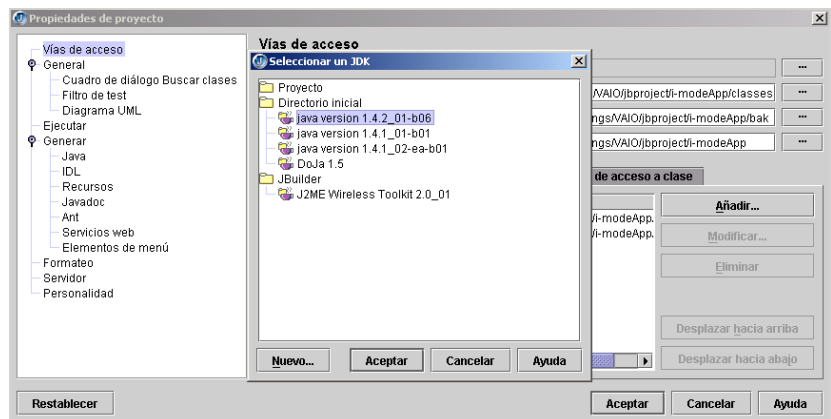
- 1 ¿Tiene instalada la MV Classic?
- 2 Si es así, ¿la versión de MVJ es anterior a la 1.3.1?

Esta selección se modifica cuando se definen parámetros de MV como `native`, `hotspot`, `green` y `server`.

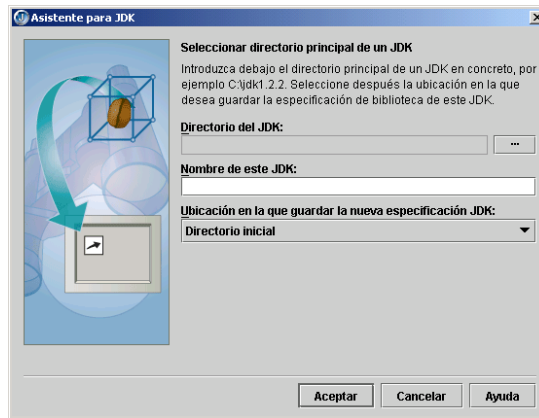
Cambiar el JDK

JBuilder permite cambiar de JDK. También es posible añadir, modificar y eliminar JDK. Para cambiar de JDK, siga estos pasos:

- 1 Seleccione Proyecto/Propiedades de proyecto y, a continuación, escoja Vías de acceso en el árbol.
- 2 Pulse el botón de puntos suspensivos (...), a la derecha de la versión de JDK. Aparece el cuadro de diálogo Seleccione un JDK:



- 3 Si en la lista aparece el JDK de destino, selecciónelo y pulse Aceptar. Si no aparece en la lista, pulse Nuevo para abrir el Asistente para JDK.



- a Haga clic en el botón de puntos suspensivos y desplácese hasta el directorio raíz del JDK que desea añadir a la lista. Pulse Aceptar. El campo Nombre del JDK se rellena automáticamente.
 - b Seleccione el lugar donde desea guardar las especificaciones JDK:
 - **Directorio inicial:** guarda las especificaciones JDK en un archivo `.library`, en el subdirectorio `.jbuilder` del directorio raíz del usuario. Guarde en esta ubicación si desea que el JDK esté disponible para todos los proyectos.
 - **JBuilder:** guarda las especificaciones JDK en un archivo `.library`, en el directorio `jbuilder`. Los diferentes usuarios que utilizan JBuilder en una red o lo comparten en una máquina pueden acceder a los JDK de esta carpeta. Es una función de JBuilder Developer y Enterprise.
 - **Proyecto:** guarda las especificaciones JDK en un archivo `.library`, en el directorio del proyecto actual. Guarde en esta ubicación si desea que el JDK sólo esté disponible para este proyecto. Es una función de JBuilder Developer y Enterprise.
 - **Carpeta definida por el usuario:** guarda las especificaciones JDK en una carpeta definida por el usuario o un directorio compartido. Para que la carpeta aparezca en la lista desplegable es necesario añadirla (elija Herramientas|Configurar JDK y pulse Añadir carpeta). Es una función de JBuilder Enterprise.
 - c Pulse Aceptar. La especificación JDK se añade al directorio especificado en el cuadro de diálogo Seleccione un JDK.
- 4 Pulse Aceptar para cerrar el cuadro de diálogo Seleccione un JDK. La vía de acceso al JDK se actualiza con la nueva.
 - 5 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

- 6** Guarde el proyecto. La versión de JDK se actualiza en el archivo de proyecto.

Sugerencia Los archivos JDK se pueden añadir, modificar y eliminar por medio de Herramientas|Configurar JDK. También se pueden modificar las propiedades de proyecto por defecto (Proyecto|Propiedades por defecto para proyectos), lo que cambia el JDK de todos los proyectos posteriores.

Configuración de los JDK

En el cuadro de diálogo Configurar JDK (Herramientas|Configurar JDK) se pueden añadir, modificar y eliminar archivos JDK. Este cuadro de diálogo permite:

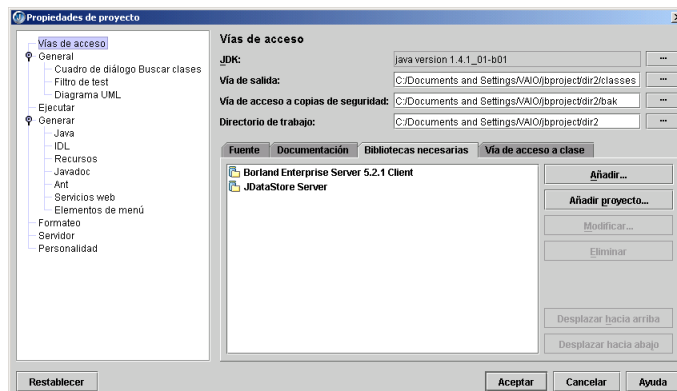
- Asignar un nombre al JDK por medio del botón Renombrar.
- Añadir, modificar, eliminar y reorganizar archivos de clase, fuente y documentación de JDK.
- Abrir el Asistente para JDK y añadir JDK por medio del botón Nuevo.
- Añadir una carpeta que otros usuarios puedan utilizar. Es una función de JBuilder Enterprise.
- Eliminar un JDK de la lista.

Consulte

- Botón Ayuda del cuadro de diálogo Configurar JDK.
- Botón Ayuda del Asistente para JDK.

Configuración de vías de acceso a bibliotecas necesarias

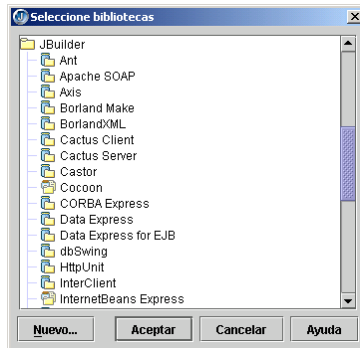
En la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto se pueden definir las bibliotecas que se utilizarán al compilar. JBuilder coloca las bibliotecas seleccionadas en la vía de acceso a clases. Para añadir, modificar, eliminar y reordenar bibliotecas, seleccione la pestaña Bibliotecas necesarias.



Se pueden seleccionar bibliotecas en la lista Bibliotecas necesarias de la ficha Vías de acceso y modificarlas, eliminarlas o cambiar su orden en la lista.

Nota Se busca en las bibliotecas por el orden en que figuran en la lista. Para cambiar el orden de las bibliotecas, seleccione una y pulse Desplazar hacia arriba o Desplazar hacia abajo.

Cuando se pulsa el botón Añadir se abre el cuadro de diálogo Seleccionar bibliotecas, en el que se pueden elegir las bibliotecas que se van a añadir al proyecto. Pulse Nuevo en el cuadro de diálogo para abrir el Asistente para bibliotecas y crear una biblioteca.



El botón Añadir proyecto abre el cuadro de diálogo Añadir proyecto JBuilder, en el que se selecciona un proyecto que se desee añadir como una biblioteca necesaria.

**Es una función de
JBuilder Enterprise**

Siga utilizando los botones Añadir y Añadir proyecto hasta que haya seleccionado todas las bibliotecas que desee especificar como necesarias. Cuando termine, pulse Aceptar.

También se pueden configurar las bibliotecas por medio de Herramientas Configurar bibliotecas.

Consulte

- [“Las bibliotecas” en la página 4-1](#)

Trabajo con varios proyectos

Es posible trabajar simultáneamente con varios proyectos dentro del entorno de desarrollo de JBuilder. Todos los proyectos abiertos están disponibles en la lista desplegable Proyectos.

Si tiene JBuilder Developer o Enterprise, puede también agrupar varios proyectos en grupos de proyectos. Los grupos de proyectos son particularmente útiles cuando se está trabajando con proyectos relacionados. Si desea más información sobre grupos de proyectos, consulte el [Capítulo 3](#), “Los grupos de proyectos”.

Cambio de un proyecto a otro

Si en el Visualizador hay varios proyectos abiertos, sólo el proyecto que se encuentre activo es visible en el panel de proyecto. Para pasar a otro proyecto abierto, selecciónelo en la lista desplegable Proyectos de la barra de herramientas que está por encima del panel de proyecto. O bien, seleccione Ventana|Seleccionar proyecto para ver una lista de proyectos abiertos y elegir en la que desea abrirlo.

También se pueden cambiar los proyectos mediante el menú Windows, que mantiene una lista de todos los proyectos abiertos. El menú Windows también recoge todos los archivos abiertos en el proyecto actual, para que pueda cambiar entre ellos. La apertura de archivos de este modo resulta muy útil si se prefiere no ver los nombres de archivo en las pestañas de archivo.

Guardar varios proyectos

Para guardar los cambios realizados en todos los archivos y proyectos, seleccione Archivo|Guardar todo. Se guardan todos los archivos.

Visualización de recopilatorios desde el panel de proyecto

En JBuilder Developer y Enterprise se puede seleccionar y ampliar un nodo de recopilatorio en el panel del proyecto para ver los archivos que contiene. Cuando se hace doble clic en un nodo de recopilatorio, este se abre en el panel de contenido. El visualizador que utiliza JBuilder para mostrar el archivo depende de su tipo. Por ejemplo, si el recopilatorio contiene un archivo GIF y un archivo HTML, la ficha Ver del panel de contenido usa el visualizador de imágenes de JBuilder para mostrar el archivo GIF, y el navegador para mostrar el archivo HTML.

Información adicional acerca de los proyectos

Mientras trabaja con los proyectos en JBuilder, debe comprender cómo utiliza JBuilder las vías de acceso, para que pueda sacar mayor provecho de funciones tales como las bibliotecas o CodeInsight (consulte el [Capítulo 4, “Gestión de las vías de acceso”](#).)

JBuilder permite agrupar los proyectos en grupos de proyectos. Si desea leer más acerca de los grupos de proyectos, consulte el [Capítulo 3, “Los grupos de proyectos”](#). La capacidad de agrupar proyectos es una característica de JBuilder Developer y Enterprise.



Los grupos de proyectos

**Es una función de
JBuilder Developer y
Enterprise**

Los grupos de proyectos son contenedores de proyectos, y pueden resultar de gran utilidad cuando se trabaja con proyectos relacionados. Por ejemplo, puede tener dos proyectos que tengan dependencias entre ellos, como un cliente y un servidor. Otra agrupación lógica podría incluir proyectos que utilizan los mismos archivos fuente pero que tienen configuraciones diferentes, como servidores de aplicaciones de destino distintos o diferentes JDK. Además, los grupos de proyectos proporcionan otras ventajas, como la fácil navegación entre proyectos y la generación de proyectos como grupos.

Los grupos de proyectos solo pueden contener proyectos, y no otros grupos de proyectos. El grupo de proyectos se guarda como un archivo XML con la extensión `.jpr` y, por defecto, en la raíz del directorio `jproject`. Los proyectos, por sí mismos, no son conscientes de la existencia de los grupos de proyectos, y se pueden abrir simultáneamente de forma autónoma y dentro de un grupo de proyectos. Los cambios que realice en un proyecto dentro de un grupo también se realizan en el proyecto abierto de forma autónoma.

Creación de grupos de proyectos

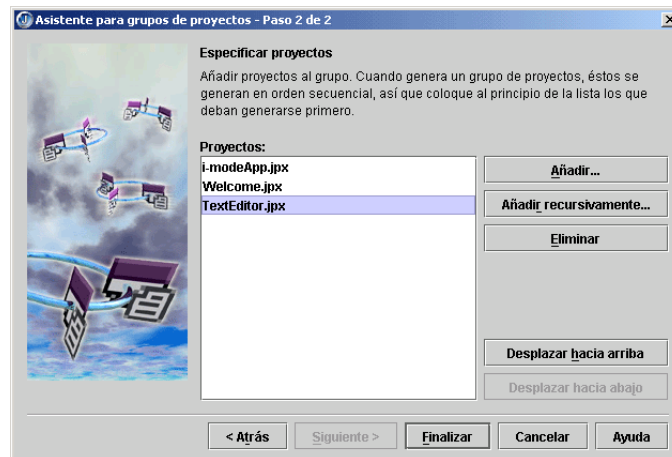
JBuilder proporciona el Asistente para grupos de proyectos, que se encuentra disponible en la ficha Proyecto de la galería de objetos (Archivo|Nuevo), para crear grupos de proyectos. Puede crear un grupo de proyectos vacío o rellenarlo con proyectos al completar el asistente. Se pueden añadir proyectos a un grupo de proyectos en cualquier momento, como se describe en [“Adición y eliminación de proyectos de los grupos de proyectos” en la página 3-3](#). Una vez creado el grupo de proyectos, puede añadir también

nuevos proyectos con el Asistente para proyectos. Al crear el nuevo proyecto, seleccione la opción Añadir proyecto al grupo activo para que se incluya en el grupo de proyectos.

Los proyectos de un grupo se generan en el orden en que aparecen en el panel del proyecto. El orden de generación se indica en el Paso 2 del Asistente para proyectos, y se puede modificar en cualquier momento en el cuadro de diálogo Propiedades del grupo de proyectos. Para obtener más información sobre grupos de proyectos, consulte el [Capítulo 6, “Generación de programas en Java”](#).

Para crear un grupo de proyectos con la ayuda del Asistente para grupos de proyectos, siga los pasos siguientes:

- 1 Seleccione Archivo|Nuevo (o pulse el botón Nuevo en la barra de herramientas de JBuilder) y escoja Proyecto en el árbol para abrir la ficha Proyecto de la galería de objetos.
- 2 Haga doble clic en el icono Grupo de proyectos para iniciar el Asistente para grupos de proyectos.
- 3 Modifique el nombre del archivo de grupo de proyectos y/o la ubicación del archivo en el campo Nombre del archivo.
- 4 Pulse Siguiente para pasar al siguiente paso.
- 5 Realice una de las operaciones siguientes:
 - a Pulse el botón Añadir, busque un proyecto y selecciónelo. Pulse Aceptar para añadir el proyecto. Repita esta acción para añadir otro proyecto.
 - b Pulse el botón Añadir recursivamente, seleccione el directorio que va a examinar y, a continuación, pulse Aceptar. JBuilder examina el directorio seleccionado y todos sus subdirectorios, y añade todos los archivos de proyecto (.jpx) al grupo de proyectos.



- 6 Seleccione un proyecto de la lista y utilice los botones Desplazar hacia arriba o Desplazar hacia abajo para reorganizar la lista de proyectos. Los

proyectos se crean y aparecen en el panel del proyecto en el orden de la lista.

7 Pulse Aceptar para cerrar el asistente.

El archivo de grupo de proyectos aparece en la parte superior del panel del proyecto y, debajo, los proyectos en el orden en que se han añadido. Haga doble clic sobre el nodo del grupo de proyectos para ampliarlo y contraerlo. Solo puede haber un proyecto activo en el grupo en cada momento. Un proyecto se puede abrir simultáneamente de forma independiente o dentro de un grupo de proyectos. Haga doble clic sobre un proyecto para convertirlo en el proyecto activo dentro de un grupo. Amplíe el nodo del proyecto para ver el contenido. Observe que el proyecto activo y abierto aparece en negrita en el panel del proyecto, y en negrita o cursiva en la lista desplegable de proyectos, según el aspecto elegido para la interfaz. Para obtener más información sobre el desplazamiento por los grupos de proyectos, consulte [“Desplazamiento por los grupos de proyectos” en la página 3-4](#).

Una vez creado un grupo de proyectos, también puede cerrarlo mediante ArchivarCerrar proyectos, con el botón cerrar de la barra de herramientas del panel del proyecto o haciendo clic con el botón derecho en el nodo del grupo de proyectos en el panel del proyecto y seleccionando Cerrar el grupo de proyectos <Nombre.jpgr>. Para abrir un grupo de proyectos, utilice ArchivarAbrir proyecto o ArchivarAbrir archivo. Por defecto, los grupos de proyectos se guardan en el directorio `jbproject` por lo que tiene que buscar los grupos de proyectos (archivos con una extensión `.jpgr`) en `jbproject` a menos que indicara una ubicación diferente para el grupo de proyectos cuando lo creó con el Asistente para grupos de proyectos.

Adición y eliminación de proyectos de los grupos de proyectos

Puede añadir proyectos a o eliminarlos de un grupo de proyectos en cualquier momento. Existen varias formas de hacer esto:

- Menú de Proyecto
- Menú contextual del panel de proyecto
- Barra de herramientas del panel del proyecto
- Propiedades del grupo de proyectos

Para añadir un proyecto al grupo de proyectos abierto, lleve a cabo una de las siguientes acciones:

- Seleccione el grupo de proyectos en el panel del proyecto y siga una de las siguientes opciones:
 - Seleccione Proyecto!Añadir proyecto.
 - Pulse el botón Añadir en la barra de herramientas del panel del proyecto y busque el proyecto que desee añadir.

- Pulse el grupo de proyectos con el botón derecho del ratón y seleccione Añadir proyecto.
- Seleccione Proyecto|Propiedades del grupo de proyectos, pulse el botón Añadir y elija el proyecto que desee añadir.

Para eliminar un proyecto de un grupo de proyectos abierto, lleve a cabo una de las siguientes acciones:

- Seleccione el proyecto en el panel del proyecto y, a continuación, Proyecto|Eliminar del grupo de proyectos.
- Pulse con el botón derecho del ratón sobre el proyecto en el panel del proyecto y seleccione Eliminar del proyecto en el menú contextual.
- Seleccione el proyecto en el panel del proyecto y pulse el botón Eliminar de la barra de herramientas del panel del proyecto.
- Seleccione Proyecto|Propiedades del grupo de proyectos, elija el proyecto que desee eliminar y, a continuación, pulse el botón Eliminar.

Cambio del orden de un grupo de proyectos

Es posible cambiar con mucha rapidez el orden de los proyectos de un grupo. Arrastre el proyecto deseado a otra posición dentro del grupo de proyectos.

Desplazamiento por los grupos de proyectos

Una de las ventajas que se deriva de la colocación de proyectos en grupos es el fácil desplazamiento. Aunque sólo puede haber un proyecto activo en cada momento, puede desplazarse rápidamente de un proyecto abierto a otro dentro del grupo. Puede seleccionar otro proyecto en la lista desplegable del panel del proyecto.

Adición de proyectos como bibliotecas necesarias

**Es una función de
JBuilder Enterprise**

A menudo, los proyectos dentro de los grupos de proyectos tienen dependencias entre ellos. Si tiene un proyecto que depende de otro, puede añadir el proyecto del cual depende el suyo a la lista de bibliotecas necesarias de su proyecto.

Para añadir un proyecto como una biblioteca necesaria:

- 1 Seleccione Proyecto|Propiedades de proyecto y abra la ficha Vías de acceso.
- 2 Haga clic en la pestaña Bibliotecas necesarias.
- 3 Pulse el botón Añadir proyecto.
- 4 Seleccione el proyecto que desea añadir.

5 Pulse Aceptar.

El proyecto que haya indicado se añade en la parte inferior de la lista de bibliotecas necesarias.

6 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

Si el proyecto especificado como biblioteca necesaria forma parte del grupo de proyectos, debe encontrarse en el grupo por delante del proyecto que depende de él. De ese modo se sabe que el proyecto necesario se genera primero. Consulte el [Capítulo 6](#), “[Generación de programas en Java](#)” si desea obtener más información sobre la generación de grupos de proyectos.



Gestión de las vías de acceso

Las vías de acceso proporcionan al programa las bibliotecas y el JDK necesarios para su ejecución. Las vías de acceso son la infraestructura del desarrollo de programas en Java. Cuando se establece un JDK se indica al programa su ubicación. Cuando se crea una biblioteca, se recopila un conjunto de vías de acceso que va a necesitar el programa. Siempre que un archivo hace referencia a otro utiliza una vía de acceso para llegar a él.

En esta sección se explica cómo estructura JBuilder las vías de acceso, cómo se deben manipular en el cuadro de diálogo Propiedades de proyecto y cómo se utilizan las herramientas basadas en vías de acceso, como las bibliotecas y las funciones de CodeInsight.

Las bibliotecas

JBuilder utiliza las bibliotecas para buscar los elementos que necesita para ejecutar los proyectos, así como para examinar el código fuente, mostrar los comentarios Javadoc, utilizar el diseñador visual, aplicar CodeInsight y compilar el código. Las bibliotecas son recopilaciones de vías de acceso que incluyen archivos de clase, código fuente y documentación. Las bibliotecas son estáticas, no dinámicas. Las vías de acceso de las bibliotecas se suelen incluir en archivos JAR o ZIP, pero también pueden residir en directorios.

Cuando se añaden bibliotecas a JBuilder, se agregan a la vía de acceso a clases, de modo que JBuilder pueda encontrarlas. Se busca en las bibliotecas por el orden en que figuran en la lista.

El orden de las bibliotecas se puede cambiar en el cuadro de diálogo Configurar bibliotecas (Herramientas|Configurar bibliotecas) y en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto (Proyecto|Propiedades de proyecto).

Consulte

- [“Cómo construye JBuilder las vías de acceso” en la página 4-9](#)

La configuración de las bibliotecas se guarda en archivos `.library`, que pueden almacenarse en varias ubicaciones:

Es una función de JBuilder Developer y Enterprise.	Directorio inicial	Guarda el archivo <code>.library</code> en el subdirectorio <code><.jbuilder></code> del directorio inicial del usuario.
	JBuilder	Guarda el archivo <code>.library</code> en el directorio <code><jbuilder>/lib</code> . Los usuarios que utilizan JBuilder en red o lo comparten en una única máquina pueden acceder a las bibliotecas de esta carpeta.
Es una función de JBuilder Developer y Enterprise.	Proyecto	Guarda el archivo <code>.library</code> en el directorio del proyecto actual. Si se utiliza el control de versiones en Enterprise, el archivo <code>.library</code> aparece junto con los demás archivos de proyecto.
Es una función de JBuilder Enterprise.	Carpeta definida por el usuario	Guarda el archivo <code>.library</code> en una carpeta definida por el usuario o en un directorio compartido. Para que la carpeta nueva aparezca en la lista desplegable es necesario añadirla en el cuadro de diálogo Configurar bibliotecas.

Adición y configuración de bibliotecas

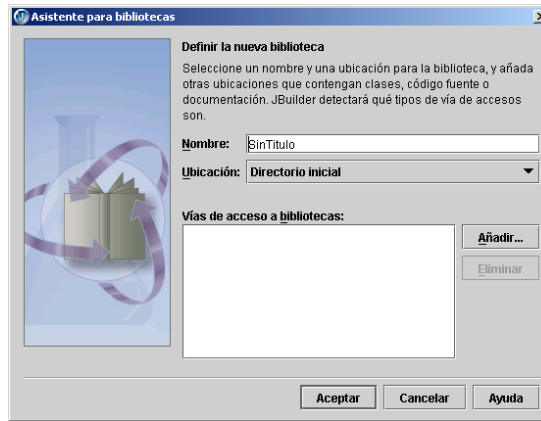
Existen varias maneras de añadir bibliotecas al proyecto. Es conveniente empezar por reunir los archivos en recopilatorios JAR, sobre todo si se tiene intención de distribuir el programa.

Para crear una nueva biblioteca:

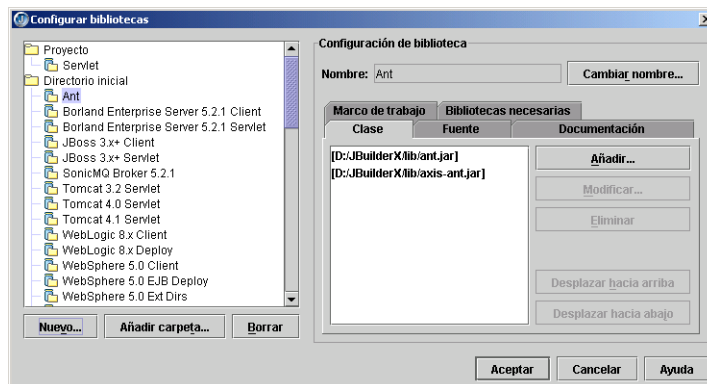
- 1 Seleccione Herramientas|Configurar bibliotecas. Se abre el cuadro de diálogo Configurar bibliotecas.

El panel izquierdo permite buscar las bibliotecas disponibles. El panel derecho muestra la configuración de la biblioteca seleccionada.

- 2 Pulse el botón Nuevo, bajo el panel izquierdo, para abrir el Asistente para bibliotecas.



- 3 Escriba un nombre en el campo Nombre.
- 4 Seleccione en la lista desplegable el lugar donde desea guardar la configuración de la biblioteca: Proyecto, Directorio inicial, JBuilder o Definido por el usuario.
- 5 Pulse el botón Añadir y seleccione vías de acceso que contengan archivos de clase, fuente y documentación. JBuilder determina automáticamente la vía de acceso correcta de los archivos. Pulse Aceptar. Observe que la selección aparece en la lista Vía de acceso a bibliotecas.
- 6 Pulse Aceptar con el fin de cerrar el Asistente para bibliotecas. Observará que la biblioteca se guarda en las vías de acceso a clases, fuente y documentación adecuadas, en el cuadro de diálogo Configurar bibliotecas. Este cuadro de diálogo también permite añadir, modificar, eliminar y reordenar las listas de bibliotecas. JBuilder Enterprise también incluye una función Añadir carpeta que permite añadir un marco como biblioteca. Si desea obtener más información acerca de la adición de marcos como bibliotecas, consulte "Marcos de trabajo JSP" en *Guía del desarrollador de aplicaciones web*.



- 7 Pulse Aceptar o *Intro* para cerrar el cuadro de diálogo Configurar bibliotecas.

Si desea añadir la biblioteca a un proyecto, consulte “[Configuración de vías de acceso a bibliotecas necesarias](#)” en la [página 2-27](#).

También es posible añadir bibliotecas en el cuadro de diálogo Propiedades de proyecto.

- 1 Seleccione Proyecto|Propiedades de proyecto.
- 2 Seleccione la pestaña Bibliotecas necesarias de la ficha Vías de acceso y pulse Añadir.
- 3 Pulse el botón Nuevo para abrir el Asistente para bibliotecas.

Consulte

- “[Configuración de vías de acceso a bibliotecas necesarias](#)” en la [página 2-27](#)
- “Using JAR Files: The Basics” (en inglés) en <http://java.sun.com/docs/books/tutorial/jar/basics/index.html>
- “Asistente para bibliotecas” en la ayuda en línea

Modificación de bibliotecas

Para modificar una biblioteca:

- 1 Seleccione Herramientas|Configurar bibliotecas.
- 2 Seleccione la biblioteca que desea modificar de la lista de bibliotecas.
- 3 Seleccione la pestaña Clase, Fuente o Documentación, Marco de trabajo o Bibliotecas necesarias para elegir la vía de acceso de la biblioteca que desea modificar.
- 4 Seleccione la biblioteca y pulse Modificar.
- 5 Busque el archivo o la carpeta en el cuadro de diálogo Seleccionar directorio. Pulse Aceptar.
- 6 Pulse Añadir para buscar la biblioteca que desea añadir.
- 7 Para borrar una vía de acceso a una biblioteca, selecciónela y pulse Eliminar.
- 8 Para reorganizar las vías de acceso a bibliotecas, seleccione una de ellas y pulse Desplazar hacia arriba o Desplazar hacia abajo.

Sugerencia JBuilder busca las bibliotecas por el orden con que figuran en la lista.

- 9 Pulse Aceptar o *Intro* para cerrar el cuadro de diálogo Configurar bibliotecas.

Adición de proyectos como bibliotecas necesarias

Es una función de JBuilder Enterprise.

Los proyectos pueden tener, a su vez, otros proyectos que dependen de ellos. Si tiene un proyecto que depende de otro, puede añadir el proyecto del cual depende el suyo a la lista de bibliotecas necesarias de su proyecto.

Para añadir un proyecto como una biblioteca necesaria:

- 1** Seleccione Proyecto|Propiedades de proyecto y abra la ficha Vías de acceso.
- 2** Haga clic en la pestaña Bibliotecas necesarias.
- 3** Pulse el botón Añadir proyecto.
- 4** Seleccione el proyecto que desea añadir.
- 5** Pulse Aceptar.

El proyecto que haya indicado se añade en la parte inferior de la lista de bibliotecas necesarias.

- 6** Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

Presentación de la lista de bibliotecas

Hay tres colores posibles para las bibliotecas que aparecen en las listas de los cuadros de diálogo de JBuilder:

Tabla 4.1 Colores de las listas de bibliotecas:

Color	Descripción	Resolución de problemas
Negro	La biblioteca se ha definido correctamente.	
Rojo	No se encuentra la definición de la biblioteca.	Normalmente indica que el proyecto hace referencia a una biblioteca que no se ha definido todavía. También puede significar que la definición de la biblioteca es defectuosa: se ha definido sin vías de acceso o existen varias bibliotecas con ese nombre.
Gris	La utilización de esta biblioteca requiere una actualización.	Es necesario actualizar su edición de JBuilder con el fin de utilizar esta biblioteca. También puede significar que hay nombres de bibliotecas repetidos.

Paquetes

Java agrupa los archivos `.java` y `.class` en un *paquete*. Los archivos que constituyen el código fuente de un paquete de Java se encuentran en el subdirectorio (`src`), y los archivos compilados, en el subdirectorio (`clases`). En la generación de aplicaciones JBuilder utiliza el nombre del proyecto como nombre por defecto para el paquete del Asistente para aplicaciones o el Asistente para applets. Por ejemplo, si el proyecto se llama `sintitulo1.jpr`, el Asistente para aplicaciones y el Asistente para applets proponen la utilización del nombre `sintitulo1`. Los nombres de paquete propuestos se basan siempre en el nombre del proyecto.

Vamos a tomar como referencia un proyecto de ejemplo para ver cómo influye el nombre del paquete en la estructura del archivo.

Nota En estos ejemplos se utilizan las vías de acceso propias de la plataforma UNIX. Si desea información sobre la forma en que se documentan aquí las vías de acceso, consulte [“Convenciones de la documentación” en la página 1-5](#).

Para organizar el proyecto, supongamos que se encuentra en una carpeta llamada Proyecto de ejemplo. Esta carpeta de proyecto contiene un archivo de proyecto (`africa.jpx`), un directorio `clases` y un directorio `src`:



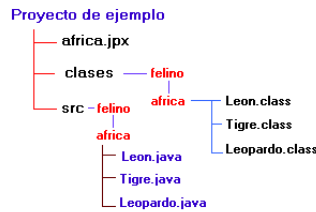
Al crear este proyecto puede ser conveniente crear paquetes propios que contengan los archivos de código fuente y clase relacionados. En este ejemplo, `africa.jpx` contiene un nombre de paquete de `felino.africa`. Este paquete contiene archivos de código fuente sobre varios felinos africanos: Leones, Tigres y Leopardos.

Los archivos de clase, guardados en una estructura de directorios que coincide con el nombre del paquete, se guardan en el subdirectorio `clases`, dentro del proyecto. El subdirectorio `src`, que contiene los archivos `.java`, tiene la misma estructura que el subdirectorio de clases.



Si las clases que contiene este proyecto son `Leon.class`, `Tigre.class` y `Leopardo.class`, se encuentran en `clases/felino/africa`. Los archivos fuente

Leon.java, Tigre.java y Leopardo.java se encuentran en src/felino/africa, tal y como indicamos.:



Ubicación del archivo .java = vía de acceso a archivos fuente + vía de acceso a paquetes

Es importante comprender qué datos utiliza JBuilder para generar el directorio en que se encuentra un archivo .java determinado. La vía de acceso a archivos fuente determina la primera parte de la vía de acceso del directorio. La vía de acceso a archivos fuente se define en cada proyecto y se puede modificar en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto.

Prosiguiendo con el Proyecto de ejemplo, la vía de acceso a archivos fuente de Leon.java es:

```
/<raíz>/<nombredeusuario>/jbproject/Proyecto de ejemplo/src.
```

Nota Si desea una definición de <directorio> raíz, consulte [“Convenciones de la documentación” en la página 1-5](#).

La segunda parte de la vía de acceso al directorio está determinada por el nombre del paquete, que en este caso es felino.africa.

Nota En la nomenclatura de Java se utiliza un punto (.) para separar las partes de los paquetes.

La ubicación del archivo .java en el caso de Leon.java es:

```
/<raíz>/<nombredeusuario>/jbproject/Proyecto de ejemplo/src/felino/africa/  
Leon.java
```

Consulte

- [“Cómo construye JBuilder las vías de acceso” en la página 4-9](#)

Ubicación del archivo .class = vía de salida + vía de acceso a paquetes

La ubicación del directorio del archivo .class está formada por la vía de salida y el nombre del paquete. La vía de acceso de salida es la "raíz" a la que JBuilder añade las vías de acceso de los paquetes para crear la estructura de directorios de los archivos .class generados por el compilador. La vía de

acceso a archivos fuente se define en cada proyecto y se puede modificar en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto.

En Proyecto de ejemplo, la vía de salida de `Leon.class` es:

```
/<raíz>/<nombredeusuario>/jbproject/Proyecto de ejemplo/classes
```

La segunda parte de la vía de acceso al directorio está determinada por el nombre de paquete, que en este caso es `felino.africa`.

Como se indica a continuación, la ubicación `.class` de `Leon.class` es:

```
/<raíz>/<nombredeusuario>/jbproject/Proyecto de ejemplo/classes/felino/  
africa/Leon.class
```

Consulte

- [“Cómo construye JBuilder las vías de acceso” en la página 4-9](#)

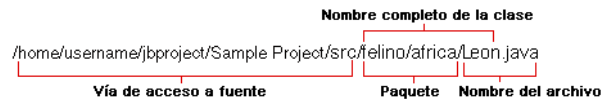
Utilización de paquetes en JBuilder

Cuando se hace referencia a clases desde un paquete, se puede utilizar una sentencia `import`. Las sentencias de importación permiten hacer referencia a cualquier clase del paquete importado utilizando solamente el nombre corto en el código. (Los diseñadores y asistentes de JBuilder añaden automáticamente las sentencias de importación.) A continuación se muestra un ejemplo de una sentencia `import`:

```
import felino.africa.*;
```

Si esta sentencia de importación se incluye en el código fuente es posible referirse a la clase `Leon` simplemente como `Leon` en el cuerpo del código.

Si no se importa el paquete, es necesario hacer referencia a una clase determinada en el código fuente, con su nombre de clase completo. Como se indica en este esquema, el nombre de clase completo de `Leon.java` es `felino.africa.Leon` (nombre de paquete + nombre de clase sin la extensión).



Se pueden excluir paquetes del proceso de creación de forma selectiva.

Consulte

- [“Filtrado de paquetes” en la página 6-39](#)

Directrices de nomenclatura de paquetes

Se recomienda utilizar las siguientes directrices de nomenclatura de paquetes en todos los programas en Java. Con el fin de mantener la coherencia, la legibilidad y la capacidad de mantenimiento, los nombres de los paquetes deben seguir estas reglas:

- Una palabra
- Singular mejor que plural
- Todo en minúsculas, incluso si hay más de una palabra (por ejemplo, `nombreproyectocuatropalabras` y **no** `NombreProyectoCuatroPalabras`).

Si los paquetes se van a utilizar fuera del grupo, los nombres han de comenzar con el nombre de un dominio de Internet invertido. Por ejemplo, si desea utilizar el nombre de dominio `foo.dominio.com`, los nombres de los paquetes deben ir precedidos por `com.dominio.foo`.

Cómo construye JBuilder las vías de acceso

El IDE de JBuilder utiliza varias vías de acceso durante el proceso:

- Vía de acceso a archivos fuente
- Vía de salida
- Vía de acceso a clases
- Vía de búsqueda
- Vía de acceso a documentos
- Vía de acceso a las copias de seguridad
- Directorio de trabajo

Las vías de acceso se establecen para cada proyecto. Para establecerlas, utilice el cuadro de diálogo Propiedades de proyecto. Para más información consulte [“Definición de las propiedades del proyecto” en la página 2-23](#).

Al construir las vías de acceso, JBuilder elimina los nombres repetidos. Así se evitan los posibles problemas de las limitaciones de DOS en Windows.

Nota En estos ejemplos se utilizan las vías de acceso propias de la plataforma UNIX. Consulte [“Convenciones de la documentación” en la página 1-5](#).

Vía de acceso a archivos fuente

La vía de acceso a archivos fuente determina el lugar donde el compilador busca los archivos fuente. Se constituye a partir de lo siguiente:

- La vía de acceso definida en la ficha Fuente de la Ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto.
- El directorio de los archivos generados. Este directorio contiene archivos fuente generados automáticamente por el IDE. Ejemplos de estos archivos fuente son archivos del servidor IDL y archivos esqueleto. Este directorio

se encuentra en la vía de salida. Esta opción se puede modificar en la ficha Generar del cuadro de diálogo Propiedades de proyecto.

La vía de acceso a archivos fuente completa consta de estos dos elementos, por este orden:

vía de acceso a archivos fuente + vía de archivos generados/código fuente generado

En Proyecto de ejemplo, la vía de acceso a archivos fuente del proyecto africa.jpx es:

```
<raíz>/<nombredeusuario>/jbproject/Proyecto de ejemplo/src
```

Vía de salida

La vía de salida contiene los archivos .class creados por JBuilder y los archivos de recursos copiados por el compilador. Se constituye a partir de la vía definida en el cuadro de texto Vía de salida, situado en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto.

Los archivos se envían a un directorio cuya vía de acceso coincide con la vía de salida más el nombre del paquete. Solo existe una vía de salida en cada proyecto.

Por ejemplo, en el caso de Proyecto de ejemplo, la vía de salida de felino.africa.jpr es:

```
<raíz>/<nombredeusuario>/jbproject/Proyecto de ejemplo/classes
```

Vía de acceso a clases

La vía de acceso a clases se emplea en las compilaciones. Esta vía se construye a partir de lo siguiente:

- La vía de salida
- La vía de acceso a clases de todas las bibliotecas que figuran en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto. Es importante tener en cuenta que las bibliotecas se añaden a la vía de acceso a clases por el orden en que aparecen en la ficha Vías de acceso. Si hay vías de acceso repetidas en varias bibliotecas, tiene preferencia la superior.
- La versión de JDK seleccionada en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto

La vía de acceso a clases completa consta de estos elementos, por este orden:

vía de archivos generados + vía de acceso a clases de la biblioteca (por el orden en que aparecen las bibliotecas en el cuadro de diálogo Propiedades de proyecto) + versión JDK destino

Por ejemplo, la vía de acceso completa a `Leon.class` es:

```
/<raíz>/<nombredeusuario>/jbproject/Proyecto de ejemplo/classes:  
/user/jbuilder/lib/dbswing.jar:/
```

La vía de acceso a clases se muestra en el panel de mensajes cuando ejecuta el proyecto.

Vía de búsqueda

El IDE utiliza la vía de búsqueda en los siguientes casos:

- Cuando se utiliza CodeInsight.
- Cuando se selecciona Buscar definición en el menú desplegable del editor.
- Cuando se selecciona Buscar|Buscar clases
- Cuando se ejecuta el depurador

La vía de búsqueda se constituye a partir de lo siguiente:

- La vía de acceso a archivos fuente
- La vía de acceso a archivos fuente de las bibliotecas enumeradas en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto (en el orden por el que aparecen)
- La vía de acceso a archivos fuente de la versión del JDK seleccionada en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto

La vía de búsqueda completa consta de estos elementos, por este orden:

vía de acceso a archivos fuente + vías de acceso a fuente de las bibliotecas (en el orden por el que aparecen las bibliotecas en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto) + vía de acceso a fuente de la versión del JDK destino.

Por ejemplo, la vía de búsqueda completa en el caso de `Leon.class` es:

```
/<raíz>/<nombredeusuario>/jbproject/Proyecto de ejemplo/src:  
/user/jbuilder/src/dbswing-src.jar:  
/user/jbuilder/src/dx-src.jar
```

Vía de acceso a documentos

La vía de acceso a documentos contiene los archivos HTML de documentación de los archivos de clase de la API. Esto permite que la documentación de referencia se muestre en la ficha Doc del panel de contenido.

La vía de acceso a documentos se puede configurar en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto. Se busca en las vías de acceso por el orden en que aparecen en la lista.

Vía de acceso a las copias de seguridad

JBuilder almacena en la vía de acceso a copias de seguridad las versiones de respaldo de los archivos de código fuente. El directorio de copia de seguridad por defecto es:

```
/ <raíz>/ <nombredeusuario>/jbproject/Proyecto de ejemplo/bak
```

Directorio de trabajo

El directorio de trabajo es el directorio inicial que JBuilder proporciona a los programas cuando se inician. Se puede configurar cualquier directorio como directorio de trabajo. Por defecto, tiene el mismo nombre que el archivo del proyecto.

Normalmente es el directorio superior del directorio fuente. También es el directorio superior por defecto de los directorios de salida (archivos generados), copia de seguridad, documentación y bibliotecas.

Localización de los archivos

Todos los archivos de un proyecto se almacenan con una vía de acceso relativa a la ubicación del archivo .jpr. JBuilder busca y guarda los archivos en las vías de acceso a archivos fuente, de comprobación, de acceso a clases, de búsqueda y de archivos generados.

Esta lista explica el propósito de cada tipo de vía de acceso:

- Las vías de acceso de prueba y a archivos fuente determinan el lugar donde el compilador busca los archivos fuente.
- La vía de acceso a clases se utiliza en la compilación y durante la ejecución, y en determinadas funciones del editor de Enterprise.
- El IDE utiliza la vía de búsqueda cuando se emplea CodeInsight o el comando Buscar definición del editor, así como la búsqueda y la depuración.
- La vía de salida incluye los archivos .class creados por JBuilder durante la compilación del proyecto.

Consulte

- [“Cómo construye JBuilder las vías de acceso” en la página 4-9](#)
- [“Las bibliotecas” en la página 4-1](#)

Cómo encuentra JBuilder los archivos al profundizar

Cuando se profundiza para explorar el código fuente, JBuilder busca los archivos `.java` en la vía de búsqueda. Si desea obtener información adicional sobre la profundización, consulte "Desplazamiento por el código fuente" en *Introducción a JBuilder*.

Cómo encuentra JBuilder los archivos al compilar

Cuando se compila el proyecto, JBuilder utiliza las siguientes vías de acceso:

- vía de acceso a clases
- vía de acceso a archivos fuente
- vía de archivos generados

JBuilder busca en la vía de acceso a clases los archivos `.class`, las bibliotecas que ha de emplear y la versión del JDK en que debe compilar. El compilador compara los archivos `.class` de la vía de acceso a clases con sus archivos fuente, ubicados en la vía de acceso a archivos fuente, para determinar si es necesario volver a compilar los archivos `.class` para actualizarlos. Los archivos `.class` resultantes se almacenan en la vía de salida que se haya definido.

Para obtener más información acerca de la compilación de archivos, consulte el [Capítulo 6, "Generación de programas en Java"](#) y el [Capítulo 5, "Compilación de programas en Java"](#).

Cómo encuentra JBuilder los archivos de clase al ejecutar o depurar

Cuando se ejecuta y se depura el programa, JBuilder utiliza la vía de acceso a clases para localizar todas las clases que emplea el programa.

Cuando se inspecciona línea a línea el código fuente con el depurador, JBuilder utiliza la vía de búsqueda para localizar los archivos de código fuente.

Para obtener más información sobre la depuración de archivos, consulte el [Capítulo 8, "Depuración de programas en Java"](#).



Compilación de programas en Java

Los compiladores de Java leen los archivos de código fuente y generan el programa Java en forma de archivos `.class` que, a su vez, contienen los bytecodes que constituyen el código máquina para la máquina virtual (MV) Java. La compilación genera un archivo `.class` por cada declaración de clase o de interfaz de un archivo fuente. Cuando se ejecuta el programa de Java resultante en una plataforma concreta, como por ejemplo Windows NT, el intérprete de Java de dicha plataforma ejecuta los bytecodes de los archivos `.class`. Para obtener información general acerca de la compilación en Java, consulte los aspectos generales del compilador del Kit de desarrollo de Java (JDK), "**javac** - Compilador del lenguaje de programación Java".

El compilador por defecto para el IDE de JBuilder, Make de Borland para Java (**bmj**), admite el lenguaje Java sin limitaciones. El Make de Borland utiliza el compilador estándar **javac** junto con la comprobación inteligente de dependencias. El verificador de dependencias, que acelera y hace más eficiente el ciclo compilación/recompilación, determina la naturaleza de los cambios en el código fuente y sólo recompila los archivos necesarios. Si desea más información, consulte ["Comprobación inteligente de dependencias" en la página 5-2](#). Para poder entender mejor cómo funciona el compilador de JBuilder, consulte ["Make de Borland para Java \(bmj\)" en la página A-9](#).

Si prefiere realizar la compilación desde la línea de comandos, JBuilder también proporciona las siguientes herramientas de línea de comandos en las ediciones Developer y Enterprise de JBuilder:

- La opción de línea de comandos de JBuilder **-build** para generar proyectos
- Make de Borland para Java (**bmj**), que utiliza el verificador de dependencias
- El compilador de Borland para Java (**bcj**)

Las ediciones Developer y Enterprise de JBuilder también permiten el cambio de compilador. Para aprovechar todas las ventajas de las funciones de JBuilder, como la comprobación independiente de dependencias, UML y el perfeccionamiento, es recomendable que utilice el Make de Borland. No obstante, si desea utilizar únicamente **javac** o una versión anterior de Borland Make (JBuilder 8), puede cambiar de compilador en la ficha de Java del cuadro de diálogo Propiedades de proyecto (Proyecto|Propiedades de proyecto|Generar|Java). Si desea más información, consulte [“Definición de opciones del compilador” en la página 5-7](#).

La compilación es sólo una de las fases del sistema de generación de JBuilder. Entre otras se incluyen la precompilación, postcompilación, limpieza, empaquetado y distribución. Si desea obtener más información sobre estas fases y el sistema de generación de JBuilder, consulte el [Capítulo 6](#), [“Generación de programas en Java”](#).

Comprobación inteligente de dependencias

Borland Make realiza una compilación rápida pero completa, utilizando la comprobación inteligente de dependencias, que reduce el número de compilaciones innecesarias de los archivos fuente relacionados entre sí y acelera, de este modo, el ciclo de edición y recompilación. Durante la compilación, en lugar de decidir la recompilación de un archivo en función del momento en el que se produjo la modificación del archivo, el Make de Borland analiza la naturaleza de los cambios realizados en los archivos fuente.

Existen varias razones posibles para recompilar el archivo fuente:

- Faltan uno o más archivos de clase que produciría el archivo fuente.
- Se ha modificado el archivo fuente después de compilar.
- Una o más de las clases que produce el archivo fuente depende de un miembro de otra clase que se ha cambiado.

Al cambiar un archivo fuente, puede que cambie el modo en que se compilan otros archivos fuente. JBuilder no sólo es capaz de detectar esta situación, sino además, de advertir si un cambio en un archivo fuente no va a afectar a otros archivos, ya que se refieren a partes que no han sufrido cambios. Si es éste el caso, JBuilder **no** vuelve a compilar los archivos.

Cuando se compilan los archivos fuente por primera vez, se crea automáticamente para cada paquete un archivo de dependencias que se guarda en el directorio de salida, junto con los archivos de clases. Este

archivo de dependencias contiene información detallada, para todas las clases de un paquete, relacionada con la utilización que unas clases hacen de otras. Este archivo tiene la extensión `.dep2` y se guarda en una carpeta de nombre `package cache` en el mismo directorio que las clases.

Los archivos de dependencias deben encontrarse en la vía de acceso a clases, para que el compilador pueda encontrarlos. Cuando la compilación se realiza desde el IDE, la vía de acceso a clases se encuentra correctamente definida por defecto. Para obtener información sobre cómo se crea la vía de acceso a clases, consulte la sección “[Vía de acceso a clases](#)” en la [página 4-10](#).

Si se compila desde la línea de comandos puede ser necesario asignar un valor a la variable de entorno `CLASSPATH`. Si desea más información, consulte “[Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos](#)” en la [página A-4](#).

El IDE de Borland Make y el compilador `make` de línea de comandos **bmj** utilizan la comprobación inteligente de dependencias, a diferencia del compilador de línea de comandos **bcj**, que no lo utiliza. **bmj** y **bcj** se encuentran disponibles en las ediciones Developer y Enterprise de JBuilder. También se debe tener en cuenta que es posible que las generaciones Ant no utilicen la comprobación inteligente de dependencias, por lo que pueden ser distintas de las generaciones de JBuilder. Para obtener más información acerca de particiones, consulte “[Generación con archivos Ant](#)” en la [página 6-12](#).

Importante Las bibliotecas se consideran "estables" y el verificador de dependencias no las comprueba.

Consulte

- “JBuilder Dependency Checker” en la página web de Blake Stone, en <http://homepages.borland.com/bstone/articles/depchecker.html>

Compilación de un programa

El IDE de JBuilder por defecto utiliza Borland Make para Java (**bmj**), que utiliza **javac** como compilador, para compilar los archivos fuente de Java. Debido a que Borland Make utiliza la comprobación inteligente de dependencias, el ciclo de compilación y recompilación es más rápido y eficaz. Si desea más información, consulte “[Comprobación inteligente de dependencias](#)” en la [página 5-2](#). Para poder entender mejor cómo funciona el compilador de JBuilder, consulte “[Make de Borland para Java \(bmj\)](#)” en la [página A-9](#).

Es posible compilar las siguientes partes de un proyecto:

- La totalidad del proyecto
- Paquetes
- Archivos Java

Para comprender el modo en que JBuilder ubica los archivos para compilar el programa, consulte [“Cómo construye JBuilder las vías de acceso” en la página 4-9](#) y [“Localización de los archivos” en la página 4-12](#).

Para compilar los archivos fuente de un programa:

- 1 Abra el proyecto que contiene el programa o abra un archivo de Java.
- 2 Realice una de las operaciones siguientes:
 - Seleccione Proyecto|Ejecutar Make del proyecto.
 - Seleccione Proyecto|Ejecutar Make <nombre del archivo>.
 - Seleccione el botón Ejecutar Make del proyecto en la barra de herramientas, si está disponible.
 - Haga clic con el botón derecho del ratón en el nodo del proyecto .jpx del panel del proyecto y seleccione Ejecutar Make.
 - Haga clic con el botón derecho del ratón en uno o varios nodos del panel del proyecto que se puedan crear, y, a continuación, seleccione Ejecutar Make.
 - Pulse con el botón derecho del ratón en la pestaña del archivo del panel de contenido y, a continuación, elija Ejecutar Make < nombredelarchivo>.



Menús de generación de JBuilder

JBuilder ofrece comandos de menú para generar proyectos: Ejecutar Make, Generar de nuevo y Limpiar. Ejecutar Make es una fase del sistema de generación de JBuilder que establece dependencias entre otras fases autónomas: Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir. El comando Ejecutar Make de JBuilder no debe confundirse con el make de compilación de Java, que sólo compila los archivos fuente de Java. El comando Ejecutar Make genera archivos fuente Java y otros archivos generables en su proyecto, como archivos recopilatorios, módulos web y otros nodos generables. JBuilder también proporciona el comando Generar de nuevo, que tiene como dependencias limpiar y ejecutar Make, para volver a generar completamente el proyecto. Al elegir el comando Generar de nuevo, se elimina toda la salida de la generación y, a continuación, se ejecuta un make. Por último, el comando Limpiar elimina toda la salida generada.

Para su comodidad, JBuilder permite configurar el menú Proyecto para proyectos individuales, así como para grupos de proyectos. Se pueden añadir tipos y tareas de generación adicionales si el proyecto las contiene. Si desea más información, consulte [“Configuración del menú Proyecto” en la página 6-34](#).

Consulte

- [“El comando Ejecutar Make comandos” en la página 6-4](#)
- [“El comando Generar de nuevo” en la página 6-5](#)
- [“El comando Limpiar” en la página 6-5](#)

Generación de proyectos con el comando Ejecutar



El comando Ejecutar se puede configurar para que ejecute un tipo de generación antes de ejecutar el proyecto. El funcionamiento por defecto del comando Ejecutar el proyecto (Ejecutar|Ejecutar el proyecto) y del botón Ejecutar el proyecto es ejecutar el Make de la aplicación y, después, ejecutarla. El funcionamiento por defecto se puede cambiar en las configuraciones de ejecución del proyecto. Por ejemplo, puede generar de nuevo su proyecto cada vez que vaya a ejecutarlo, en lugar de utilizar el Make por defecto. Los tipos de generación disponibles varían según el tipo de proyecto en el que esté trabajando. Entre los tipos disponibles pueden estar Generar de nuevo, Limpiar, Ninguno, tareas externas de generación, tipos Ant y cualquier tarea de generación personalizada que haya agregado al ampliar el sistema de generación por medio de Open Tools. Si desea más información sobre cómo modificar el tipo de generación, consulte [“Selección de tipos de generación” en la página 7-12](#).

Consulte

- [“Generación de proyectos Ant con el comando Ejecutar” en la página 6-22](#)

Mensajes de error

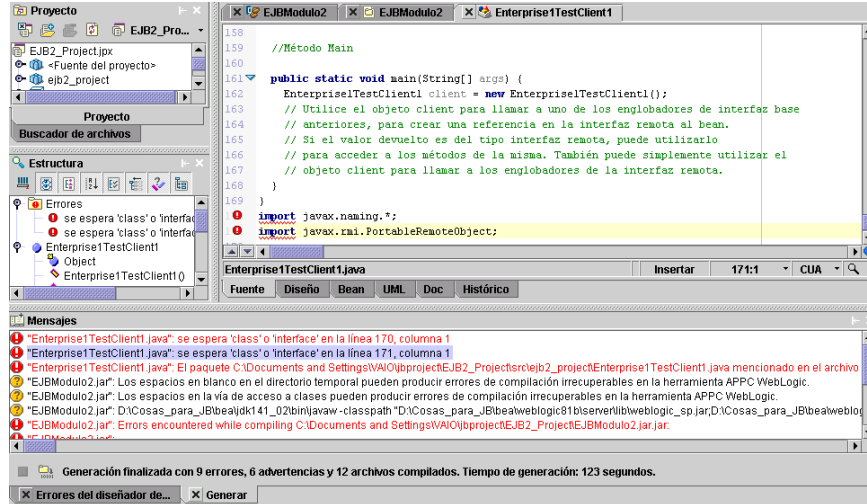


Se muestran mensajes de error con todos los errores que transgredan las reglas sintácticas del lenguaje de programación Java. El compilador detecta estos errores dinámicamente, a medida que se presentan en el editor, por lo que es posible corregirlos antes de realizar la compilación. Una función del editor, ErrorInsight, le ayuda a acceder y resolver rápidamente la mayoría de los errores de código. Mientras escribe su código fuente, los errores se subrayan en rojo en el editor, y también aparecen de forma dinámica en el panel de estructura en una carpeta Errores. Los iconos de ErrorInsight muestran los errores que se pueden corregir con la herramienta ErrorInsight, y aparecen junto al error de codificación en el editor y en el panel de estructura. Los iconos de errores de código representan los errores que no se pueden corregir con ErrorInsight. Si desea obtener más información, consulte "ErrorInsight" en *Introducción a JBuilder*.

Para encontrar un error, haga clic en el mensaje de error, en la carpeta Errores del panel de estructura, y la línea de código correspondiente se resaltará en el editor. Haga doble clic en el error para centrarse en la línea de código del editor.

Los mensajes de error también se muestran en la pestaña Generar del panel de mensajes tras la compilación. Si desea ver la descripción de un error, seleccione el mensaje de error y pulse *F1*. Las teclas de desplazamiento del panel de mensajes permiten moverse por los mensajes de error del compilador. Haga clic en uno de ellos para resaltar el código del archivo abierto. Haga doble clic en el error para desplazar el cursor a la línea de código, en el editor.

Cuando examine los errores, recuerde que su origen puede encontrarse fuera de la línea que indica el mensaje. Si no lo ve inmediatamente, examine las líneas anteriores.



Consulte

- “Mensajes de advertencia y error” en la ayuda en línea
- “Mensajes de error del compilador” en la ayuda en línea

Problemas de compilación al abrir proyectos

Si abre un proyecto y no se compila, compruebe que está bien definida la vía de acceso en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto. JBuilder utiliza los valores de la vía de acceso para construir la vía de acceso a clases y la vía de acceso a archivos fuente, lugares en los que el compilador busca los archivos.

Además, compruebe la lista de Bibliotecas necesarias de la ficha Vías de acceso. Si alguna de las bibliotecas está resaltada en rojo, significa que no se ha definido al instalar JBuilder. Haga doble clic sobre el nombre de la biblioteca o selecciónelo y, a continuación, seleccione Modificar para redefinirlo. Por último, recompile el proyecto.

Los proyectos con módulos web pueden necesitar un servidor web correctamente configurado para poder compilar. Si desea obtener más información, consulte "Configuración del servidor de aplicaciones de destino" en *Desarrollo de aplicaciones para servidores J2EE*.

Para definir las vías de acceso por defecto de proyectos nuevos (a fin de evitar posibles problemas en el futuro), acceda al cuadro de diálogo de Propiedades por defecto para proyectos (Proyecto|Propiedades por defecto para proyectos). Si desea más información, consulte [Definición de las](#)

[propiedades del proyecto](#) en la página 2-23 y [“Cómo construye JBuilder las vías de acceso”](#) en la página 4-9.

Comprobación de correspondencia entre paquetes y directorios

JBuilder cuenta con una comprobación de protección de definiciones duplicadas de clases dentro de un proyecto y de correspondencia entre paquetes y directorios. El compilador **bmj** de Borland Make, compilador por defecto del IDE, comprueba que la sentencia `package` de un archivo fuente se corresponda con el directorio del paquete y que una misma clase no esté definida por dos archivos fuente.

La primera vez que se construye un proyecto, se verifican y compilan todos los archivos `.java` que se encuentran dentro de un directorio de paquete. Si tiene fuentes temporales que no desea compilar, utilice una extensión que no sea `.java`. Por ejemplo, si el proyecto contiene una versión anterior de un archivo en el que se está trabajando y que contiene una definición diferente de la misma clase, aparecerá un mensaje de "definición duplicada de una clase". Esta comprobación impide la aparición de problemas sutiles que serían difíciles de encontrar.

Definición de opciones del compilador

Puede definir las opciones del compilador del proyecto actual en la ficha Java de Propiedades de proyecto (Proyecto|Propiedades de proyecto|Generar|Java). Las opciones del compilador varían según el que se haya seleccionado. El compilador por defecto es Borland Make. En las ediciones Developer y Enterprise de JBuilder también hay compiladores adicionales disponibles.

Las opciones del compilador se aplican a todos los archivos del árbol del proyecto. Si cambia las opciones del compilador, debe generar de nuevo los paquetes o todo el proyecto en lugar de utilizar solamente Ejecutar Make. Las opciones del proyecto se aplican a todas las clases que vuelvan a generarse, tanto fuera como dentro del árbol del proyecto.

Si compila su proyecto con el Make de Borland, las opciones del compilador se aplican a todos los archivos del árbol del proyecto y a todos los archivos a los que se hace referencia en estos archivos, y se detiene en los paquetes marcados como estables y que no tengan clases en el árbol del proyecto. Make de Borland ofrece opciones adicionales del compilador, como Confundir, Sincronizar directorio de salida y Excluir la clase. Si desea obtener más información acerca de estas opciones, pulse el botón Ayuda de la ficha Java de Propiedades de proyecto.

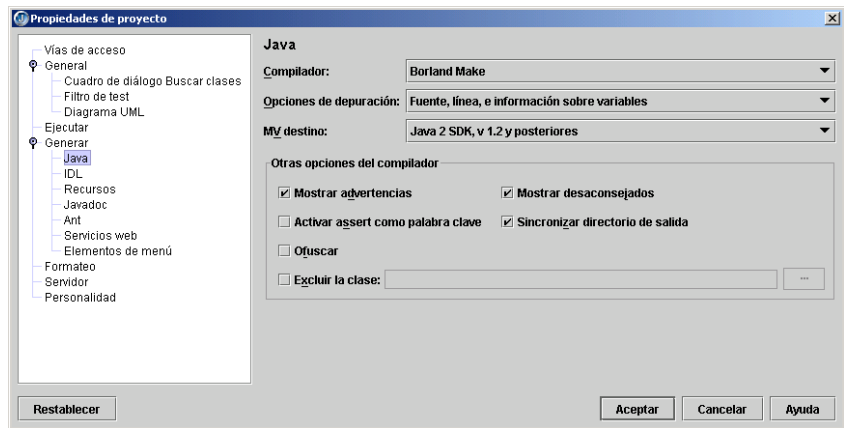
No se pueden definir las opciones de compilación para cada archivo; no obstante, un archivo se puede utilizar en dos proyectos, cada uno de ellos con diferentes opciones de compilación. No se pueden activar opciones de forma

individual a clases y paquetes, dado que en Java las cabeceras y módulos no se compilan de forma independiente. Si falta información de importación (por ejemplo acerca de un archivo de clase), la clase importada se compila al mismo tiempo que la clase que la importa, utilizando las mismas opciones aplicadas a todo el proyecto.

También se pueden configurar las opciones del compilador para proyectos futuros en el cuadro de diálogo Propiedades de proyecto por defecto (Proyecto|Propiedades de proyecto por defecto). Después de configurar las propiedades por defecto para proyectos, siempre que cree un proyecto con el Asistente para proyectos, se aplican las opciones por defecto.

Para definir las opciones del compilador para su proyecto, siga los pasos siguientes:

- 1 Seleccione Proyecto|Propiedades de proyecto|Generar|Java.



- 2 Seleccione un compilador y las opciones de depuración y compilación que desee. Las opciones del compilador disponibles varían según el compilador seleccionado. El cambio de compiladores es una función de JBuilder Developer y Enterprise. Si desea más información sobre las opciones disponibles, pulse el botón Ayuda situado en la ficha Java (Proyecto|Propiedades de proyecto|Generar|Java).
- 3 Configure las opciones de generación que desee en las demás fichas del cuadro de diálogo Propiedades de proyecto.
- 4 Seleccione Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto y guardar la configuración.
- 5 Seleccione Proyecto|Generar de nuevo el proyecto para generar el proyecto con las opciones revisadas.
- 6 Haga clic en Aceptar para cerrar el cuadro de diálogo.

Definición de un compilador

Es una función de JBuilder Developer y Enterprise.

Por defecto, JBuilder compila los proyectos con Borland Make para Java (**bmj**), que utiliza el compilador estándar **javac** junto con la comprobación inteligente de dependencias. Normalmente, se recomienda que realice la compilación con Borland Make para aprovechar toda la funcionalidad de JBuilder como, por ejemplo, la comprobación de dependencias y el perfeccionamiento. Tenga en cuenta que si el código fuente antiguo no se compila con el Make de Borland, puede utilizar el Make de Borland (JBuilder 8) para generar el proyecto. Éste no utiliza **javac**, sino una versión anterior del compilador de JBuilder. Otros compiladores disponibles son Borland Make (JBuilder 8), **javac** y Project **javac**.

Importante

Si no se utiliza Borland Make o Borland Make (JBuilder 8), los archivos que se deben compilar debido a los cambios en las dependencias no lo harán, a menos que en primer lugar se limpie o se vuelva a generar el proyecto.

Tabla 5.1 Compiladores disponibles

Compilador	Descripción	Edición de JBuilder
Make de Borland	Usa el javac estándar y la comprobación inteligente de dependencias.	Todas
Borland Make (JBuilder 8)	Utiliza Borland Make de JBuilder 8, que emplea la comprobación inteligente de dependencias pero no javac . Si el código fuente antiguo no se compila con el Make de Borland, utilice el Make de Borland (JBuilder 8) para generar el proyecto. Éste no utiliza javac , sino una versión anterior del compilador de JBuilder.	Developer Enterprise
Project javac	Usa el javac del JDK indicado en la ficha Vías de acceso de Propiedades de proyecto (Proyecto Propiedades de proyecto Vías de acceso).	Developer Enterprise
javac	Utiliza javac del JDK que se encuentra en el directorio de JBuilder.	Developer Enterprise

Para cambiar el compilador del proyecto, siga los pasos siguientes:

- 1 Seleccione Proyecto|Propiedades de proyecto|Generar|Java.
- 2 Seleccione un compilador de la lista desplegable Compilador.
- 3 Haga clic en Aceptar para cerrar el cuadro de diálogo.

Configuración de las opciones adicionales de compilación y generación

Además, hay opciones adicionales en la ficha Generar y en las subfichas de Generar del cuadro de diálogo Propiedades de proyecto (Proyecto|Propiedades de proyecto) que afectan a la generación. Por ejemplo, puede especificar un compilador de IDL, los recursos que se copian en la vía de

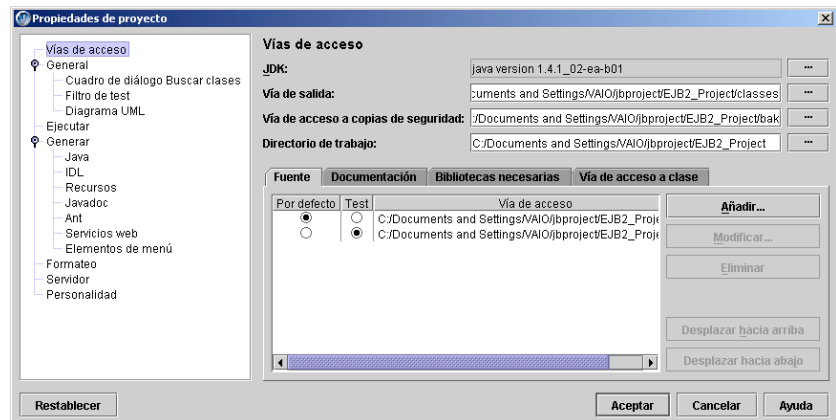
salida, un SQLj translator, bibliotecas Ant, etc. Para obtener más información sobre las demás opciones, pulse Ayuda en cualquiera de las fichas.

Configuración de la vía de salida

En el cuadro de diálogo Propiedades de proyecto se puede determinar la vía de salida de los archivos de clase compilados.

Para configurar la vía de salida:

- 1 Seleccione Proyecto|Propiedades de proyecto|Vías de acceso para abrir la ficha Vías de acceso.



- 2 Pulse el botón puntos suspensivos (...), situado a la derecha del campo Vía de salida.
- 3 Busque el directorio en el que desea guardar los archivos de clase compilados y selecciónelo. Si el directorio no existe, seleccione el botón Nueva Carpeta para crearlo. Pulse Aceptar.
- 4 Haga clic en Aceptar para cerrar el cuadro de diálogo.

Consulte

- “Cómo construye JBuilder las vías de acceso” en la página 4-9
- “Localización de los archivos” en la página 4-12

Compilación de proyectos en un grupo de proyectos

Es una función de
JBuilder Developer y
Enterprise.

Los grupos de proyectos son contenedores de proyectos, y pueden resultar de gran utilidad cuando se trabaja con proyectos relacionados. Los grupos de proyectos permiten controlar el orden de generación de los proyectos dentro del grupo. Esto resulta de gran utilidad si un proyecto depende de otro. Si desea obtener más información sobre la compilación de proyectos en un grupo de proyectos, consulte el [Capítulo 6, “Generación de programas en Java”](#). Para obtener información general sobre grupos de proyecto, consulte el [Capítulo 3, “Los grupos de proyectos”](#).

Compilación desde la línea de comandos

Estas funciones
pertenecen a las
ediciones Developer y
Enterprise de
JBuilder.

Puede compilar desde la línea de comandos si utiliza los comandos `bmj` o `bcj`. Para ver la sintaxis y una lista de las opciones, escriba `bmj` o `bcj` en la línea de comandos del directorio `<jbuilder>/bin`. También es posible generar proyectos desde la línea de comandos de JBuilder. Es posible que necesite definir la variable de entorno `CLASSPATH` de la línea de comandos, de forma que las clases necesarias estén disponibles.

Además, se pueden generar proyectos desde la línea de comandos mediante la opción de la interfaz de línea de comandos **-build** de JBuilder. Consulte [“Creación de proyectos desde la línea de comandos” en la página 5-12](#).

bmj (Make de Borland para Java)

Es una función de
JBuilder Developer y
Enterprise.

El compilador **bmj** es Borland Make para Java, que utiliza el compilador estándar **javac** en combinación con la comprobación inteligente de dependencias. **bmj** compila todos los archivos `.java` que tienen archivos `.class` no actualizados o inexistentes. **bmj** compila también todas las clases importadas que tienen archivos `.class` no actualizados o inexistentes.

bmj busca archivos de dependencias en la vía de acceso a clases y las comprueba. Si se especifica un conjunto de archivos fuente, es posible que no se recompilen todos ellos. Por ejemplo, podría determinarse que los archivos de clase están actualizados si se han guardado pero no se han editado desde la última compilación. Es posible forzar la recompilación por medio de la opción **-rebuild**.

Para comprobar un conjunto (o "gráfico") de módulos interdependientes, basta con llamar a **bmj** en el archivo fuente raíz (o en varios archivos fuente raíz, si uno no está debajo de otro). Puede definir este argumento utilizando nombres de fuente, nombres de paquetes, nombres de clases o una combinación de ellos.

Nota Si desea utilizar la versión anterior de **bmj** de JBuilder 8, utilice el comando `oldbmj`.

Consulte

- [“Make de Borland para Java \(bmj\)” en la página A-9](#)
- [“Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos” en la página A-4](#)

bcj (Compilador de Borland para Java)

Es una función de JBuilder Developer y Enterprise.

El compilador **bcj** es Borland Compiler para Java. En el caso de JBuilder 9, **bcj** es un englobador de **javac** que permite compatibilidad con versiones anteriores. **bcj** compila las fuentes especificadas, aunque los archivos `.class` se hayan o no actualizado. Compila también todos los archivos `.java` importados directamente que no tengan archivos `.class`. Los archivos `.java` importados que ya tengan archivos `.class` no se vuelven a compilar, incluso si los archivos `.class` no están actualizados. Después de utilizar **bcj**, algunas clases importadas pueden, todavía, tener archivos `.class` sin actualizar.

bcj no comprueba las dependencias, y no utiliza ni genera un archivo de dependencia. **bcj** se limita a compilar los elementos especificados.

Nota Si desea utilizar la versión anterior de **bcj** de JBuilder 8, utilice el comando `oldbcj`.

Consulte

- [“El compilador de Borland para Java \(bcj\)” en la página A-16](#)
- [“Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos” en la página A-4](#)

Creación de proyectos desde la línea de comandos

Esta es una opción de las ediciones Developer y Enterprise de JBuilder.

Es posible generar archivos de proyecto y especificar tipos de generación desde la línea de comandos de JBuilder por medio de la opción **-build** del directorio `<jbuilder>/bin`. Si desea más información, consulte [“Interfaz de la línea de comandos de JBuilder” en la página A-6](#).

Cambio entre la línea de comandos y el IDE

Si edita un archivo fuera del IDE, asegúrese de incluir en el proyecto el paquete correspondiente, o al menos uno de sus archivos fuente, para que el paquete se compruebe durante la compilación. Si no lo hace así, el cambio no se detecta y el paquete fuente no se recompila.



Generación de programas en Java

Las funciones de generación disponibles varían según la edición de JBuilder.

El sistema de generación de JBuilder, basado en la herramienta de generación en Java, Ant, incluye varias fases de generación. Las fases de generación, que son objetivos especiales que siempre crea el sistema de generación para cada proceso de generación, pueden incluir tareas de generación tales como la preparación de archivos ajenos a Java para su compilación, la compilación de archivos Java de código fuente, la recopilación, etc. El sistema de generación se puede personalizar y ampliar con la clase del `Creador` de `OpenTool`.

El compilador de JBuilder, `Make` de Borland para Java (**bmj**), admite el lenguaje Java sin limitaciones, incluidas las clases internas y los archivos JAR. Debido a que el compilador de JBuilder utiliza la comprobación inteligente de dependencias, el ciclo de compilación y recompilación es más rápido y eficaz. El verificador de dependencias determina la naturaleza de los cambios y sólo compila los archivos necesarios.

Consulte

- “Comprobación inteligente de dependencias” en la página 5-2
- Capítulo 5, “Compilación de programas en Java”
- “Make de Borland para Java (bmj)” en la página A-9

El sistema de generación de JBuilder

El sistema de generación de JBuilder utiliza Ant, una herramienta de generación de código abierto basada en Java para ejecutar generaciones en oposición a la utilización de archivos de generación Ant estáticos. El sistema de generación, ampliable también como una OpenTool, permite realizar las siguientes operaciones:

- Generar grupos de proyectos, una función de las ediciones Developer y Enterprise de JBuilder.
- Generar proyectos Ant existentes en JBuilder.
- Exportar proyectos de JBuilder a Ant, una función de JBuilder Developer y Enterprise.
- Filtrar paquetes y eliminarlos del proceso de generación, una función de las ediciones Developer y Enterprise de JBuilder.
- Ampliar el sistema de generación con un OpenTool.

Consulte

- [“Generación de grupos de proyectos” en la página 6-9](#)
- [“Generación con archivos Ant” en la página 6-12](#)
- [“Exportación de proyectos de JBuilder a Ant” en la página 6-25](#)
- [“Filtrado de paquetes” en la página 6-39](#)
- "JBuilder Build System Concepts" en la ayuda en línea de OpenTools

Términos del sistema de generación

Para hacer referencia al sistema de generación, se utilizan los siguientes términos.

Tabla 6.1 Términos del sistema de generación

PLAZO	Definición
Tarea de generación	Parte del código que se puede ejecutar durante la generación, como, por ejemplo, la compilación en java, FTP, la generación de un archivo JAR, etc.
Destino	Colección de tareas de generación que se van a ejecutar. Los tipos pueden depender de otros tipos. Por ejemplo, si el tipo A depende de los tipos B y C, se ejecutarán B y C antes de que se ejecute A.
Fase	Tipos especiales que crea el sistema de generación de JBuilder para cada generación. Existen ocho fases: seis fases autónomas sin dependencias (Limpiar, Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir) y dos fases que establecen dependencias entre fases (Ejecutar Make y Generar de nuevo). Cada fase cuenta con sus tipos específicos.

Fases de generación

Las fases de generación en el IDE de JBuilder incluyen seis fases autónomas sin dependencias y dos fases que establecen dependencias entre otras fases. Cada uno de los proyectos de JBuilder incluyen las siguientes fases autónomas: Limpiar, Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir. Ya que son fases autónomas, una fase se puede ejecutar sin necesidad de ejecutar otra. Cada fase cuenta con sus propios tipos como dependencias. Por ejemplo, SQLJ es una dependencia de la fase de precompilación.

Existen dos fases adicionales que establecen dependencias entre las seis fases autónomas: Ejecutar Make y Generar de nuevo. Ejecutar Make cuenta con las siguientes dependencias: Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir. Generar de nuevo tiene como dependencias Limpiar y Ejecutar Make.

Ya que el sistema de generación de JBuilder se muestra como una OpenTool puede crear sus propias tareas de generación y especificar las fases ya existentes como dependencias o, bien, no ateniéndose en absoluto a ellas. Consulte "Conceptos básicos sobre el sistema de generación de JBuilder" en *Desarrollo de Open Tools* para obtener más información sobre cómo ampliar el sistema de generación. Si desea un ejemplo para hacer `Creadores`, consulte el ejemplo del ofuscador en el directorio `samples/opentoolsAPI/Build` de JBuilder.

Tabla 6.2 Etapas del sistema de generación independiente




PLAZO	Icono	Definición
Limpiar		Elimina todas las salidas generadas, como los archivos <code>.class</code> y los archivos JAR.
Precompilar		Tareas que tienen lugar antes de la compilación. Los archivos IDL, que se convierten en archivos fuente Java antes de la compilación, son ejemplos de un tipo de la precompilación.
Compilar		Generación de archivos de clase Java a partir de archivos fuente Java.
Postcompilar		Tareas que tienen lugar después de la compilación. Esta etapa requiere la creación de archivos de clase Java. Por ejemplo, <code>java2iio</code> y el código ofuscado pueden ser destinos de esta etapa.
Paquete		Tareas que generan archivos recopilatorios.
Distribuir		Tareas que mueven a una ubicación distinta los archivos ya distribuidos. Por ejemplo, esta fase debe tener una tarea para los archivos FTP.

Tabla 6.3 Etapas del sistema de generación que establecen dependencias

PLAZO	Icono	Definición
Ejecutar Make		Ejecutar Make establece dependencias entre las fases autónomas en el orden siguiente: Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir.
Generar de nuevo		Generar de nuevo tiene como dependencias Limpiar y Ejecutar Make. Para volver a generar sin el filtrado de paquetes, consulte “Generar de nuevo sin filtros” en la página 6-41 .

El comando Ejecutar Make comandos

Ejecutar Make es una fase que establece dependencias entre las fases autónomas. Ejecutar Make cuenta con las siguientes dependencias listadas por orden: Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir.

El comando Ejecutar Make no debe confundirse con el make de compilación de Java, que sólo compila los archivos fuente de Java. Si desea más información sobre el compilador de JBuilder, consulte [“bmj \(Make de Borland para Java\)” en la página 5-11](#) y [“Comprobación inteligente de dependencias” en la página 5-2](#).

Ejecutar Make ejecuta varias tareas de ejecución, dependiendo de los nodos seleccionados. Los nodos seleccionados pueden ser un proyecto, paquetes, un archivo fuente de Java u otro tipo de nodo, como nodos de recopilatorios, de documentación, de aplicaciones web, de tareas externas de generación o de tipos Ant. Por ejemplo, si ejecuta Make en un nodo de recopilatorio, se genera un archivo de recopilatorios. Si ejecuta Make en un paquete, se compilan los archivos fuente de Java y se copian los recursos de estos paquetes en la vía de salida del proyecto. Al ejecutar Make en un proyecto, se compilan los archivos fuente de Java del proyecto y se ejecutan las tareas de generación adecuadas en cualquier nodo que se pueda generar.

Existen varias formas de Ejecutar Make en un archivo, proyecto, paquete u otro nodo apropiado:

- Seleccione Proyecto|Ejecutar Make del proyecto.
- Seleccione Proyecto|Ejecutar Make <nombre del archivo>.
- Seleccione el botón Ejecutar Make del proyecto en la barra de herramientas, si está disponible.
- Haga clic con el botón derecho del ratón en el nodo del proyecto .jpx del panel del proyecto y seleccione Ejecutar Make.
- Haga clic con el botón derecho del ratón en uno o varios nodos del panel del proyecto que se puedan crear y, a continuación, seleccione Ejecutar Make.
- Pulse con el botón derecho del ratón en la pestaña del archivo del panel de contenido y, a continuación, elija Ejecutar Make <nombre del archivo>.

Asimismo, en las ediciones Developer y Enterprise de JBuilder se puede crear un grupo de proyectos. Para obtener más información sobre grupos de proyectos, consulte el [Capítulo 3, “Los grupos de proyectos”](#).

El comando Generar de nuevo

Generar de nuevo es otra fase que establece dependencias entre fases autónomas. Su dependencias son Limpiar y Ejecutar Make. Generar de nuevo borra todas las salidas de generación con Limpiar, para después ejecutar Make. El nodo seleccionado puede ser cualquier cosa que se pueda generar que admita Limpiar. Algunos ejemplos incluyen proyectos, paquetes, archivos fuente de Java, recopilatorios y recursos.

Debido a que Generar de nuevo ejecuta Limpiar y, posteriormente, Ejecutar Make, necesita más tiempo que Ejecutar Make. Pero puede resultar de gran utilidad si desea una generación limpia. Por ejemplo, si ha borrado los archivos fuente de Java, tendría que utilizar Generar de nuevo. Generar de nuevo ejecuta Limpiar, que elimina todas las salidas de generación, incluidos los archivos de clase. Después, se ejecuta Ejecutar Make. Si ejecuta Make después de borrar los archivos fuente de Java, se conservarían los archivos de clase.

Importante Si cambia cualquier opción de depuración o de confusión en la ficha Generar de Propiedades de proyecto, deberá generar de nuevo el proyecto para que se activen esos cambios.

Existen varias formas de generar de nuevo en un archivo, proyecto, paquete u otro nodo apropiado:

- Seleccione Proyecto|Generar el proyecto.
- Seleccione Proyecto|Generar de nuevo <nombre del archivo>.
- Pulse con el botón derecho del ratón en el panel del proyecto y seleccione Generar de nuevo.
- Pulse con el botón derecho del ratón en la pestaña del archivo en panel de contenido y elija Generar de nuevo <nombredelarchivo>.
- Abra la lista desplegable situada junto al botón Ejecutar Make de la barra de herramientas y seleccione la opción Generar de nuevo el proyecto, si está disponible.

Asimismo, en las ediciones Developer y Enterprise de JBuilder se puede volver a crear un grupo de proyectos. Para obtener más información sobre grupos de proyectos, consulte el [Capítulo 6, “Generación de programas en Java”](#).

El comando Limpiar

El comando Limpiar elimina todos los archivos de salida de los otros tipos, como el directorio `classes`, los archivos JAR, WAR, etc. Si se utiliza una única vía como vía de acceso y de salida, el directorio de salida no se borra, pero sí se borra la salida de generación. Lo que hace el comando Limpiar es eliminar todos los elementos que dependen del nodo seleccionado:

- Nodos del proyecto: elimina de forma recursiva el directorio de salida. Esto solamente tiene lugar si el directorio de salida es un subdirectorio del proyecto. El comando Limpiar no elimina el directorio de salida si coincide con el directorio fuente o si es un subdirectorio de éste.

- **Nodos Java:** elimina los archivos `.class` correspondientes y todos los archivos creados, como los `java2iio.p`. También elimina recursos.
- **Nodos de paquete:** borra los archivos `.class` correspondiente y cualquier recurso.
- **Nodos de recursos:** borra las copias en el directorio de salida.
- **Nodos de documentación:** borra todos los archivos HTML y HTM del directorio de salida de Javadoc.
- **Nodos de recopilatorios:** borra los archivos recopilatorios y los ejecutables.
- **Nodos módulo web:** borra los archivos WAR y los directorios `WEB-INF/lib` y `WEB-INF/classes`.

Existen varios modos de ejecutar el comando Limpiar:

- Pulse con el botón derecho del ratón en el archivo del proyecto del panel del proyecto y, a continuación, seleccione Limpiar.
- Pulse con el botón derecho del ratón sobre los nodos adecuados del panel del proyecto y seleccione Limpiar.
- Añada el comando Limpiar proyecto al menú Proyecto y seleccione Proyecto|Limpiar proyecto. Consulte [“Configuración del menú Proyecto” en la página 6-34](#).

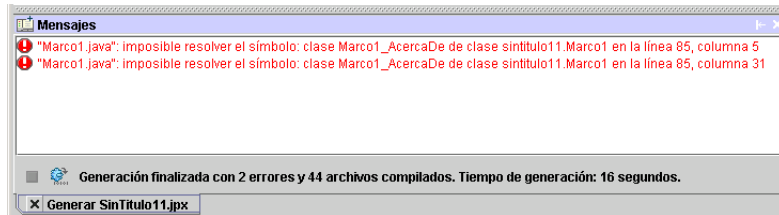
Al igual que los comandos Ejecutar Make y Generar de nuevo, el comando Limpiar solamente aparece en el menú contextual cuando se seleccionan los nodos adecuados.

Utilización del panel de mensajes

Al ejecutar una fase de generación en JBuilder, el comportamiento por defecto es mostrar el panel de mensajes y bloquear todas las demás actividades durante la generación. No obstante, también puede configurar la generación para que se ejecute en segundo plano, y así poder seguir trabajando. Si desea más información, consulte [“Generación en segundo plano” en la página 6-7](#).

Si se lleva a cabo la generación correctamente, el panel de mensajes se cierra automáticamente. Puede cambiar este comportamiento si desactiva la opción Cerrar la pestaña después de generar con éxito en el cuadro de diálogo Preferencias (Herramientas|Preferencias|Visualizador|Generar). Si desea más información, consulte [“Configuración de las preferencias de generación” en la página 6-8](#). La pestaña Generar permanece abierta si hay algún resultado de generación del que informar o si se producen errores durante la generación. Seleccione un mensaje de error del panel para resaltarlo en el archivo en el editor. Haga doble clic en un error para mover el foco al editor. Para obtener más información acerca de mensajes de error, consulte [“Mensajes de error” en la página 5-5](#). Una vez que haya corregido los errores, puede seleccionar el botón Generar de la barra de herramientas del panel de mensajes para ejecutar de nuevo la misma fase de generación.

Si la generación se realiza correctamente, el panel de mensajes se cierra automáticamente.



Generación en segundo plano

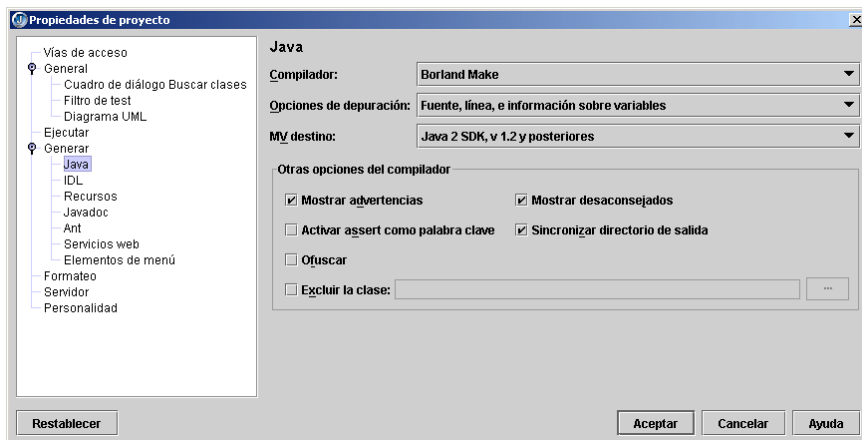
Durante extensas generaciones de proyectos, el cuadro de diálogo Avance de la generación está visible durante el tiempo que le queda al proceso de generación. Para continuar trabajando en JBuilder mientras se ejecuta la generación, seleccione el botón Segundo plano para enviar la generación a un segundo plano. El avance de la generación aparece en la barra de estado del panel de mensajes. Para detener la generación mientras se está ejecutando en segundo plano, pulse el botón Detener de la barra de herramientas del panel de mensajes. Si prefiere ejecutar todas las generaciones del proyecto en segundo plano, puede configurar la opción global Generar en segundo plano en el cuadro de diálogo Preferencias (Herramientas|Preferencias|Visualizador|Generar). Si desea más información, consulte [“Configuración de las preferencias de generación” en la página 6-8](#).

Si desea obtener más información acerca de las funciones del panel de mensajes, consulte "Panel de mensajes" en *Introducción a JBuilder*.

Asignación de valores a propiedades de generación

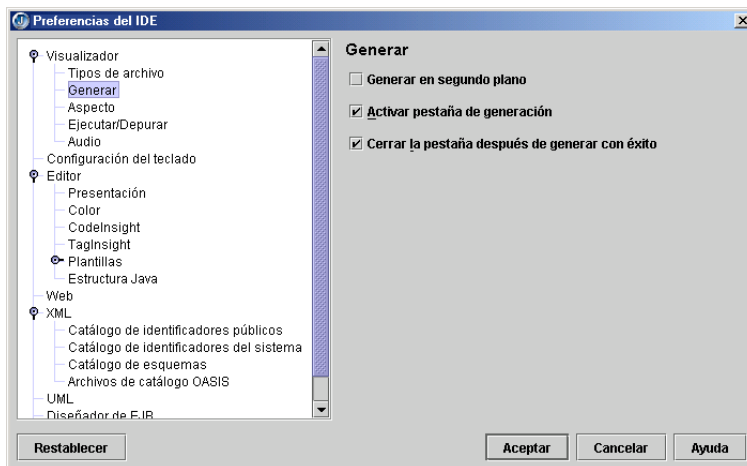
Para asignar valores a propiedades de generación para su proyecto, seleccione Proyecto|Propiedades de proyecto, o bien, haga clic con el botón derecho en el nodo del proyecto del panel del proyecto y seleccione Propiedades. Escoja el nodo Generar o cualquiera de sus nodos dependientes en el árbol del cuadro de diálogo Propiedades de proyecto. Las opciones de generación incluyen opciones generales de generación, Java, IDL, recursos, Javadoc, Ant, servicios web y elementos de menú. Para asignar valores a propiedades de generación para todos los proyectos futuros, seleccione Proyecto|Propiedades por defecto para proyectos.

Seleccione el botón Ayuda de cualquiera de estas fichas para obtener más información sobre estas opciones.



Configuración de las preferencias de generación

Existen varias opciones de generación que se pueden configurar para el IDE de JBuilder que facilitan la generación de los proyectos. Estas opciones se encuentran en la ficha Generar del cuadro de diálogo Preferencias (Herramientas|Preferencias|Visualizador|Generar).



Entre las opciones de generación están:

- Generar en segundo plano inicia el proceso de generación en segundo plano.
- Activar pestaña de generación hace que la pestaña Generar sea la activa cuando se muestran varias pestañas del panel de mensajes. Por ejemplo, si hay abiertas otras pestañas, como Buscar, la pestaña activa es Generar.

Si esta opción se encuentra desactivada, se activa la última pestaña del panel de mensajes que estuviera activa.

- Cerrar la pestaña después de generar con éxito la pestaña Generar del panel de mensajes se cierra automáticamente si la generación es correcta.

Si la opción Generar en segundo plano se encuentra activada, la generación tiene lugar en segundo plano, con lo que es posible seguir trabajando. Puede detener la generación en cualquier momento mediante el botón Detener de la barra de herramientas del panel de mensajes.

Si la opción Generar en segundo plano se encuentra desactivada, el cuadro de diálogo Avance de la generación se presenta, y se bloquea toda la actividad hasta que concluye el proceso de generación. Sin embargo, este cuadro de diálogo cuenta con el botón Segundo plano, que permite continuar con la generación en segundo plano y seguir trabajando con JBuilder. Seleccione el botón Ayuda para obtener más información acerca de estas opciones.

Generación de grupos de proyectos

**Es una función de
JBuilder Developer y
Enterprise.**

Los grupos de proyectos permiten controlar el orden de generación de los proyectos dentro del grupo. Esto resulta de gran utilidad si un proyecto depende de otro. En este caso, puede que le interese crear las dependencias en primer lugar. Por ejemplo, si el proyecto B depende del proyecto A, genere en primer lugar el proyecto A y, a continuación, el proyecto B.

El orden de generación de proyectos dentro de un grupo de proyectos se puede modificar en la ficha Generar del cuadro de diálogo Propiedades del grupo de proyectos (Proyecto|Propiedades del grupo de proyectos|Orden de generación) o arrastrando un proyecto a una nueva ubicación en el panel del proyecto.

Consulte

- [Capítulo 3, “Los grupos de proyectos”](#)

Definición del orden de generación de un grupo de proyectos

El orden de generación de un grupo de proyectos depende del orden de los proyectos en el panel del proyecto. Por ejemplo, si un grupo de proyectos cuenta con dos nodos de proyectos, `project1.jpx` y `project2.jpx`, y `project1.jpx` es el primer nodo dependiente del grupo de proyectos, JBuilder genera primero `project1.jpx` y, a continuación, `project2.jpx`. La orden de generación se puede modificar en la ficha Generar del cuadro de diálogo Propiedades de grupo de proyectos.

Puede resultar muy útil controlar el orden de generación de un grupo de proyectos si se ha añadido un proyecto a otro en forma de biblioteca

necesaria. Si desea que el proyecto necesario se genere en primer lugar, hay que colocar ambos proyectos en un grupo de proyectos y hacer que el necesario esté el primero en el grupo. La adición de un proyecto como una biblioteca necesaria es una característica de JBuilder Enterprise. Si desea más información, consulte [“Adición de proyectos como bibliotecas necesarias” en la página 3-4](#).

Existen dos formas de cambiar el orden de generación de un grupo de proyectos:

- Reordenar los proyectos de un grupo arrastrándolos a una nueva posición dentro del panel del proyecto.
- Reordenar los proyectos en el cuadro de diálogo Propiedades del grupo de proyectos de la siguiente manera:
 - a Seleccione Proyecto|Propiedades del grupo de proyectos, o bien, haga clic con el botón derecho del ratón en el nodo del grupo de proyectos, en el panel del proyecto, y elija Propiedades.
 - b Haga clic en el nodo Generar del árbol.
 - c Seleccione un proyecto de la lista y utilice los botones Desplazar hacia arriba o Desplazar hacia abajo para reorganizar el orden.
 - d Haga clic en Aceptar para cerrar el cuadro de diálogo. Observe que el orden de los proyectos en el panel del proyecto ha cambiado de acuerdo con el nuevo orden de generación que acaba de especificar.

Sugerencia

También se pueden añadir proyectos en el cuadro de diálogo Propiedades del grupo de proyectos. Seleccione el botón Ayuda si desea obtener más información.

Generación de un grupo de proyectos

Para generar o generar de nuevo un grupo de proyectos:

- 1 Especifique el orden de generación de los proyectos del grupo tal y como se describe en [“Definición del orden de generación de un grupo de proyectos” en la página 6-9](#).
- 2 Realice una de las operaciones siguientes:
 - Seleccione Proyecto|Ejecutar Make del grupo de proyectos, o bien, Proyecto|Generar de nuevo el grupo de proyectos.
 - En el panel del proyecto, haga clic con el botón derecho del ratón en el archivo del grupo de proyectos (.jpr) y, a continuación, seleccione Ejecutar Make del grupo de proyectos o Generar de nuevo el grupo de proyectos.
 - Seleccione Ejecutar Make del grupo de proyectos en la barra de herramientas o elija Generar de nuevo el grupo de proyectos en la lista desplegable de la barra de herramientas, si está disponible.

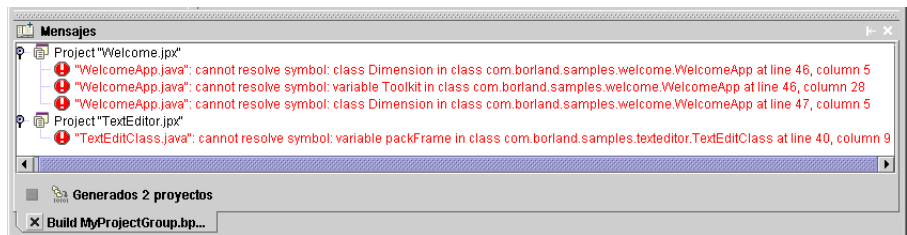


Las opciones Ejecutar Make del grupo de proyectos y Generar de nuevo el grupo de proyectos también se encuentran en la barra de herramientas. Por

defecto, Ejecutar Make del grupo de proyectos es el botón de la barra de herramientas y Generar de nuevo el grupo de proyectos se encuentra en la lista desplegable junto al botón. Si añade tipos de generación personalizados, también aparecen en la lista desplegable. A las dos primeras opciones de menú de la porción de generación del grupo de proyectos del menú Proyecto se les asignan configuraciones de teclado, que se pueden modificar en el Editor de configuración de teclado.

Durante la generación de un proyecto, todas las acciones de JBuilder se bloquean. Si desea continuar trabajando durante la generación, puede hacer clic en el botón Segundo plano en el cuadro de diálogo Avance de la generación para enviar la generación al segundo plano. El avance de la generación aparece en la barra de estado del panel de mensajes. Si prefiere ejecutar todas las generaciones del proyecto en segundo plano, puede configurar la opción global Generar en segundo plano en el cuadro de diálogo Preferencias (Herramientas|Preferencias|Visualizador|Generar). Si desea más información, consulte [“Configuración de las preferencias de generación” en la página 6-8](#).

Cuando se genera un grupo de proyectos, los archivos generados se agrupan por proyectos en la pestaña Generar del panel de mensajes. Si desea más información sobre el panel de mensaje, consulte [“Utilización del panel de mensajes” en la página 6-6](#).



Consulte

- [“El comando Ejecutar Make comandos” en la página 6-4](#)
- [“El comando Generar de nuevo” en la página 6-5](#)

Adición de tipos de generación de grupos de proyectos al menú Proyecto



JBuilder permite añadir nuevos tipos de generación al menú Proyecto para grupos de proyectos, al igual que personalizar el orden de los menús. Puede añadir el comando de menú Limpiar el grupo de proyectos, así como tipos de generación personalizados que especifiquen una colección de tipos de generación que se ejecuten con un comando de menú. Todos los tipos que se añadan al menú Proyecto también aparecen en el menú contextual. Si desea más información, consulte [“Configuración del menú Proyecto para grupos de proyectos” en la página 6-36](#).

Generación con archivos Ant

JBuilder admite la generación de proyectos con Ant. También es posible añadir archivos de generación Ant al proyecto, importar proyectos de Ant y exportar proyectos de JBuilder a archivos de generación Ant. El archivo de generación Ant permite ejecutar Ant de forma independiente o en JBuilder. Ant es una herramienta de generación basada en Java que utiliza archivos de generación creados en XML. Los archivos de generación utilizan un árbol de tipos en el que se ejecutan varias tareas. Un tipo, que es el conjunto de tareas que hay que ejecutar, puede depender de otros tipos. Entre los ejemplos de tipos se incluyen la compilación, el empaquetado en archivos JARS para la distribución, la limpieza de directorios, etc.

Por ejemplo, el siguiente archivo de generación cuenta con dos tipos, `init` y `compile`. El tipo `init` ejecuta una tarea que crea un directorio `build`. El tipo `compile`, que depende del tipo `init`, ejecuta la tarea **javac** en el directorio `src` y envía las clases compiladas al directorio `build`. Como `compile` depende de `init`, `init` debe ejecutarse primero. El directorio `build` debe crearse antes, para poder compilar las clases. Los archivos de generación también cuentan con un tipo por defecto, que en este ejemplo es `compile`, y que se ejecuta si no se especifica tipo.

Ejemplo de archivo de generación

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="MyProject" default="compile" basedir=".">
  <!--
    [ property definitions ]
    [ path and patternset ]
    [ targets ]
  -->
  <property name="build" value="build"/>
  <property name="src" value="src"/>
    <target name="init">
      <mkdir dir="${build}"/>
    </target>
    <target name="compile" depends="init">
      <javac srcdir="${src}" destdir="${build}"/>
    </target>
  </project>
```

Los archivos de generación pueden tener un conjunto de propiedades con un nombre que distingue entre mayúsculas y minúsculas y con un valor. Las propiedades se pueden utilizar como valores en las tareas, y van entre `{ }`. En el ejemplo anterior, la propiedad `build` tiene un valor `build`. El tipo `init`, cuando ejecuta la tarea `<mkdir dir="${build}"/>`, crea un directorio `build` de acuerdo con el valor de la propiedad `build`.

Las propiedades son un mecanismo útil utilizado para pasar parámetros a tareas y a tipos de generación sin modificar las propiedades existentes en el archivo de generación. El cuadro de diálogo Propiedades Ant permite añadir y

eliminar este tipo de propiedades. Si desea más información, consulte [“Configuración de las propiedades de Ant” en la página 6-23](#).

Importante

Si utiliza fragmentos de XML en el archivo de configuración Ant, debe desactivar la opción Omitir DTD de la ficha XML del cuadro de diálogo Preferencias (Herramientas/Preferencias). Por defecto, el editor de JBuilder omite los DTD.

Consulte

- [Capítulo 18, “Tutorial: Creación de un proyecto con un archivo de generación Ant”](#)
- El proyecto Jakarta en Apache: <http://ant.apache.org/>
- La documentación de Ant en <http://ant.apache.org/manual/>
- La documentación de Ant en el archivo ZIP Ant del directorio `extras` de JBuilder

Adición de archivos de generación Ant a los proyectos

Existen dos modos de añadir archivos de generación Ant a un proyecto: automáticamente, con el Asistente para Ant, o manualmente, en Proyecto/Añadir archivos/Paquetes/Clases. Si añade archivos de generación con el Asistente para Ant, JBuilder los reconoce automáticamente como nodos Ant, y muestra iconos Ant para los nodos del archivo de generación. Si añade archivos de generación manualmente mediante Proyecto/Añadir archivos/Paquetes/Clases, los archivos de generación de nombre `build.xml` son los únicos que se reconocerán como archivos de generación Ant. Puede utilizar otros nombres para los archivos de generación, pero debe configurar una opción en las propiedades del nodo para que JBuilder los reconozca como archivos Ant. Además, si el archivo de generación Ant tiene por nombre `build.xml`, la vía de acceso correspondiente al archivo aparece en el panel del proyecto. Si desea obtener más información sobre el cambio de propiedades de los nodos Ant, consulte [“Configuración de las propiedades de Ant” en la página 6-23](#).

Adición de archivos Ant con el Asistente para Ant

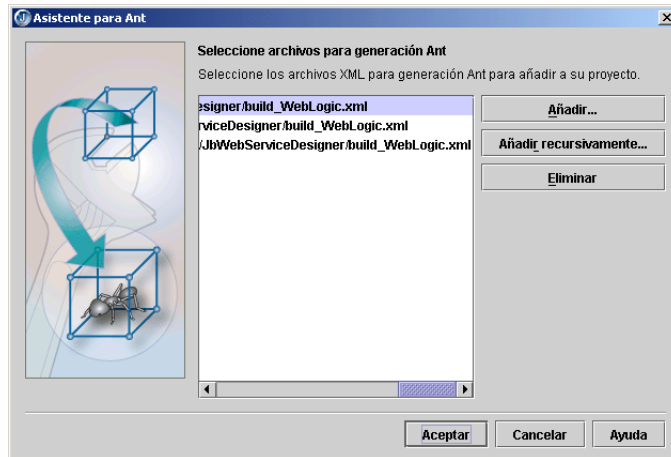
El modo más sencillo de añadir archivos de generación Ant a su proyecto es mediante el Asistente para Ant. El asistente asigna automáticamente el valor del archivo de generación Ant a la propiedad del archivo, para que JBuilder lo reconozca como un nodo Ant, sin tener en cuenta el nombre del archivo. Una vez que haya añadido un archivo de generación al proyecto con la ayuda del asistente, aparece en el panel del proyecto con un icono Ant.

Para añadir un archivo de generación Ant mediante el asistente:

- 1 Seleccione Archivo|Nuevo, haga clic en Generar en el árbol y haga doble clic en el icono Ant, o bien, seleccione Asistentes|Ant.

2 Realice una de las operaciones siguientes:

- Seleccione el botón Añadir, busque todos los archivos de generación XML que desee añadir y, a continuación, pulse Aceptar. Si utiliza el botón Añadir, todos los archivos XML que añada tienen asignado automáticamente el valor de archivo de generación Ant para que JBuilder los reconozca como un nodo Ant, sin tener en cuenta el nombre del archivo. Una vez que haya añadido un archivo de generación mediante el botón Añadir, éste aparece en el panel del proyecto con un icono Ant.
- Seleccione el botón Añadir con subpaquetes, elija un directorio y pulse Aceptar. JBuilder examina todos los archivos que tengan por nombre `build*.xml` en el directorio seleccionado y en todos sus subdirectorios, y los añade al proyecto.



3 Pulse Aceptar para cerrar el asistente.

Adición manual de archivos Ant

Si añade manualmente los archivos de generación Ant al proyecto, deben tener por nombre `build.xml` para que JBuilder los reconozca automáticamente. Si un archivo tiene un nombre diferente, aparece en el panel del proyecto con el icono XML. Para que JBuilder lo reconozca como un archivo Ant, debe configurar las propiedades del nodo tal y como se describe en el último paso.

Para añadir manualmente un archivo de generación Ant:

- 1 Pulse el botón Añadir archivos/Paquetes/Clases de la barra de herramientas del panel del proyecto o, bien, seleccione Proyecto/Añadir archivos/Paquetes/Clases.
- 2 Busque y seleccione el archivo de generación que desee añadir.
- 3 Pulse Aceptar.

- 4 Si el archivo de generación Ant no se llama `build.xml`, cambie las propiedades del nodo del archivo como se describe a continuación:
 - a Haga clic con el botón derecho del ratón sobre el archivo de generación en el panel del proyecto y seleccione Propiedades.
 - b Seleccione la ficha Ant y escoja la opción Archivo de generación Ant.
 - c Pulse Aceptar para cerrar la ficha Propiedades. El archivo de generación aparece con un icono Ant.

Creación y modificación de archivos de generación Ant

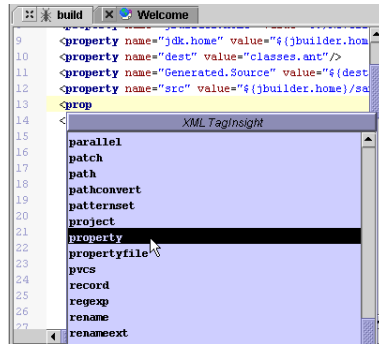
Si no cuenta con un archivo de generación, puede crearlo en el editor de JBuilder, que también ofrece resaltado de sintaxis. También puede utilizar el editor de JBuilder para modificar los archivos de generación Ant ya creados. Si desea generar nuevos archivos de generación en JBuilder:

- 1 Abra un proyecto o cree uno nuevo.
- 2 Seleccione Archivo|Nuevo archivo o haga clic con el botón derecho del ratón en el nodo de proyecto `.jpx` y seleccione Nuevo|Archivo.
- 3 Escriba un nombre para el archivo de generación y elija la extensión XML en la lista desplegable Tipo. Si lo guarda con el nombre `build.xml`, lo reconoce automáticamente como un archivo de generación Ant. Si el archivo no tiene el nombre `build.xml`, debe activar la opción Archivo de generación Ant en Propiedades Ant para que JBuilder pueda reconocerlo como nodo Ant. Consulte [“Configuración de las propiedades de Ant” en la página 6-23](#).
- 4 Asegúrese de que el directorio del proyecto se encuentra en la lista desplegable Directorio.
- 5 Active la opción Añadir archivo al proyecto y pulse Aceptar.
- 6 Introduzca la información de generación apropiada del nuevo archivo en el editor y guarde el archivo. Mientras se escribe, se recogen los errores en el panel de estructura. Para obtener más información acerca de XML y el editor, consulte “XML en el editor” en la *Guía del desarrollador de XML*.

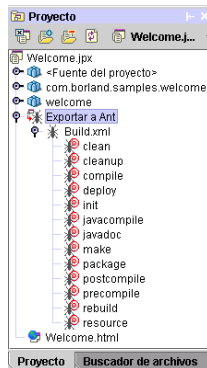
Sugerencia

Se puede utilizar TagInsight de XML para facilitar la entrada automática de elementos y atributos. Para llamar a XML TagInsight, utilice la tecla abreviada MemberInsight, `Ctrl+H` y `Ctrl+Space` en la configuración de teclado CUA. Las combinaciones de teclas de otras configuraciones de teclado se enumeran en el Editor de configuración de teclado (Herramientas|Preferencias|Configuración del teclado). Cuando se abre TagInsight, éste intenta abrir la ventana adecuada sensible al contexto en la posición del cursor. Si las características automáticas emergentes están activadas en la ficha TagInsight del cuadro de diálogo Opciones del IDE (Herramientas|Opciones del IDE|Editor|TagInsight), puede introducir un corchete angular izquierdo (`<`) o un caracter ampersand (`&`) en el editor para mostrar automáticamente TagInsight. Si se introduce un espacio o un signo igual =

dentro de las etiquetas XML también se activa el temporizador. Si desea obtener más información, consulte "TagInsight" en *Introducción a JBuilder*.



- 7 Pulse el botón de actualización y amplíe el nodo Ant del panel del proyecto para que aparezcan los tipos, los cuales aparecen en orden alfabético.



Si lo prefiere, puede crear automáticamente el archivo de generación Ant a partir de un proyecto de Jbuilder con el Asistente Exportar a Ant. Si desea más información, consulte ["Exportación de proyectos de JBuilder a Ant"](#) en la [página 6-25](#).

Si desea personalizar los colores de los elementos y atributos XML en el editor, seleccione Herramientas|Preferencias|Editor|Colores|HTML/XML/JSP. Si desea obtener más información acerca del uso de XML, consulte "Utilización de XML en el editor" en la *Guía del desarrollador de XML*.

Especificación de las vías de acceso en los archivos de generación Ant

Para crear un archivo de generación Ant o añadir un archivo de generación externo al proyecto, debe especificar las vías de acceso en el archivo de generación Ant. Por ejemplo, si el archivo de generación contiene tareas de generación que utilizan un servidor o emplea controladores JDBC, es necesario especificar la ubicación del servidor o los controladores JDBC

colocando `<pathelement>` en el elemento `<path>`. El atributo `location` de `<pathelement>` indica un solo archivo o directorio con relación al directorio raíz del proyecto o un nombre de archivo absoluto.

Cuando se deben indicar los valores de vía de acceso se puede utilizar un elemento anidado. Normalmente, la estructura es la siguiente:

```
<vía de acceso a clases>
  <pathelement path="${classpath}"/>
  <pathelement location="lib/helper.jar"/>
</classpath>
```

Si se desea usar una vía de acceso para varias tareas, puede definir las con un elemento `<path>` en el mismo nivel que los tipos de generación y hacer referencias a ellas por medio de sus atributos `id`:

```
<project ... >
  <path id="base.path" >
    <pathelement path="${classpath}"/ >
    <pathelement location="classes"/ >
  </path >

  <path id="tests.path" >
    <path refid="base.path"/ >
    <pathelement location="testclasses"/ >
  </path >

  <project ... >
    <path id="project.class.path">
      <pathelement location="lib/" >
      <pathelement path="${java.class.path}"/ >
      <pathelement path="${additional.path}"/ >
    </path>

    <target ... >
      <rmic ...>
        <classpath refid="project.class.path"/>
      </rmic>
    </target>

    <target ...>
      <javac ...>
        <classpath refid="project.class.path"/>
      </javac>
    </target>
  </project>
```

En el ejemplo siguiente, la propiedad de DataExpress tiene un valor del directorio lib de JBuilder, y la vía de acceso al directorio DataExpress se define en el atributo location de <pathelement>:

```
<project basedir="." default="rebuild" name="Employee.jpx">
  <!-- set global properties for this build -->
  <property name="Data.Express.home" value="${jbuilder.home}/lib"/>
  <property name="Server.home" value="C:/MyAppServer"/>

  . . .

  <path id="project.class.path">
    <pathelement location="${jbuilder.home}/samples/JDataStore/dsbasic"/>
    <pathelement location="${Data.Express.home}/dx.jar"/>
    <pathelement location="${Data.Express.home}/beandt.jar"/>
    <pathelement location="${Data.Express.home}/dbswing.jar"/>
    <pathelement location="${Server.home}/SonicMQ/lib/sonic_Client.jar"/>
  </path>

  . . .
</project>
```

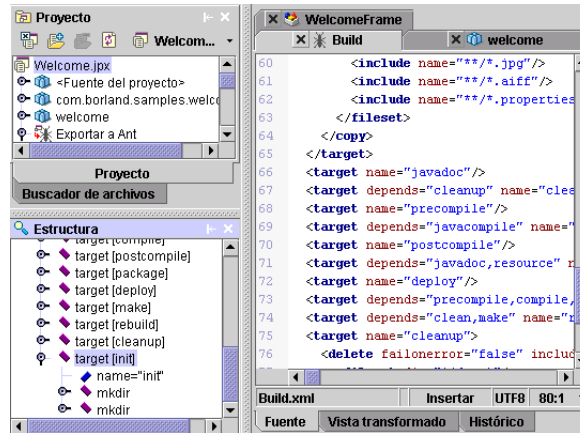
Para obtener más información sobre la forma de especificar las vías de acceso en Ant, consulte la documentación de Ant, en <http://ant.apache.org/manual/index.html>.

Cuando se exporta un proyecto de JBuilder a Ant mediante el Asistente Exportar a Ant, JBuilder define automáticamente las vías de acceso. Si desea más información, consulte [“Exportación de proyectos de JBuilder a Ant” en la página 6-25](#).

Desplazamiento por archivos de generación Ant

Los archivos de generación Ant se muestran como nodos en el panel del proyecto, con los tipos de generación como nodos dependientes. Para desplazarse por un archivo de generación es necesario abrirlo en el editor y utilizar el panel de estructura o elegir Buscar|Buscar.

- 1 Haga doble clic en el archivo de generación Ant del panel del proyecto para abrirlo en el editor y mostrar su estructura en el panel de estructura.
- 2 Realice una de las operaciones siguientes:
 - Elija un nodo en el panel de estructura (no en el panel del proyecto), por ejemplo, una propiedad o un tipo de generación, para resaltar la línea correspondiente en el editor. Haga doble clic en un nodo en el panel de estructura para desplazar el cursor hasta la línea de código en el editor. Observe que el panel de estructura muestra el valor del atributo name para facilitar la navegación. Por ejemplo, target [clean].
 - Seleccione Buscar|Buscar en el editor y escriba el nombre de la propiedad o el tipo de generación para realizar la búsqueda.



Generación de proyectos en Ant

Mientras trabaja con un proyecto Ant, puede ejecutar Ant como parte del proceso de generación de JBuilder. Para poder hacer esto, debe activar la opción **Ejecutar siempre Ant al generar el proyecto** para cualquier nodo Ant que desee incluir en el proceso de generación. Consulte [“Configuración de las propiedades de Ant” en la página 6-23](#). Si se elige esta opción, los comandos **Ejecutar Make** del proyecto y **Generar de nuevo el proyecto** ejecutan Ant como parte del proceso de generación de JBuilder. Si esta opción está desactivada, los comandos **Ejecutar Make** del proyecto y **Generar de nuevo el proyecto** realizan el proceso de generación de JBuilder sin ejecutar Ant en los nodos. Consulte [“Generación de proyectos Ant con el comando Ejecutar” en la página 6-22](#).

Sugerencia También puede añadir tipos Ant al menú **Proyecto** y a la barra de herramientas. Consulte [“Configuración del menú Proyecto” en la página 6-34](#).

Ant se puede ejecutar manualmente desde el panel del proyecto. Pulse con el botón derecho del ratón en el nodo Ant y seleccione **Ejecutar Make** para ejecutar el tipo por defecto en el archivo de generación Ant. El tipo por defecto es un valor que se especifica en el elemento `<project>`. Si desea ejecutar varios tipos en el archivo de generación, elija uno o varios nodos de tipos, pulse con el botón derecho y, a continuación, elija **Ejecutar Make**.

Nota JBuilder puede utilizar distintas vías de acceso y directorios para los archivos fuente, archivos de clase y otros archivos. Puede modificar las vías de acceso a JBuilder para que coincidan con sus tipos Ant en la ficha **Vías de acceso de Propiedades de proyecto**. También puede modificar las vías de acceso a Ant si cambia las propiedades Ant. Consulte [“Configuración de las propiedades de Ant” en la página 6-23](#).

El archivo de salida de Ant aparece en la pestaña Generar del panel de mensajes. Hay dos nodos que pueden mostrar mensajes:

- StdErr: muestra el flujo de salida de errores estándar.
- StdOut: muestra el flujo de salida estándar.

Si desea desplazarse por los archivos que contengan errores, pulse en los mensajes de error del panel de mensajes. Haga doble clic sobre cualquier error para desplazar el cursor al error del código.

Si desea introducir distintos parámetros de tipos de generación sin modificar el archivo de generación, pulse con el botón derecho del ratón sobre el nodo Ant, seleccione Propiedades y añada los parámetros. Consulte [“Configuración de las propiedades de Ant” en la página 6-23](#).

Definición del JDK

Por defecto, Ant utiliza el JDK que se distribuye con JBuilder para generar los proyectos. En algunos casos, puede que el proyecto utilice un JDK diferente. Si desea que Ant utilice el mismo JDK que el proyecto, puede definir la opción Utilizar JDK del proyecto al ejecutar Ant en la ficha Ant del cuadro de diálogo Propiedades de proyecto (Proyecto/Propiedades de proyecto/Generar/Ant).

Para configurar Ant para que utilice el JDK del proyecto:

- 1 Seleccione Proyecto|Propiedades de proyecto, o bien, haga clic con el botón derecho del ratón en el archivo .jpx del panel del proyecto y, a continuación, elija Propiedades.
- 2 Amplíe el nodo Generar en el árbol y seleccione Ant.
- 3 Marque la opción Utilizar JDK del proyecto al ejecutar Ant.
- 4 Haga clic en Aceptar para cerrar el cuadro de diálogo.

Adición de bibliotecas

Si el proyecto utiliza bibliotecas, es necesario que se especifiquen en la ficha Ant del cuadro de diálogo Propiedades de proyecto (Proyecto|Propiedades de proyecto|Generar|Ant), para que Ant las pueda encontrar al generar. Sólo tiene que configurar la opción Utilizar las bibliotecas del proyecto al ejecutar Ant para que se añadan automáticamente todas las bibliotecas del proyecto.

Puede haber casos en los que tenga bibliotecas personalizadas que contengan tareas de generación Ant hechas a medida. Puede añadir estas bibliotecas al proyecto en la ficha Ant del cuadro de diálogo Propiedades de proyecto (Proyecto|Propiedades de proyecto|Generar|Ant). Por ejemplo, puede tener tareas de generación en su archivo de generación Ant que necesiten ejecutar herramientas tales como ANTLR Translator generator, Java mail o JUnit testing. Puede crear una biblioteca personalizada que incluya las vías de acceso a estas herramientas y añadirla a su proyecto en la ficha Ant.

También puede utilizar una versión diferente de Ant si añade una biblioteca con los JAR de Ant. Si no especifica ningún JAR de Ant, JBuilder utiliza el Ant que se suministra en el directorio lib de JBuilder.

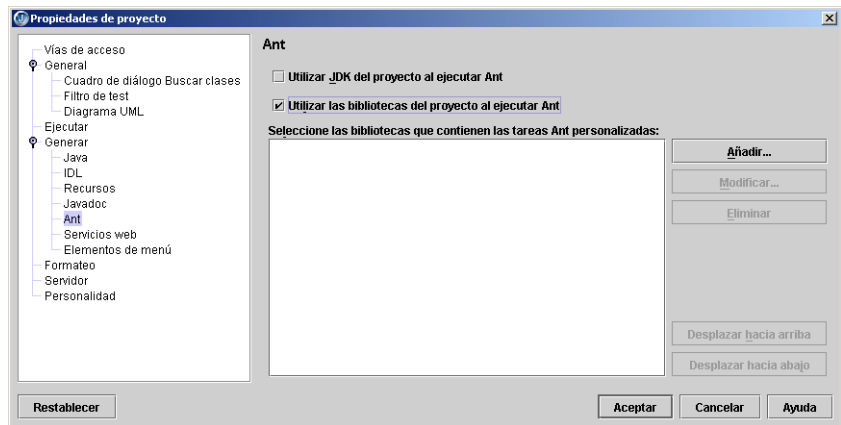
Hay dos modos de añadir bibliotecas en la ficha Ant (Proyecto|Propiedades de proyecto|Generar|Ant):

- Añadir bibliotecas de proyecto automáticamente.
- Añadir manualmente bibliotecas Ant personalizadas y/o una versión diferente de Ant.

Adición de bibliotecas de proyecto

Para añadir automáticamente bibliotecas de proyecto, para que Ant las pueda encontrar al generar:

- 1 Seleccione Proyecto|Propiedades de proyecto|Generar|Ant.
- 2 Marque la opción Utilizar las bibliotecas del proyecto al ejecutar Ant. Si se activa esta opción, las bibliotecas de proyecto se añaden automáticamente, para que Ant sepa dónde encontrarlas al generar el proyecto.

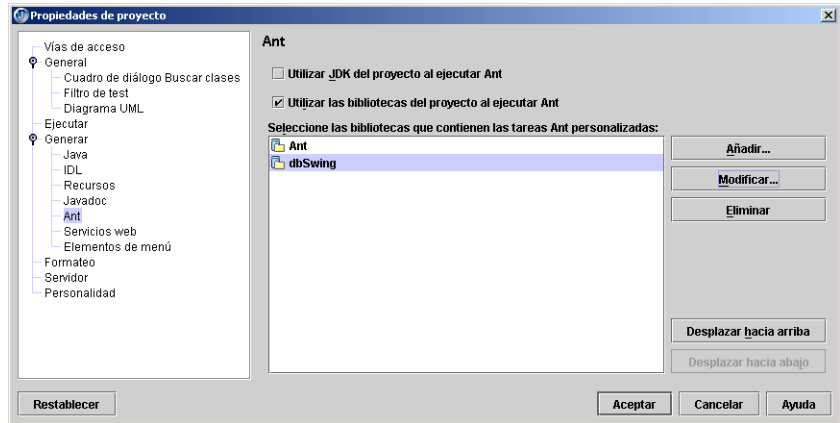


Adición de bibliotecas Ant

Para añadir manualmente bibliotecas Ant personalizadas o una versión diferente de Ant:

- 1 Seleccione Proyecto|Propiedades de proyecto|Generar|Ant.
- 2 Pulse el botón Añadir para abrir el cuadro de diálogo en la ficha Ant Seleccionar bibliotecas.
- 3 Seleccione una biblioteca de la lista o pulse el botón Nueva para crearla con el Asistente para bibliotecas.

- 4 Pulse Aceptar para cerrar el cuadro de diálogo Seleccione una biblioteca y añadir la biblioteca a la lista de Bibliotecas que contienen tareas Ant personalizadas.



- 5 Reorganice las bibliotecas de la lista mediante los botones Desplazar hacia arriba o Desplazar hacia abajo.

Nota

La búsqueda en las bibliotecas se realiza por el orden en que figuran en la lista, de arriba a abajo.

- 6 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

Generación de proyectos Ant con el comando Ejecutar

Cuando se ejecuta un proyecto Ant en JBuilder con el comando Ejecutar el proyecto (Ejecutar|Ejecutar el proyecto), JBuilder ejecuta el tipo de generación por defecto Ejecutar Make. A continuación, JBuilder ejecuta el proyecto sin ejecutar Ant. Si desea ejecutar también Ant con este comando, debe activar la opción Ejecutar siempre Ant al generar el proyecto para cualquier nodo Ant que desee incluir en el proceso de generación. En ese momento, JBuilder ejecuta Make para el proceso de generación de JBuilder y para Ant, utilizando el tipo Ant por defecto en el archivo de generación. Consulte [“Configuración de las propiedades de Ant” en la página 6-23](#). Tras haber activado esta opción, el comando Ejecutar el proyecto genera el proyecto con Ant en el nodo especificado como parte del proceso de generación de JBuilder. A continuación, se ejecuta el programa.

Además, puede cambiar el tipo de generación por defecto, Ejecutar Make, antes de ejecutar el programa. Por ejemplo, quizás desee ejecutar el tipo Ant antes de ejecutar el proyecto. En este caso, no necesitaría seleccionar la opción Ejecutar siempre Ant al generar el proyecto. El tipo de generación se especifica en las configuraciones para la ejecución en el cuadro de diálogo Propiedades de ejecución (Ejecutar|Configuraciones). Si desea más información sobre cómo modificar el tipo de generación, consulte [“Selección de tipos de generación” en la página 7-12](#).

Consulte

- [“Generación de proyectos con el comando Ejecutar” en la página 5-5](#)

Configuración de las propiedades de Ant

Un proyecto Ant puede tener un conjunto de propiedades. La mayoría de los tipos y tareas Ant están enlazados a la propiedad. Por ejemplo, la propiedad `build.compiler` especifica qué compilador utiliza la tarea **javac**. También se puede especificar si desea que una tarea se ejecute dependiendo de la existencia o ausencia de una determinada propiedad. Las propiedades también se utilizan para pasar parámetros a las tareas sin necesidad de redefinir las propiedades del archivo de generación.

En el cuadro de diálogo Propiedades del archivo de generación se pueden pasar parámetros y controlar las opciones para iniciar Ant. Pulse con el botón derecho del ratón en el nodo Ant del panel del proyecto, elija Propiedades y seleccione Ant en el árbol. En el cuadro de diálogo Propiedades puede definir las siguientes opciones:

- **Archivo de generación Ant**

Elija esta opción para identificar el archivo XML como un archivo de generación Ant. Si se llamara a un archivo de generación `build.xml`, se seleccionaría esta opción por defecto y se desactivaría.

- **Mostrar vía de acceso relativa**

Elija esta opción para que aparezca en el panel del proyecto la vía de acceso relativa de todos los archivos de generación Ant que tengan por nombre `build.xml`.

- **Nivel del histórico**

Puede elegir entre silencioso, normal, verbose o depurar para la salida del mensaje.

- **Utilizar archivo de registro**

Envía los mensajes de salida a un archivo histórico en lugar de al panel de mensajes.

- **Propiedades**

Cómo añadir nuevas propiedades y modificar las ya existentes sin sobrescribirlas en el archivo de generación. Las modificaciones se guardan en el archivo `<project>.jpx` file, no en el archivo XML.

- **Utilizar el compilador Java de Borland**

Utilizar el compilador Borland de Java, Borland Make (**bmj**), para las tareas **javac** de Ant.

- **Parámetros de la MV**

Especifica parámetros adicionales de la MV cuando se ejecuta Ant como un proceso externo desde dentro de JBuilder. Por ejemplo, si desea que el tamaño máximo de memoria dinámica de la MV de Java en la que está

ejecutando Ant sea de 256 MB, escriba `--Xmx256m` en el campo Parámetros de MV.

■ Programación de tareas

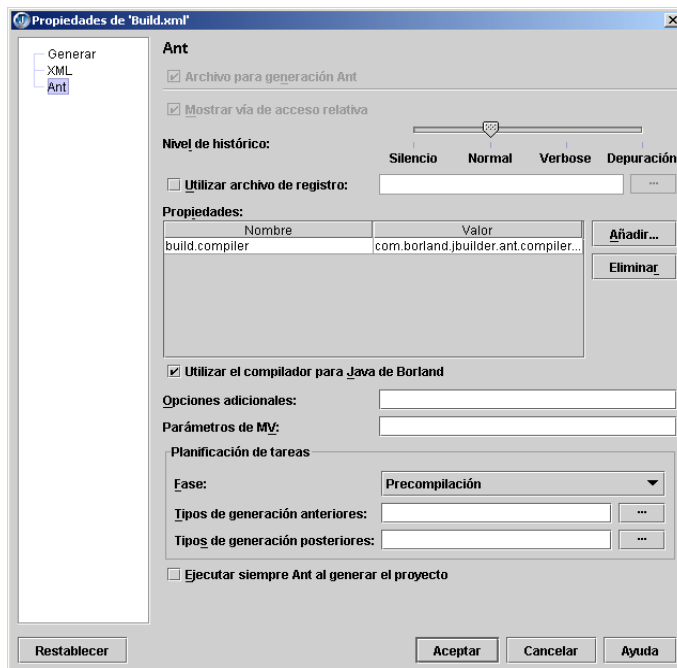
Le permite especificar la fase de generación en que se ha de ejecutar el archivo de generación y seleccionar los tipos que se han de ejecutar antes y después de que se genere el archivo Ant.

■ Ejecutar siempre Ant al generar el proyecto

Le permite ejecutar Ant en cualquier nodo Ant que tenga esta opción seleccionada durante la generación del proyecto.

Importante

Cuando se utilizan estas funciones de JBuilder como, por ejemplo, el perfeccionamiento, se recomienda que acepte la opción por defecto Utilizar el compilador Borland de Java. Si esta opción está seleccionada y tiene alguna tarea **javac** en su archivo `build.xml`, esas tareas utilizarán **bmj**. Por ejemplo, **bmj** añade información adicional en los archivos de dependencias que permitan el perfeccionamiento para poder trabajar en determinados casos en los que no se puede recabar la información necesaria para el perfeccionamiento desde los archivos `.class` por sí solos.



Para obtener información adicional sobre la configuración de las propiedades Ant, pulse el botón Ayuda en el cuadro de diálogo Propiedades Ant.

Puede definir opciones adicionales de Ant en la ficha Ant del cuadro de diálogo Propiedades de proyecto. Si desea más información, consulte [“Definición del JDK” en la página 6-20](#) y [“Adición de bibliotecas” en la página 6-20](#).

Opciones Ant

Puede introducir opciones Ant adicionales en el campo Opciones adicionales del cuadro de diálogo Propiedades.

Algunos de estas opciones son:

```
-help imprime el mensaje
-projecthelp imprime la ayuda del proyecto
-version imprime los datos de la versión y sale de la aplicación
-emacs produce información de seguimiento sin adornos
-logger classname muestra la clase que ha de llevar a cabo el seguimiento
-listener classname añade una instancia de clase como monitor del proyecto
-find file busca el archivo de generación en dirección a la raíz del
sistema de generación y utiliza el primero que encuentra
```

Si desea más información sobre opciones Ant, consulte la documentación de Ant en <http://ant.apache.org/manual/index.html> o en el archivo ZIP Ant en el directorio extras de JBuilder.

Importación de proyectos Ant

**Es una función de
JBuilder Developer y
Enterprise.**

Si ya tiene un proyecto Ant con el que le gustaría trabajar en JBuilder, utilice el Asistente para Proyecto con código existente. El asistente para proyecto con código existente crea un proyecto nuevo en JBuilder a partir de un proyecto ya creado, derivando las vías de acceso desde el árbol de directorios. JBuilder reconoce de forma automática los archivos `build.xml` de Ant, y los añade al nuevo proyecto en JBuilder. Si el archivo Ant no tiene el nombre `build.xml`, debe activar la opción Archivo de generación Ant en la ficha Propiedades Ant. Consulte [“Configuración de las propiedades de Ant” en la página 6-23](#). Si desea abrir el asistente Proyecto para código existente, seleccione Archivo|Nuevo|Proyecto y haga doble clic sobre el icono Proyecto para código existente en la ficha Proyecto.

Nota Es posible que JBuilder importe incorrectamente algunos proyectos Ant, como los que tienen interdependencias Ant complejas.

Exportación de proyectos de JBuilder a Ant

**Es una función de
JBuilder Developer y
Enterprise.**

JBuilder admite la exportación de un proyecto de JBuilder a un archivo de generación Ant que contenga tareas estándar Ant. Puede entonces utilizar este archivo de generación Ant para generar su proyecto independientemente de JBuilder. Esto resulta útil para ejecutar las generaciones de integración de línea de comandos y también por razones de concesión de licencias.

Sin embargo, las generaciones de JBuilder y de Ant no tienen por qué ser idénticas. JBuilder utiliza tareas de Ant personalizadas, que puede ejecutar por un orden distinto al de Ant, si así lo requieren el comprobador de dependencias y el depurador de JBuilder. Algunas tareas de Ant personalizadas de JBuilder, como los ejecutables nativos y los recopilatorios JAR ejecutables, no se pueden asociar con las tareas estándar de Ant, por lo que no se incluyen en la salida de generación de Ant. Además, cuando se

ejecutan determinadas herramientas, como `rmic` y `java2iio`, JBuilder suprime la compilación de los archivos generados que realizan estas herramientas y los compila en su lugar con Borland Make (**bmj**). También ocurre esto cuando JBuilder crea archivos JAR para EJB: los archivos generados se guardan para facilitar la depuración.

Exportación de proyectos con el Asistente Exportar a Ant

JBuilder proporciona el Asistente Exportar a Ant, que crea un nodo Exportar a Ant en el panel del proyecto. Cuando se genera este nodo, JBuilder lee el archivo del proyecto, asigna las tareas de Ant personalizadas de JBuilder a tareas estándar de Ant siempre que sea posible, y genera las tareas de Ant correspondientes en un archivo de generación. No todas las tareas de Ant personalizadas de JBuilder se pueden asignar exactamente a tareas estándar de Ant.

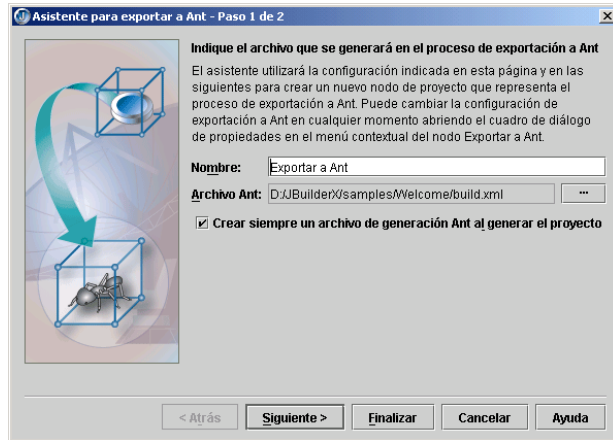
A causa de las diferencias entre la generación de JBuilder y la de Ant, es mejor separar la salida de generación de uno y otro. Por este motivo, el archivo de generación de Exportar a Ant genera las clases en un directorio de salida distinto al de JBuilder. Sin embargo, debe tener en cuenta que los recopilatorios creados por los dos procesos de generación se sobrescriben mutuamente, ya que por lo general los JAR creados por archivo y los nodos EJB no se crean en la vía de acceso de salida. Por defecto, el archivo de generación Ant generado a partir del archivo de proyecto define el directorio de salida como `<directorio de salida del proyecto>.ant`. Por ejemplo, si la vía de acceso de salida del proyecto en JBuilder es `c:/myprojects/untitled1/classes`, la vía de acceso de salida de la generación Ant es `c:/myprojects/untitled1/classes.ant`. La vía de acceso de salida de Ant es completamente configurable.

El archivo de generación Ant creado por JBuilder contiene vías de acceso que, normalmente, se relacionan con el directorio raíz Ant u otros directorios, como el de instalación de JBuilder, del JDK, de un servidor, etc. Por ello, no es necesario por lo general modificar el archivo de generación. Sin embargo, es preciso definir una propiedad antes de abrir Ant. Si modifica el archivo de generación Ant, debe desactivar la opción Crear siempre un archivo de generación Ant al generar el proyecto, ya que de lo contrario se sobrescribirá cuando se genere el proyecto. No obstante, debe tener en cuenta que si desactiva esta opción, los cambios realizados en el proyecto no se incluirán en el archivo de generación Ant. La opción Crear siempre un archivo de generación Ant al generar el proyecto se encuentra en el asistente y en la ficha Propiedades de Exportar a Ant del nodo Exportar a Ant.

Si desea crear un archivo de generación Ant para el proyecto, haga lo siguiente:

- 1 Abra el Asistente Exportar a Ant con cualquiera de estos métodos:
 - Seleccione Archivo|Nuevo|Generar y haga doble clic en el icono Exportar a Ant de la ficha Generar de la galería de objetos.

- Seleccione Asistentes|Exportar a Ant para abrir el asistente.



- 2 Acepte el nombre por defecto del nodo Exportar a Ant o modifíquelo.
- 3 Acepte el nombre y la vía de acceso por defecto del archivo de generación o pulse el botón puntos suspensivos (...) para modificarlos.
- 4 Pulse Siguiente para pasar a la ficha siguiente.
- 5 Elija una opción de Problema al exportar acción o acepte la opción por defecto. Los problemas ocurren cuando Ant no puede reproducir una tarea personalizada de JBuilder, como los ejecutables nativos. Para obtener más información sobre las acciones que ocasionan problemas de exportación, pulse el botón Ayuda del asistente.
- 6 Modifique el destino de salida de Ant o acepte la opción por defecto. Dado que las generaciones de JBuilder y Ant no tienen por qué ser idénticas, es recomendable mantenerlas separadas.
- 7 Pulse Finalizar para cerrar el Asistente Exportar a Ant.
- 8 Haga clic con el botón derecho del ratón en el nodo Exportar a Ant generado y elija Make para crear el archivo de generación Ant a partir del archivo de proyecto de JBuilder.
- 9 Amplíe el nodo Exportar a Ant y haga doble clic en el archivo de generación Ant creado para abrirlo en el editor. Las tareas se muestran en el panel del proyecto como nodos dependientes del archivo de generación Ant. Si desea desplazarse a una tarea en el editor, haga doble clic en un tipo de generación en el panel de estructura (no en el panel del proyecto) o elija Buscar|Buscar en el editor.
- 10 Haga clic con el botón derecho del ratón en un tipo de generación Ant o en el nodo del archivo de generación Ant y seleccione Ejecutar Make para ejecutar Ant.

Nota Si desea añadir al menú Proyecto tipos de generación del archivo de generación creado por el Asistente Exportar a Ant, añada el archivo de generación al proyecto y los tipos aparecerán en la lista. Para añadir el

archivo de generación basta con arrastrarlo al nodo del proyecto, en el panel del proyecto. Para obtener más información sobre la adición de tipos de generación al menú Proyecto, consulte [“Configuración del menú Proyecto” en la página 6-34](#).

Para obtener más información sobre grupos de proyectos, consulte [“Generación de proyectos en Ant” en la página 6-19](#).

Tareas de generación de JBuilder no admitidas

Dado que algunas tareas de generación personalizadas de JBuilder no tienen equivalentes en Ant, el archivo de exportación de Ant no incluye las siguientes:

- Creación de ejecutables nativos y archivos JAR ejecutables.
- Compilación de fuentes `.java` en archivos JAR.
- Es posible que la extracción de dependencias a recopilatorios no se reproduzca exactamente con la generación Ant. Por ejemplo, si se activa la opción Incluir dependencias de clases y se incluyen dependencias de biblioteca en el Creador de recopilatorios, JBuilder las incluye en el recopilatorio, pero no cuando se realiza la generación con el archivo Ant creado mediante Exportar a Ant. Esto también se aplica a los WAR y a los EJB.
- Es posible que la extracción de clases internas a recopilatorios no se reproduzca exactamente con la generación Ant. Por ejemplo, si se define en el Creador de recopilatorios un filtro que especifica expresamente una clase, el recopilatorio incluye sus clases internas cuando se genera con JBuilder, pero no cuando se realiza la generación con el archivo Ant creado mediante Exportar a Ant. Puede crear filtros que tienen el mismo comportamiento en JBuilder y en el archivo Ant exportado.

Por ejemplo, si se tiene la clase `com.borland.Application1.class`, la generación de JBuilder incluye en el archivo `Application1.class` y todas sus clases internas. Sin embargo, Ant sólo incluye `Application1.class`. Si se crean dos filtros en el Creador de recopilatorios, `com.borland.Application1.class` y `com.borland.Application1$.class`, tanto JBuilder como Ant incluyen en el archivo `Application1.class` y todas sus clases internas.

- Creación de servicios web con el kit de herramientas Apache SOAP.

Exportación de EJB a Ant

En JBuilder, los descriptores de distribución EJB e EAR se encuentran almacenados dentro de módulos EJB y EAR. Ant lleva a cabo la generación en descriptores de distribución autónomos (archivos XML). Si se modifica un módulo EJB o EAR, estos cambios se deben guardar en descriptores de distribución autónomos, o la generación de Ant tendrá lugar en descriptores de distribución desfasados o inexistentes.

La exportación de proyectos EJB a Ant requiere la desactivación de dependencias de clase para un módulo EJB y la configuración manual de dependencias de clase del siguiente modo:

- 1 Haga clic con el botón derecho del ratón en el módulo EJB, en el panel del proyecto, y elija Propiedades.
- 2 Seleccione Contenido\Clases y desactive la opción Incluir dependencias de clases en la ficha Clases.

Precaución Cuando se desactiva la opción Incluir dependencias de clases de los JAR de EJB se pueden producir errores de compilación si alguna de las clases de EJB utiliza tipos de datos personalizados, herencia, etc. En este caso, utilice la ficha Clases del cuadro de diálogo Propiedades de EJB para añadir las dependencias.

Notas adicionales sobre los EJB

No se debe utilizar el compilador Borland de Java para compilar JSP con Borland Enterprise Server cuando se usa Exportar a Ant. Para desactivar esta opción, haga clic con el botón derecho en `build.xml` en el panel del proyecto, seleccione Propiedades\Ant y desactive la opción Utilizar el compilador para Java de Borland, en la ficha Ant.

Para compilar un proyecto de servicios web basado en el kit de herramientas WebLogic mediante Exportar a Ant:

- 1 Seleccione Proyecto\Propiedades de proyecto\Generar\Ant.
- 2 Añada la biblioteca cliente WebLogic antes de compilar el proyecto con `build.xml`.

Si utiliza una plataforma UNIX, compruebe que la variable de sistema PATH incluye JDK_HOME/bin al compilar proyectos WebLogic (web/EJB/basado en servicios web).

Configuración de las propiedades de Exportar a Ant

El nodo Exportar a Ant cuenta con una ficha de propiedades en la que se puede modificar la configuración. Para abrir el cuadro de diálogo Propiedades de Exportar a Ant, haga clic con el botón derecho del ratón en el nodo Exportar a Ant y seleccione Propiedades. El cuadro de diálogo Propiedades Exportar a Ant cuenta con dos fichas: Exportar y Generar.

En la ficha Exportar se puede modificar el nombre del nodo, así como el nombre y la vía de acceso del archivo de generación. También puede especificar si desea crear el archivo de generación Ant cuando se genere el proyecto.

En la ficha Generar se puede configurar la forma de informar sobre los problemas de exportación. Estos problemas ocurren cuando no es posible exportar todas las funciones de las tareas personalizadas de JBuilder a tareas estándar de Ant. También es posible modificar el directorio de destino de la salida de generación Ant. Para obtener más información sobre estas opciones pulse el botón Ayuda.

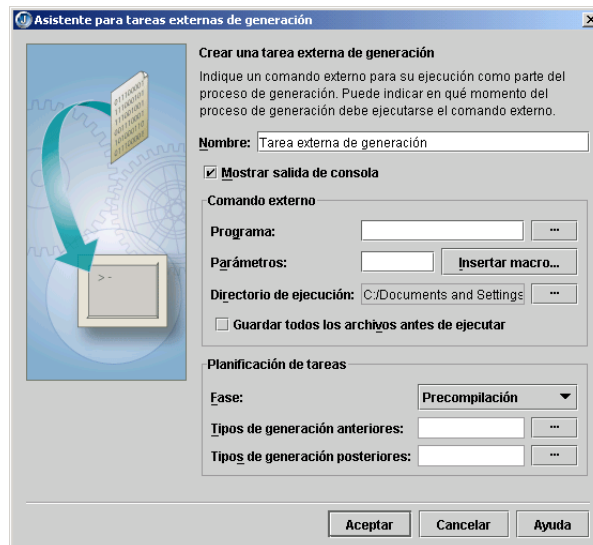
Creación de tareas externas de generación

Es una función de JBuilder Developer y Enterprise.

En algunos casos puede que desee ejecutar tareas externas cada vez que genere un proyecto. Por ejemplo, puede tener un archivo `.bat` o `.exe` para Windows o un archivo `.sh` o un ejecutable para Linux o UNIX que desee ejecutar cada vez que se genere un proyecto. Algunos ejemplos de tareas externas son: el paso de archivos compilados a un ofuscador después de la fase Empaquetar, la distribución de un programa y su incorporación a CVS o el paso de un programa a un preverificador tras la compilación. Con la ayuda del Asistente para tareas externas de generación podrá crear tareas externas que le permitan ejecutar comandos externos shell o de consola como parte del proceso de generación.

Asistente para tareas externas de generación

Con la ayuda del Asistente para tareas externas de generación se puede generar una tarea externa de generación. Para abrir el asistente, seleccione Archivo|Nuevo, a continuación la ficha Generar, y haga doble clic sobre el icono Tarea externa de generación.



En el Asistente para tareas externas de generación se pueden configurar las siguientes opciones:

- **Nombre**

Escriba un nombre para el nodo que se muestra en el panel del proyecto.

- **Mostrar salida de consola**

Muestra en el panel de mensajes de JBuilder toda la información de salida de la consola. Si esta opción está desactivada no se muestra ninguna información de salida.

- **Programa**

Puede desplazarse al archivo de programa externo que desee incluir en la generación.

- **Parámetros**

Le permite introducir parámetros y/o seleccionarlos en la lista Macros.

- **Las comillas como delimitador**

Seleccione esta opción si está usando las comillas (") como delimitador de parámetros y macros.

- **Directorio de ejecución**

Especifique el directorio desde el que se inicia la tarea externa de generación.

- **Guardar todos los archivos antes de ejecutar**

Se guardan todos los archivos antes de ejecutar el comando externo.

- **Planificación de tareas**

Indique la fase de la generación en la que debe ejecutarse la tarea externa y seleccione los elementos que se deben ejecutar antes y después de ella. La propiedad Programación de tareas determina en qué fase se va a ejecutar la tarea. Por ejemplo, si su tarea externa de generación es un ofuscador, puede configurarla en la fase Empaquetar. También puede especificar los tipos que tienen lugar antes y después de esa fase.

Si desea obtener más información sobre estas opciones, pulse el botón de ayuda del asistente.

Una vez completado el asistente, aparece un nodo en el panel del proyecto. Puede haber varios nodos de tareas externas en un proyecto. Cuando se coloca el ratón encima de cada uno de ellos, aparece la ayuda inmediata con el nombre del ejecutable.

Se pueden añadir tareas externas de generación al menú Proyecto. Consulte [“Configuración del menú Proyecto” en la página 6-34](#). También se pueden definir como tipos de generación para que se ejecuten antes que el proyecto. Consulte [“Selección de tipos de generación” en la página 7-12](#).

Generación de tareas externas

Si desea generar solamente el nodo de tareas externas, pulse con el botón derecho del ratón en el panel del proyecto y seleccione Ejecutar Make. Si se selecciona la opción Mostrar salida de consola, se envían todos los mensajes a la pestaña Generar del panel de mensajes. Hay dos nodos que pueden mostrar mensajes:

- StdErr: muestra el flujo de salida de errores estándar.
- StdOut: muestra el flujo de salida estándar.

Seleccione Proyecto|Ejecutar Make del proyecto para generar todo el proyecto y el nodo de tareas externas.

Configuración de propiedades de las tareas externas de generación

Las tareas externas de generación poseen un conjunto de propiedades que se configuran de forma inicial en el Asistente para tareas externas de generación. Estas propiedades incluyen el nombre, el ejecutable que se va a utilizar, la programación de tareas y los parámetros. Si desea modificar cualquiera de estas propiedades después de crear la tarea externa de generación, pulse con el botón derecho del ratón en el nodo del panel del proyecto y seleccione Propiedades. Para obtener más información sobre estas opciones, pulse el botón Ayuda.

Generación de archivos SQLJ

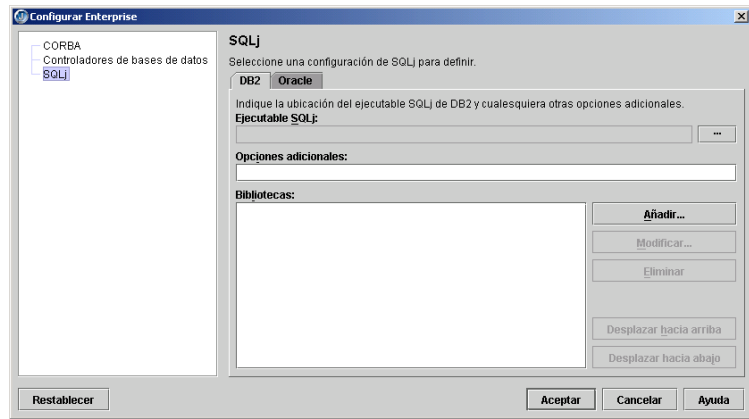
Es una función de JBuilder Enterprise.

El sistema de generación de JBuilder admite la generación de archivos SQLJ. SQLJ combina el lenguaje de programación Java con SQL (Lenguaje de consulta estructurado), que se utiliza para acceder a bases de datos relacionales. SQLJ, complementario a JDBC, permite que un programa en Java tenga acceso a una base de datos mediante sentencias SQL incrustadas. Una vez que se han incrustado las sentencias SQL, se ejecuta en el programa un traductor de SQLJ. Este traductor cambia el programa SQLJ a Java y sustituye las sentencias SQL por llamadas a la ejecución SQLJ. A continuación, el programa Java se compila y se ejecuta sobre la base de datos. Ya que SQLJ solamente es compatible con el SQL estático, puede combinarlo con JDBC para que también pueda funcionar con el SQL dinámico.

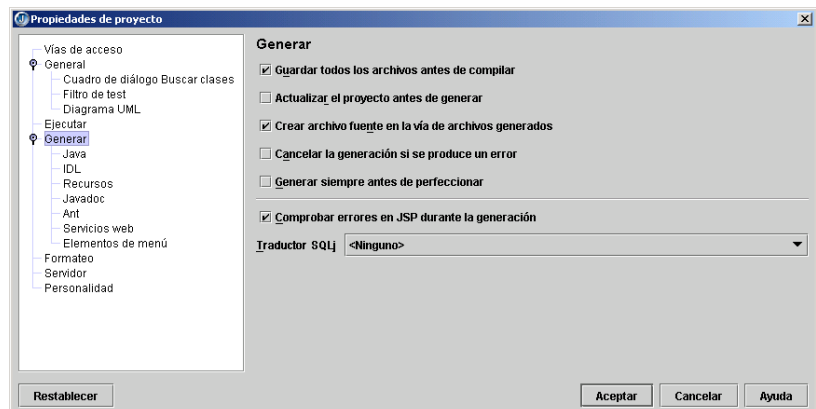
Para obtener más información sobre cómo conectarse con bases de datos, consulte *Desarrollo de aplicaciones de base de datos*.

JBuilder reconoce los archivos `.sqlj` del proceso de generación. Para generar archivos `.sqlj`, en primer lugar debe configurar un traductor y, después, especificar uno para su proyecto tal y como se explica a continuación:

- 1 Configure DB2 o Oracle SQLJ en el cuadro de diálogo Configurar Enterprise tal y como se detalla a continuación:
 - a Seleccione Herramientas|Configurar Enterprise|SQLJ.



- b Seleccione una configuración SQLJ, como Oracle o DB2.
 - c Navegue hasta la ubicación del archivo ejecutable.
 - d Introduzca las opciones adicionales.
 - e Añada cualquier biblioteca SQLJ dependiente y controladores JDBC que requiera el traductor SQLJ. Examine la documentación de DB2 u Oracle para ver qué archivos JAR se necesitan en el CLASSPATH. Cree una biblioteca con estos JARs con ayuda del Asistente para bibliotecas y añádala a la configuración de SQLJ.
 - f Haga clic en Aceptar para cerrar el cuadro de diálogo Configurar Enterprise.
- 2 Especifique el traductor de SQLJ que va a utilizar en el proyecto:
 - a Seleccione Proyecto|Propiedades de proyecto|Generar.
 - b Seleccione un Traductor SQLJ para el proyecto en la lista desplegable.



- c Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

Cuando el proyecto tiene un conversor SQLj activo, SQLj se ejecuta con todos los archivos `.sqlj` del proyecto como parte del proceso de generación, y los archivos `.java` generados aparecen como descendientes del nodo SQLj. Los archivos `.java` generados se compilan posteriormente como parte del proceso global de generación.

Consulte

- SQLJ.org en <http://www.sqlj.org>

Configuración del menú Proyecto

Para su comodidad, JBuilder permite configurar el menú Proyecto para proyectos individuales, así como para grupos de proyectos. Se pueden añadir tipos y tareas de generación adicionales si el proyecto las contiene. Los grupos de proyectos son una característica de las ediciones Developer y Enterprise de JBuilder.

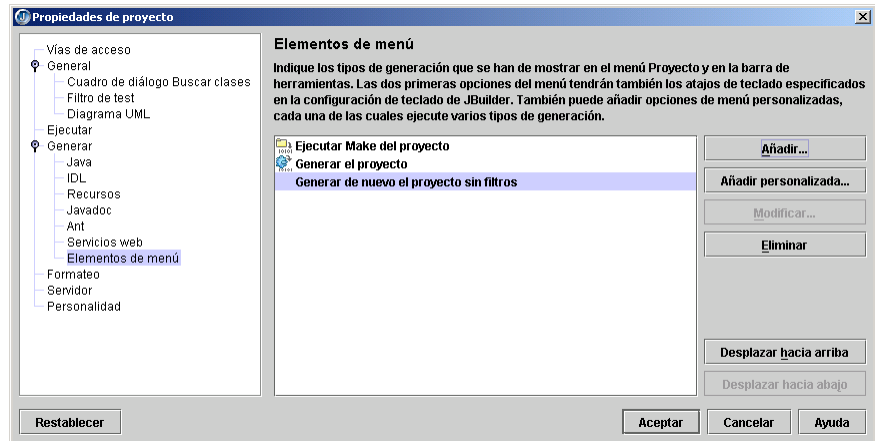
Configuración del menú Proyecto para proyectos

El primer grupo del menú Proyecto se puede configurar totalmente y muestra por defecto Ejecutar Make y Generar de nuevo. Además, puede añadir más tipos y tareas de generación, como Limpiar, tareas externas de generación y los tipos Ant, si su proyecto lo requiere.

En el menú Proyecto, se le asignan teclas por defecto a los dos primeros elementos. El primer elemento del menú aparece en la barra de herramientas. Junto al botón del menú de la barra de herramientas, hay un menú desplegable que contiene la opción Generar de nuevo, a menos que la haya eliminado del menú y haya añadido otros tipos personalizados al menú Proyecto.

Se pueden modificar estas opciones por defecto y añadir tipos personalizados en la ficha Elementos de menú en Propiedades de proyecto. En el caso de los dos primeros tipos de la ficha Elementos de menú, las teclas asignadas pueden personalizarse. Se pueden cambiar las asignaciones de teclas para estos dos primeros tipos en el Editor de configuración de teclado, que se encuentra en la ficha Asignación de teclas del cuadro de diálogo Preferencias (Herramientas|Preferencias|Asignación de teclas). El primer tipo de la lista

aparece, con el icono adecuado, en la barra de herramientas principal con una lista desplegable del resto de tipos añadidos al menú Proyecto.



Si desea añadir un tipo de proyecto al menú Proyecto:

- 1 Seleccione Proyecto|Propiedades de proyecto|Generar|Elementos de menú.
- 2 Realice una de las operaciones siguientes:
 - Pulse el botón Añadir y seleccione un tipo de la lista. Después, seleccione Aceptar para cerrar el cuadro de diálogo Añadir tipo de generación al menú.
 - Pulse el botón Añadir personalizada para añadir al menú una opción personalizada que ejecute varios tipos de generación. Escriba el nombre del menú en el campo Etiqueta de menú, pulse el botón Añadir y, a continuación, seleccione los tipos que desee. Los tipos se ejecutan en el orden en el que figuran en la lista. Utilice los botones Desplazar hacia arriba o Desplazar hacia abajo para reorganizar la lista. Pulse Aceptar para cerrar el cuadro de diálogo Nuevo tipo personalizado.
- 3 Puede cambiar el orden de los tipos en la ficha Elementos de menú seleccionando un tipo y, a continuación, Desplazar hacia arriba o Desplazar hacia abajo. Esta operación cambia el orden de los tipos en el menú Proyecto. Tal y como se ha indicado anteriormente, el primer tipo de la lista aparece en la barra de herramientas y, a los dos primeros, se les puede asignar la tecla que se desee.
- 4 Seleccione Aceptar para cerrar Propiedades de proyecto. Los nuevos tipos aparecen en el menú Proyecto y en el menú desplegable junto al tipo de la barra de herramientas.

Nota

Para configurar el menú Proyecto para otros proyectos, realice los cambios en el cuadro de diálogo Propiedades por defecto para proyectos.

Consulte

- [“Creación de tareas externas de generación” en la página 6-30](#)

- “Configuración de teclado de las emulaciones de editores” en *Introducción a JBuilder*

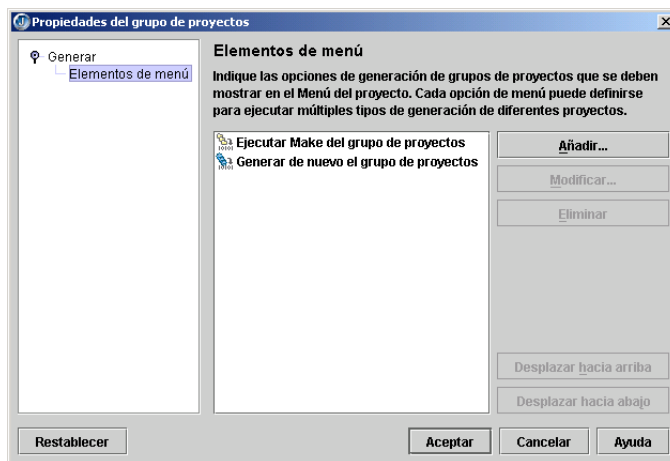
Configuración del menú Proyecto para grupos de proyectos

Es una función de JBuilder Developer y Enterprise.

JBuilder permite configurar el menú Proyecto para grupos de proyectos, al igual que para proyectos. Se puede añadir Limpiar grupo de proyectos al menú Proyecto, al igual que otros tipos del grupo de proyectos. Por defecto, Ejecutar Make del grupo de proyectos y Generar de nuevo el grupo de proyectos son los tipos de generación de grupos de proyectos que se encuentran en el menú Proyecto. Todos los tipos que se añadan al menú Proyecto también aparecen en el menú contextual.

En el menú Proyecto, se le asignan teclas por defecto a los dos primeros elementos de menú para grupos de proyectos. El primer elemento de menú para grupos de proyectos aparece en la barra de herramientas. Junto al botón del menú de la barra de herramientas, hay un menú desplegable que contiene la opción Generar de nuevo el grupo de proyectos, a menos que la haya eliminado del menú y haya añadido otros tipos personalizados al menú Proyecto para grupo de proyectos.

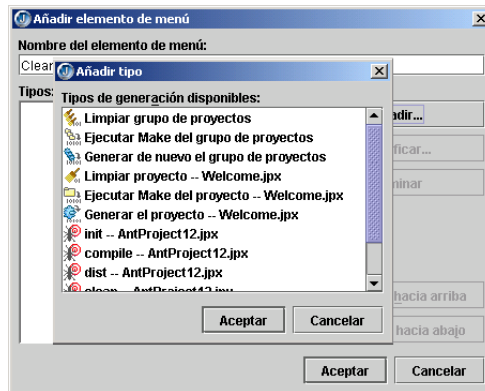
Se pueden modificar estas opciones por defecto y añadir tipos personalizados en la ficha Elementos de menú en Propiedades del grupo de proyectos. En el caso de los dos primeros tipos de la ficha Elementos de menú, las teclas asignadas pueden personalizarse. Se pueden cambiar las asignaciones de teclas para estos dos primeros tipos en el Editor de configuración de teclado, que se encuentra en la ficha Asignación de teclas del cuadro de diálogo Preferencias (Herramientas|Preferencias|Asignación de teclas). El primer tipo



de la lista aparece, con el icono adecuado, en la barra de herramientas principal con una lista desplegable del resto de tipos añadidos al menú Proyecto.

Si desea añadir un tipo de grupo de proyectos al menú Proyecto:

- 1 Elija Proyecto|Propiedades de grupo de proyectos para abrir el cuadro de diálogo Propiedades del grupo de proyectos. También se puede conseguir haciendo clic con el botón derecho del ratón sobre el nodo del grupo de proyectos en el panel del proyecto y seleccionar Propiedades.
- 2 Elija Generar|Ant.
- 3 Pulse Añadir para abrir el cuadro de diálogo Añadir elemento de menú.
- 4 Escriba un nombre de menú para el tipo.
- 5 Pulse el botón Añadir, seleccione los tipos que desee añadir y, a continuación, pulse Aceptar.

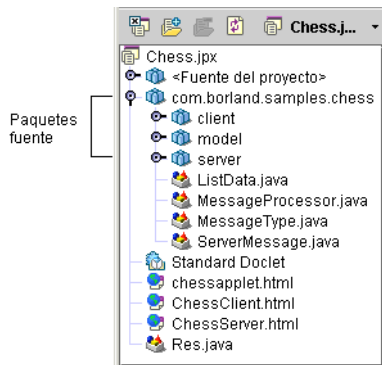


- 6 Seleccione un tipo de la lista y utilice los botones Desplazar hacia arriba o Desplazar hacia abajo en el cuadro de diálogo Añadir elemento de menú para reorganizar los tipos de la lista. Los tipos se ejecutan en el orden en el que figuran en la lista.
- 7 Haga clic en Aceptar para cerrar el cuadro de diálogo Añadir tipo.
- 8 Seleccione en la lista un elemento de menú y utilice los botones Desplazar hacia arriba o Desplazar hacia abajo de la ficha Elementos de menú para cambiar el orden del tipo en el menú Proyecto.
- 9 Haga clic en Aceptar para cerrar el cuadro de diálogo.

El nuevo tipo aparece ahora en el menú Proyecto con los demás tipos de generación de grupos de proyectos.

Recopilación automática de paquetes fuente

Si se activa la recopilación automática de paquetes fuente, todos los paquetes de la vía de fuentes del proyecto se muestran automáticamente en el panel del proyecto. Durante la generación con esta opción activada, cualquier paquete que contenga archivos que se puedan generar se genera automáticamente y se copia conjuntamente con los recursos en la vía de salida del proyecto. Por ejemplo, si un proyecto contiene archivos Java y SQLJ, los archivos Java se compilan y SQLJ se ejecuta con todos los archivos SQLJ. Por defecto, JBuilder considera que los recursos son archivos de imagen, sonido y propiedades. Los recursos se pueden definir en archivos individuales y por extensión de archivos en todo el proyecto. Para más información sobre recursos consulte [“Copia selectiva de los recursos” en la página 6-42](#). Si desea más información acerca de las vías de salida de un proyecto, consulte [“Configuración de la vía de salida” en la página 5-10](#).



La recopilación automática de paquetes fuente está activada por defecto y se encuentra en la ficha General del cuadro de diálogo Propiedades de proyecto. Para cambiar la configuración, seleccione Proyecto|Propiedades de proyecto|General. Active o desactive la opción Activar la localización y compilación de paquetes fuente. También es posible controlar el número de niveles de paquetes que se muestran en el panel del proyecto modificando el valor en el campo Niveles jerárquicos mostrados. Para más información acerca de esta característica, pulse el botón Ayuda en la ficha General (Proyecto|Propiedades de proyecto).

Para reducir al mínimo el número de nodos de paquete de la lista, JBuilder muestra automáticamente un subconjunto de los paquetes del proyecto. Aunque es posible que algunos de los paquetes fuente no aparezcan en los primeros niveles del proyecto, JBuilder los genera.

Importante



Después de añadir nuevos archivos fuente al proyecto, seleccione el botón Actualizar en la barra de herramientas del proyecto para actualizar la lista de recopilación automática de paquetes fuente.

También se muestra un nodo <Fuente del proyecto> en la parte superior del panel del proyecto cuando está activada la función Recopilación automática de paquetes fuente. Este nodo contiene todos los paquetes y archivos fuente

del proyecto. Los archivos fuente del paquete por defecto también se muestran en este nodo. Sin embargo, si se añaden paquetes y archivos de forma manual, no aparecen en el nodo <Fuente del proyecto>. Para obtener más información sobre los paquetes, consulte [“Filtrado de paquetes” en la página 6-39](#).

Nota Si en su proyecto hay tipos de archivos que JBuilder no reconoce, puede añadirlos como archivos de recursos genéricos. Si desea más información, consulte [“Adición de tipos de archivos no reconocidos como archivos de recursos genéricos” en la página 6-44](#).

Para obtener más información sobre los paquetes Java, consulte el "Chapter 7: Packages" en la Especificación del lenguaje Java en http://java.sun.com/docs/books/jls/second_edition/html/packages.doc.html#60384.

Consulte

- [“Cómo construye JBuilder las vías de acceso” en la página 4-9](#)
- Ficha General del cuadro de diálogo Propiedades de proyecto

Filtrado de paquetes

Es una función de JBuilder Developer y Enterprise.

JBuilder cuenta con una característica de filtrado que permite excluir paquetes del proceso de generación. Sin embargo, si el verificador de dependencias de JBuilder determina que existe alguna dependencia entre las clases de los paquetes excluidos, esas clases se compilan. Cuando se excluyen paquetes de un proyecto, aparece la carpeta Filtros de paquetes en el panel del proyecto. Esta carpeta contiene una descripción general de los filtros que se aplican a los paquetes del proyecto. Los iconos de la carpeta indican qué filtro se ha utilizado. Si desea ver definiciones de este tema, consulte [Tabla 6.4, “Iconos del filtrado de paquetes” en la página 6-41](#).

Importante La función recopilación automática de paquetes fuente, que es la opción por defecto, debe activarse en la ficha General de Propiedades de proyecto (Proyecto|Propiedades de proyecto|General). Si cuenta con paquetes anidados profundamente y sólo ha excluido unos pocos, es recomendable que aumente el valor del campo Niveles jerárquicos mostrados para que puedan mostrarse más paquetes en el panel del proyecto. Para obtener más información sobre la recopilación automática de paquetes fuente, consulte [“Recopilación automática de paquetes fuente” en la página 6-38](#).

Los archivos añadidos manualmente no se pueden excluir mediante el comando Aplicar filtro. Para excluirlos del proceso de generación es necesario eliminarlos del proyecto. En resumen, los archivos que aparecen por encima de la carpeta Filtros de paquetes no se filtran, y se incluyen en el proceso de generación.

Exclusión de paquetes

Para excluir paquetes del proceso de generación:

- 1** Seleccione en el panel del proyecto los paquetes que desee excluir.
- 2** Pulse con el botón derecho del ratón y seleccione Aplicar filtro o, bien, Proyecto|Aplicar filtro.
- 3** En el submenú, elija uno de los siguientes comandos de menú:
 - Excluir paquete y subpaquetes: excluye los paquetes seleccionados y sus subpaquetes.
 - Excluir paquete: excluye el paquete seleccionado, pero no sus subpaquetes.

Nota Si se excluyen el paquete y los subpaquetes, el paquete no aparece en la parte superior del panel del proyecto. Solamente aparecerá en la carpeta Filtros de paquetes. Si se excluye un paquete, pero se incluye un subpaquete, aparece con el icono correspondiente en la parte superior del panel del proyecto y en la carpeta Filtros de paquetes.

Si desea excluir todos los paquetes fuente del proyecto:

- 1** Pulse con el botón derecho del ratón en el nodo <Fuente del proyecto> del panel del proyecto, y seleccione Aplicar filtro.
- 2** En el submenú, seleccione Excluir paquete y subpaquetes.

En algunos casos, el proceso de filtrado puede componerse de dos pasos. Por ejemplo, si desea excluir todos los paquetes excepto algunos subpaquetes, debe realizar lo siguiente:

- 1** Pulse con el botón derecho del ratón en el nodo <Fuente del proyecto> del panel del proyecto, y seleccione Proyecto|Aplicar filtro.
- 2** En el submenú, seleccione Excluir paquete y subpaquetes.
- 3** Abra la carpeta Filtros de paquetes.
- 4** De la lista desplegable, seleccione los subpaquetes que desee incluir.
- 5** Pulse con el botón derecho y seleccione Aplicar filtro|Incluir paquete.

La configuración del filtro se guarda de forma local en el archivo del proyecto. Cualquier nueva configuración del filtro que se aplique a un paquete, sustituye a la configuración anterior para ese paquete.





Inclusión de paquetes

Una vez que se han excluido los paquetes, se pueden volver a incluir en el proceso de generación:

- Seleccione Proyecto|Eliminar todos los filtros.
- Pulse con el botón derecho del ratón en la carpeta Filtros de paquetes y seleccione Eliminar todos los filtros.

- Ampliación de la carpeta Filtros de paquetes. Pulse con el botón derecho del ratón sobre uno o varios paquetes y seleccione Aplicar filtro. En el submenú, elija uno de los siguientes comandos de menú:
 - Incluir paquete y subpaquetes: incluye los paquetes seleccionados junto con sus subpaquetes.
 - Incluir paquete: incluye el paquete seleccionado, pero no los subpaquetes.

Tabla 6.4 Iconos del filtrado de paquetes

Icono	Comando menú	Descripción
	Excluir paquetes y subpaquetes	Excluye del proceso de generación los paquetes seleccionados y todos sus subpaquetes.
	Incluir paquete y subpaquetes	Incluye en el proceso de generación los paquetes seleccionados y todos sus subpaquetes.
	Excluir paquete	Excluye del proceso de generación solamente los paquetes seleccionados, pero no excluye los subpaquetes.
	Incluir paquete	Incluye en el proceso de generación solamente los paquetes seleccionados, pero no incluye los subpaquetes.

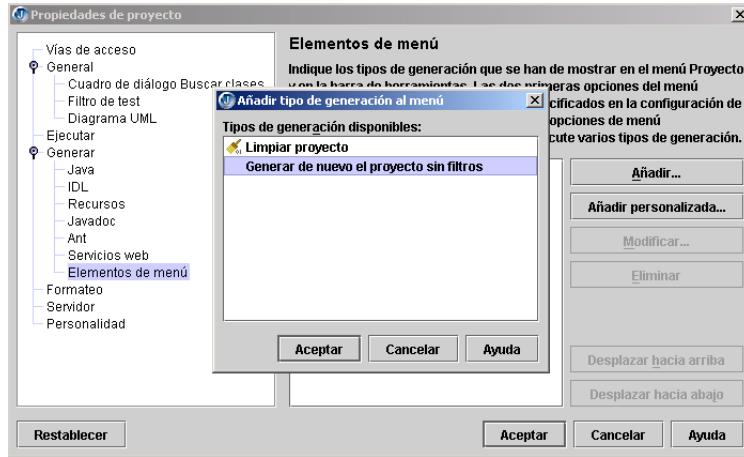
Generar de nuevo sin filtros

En ocasiones puede ser necesario omitir los filtros y generar el proyecto completo. Por ejemplo, se puede modificar código en un paquete filtrado y volver a generarlo mientras se omiten los filtros aplicados. El tipo Generar de nuevo el proyecto sin filtros permite llevar esto a cabo sin eliminar los filtros.

Si desea generar de nuevo su proyecto sin filtros, puede añadir el tipo Generar de nuevo el proyecto sin filtros al menú Proyecto del siguiente modo:

- 1 Seleccione Proyecto|Propiedades de proyecto|Generar|Elementos de menú.
- 2 Pulse el botón Añadir.

- 3 Elija Generar de nuevo el proyecto sin filtros para añadirlo al menú Proyecto.



- 4 Pulse dos veces sobre Aceptar para cerrar los cuadros de diálogo.
- 5 Seleccione Proyecto|Generar de nuevo sin filtros para volver a generar el proyecto sin el filtrado de paquetes.

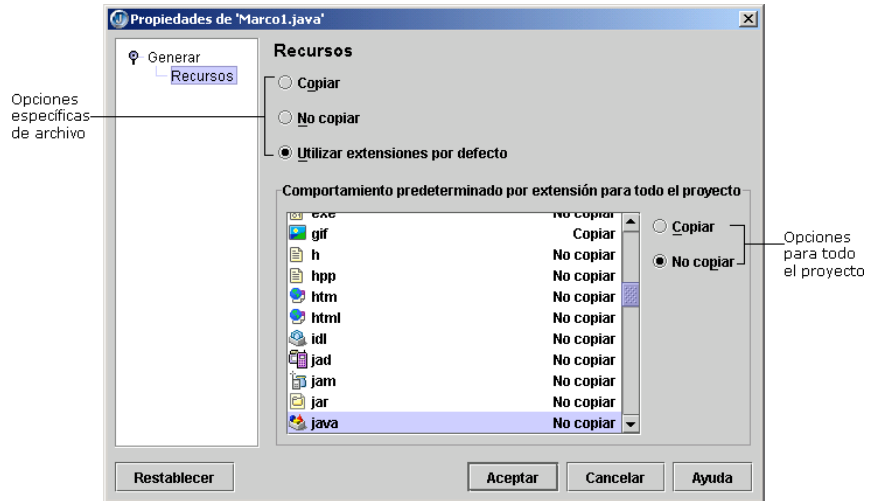
Copia selectiva de los recursos

Es una función de JBuilder Developer y Enterprise.

JBuilder copia todos los tipos de recursos conocidos de la vía de acceso a archivos fuente del proyecto a la vía de salida durante el proceso de compilación. Por defecto, JBuilder reconoce todos los archivos de imagen, sonido y propiedades como recursos, y los copia en la vía de salida. Estas definiciones de recursos por defecto se pueden modificar en archivos individuales o en todo el proyecto, a partir de las extensiones de archivo. Si desea más información adicional sobre la vía de salida, consulte [“Configuración de la vía de salida” en la página 5-10](#).

Propiedades de recursos individuales

Para cambiar la opción por defecto para archivos individuales en un proyecto, haga clic con el botón derecho en los archivos seleccionados en el panel del proyecto, seleccione Propiedades y elija las opciones que desee.

Figura 6.1 Ficha Recursos del cuadro de diálogo Propiedades de proyecto

Opciones específicas de archivos

Los tres botones superiores de radio son opciones específicas de los archivos que controlan los archivos seleccionados. Estas opciones son:

- Copiar: copia el archivo o archivos seleccionados en la vía de salida.
- No copiar: no copia el archivo o archivos seleccionados en la vía de salida.
- Utilizar extensiones por defecto: utiliza las extensiones por defecto tal y como aparecen en la lista Comportamiento predeterminado por extensión para todo el proyecto.

Las opciones Copiar y No copiar tienen un comportamiento absoluto: copiar siempre o no copiar nunca en la vía de salida cuando se genera el proyecto, sin tener en cuenta de si normalmente se considera o no al tipo de archivo como un recurso.

La tercera opción, Usar extensiones de archivo por defecto, permite que JBuilder distribuya el archivo según la extensión definida en la lista inferior. Éste es el comportamiento por defecto de todos los archivos, tanto si están recién creados como si pertenecen a proyectos anteriores. Las extensiones correctas de los archivos seleccionados se eligen automáticamente en la lista para resaltar el comportamiento por defecto.

Importante

Si los archivos o extensiones seleccionados no comparten la configuración, no se selecciona **ninguno** de los botones de radio del grupo correspondiente. Cuando se selecciona uno de los botones de radio se cambia todo al mismo valor, mientras que si no se selecciona ninguno, los valores distintos se respetan.

Si cambia las opciones para archivos individuales y desea que vuelvan a ser las opciones por defecto del proyecto, seleccione de nuevo los archivos y elija la opción Utilizar extensiones por defecto.

Opciones específicas del proyecto

A continuación de las tres opciones específicas de los archivos, se encuentra una lista de todos los comportamientos predeterminados por extensión y su comportamiento de distribución por defecto. Estas opciones por defecto pueden cambiar según el proyecto. Seleccione una o varias extensiones y elija un botón de radio de la derecha para cambiar el comportamiento por defecto de esas extensiones en el proyecto actual. Entre las opciones específicas del proyecto se incluyen:

- Copiar: copia el archivo o archivos seleccionados en la vía de salida.
- No copiar: no copia el archivo o archivos seleccionados en la vía de salida.

Utilice el botón Restablecer para que todos los archivos de la lista de extensión de archivos vuelvan al estado en el que se encontraban cuando se mostró, inicialmente, el cuadro de diálogo. Recuerde que esta acción no devuelve a la configuración predeterminada los parámetros individuales ajustados en los archivos.

También se pueden configurar estas opciones para los proyectos futuros en el cuadro de diálogo Propiedades por defecto para proyectos (Project Properties).

Adición de tipos de archivos no reconocidos como archivos de recursos genéricos

Si cuenta con tipos de archivos que no reconoce JBuilder, puede definirlos como archivos de recursos genéricos. De este modo, JBuilder los reconoce y los incluye en el proceso de localización automática de fuentes. También podrá agruparlos en recopilatorios. Por ejemplo, JBuilder no reconoce los archivos Flash, pero es posible que resulte necesario tener acceso a ellos en su proyecto y distribuirlos en un recopilatorio junto con el resto del proyecto. Si desea agrupar un tipo de archivo no reconocido en un recopilatorio, debe seguir un proceso de dos pasos. En primer lugar, es necesario que lo añada como un tipo de archivo reconocido. A continuación, debe configurarlo para que se copie en la vía de salida cuando se genere el recopilatorio.

- 1 Para añadir tipos de archivos no reconocidos a la lista de tipos reconocidos para JBuilder, lleve a cabo estos pasos:
 - a Seleccione Herramientas|Preferencias|Visualizador|Tipos de archivo.
 - b Seleccione Archivo de recursos genéricos en la lista Tipos de archivo reconocidos.
 - c Pulse el botón Añadir, escriba la extensión del nuevo archivo y, a continuación, pulse Aceptar.
 - d Pulse Aceptar para cerrar el cuadro de diálogo Opciones del IDE.

Importante

Por defecto, todos los tipos de archivo que se añadan como archivos de recursos genéricos **no** se incluyen en los recopilatorios. Es necesario configurar el nuevo tipo de archivo para que se copie en la vía de salida de la ficha Recursos en Propiedades de proyecto (Project Properties).

Propiedades de proyecto|Generar|Recursos). Pase al paso siguiente para llevarlo a cabo.

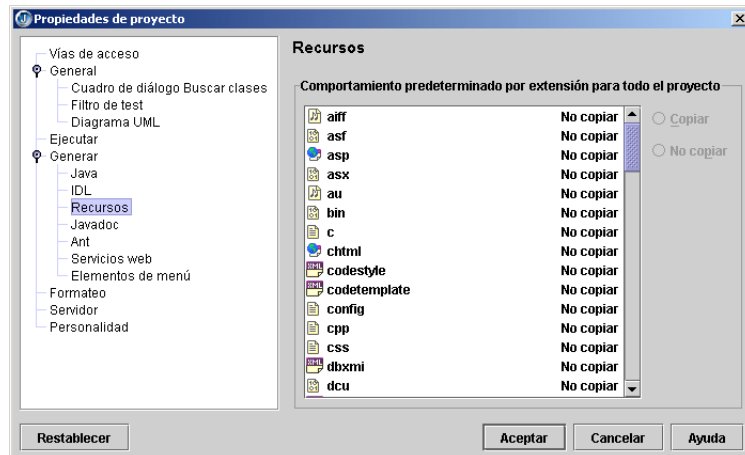
2 Para incluir el nuevo tipo de archivo en su recopilatorio, siga estos pasos:

- a** Seleccione Proyecto|Propiedades de proyecto|Generar|Recurso. Para incluir el nuevo tipo de archivo para todos los proyectos venideros, seleccione Proyecto|Propiedades por defecto para proyectos.
- b** Seleccione el nuevo tipo de archivo en la lista Tipos de archivo reconocidos. Observe que, por defecto, se activa No copiar.
- c** Seleccione el botón circular Copiar para hacer que este tipo de archivo se copie en la vía de salida.
- d** Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

La ficha Recursos del cuadro de diálogo Propiedades de proyecto

Es una función de
JBuilder Developer y
Enterprise.

El cuadro de diálogo Propiedades de proyecto cuenta con la correspondiente ficha Recursos (Proyecto|Propiedades de proyecto|Generar|Recursos), que ofrece un control sobre los comportamientos por defecto de las extensiones de archivo del proyecto completo, en lugar de hacerlo siguiendo un criterio de archivo individual. Pulse el botón Restablecer si desea que todos los archivos de la lista de extensiones de archivo vuelvan al estado en el que se encontraban cuando se abrió el cuadro de diálogo. Si desea ver una descripción de las opciones, consulte [“Opciones específicas del proyecto” en la página 6-44](#).





Ejecución de programas en JBuilder

Cuando esté listo para ejecutar el programa, sólo tiene que ejecutarlo, ejecutarlo y depurarlo al mismo tiempo u optimizarlo, si cuenta con un optimizador compatible. Cuando se ejecuta el programa, JBuilder utiliza la vía de acceso a clases con el fin de localizar todas las clases que emplea el programa. En este capítulo se van a utilizar los términos "ejecutar" y "ejecución" para incluir la ejecución, depuración y optimización, ya que para estas tres funciones están disponibles las mismas configuraciones.

Se pueden ejecutar archivos ejecutables o proyectos completos, incluido OpenTools. Los archivos ejecutables son archivos que contienen un punto de entrada al programa. Por ejemplo, un archivo de aplicación necesita que una clase principal sea ejecutable, y un archivo de applet necesita tener un método `init()` o una etiqueta `<applet>`.

Si se tiene una configuración de ejecución o si se ha establecido una configuración de ejecución por defecto, se puede ejecutar el proyecto con ella pulsando *F9* o el botón Ejecutar. Se puede depurar pulsando *Mayús+F9* o el botón Depurar. (Consulte la documentación de Optimizeit acerca del uso del botón Optimizar).

Si hay más de una configuración de ejecución (JBuilder Developer y Enterprise), la lista de configuraciones se encuentra disponible en varias ubicaciones:

- Seleccione Ejecutar|<Nombre del archivo>. Las configuraciones de ejecución se recogen en el submenú.

- Haga clic en las flechas desplegables junto a los botones Ejecutar, Depurar y Optimizar de la barra de herramientas. Las configuraciones de ejecución se recogen en un menú desplegable.
- Haga clic con el botón derecho del ratón en el nodo de un archivo ejecutable en el panel del proyecto. Aparecen varias configuraciones en el submenú Ejecutar.
- Haga clic con el botón derecho del ratón en la pestaña de nombre de archivo de un archivo ejecutable en el panel de contenido. Aparecen varias configuraciones en el submenú Ejecutar.

A estos menús se hace referencia de forma colectiva como *menús de ejecución*.

El menú de ejecución, en todas sus ubicaciones, se puede configurar con conjuntos de parámetros de ejecución predefinidos, incluidos los tipos de generación, parámetros de MV y parámetros de aplicaciones. Estas configuraciones de ejecución se guardan como selecciones del menú de ejecución. Debe tener creada al menos una configuración de ejecución para poder ejecutar un proyecto mediante *F9* o el botón Ejecutar de la barra de herramientas; si no es así, aparece el cuadro de diálogo Configuraciones de ejecución para poder definir una. Si tiene varias configuraciones, puede seleccionar una de ellas para utilizarla como configuración por defecto en la ejecución. (Las distintas configuraciones de ejecución son una característica de las ediciones Developer y Enterprise de JBuilder).

Sugerencia

Los errores que se produzcan durante la compilación se muestran en el panel de mensajes de la parte inferior de la ventana. Si se producen errores, la compilación se detiene para que pueda solucionarlos e intentarlo de nuevo. Seleccione un error en el panel de mensajes o en la carpeta **Errores** del panel de estructura para resaltar el código en el panel fuente. Si necesita ayuda con un mensaje de error, seleccione el mensaje en el panel y pulse *F1* (Ayuda).

Consulte

- [“Ejecución de archivos” en la página 7-6](#) si quiere saber más sobre la ejecución de archivos.
- [“Definición de las configuraciones de ejecución” en la página 7-8.](#)
- [“Cómo construye JBuilder las vías de acceso” en la página 4-9](#) si quiere saber cómo ubica JBuilder los archivos para ejecutar el programa.
- [“Localización de los archivos” en la página 4-12.](#)
- [Capítulo 8, “Depuración de programas en Java”](#) si desea más información sobre la ejecución y depuración al mismo tiempo.
- [Capítulo 6, “Generación de programas en Java”](#) si desea más información sobre mensajes de error.
- [Capítulo 5, “Compilación de programas en Java”](#) si desea más información sobre mensajes de error.

Ejecución de proyectos

JBuilder puede ejecutar proyectos de forma individual o como parte de un grupo de proyectos. JBuilder también puede ejecutar proyectos de OpenTools de forma eficaz. Cada uno de estos modos cuenta con requisitos ligeramente diferentes durante la configuración.

Ejecución de proyectos individuales

JBuilder ejecuta el proyecto mediante el punto de entrada al programa (clase principal) que se especifique en el cuadro de diálogo Configuraciones de ejecución. Si no hay ninguna clase principal definida, JBuilder solicita que se especifique una.

Para ejecutar un proyecto con una clase principal externa o con una clase principal interna predefinida:

- Desde el teclado, pulse *F9*.
- Desde la barra de herramientas, haga clic en el icono Ejecutar el proyecto.



De esta forma se ejecuta la clase principal seleccionada en la configuración de ejecución por defecto. Si no se elige ningún valor por defecto y sólo hay una configuración, se utiliza de forma predeterminada. Si se tienen varias configuraciones y no hay ninguna por defecto, JBuilder solicita que se elija una configuración de ejecución.

Sugerencia

Si desea realizar tareas de generación de vez en cuando, sin tener que crear para ello una configuración de ejecución independiente, haga clic con el botón derecho del ratón en el proyecto en el panel del proyecto y seleccione una tarea de generación, como Limpiar, Ejecutar Make o Generar de nuevo.

Defina configuraciones de ejecución en el cuadro de diálogo Configuraciones de ejecución seleccionando Ejecutar|Configuraciones en el menú principal o, bien, Configuraciones en el menú de ejecución. También se puede crear una configuración de ejecución por defecto en el último paso de algunos asistentes, como los de aplicaciones, applets y servlets y tests.

Si aún no se ha seleccionado una clase principal, se muestra el cuadro de diálogo Modificar configuración de ejecución para seleccionarla. El Asistente para aplicaciones define automáticamente la clase principal en la configuración de ejecución.

El proyecto se compila y, si no se producen errores, se ejecuta. La barra de estado muestra el avance de la generación y el panel de mensajes muestra los errores del compilador. Si el proyecto se compila correctamente, se muestra la línea de comandos de Java en el panel de mensajes y se ejecuta el proyecto.

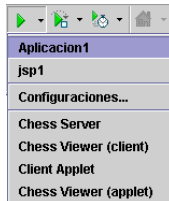
Consulte

- [“Definición de las configuraciones de ejecución” en la página 7-8](#) para obtener más información sobre el cuadro de diálogo Configuraciones de ejecución y la gestión de esas configuraciones a cargo de JBuilder.
- Pulse el botón Ayuda del cuadro de diálogo Configuraciones de ejecución.
- [“Ejecución de tests” en la página 14-14.](#)
- [Capítulo 6, “Generación de programas en Java”](#) para más información sobre tipos y tareas de generación.

Ejecución de proyectos agrupados

Es una función de JBuilder Enterprise.

Un grupo de proyectos es una herramienta de organización. Entre otras cosas, permite ejecutar un proyecto mientras se trabaja en otro. Las configuraciones aplicables a proyectos inactivos del grupo aparecen en menús de ejecución desplegados bajo la opción Configuraciones:



Sin cambiar el proyecto activo, se puede ejecutar cualquier otro proyecto del grupo si se selecciona para él una configuración de ejecución en la parte del grupo del menú de ejecución.

Nota El menú Ejecutar y los botones Ejecutar/Depurar/Optimizar proyecto sólo ejecutan el proyecto activo, no cualquier otro miembro del grupo de proyectos.

Puede especificar qué configuraciones de ejecución del grupo de proyectos se han de añadir al menú desplegable Ejecutar al crear o modificar cada configuración de ejecución de los proyectos del grupo. Para ello, mire al cuadro de diálogo Configuraciones de ejecución bajo la columna Grupo. Active o desactive la opción Grupo para cada configuración del proyecto activo.

Para que aparezcan las configuraciones de ejecución de cada proyecto en el grupo de proyectos:

- 1 Abra el grupo de proyectos en el panel del proyecto.
- 2 Pulse con el botón derecho del ratón un proyecto del grupo y seleccione Propiedades.
- 3 Abra la pestaña Ejecutar del cuadro de diálogo Propiedades de proyecto y, a continuación, seleccione la configuración de ejecución que desee.
(Consulte [“Definición de las configuraciones de ejecución” en la página 7-8](#) si desea obtener más información sobre la creación y modificación de las configuraciones de ejecución).

- 4 Asegúrese de que se ha marcado el cuadro Grupo para esa configuración. (Grupo está marcado por defecto).
- 5 Repita esto para todas las demás configuraciones de este u otro proyecto del grupo con las configuraciones que desee que estén disponibles en la lista desplegable Ejecutar.
- 6 Pulse Aceptar.

Ahora, si pulsa la flecha hacia abajo junto al icono Ejecutar proyecto de la barra de herramientas, además de las configuraciones de ejecución del proyecto actual, se ven todas las demás configuraciones de ejecución de los demás proyectos del grupo, tal y como se especifica en la figura de arriba.

Consulte

- [Capítulo 3, “Los grupos de proyectos”](#)

Ejecución de OpenTools

JBuilder cuenta con un tipo de configuración de ejecución OpenTool que permite ejecutar, depurar y optimizar su proyecto OpenTool en JBuilder como cualquier otro proyecto. Ya no tiene que cerrar JBuilder, crear un archivo JAR, copiarlo en el directorio `<JBuilder home>\lib\ext` o `<JBuilder home>\patch` ni reiniciar JBuilder.

Al ejecutar un proyecto OpenTool mediante el ejecutor OpenTool, se genera un archivo de configuración temporal en el directorio `<outpath>\..\config-temp`, y se inicia una nueva instancia de JBuilder mediante ese archivo de configuración temporal. Este archivo se elimina cuando se cierra el rastreador.

Para crear un ejecutor de OpenTool:

- 1 Abra su proyecto OpenTool en JBuilder.
- 2 Seleccione Ejecutar!Configuraciones o, bien, haga clic en la flecha desplegable del botón Ejecutar proyecto y seleccione Configuraciones.
Se abre el cuadro de diálogo Configuraciones de ejecución.
- 3 Pulse Nuevo para abrir el cuadro de diálogo Nueva configuración de ejecución.
- 4 Escriba un nombre para la configuración.
- 5 Seleccione un Tipo de generación para este ejecutor.
- 6 Indique la ubicación de los archivos de clase que se deben utilizar durante la ejecución: los que se encuentran en el directorio del proyecto OpenTool, o bien, los que están en el archivo JAR.
- 7 Especifique los parámetros que OpenTool debe pasar a la MV y a JBuilder durante la ejecución.

Sugerencia

Estos parámetros se pueden crear mediante Macros.

- 8 Seleccione Redefinir las clases y recursos si desea que las clases y recursos OpenTool sean los primeros en la vía de acceso a clases.
- 9 Pulse el botón Aceptar para volver al cuadro de diálogo Configuraciones de ejecución.
- 10 Marque cualquiera de las casillas que se aplican a esta configuración (Por defecto, Menú contextual o Grupo), y utilice los botones Desplazar hacia arriba o Desplazar hacia abajo para desplazarse a la ubicación que prefiera en la lista. Este es el mismo orden en que las configuraciones aparecen en los menús y listas desplegables de JBuilder.
- 11 Pulse Aceptar cuando haya terminado.

Si necesita ayuda para definir estas configuraciones de ejecución, pulse el botón de la ayuda de la parte inferior del cuadro de diálogo Propiedades de ejecución.

Ejecución de archivos

En JBuilder se pueden ejecutar archivos .java, archivos web, archivos JAR ejecutables e incluso archivos J2ME en JBuilder. Todos estos archivos deben cumplir determinados criterios para ser ejecutables. JBuilder automatiza y acelera el proceso de determinar y ajustar estos criterios, y permite ejecutar el archivo rápidamente.

Ejecución de archivos .java

Se pueden ejecutar archivos .java ejecutables desde el panel del proyecto, el panel de contenido o la barra de menús principal. Los archivos ejecutables son archivos que contienen un punto de entrada al programa. Para un archivo de aplicación, esto significa que necesita una clase principal para ser ejecutable. Es necesario guardar un archivo para poder ejecutarlo. Antes de ejecutar un archivo, compruebe los mensajes de error de la carpeta Errores del panel de estructura.

Para ejecutar un archivo .java guardado y ejecutable, utilice cualquiera de estas técnicas:

- En el panel del proyecto, haga clic con el botón derecho en el nodo de archivo y seleccione Ejecutar.
- En el panel de contenido, haga clic con el botón derecho del ratón en la pestaña del archivo y seleccione Ejecutar.
- En la barra de menús principal, asegúrese de que se trata del archivo activo en el panel de contenido y, a continuación, seleccione Ejecutar | Ejecutar <Nombre de archivo>.

Si no tiene una configuración de ejecución para ese archivo, JBuilder abre el cuadro de diálogo Configuraciones de ejecución para poder crear una. Una vez que se ha hecho y se ha pulsado Aceptar, ejecute de nuevo el comando

Ejecutar para ejecutar el archivo mediante la nueva configuración de ejecución.

Nota Cuando se ejecuta un solo archivo ejecutable, JBuilder utiliza la clase principal de ese archivo. Utiliza cualquier otro parámetro definido en la configuración de ejecución especificada, pero si la configuración de ejecución llama a una clase principal fuera del archivo ejecutable, se sustituye en su lugar la clase principal del archivo ejecutable. Ejecute archivos ejecutables desde cualquiera de las siguientes opciones:

- El nodo de archivo en el panel del proyecto
- La pestaña Nombre de archivo en el panel de contenido

La aplicación se compila y, si no se producen errores, se ejecuta. La barra de estado muestra el avance de la generación y el panel de mensajes muestra los errores del compilador. Si el archivo se compila correctamente, se muestra la línea de comandos de Java en el panel de mensajes, y el archivo se ejecuta.

Consulte

- [“Definición de las configuraciones de ejecución” en la página 7-8](#) para obtener más información sobre el cuadro de diálogo Configuraciones de ejecución y la gestión de esas configuraciones a cargo de JBuilder.
- Pulse el botón Ayuda del cuadro de diálogo Configuraciones de ejecución.
- [“Ejecución de proyectos” en la página 7-3](#)

Ejecución de archivos Web

Las Applets son una función de todas las ediciones de JBuilder.

(Las distintas configuraciones de ejecución son una característica de las ediciones Developer y Enterprise de JBuilder).

Con JBuilder también se pueden ejecutar archivos Web, tales como JSP, servlet, SHTML y HTML por medio de un servidor Web para obtener una vista dinámica de la aplicación Web.

- Para ejecutar un applet, haga clic con el botón derecho del ratón en el archivo HTML que contiene la etiqueta `<applet>` y elija Ejecutar.
- Para ejecutar archivos JSP, SHTML, y HTML, pulse el botón derecho sobre el archivo en el panel de proyecto y seleccione Ejecutar web.
- En el caso de los servlets, primero debe configurar la opción de menú Activar Ejecutar/Depurar/Optimizar en el menú contextual.

Para activar esta opción:

- a** Haga clic con el botón derecho del ratón sobre el archivo servlet en el panel del proyecto y seleccione Propiedades.
- b** Seleccione la pestaña Ejecutar web.
- c** Active la opción Activar Ejecutar/Depurar/Optimizar web en el menú contextual.
- d** Tras hacer esto, pulse el archivo servlet con el botón derecho y seleccione Ejecutar web.

Sugerencia

Si se crean los servlets con el Asistente para servlets de JBuilder, esta opción se activa automáticamente.

Consulte

- "Uso de applets" en la *Guía del desarrollador de aplicaciones web*
- "Las aplicaciones web en JBuilder" en *Guía del desarrollador de aplicaciones web*
- "Los servlets en JBuilder" en *Guía del desarrollador de aplicaciones web*
- JSP (Páginas JavaServer) en *Guía del desarrollador de aplicaciones web*

Definición de las configuraciones de ejecución

En la versión de JBuilder Foundation solamente hay una configuración para la ejecución.

(Las distintas configuraciones de ejecución son una característica de las ediciones Developer y Enterprise de JBuilder.)

Las configuraciones de ejecución son parámetros de ejecución, vías de acceso y tareas de generación predefinidos. La utilización de configuraciones predefinidas ahorra tiempo al ejecutar y al depurar. Estas configuraciones preestablecidas permiten, cada vez que ejecute o depure la aplicación, simplemente seleccionar la configuración deseada.

Las configuraciones de ejecución se pueden crear y gestionar en el cuadro de diálogo Configuraciones de ejecución, que está disponible en Ejecutar|Configuraciones o desde la flecha desplegable que se encuentra junto a los iconos Ejecutar, Depurar y Optimizar de la barra de herramientas principal. Se pueden añadir, modificar, copiar o eliminar configuraciones, y controlar también el orden que utilizan los menús para mostrarlas. También se pueden especificar las configuraciones que aparecen en los menús contextuales de un proyecto o grupo de proyectos en concreto.

Se puede designar una de las configuraciones como configuración por defecto para utilizarla al seleccionar Ejecutar, Depurar u Optimizar sin seleccionar una configuración. Las configuraciones de ejecución individuales también se llaman *ejecutores*.

Abra el cuadro de diálogo Configuraciones de ejecución de cualquiera de los siguientes modos:

- Seleccione Proyecto|Propiedades de proyecto|Ejecutar.
- Seleccione Ejecutar|Configuraciones.
- Haga clic en la flecha desplegable junto a los botones Ejecutar, Depurar u Optimizar y seleccione Configuraciones.
- Haga clic con el botón derecho del ratón en el archivo del proyecto, en el panel del proyecto, y seleccione Propiedades|Ejecutar.

Algunos de los asistentes de JBuilder, como los asistentes para aplicaciones, applets, servlets o tests, le ofrecen la opción de crear una configuración para la ejecución para aquellos archivos que crean. Las configuraciones de ejecución que crean también están disponibles para el proyecto.

JBuilder selecciona la configuración de ejecución para utilizar según estos criterios:

- Si se especificó una configuración por defecto, JBuilder la abre en fuente resaltada en las listas desplegables o submenús Ejecutar, Depurar u Optimizar y la utiliza al ejecutar o depurar el proyecto.

- Si no se especificó una configuración por defecto, JBuilder solicita que seleccione la configuración que se va a utilizar en la operación actual.
 - Si solamente cuenta con una configuración para la ejecución (ya sea o no además la configuración por defecto), JBuilder utiliza esa configuración cuando se selecciona Ejecutar|Ejecutar el proyecto,(F9), o Ejecutar|Depurar proyecto (*Mayús-F9*).
 - Si no cuenta con ninguna configuración para el tipo de archivo que se está ejecutando, cuando:
 - se selecciona en el menú Ejecutar archivo o Ejecutar proyecto,
 - se pulsa F9, o
 - se hace clic en el botón Ejecutar en la barra de herramientas,
 se abre el cuadro de diálogo Configuraciones de ejecución, para poder crear una configuración en ese momento. Después de crear una configuración, ejecute de nuevo el comando Ejecutar, Depurar u Optimizar para utilizarla.
- Sin embargo, si
- hace clic con el botón derecho del ratón sobre el archivo de ejecución en el panel del proyecto y selecciona Ejecutar,
- JBuilder lo ejecuta automáticamente según las opciones por defecto que supone que son adecuadas para ese tipo de archivo.

Tabla 7.1 Resumen de las condiciones que determinan el uso de las configuraciones de ejecución a cargo de JBuilder

¿Configuración de ejecución?	Si es así, ¿cuántas?	¿Por defecto especificada?	Lo que hace JBuilder:	Cómo se ejecuta:
No			Aparece el cuadro de diálogo Configuraciones de ejecución, para poder crear una configuración de ejecución.	Una vez creada la configuración, ejecute de nuevo el comando Ejecutar para utilizarla.
Sí	1		JBuilder se ejecuta mediante una configuración de ejecución, esté o no seleccionada por defecto.	Se ejecuta automáticamente.

Tabla 7.1 Resumen de las condiciones que determinan el uso de las configuraciones de ejecución a cargo de JBuilder

¿Configuración de ejecución?	Si es así, ¿cuántas?	¿Por defecto especificada?	Lo que hace JBuilder:	Cómo se ejecuta:
Sí	> 1	No	<p>Si se abre</p> <ul style="list-style-type: none"> Desde <i>F9</i> o desde el icono Ejecutar proyecto, JBuilder abre el cuadro de diálogo Seleccionar configuración de ejecución. Desde los menús de ejecución de la barra de menús principal o de las pestañas de archivo, JBuilder abre un submenú de configuraciones de ejecución disponibles. Haciendo clic con el botón derecho en el nodo de un archivo ejecutable en el panel del proyecto, JBuilder la ejecuta mediante opciones por defecto que infiere para ese tipo de archivo. <p>Nota: al seleccionar una configuración mediante estos métodos no se convierte en la configuración por defecto.</p>	<p>Después de seleccionar en el submenú y hacer clic en Aceptar en el cuadro de diálogo, ejecuta el archivo o el proyecto automáticamente.</p> <p>Después de hacer clic con el botón derecho en el archivo en el panel del proyecto, ejecuta el archivo automáticamente basándose en el tipo de archivo.</p>
Sí	> 1	Sí	JBuilder se ejecuta mediante la configuración de ejecución por defecto.	Se ejecuta automáticamente.

Nota Cuando se ejecuta un solo archivo ejecutable, JBuilder utiliza la clase principal de ese archivo. Utiliza cualquier otro parámetro definido en la configuración de ejecución especificada, pero si la configuración de ejecución llama a una clase principal fuera del archivo ejecutable, se sustituye en su lugar la clase principal del archivo ejecutable. Los archivos ejecutables se ejecutan solos desde el nodo de archivo en el panel del proyecto y desde la pestaña de nombre de archivo en el panel de contenido.

Nota La mayoría de los tipos de ejecutores requieren una clase principal:

- Si al ejecutar un proyecto todavía no se ha seleccionado una clase principal, aparece el cuadro de diálogo Modificar configuración de ejecución, para que pueda seleccionar una clase principal.
- Si el archivo se ha creado con el Asistente para aplicaciones, la clase main se selecciona automáticamente.

Gestión de las configuraciones de ejecución

El cuadro de diálogo Configuraciones de ejecución ofrece funciones de personalización que permiten aprovechar al máximo estas configuraciones.



La lista de configuraciones tiene las opciones Por defecto, Menú contextual y Grupo para cada configuración:

- Marque la casilla Por defecto para especificar la configuración de ejecución que se utiliza como ejecutor por defecto. Sólo se puede seleccionar una opción por defecto en cada momento.
- Marque la casilla Menú contextual para cada configuración que desee que aparezca en los menús desplegables y submenús Ejecutar, Depurar y Optimizar.
- Pulse el botón Grupo si desea que la configuración esté disponible para el grupo de proyectos al que pertenece este proyecto.

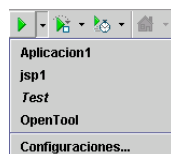
Pulse el botón Nuevo para crear una nueva configuración de ejecución.

Pulse el botón Copiar para crear una copia de una configuración. De este modo, si se necesita una configuración parecida a una ya creada, se pueden duplicar todas las opciones y cambiar sólo las que sean necesarias, en lugar de configurar de nuevo todas las opciones.

Pulse Modificar para cambiar las opciones de un ejecutor ya creado.

Las configuraciones se recogen en el orden en que aparecen en los menús. Desplácelas hacia arriba o abajo de la lista mediante los botones Desplazar arriba y Desplazar abajo.

Para seleccionar una configuración durante la ejecución, seleccione Ejecutar! Ejecutar archivo, Ejecutar! Ejecutar proyecto o, bien, haga clic en la flecha desplegable junto a los iconos Ejecutar, Depurar u Optimizar de la barra de herramientas principal. Seleccione una configuración en el menú.



- Sugerencia** Utilice un nombre para cada configuración que sea evidente cuando se vea en este menú.
- Si se producen errores, excepciones en tiempo de ejecución u otro mal comportamiento del programa, es posible que desee depurar el programa conforme se ejecuta, para detectar problemas.

Consulte

- [Capítulo 8, “Depuración de programas en Java”](#)

Creación y modificación de configuraciones de ejecución

Se pueden crear o modificar configuraciones de ejecución haciendo clic en Nuevo, Copiar o Modificar en el cuadro de diálogo Configuraciones de ejecución. Si se hace así, se abre un cuadro de diálogo con un campo para el nombre de la configuración, los tipos de generación que se utilizan y el tipo de ejecutor que se usa. Una vez seleccionado el tipo de ejecutor, el resto de opciones reflejan lo necesario para ejecutar ese tipo.

Los cuadros de diálogo Crear configuración de ejecución y Modificar configuración de ejecución admiten macros de la línea de comandos. Escribálos en los campos de vía de acceso y parámetros para reducir así lo que es necesario escribir.

Consulte

- [“Utilización de macros de la línea de comandos” en la página A-2](#)

Selección de tipos de generación

Para cada configuración de ejecución que se crea o modifica, se puede especificar el tipo de generación que se ejecuta. JBuilder proporciona un conjunto de tipos estándar entre los que elegir, y según el proyecto, se añaden otros a la lista. A continuación se recogen los elementos que aparecen en la lista Tipos de generación:

- **<Ninguno>**
Se ejecuta sin compilar en primer lugar.
- **Limpiar**
Limpiar elimina todos los archivos de salida de los otros tipos, como el directorio `classes`, los archivos JAR, WAR, etc. Elimina la salida adecuada al nodo seleccionado.
- **Ejecutar Make**
Establece dependencias entre las fases autónomas por el siguiente orden: Precompilar, Compilar, Postcompilar, Empaquetar y Distribuir.
- **Generar de nuevo**
Tiene como dependencias Limpiar y Ejecutar Make. Si se han definido filtros de paquetes, no genera los nodos filtrados.

- **Generar de nuevo sin filtros**

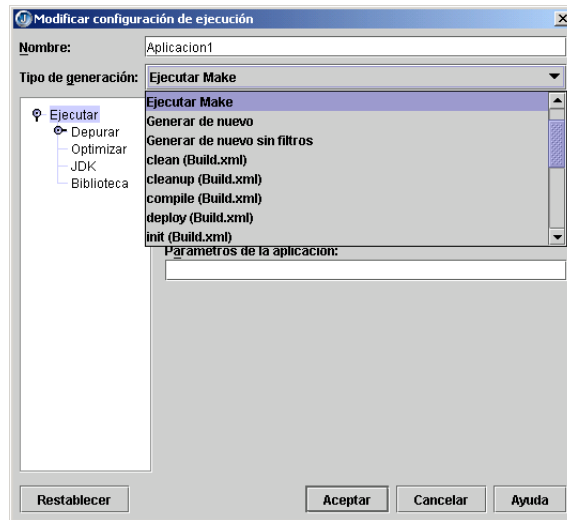
Vuelve a generar el proyecto completo, incluidos los paquetes filtrados. Esto proporciona un modo automático de redefinir filtros temporalmente sin tener que eliminarlos ni restablecerlos.

- **Tareas externas de generación** (en las ediciones Developer y Enterprise de JBuilder)

Si se ha creado una tarea externa de generación, aparece en la lista de tipos disponibles.

- **Tipos de generación de archivos Ant**

Si hay archivos de generación Ant en el proyecto, los tipos dentro del archivo Ant se añaden a la lista de tipos disponibles.



Además, si ha ampliado su sistema de generación con la ayuda de OpenTools, en la lista aparece también cualquiera de esos tipos de tareas de generación.

Consulte

- [Capítulo 6, “Generación de programas en Java”](#)
- [Capítulo 5, “Compilación de programas en Java”](#)
- [“Creación de tareas externas de generación” en la página 6-30](#)
- [“Generación con archivos Ant” en la página 6-12](#)

Tipos de configuraciones de ejecución

Los tipos de configuraciones de ejecución disponibles varían según la versión de JBuilder.

Cada tipo de configuración de ejecución requiere un conjunto diferente de propiedades de ejecución. Donde se llame a una clase principal, se debe especificar una. En el lugar donde se introducen los parámetros, se pueden utilizar macros de la línea de comandos en esos campos.

■ Aplicación

Crea un ejecutor para una aplicación Java. Especifique el archivo de clase que contiene el método `main()` que desea que utilice este ejecutor para la aplicación. La clase principal debe residir en el proyecto activo. Escriba los parámetros que desea pasar a la MV y a la aplicación durante la ejecución.

■ Applet

Crea una configuración de ejecución para un applet. Primero, realice una de las siguientes operaciones:

- Seleccione Clase principal y haga clic sobre el botón puntos suspensivos para buscar la clase que contenga el método `init()`. Esta opción ejecuta el applet en AppletTestbed, el visualizador de applets de JBuilder.
- Seleccione el archivo HTML y haga clic sobre el botón puntos suspensivos para buscar el archivo HTML de applet que contiene la etiqueta `<applet>`. La opción HTML ejecuta el applet en el **appletviewer** de Sun.

A continuación, especifique Parámetros de MV, Anchura, Altura y Parámetros del applet.

■ Servidor

Crea una configuración de ejecución para una aplicación web. Defina los parámetros del servidor y especifique el servlet o JSP que se ejecuta.

■ Test

Crea una configuración para un Test o Conjunto de tests. Elija la clase principal de la clase del conjunto de pruebas, configure los parámetros de la MV y especifique el ejecutor de test deseado.

■ OpenTool

Crea una configuración de ejecución para ejecutar, depurar y optimizar un proyecto de OpenTool, directamente en JBuilder.

■ JAR ejecutable (JBuilder Developer y Enterprise)

Crea una configuración para ejecutar de forma eficaz un archivo JAR. JBuilder ejecuta el ejecutable especificado en el archivo descriptor JAR, sin generar archivos extra ni obligar a realizar pasos adicionales.

■ MIDlet (Ediciones Developer y Enterprise de JBuilder)

Crea una configuración de ejecución para un MIDlet J2ME.

■ Aplicación i-mode (JBuilder Developer y Enterprise)

Crea una configuración de ejecución para una aplicación i-mode, que utiliza el JDK DoJa.

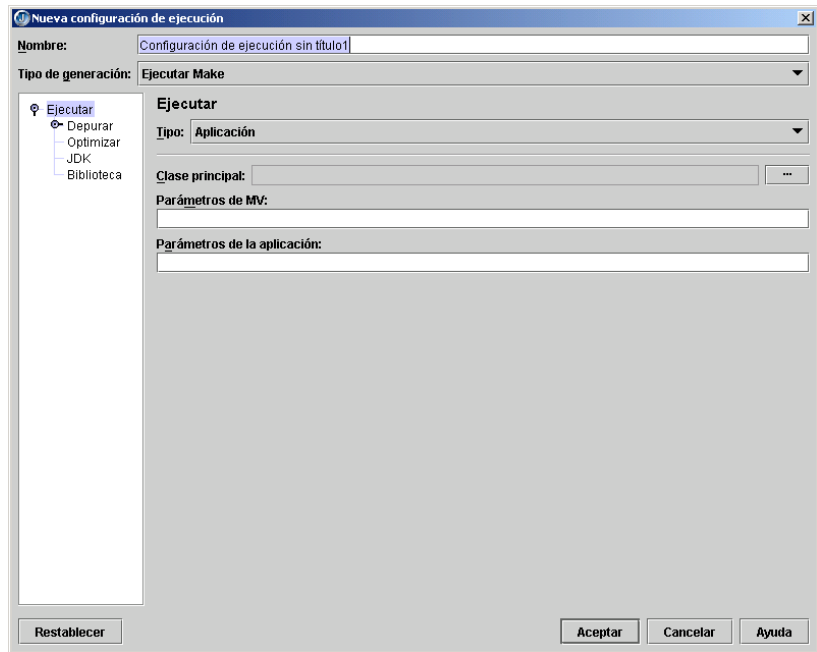
Consulte

- “Creación de una configuración de ejecución para un applet” en *Guía del desarrollador de aplicaciones web*
- “Las aplicaciones web en JBuilder” *Guía del desarrollador de aplicaciones web* para obtener más información sobre los parámetros de ejecución del servidor
- [Capítulo 14, “Test de módulos”](#)
- [“Ejecución de OpenTools” en la página 7-5](#)
- [“Utilización del Creador de recopilatorios” en la página 16-16](#) si desea obtener más información sobre el uso de un ejecutor de JAR ejecutable
- “Creación de aplicaciones MIDP” en *Desarrollo de aplicaciones para dispositivos móviles para MIDP*
- Generar, ejecutar y depurar aplicaciones i-mode en *Desarrollo de aplicaciones para dispositivos móviles para i-mode*

Creación de configuraciones de ejecución

Para crear configuraciones de ejecución, utilice el botón Nuevo del cuadro de diálogo Configuraciones de ejecución. Para abrir el cuadro de diálogo Configuraciones de ejecución, seleccione Ejecutar|Configuraciones o Proyecto|Propiedades de proyecto|Ejecutar.

- 1 Pulse Nuevo para abrir el cuadro de diálogo Nueva configuración de ejecución.



- 2 Escriba un nombre para la configuración si desea modificar el nombre que aparece por defecto. Este nombre se añade a las listas desplegables y submenús de Ejecutar, Depurar y Optimizar.
- 3 Seleccione un Tipo de generación para esta configuración de la lista desplegable.
- 4 Seleccione el tipo adecuado de configuración de ejecución para el proyecto. (Los tipos de configuraciones de ejecución son mutuamente excluyentes: sólo se puede utilizar un tipo por cada configuración de ejecución).
Cada tipo de configuración de ejecución cuenta con opciones específicas para ese tipo.
- 5 Seleccione Depurar en el panel de desplazamiento y defina las opciones de depuración.
- 6 Seleccione Optimizar, si se ha instalado Optimizeit, y defina estas opciones.
- 7 Si desea que esta configuración de ejecución utilice opciones de JDK diferentes de las de Propiedades de proyecto, seleccione la ficha JDK y especifique el JDK de destino.
- 8 Si desea que esta configuración de ejecución utilice opciones de bibliotecas diferentes de las de Propiedades de proyecto, seleccione la

ficha Biblioteca y especifique las bibliotecas y proyectos que se deben utilizar.

- 9 Pulse el botón Aceptar para volver al cuadro de diálogo Configuraciones de ejecución.
- 10 Realice los cambios adicionales que desee en el cuadro de diálogo Configuraciones de ejecución.
- 11 Pulse Aceptar cuando haya terminado.

Modificación de configuraciones de ejecución

Se puede modificar todo lo relacionado con una configuración de ejecución existente, menos su tipo. Si desea un tipo de ejecutor diferente, debe crear una nueva.

Para modificar una configuración de ejecución ya creada:

- 1 Seleccione Ejecutar!Configuraciones o, bien, pulse la flecha hacia abajo junto al botón Ejecutar, Depurar u Optimizar de la barra de herramientas y seleccione Configuraciones.

Se abre el cuadro de diálogo Configuraciones de ejecución.

- 2 Seleccione una configuración de ejecución y haga clic en Modificar o, bien, haga doble clic en el nombre de la configuración.

Se abre el cuadro de diálogo Modificar configuración de ejecución. Se parece mucho al cuadro de diálogo Nueva configuración de ejecución, excepto en que el campo Tipo es de sólo lectura.

- 3 Realice los cambios que desee en la configuración en el cuadro de diálogo Modificar configuraciones de ejecución. Las opciones de configuración disponibles dependen del tipo de configuración que se modifique.
- 4 De forma optativa, seleccione las fichas Depurar, Optimizar, JDK o Biblioteca en el árbol de desplazamiento y modifique las opciones que desee en esas fichas.
- 5 Pulse Aceptar cuando haya terminado.
- 6 Realice los cambios adicionales que desee en el cuadro de diálogo Configuraciones de ejecución.
- 7 Pulse Aceptar cuando haya terminado.

Ejecución de programas desde la línea de comandos

Para ejecutar programas fuera de JBuilder, todas las bibliotecas que necesita el programa se deben poner en la vía de acceso a clases o añadirse al argumento **-classpath** del comando `java`. Por ejemplo:

```
java -classpath /<jbuilder>/lib/dbswing.jar
/home>/jbproject/mypackage/classes/mypackage.application1
```

En el ejemplo:

- `<jbuilder>` es el nombre del directorio JBuilder
- `<raíz>` = el directorio inicial, como por ejemplo, `c:\winnt\profiles\<nombre de usuario>`

Ejecución de programas distribuidos desde la línea de comandos

Tras la distribución del programa por medio del Creador de recopilatorios o la herramienta **jar**, es posible ejecutar el archivo JAR desde la línea de comandos.

Nota El atributo `Main-Class` en el archivo descriptor JAR debe contener el nombre de la clase principal.

1 Abra la ventana de línea de comandos.

Sugerencia En Windows, utilice barras invertidas (\) con todos los comandos mencionados.

2 Introduzca el siguiente comando en una línea del indicador desde cualquier ubicación:

```
java -classpath nombre-de-paquete.nombre-de-clase-principal
```

Nota El directorio `<jdk>/bin/` debe estar en la vía de acceso. `<jdk>` representa el nombre del directorio de JDK.

Por ejemplo, el comando podría tener el siguiente aspecto:

```
java -classpath <home>/jbproject/hello/classes/HelloWorld.jar  
hello.HelloWorldClass
```

En este ejemplo, `<home>` representa el directorio inicial, como por ejemplo, `c:\winnt\profiles\<nombre de usuario>`.

Puede utilizar también la opción **-jar**:

```
java -jar HolaATodos.jar
```

Nota Primero, debe cambiarse al directorio que contiene el archivo `.jar` para poder ejecutar este comando con la opción **-jar**.

Consulte

- [Capítulo 6, “Generación de programas en Java”](#)
- [Capítulo 5, “Compilación de programas en Java”](#)
- [Capítulo 16, “Distribución de programas en Java”](#)
- El tutorial JAR en <http://java.sun.com/docs/books/tutorial/jar/index.html>
- [Apéndice A, “Las herramientas de línea de comandos”](#)



Depuración de programas en Java

La depuración es el proceso de búsqueda y solución de errores en el programa. El depurador integrado de JBuilder permite depurar dentro del entorno de JBuilder. El menú Ejecutar permite acceder a numerosas funciones del depurador. También se puede acceder a las funciones de depuración por medio de los menús contextuales del editor y el depurador.

Cuando el depurador interrumpe el programa es posible comprobar los valores actuales de los elementos de datos del programa. La modificación de los valores de los datos durante una sesión de depuración proporciona una manera de comprobar posibles soluciones a los errores durante la ejecución. Si encuentra una modificación que soluciona un error de programa, puede salir de la sesión de depuración, implementar la solución necesaria en el código del programa y recompilar para que la solución sea permanente. Si está depurando con JDK 1.4 o superior, no necesita salir de la sesión de depuración para que la solución de errores tenga efecto. Para más información consulte [“Modificación del código durante la depuración” en la página 8-76](#).

Si desea un tutorial sobre la depuración, consulte el [Capítulo 20, “Tutorial: Compilación, ejecución y depuración”](#).

Existe información adicional y sugerencias sobre estos temas relacionados con la depuración:

- Si el programa utiliza un componente `JDataStore`, consulte el capítulo “Resolución de problemas” de la *Guía del desarrollador de JDataStore*.

- Si está depurando una aplicación distribuida, consulte el [Capítulo 9](#), “[Depuración remota](#)”. (Ediciones Developer y Enterprise de JBuilder).
- Si va a depurar un test de módulos, consulte el [Capítulo 14](#), “[Test de módulos](#)”.

Para efectuar la depuración fuera de JBuilder, utilice la herramienta **jdb** del directorio `<jdk> /bin`. Consulte la documentación JDK en <http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html> si desea obtener más información sobre esta herramienta.

Tipos de errores

El depurador puede ayudar a encontrar errores de ejecución y errores lógicos. Si encuentra o sospecha que existe un error de ejecución o un error lógico en el programa, se puede comenzar la sesión de depuración ejecutando el programa bajo el control del depurador.

Errores de ejecución

Si su programa contiene sentencias válidas, pero producen errores cuando se están ejecutando, se trataría de un error en el tiempo de ejecución. Por ejemplo, el programa podría estar intentando abrir un archivo que no existe o dividir un número por cero.

Sin un depurador, los errores de ejecución pueden ser difíciles de localizar, porque el compilador no proporciona información sobre ellos. A menudo, la única pista de la que se dispone es el resultado de la ejecución, como por ejemplo el aspecto de la pantalla y el mensaje de error generado.

Aunque puede encontrar los errores de ejecución buscándolos en el código fuente del programa, el depurador le ayudará a hacerlo más rápidamente. El depurador, permite ejecutar un programa hasta una ubicación determinada. Desde ahí, puede comenzar la ejecución del programa una sentencia tras otra, observando el comportamiento del programa en cada paso. Cuando se ejecuta la sentencia que hace fallar al programa, se detecta el error. A partir de ahí, puede solucionar los problemas del código fuente y reanudar el test.

Si el programa produce una excepción de ejecución, imprimirá un seguimiento de la pila en la Vista Salida, entrada y errores de consola. Puede pulsar sobre el nombre del archivo subrayado y el número de línea en el seguimiento de la pila para desplazarse a la línea de código en el archivo fuente que aparece en dicha pila. (Esta función pertenece a las ediciones Developer y Enterprise de JBuilder).

Errores lógicos

Los errores lógicos son errores en el diseño o la implementación del programa. Las sentencias del programa son sintácticamente correctas (realizan operaciones), pero las acciones que éstas realizan no son exactamente las que el programador tenía en mente. Por ejemplo, los errores lógicos pueden producirse cuando las variables contienen valores incorrectos, cuando los gráficos no son los adecuados o cuando los datos producidos por el programa son incorrectos.

Los errores lógicos son con frecuencia los más difíciles de encontrar porque pueden aparecer en los lugares más insospechados. Para asegurarse de que el programa funciona como estaba previsto en su diseño, es necesario comprobar exhaustivamente todos sus aspectos. Sólo si se comprueban con detalle todos los elementos de la interfaz de usuario y el resultado del programa, puede tener garantías de que su comportamiento se ajusta a lo esperado. Al igual que con los errores de ejecución, el depurador ayuda a encontrar errores lógicos al permitirle hacer un seguimiento de los valores de las variables del programa y de los objetos de datos durante la ejecución.

Descripción general del proceso de depuración

Después de la fase de diseño del programa, el desarrollo de éste se basa en ciclos de codificación y depuración. Sólo después de una comprobación exhaustiva del programa, se puede pensar en distribuirlo. Para asegurarse de que comprueba todos los aspectos del programa, es mejor tener un plan minucioso de comprobación y depuración.

Un buen método de depuración incluye la división del programa en secciones diferentes que puedan depurarse sistemáticamente. Mediante el seguimiento cuidadoso de las sentencias de cada una de las secciones del programa, es posible comprobar que todas las áreas del programa funcionan como se espera. Si encuentra un error de programación, puede corregir el problema en el código fuente, recompilar el programa y reanudar la comprobación a continuación.

JBuilder Enterprise permite depurar código fuente distinto de Java, incluidas la Páginas JavaServer (JSP). (Con JBuilder Developer sólo se puede depurar el código JSP, no todos los tipos de código fuente ajeno a Java). Puede establecer un punto de interrupción en un archivo fuente distinto de Java y depurar ese archivo de forma local o remota. También puede alternar entre la visión del código fuente distinto de Java y la del código Java generado. Si desea más información, consulte [“Depuración de código fuente ajeno a Java” en la página 8-32](#).

Nota Es posible depurar con un JDK que admita la API de depuración Java Process Debugging Architecture (JPDA -Arquitectura de depuración de procesos de Java). Por lo general, se depura con la versión de JDK que acompaña a JBuilder. (Esta es la JDK por defecto que se selecciona para el

proyecto, si no se ha definido y seleccionado ninguna otra). Si se va a utilizar JDK 1.2.2, es necesario descargar la JPDA desde el sitio web de Sun.

Creación de una configuración de ejecución

Antes de ejecutar o depurar, necesita crear una configuración para la ejecución. Esta configuración está compuesta por un conjunto de parámetros preconfigurados. La utilización de parámetros preestablecidos ahorra mucho tiempo durante la ejecución y la depuración, porque así sólo se definen una vez. Estas configuraciones preestablecidas permiten, cada vez que ejecute o depure la aplicación, simplemente seleccionar la configuración deseada. Se establecen las opciones del depurador, tales como los parámetros del Paso inteligente o las opciones de depuración remota, a través de la configuración para la ejecución.

Para crear y gestionar configuraciones, ha de utilizar el cuadro de diálogo Configuración de ejecución. Para obtener más información acerca de capturas, consulte [“Definición de las configuraciones de ejecución” en la página 7-8](#). Si desea más información sobre opciones del depurador, consulte [“Configuración de las opciones de depuración” en la página 8-81](#). (Las distintas configuraciones de ejecución son una característica de las ediciones Developer y Enterprise de JBuilder).

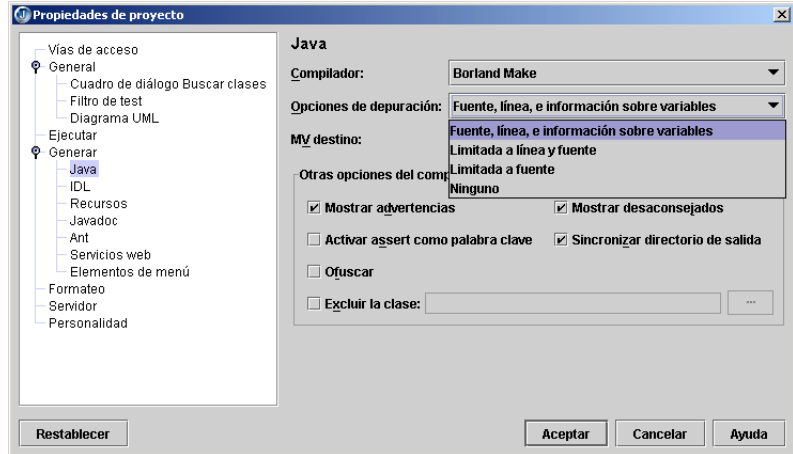
Compilación del proyecto con información simbólica de depuración

La eliminación de la información de depuración es una característica las ediciones Developer y Enterprise de JBuilder.

Antes de comenzar una sesión de depuración, es necesario compilar el proyecto con información simbólica de depuración. La información simbólica de depuración permite al depurador hacer conexiones entre el código fuente del programa y el bytecode de Java que genera el compilador.

Por defecto, JBuilder incluye información simbólica de depuración cuando se compila. Para comprobar que esta opción se encuentra activada en el proyecto actual:

- 1 Seleccione Proyecto|Propiedades de proyecto|Generar|Java para abrir la ficha Generar Java del cuadro de diálogo Propiedades de proyecto. El nodo Generar|Java tiene un aspecto parecido a la siguiente imagen:



- 2 Seleccione una de las siguientes opciones en la lista desplegable Opciones de depuración:
 - Fuente, línea, e información sobre variables: incluye información de depuración simbólica con el nombre del archivo fuente, número de línea e información de la variable local en el archivo `.class` cuando se compila, crea o reconstruye un nodo.
 - Limitada a línea y fuente: incluye información de depuración simbólica sólo con el nombre del archivo fuente y el número de línea en el archivo `.class` cuando se compila, crea o reconstruye un nodo.
 - Limitada a fuente: incluye información de depuración simbólica en el archivo `.class` cuando se compila, crea o reconstruye un nodo.
 - Ninguno: no se incluye ninguna información de depuración. De todos modos, con esta opción aún se puede realizar la operación de depuración; el objeto `this` aún se encuentra disponible para ello. Si se selecciona esta opción, se reduce el tamaño de la clase al mínimo posible.

Sugerencia Para configurar esta opción para todos los proyectos nuevos, seleccione Proyecto|Propiedades por defecto para proyectos|Generar|Java. El cambio de valores de propiedades por defecto no altera la configuración de proyectos creados anteriormente.

Nota No podrá ver la información variable en las clases API de Java, ya que éstas se compilaban únicamente con información de línea y código fuente. No obstante, sí puede inspeccionar su código. Para obtener información sobre el

modo de seguimiento de las clases, consulte [“Determinación de las clases que se han de inspeccionar” en la página 8-43](#).


Cuando se genera información simbólica de depuración, el compilador almacena esta información en el archivo `.class` asociado. Esto puede agrandar considerablemente el archivo `.class` si se compila sin la información de depuración. Puede ser mejor desactivar esta opción antes de la distribución.

Para que la compilación se realice de forma automática antes de la depuración, asigne a Tipo de generación, en la parte superior del cuadro de diálogo Configuración de ejecución nuevo/edición (cuando se asignan los valores de las opciones de depuración para la configuración de ejecución) el valor Make. Esta opción compila automáticamente el proyecto antes de ejecutarlo bajo el control del depurador. Si esta opción se configura como `<Ninguno>`, JBuilder no compila el programa antes de depurarlo, con lo que es posible que los archivos fuente y clase no estén sincronizados. Si desea más información sobre el archivo JAD, consulte [“Selección de tipos de generación” en la página 7-12](#).

Inicio del Depurador

Una vez haya creado una configuración para la ejecución y haya compilado el proyecto con información de depuración puede iniciar la depuración mediante una de las siguientes opciones de menú. Si desea más información sobre las configuraciones de ejecución, consulte [“Definición de las configuraciones de ejecución” en la página 7-8](#).

Tabla 8.1 Comandos de menú para iniciar el depurador

Comando	Atajo	Descripción
Ejecutar Depurar proyecto	<i>Mayús + F9</i> o 	Inicia el programa en el depurador utilizando la configuración por defecto o la seleccionada. Se suspenderá la ejecución en un punto de interrupción o en la primera línea de código en que se requiera que el usuario introduzca algún dato que siempre tendrá prioridad.
Ejecutar Omitir inspección	<i>F8</i>	Interrumpe la ejecución en la primera línea de código ejecutable.
Ejecutar Inspeccionar	<i>F7</i>	Interrumpe la ejecución en la primera línea de código ejecutable.

Para depurar:

- Un archivo de clases del proyecto ejecutable, y no el proyecto en su totalidad, seleccione el archivo fuente en el panel del proyecto y haga clic con el botón derecho. Seleccione el comando Depurar para obtener la configuración deseada. También puede hacer clic en la pestaña Archivo del editor.
- Una aplicación web, haga clic con el botón derecho del ratón sobre el servlet o el archivo JSP y seleccione Depurar web para obtener la configuración deseada. Para obtener más información, consulte

"Depuración web del servlet o de la JSP" en el capítulo "Aplicaciones web" de la *Guía del desarrollador de aplicaciones web* *Guía del desarrollador de aplicaciones Web*. (El desarrollo de aplicaciones web es una característica de las ediciones Developer y Enterprise de JBuilder).

- Un test, pulse con el botón derecho del ratón en ese test del panel del proyecto y, a continuación, seleccione Depurar test. Para obtener más información sobre el test de módulos, consulte [Capítulo 14, "Test de módulos"](#).

Cada vez que se inicia el depurador comienza una sesión de depuración. Si desea más información, consulte ["Sesiones de depuración" en la página 8-9](#).

Nota



Si desea seleccionar una configuración para la ejecución de una sesión de depuración, haga clic en la flecha abajo, que se encuentra a la derecha del icono Depurar proyecto de la barra de herramientas principal antes de empezar. Si no selecciona una configuración concreta, se utiliza la configuración por defecto definida en la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto. (Las distintas configuraciones de ejecución son una característica de las ediciones Developer y Enterprise de JBuilder).

Inicio del depurador con la opción -classic

En las ediciones de la MV de Java anteriores a la 1.3.1, la opción `-classic` mejoró el rendimiento del depurador. Esta opción no se aplica a las MV más recientes; por ejemplo, JDK 1.4x y JDK 1.3.1 de Solaris no necesitan utilizar la opción `-classic`. En estas ediciones, la MV utiliza la "depuración a toda velocidad", que supone una mejora en el rendimiento.

Si está utilizando una Máquina virtual de Java inferior a la versión 1.3.1 para el proyecto, la opción Depurar siempre con -Classic del cuadro de diálogo Configurar JDK (Herramientas\Configurar JDK) proporciona un mejor rendimiento. JBuilder lo comprueba automáticamente para ver si esta opción aumenta el desarrollo, a continuación selecciona o elimina la selección de esta casilla de acuerdo con lo que proporcione mejores resultados. Esta función está disponible en todas las ediciones de JBuilder.

Cuando se efectúa la evaluación, JBuilder lleva a cabo dos comprobaciones:

- 1 ¿Tiene instalada la MV Classic?
- 2 Si es así, ¿la versión de JVM es anterior a la 1.3.1?

Esta selección se modifica cuando se definen parámetros de MV como `native`, `hotspot`, `green` y `server`.

Ejecución bajo el control del depurador

Cuando se ejecuta un programa bajo el control del depurador, su comportamiento es el habitual: crea ventanas, acepta información del usuario, calcula valores y presenta resultados. El depurador detiene el programa, de forma que se pueden utilizar las vistas del depurador para examinar su estado actual. Visualizando los valores de las variables, los métodos de la pila de

llamadas y el resultado del programa, es posible cerciorarse de que la parte de código que se está examinando funciona según lo deseado.

Conforme se ejecuta el programa bajo el control del depurador, puede observar el comportamiento de la aplicación en las ventanas que se crean. Coloque las ventanas de manera que vea simultáneamente la ventana del depurador y la de la aplicación durante la depuración. Para impedir que las ventanas parpadeen cuando el foco pasa de las vistas del depurador a la de la aplicación y viceversa, ordene las ventanas de forma que no se solapen.

Pausar la ejecución del programa

Cuando se está utilizando el depurador, el programa puede estar en uno de dos estados posibles: *en ejecución* o *en interrupción*.



- El programa está en modo de ejecución cuando el botón Pausa está disponible en la barra de herramientas del depurador .
- El programa se interrumpe si el botón Pausa no está disponible. Cuando se interrumpe el programa es posible analizar y modificar los valores de los datos. Entonces se habilitan los botones de inspección de la barra de herramientas del depurador. La ejecución también se interrumpe al alcanzar un punto de interrupción o cuando se realiza una inspección.



Para continuar ejecutando el programa, seleccione el botón Reanudar el programa en la barra de herramientas del depurador. Cuando la sesión de depuración finaliza, este botón pasa a ser Reiniciar el programa y reinicia la sesión.

Mientras el programa está interrumpido, puede modificar el código y continuar con la ejecución en todos los marcos activos. Si desea más información, consulte [“Modificación del código durante la depuración” en la página 8-76](#).

Finalización de una sesión de depuración



Para finalizar la sesión de depuración actual y liberar la memoria, seleccione el icono Reanudar el programa.

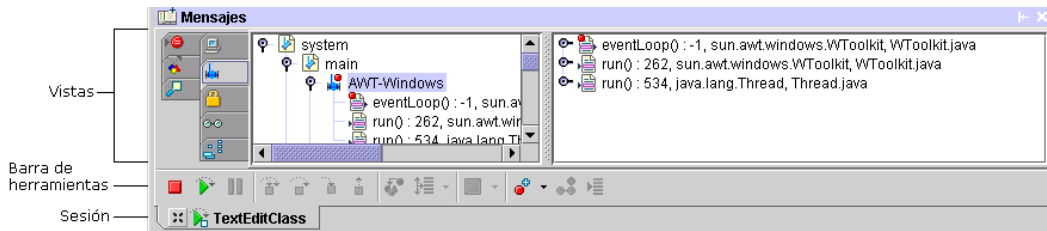
También es posible salir de la aplicación para cerrar la sesión de depuración. Para cerrar la pestaña de la sesión de depuración, haga clic con el botón derecho del ratón sobre la pestaña y seleccione Eliminar o pulse sobre la X de la pestaña.

Interfaz del depurador

Si el proyecto o la clase se compila con éxito, se muestra el depurador en la parte inferior del IDE.

- Las pestañas horizontales de la parte inferior del IDE representan sesiones de depuración. Cada pestaña representa una sesión.
- Las pestañas verticales de la parte inferior izquierda del IDE representan las vistas del depurador. Las vistas corresponden a la sesión de depuración seleccionada actualmente. Las distintas vistas tienen iconos que indican el estado y el tipo del elemento seleccionado.
- La barra de herramientas del depurador corresponde a la sesión de depuración seleccionada actualmente.

Figura 8.1 Interfaz del depurador



Sesiones de depuración

El depurador permite depurar varios procesos en varias sesiones de depuración. Los procesos pueden encontrarse en el mismo proyecto de JBuilder o en diferentes. Es posible depurar simultáneamente un proceso servidor y otro cliente durante una sesión de JBuilder.

Los puntos de observación, de interrupción, y las clases que no tengan desactivada la inspección se almacenan por proyecto. Todos los puntos de observación e interrupción son aplicables a todos los procesos de un proyecto. Los puntos de interrupción pueden desactivarse de forma selectiva en cada configuración de ejecución.

Cuando se utilizan comandos del menú Ejecutar que no sean Ejecutar proyecto, Depurar proyecto ni Configuraciones, se continúa con la sesión de depuración seleccionada. Cuando se utilizan los botones de la barra de herramientas del depurador también se continúa con la sesión seleccionada.



Para finalizar la sesión de depuración actual y liberar la memoria, seleccione el icono Reanudar el programa. También puede finalizar la sesión haciendo clic en X en la pestaña de sesión de depuración o haciendo clic con el botón derecho del ratón en la pestaña y seleccionando Eliminar pestaña. A pesar de que el programa preguntará si desea detener el proceso antes de eliminar la pestaña, resulta conveniente utilizar primero Ejecutar/Terminar el programa.

Vistas del depurador

La vista del depurador permite analizar el interior del programa. Mediante las vistas del depurador es posible examinar y modificar valores de datos, rastrear el programa hacia delante o hacia atrás, examinar los procesos internos de un método y de la llamada a dicho método, así como seguir un hilo dentro del programa.

Las vistas del depurador aparecen a lo largo de la parte izquierda del depurador de la interfaz del usuario. Para seleccionar una vista, elija la pestaña correspondiente en la parte izquierda del depurador. Las vistas (excepto la vista Salida, entrada y errores de la consola) también se pueden mostrar como ventanas flotantes. Las ventanas flotantes permiten ver varias vistas del depurador simultáneamente, en lugar de tener que alternar entre ellas. (Las ventanas flotantes son una característica de las ediciones Developer y Enterprise de JBuilder).

- Para mostrar una vista como ventana flotante, haga clic con el botón derecho sobre una zona vacía de la vista y seleccione Ventana flotante.
- Para cerrar dicha ventana, haga clic sobre el botón Cerrar en la ventana flotante, o haga clic con el botón derecho sobre una zona vacía de la vista y desactive Ventana flotante.
- Si desea restaurar el orden de vista por defecto una vez haya cerrado una ventana flotante, haga clic con el botón derecho sobre una zona vacía de la vista y seleccione Restablecer orden predeterminado.

Las vistas del depurador también tienen menús contextuales. Los comandos de estos menús a menudo duplican aquellos que aparecen en los menús Ejecutar y Ver para controlar el depurador con más facilidad.

Además, cada depurador muestra una gama de iconos que indican el estado del elemento seleccionado. Por ejemplo, un punto de interrupción puede estar desactivado, verificado, sin verificar o no ser válido; cada uno de estos estados se indica de forma visual por medio de un icono de pequeño tamaño situado en el margen izquierdo de la vista.

A continuación se describen los iconos y vistas del depurador.

Tabla 8.2 Vistas del depurador









Pestaña	Ver	Descripción
	Vista Consola	Muestra el resultado y los errores del programa. También permite introducir los datos necesarios. La imagen del icono cambia si hay datos de resultado del programa y si se muestran mensajes de error.
	Vista Hilos, pilas de llamada y datos	Muestra los grupos de hilos del programa. Los grupos de hilos se expanden para mostrar sus hilos y contienen un seguimiento de marcos en la pila representando la secuencia de llamadas actual. Todos los marcos de pila se pueden ampliar y mostrar los elementos de datos disponibles en el ámbito. (Los datos estáticos no se muestran en esta vista, sino en la vista Datos estáticos y clases cargadas).

Tabla 8.2 Vistas del depurador (continuación)

Pestaña	Ver	Descripción
	Vista Monitores de sincronización	Muestra los monitores de sincronización que utilizan los hilos y su estado actual, lo que resulta útil para detectar bloqueos y conflictos. Esta vista sólo se encuentra disponible si la MV actual la admite. (Algunas MV de HotSpot no aceptan esta función). La posibilidad de detectar hilos en conflicto o bloqueados es una característica de las ediciones Developer y Enterprise de JBuilder.
	Vista Puntos de observación de datos	Muestra los valores actuales de los miembros de datos que está siguiendo.
	Vista Clases cargadas y datos estáticos	Muestra las clases cargadas actualmente por el programa. Si una clase tiene datos estáticos, se muestran cuando se amplía la clase. Si se muestra un paquete en el árbol, aparece el número de clases cargadas para ese paquete.
	Vista Puntos de interrupción de datos y código	Muestra todos los puntos de interrupción del archivo y su estado actual. Esta vista también está disponible desde Ejecutar Puntos de interrupción antes de que comente la sesión de depuración.
	Vista Clases con inspección desactivada	Muestra una lista en orden alfabético de las clases y paquetes que no se deben inspeccionar. También se puede acceder a esta vista desde Ejecutar Clases con inspección desactivada antes de que comience la sesión de depuración.
	Vistas personalizadas	Permite añadir, modificar o eliminar un visualizador personalizado para un objeto en concreto. Esta vista también está disponible desde Ejecutar Vistas personalizadas antes de que comience la sesión de depuración. (Ediciones Developer y Enterprise de JBuilder).

Sugerencia

Es posible hacer flotantes todas las vistas del depurador, excepto la de la Consola. Para ello, haga clic con el botón derecho del ratón en el panel de mensajes y elija Ventana flotante. (Esta función pertenece a las ediciones Developer y Enterprise de JBuilder).

Vista Salida, entrada y errores de consola






La vista de salida, entrada y errores de consola muestra las salidas de datos desde el programa y los errores del programa. También permite introducir datos requeridos por el programa. Cuando la pestaña Consola no se encuentra seleccionada, el icono cambia si hay alguna salida del programa o si ha aparecido algún mensaje de error.

Las excepciones producidas durante la ejecución se muestran en esta vista. Para abrir el archivo en el que ha tenido lugar la excepción de ejecución y poder colocar el cursor en el número de línea de la excepción, pulse sobre el nombre del archivo que se encuentra subrayado. (Esta función pertenece a las ediciones Developer y Enterprise de JBuilder).

En esta vista, la salida de error aparece en color rojo. La salida estándar aparece en negro.

En la siguiente tabla aparecen los iconos utilizados en esta vista.

Tabla 8.3 Iconos de la vista Consola

Icono	Descripción
	Se han escrito mensajes de salida en la vista.
	Se han escrito mensajes de error en la vista.
	No hay ninguna salida en la vista o la vista está en primer plano.

En la siguiente tabla aparecen los comandos del menú contextual.

Tabla 8.4 Menú contextual de la vista Consola

Comando	Descripción
Borrar todo	Borra todos los mensajes de la vista.
Copiar todo	Copia al portapapeles el contenido de la vista.
Copiar los seleccionados	Copia el contenido seleccionado al portapapeles.
Ajuste de palabras	Ajusta las líneas largas en la salida.

(El icono es en color)



Vista Clases con inspección desactivada

La vista de clases con inspección desactivada muestra una lista ordenada alfabéticamente con las clases y paquetes que no se utilizan durante la inspección. Esta información está disponible antes de iniciar la depuración desde el comando EjecutarClases con inspección desactivada.

Cuando se inicia una sesión de depuración, la inspección de todas las clases y paquetes que aparecen en la vista está desactivada por defecto, como indica el icono Inspección desactivada. Esto evita que el depurador inspeccione las bibliotecas proporcionadas con JBuilder así como las clases JDK, y que se concentre en el nuevo código en lugar de en el código que ya se ha depurado.

Para obtener información sobre cómo decidir qué clases que deben inspeccionar, consulte [“Determinación de las clases que se han de inspeccionar” en la página 8-43](#).

Nota



En JBuilder Foundation solamente se añaden a esta vista tres clases (`java.lang.Object`, `java.lang.String` y `java.lang.ClassLoader`). No se pueden añadir, modificar ni eliminar elementos de la lista; sin embargo, es posible decidir si se desea o no inspeccionar esas clases. Para más información consulte [“Paso inteligente” en la página 8-40](#).

(El icono es gris)



En la siguiente tabla aparecen los iconos utilizados en esta vista.

Tabla 8.5 Iconos en la vista Clases con inspección desactivada

Icono	Descripción
 (El icono es gris)	La inspección está desactivada para la clase o el paquete seleccionado. La clase o el paquete no se inspeccionan cuando la inspección línea por línea está activada. Esto impide que se inspeccionen las bibliotecas de JBuilder, Java o terceros que ya se hayan comprobado. Éste es el valor por defecto.
 (El icono es en color)	La inspección de la clase o el paquete seleccionado se activa, aunque la inspección línea por línea se encuentre activada.

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al seleccionar un elemento en la vista.

Tabla 8.6 Menú contextual con clase o paquete seleccionado en la vista Clases con inspección desactivada

Comando	Descripción
Modificar clase/paquete	Muestra el cuadro de diálogo Cambiar clase/paquete con inspección desactivada, en el que puede utilizar comodines para modificar la clase o el paquete y para abrir el cuadro de diálogo Seleccionar clase o paquete. Si se selecciona un paquete, no se inspeccionan todas las clases del paquete. (Ediciones Developer y Enterprise de JBuilder).
Inspeccionar clase/paquete	Permite inspeccionar la clase o paquete. Si se selecciona un paquete, se inspeccionan las clases de ese paquete.
Eliminar clase/paquete	Elimina la clase o paquete seleccionado de la vista. Esto permite inspeccionar la clase o paquete seleccionados. (Ediciones Developer y Enterprise de JBuilder).

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al hacer clic con el botón derecho en una zona vacía de la vista.

Tabla 8.7 Menú contextual sin selección en la vista Clases con inspección desactivada

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (Ediciones Developer y Enterprise de JBuilder).
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (Ediciones Developer y Enterprise de JBuilder).

Tabla 8.7 Menú contextual sin selección en la vista Clases con inspección desactivada

Comando	Descripción
Añadir	Muestra el cuadro de diálogo Seleccionar clase o paquete, donde se elige la clase o paquete que se desea añadir a la vista. Esto evita que el depurador inspeccione esa clase o paquete. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (Ediciones Developer y Enterprise de JBuilder).
Eliminar todos	Elimina todas las clases y paquetes de la vista. Se inspeccionarán todos los paquetes y clases, incluidos los de JBuilder y las bibliotecas del JDK. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (Ediciones Developer y Enterprise de JBuilder).

Vista Puntos de interrupción de datos y código



La vista de puntos de interrupción en datos y muestra todos los puntos de interrupción establecidos en el archivo y su estado actual. Esta información también está disponible antes de dar comienzo a la depuración por medio del comando Ejecutar | Puntos de interrupción.

Si desea más información, consulte [“Puntos de interrupción” en la página 8-46](#).

En la siguiente tabla aparecen los iconos utilizados en esta vista.

Tabla 8.8 Iconos de la vista Puntos de interrupción por datos y código

Icono	Descripción
	Un punto de interrupción sin verificar.
	Un punto de interrupción verificado.
	Un punto de interrupción no válido.
	Un punto de interrupción desactivado para una configuración. (Ediciones Developer y Enterprise de JBuilder).
	Un punto de interrupción de campo. Un campo es una variable Java que está definida en un objeto Java. (Ediciones Developer y Enterprise de JBuilder).

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al seleccionar un elemento en la vista.

Tabla 8.9 Menú contextual con punto de interrupción seleccionado en la vista Puntos de interrupción de datos y código

Comando	Descripción
Ir al punto de interrupción	Va al punto de interrupción seleccionado del código fuente. Esto resulta útil si hay varios archivos abiertos en el editor y se desea encontrar la línea de código interrumpida con rapidez. Este comando está disponible cuando hay un punto de interrupción seleccionado.
Activar punto de interrupción	Activa el punto de interrupción seleccionado.
Desactivar en la configuración	Desactiva el punto de interrupción seleccionado en la configuración seleccionada. Este comando sólo está disponible si se cuenta con más de una configuración de ejecución. Por defecto, todos los puntos de interrupción se aplican a todas las configuraciones definidas. Este comando está disponible cuando hay un punto de interrupción seleccionado. (Ediciones Developer y Enterprise de JBuilder).
Eliminar punto de interrupción	Elimina el punto de interrupción seleccionado.
Propiedades de punto de interrupción	Muestra el cuadro de diálogo Propiedades de punto de interrupción, donde se definen las propiedades del punto de interrupción seleccionado.
Interrumpir al leer	Obliga al depurador a detenerse cuando está a punto de leer el punto de interrupción del campo seleccionado. Un campo es una variable Java que está definida en un objeto Java. Este comando está disponible cuando hay un punto de interrupción de campo seleccionado. (Ediciones Developer y Enterprise de JBuilder).
Interrumpir al escribir	Obliga al depurador a detenerse cuando está a punto de escribir en el punto de interrupción del campo seleccionado. Un campo es una variable Java que está definida en un objeto Java. Este comando está disponible cuando hay un punto de interrupción de campo seleccionado. (Ediciones Developer y Enterprise de JBuilder).

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al hacer clic con el botón derecho en una zona vacía de la vista.

Tabla 8.10 Menú contextual sin selección en la vista Puntos de interrupción de datos y código

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (Ediciones Developer y Enterprise de JBuilder).
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista. (Ediciones Developer y Enterprise de JBuilder).

Tabla 8.10 Menú contextual sin selección en la vista Puntos de interrupción de datos y código (continuación)

Comando	Descripción
Añadir punto de interrupción de línea	Muestra el cuadro de diálogo Añadir punto de interrupción de línea, lo que permite añadir un punto de interrupción de línea.
Añadir punto de interrupción por excepción	Muestra el cuadro de diálogo Añadir punto de interrupción por excepción, lo que permite añadir un punto de interrupción por excepción (Ediciones Developer y Enterprise de JBuilder).
Añadir punto de interrupción de clase	Muestra el cuadro de diálogo Añadir punto de interrupción de clase, dónde añade un punto de interrupción de clase (Ediciones Developer y Enterprise de JBuilder).
Añadir punto de interrupción de método	Muestra el cuadro de diálogo Añadir punto de interrupción de clase, dónde añade un punto de interrupción de método (Ediciones Developer y Enterprise de JBuilder).
Añadir punto de interrupción interproceso	Muestra el cuadro de diálogo Añadir punto de interrupción interproceso, lo que permite añadir un punto de interrupción de interproceso (Ediciones Developer y Enterprise de JBuilder).
Desactivar todos	Desactiva todos los puntos de interrupción.
Activar todos	Activa todos los puntos de interrupción.
Eliminar todos	Elimina todos los puntos de interrupción.

Vista Hilos, pilas de llamada y datos



La vista Hilos, pilas de llamada y datos muestra el estado actual de los grupos de hilos del programa. Los grupos de hilos se expanden para mostrar sus hilos y contienen un seguimiento de marcos de pila representando la secuencia actual de llamadas a métodos. Cada marco de pila puede ampliarse para mostrar los elementos de datos disponibles en ámbito. Los iconos identifican el tipo de elemento de datos. (Los datos estáticos no se muestran en esta vista, sino en la vista Datos estáticos y clases cargadas). Se heredan los elementos atenuados.

Por defecto, esta vista se divide en dos paneles. El panel izquierdo se puede ampliar para que muestre los marcos de pila. El panel derecho muestra el contenido del elemento seleccionado a la izquierda e indica todo, desde un grupo de hilos hasta una variable. Por ejemplo, si se selecciona un hilo en el panel de la izquierda, el panel de la derecha mostrará los marcos de pilas de ese hilo. Por otra parte, si se selecciona un marco de pila en el panel de la izquierda, el panel de la derecha mostrará las variables disponibles en esa vista. (El panel de división es una característica de las ediciones Developer y Enterprise de JBuilder).

Si desea más información sobre hilos, consulte [“Gestión de hilos” en la página 8-36](#).

En la siguiente tabla se muestran algunos de los iconos más frecuentes de la vista Hilos. Si desea ver una lista completa de los iconos, consulte la tabla "Iconos del panel de estructura: código fuente", en la sección "El panel de estructura y los iconos UML" de *Introducción a JBuilder*.

En la siguiente tabla aparecen los iconos utilizados en esta vista.

Tabla 8.11 Iconos en la vista Hilos, pilas de llamadas y datos

Icono	Descripción
	Un grupo de hilos
	Un hilo
	Hilo inspeccionado
	Un hilo bloqueado
	Hilo inspeccionado y bloqueado
	Un hilo suspendido
	Hilo interrumpido por el usuario
	Hilo inspeccionado e interrumpido por el usuario
	Un hilo muerto
	Una clase
	Una interfaz
	Un objeto
	Un objeto null
	Punto de ejecución actual
	Punto de ejecución antiguo
	Seguimiento de marcos de pilas
	Una matriz
	Una primitiva
 (Rojo)	Un error
 (Gris)	Un mensaje de información

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al seleccionar un elemento en la vista.

Tabla 8.12 Menú contextual con selección en la vista Hilos, pilas de llamadas y datos

Comando	Descripción
Mantener hilo suspendido	La ejecución del hilo seleccionado no se reanuda cuando se selecciona Ejecutar/Reanudar el programa. Esto le permite observar el comportamiento de otros hilos (Ediciones Developer y Enterprise de JBuilder).
Establecer hilo inspeccionado	El hilo seleccionado se inspecciona al seleccionar Ejecutar/Reanudar el programa. Permite observar el funcionamiento de este hilo (Ediciones Developer y Enterprise de JBuilder).
Establecer punto de ejecución	Establece el marco de pila en el que se llevan a cabo las operaciones de reanudación (Ediciones Developer y Enterprise de JBuilder).
Cortar	Elimina el valor de la variable y la coloca en el portapapeles. Este comando está disponible cuando hay una variable seleccionada (Ediciones Developer y Enterprise de JBuilder).
Copiar	Copia al portapapeles el valor de la variable. Este comando está disponible cuando hay una variable seleccionada (Ediciones Developer y Enterprise de JBuilder).
Pegar	Pega el contenido del portapapeles en otra variable. Cuando se utiliza el comando Pegar, tanto la variable cortada o copiada del objeto como la variable pagada pegada apuntan al mismo objeto. Este comando está disponible cuando hay una variable seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en una variable local	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la variable local seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una variable o una matriz de variables seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la matriz seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una matriz seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en un componente de una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el componente de matriz seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un componente seleccionado en una matriz (Ediciones Developer y Enterprise de JBuilder).

Tabla 8.12 Menú contextual con selección en la vista Hilos, pilas de llamadas y datos

Comando	Descripción
Ajustar rango de visualización	Muestra el cuadro de diálogo Ajustar rango, donde se puede reducir el número de elementos de matriz que se muestran en la vista. Este comando está disponible cuando hay una matriz seleccionada. (Ediciones Developer y Enterprise de JBuilder)
Crear punto de observación 'this'	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto <code>this</code> seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto <code>this</code> seleccionado. (Ediciones Developer y Enterprise de JBuilder)
Crear punto de observación de clases	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la clase seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una clase seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear punto de observación de objetos	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto seleccionado. Un punto de observación de objetos vigila el objeto Java seleccionado. Se amplía para mostrar los miembros de datos de la instanciación actual (Ediciones Developer y Enterprise de JBuilder).
Mostrar toString()	Ejecuta el método <code>toString()</code> en el objeto seleccionado y muestra la cadena resultante. Por ejemplo, si el objeto seleccionado es <code>City</code> , el comando <code>Mostrar toString()</code> mostrará el valor del objeto, como <code>San Francisco</code> en lugar de <code>com.mycode.City@391</code> (Ediciones Developer y Enterprise de JBuilder).
Crear punto de observación de cadenas	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la <code>cadena</code> seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una <code>cadena</code> seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en un campo estático	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el campo estático seleccionado. El punto de observación se añade a la vista de Observación de datos. Un campo estático es una variable Java definida como estática (una variable de clase). Este comando está disponible cuando hay un campo estático seleccionado (Ediciones Developer y Enterprise de JBuilder).

Tabla 8.12 Menú contextual con selección en la vista Hilos, pilas de llamadas y datos

Comando	Descripción
Crear un punto de observación en un campo	Crea un punto de observación en el campo seleccionado y añade automáticamente el punto de observación a la vista Puntos de observación de datos. Un campo es una variable Java que está definida en un objeto Java. Este comando está disponible cuando hay un campo seleccionado (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de interrupción en un campo	Crea un punto de interrupción en el campo seleccionado y añade automáticamente el punto de interrupción a la vista Puntos de interrupción de datos y código. Un campo es una variable Java que está definida en un objeto Java. Para activar el punto de interrupción, vaya a la vista de Puntos de interrupción de datos y código y haga clic con el botón derecho sobre el punto de interrupción. Seleccione Interrumpir al leer para que el depurador detenga la ejecución del programa justo antes de leer el campo, o Interrumpir al escribir para que lo haga justo antes de escribir en el campo. Este comando está disponible cuando hay un campo seleccionado (Ediciones Developer y Enterprise de JBuilder).
Mostrar valor decimal/hexadecimal	Modifica la base numérica en la que se realiza la visualización de una variable numérica o de carácter. Este comando está disponible cuando hay una variable numérica o de carácter seleccionada. En el caso de las matrices, la selección de este comando modificará la base de sus elementos.
Mostrar/Ocultar valores nulos	Conmuta la visualización de los valores nulos en una matriz. Este comando resulta útil cuando se depura un objeto tabla de colisiones. Está disponible cuando se selecciona una matriz de tipo <code>Object</code> (Ediciones Developer y Enterprise de JBuilder).
Cambiar valor	Muestra el cuadro de diálogo Modificar valor , donde puede modificar directamente el valor de una variable. Este comando está disponible cuando hay una variable seleccionada (Ediciones Developer y Enterprise de JBuilder).

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al hacer clic con el botón derecho en una zona vacía de la vista.

Tabla 8.13 Menú contextual sin selección en la vista Hilos, pilas de llamadas y datos

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista (Ediciones Developer y Enterprise de JBuilder).
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista (Ediciones Developer y Enterprise de JBuilder).

Tabla 8.13 Menú contextual sin selección en la vista Hilos, pilas de llamadas y datos

Comando	Descripción
Mostrar sólo el hilo actual	Muestra únicamente las pilas de llamadas y los datos del hilo actual. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista (Ediciones Developer y Enterprise de JBuilder).
Dividir las vistas de hilos y datos	Divide la vista en dos paneles. La parte izquierda se amplía y muestra marcos de pilas; la derecha muestra el contenido del elemento seleccionado en la parte izquierda. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista (Ediciones Developer y Enterprise de JBuilder).

Vista Puntos de observación de datos








La vista Puntos de observación de datos muestra los valores actuales de los componentes de los datos que se desea controlar. Algunos tipos de expresiones de punto de observación pueden ampliarse, de manera que muestren los elementos de datos en ámbito. Si los elementos no están en el ámbito, aparece en la vista el mensaje <fuera del ámbito>. Los elementos que aparecen en gris son heredados.

Para obtener más información sobre los puntos de observación de datos, consulte ["Observación de expresiones" en la página 8-70](#).

En la siguiente tabla se muestran algunos de los iconos más frecuentes de la vista Puntos de observación de datos. Si desea ver una lista completa de los iconos, consulte la tabla "Iconos del panel de estructura: código fuente", en la sección "El panel de estructura y los iconos UML" de *Introducción a JBuilder*.

En la siguiente tabla aparecen los iconos utilizados en esta vista.

Tabla 8.14 Iconos en la vista Puntos de observación de datos

Icono	Descripción
	Un objeto
	Una matriz
	Una primitiva
 (Rojo)	Un error
 (Gris)	Un mensaje de información

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al seleccionar un elemento en la vista.

Tabla 8.15 Menú contextual con punto de observación seleccionado en la vista Puntos de observación de datos

Comando	Descripción
Eliminar punto de observación	Elimina el punto de observación seleccionado.
Ir a punto de observación	Se desplaza al método del editor en el que se define la variable del punto de observación seleccionado. Este comando sólo está disponible para los puntos de observación de variable de ámbito (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en una variable local	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la variable local seleccionada. Este comando está disponible cuando hay una variable o una matriz de variables seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la matriz seleccionada. Este comando está disponible cuando hay una matriz seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en un componente de una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el componente de matriz seleccionado. Este comando está disponible cuando hay un componente seleccionado de una matriz (Ediciones Developer y Enterprise de JBuilder).
Ajustar rango de visualización	Muestra el cuadro de diálogo Ajustar rango, donde se puede reducir el número de elementos de matriz que se muestran en la vista. Este comando está disponible cuando hay una matriz seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear punto de observación 'this'	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto <code>this</code> seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto <code>this</code> seleccionado (Ediciones Developer y Enterprise de JBuilder).
Crear punto de observación de clases	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la clase seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una clase seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear punto de observación de objetos	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto seleccionado. Un punto de observación de objetos vigila el objeto Java seleccionado. Se amplía para mostrar los miembros de datos de la instanciación actual (Ediciones Developer y Enterprise de JBuilder).

Tabla 8.15 Menú contextual con punto de observación seleccionado en la vista Puntos de observación de datos (continuación)

Comando	Descripción
Mostrar toString()	Ejecuta el método <code>toString()</code> en el objeto seleccionado y muestra la cadena resultante. Por ejemplo, si el objeto seleccionado es <code>City</code> , el comando <code>Mostrar toString()</code> mostrará el valor del objeto, como <code>San Francisco</code> en lugar de <code>com.mycode.City@391</code> (Ediciones Developer y Enterprise de JBuilder).
Crear punto de observación de cadenas	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la cadena seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una cadena seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en un campo estático	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el campo estático seleccionado. El punto de observación se añade a la vista de Observación de datos. Un campo estático es una variable Java definida como estática (una variable de clase). Este comando está disponible cuando hay un campo estático seleccionado (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en un campo	Crea un punto de observación en el campo seleccionado y añade automáticamente el punto de observación a la vista Puntos de observación de datos. Un campo es una variable Java que está definida en un objeto Java. Este comando está disponible cuando hay un campo seleccionado (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de interrupción en un campo	Crea un punto de interrupción en el campo seleccionado y añade automáticamente el punto de interrupción a la vista Puntos de interrupción de datos y código. Un campo es una variable Java que está definida en un objeto Java. Para activar el punto de interrupción, vaya a la vista de Puntos de interrupción de datos y código y haga clic con el botón derecho sobre el punto de interrupción. Seleccione Interrumpir al leer para que el depurador detenga la ejecución del programa justo antes de leer el campo, o Interrumpir al escribir para que lo haga justo antes de escribir en el campo. Este comando está disponible cuando hay un campo seleccionado (Ediciones Developer y Enterprise de JBuilder).
Mostrar/Ocultar valores nulos	Conmuta la visualización de los valores nulos en una matriz. Este comando resulta útil cuando se depura un objeto tabla de colisiones. Está disponible cuando se selecciona una matriz de tipo <code>Object</code> (Ediciones Developer y Enterprise de JBuilder).
Cambiar valor	Muestra el cuadro de diálogo Modificar valor, donde puede modificar directamente el valor de una variable. Este comando está disponible cuando hay un tipo de datos primitivos seleccionado (Ediciones Developer y Enterprise de JBuilder).
Cambiar punto de observación	Muestra el cuadro de diálogo Cambiar punto de observación, con el que se puede cambiar la descripción y la expresión de observación.

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al hacer clic con el botón derecho en una zona vacía de la vista.

Tabla 8.16 Menú contextual sin selección en la vista Puntos de observación de datos

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista (Ediciones Developer y Enterprise de JBuilder).
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista (Ediciones Developer y Enterprise de JBuilder).
Añadir punto de observación	Muestra el cuadro de diálogo Añadir punto de observación, en el que se pueden añadir puntos de observación. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista.
Eliminar todos	Elimina todos los puntos de observación. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista.

Vista Clases cargadas y datos estáticos



La vista Clases cargadas y datos estáticos muestra las clases cargadas en ese momento por el programa. Si una clase tiene datos estáticos, se muestran cuando se amplía la clase. Si se muestra un paquete en el árbol, aparece el número de clases cargadas para ese paquete.

Las clases de esta vista que contengan \$ seguido de un número representan clases internas. El compilador crea clases internas para los manejadores de sucesos definidos como Adaptadores anónimos en la nodo Generados del cuadro de diálogo Propiedades de proyectoFormateo.

Si desea más información, consulte [“Presentación de las variables en el depurador” en la página 8-62](#).





En la siguiente tabla se muestran algunos de los iconos más frecuentes de la vista Clases cargadas y datos estáticos. Si desea ver una lista completa de los iconos, consulte la tabla "Iconos del panel de estructura: código fuente", en la sección "El panel de estructura y los iconos UML" de *Introducción a JBuilder*.

En la siguiente tabla aparecen los iconos utilizados en esta vista.

Tabla 8.17 Iconos en la vista Clases cargadas y datos estáticos

Icono	Descripción
	Un paquete
	Una clase
	Una interfaz
	Una clase bloqueada

Tabla 8.17 Iconos en la vista Clases cargadas y datos estáticos (continuación)

Icono	Descripción
	Un objeto
	Un objeto null
	Una matriz
	Una primitiva

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al seleccionar un elemento en la vista.

Tabla 8.18 Menú contextual con selección de la vista Clases cargadas y datos estáticos

Comando	Descripción
Cortar	Elimina el valor de la variable y la coloca en el portapapeles. Este comando está disponible cuando hay una variable seleccionada (Ediciones Developer y Enterprise de JBuilder).
Copiar	Copia al portapapeles el valor de la variable. Este comando está disponible cuando hay una variable seleccionada (Ediciones Developer y Enterprise de JBuilder).
Pegar	Pega el contenido del portapapeles en otra variable. Cuando se utiliza el comando Pegar, tanto la variable cortada o copiada del objeto como la variable pagada pegada apuntan al mismo objeto. Este comando está disponible cuando hay una variable seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en una variable local	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la variable local seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una variable seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la matriz seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una matriz seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en un componente de una matriz	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el componente de matriz seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un componente seleccionado en una matriz (Ediciones Developer y Enterprise de JBuilder).
Ajustar rango de visualización	Muestra el cuadro de diálogo Ajustar rango, donde se puede reducir el número de elementos de matriz que se muestran en la vista. Este comando está disponible cuando hay una matriz seleccionada (Ediciones Developer y Enterprise de JBuilder).

Tabla 8.18 Menú contextual con selección de la vista Clases cargadas y datos estáticos

Comando	Descripción
Crear punto de observación 'this'	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto <code>this</code> seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto <code>this</code> seleccionado (Ediciones Developer y Enterprise de JBuilder).
Crear punto de observación de clases	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la clase seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una clase seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear punto de observación de objetos	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el objeto seleccionado. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay un objeto seleccionado. Un punto de observación de objetos vigila el objeto Java seleccionado. Se amplía para mostrar los miembros de datos de la instanciación actual (Ediciones Developer y Enterprise de JBuilder).
Crear punto de observación de cadenas	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en la <code>cadena</code> seleccionada. El punto de observación se añade a la vista de Observación de datos. Este comando está disponible cuando hay una <code>cadena</code> seleccionada (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en un campo estático	Muestra el cuadro de diálogo Añadir punto de observación, con el que se puede crear un punto de observación en el campo estático seleccionado. El punto de observación se añade a la vista de Observación de datos. Un campo estático es una variable Java que está definida como estática en un objeto Java. Este comando está disponible cuando hay un campo estático seleccionado (Ediciones Developer y Enterprise de JBuilder).
Crear un punto de observación en un campo	Crea un punto de observación en el campo seleccionado y añade automáticamente el punto de observación a la vista Puntos de observación de datos. Un campo es una variable Java que está definida en un objeto Java. Este comando está disponible cuando hay un campo seleccionado (Ediciones Developer y Enterprise de JBuilder).
Cambiar valor	Muestra el cuadro de diálogo Modificar valor , donde puede modificar directamente el valor de una variable. Este comando está disponible cuando hay una variable seleccionada (Ediciones Developer y Enterprise de JBuilder).

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al hacer clic con el botón derecho en una zona vacía de la vista.

Tabla 8.19 Menú contextual sin selección de la vista Clases cargadas y datos estáticos

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista (Ediciones Developer y Enterprise de JBuilder).
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista (Ediciones Developer y Enterprise de JBuilder).



Es una función de
JBuilder Developer y
Enterprise.

Vista Monitores de sincronización

La vista Monitores de sincronización muestra los monitores de sincronización que utilizan los hilos del programa y su estado actual. Esto resulta útil para detectar situaciones de bloqueo. En JBuilder Foundation, aparece la pestaña, pero no se muestra el estado de bloqueo.

Nota

Algunas MV como, por ejemplo, versiones anteriores de HotSpot, no proporcionan esta información. Si la vista de los monitores de sincronización no se encuentra disponible y su MV admite `classic`, es necesario añadir `-classic` como un parámetro MV de la siguiente manera:

- 1 Abra el cuadro de diálogo Configuraciones de ejecución (Ejecutar! Configuraciones) y seleccione la configuración de ejecución que esté utilizando.
- 2 Haga clic en Modificar para modificarla.
- 3 En el nodo Ejecutar, escriba `-classic` en el campo Parámetros de la MV.
- 4 Pulse dos veces sobre Aceptar para cerrar los cuadros de diálogo.

Si desea más información sobre hilos, consulte [“Gestión de hilos” en la página 8-36](#).

En la siguiente tabla aparecen los iconos utilizados en esta vista.

Tabla 8.20 Iconos en la vista Monitores de sincronización

Icono	Descripción
	El monitor de sincronización utilizado por el hilo especificado no está bloqueado.
	El monitor de sincronización utilizado por el hilo especificado está bloqueado.

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles en esta vista.

Tabla 8.21 Menú contextual de la vista Monitores de sincronización

Comando	Descripción
Ventana flotante	Convierte la vista en ventana flotante. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista (Ediciones Developer y Enterprise de JBuilder).
Restablecer orden predeterminado	Restablece el orden por defecto de las vistas del depurador. Este comando está disponible pulsando con el botón derecho en una zona vacía de la vista (Ediciones Developer y Enterprise de JBuilder).



Es una función de
JBuilder Developer y
Enterprise.

Vista Personalizar

La vista Personalizar permite añadir, modificar o eliminar un visualizador personalizado para un objeto en concreto. Un visualizador personalizado se utiliza para personalizar información de objetos que aparecen en la vista Hilos, pilas de llamada y datos o en la vista Puntos de observación de datos. Antes de depurar, se puede abrir esta vista mediante el comando Ejecutar! Vistas personalizadas.

Si desea obtener más información acerca del uso de la vista Personalizar, consulte [“Utilización de un visualizador personalizado en un objeto” en la página 8-67](#).

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al seleccionar un elemento en la vista.

Tabla 8.22 Menú contextual con visualizador personalizado seleccionado

Comando	Descripción
Edición	Abre el Editor de vista personalizada, en el que se modifica la vista personalizada seleccionada.
Eliminar	Borra la plantilla seleccionada.

En la siguiente tabla aparecen los comandos del menú contextual que están disponibles al hacer clic con el botón derecho en una zona vacía de la vista.

Tabla 8.23 Menú contextual sin selección en la vista Puntos de observación de datos

Comando	Descripción
Añadir	Abre el cuadro de diálogo Seleccionar archivo fuente Java, en el que se escribe el nombre de la clase de la vista personalizada.
Eliminar todos	Elimina todos los visualizadores personalizados.

Barra de herramientas del depurador

La barra de herramientas de la parte inferior del depurador proporciona un acceso rápido a las acciones más utilizadas por el depurador. La parte derecha de la barra de herramientas, que corresponde con la barra de estado del depurador, muestra mensajes de estado.

Figura 8.2 Barra de herramientas del depurador








La tabla que aparece a continuación explica detalladamente los iconos de la barra de herramientas.

Tabla 8.24 Botones de la barra de herramientas

Icono	Acción	Descripción
	Terminar el programa	Pone fin a la ejecución actual de la aplicación y libera la memoria. Su efecto es el mismo que el de Ejecutar Terminar el programa.
	Reanudar el programa	Reinicia el depurador que se ha apagado o reiniciado o continua con el actual. Su efecto es el mismo que el de Ejecutar Reanudar el programa.
	Pausar el programa	Detiene provisionalmente la sesión de depuración actual. Su efecto es el mismo que el de Ejecutar Pausar el programa.
	Paso inteligente activado/Paso inteligente desactivado	Controla el uso de los parámetros de Paso inteligente de la Vista de clases con inspección desactivada y las opciones de Paso inteligente de la ficha Depurar del cuadro de diálogo Propiedades de ejecución.
	Omitir inspección	Omite la inspección de la línea de código actual. Su efecto es el mismo que el de Ejecutar Omitir inspección.
	Inspeccionar	Inspecciona la línea de código actual. Su efecto es el mismo que el de Ejecutar Inspeccionar código.
	Abandonar inspección	Sale del método actual y vuelve al elemento que ha efectuado la llamada. Su efecto es el mismo que el de Ejecutar Abandonar inspección.
	Intercambio inteligente	Compila los archivos modificados y actualiza las clases compiladas. Su efecto es el mismo que el de Ejecutar Intercambio inteligente (Ediciones Developer y Enterprise de JBuilder).

Tabla 8.24 Botones de la barra de herramientas (continuación)

Icono	Acción	Descripción
	Establecer punto de ejecución	Define el lugar en el que el programa está para iniciarse. El punto de ejecución se traslada a la nueva ubicación. Su efecto es el mismo que el de Ejecutar Establecer punto de ejecución (Ediciones Developer y Enterprise de JBuilder).
	Fuente inteligente	Establece el tipo de archivo fuente, basado en el lenguaje de código fuente original ajeno a Java. Coloca el cursor en la vista del nuevo archivo del marco de pila actual. Su efecto es el mismo que el de Ejecutar Fuente inteligente (Ediciones Developer y Enterprise de JBuilder).
	Añadir punto de interrupción	Añade un punto de interrupción a la sesión de depuración actual. Haga clic en la flecha de lista desplegable que aparece a la derecha del botón para seleccionar el tipo de punto de interrupción. Su efecto es el mismo que el de Ejecutar Añadir punto de interrupción.
	Añadir punto de observación	Añade un punto de observación a la sesión de depuración actual. Su efecto es el mismo que el de Ejecutar Añadir punto de observación.
	Mostrar marco actual	Muestra la pila de llamadas del hilo actual y resalta en el código fuente el punto de ejecución.

Métodos abreviados para el depurador

Las siguientes teclas de método abreviado permiten acceder fácilmente a las funciones de depuración.

Tabla 8.25 Métodos abreviados para el depurador

Contraseña	Acción
<i>Mayús+F9</i>	Depurar proyecto
<i>Ctrl+F2</i>	Terminar el programa
<i>F4</i>	Ejecutar hasta el cursor
<i>F5</i>	Conmutar punto de interrupción en el editor
<i>F6</i>	cuadros de diálogo:Evaluar/Modificar
<i>F7</i>	Inspeccionar
<i>F8</i>	Omitir inspección
<i>F9</i>	Reanudar el programa (reanuda la sesión de depuración actual)

Tabla 8.25 Métodos abreviados para el depurador (continuación)

Contraseña	Acción
<i>Ctrl+clic con el botón derecho del ratón en el margen sobre el punto de interrupción</i>	Mostrar el cuadro de diálogo Propiedades del punto de interrupción Las propiedades centrales son de sólo lectura, pero las opciones se pueden modificar.
<i>Ctrl+botón derecho en el editor y sobre la expresión</i>	Mostrar la ventana ExpressionInsight de esa expresión.

ExpressionInsight

Es una función de
JBuilder Developer y
Enterprise.

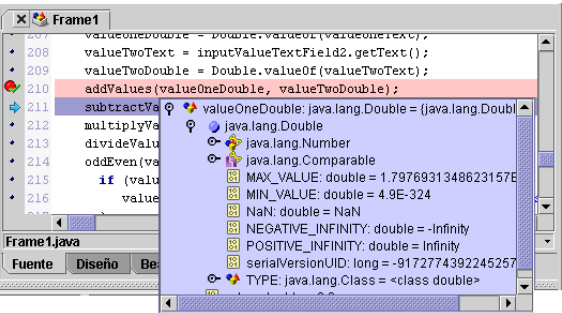
Cuando se interrumpe el depurador, se puede acceder a ExpressionInsight, una pequeña ventana emergente que muestra, mediante una representación en árbol, el contenido de la expresión seleccionada. Para mostrar la ventana ExpressionInsight:

- Mantenga pulsado el botón *Ctrl* y lleve el ratón sobre el código en el editor. La ventana ExpressionInsight se muestra cuando el ratón pasa sobre una expresión con sentido.
- Mueva el ratón a la expresión que se desea analizar en más detalle y pulse *Ctrl* junto con el botón derecho del ratón.

La ventana ExpressionInsight se abre hasta que pulsa una tecla para cerrar.

La ventana ExpressionInsight permite descender por los miembros de la expresión. Si la expresión es un objeto, el menú contextual muestra los mismos comandos de menú que los que están disponibles en la vista Hilos, pilas de llamadas y datos cuando se selecciona un objeto. También puede pulsar con el botón derecho en un descendiente de la ventana para que aparezca un menú contextual.

Figura 8.3 Ventana ExpressionInsight



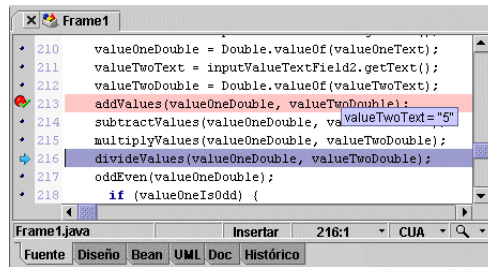
La ventana ExpressionInsight está desactivada cuando no ha finalizado o no se ha interrumpido la sesión de depuración.

Ayuda inmediata

Es una función de JBuilder Developer y Enterprise.

Mientras la depuración está detenida, si se coloca el cursor en una variable, en el editor, se muestra su valor. El valor aparece en una pequeña ventana emergente llamada "ayuda inmediata". Si se selecciona texto, se ve su valor.

Figura 8.4 Ventana Ayuda inmediata



La ayuda inmediata se desactiva cuando la sesión de depuración finaliza o no se ha suspendido.

Depuración de código fuente ajeno a Java

Nota

Es una función de JBuilder Developer y Enterprise.

Con JBuilder Developer sólo es posible depurar el código JSP. No permite depurar otros tipos de código fuente ajeno a Java.

JBuilder se puede utilizar para depurar código fuente distinto de Java, incluidos JSP, SQLJ y código LegacyJ. Se puede depurar de forma local y de forma remota. Para conseguir esto, JBuilder utiliza la información de mapeo que se guarda en el archivo de clase (consulte JSR-45). Esto le permite realizar la depuración del modo en que lo haga normalmente. Puede ejecutar e interrumpir el programa, establecer y ejecutar puntos de interrupción, inspeccionar el código línea a línea, así como examinar y modificar los valores de los datos.

Al interrumpir el programa, puede cambiar la vista de su código, lo que permite ver el código fuente Java o el código distinto de Java. Por ejemplo, si está depurando una JSP y se ha detenido en un punto de interrupción, puede ver el código fuente Java de esa JSP o la propia JSP.



Para intercambiar las vistas, utilice el botón Fuente inteligente de la barra de herramientas del depurador. Una ventana emergente muestra la vista del código fuente seleccionada, así como las disponibles. Si selecciona una vista fuente diferente de la actual, el editor vuelve a dibujar el archivo asociado con la nueva vista fuente. El estado de la vista fuente se mantiene durante la sesión de depuración, con lo que si se cambia el código fuente se cambia el archivo que aparece cuando se suspende la MV.

Nota

La vista fuente por defecto es la que JBuilder determina como la mejor para el código actual.

Si cambia las vistas fuente, también se cambian el marco de la pila actual y la ubicación del cursor. Por ejemplo, si está depurando una JSP y ve el código de la JSP, debe haber establecido el punto de interrupción en la línea 25. Sin embargo, si cambia al código fuente Java, el cursor cambia a la línea 75. Esto se produce porque los marcos de pila análogos no se encuentran en las mismas líneas de código en los dos archivos.

Control de la ejecución del programa

La característica más importante de un depurador es que permite controlar la ejecución del programa. Es posible controlar si el programa ejecuta una sola línea de código, un método completo o un bloque de programa completo. Controlando manualmente cuándo debe ejecutarse y cuándo debe detenerse el programa, puede pasar rápidamente por las secciones que sabe que funcionan correctamente y concentrarse en las secciones que causan los problemas.

Ejecución e interrupción del programa

Cuando el programa se ejecuta en el depurador es necesario detenerlo momentáneamente para analizar los valores de los datos. La detención temporal hace que el depurador suspenda la ejecución del programa. Desde aquí puede utilizar el depurador para examinar el estado del programa respecto a la ubicación del programa.

Cuando se está utilizando el depurador, el programa puede estar en uno de dos estados posibles: *en ejecución* o *en interrupción*.



- El programa está en modo de ejecución cuando el botón Pausa está disponible en la barra de herramientas del depurador.
- El programa se encuentra suspendido cuando el botón Pausa aparece atenuado. Cuando se interrumpe el programa es posible analizar los valores de los datos. Entonces se habilitan los botones de inspección de la barra de herramientas del depurador.




Para continuar ejecutando el programa, seleccione el botón Reanudar el programa en la barra de herramientas del depurador. Cuando la sesión de depuración finaliza, este botón pasa a ser Reiniciar el programa y reinicia la sesión.

Mientras el programa está interrumpido, puede modificar el código y continuar con la ejecución en todos los marcos de pilas activos. Si desea más información, consulte [“Modificación del código durante la depuración” en la página 8-76](#).

Reinicio del programa

En ocasiones, durante la depuración, resulta necesario reiniciar el programa. Por ejemplo, puede ser necesario reiniciar el programa si se ha pasado por alto la ubicación de un error, o si se dañan variables o estructuras de datos con valores no deseados.

Para detener la ejecución actual del programa, efectúe una de las siguientes acciones:

- Seleccione Ejecutar|Terminar el programa.
-  ▪ Haga clic en el botón Reanudar el programa de la barra de herramientas del depurador.

El reinicio de un programa libera recursos y borra la configuración de todas las variables. Sin embargo, reiniciar un programa no borra los puntos de interrupción ni los puntos de observación que haya definido, por lo que resulta sencillo reanudar la sesión de depuración.



Para reiniciar el programa, pulse el botón Reiniciar el programa de la barra de herramientas del depurador.

El punto de ejecución



Durante una sesión de depuración interrumpida, la línea de código que sea el punto de ejecución actual para un hilo aparece resaltada en el editor mediante una flecha en el margen izquierdo del editor.

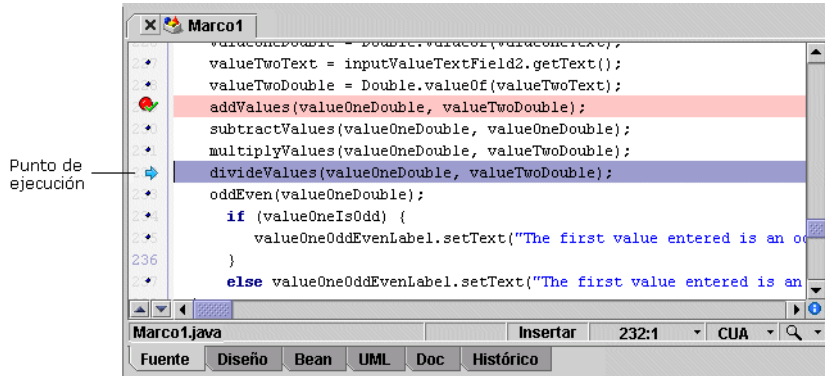
El punto de ejecución marca la actual línea del código fuente que va a ejecutar el depurador. Cuando detiene la ejecución del programa dentro del depurador, se resalta el punto de ejecución actual en el hilo de proceso seleccionado. El punto de ejecución siempre muestra la línea actual de código que va a ejecutarse, tanto si desea omitir la inspección, inspeccionar código o ejecutar el programa sin detenerse.

Para encontrar el punto de ejecución actual, utilice uno de estos métodos:



- Seleccione Ejecutar|Mostrar punto de ejecución.
- Pulse en el botón Mostrar marco de la pila actual en la barra de herramientas del depurador.

El editor muestra el bloque de código en el área donde se encuentre en ese momento el punto de ejecución. El punto de ejecución se marca con una flecha en el margen izquierdo del editor, y se resalta esa línea de código. La ejecución del programa continúa a partir de ese punto.

Figura 8.5 El punto de ejecución

Durante la depuración, es posible abrir, cerrar y desplazarse por cualquier archivo en el editor. Por ello resulta fácil perder la pista de qué sentencia de programa se ejecutará a continuación, o de la ubicación del ámbito del programa actual. Para volver rápidamente al punto de ejecución, seleccione Ejecutar!Mostrar punto de ejecución, o bien, pulse el botón Mostrar marco de la pila actual de la barra de herramientas del depurador.



Es una función de
JBUILDER Developer y
Enterprise.

Definición del punto de ejecución

Cuando el programa se suspende, se puede definir el punto de ejecución del hilo de inspección actual. Esto cambia el punto de ejecución de su ubicación actual. También es posible definir el punto de ejecución una vez que se ha utilizado el botón Intercambio inteligente. (Si desea obtener más información acerca del Intercambio inteligente, consulte [“Modificación del código durante la depuración” en la página 8-76](#)).

Para definir el punto de ejecución del hilo de inspección actual:

- 1 Abra la vista Hilos, pilas de llamadas y datos.
- 2 Seleccione el marco de pila en el que desea continuar las operaciones, pulse con el botón derecho del ratón y, a continuación, seleccione Establecer punto de ejecución.
- 3 El editor muestra el código fuente del área del nuevo punto de ejecución. Si este es el hilo de inspección, el punto de ejecución se resalta en el margen izquierdo del editor con una flecha, y se resalta la línea de código. La ejecución del programa continúa a partir de ese punto.



Puede utilizar el botón Establecer punto de ejecución de la barra de herramientas del depurador para definir el punto de ejecución. El botón muestra una ventana emergente que enumera los marcos de pilas del hilo de inspección. Puede elegir el que desee. Observe que la selección del marco de pila actual está atenuada. El marco de pila donde va a continuar la ejecución del programa se marca en la vista Hilos, pilas de llamada y datos con el botón de inspección. También puede utilizar el comando de menú Ejecutar! Establecer punto de ejecución para establecer un nuevo marco de pila para continuar con las operaciones.

Nota La definición del punto de ejecución no redefine el valor de las variables de objeto que se han modificado en las pilas de llamadas.

Gestión de hilos

Para gestionar los hilos del programa con el depurador se pueden utilizar la vista Hilos, pila de llamadas y datos y la vista Monitores de sincronización.

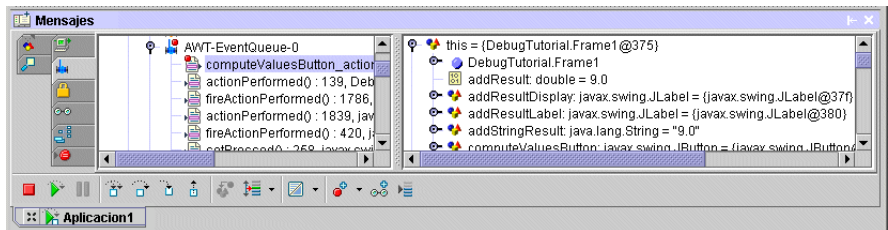
- La vista Hilos, pila de llamadas y datos muestra el estado actual de todos los grupos de hilos del programa. También muestra todas las llamadas a métodos que ha realizado el programa, y el orden en el que han sido llamadas. Esto permite conocer las llamadas que se han realizado y que han conducido al error actual. Mediante este panel también se puede retroceder al lugar desde donde se llamó al método.
- La vista de Monitores de sincronización muestra todos los monitores de sincronización que utilizan todos los hilos del programa depurado, así como su estado actual.

Utilización del panel dividido

**Es una función de
JBuilder Developer y
Enterprise.**

La visualización por defecto de los Hilos, pilas de llamadas y datos se divide en dos paneles. El panel izquierdo se puede ampliar para que muestre los marcos de pila. El panel derecho muestra el contenido del elemento seleccionado a la izquierda, y muestra todo, desde un grupo de hilos hasta una variable. Por ejemplo, si se selecciona un hilo en el panel izquierdo, el panel derecho muestra los marcos de pila de ese hilo. Por otra parte, si se selecciona un marco de pila en el panel de la izquierda, el panel de la derecha mostrará las variables disponibles en esa vista.

Figura 8.6 Panel dividido de la vista Hilos, pilas de llamadas y datos



Presentación del hilo actual

**Es una función de
JBuilder Developer y
Enterprise.**

Para mostrar únicamente las pilas de llamadas y los datos del hilo actual:

- 1 Abra la vista Hilos, pilas de llamadas y datos.
- 2 Haga clic con el botón derecho del ratón en una zona vacía de la vista.
- 3 Seleccione Mostrar sólo el hilo actual. Todos los hilos excepto el actual se eliminan de la vista.

- 4 Para mostrar de nuevo todos los hilos, haga clic con el botón derecho del ratón en una zona vacía de la vista y seleccione Mostrar sólo el hilo actual.




Presentación del marco de pila superior



Para mostrar el marco de pila superior del hilo actual, pulse el botón Mostrar marco de la pila actual de la barra de herramientas del depurador.

Elección de un hilo para inspeccionarlo

Para elegir el hilo que se va a inspeccionar:

- 1 Compruebe que se muestran todos los hilos en la vista Hilos, pila de llamadas y datos. (Haga clic con el botón derecho del ratón y compruebe que la opción Mostrar sólo el hilo actual está desactivada).
- 2 Seleccione el hilo que desea inspeccionar.
- 3 Haga clic con el botón derecho del ratón y seleccione Establecer hilo inspeccionado. El icono del nuevo hilo inspeccionado cambia a . Tenga en cuenta que también se puede elegir para la inspección un hilo suspendido por el usuario o bloqueado. El icono incluye un pequeño punto rojo, por ejemplo,  or .

Interrupción prolongada de un hilo

Es una función de
JBuilder Developer y
Enterprise.



Una vez se ha interrumpido la depuración y está preparado para continuar la ejecución, puede mantener interrumpido un hilo. Esto permite observar el comportamiento de determinados hilos, sin que los otros interfieran con ellos.

Para continuar ejecutando el programa, seleccione el botón Reanudar el programa en la barra de herramientas del depurador. Cuando continúe la ejecución del programa, sólo los hilos que no se mantengan interrumpidos reanudarán la ejecución.

Advertencia

Esto puede dar lugar a una situación conflictiva.

Para mantener interrumpido un hilo,



- 1 Depure el programa y detenga el depurador mediante el botón Pausa si todavía no se ha detenido.
- 2 En la vista Hilos, pilas de llamadas y datos, haga clic con el botón derecho sobre el hilo que quiere mantener interrumpido.
- 3 Seleccione Mantener hilo interrumpido.
- 4 Pulse el botón Reanudar el programa.
- 5 El hilo seleccionado no reanudará la ejecución.




Un hilo interrumpido por el usuario viene indicado por el icono .

La posibilidad de detectar hilos en conflicto o bloqueados es una característica de las ediciones Developer y Enterprise de JBuilder.

Detección de conflictos

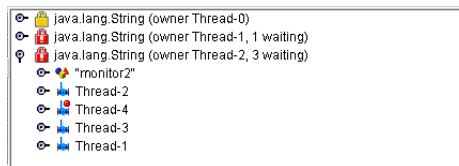
La vista Monitores de sincronización permite detectar los hilos bloqueados. Esta vista se puede utilizar para comprobar qué hilos esperan a qué monitores.

En esta vista, el monitor muestra el hilo que es su propietario, y los hilos, si los hay, que esperan para obtenerlo. Cuando un monitor se encuentra bloqueado, más de un hilo intenta obtenerlo. Sin embargo, no es posible liberarlo, porque el hilo que lo retiene espera a que se libere otro monitor. El icono  muestra que un monitor está bloqueado.

Cuando se expande un monitor en la vista se ven todos los hilos que lo esperan, así como su propietario. Los iconos de esta vista muestran si el hilo es el que se encuentra activo actualmente. Los hilos se pueden expandir para mostrar su pila actual, como en la vista de Hilos, pila de llamadas y datos.



En el ejemplo siguiente, los hilos 1 y 2 tienen hilos en espera. Los hilos 4, 3 y 1 esperan al hilo 2; el hilo 2 es el propietario. El objeto de hilo se denomina "monitor2". El hilo 4 es el que se está inspeccionando.

Figura 8.7 Vista Monitores de sincronización



Desplazamiento a través del código

Los comandos Ejecutar|Inspeccionar y Ejecutar|Omitir inspección ofrecen el método más sencillo para desplazarse en el código del programa. A pesar de que los dos comandos son muy similares, cada uno ofrece una manera diferente de controlar la ejecución del código.

Nota También se pueden utilizar los botones Inspeccionar  u Omitir inspección  de la barra de herramientas del depurador.

El mínimo incremento con el que puede recorrer un programa es una sola línea de código. Si una línea de texto contiene varias sentencias de programa, se tratan como una sola línea de código; no es posible depurar individualmente varias sentencias contenidas en una sola línea de texto. Lo más sencillo es situar cada sentencia en su propia línea. Las sentencias que ocupan varias líneas de texto se tratan como una sola línea de código.

Conforme vaya depurando, puede inspeccionar el código de algunos métodos y excluir otros. Si está seguro de que un método funciona correctamente, puede omitir la inspección de dicho método, sabiendo que la llamada a métodos no provoca errores. Si no está seguro de que un método funciona bien, inspeccione el código del método y verifique si funciona. Debe omitir la inspección de métodos de bibliotecas proporcionadas por JBuilder u otros

proveedores. Esto aumentará considerablemente la velocidad del ciclo de depuración.

Inspección de código de llamadas a métodos

El comando Ejecutar|Inspeccionar ejecuta las sentencias de programa de una en una. Si la función Paso inteligente se encuentra activada, no se controlan las clases de la vista Clases con inspección desactivada, que tengan desactivada la inspección. Si Paso inteligente se encuentra desactivada, las clases de la Vista Clases con inspección desactivada se pasan por alto, de forma que es posible inspeccionarlas todas.

Si el punto de ejecución se sitúa en una llamada a un método, el comando Inspeccionar código entra en el método y coloca el punto de ejecución en su primera sentencia. Los comandos Inspeccionar posteriores ejecutan las líneas de código del método una por una.

Si el punto de ejecución se encuentra en la última sentencia de un método, Inspeccionar hace que el depurador regrese desde el método, situando el punto de ejecución en la línea de código que sigue a la llamada al método del que regresa.

El término "avance paso a paso" se refiere a la utilización de Inspeccionar para ejecutar sucesivamente las sentencias del código del programa.

Existen varias maneras de ejecutar el comando Inspeccionar.

- Seleccione Ejecutar|Inspeccionar.
- Pulse *F7*.
- Pulse el botón Inspeccionar de la barra de herramientas del depurador.



Omisión de inspección en las llamadas a métodos

El comando Ejecutar|Omitir inspección, igual que Ejecutar|Inspeccionar, permite ejecutar las sentencias de programa de una en una. Sin embargo, si ejecuta el comando Omitir inspección cuando el punto de ejecución se encuentra en una llamada a un método, el depurador ejecuta ese método sin detenerse (en lugar de seguir su ejecución), situando a continuación el punto de ejecución en la sentencia que sigue a la llamada al método.

Existen varias maneras de ejecutar el comando Omitir inspección.

- Seleccione Ejecutar|Omitir inspección.
- Pulse *F8*.
- Pulse el botón Omitir inspección de la barra de herramientas del depurador.



Salida de métodos

El comando Ejecutar|Abandonar inspección permite abandonar la inspección de un método para pasar a la rutina de llamada.

Si la función Paso inteligente se encuentra activada, las clases de la vista Clases con inspección desactivada no se inspeccionan.

Existen varias maneras de ejecutar el comando Abandonar inspección:



- Seleccione Ejecutar|Abandonar inspección.
- Pulse el botón Abandonar inspección de la barra de herramientas del depurador.

Paso inteligente



El conmutador Paso inteligente permite determinar si la inspección es "inteligente". Para habilitar este conmutador, elija Activar Paso inteligente en la ficha Depurar del cuadro de diálogo Propiedades de ejecución. Si lo prefiere, pulse el botón Paso inteligente de la barra de herramientas del depurador para activar el Paso inteligente en la sesión actual.

Cuando esta función se encuentra activada, las operaciones de inspección utilizan las clases que aparecen en la vista Clases con inspección desactivada. En las ediciones Developer y Enterprise de JBuilder, la inspección también se controla mediante las opciones de Paso inteligente de la ficha Depurar del cuadro de diálogo Propiedades de ejecución.

- La vista Clases con inspección desactivada permite determinar las clases de las que no se efectúa un seguimiento. En JBuilder Foundation sólo hay tres clases disponibles en esta vista: `java.lang.Object`, `java.lang.String` y `java.lang.ClassLoader`. No es posible añadir, modificar ni eliminar elementos de esta vista.
- En las ediciones Developer y Enterprise de JBuilder, las opciones de Paso inteligente de la ficha Depurar del cuadro de diálogo Propiedades de ejecución determinan la forma de inspección de las clases que se controlan. Estas opciones son:
 - Omitir métodos sintéticos
Omite los métodos sintéticos cuando se inspeccionan clases.
 - Omitir constructores
Omite los constructores cuando se inspeccionan clases.
 - Omitir inicializadores estáticos
Omite los inicializadores estáticos cuando se inspeccionan clases.
 - Avisa si se detiene en una clase con la inspección desactivada (Es una función de todas las ediciones de JBuilder).
Muestra un mensaje de advertencia si hay un punto de interrupción en una clase que tenga la inspección desactivada. Si desea más información, consulte ["Puntos de interrupción y configuración de Inspección desactivada" en la página 8-46](#).

Si la opción Paso inteligente se encuentra desactivada, las clases de la vista Clases con inspección desactivada y las opciones de Paso inteligente se pasan por alto, de forma que es posible inspeccionarlas todas.

Paso inteligente se encuentra activado por defecto cuando se inicia una sesión de depuración.



- Si desea desactivarlo en la sesión actual, pulse el botón Paso inteligente de la barra de herramientas del depurador. Para desactivarlo al principio de la sesión de depuración, desactive la opción Activar paso inteligente en la ficha Depurar en el cuadro de diálogo Propiedades de ejecución.
- El botón Paso inteligente de la barra de herramientas del depurador se atenúa cuando esta opción se encuentra desactivada: para volver a activarlo, pulse el botón o active la opción Activar paso inteligente de la ficha Depurar del cuadro de diálogo Propiedades de ejecución.

Ejecución hasta un punto de interrupción

Defina puntos de interrupción en las líneas de código fuente donde quiera que se detenga temporalmente la ejecución del programa. La ejecución hasta un punto de interrupción es similar a la ejecución hasta una posición del cursor, en ambos casos, el programa se ejecuta, hasta que alcanza un determinado punto del código fuente en el que se detiene.

El código puede tener varios puntos de interrupción. Puede personalizar cada uno de estos puntos para establecer las condiciones particulares bajo las que se debe detener el programa.



Si está depurando código distinto de Java y el programa está detenido en un punto de interrupción, puede cambiar las vistas. Pulse el botón Fuente inteligente de la barra de herramientas del depurador o, bien, seleccione Ejecutar|Fuente inteligente. Seleccione la vista del código que desee ver. La fuente se muestra en el editor, y el cursor se coloca en el marco de pila actual. Tenga en cuenta que se trata probablemente de un número de línea diferente que en la vista anterior. Por ejemplo, si está depurando una JSP, debe estar en la línea 120 del código fuente Java, y en la línea 55 del código fuente de la JSP (el código fuente distinto de Java). Si desea más información, consulte [“Depuración de código fuente ajeno a Java” en la página 8-32](#).

Si desea más información, consulte [“Puntos de interrupción” en la página 8-46](#).

Ejecución hasta el final de un método

El comando Ejecutar|Ejecutar hasta el final del método ejecuta una aplicación hasta que llega al final del método actual. Este comando resulta muy útil cuando se inspecciona un método para el que se pretendía omitir la inspección.

Ejecución hasta la posición del cursor

Puede ejecutar el programa hasta un punto justamente anterior a donde sospecha que puede encontrarse el problema. En ese punto, verifique que todos los valores de los datos sean correctos. A continuación, ejecute el programa hasta otra ubicación y verifique ahí los valores.

Para ejecutar el programa hasta una ubicación específica:

- 1 En el editor, sitúe el cursor en la línea de código en la que desea comenzar (o reanudar) la depuración.
- 2 Seleccione Ejecutar|Ejecutar hasta el cursor o haga clic con el botón derecho y seleccione Ejecutar hasta el cursor.

Con el comando Ejecutar hasta el cursor, el programa se ejecuta sin detenerse, hasta que alcanza la posición marcada por el cursor en el editor. Cuando la ejecución llega al código marcado por el cursor, el depurador recupera el control, interrumpe el programa y sitúa el punto de ejecución en esa línea del código. Si desea más información sobre el punto de ejecución, consulte [“El punto de ejecución” en la página 8-34](#).

Este comando acelera el proceso de depuración, dado que permite desplazarse con rapidez a través de código sin errores.

Visualización de llamadas a métodos

La vista Hilos, pila de llamadas y datos muestra todos los grupos de hilos del programa. Para cada hilo se muestra la secuencia de llamadas a métodos que llevaron al programa a su estado actual. Cada marco de pila se puede ampliar con el fin de mostrar los datos disponibles.

Si su programa se ha compilado con información de depuración (por defecto), esta vista también muestra los argumentos que se han pasado a una llamada de método. Cada método va seguido de una lista de los parámetros con los que se ha realizado la llamada. Además, la vista muestra dónde reside cada método. Recoge la línea en la que está la llamada del método, el nombre de la clase y el nombre del archivo fuente.

Para ver el código fuente y el estado de los datos de una llamada a un método, haga clic en el método.

Localización de una llamada a un método

Gracias a que es posible encontrar el lugar en el que se ha llamado a un método en el código fuente, se puede retroceder en una sesión de depuración.

Para buscar una llamada a un método, siga uno de estos procedimientos:

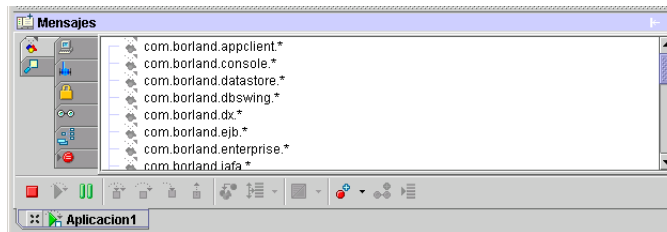
- Haga clic sobre el método en la vista Hilos, pilas de llamadas y datos. Esto le lleva al editor, en el cual aparece el cursor en la línea de código del archivo desde la que se llamó al método.
- Haga clic con el botón derecho sobre el editor y seleccione el comando Ejecutar hasta el cursor.

Determinación de las clases que se han de inspeccionar

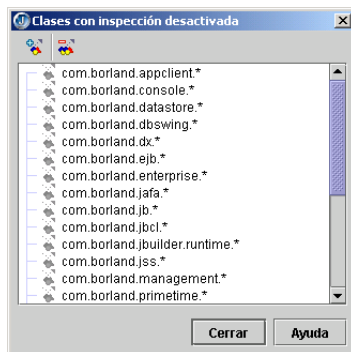
Si desea examinar detenidamente una parte del programa, puede indicar al depurador que inspeccione únicamente los archivos deseados. De esta forma, puede concentrarse en un área problemática conocida, en lugar de tener que avanzar manualmente paso a paso por todas y cada una de las líneas de código de todo el programa. Por ejemplo, normalmente no inspeccionará paso a paso las clases de la biblioteca JDK, porque no le hace falta resolver problemas en ellas; por lo general, solamente deseará inspeccionar y resolver los problemas de sus propias clases.

Para determinar las clases en las que se va a inspeccionar el código:

- Si está en sesión de depuración, seleccione la vista Clases con inspección desactivada. Este comando muestra las clases en las que no se debe inspeccionar el código. Por defecto, no se inspeccionan las clases.





- Si la sesión de depuración no ha comenzado, seleccione Ejecutar/Clases con inspección desactivada. Entonces se muestra el cuadro de diálogo Clases con inspección desactivada. Observe que el cuadro de diálogo cuenta con una barra de herramientas para facilitar la adición y eliminación de clases y paquetes.



Nota En JBuilder Foundation solamente se añaden a esta vista tres clases (`java.lang.Object`, `java.lang.String` y `java.lang.ClassLoader`). No se pueden añadir, modificar o eliminar elementos de la lista, aunque puede elegir entre inspeccionar o no esas clases. Para más información consulte [“Paso inteligente” en la página 8-40](#).

Las clases y paquetes se pueden activar y desactivar en cualquier momento desde la vista Clases con inspección desactivada: basta con hacer clic con el

botón derecho del ratón en la clase o el paquete y activar o desactivar la opción Inspeccionar clase/paquete. Si se encuentra desactivada (por defecto), la clase o el paquete no se inspecciona. Si se encuentra activada, se realiza la inspección. Observe que el icono cambia según el estado.

- Cuando la inspección está activada, el icono es:  (El icono es en color).
- Cuando está desactivada, el icono es:  (El icono es gris).

Nota Cuando se desactiva la inspección de un paquete se desactiva la inspección de todas las clases que contiene.

En las ediciones Developer y Enterprise de JBuilder, si desea eliminar una clase o un paquete de la lista, selecciónelo y pulse *Borrar* o, bien, selecciónelo, haga clic con el botón derecho del ratón y seleccione Eliminar clase/paquete. Si desea borrar todas las clases y paquetes, haga clic con el botón derecho del ratón en una zona vacía de la vista y seleccione Eliminar todos. Si se borran todas las clases y paquetes de la vista, se activa automáticamente la inspección de todas las clases a las que llama el programa.

En JBuilder Developer y Enterprise, se pueden añadir clases o paquetes a la lista haciendo clic con el botón derecho en una zona vacía de la vista y seleccionando Añadir. Aparece el cuadro de diálogo Seleccione una clase o paquete, en el que se puede elegir el nombre de la clase o paquete para que el se desea desactivar la inspección.

En las ediciones Developer y Enterprise de JBuilder, puede modificar clases y paquetes de la lista haciendo clic con el botón derecho del ratón en una clase o paquete de la vista y seleccionando Modificar clase/paquete. Aparece el cuadro de diálogo Seleccionar clase o paquete, en el que se puede introducir el nombre de la clase o el paquete que se desea inspeccionar.

Los cambios se efectúan de forma inmediata. No es necesario reiniciar la sesión de depuración.

Las clases de la vista Clases con inspección desactivada, con su estado activado o desactivado, se guardan en el archivo del proyecto.



Después de seleccionar las clases que no desea inspeccionar, pulse el botón Paso inteligente de la barra de herramientas del depurador para controlar la inspección. Cuando esta función se encuentra activada, las operaciones de inspección utilizan las clases que aparecen en la vista Clases con inspección desactivada y las opciones de paso inteligente seleccionadas en la nodo Depurar del cuadro de diálogo Nuevo/Editar configuración de ejecución:

- La vista Clases con inspección desactivada permite determinar las clases de las que no se efectúa un seguimiento.
- Los parámetros de Paso inteligente de la ficha Depurar del cuadro de diálogo Propiedades de ejecución controlan la forma de inspección de las clases de las que se efectúa un seguimiento.

Si la opción Paso inteligente se encuentra desactivada, las clases de la vista Clases con inspección desactivada y los parámetros de Paso inteligente se pasan por alto, de forma que es posible inspeccionarlas todas.

Es una función de
JBuilder Developer y
Enterprise.

Inspección de clases cuando el archivo fuente no está disponible

Si se desactiva el Paso inteligente mientras se utiliza una clase cuyo archivo de código fuente no está disponible, se genera un archivo fuente stub, que aparece cuando se inspecciona el código. El archivo fuente stub sólo muestra las firmas de los métodos de esa clase. Si no desea que se muestre la fuente stub, conserve la clase en la vista Clases con inspección desactivada y deje activado Paso inteligente.

Archivos fuente de stub

Si la fuente stub se ha generado a partir de archivos para los que existe una fuente, compruebe la vía de acceso a archivos fuente. El depurador busca archivos fuente en la vía de acceso a archivos fuente. La vía de acceso a los archivos fuente se describe en [“Vía de acceso a archivos fuente” en la página 4-9](#). El archivo .java que se depura debe existir en una bifurcación que sea equivalente a su nombre de paquete.

Por ejemplo, si la vía de acceso a archivos fuente contiene un elemento:

```
c:\MyProjects\Test\src
```

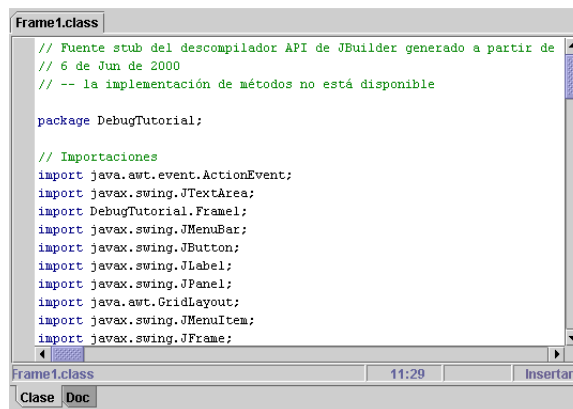
y el archivo .java se encuentra en un paquete llamado mypackage, el depurador espera que el archivo .java exista en el directorio:

```
c:\MyProjects\Test\src\mypackage
```

El nombre del paquete se agrega al nombre del elemento fuente. Si cuenta con múltiples elementos fuente, el depurador intentará localizarlos todos mediante el sistema explicado anteriormente. Si el depurador no puede localizar el archivo fuente, genera una fuente stub.

Se muestra un archivo fuente stub en el panel de contenidos. Contiene una cabecera y los stubs del método.

Figura 8.8 Ejemplo de archivo fuente stub



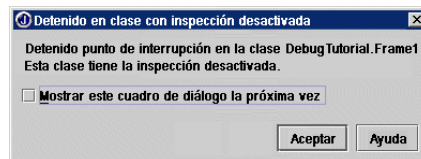
Puntos de interrupción y configuración de Inspección desactivada

Cuando se define un punto de interrupción en una clase, en la vista Clases con inspección desactivada, se redefine la configuración de inspección y se detiene provisionalmente la clase, porque se ha indicado al depurador que acuda a ese punto.

El cuadro de diálogo de advertencia Detenido en clase con inspección desactivada se muestra si:

- El paso inteligente se encuentra activado y, además, si
- La opción Advertir de punto de interrupción en clase con inspección desactivada se encuentra activada en la ficha Depurar del cuadro de diálogo Propiedades de ejecución.

Figura 8.9 Cuadro de diálogo Detenido en clase con inspección desactivada

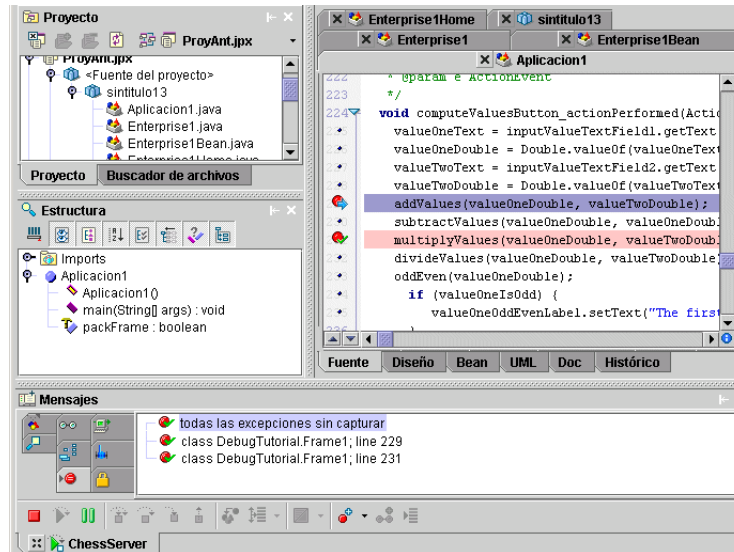


Si esto ocurre, cuando se efectúa la inspección después de alcanzar el punto de interrupción, el depurador sale de la clase. Para permanecer en la clase, desactive el paso inteligente y utilice los botones de inspección.

Puntos de interrupción

Cuando la ejecución encuentra un punto de interrupción, el programa se interrumpe y el depurador muestra en el editor, la línea que contiene el punto de interrupción. Puede utilizar entonces el depurador para ver el estado del programa. Los puntos de interrupción son flexibles, ya que pueden definirse antes de comenzar la ejecución de programa o en cualquier momento en que el depurador tiene el control. Definiendo puntos de interrupción en áreas potencialmente problemáticas del código fuente, es posible ejecutar el programa sin pausa hasta que su ejecución llegue a una posición que desee depurar.

Los puntos de interrupción se presentan y se modifican en la vista Puntos de interrupción de datos y código. Se muestra el tipo de punto de interrupción y su estado, así como información relacionada con el tipo de punto de interrupción, como el número de línea, el nombre de la clase y del método. Puede activar, desactivar, añadir y eliminar puntos de interrupción desde el menú contextual. Esta información se encuentra disponible desde el comando de menú Ejecutar|Puntos de interrupción antes de la depuración. Observe que el cuadro de diálogo Puntos de interrupción cuenta con una barra de herramientas para facilitar la adición, la eliminación y la activación de puntos de interrupción.

Figura 8.10 Vista Puntos de interrupción de datos y código

Definición de puntos de interrupción


Los puntos de interrupción interprocesal, del método, la excepción, el campo y la clase son características de las ediciones Developer y Enterprise de JBuilder.


Es posible definir puntos de interrupción de línea, excepción, clase, método, campo e interprocesales en el depurador:

- Los puntos de interrupción de línea se establecen en una línea determinada del código fuente Java o distinto de Java. El depurador se interrumpe en esa línea.
- Los puntos de interrupción por excepción hacen que el depurador se detenga cuando está a punto de lanzarse la excepción especificada.
- Los puntos de interrupción de clase hacen que el depurador se detenga cuando se llama a un método de la clase especificada o cuando se instancia la clase especificada.
- Los puntos de interrupción de método hacen que el depurador se detenga cuando se llama al método especificado de la clase especificada.
- Los puntos de interrupción de campo hacen que el depurador se detenga cuando se va a leer o escribir en el campo especificado. Un campo es una variable Java que está definida en un objeto Java.
- Un punto de interrupción interprocesal hace que se detenga el depurador cuando un método o el método especificado en la clase especificada se inspecciona en un proceso aparte.

Definición de puntos de interrupción de línea

Los puntos de interrupción de línea hacen que el depurador se detenga cuando alcanza una línea de código determinada. Los puntos de interrupción de línea se pueden definir en código fuente Java o distinto de Java. Se pueden definir directamente en el editor o por medio del cuadro de diálogo Añadir punto de interrupción de línea.

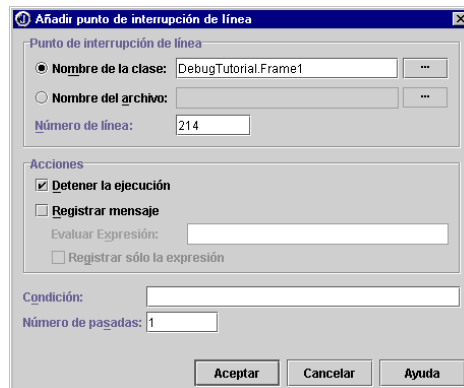
Para definir un punto de interrupción de línea en el código fuente, haga clic en el margen izquierdo de la línea en la que se desea establecer el punto de interrupción. También puede pulsar *F5* en una línea de código fuente para conmutar un punto de interrupción o haga clic con el botón derecho y elija Conmutar punto de interrupción. Cuando el depurador dispone del foco, aparecen pequeños puntos azules  en el editor, a la izquierda de las líneas de código ejecutable, indicando que el punto de interrupción se puede definir en esas líneas.

Las definiciones de puntos de interrupción en las líneas de comentarios, en las declaraciones y en otras líneas no ejecutables de código no son válidos. Los puntos de interrupción no válidos se indican mediante  en el margen del editor cuando se ejecuta el programa.

Para definir un punto de interrupción de línea por medio del cuadro de diálogo Añadir punto de interrupción de línea, siga las instrucciones adecuadas a su caso:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar/Añadir punto de interrupción y seleccione Añadir punto de interrupción de línea. Observe que el cuadro de diálogo cuenta con una barra de herramientas para facilitar la adición y eliminación de puntos de interrupción.
- Durante la sesión de depuración, pulse la flecha abajo situada a la derecha del botón Añadir punto de interrupción de la barra de herramientas del depurador y seleccione Añadir punto de interrupción de línea.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista de puntos de interrupción de código y datos y seleccione Añadir punto de interrupción de línea.

Aparece el cuadro de diálogo Añadir punto de interrupción de línea.



Añadir punto de interrupción de línea

Punto de interrupción de línea

☒ Nombre de la clase: ...

☐ Nombre del archivo: ...

Número de línea:

Acciones

☒ Detener la ejecución

☐ Registrar mensaje

Evaluar Expresión:

☐ Registrar sólo la expresión

Condición:

Número de pasadas:

Aceptar Cancelar Ayuda

Defina el punto de interrupción de línea, escogiendo entre las siguientes opciones:

- 1 Si se está definiendo un punto de interrupción en un archivo `.java`, utilice el campo Nombre de clase. Si el punto de interrupción se encuentra en un archivo que no es `.java`, utilice el campo Nombre del archivo.
 - Si se trata de un archivo `.java`, escriba el nombre o pulse el botón de puntos suspensivos (...). para acceder a un archivo `.java`.
 - Si no se trata de un archivo `.java`, pulse el botón puntos suspensivos (...) para acceder al archivo.
- 2 En el campo Número de línea, escriba el número de la línea en la que se va a definir el punto de interrupción.
- 3 Seleccione las acciones del punto de interrupción. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Si desea más información, consulte [“Definición de las acciones de puntos de interrupción” en la página 8-57](#). (La acciones son una característica de las ediciones Developer y Enterprise de JBuilder).
- 4 En el campo Condición, asigne, si existe, la condición de punto de interrupción para este punto. Si desea más información, consulte [“Creación de puntos de interrupción condicionales” en la página 8-59](#).
- 5 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Si desea más información, consulte [“Utilización de puntos de interrupción por número de pasadas” en la página 8-60](#).
- 6 Haga clic en Aceptar para cerrar el cuadro de diálogo.



Si el punto de interrupción es válido (se establece en una línea de código ejecutable), la línea se resalta y aparece un icono de círculo rojo con una marca de selección en el margen izquierdo de la línea del punto de interrupción.

Definición de puntos de interrupción por excepción

Los puntos de interrupción por excepción son una característica de las ediciones de Developer y Enterprise de JBuilder.

Los puntos de interrupción por excepción hacen que el depurador se detenga cuando está a punto de lanzarse la excepción especificada. El depurador puede detenerse sobre excepciones capturadas o sin capturar. Para fijar un punto de interrupción por excepción, utilice el cuadro de diálogo Añadir punto de interrupción por excepción.

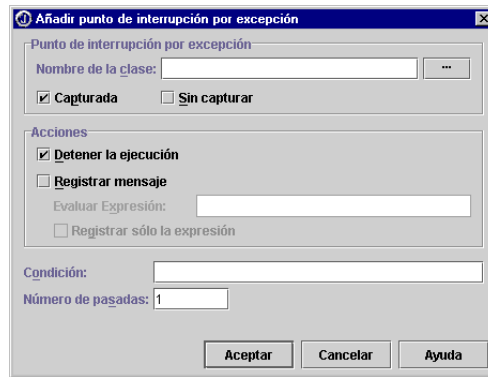
Para abrir el cuadro de diálogo Añadir punto de interrupción por excepción, haga una de las siguientes cosas:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar/Añadir punto de interrupción y seleccione Añadir punto de interrupción por excepción.
- Durante la sesión de depuración, pulse la flecha abajo situada a la derecha del botón Añadir punto de interrupción de la barra de herramientas del depurador y seleccione Añadir punto de interrupción por excepción.



- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista Puntos de interrupción de datos y código y seleccione Añadir punto de interrupción por excepción.

Aparece el cuadro de diálogo Añadir punto de interrupción por excepción.



Para definir un punto de interrupción por excepción:

- 1 En el campo Nombre de la clase, escriba el nombre del archivo de clase de excepción en que se va a detener el depurador. Puede escribir el nombre o pulsar el botón de puntos suspensivos (...). para acceder a un archivo .class.
- 2 Seleccione cuándo debe detenerse el depurador:
 - Seleccione la opción Capturada, de modo que el depurador se detenga cuando se capture la excepción especificada.
 - Seleccione la opción Sin capturar, de modo que el depurador se detenga cuando no se capture la excepción especificada.También es posible elegir las dos opciones para que el depurador se detenga en ambos casos.
- 3 Elija qué acciones realizará el depurador. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Si desea más información, consulte [“Definición de las acciones de puntos de interrupción” en la página 8-57](#). (Las acciones son una característica de las ediciones Developer y Enterprise de JBuilder).
- 4 En el campo Condición, si existe alguna, asígnela al punto de interrupción. Si desea más información, consulte [“Creación de puntos de interrupción condicionales” en la página 8-59](#).
- 5 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Si desea más información, consulte [“Utilización de puntos de interrupción por número de pasadas” en la página 8-60](#).
- 6 Haga clic en Aceptar para cerrar el cuadro de diálogo.

Los puntos de interrupción de clase son una característica de las ediciones de Developer y Enterprise de JBuilder.

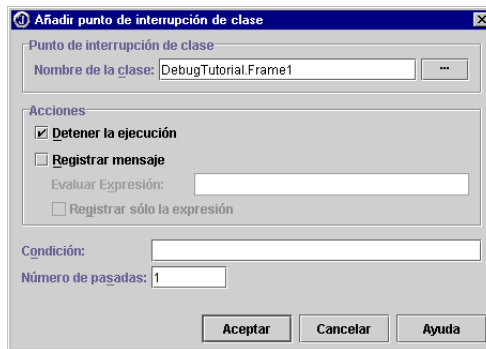
Definición de puntos de interrupción de clase

Los puntos de interrupción de clase hacen que el depurador se detenga cuando se llama a un método de la clase especificada o cuando se instancia la clase especificada. Para definir un punto de interrupción de clase, utilice el cuadro de diálogo Añadir punto de interrupción de clase.

Para abrir el cuadro de diálogo Añadir punto de interrupción de clase, haga una de las siguientes cosas:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar|Añadir punto de interrupción y seleccione Añadir punto de interrupción de clase.
- Durante la sesión de depuración, pulse la flecha abajo situada a la derecha del botón Añadir punto de interrupción de la barra de herramientas del depurador y seleccione Añadir punto de interrupción de clase.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista Puntos de interrupción por datos y código y seleccione Añadir punto de interrupción de clase.

Aparece el cuadro de diálogo Añadir punto de interrupción de clase.



Para definir un punto de interrupción de la clase:

- 1 En el campo Nombre de la clase, escriba el nombre del archivo de clase en el que desea que se detenga el depurador. Puede escribir el nombre o pulsar el botón de puntos suspensivos (...) para acceder a un archivo .class.
- 2 Elija qué acciones realizará el depurador. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Si desea más información, consulte [“Definición de las acciones de puntos de interrupción” en la página 8-57](#). (Las acciones son una característica de las ediciones Developer y Enterprise de JBuilder).
- 3 En el campo Condición, si existe alguna, asígnela al punto de interrupción. Si desea más información, consulte [“Creación de puntos de interrupción condicionales” en la página 8-59](#).
- 4 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Si desea más

información, consulte “Utilización de puntos de interrupción por número de pasadas” en la página 8-60.

5 Haga clic en Aceptar para cerrar el cuadro de diálogo.

Definición de puntos de interrupción de método

Los puntos de interrupción de método hacen que el depurador se detenga cuando se llama al método especificado de la clase especificada. Para configurar un punto de interrupción de método, utilice el cuadro de diálogo Añadir punto de interrupción de método.

Para abrir el cuadro de diálogo Añadir punto de interrupción de método, haga una de las siguientes cosas:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar|Añadir punto de interrupción y seleccione Añadir punto de interrupción de método.
- Durante la sesión de depuración, pulse la abajo situada a la derecha del botón Añadir punto de interrupción de la barra de herramientas del depurador y seleccione Añadir punto de interrupción de método.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista de puntos de interrupción de código y datos y elija Añadir punto de interrupción de método.

Aparece el cuadro de diálogo Añadir punto de interrupción de método.

Para definir un punto de interrupción de método:

- 1 En el campo Nombre de la clase, escriba el nombre del archivo de clase que contiene el método en el que desea que se detenga el depurador. Puede escribir el nombre o pulsar el botón de puntos suspensivos (...) para acceder a un archivo .class.
- 2 En el campo Nombre del método, escriba el nombre del método en el que desea se detenga el depurador. Haga clic sobre el botón de puntos suspensivos para hallar el método que desea utilizar.

Los puntos de interrupción de método son una característica de las ediciones de Developer y Enterprise de JBuilder.




- 3 En el campo Argumentos del método, introduzca una lista de argumentos del método separados por comas. Esto hace que el depurador se detenga únicamente cuando coinciden el nombre del método y la lista de argumentos. Esto resulta útil con los métodos sobrecargados. Si utiliza el visualizador para añadir el método, los argumentos de método se rellenan automáticamente. Si no se especifican argumentos, el depurador detiene todos los métodos con el nombre de método especificado.
- 4 Elija qué acciones realizará el depurador. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Si desea más información, consulte [“Definición de las acciones de puntos de interrupción” en la página 8-57](#). (Las acciones son una característica de las ediciones Developer y Enterprise de JBuilder).
- 5 En el campo Condición, si existe alguna, asígnela al punto de interrupción. Si desea más información, consulte [“Creación de puntos de interrupción condicionales” en la página 8-59](#).
- 6 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Si desea más información, consulte [“Utilización de puntos de interrupción por número de pasadas” en la página 8-60](#).
- 7 Haga clic en Aceptar para cerrar el cuadro de diálogo.

Definición de puntos de interrupción de campo

Los puntos de interrupción de campo hacen que el depurador se detenga cuando se va a leer o escribir, según lo elegido, en el campo especificado. Un campo es una variable Java que está definida en un objeto Java. En el siguiente ejemplo:

```
class Test {
    private int x;
    private Object y;
}
Test myTest = new Test();
```

`myTest` es una variable. Las variables Java `x` e `y` representan campos.

Para añadir un punto de interrupción de campo, haga clic con el botón derecho del ratón en una variable de campo de la vista Hilos, pila de llamadas y datos y seleccione Añadir punto de interrupción de campo. El punto de interrupción se añade automáticamente a la vista Puntos de interrupción de datos y código. Un punto de interrupción de campo se indica con .

Para controlar si el depurador se interrumpe en una acción de lectura o escritura, abra la vista Puntos de interrupción de datos y código. Haga clic con el botón derecho del ratón en el punto de interrupción de campo que acaba de definir. Por defecto, los comandos Interrumpir al leer e Interrumpir al escribir del menú contextual se encuentran activados, lo que significa que el depurador se detendrá cuando se vaya a leer o escribir el campo especificado. Puede desactivar una o ambas opciones y permitir que continúe el depurador, en lugar de detener el depurador cuando está a punto de leer o escribir un campo.

Los puntos de interrupción de campo son una característica de las ediciones de Developer y Enterprise de JBuilder.

Los puntos de interrupción interprocesales son características de las ediciones Developer y Enterprise de JBuilder.

Configuración de un punto de interrupción interprocesal

Un punto de interrupción interprocesal detiene el depurador cuando se inspecciona, en un proceso aparte, un método o el método especificado en la clase especificada. Esto permite inspeccionar un proceso servidor desde un proceso cliente, en lugar de tener que configurar puntos de interrupción en el cliente y en el servidor. Por lo general, se configura un punto de interrupción de línea en el cliente y un punto de interrupción interprocesal en el servidor. Si desea seguir un tutorial que explique paso a paso la inspección interprocesal, consulte el [Capítulo 21, “Tutorial: Depuración remota”](#).

Para activar un punto de interrupción interprocesal configurado en un proceso de servidor:

- 1 Inicie el proceso de servidor en el ordenador remoto en modo depuración.
- 2 En el ordenador cliente y desde JBuilder, conéctese con el servidor que ya está en ejecución en el ordenador remoto.
- 3 Configure un punto de interrupción en el código del cliente y comience a depurar el cliente. Cuando llegue al punto de interrupción, inspeccione el código del servidor. No utilice Omitir inspección; la omisión no se detiene en el punto de interrupción interprocesal.

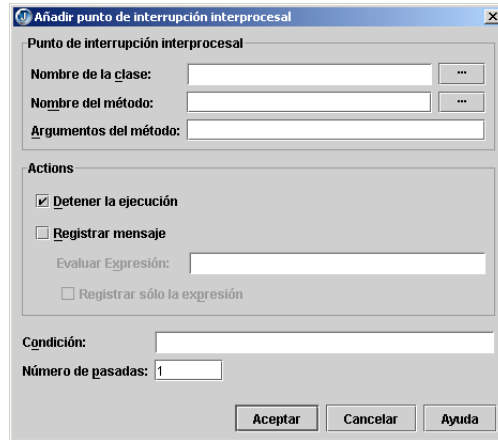
Nota Puede utilizar los puntos de interrupción interprocesal para depurar de manera local, por ejemplo, una aplicación cliente/servidor que se ejecuta en un ordenador.

Para configurar un punto de interrupción interprocesal, utilice el cuadro de diálogo Añadir punto de interrupción interprocesal. Para abrir el cuadro de diálogo Añadir punto de interrupción interprocesal, haga una de las siguientes cosas:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar!Añadir punto de interrupción y seleccione Añadir punto de interrupción interprocesal.
- Durante la sesión de depuración, pulse la flecha de lista desplegable situada a la derecha del botón Añadir punto de interrupción; de la barra de herramientas del depurador y seleccione Añadir punto de interrupción interprocesal.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista Puntos de interrupción de datos y de código y seleccione Añadir punto de interrupción interprocesal.



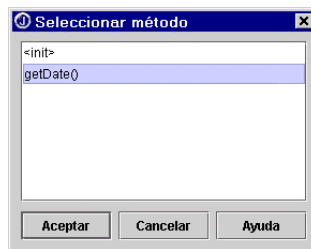
Aparece el cuadro de diálogo Añadir punto de interrupción interprocesal.



Si desea seguir un tutorial que explique paso a paso la inspección interprocesal, consulte el [Capítulo 21](#), “Tutorial: Depuración remota”.

Para configurar un punto de interrupción interprocesal:

- 1 En el campo Nombre de clase, escriba el nombre de la clase del servidor que contiene el método en el que desea se detenga el depurador. Puede escribir el nombre o pulsar el botón de puntos suspensivos (...). para acceder a un archivo `.class`.
- 2 En el campo Nombre del método, escriba el nombre del método en el que desea se detenga el depurador. Utilice el botón Examinar para mostrar el cuadro de diálogo Seleccionar método, en el cual se puede acceder a la lista de métodos disponibles en la clase seleccionada.




No es necesario escribir el nombre del método. Si no se especifica un nombre de método, el depurador se detiene en todas las llamadas a métodos en la clase especificada.

Nota

No se puede seleccionar un método si la clase seleccionada contiene errores de sintaxis o de compilación.

- 3 En el campo Argumentos del método, introduzca una lista de argumentos del método separados por comas. Esto hace que el depurador se detenga cuando coinciden el nombre del método y la lista de argumentos. Esto resulta útil con los métodos sobrecargados.

- Si no se establece ningún argumento, el depurador se detiene en todos los métodos cuyo nombre coincida con el especificado.
 - Si se elige un nombre de método del cuadro de diálogo Seleccionar método, el campo Argumentos de método se rellena automáticamente.
- 4 Elija qué acciones realizará el depurador. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Si desea más información, consulte [“Definición de las acciones de puntos de interrupción” en la página 8-57](#). (Las acciones son una característica de las ediciones Developer y Enterprise de JBuilder).
 - 5 En el campo Condición, si existe alguna, asígnela al punto de interrupción. Si desea más información, consulte [“Creación de puntos de interrupción condicionales” en la página 8-59](#).
 - 6 En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Si desea más información, consulte [“Utilización de puntos de interrupción por número de pasadas” en la página 8-60](#).
 - 7 Haga clic en Aceptar para cerrar el cuadro de diálogo.
 - 8 Fije un punto de interrupción de línea en el cliente, en el método que llama al punto de interrupción interprocesal.
- 
- 9 Pulse el botón Inspeccionar de la barra de herramientas del depurador para inspeccionar el método con punto de interrupción del servidor. (Si utiliza Omitir inspección, el depurador no se detendrá).

Definición de las propiedades de los puntos de interrupción

Después de crear un punto de interrupción se pueden definir y modificar sus propiedades. Para definir las propiedades de punto de interrupción,

- 1 Abra la vista Puntos de interrupción de datos y código.
- 2 Elija el punto de interrupción que desea configurar. Haga clic con el botón derecho y seleccione Propiedades de punto de interrupción.

Se muestra el cuadro de diálogo de Propiedades de punto de interrupción.

Nota El cuadro de diálogo Propiedades de punto de interrupción contiene las mismas opciones que el cuadro de diálogo utilizado para crearlo.

- 3 Es posible cambiar las siguientes propiedades:
 - Acciones

Las acciones que se deben realizar cuando se alcanza un punto de interrupción. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Si desea más información, consulte [“Definición de las acciones de puntos de interrupción” en la página 8-57](#).
 - Condición

La condición de este punto de interrupción, si la hay. Si desea más información, consulte [“Creación de puntos de interrupción condicionales” en la página 8-59.](#)

- **Número de pasadas**

El número de veces que se debe alcanzar el punto de interrupción para que se active. Si desea más información, consulte [“Utilización de puntos de interrupción por número de pasadas” en la página 8-60.](#)

Para mostrar el cuadro de diálogo Propiedades de punto de interrupción en modo de sólo lectura, coloque el ratón en el margen interior contiguo a la línea con punto de interrupción. Pulse *Ctrl* junto con el botón derecho del ratón. Se muestran las propiedades principales del punto de interrupción, pero no es posible modificarlas. Los campos Acciones, Condición y Número de pasadas pueden modificarse.

Definición de las acciones de puntos de interrupción

**Es una función de
JBuilder Developer y
Enterprise.**

Es posible seleccionar varias acciones que deben realizarse cuando llega un punto de ejecución. El depurador puede:

- Detener la ejecución del programa y mostrar un mensaje en la barra de estado del depurador (la barra por defecto).
- Registrar un mensaje en la vista Salida, entrada y errores de consola.
- Evaluar una expresión y guardar los resultados.

Las acciones se definen en la zona central del cuadro de diálogo Puntos de interrupción.

Figura 8.11 Acciones de punto de interrupción



Detención de la ejecución del programa

Para detener la ejecución del programa cuando se alcance el punto de interrupción especificado, seleccione la opción por defecto Detener la ejecución. Si detiene la ejecución del programa, el depurador se detendrá en el punto de interrupción especificado y se mostrará un mensaje de estado en la barra de herramientas del depurador y la ventana Puntos de interrupción de datos y código.

El mensaje que aparezca en la barra de estado dependerá del tipo de punto de interrupción. El siguiente ejemplo muestra el mensaje de la barra de estado que aparece para un punto de interrupción de línea.

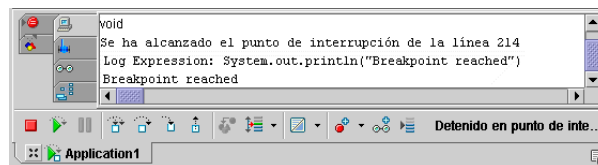
Figura 8.12 Mensaje de la barra de estado de punto de interrupción

Registro de mensajes

Para registrar un mensaje en la vista de errores, de entrada y de salida de consola cuando se alcanza un punto de interrupción, seleccione la opción Registrar mensaje y escriba un mensaje en el cuadro Evaluar Expresión. Cuando el depurador alcance el punto de interrupción seleccionado, se registrará un mensaje en la vista. Si además se selecciona la opción Detener la ejecución, el programa se detendrá. Si no, continúa la ejecución del programa.

Registro de un mensaje con una sentencia println

Es posible utilizar sentencias `println` para registrar mensajes de salida. El siguiente ejemplo muestra un mensaje en la vista de puntos de interrupción por datos y código que se registró con la sentencia `System.out.println("Punto de interrupción alcanzado")`. La sentencia se introdujo en el cuadro de entrada Evaluar expresión y se activó la opción Registrar mensaje.

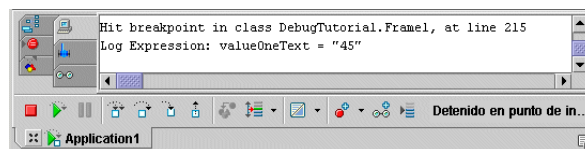


Registro de un mensaje con evaluación de expresiones

También es posible utilizar la evaluación de expresiones, en lugar de las sentencias `println`, para registrar mensajes durante la depuración. Para hacerlo, escriba una expresión en el campo de entrada Evaluar Expresión. El depurador evalúa la expresión cuando se alcanza el punto de interrupción y escribe los resultados de la evaluación en la vista Salida, entrada y errores de la consola. La expresión puede ser cualquier sentencia válida en lenguaje Java.

Esta opción sólo está disponible si se selecciona la opción Registrar mensaje. Puede detener la ejecución del programa cuando se evalúa una expresión si selecciona la opción Detener la ejecución.

El siguiente ejemplo muestra los resultados de una expresión `valueOneText`. La expresión se introduce en el campo Evaluar expresión, con la opción Registrar mensaje marcada.



La opción Registrar sólo la expresión permite registrar sólo los resultados de esta expresión, de forma que el registro no se llene con otra información.

Creación de puntos de interrupción condicionales

Cuando se crea un punto de interrupción, la ejecución del programa se interrumpe por defecto cada vez que se llega a él. Sin embargo, el cuadro de diálogo Propiedades de punto de interrupción permite personalizar los puntos de interrupción de forma que se activen sólo bajo determinadas condiciones.

Si se escribe una expresión booleana en el campo Condición, el punto de interrupción pasa a ser condicional y la ejecución del programa se detiene en este punto sólo si la condición es `true`. También se puede basar un punto de interrupción en el número de pasadas, que se especifica en el campo Número de pasadas. Este campo resulta útil para los bucles de depuración. La ejecución del programa se detiene en el punto de interrupción cuando recorre el bucle el número de veces especificado.

Las condiciones se definen en la parte inferior del cuadro de diálogo Puntos de interrupción.

Figura 8.13 Puntos de interrupción condicionales



Definición de la condición de punto de interrupción

El cuadro de edición del cuadro de diálogo Propiedades de punto de interrupción permite escribir una expresión que se evalúa cada vez que alcanza el punto de interrupción durante la ejecución del programa.

- Si la condición da como resultado `true`, el depurador se detendrá en el punto de interrupción si está activada la opción Detener la ejecución.
- Si la evaluación de la expresión da como resultado `false`, el depurador no se detiene en el punto de interrupción.

Los puntos de interrupción condicionales permiten ver cómo se comporta el programa cuando una variable contiene valores comprendidos en un rango concreto o qué ocurre cuando se define un flag (indicador) determinado.

Por ejemplo, suponga que desea que un punto de interrupción se interrumpa en una línea de código sólo cuando la variable `mediumCount` supere 10. Para hacer esto:

- 1 Defina un punto de interrupción en una línea de código, haciendo clic a la izquierda de la línea en el editor.
- 2 Haga clic con el botón derecho y seleccione Propiedades de punto de interrupción.
- 3 Introduzca la expresión siguiente en el cuadro de edición Condición y haga clic en Aceptar:

```
mediumCount > 10
```

Puede escribir una expresión válida en lenguaje Java en el cuadro de edición Condición, pero todos los símbolos de la expresión deben ser accesibles desde el punto de interrupción.

Utilización de puntos de interrupción por número de pasadas

La condición Número de pasadas del cuadro de diálogo Propiedades de punto de interrupción determina el número de veces que se debe pasar el punto de interrupción para que se active. El Depurador detiene el programa tras encontrar *n* veces el punto de interrupción durante la ejecución del programa. El valor por defecto de *n* es 1.

Los números de pasadas resultan útiles cuando se piensa que un bucle está fallando en la iteración número *n*. Cuando se utiliza el número de pasadas con condiciones booleanas, la ejecución del programa se suspende la *n* vez que la condición tiene el valor `true`.

Desactivación y activación de puntos de interrupción

La desactivación de un punto de interrupción lo oculta de la ejecución actual del programa. Si desactiva un punto de interrupción, todos sus valores permanecen definidos, pero durante la ejecución del programa se obviará el punto. El programa no se detiene en los puntos de interrupción desactivados. La desactivación de un punto de interrupción resulta útil si ha definido un punto de interrupción condicional que no necesita utilizar ahora, pero que podría necesitar más adelante.

- Para desactivar y activar un solo punto de interrupción, haga clic en él con el botón derecho del ratón en la vista Puntos de interrupción de datos y código o el editor. Seleccione Activar punto de interrupción para conmutar el punto de interrupción. Si este conmutador está desactivado, también lo está el punto de interrupción, y la ejecución no se detiene al llegar a él. Si está activado, también lo está el punto de interrupción, y la ejecución se detiene al llegar a él. Los puntos de interrupción están activados por defecto.
- Para desactivar o activar todos los puntos de interrupción definidos para una sesión de depuración, abra la vista de Puntos de interrupción de datos y código. Haga clic con el botón derecho del ratón en una zona vacía de la vista y seleccione Desactivar todos o Activar todos.

También es posible desactivar los puntos de interrupción para una configuración de ejecución determinada. Si hay más de una configuración de ejecución definida en el proyecto actual, dispondrá del comando Desactivar en la configuración cuando haga clic con el botón derecho en un punto de interrupción. Basta con elegir las configuraciones para las que se desea desactivar el punto de interrupción. Cuando se efectúe la depuración con la configuración especificada, el punto de interrupción elegido estará desactivado.

Eliminación de puntos de interrupción

Cuando ya no necesite examinar el código correspondiente a un punto de interrupción, puede eliminar la interrupción de la sesión. Los puntos de interrupción de línea se pueden eliminar en el editor. Los demás tipos de puntos de interrupción se eliminan con la vista de Puntos de interrupción de datos y código.

Tenga en cuenta que no es posible eliminar el punto de interrupción por defecto, definido en `todas las excepciones sin capturar`. Sin embargo, se puede desactivar.

Utilice uno de los métodos siguientes para borrar puntos de interrupción:

- En el editor, coloque el cursor en la línea que contenga el punto de interrupción, pulse *F5* o haga clic con el botón derecho del ratón y seleccione *Conmutar punto de interrupción*.
- En la vista Puntos de interrupción de datos y código, resalte el punto de interrupción que desea eliminar, haga clic con el botón derecho del ratón y seleccione *Eliminar punto de interrupción* o pulse *Eliminar*.
- Para borrar todos los puntos de interrupción definidos para una sesión de depuración, abra la vista Puntos de interrupción de datos y código. Haga clic con el botón derecho del ratón en una zona vacía de la vista y seleccione *Eliminar todos*.
- Seleccione un grupo de puntos de interrupción en la vista de Puntos de interrupción de datos y código y pulse *Eliminar*.

Advertencia Los puntos de interrupción borrados no pueden recuperarse.

Ubicación de los puntos de interrupción de línea

Si no encuentra un punto de interrupción de línea en el editor, puede utilizar la vista Puntos de interrupción de datos y código para encontrar rápidamente su ubicación en el código fuente.

Para encontrar un punto de interrupción de línea:

- 1 Seleccione un punto de interrupción de línea en la vista Puntos de interrupción de datos y código.
- 2 Haga clic con el botón derecho y seleccione *Ir al punto de interrupción*. También puede hacer doble clic sobre el punto de interrupción seleccionado.

El editor muestra la ubicación del punto de interrupción.

Examen de los valores de datos del programa

A pesar de que puede descubrir muchas cosas interesantes acerca del programa ejecutándolo y recorriéndolo paso a paso, normalmente necesitará examinar los valores de las variables del programa para descubrir los errores. Por ejemplo, resulta útil conocer el valor de una variable de índice cuando se recorre un bucle `for`, o los valores de los parámetros que se pasan en una llamada a un método.

Tras detener el programa durante la depuración, puede examinar los valores de las variables de instancia, variables locales, propiedades, parámetros del método y elementos de las matrices.

La evaluación de los datos se realiza dentro de las expresiones. Las expresiones están formadas por constantes, variables y valores en estructuras de datos, que pueden ser combinados con los operadores del lenguaje. De hecho, casi cualquier cosa que se encuentre a la derecha de un operador de asignación puede utilizarse como una expresión de depuración.

JBuilder cuenta con varias funciones que permiten comprobar el estado del programa y se describen en la tabla siguiente.

Tabla 8.26 Características del depurador para ver el estado del programa

Función	Activa
Vista Clases cargadas y datos estáticos	Ver las clases cargadas actualmente por el programa y los datos estáticos que haya en ellas.
Vista Hilos, pilas de llamada y datos	Ver los grupos de hilos del programa. Los grupos de hilos se expanden para mostrar sus hilos y contienen un seguimiento de marcos de pila representando la secuencia actual de llamadas a métodos. Cada marco de pila puede ampliarse para mostrar los elementos de datos en ámbito.
Vista Puntos de observación de datos	Ver los valores actuales de variables de las que se desea efectuar un seguimiento. Los puntos de observación evalúan una expresión en función del contexto actual. Si se cambia de contexto, la expresión se evalúa otra vez, dentro del nuevo contexto. Si ya no está en ámbito no se podrá evaluar.
Cuadros de diálogo: Evaluar/Modificar	Evaluación de expresiones, llamadas a métodos y variables.
ExpressionInsight	Visualizar los valores de las expresiones.

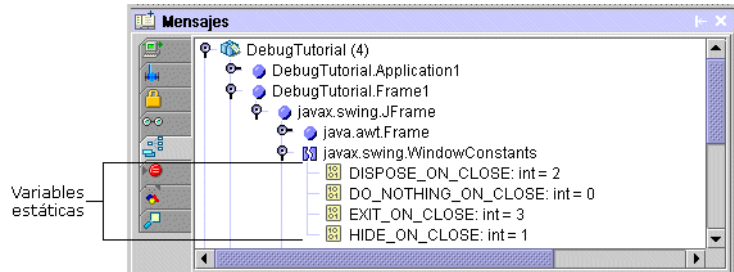
Presentación de las variables en el depurador

Las variables pueden tener diferentes ámbitos, existen variables estáticas, de clase, locales y de miembros. Una variable puede contener un único valor, como por ejemplo un escalar (un solo número) o varios valores, como las matrices.

Visualización de variables estáticas

Las variables estáticas aparecen en la vista Clases cargadas y datos estáticos. Si se amplía el árbol de clases cargadas, aparecen todas las variables estáticas definidas para una clase junto con sus valores. El menú contextual permite definir puntos de observación de estas variables, cambiar los valores de los tipos de datos originales y cambiar el valor de la base de presentación.

Figura 8.14 Vista Clases cargadas y datos estáticos

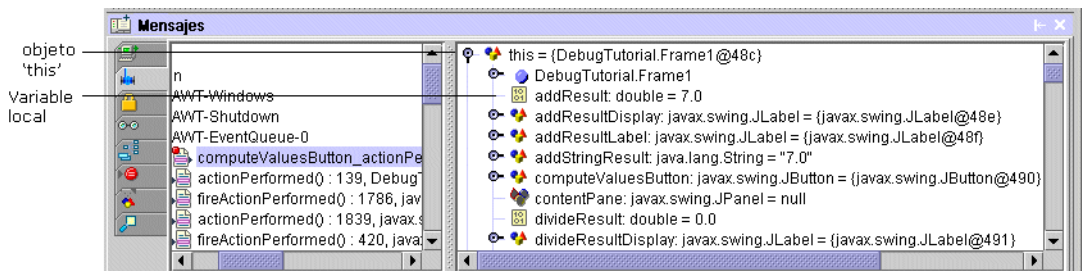


Formato de visualización de variables locales y de miembros

Las variables locales y de miembro se muestran en la vista Hilos, pilas de llamada y datos. Cuando se amplía un grupo de hilos, se muestra la pila de marcos que representan la secuencia de llamada al método actual. Para mostrar variables de miembro, amplíe el objeto `this` desde el interior de la clase. Cuando se amplía el nodo, se ven todas las variables pertenecientes a esa clase y la instanciación con la que está trabajando. Los elementos que aparecen en gris son heredados.

A continuación, se puede utilizar el menú contextual con el fin de definir puntos de observación para estas variables y, si la variable es una matriz, crear un punto de observación de la matriz y averiguar cuántos elementos se mostrarán en la vista. También se puede crear un punto de observación para el objeto `this`.

Figura 8.15 Vista Hilos, pilas de llamada y datos



Nota La ventana de división es una característica de las ediciones Developer y Enterprise de JBuilder.

Cambio de valores de datos

Los valores de datos de las variables se pueden examinar y modificar por medio de las vistas Puntos de observación de datos , Hilos, pilas de llamada y datos y Clases cargadas y datos estáticos.

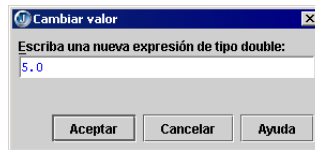
Se puede hacer clic con el botón derecho del ratón y elegir Modificar valor para modificar directamente el valor de una cadena o un tipo de datos primitivo, como el numérico y el booleano. (Esta función pertenece a las ediciones Developer y Enterprise de JBuilder).

Cambiar los valores de las variables

Para modificar el valor de una variable:

- 1 Seleccione la variable cuyo valor desea modificar.
- 2 Haga clic con el botón derecho y seleccione Cambiar valor.

Se muestra el cuadro de diálogo Cambiar valor.



- 3 Introduzca el valor. Su tipo debe coincidir con el del anterior. Las instrucciones del cuadro de diálogo indican el tipo de valor que se espera obtener. Si el nuevo valor es de un tipo distinto, no se cambia. Una constante `String` debe estar rodeada de caracteres " de apertura y cierre, mientras que una constante `char` debe rodearse de caracteres ' de apertura y cierre.

- 4 Pulse Aceptar.

Nota La selección anterior se recuerda y se mantiene. Por ejemplo, el contenido del cuadro de diálogo no varía cuando se vuelve a trazar un nodo, se alcanza un punto de interrupción o se pulsa un botón de inspección.

Para cambiar la base de presentación de una variable numérica, haga clic con el botón derecho del ratón y seleccione Mostrar valor hexadecimal o Mostrar valor decimal. Si el valor se presenta en formato hexadecimal, el comando permite pasar al formato decimal, y viceversa.

Modificación del valor de los objetos/primitivos

Las variables de objeto/primitivo se pueden cortar, copiar y pegar en otros objetos/primitivos por medio de los comandos Cortar, Copiar y Pegar de los menús contextuales. Si un objeto se pega en otra variable, las dos apuntan al mismo objeto. (Los comandos Cortar, Copiar y Pegar son características de las ediciones Developer y Enterprise de JBuilder).


Es una función de
JBuilder Developer y
Enterprise.

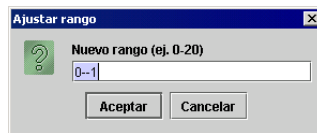
Modificación de los valores de una matriz de variables

Si desea cambiar el valor de un elemento de matriz, haga clic con el botón derecho del ratón sobre el elemento que desea modificar y elija Modificar. Para más información consulte [“Cambio de valores de datos” en la página 8-64](#).



También es posible modificar la forma en que se muestra la matriz. Puede ver los detalles de la matriz si la amplía. Sin embargo, es posible que se muestren tantos elementos que se vea obligado a desplazarse dentro de la vista para ver todos los valores. Sin embargo, para verlos con más facilidad, puede reducir o aumentar el número de elementos mostrados con el cuadro de diálogo Ajustar rango. Por defecto, sólo se muestran los primeros 50 elementos de una matriz.

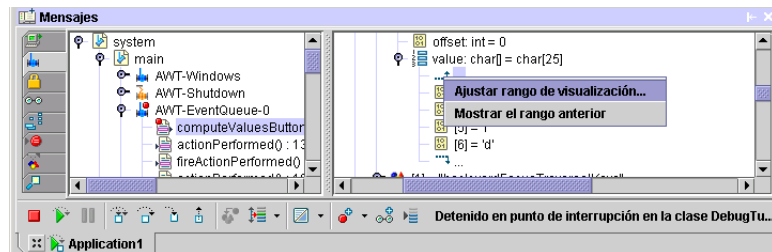
Para reducir el número de elementos de matriz que se muestran:

- 1 Haga clic con el botón derecho del ratón en la matriz (el elemento de la vista precedido por ) y elija Ajustar rango de visualización.
- 2 Se muestra el cuadro de diálogo Ajustar rango.



- 3 Introduzca los elementos de matriz que desea ver.
- 4 Pulse Aceptar.

Cuando se ve una matriz en la vista del depurador, sólo se muestran los elementos especificados de esa matriz. Al ampliar la matriz, se ve el icono Rango anterior  o Rango siguiente . Se puede hacer clic con el botón derecho en los puntos suspensivos junto al icono para ver un menú contextual, tal y como aparece en la siguiente imagen:



Los comandos del menú contextual se explican en la siguiente tabla.

Tabla 8.27 Menú contextual del elemento de la matriz

Elemento del menú contextual	Descripción
Ajustar rango de visualización	Abre el cuadro de diálogo Ajustar rango, en el que se pueden cambiar los elementos de la matriz que aparecen.
Rango anterior – Disponible si se hace clic con el botón derecho en el icono Rango anterior	Muestra el rango anterior de la matriz. El número de elementos que se muestran coincide con el número de elementos que aparecen actualmente.
Rango siguiente – Disponible si se hace clic con el botón derecho en el icono Rango siguiente	Muestra el rango siguiente de la matriz. El número de elementos que se muestran coincide con el número de elementos que aparecen actualmente.

También es posible ocultar o mostrar un valor nulo en una variable matriz. Esto resulta muy útil cuando se depura un objeto con un mapa de direccionamiento calculado. Para activar esta característica, haga clic con el botón derecho del ratón sobre una matriz de tipo `Object` y elija Mostrar/Ocultar valor nulo. (Esta función pertenece a las ediciones Developer y Enterprise de JBuilder).

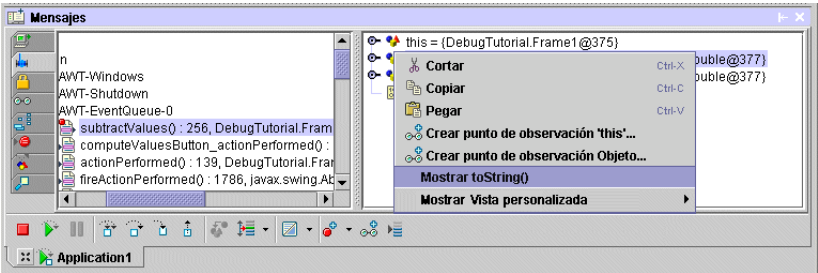
Nota En este cuadro de diálogo se conserva la configuración anterior. Por ejemplo, el contenido del cuadro de diálogo no varía cuando se vuelve a trazar un nodo, se alcanza un punto de interrupción o se pulsa un botón de inspección.

Presentación de objetos como una Cadena

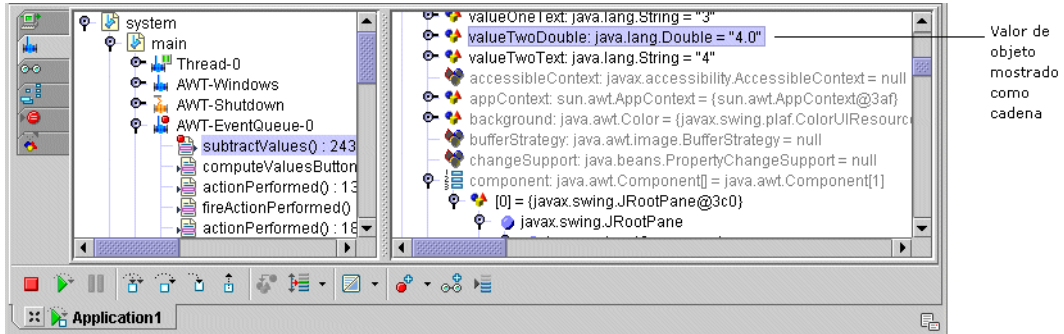
Es una función de
JBuilder Developer y
Enterprise.

El comando Mostrar toString() ejecuta el método toString() en el objeto seleccionado (incluido el objeto `this`) y muestra la cadena resultante. Por ejemplo, si el objeto seleccionado es `City`, el comando Mostrar toString() mostrará el valor del objeto, como `San Francisco` en lugar de `com.mycode.City@391`. Este comando se encuentra disponible al seleccionar un objeto en la vista Puntos de observación de datos y en la vista Hilos, pilas de llamada y datos. (Ediciones Developer y Enterprise de JBuilder).

Para mostrar un objeto como cadena, haga clic con el botón derecho del ratón en él, en la vista Hilos, pila de llamadas y datos, y elija Show toString().



El valor del objeto se muestra en forma de cadena.



Utilización de un visualizador personalizado en un objeto

**Es una función de
JBuilder Developer y
Enterprise.**

Un visualizador personalizado permite personalizar la información que muestra el depurador relativa a un objeto o matriz de objetos. Como mínimo, un visualizador personalizado debe contener un método `public, static` con un valor devuelto o una matriz de valores devueltos. El único requisito es que el parámetro del método sea del tipo `Object` o una matriz de objetos.

Nota En un solo objeto o una matriz de objetos puede haber varias vistas personalizadas.

Por ejemplo, el siguiente ejemplo de código imprime un valor `String`: Esto es un test.

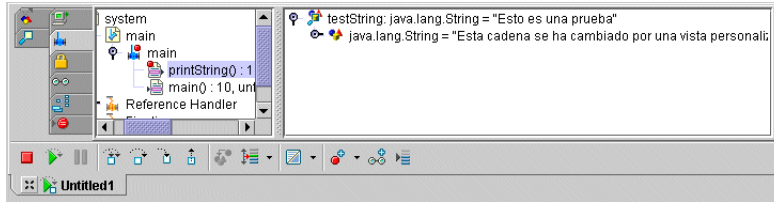
```
public class Untitled1 {
    public static String testString = "Esto es un test.";
    public static void main(String[] args) {
        printString(testString);
    }
    public static void printString(String testString) {
        System.out.println(testString);
    }
}
```

El siguiente visualizador personalizado cambia la apariencia del valor `String` en el depurador por: Esta cadena se ha cambiado por una vista personalizada.

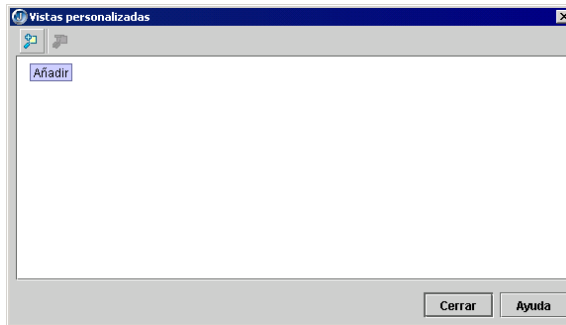
```
public class customView{
    public static String changeString(String testString) {
        testString = "Esta cadena se ha cambiado por una vista personalizada.";
        return testString;
    }
}
```

A continuación aparece un ejemplo de cómo se muestra este visualizador personalizado en la vista Hilos, pilas de llamada y datos. Observe que la

información personalizada sólo se muestra una vez que se ha ampliado el objeto; no se muestra en el nivel superior del objeto.

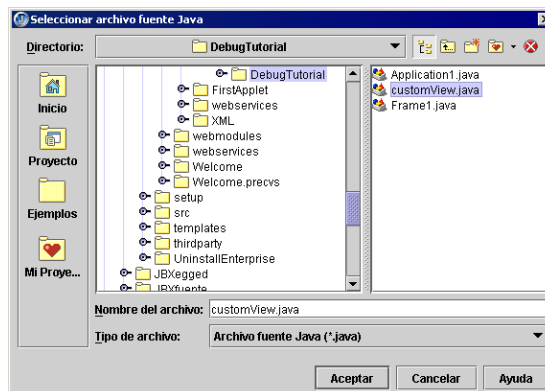


Se puede crear un visualizador personalizado desde el cuadro de diálogo Vistas personalizadas (Ejecutar/Vistas personalizadas) o desde la pestaña Vistas personalizadas del depurador. Observe que el cuadro de diálogo cuenta con una barra de herramientas para facilitar la adición y eliminación de vistas personalizadas, tal y como se muestra más adelante:



Para crear una vista personalizada:

- 1 Antes de iniciar el depurador, seleccione Ejecutar/Vistas personalizadas para abrir el cuadro de diálogo Vistas personalizadas. Una vez que está depurando, puede hacer clic en la pestaña Vistas personalizadas del depurador.
- 2 Haga clic con el botón derecho en una zona vacía del cuadro de diálogo o de la vista y seleccione Añadir. Aparece el cuadro de diálogo Seleccionar archivo fuente Java:

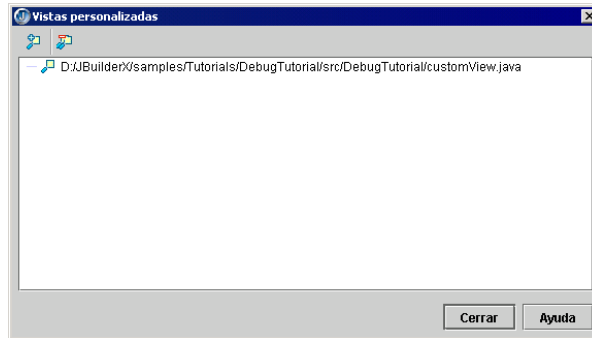


Escriba un nombre de archivo y haga clic en Aceptar. (El nombre de archivo debe tener la extensión `.java`). JBuilder crea el archivo en el directorio seleccionado. Observe que el directorio por defecto es el directorio raíz del proyecto.

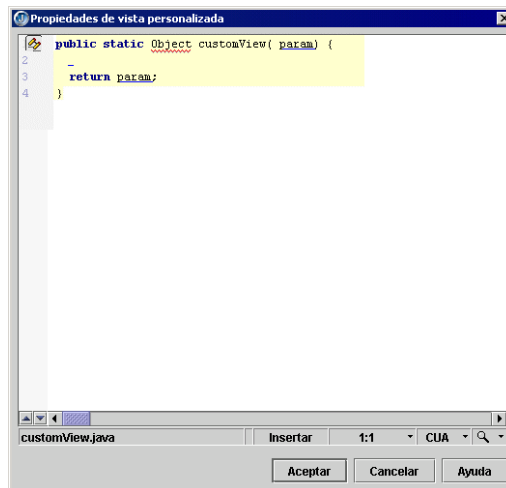
Importante

No coloque la clase del visualizador personalizado en la vía de acceso a fuentes del proyecto. Si lo hace, se incluye en el recopilatorio del proyecto.

Una vez que JBuilder crea el archivo, el nombre aparece en el cuadro de diálogo Vistas personalizadas o en la vista, tal y como se muestra en la siguiente imagen:



- 3 Escoja el nombre de archivo, haga clic con el botón derecho y seleccione Modificar para añadir código. Aparece el editor emergente Propiedades de vista personalizada, con una declaración de clase.
- 4 Para utilizar la plantilla de vista personalizada, elimine el código existente y escriba `customv` en el editor y, a continuación, `Ctrl+J`. Se presenta la siguiente plantilla:



Se le solicita que sustituya el tipo devuelto, el tipo de objeto, el nombre del parámetro y el valor devuelto.

Importante

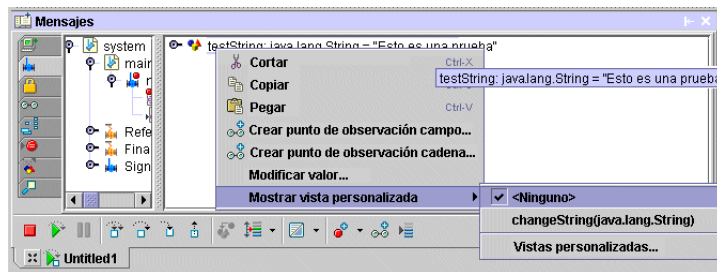
El tipo de parámetro del visualizador personalizado debe coincidir con el tipo de objeto para el que se está creando, o debe ser la superclase o superinterfaz de ese tipo de objeto.

- 5 Haga clic en Aceptar para cerrar el editor y guardar el visualizador personalizado.

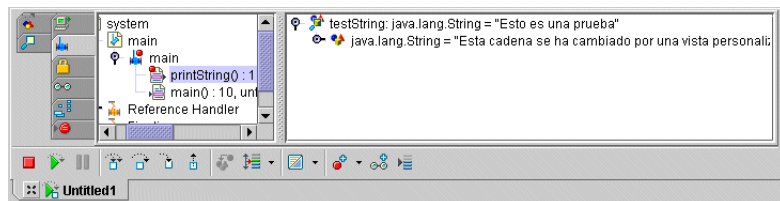
El visualizador personalizado se puede utilizar en la vista del depurador Hilos, pilas de llamada y datos o en Puntos de observación de datos. Si hay un visualizador personalizado, el icono para ese objeto se cambia en la vista del depurador:

Para ver los datos personalizados:

- 1 Haga clic con el botón derecho en el objeto original en la vista Hilos, pilas de llamada y datos o en la vista Puntos de observación de datos. Seleccione Mostrar vista personalizada, tal y como se muestra en la siguiente imagen:



- 2 Seleccione el visualizador para aplicar al objeto. Se cambia la presentación de los nodos dependientes del objeto; sin embargo no se cambia el nivel superior, tal y como se muestra en la siguiente imagen:



Nota

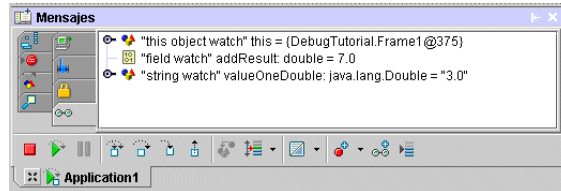
El comando Mostrar vista personalizada sólo está disponible para un objeto si el tipo de parámetro de la vista personalizada coincide con el tipo exacto del objeto original, o se puede derivar de ese tipo.

Observación de expresiones

Los puntos de observación permiten hacer un seguimiento de los cambios de valor de las variables o las expresiones, durante la ejecución del programa. Al introducir una expresión de punto de observación, la vista Puntos de observación de datos muestra el valor actual de la expresión. Los puntos de observación se evaluarán cuando estén dentro del ámbito.

La expresión de observación que devuelva un objeto o valor de matriz puede ampliarse para mostrar los elementos de datos que estén dentro del ámbito. Por ejemplo, si se define un punto de observación en un objeto `this` o en un único objeto, el punto de observación puede ampliarse. Sin embargo, si se asigna un punto de observación a un valor primitivo, el punto de observación no se puede ampliar debido a que es un elemento único. Los elementos que aparecen en gris en la vista ampliada son heredados.

Figura 8.16 Vista Puntos de observación de datos



Es posible definir dos tipos de puntos de observación:

- Puntos de observación de variables.
- Puntos de observación de objetos.

Puntos de observación de variables

Existen dos tipos de puntos de observación de variables:

- Puntos de observación de variables con nombre.
- Puntos de observación de variables de ámbito.

Puntos de observación de variables con nombre

Los puntos de observación de variables con nombre se añaden a un nombre; cuando se recorre el código, la variable que tenga el nombre seleccionado en el contexto actual será la evaluada para el punto de observación. Si no existe una variable con el nombre que ha seleccionado, el depurador mostrará el siguiente mensaje en la vista Puntos de observación de datos:

```
nombre de la variable = <está fuera de ámbito>
```

Para añadir puntos de observación con nombre, utilice el cuadro de diálogo Añadir punto de observación. Para presentar este cuadro de diálogo:

- Seleccione Ejecutar|Añadir punto de observación. Introduzca la expresión que desea observar en el campo Expresión. Si lo desea, puede introducir una descripción en el campo Descripción.
- En el editor, seleccione la expresión que quiere supervisar. Haga clic con el botón derecho y seleccione Añadir punto de observación. La expresión se introduce automáticamente en el cuadro de diálogo Añadir punto de observación. Si lo desea, puede introducir una descripción en el campo Descripción.

**Es una función de
JBuilder Developer y
Enterprise.**

Puntos de observación de variables de ámbito.

Los puntos de observación de variables de ámbito efectúan un seguimiento de la variable en el contexto en que se han creado. Cuando se recorre el código, sólo se muestra el valor de la variable especificada.

Si la expresión de la variable de ámbito no está en ámbito, el depurador mostrará el siguiente mensaje:

```
nombre de la variable = <está fuera de ámbito>
```

Si la expresión está en ámbito, el depurador mostrará su valor.

Para añadir un punto de observación de variables de ámbito, seleccione la variable deseada en la vista Hilos, pilas de llamada y datos. A continuación, haga clic con el botón derecho del ratón. El menú permite añadir un punto de observación de ámbito para el tipo de variable seleccionado. Estos tipos son:

- **Campo:** variable de Java definida en un objeto Java.
- **Campo estático:** variable de Java definida como estática (variable de clase).
- **Variable local:** variable que es local a un método o constructor.
- **Objeto `this`:** la creación de la instancia de la clase con la que se está trabajando.
- **Matriz:** conjunto de objetos idénticos.
- **Componente de matriz:** elemento individual de matriz.
- **Cadena:** tipo `String` de Java.

La siguiente tabla muestra la forma en que se ven algunos de estos tipos de puntos de observación en la vista Puntos de observación de datos:

Tabla 8.28 Tipos de variables de puntos de observación en ámbito







Tipos de puntos de observación	Presentación	Descripción
Punto de observación de campo	 "addResult"DebugTutorial.Frame1.this.addResult: double=68.0	El campo en punto de observación, <code>addResult</code> , es de tipo primitivo. Se encuentra en <code>DebugTutorial.Frame1</code> . Su valor es 68.0.
Punto de observación de variables locales	 "valueOneDouble"valueOneDouble: java.lang.double={java.lang.Double@354}	La variable local en punto de observación es <code>valueOneDouble</code> . Se define como un objeto <code>Double</code> .

Tabla 8.28 Tipos de variables de puntos de observación en ámbito (continuación)

Tipos de puntos de observación	Presentación	Descripción
Punto de observación de objetos	 <code>"DebugTutorial.Frame1"<reference>DebugTutorial.Frame1={DebugTutorial.Frame1@353}</code>	El objeto en punto de observación es <code>DebugTutorial.Frame1</code> . El objeto se amplía para mostrar los miembros de datos.
Punto de observación <code>this</code>	 <code>"this" this:{DebugTutorial.Frame1@353}</code>	La instanciación actual de <code>DebugTutorial.Frame1</code> . El objeto se amplía para mostrar los miembros de datos de la instanciación actual.
Punto de observación de matriz	 <code>"valueOneText"<reference>char[]=char[2]</code>	La matriz en punto de observación recibe el nombre <code>valueOneText</code> . Contiene dos elementos de matriz.
Punto de observación de componentes de matriz	 <code>"[0] = '3'"</code>	El primer elemento de la matriz <code>valueOneText</code> . Contiene el valor <code>'3'</code> .

Desplazamiento a un punto de observación de variable con ámbito

Puede dirigirse a un método en el editor en el que se defina la variable para el punto de observación de variable con ámbito seleccionado. Los puntos de observación de variable con ámbito efectúan un seguimiento de la variable en el contexto en que se han creado. Los puntos de observación de variables con ámbito se crean en la vista Hilos, pilas de llamada y datos. Diríjase al método, pulse un punto de observación de variable con ámbito de la vista Puntos de observación de datos y seleccione Ir a punto de observación.

Puntos de observación de objetos

Es una función de
JBuilder Developer y
Enterprise.

Los puntos de observación de objetos efectúan un seguimiento de un objeto Java determinado.

Para añadir un punto de observación de objetos:

- 1 Seleccione el objeto que desea observar. Puede hacerlo en la vista Puntos de observación de datos, en la vista Hilos, pilas de llamada y datos y en la vista Clases cargadas y datos estáticos.
- 2 Haga clic con el botón derecho del ratón y seleccione Crear punto de observación de objetos.

Un punto de observación del objeto `this` observa la instanciación actual del objeto seleccionado.

Edición de un punto de observación

Para modificar una expresión de observación, selecciónela en la vista Puntos de observación de datos y haga clic con el botón derecho del ratón. Elija Cambiar punto de observación.

- Para cambiar el nombre del punto de observación, escriba el nuevo nombre en el campo Expresión.
- Para cambiar la descripción, escriba el nuevo nombre en el campo Descripción.

Eliminación de puntos de observación

Para cambiar una expresión de observación, selecciónela en la vista Puntos de observación de datos y seleccione Eliminar punto de observación o pulse *Eliminar*. Si desea borrar todos los puntos de observación, puede hacer clic con el botón derecho del ratón en una zona vacía de la vista Puntos de observación de datos y elegir Eliminar todos.

Precaución El comando Eliminar todo no es reversible.

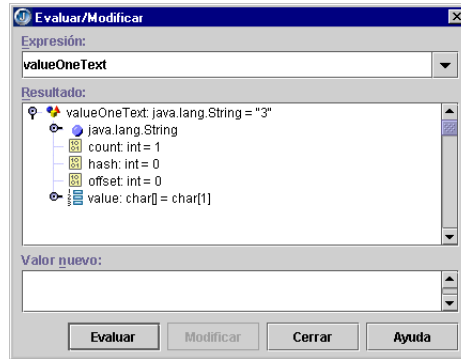
Evaluación y modificación de expresiones

Mediante el cuadro de diálogo Evaluar/Modificar (Ejecutar|Evaluar/Modificar o *F6*) es posible evaluar expresiones, cambiar los valores de los elementos de datos y evaluar llamadas a métodos. Esto puede resultar útil si cree que ha encontrado la solución a un error y quiere probar la corrección antes de salir del depurador, modificar el código fuente y volver a compilar el programa. CodeInsight y la sintaxis resaltada se muestran cuando se introduce una expresión en el campo correspondiente.

Para abrir el cuadro de diálogo Evaluar/Modificar, seleccione Ejecutar|Evaluar/Modificar o, bien, seleccione la expresión, haga clic con el botón derecho y escoja Evaluatr/Modificar.

Evaluación de expresiones

Para evaluar una expresión, introdúzcala en el campo Expresión. Si ya está seleccionada en el editor, se introduce automáticamente en este campo. Haga clic en el botón Evaluar. Este cuadro de diálogo permite evaluar cualquier expresión válida del lenguaje, excepto las que están fuera del ámbito actual. Si el resultado es un objeto, se muestra el contenido del objeto.

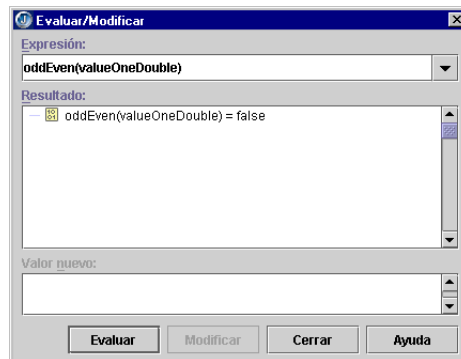
Figura 8.17 Evaluación de expresiones en el cuadro de diálogo Evaluar/Modificar

Evaluación de las llamadas a métodos

Es una función de
JBuilder Developer y
Enterprise.

Los resultados que se obtienen de una llamada a métodos también pueden evaluarse. Para evaluar una llamada a un método, escriba el nombre del método y los parámetros en el campo Expresión del cuadro de diálogo Evaluar/Modificar. Pulse Evaluar.

En este caso, el valor de retorno del método es `true`.

Figura 8.18 Evaluación de métodos en el cuadro de diálogo Evaluar/Modificar

Modificación de los valores de las variables

Es una función de
JBuilder Developer y
Enterprise.

Es posible cambiar los valores de las variables durante la sesión de depuración, con el fin de probar diferentes hipótesis de error y comprobar cómo se comporta una sección de código bajo diferentes circunstancias.

Si cambia el valor de una variable en el depurador, la modificación sólo se aplica en esa ejecución específica del programa; los cambios realizados en el cuadro de diálogo Evaluar/Modificar no afectan al código fuente del programa ni al programa compilado. Si desea que el cambio sea permanente, debe modificar el código fuente del programa en el editor y volver a compilarlo.

Para modificar el valor de una variable, introduzca:

```
variable = <valor nuevo>
```

en el cuadro de edición Expresión. El depurador mostrará los resultados en el cuadro de visualización de resultados Resultado. El resultado debe ser un tipo compatible con la variable.

Nota Tanto el campo Expresión como el campo Valor nuevo son compatibles con CodeInsight.

También es posible modificar el valor de una variable siguiendo estos pasos:

- 1 Abra el cuadro de diálogo Evaluar/Modificar e introduzca en el cuadro de edición Expresión el nombre de la variable que desea modificar.
- 2 Haga clic en Evaluar para evaluar la variable.
- 3 Introduzca un valor en el cuadro de edición Valor nuevo (o seleccione un valor en la lista desplegable), y haga clic en Modificar para actualizar la variable.

La expresión del cuadro de entrada Expresión o el valor del cuadro Valor nuevo deben tener un tipo compatible con la variable a la que desea asignarlos. En general, si la asignación diera lugar a un error durante la compilación o la ejecución, se trata de un valor de modificación no válido.

Por ejemplo, suponga que `valueOneText` es un objeto `String`. Si escribe:

```
valueOneText=34
```

en el campo de entrada Expresión, aparecerá el siguiente mensaje indicando un conflicto de tipos en el campo Resultado:

```
incompatible types; found int; required java.lang.String
```

Y deberá escribir:

```
valueOneDouble="34"
```

en el campo de entrada Expresión para que la expresión quede configurada con el nuevo valor.

Modificación del código durante la depuración

**Es una función de
JBuilder Developer y
Enterprise.**

El depurador permite realizar cambios en el código fuente mientras se depura. También puede actualizar todos los archivos de clase del proyecto o actualizarlos de manera individual.

Actualización de todos los archivos de clase



Cuando los archivos se modifican durante la depuración, el botón Intercambio inteligente se encuentra disponible en la barra de herramientas del depurador. Cuando se pulsa este botón, todos los archivos modificados en el proyecto se compilan y actualizan. Con Intercambio inteligente puede comprobar el código, realizar cambios y seguir depurando en la misma sesión de depuración, desde el punto de ejecución actual.

Para utilizar Intercambio inteligente:

- 1 Abra el código fuente de los archivos que desea modificar.
- 2 Modifique el código fuente.
- 3 Pulse el botón Intercambio inteligente o, bien, seleccione Ejecutar|Intercambio inteligente.

Intercambio inteligente compila todos los archivos modificados del proyecto. Puede continuar la depuración en la misma sesión.

Importante

Si la MV de destino del proyecto (Propiedades de proyecto|Generar|Java) está configurada en Todos los SDK de Java, el intercambio inteligente no funciona correctamente. La opción MV de destino de Todos los SDK de Java genera archivos de clase que pueden cargar todas las MV. Sin embargo, este archivo de clase incluirá el código que da instrucciones al cargador de clases para verificar todas las clases que se encuentran referenciadas en esta clase. Si la clase referenciada no se ha cargado todavía, la clase cargada cargará el archivo de clase en su memoria caché. Intercambio inteligente no puede acceder a este caché para actualizarlo. Por consiguiente, si está depurando y realiza un cambio en un archivo de clase que todavía no se ha cargado, el depurador no sabe que se ha llevado a cabo un cambio. Para solucionarlo, añada el siguiente parámetro de la MV al campo Parámetros de la MV del nodo Ejecutar del cuadro de diálogo Nueva o Modificar configuraciones de ejecución (Sólo es necesario que añada este parámetro si selecciona Todos los SDK de Java para la MV de destino).

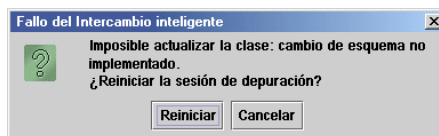
```
-Xverify:none
```

Este parámetro de la MV indica al cargador de clases que verifique sólo la clase actual.

Nota

Smart Swap no funciona con archivos JSP.

Si se añaden o eliminan campos o métodos en una clase, JBuilder abre el cuadro de diálogo Fallo del intercambio inteligente después de pulsar el botón Intercambio inteligente. El Intercambio inteligente falla debido a que JDK 1.4.x no admite esta característica. Si desea reiniciar la sesión de depuración, pulse el botón Reiniciar; si no es así, haga clic en Cancelar para seguir depurando. (En este caso, de todas formas, los archivos están desincronizados).



Actualización de archivos de clase individuales

Mientras depura, puede actualizar sólo clases individuales del proyecto. Para actualizar un solo archivo de clase durante la depuración:

- 1 Antes de depurar, asegúrese de que la opción Redefinir clases después de compilar del nodo Depurar del cuadro de diálogo Nueva o Modificar configuraciones de ejecución se encuentra activada.
- 2 Mientras depura, abra el código fuente del archivo que desea modificar.
- 3 Haga clic con el botón derecho del ratón en el archivo, en el panel de proyecto, y seleccione Ejecutar Make. JBuilder compilará automáticamente el archivo y actualizará las clases. Continuará con la misma sesión de depuración.

Nota Si la opción Redefinir clases después de compilar se encuentra desactivada (nodo Depurar/cuadro de diálogo Nueva o Modificar configuración de ejecución) y la opción Advertir de la modificación de archivos está activada, el cuadro de diálogo Archivos modificados se abrirá y podrá elegir cómo proceder. Se puede:

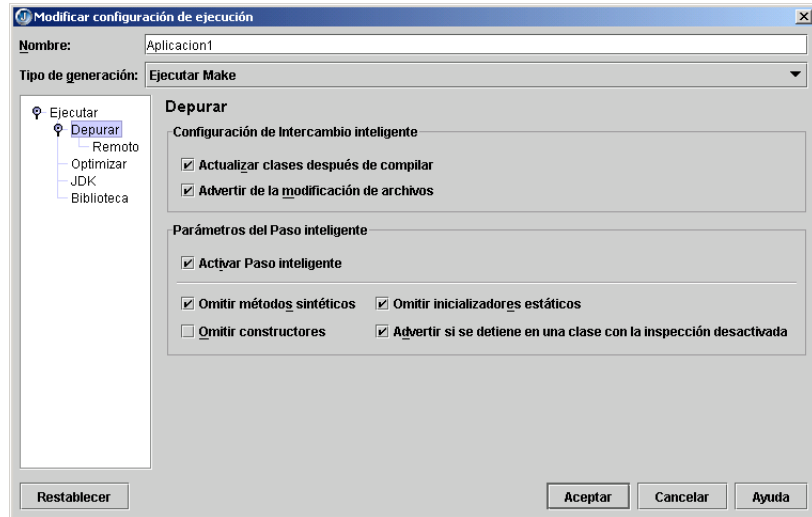
- Compilar, actualizar las clases compiladas y continuar con la sesión de depuración.
- Continuar con la sesión de depuración sin compilar ni actualizar.
- Reiniciar la sesión de depuración.

Restablecimiento del punto de ejecución

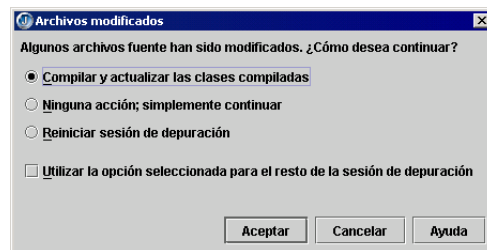
Una vez que ha modificado el código, puede restablecer el punto de ejecución (todavía seguirá en la misma sesión de depuración). El restablecimiento del punto de ejecución permite volver al punto anterior, al valor cambiado, para que pueda volver a comprobarlo y ver si los ajustes funcionan. Si desea más información, consulte [“Definición del punto de ejecución” en la página 8-35](#).

Opciones para modificar el código

Las opciones de intercambio inteligente de la parte superior del nodo Depurar del cuadro de diálogo Nueva o Modificar configuración de ejecución controlan el modo de gestión de los archivos modificados durante una sesión de depuración.

Figura 8.19 Nodo Depuración del cuadro de diálogo Modificar configuración de ejecución

- La opción Redefinir clases después de compilar actualiza automáticamente todos los archivos modificados que se han compilado. Cuando esta opción está activa, no se le advertirá de que ha modificado el código fuente si se compila los archivos modificados. Si esta opción no está activada, y si se ha activado la opción Advertir de la modificación de archivos, aparece el cuadro de diálogo Archivos modificados, donde puede decidir cómo continuar. (Esta función pertenece a las ediciones Developer y Enterprise de JBuilder).
- La opción Advertir de la modificación de archivos abre el cuadro de diálogo Archivos modificados en el que selecciona cómo continuar. El cuadro de diálogo Archivos modificados tendrá un aspecto parecido a éste:



Puede elegir:

- Actualizar los archivos y continuar en la misma sesión de depuración,
- continuar la depuración sin actualizar los archivos o, bien,
- iniciar una nueva sesión de depuración.

Si inicia una nueva sesión, se detiene la sesión actual y se restablece el punto de ejecución al principio del programa. Si continúa con la sesión actual, se comienza la ejecución en el punto de ejecución actual.

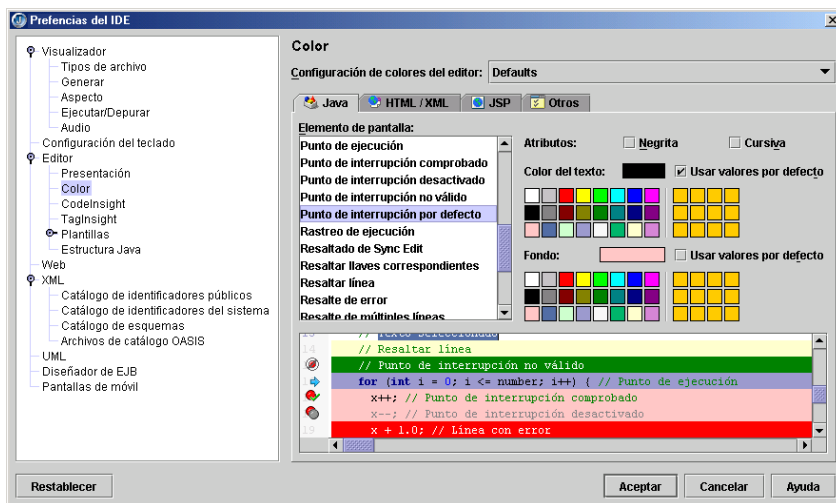
Personalización del depurador

Puede personalizar los colores utilizados para indicar el punto de ejecución y activar, desactivar y convertir en no válidas las líneas de los puntos de interrupción.

Personalización de la presentación del depurador

Para definir los colores de los puntos de interrupción y los puntos de ejecución:

- 1 Seleccione Herramientas|Preferencias|Editor|Color|Java, donde se configuran los colores de los elementos de la pantalla del depurador.



- 2 En la lista Elemento de pantalla, seleccione un elemento relacionado con la depuración y, a continuación, los colores de fondo y de texto del elemento.

Los elementos de pantalla que participan en la depuración son:

- Punto de interrupción por defecto.
- Punto de interrupción desactivado.
- Rastreo de ejecución.
- Punto de ejecución.
- Punto de interrupción no válido.
- Punto de interrupción comprobado.

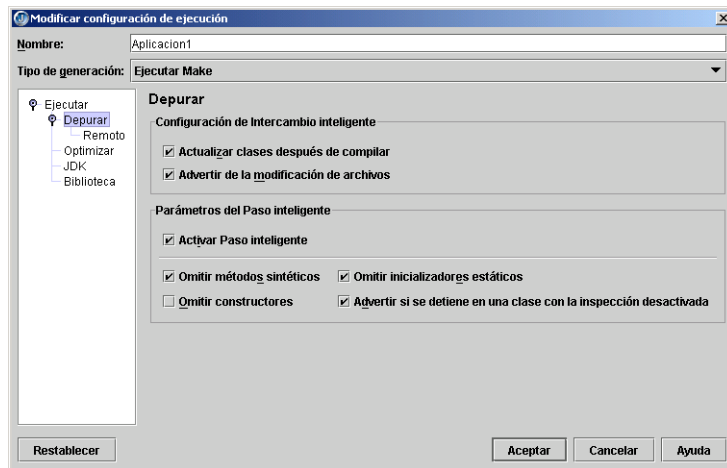
Configuración de las opciones de depuración

(Las distintas configuraciones de ejecución son una característica de las ediciones Developer y Enterprise de JBuilder).

Las configuraciones de depuración se pueden crear de forma autónoma o como parte de la configuración de ejecución. Si desea más información sobre las configuraciones de ejecución, consulte [“Definición de las configuraciones de ejecución” en la página 7-8](#).

Para establecer las opciones de una configuración de depuración:

- 1 Seleccione Ejecutar!Configuraciones. Se muestra el cuadro de diálogo Propiedades de configuración de ejecución.
- 2 Elija la configuración deseada y pulse Modificar.
- 3 En el cuadro de diálogo Modificar configuración de ejecución, abra el nodo Depurar.



- 4 Para configurar cómo el depurador gestiona los archivos modificados, configure las siguientes opciones.
 - Redefinir clases después de compilar
Cuando se realiza la compilación se actualizan automáticamente todos los archivos modificados. No se recibe ningún aviso de que se ha modificado el código fuente si éste se ha compilado. Si esta opción no está activada, y si se ha activado la opción Advertir de la modificación de archivos, aparece el cuadro de diálogo Archivos modificados, donde puede decidir cómo continuar. (Esta función pertenece a las ediciones Developer y Enterprise de JBuilder).
 - Advertir de la modificación de archivos
Muestra el cuadro de diálogo Archivos modificados donde elige cómo continuar. Puede actualizar los archivos y continuar con la sesión de depuración actual reanudar la sesión de depuración si actualizar los archivos o iniciar una nueva sesión de depuración.



- 5 Para activar Paso inteligente, elija la opción Activar paso inteligente o pulse el botón Paso inteligente de la barra de herramientas del depurador. Paso inteligente se encuentra activado por defecto.
- 6 Para configurar la activación y desactivación de Paso inteligente, defina las siguientes opciones: (La configuración de Paso inteligente es una característica de las ediciones Developer y Enterprise de JBuilder).
 - Omitir métodos sintéticos
Omite los métodos sintéticos cuando se inspeccionan clases.
 - Omitir constructores
Omite los constructores cuando se inspeccionan clases.
 - Omitir inicializadores estáticos
Omite inicializadores estáticos (clase) al inspeccionar las clases.
 - Advertir si se detiene en una clase con la inspección desactivada
Muestra un mensaje de advertencia si hay un punto de interrupción en una clase que tenga la inspección desactivada. Para obtener más información, consulte la sección [“Puntos de interrupción y configuración de Inspección desactivada” en la página 8-46](#).

Si desea información sobre las opciones de depuración remota, consulte el [Capítulo 9, “Depuración remota”](#). (Depuración remota es una función de las ediciones Developer y Enterprise de JBuilder).

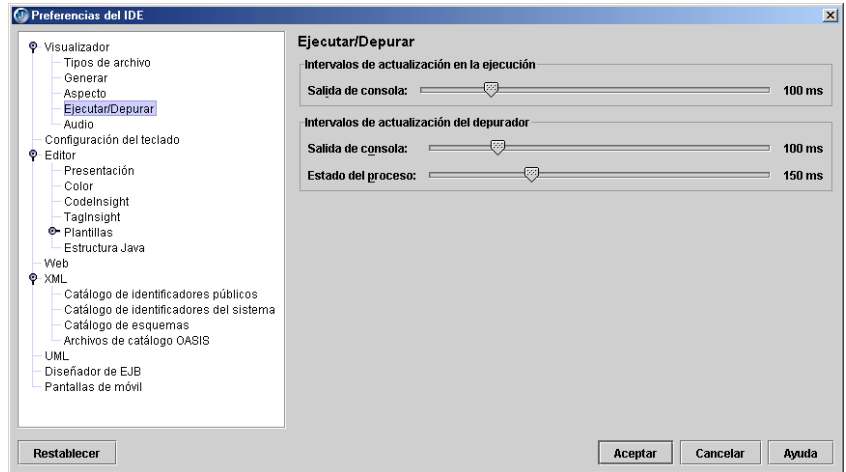
Definición de los intervalos de actualización

También es posible especificar la frecuencia de los intervalos que controlan cuándo se registran los cambios de estado de la consola/proceso. Si los intervalos son pequeños, las respuestas del depurador/de ejecución de la salida y otros sucesos, como la ejecución paso a paso, serán más rápidos, pero JBuilder utilizará la mayor parte del tiempo del procesador.

Por lo general, se pueden asignar valores muy reducidos a estas propiedades, a menos que se estén ejecutando otras aplicaciones además de JBuilder, o que el programa que se está depurando requiera muchos recursos. Si es este el caso, debe alargar los intervalos.

Para cambiar los intervalos de actualización:

1 Seleccione Herramientas|Preferencias|Visualizador|Ejecutar/Depurar.



- 2** Para definir el intervalo de salida de consola de los procesos de ejecución, mueva la barra de desplazamiento de la Salida de consola en la parte superior del cuadro de diálogo.
- 3** Para definir el intervalo de salida de consola de los procesos de depuración, mueva la barra de desplazamiento de la Salida de consola en el centro del cuadro de diálogo.
- 4** Para definir el intervalo de las actualizaciones de estado del proceso de depuración, mueva la barra de desplazamiento de Estado del proceso.



Depuración remota

Es una función de JBuilder Developer y Enterprise.

JBuilder incluye varias características que facilitan la depuración de aplicaciones distribuidas. En concreto, incluye apoyo para la depuración interprocesal y la depuración remota.

Estas características son complementarias a las funciones básicas de depuración de JBuilder. Si es la primera vez que utiliza JBuilder, consulte el [Capítulo 8, “Depuración de programas en Java”](#) para encontrar información sobre el entorno de depuración de JBuilder.

La depuración remota consiste en depurar desde un ordenador el código que se ejecuta en otro. Esta posibilidad resulta ideal, por ejemplo, en los casos en que una aplicación experimenta un problema que sólo se produce en uno de los ordenadores de la red, y no en los demás. El depurador remoto de JBuilder también permite depurar entre distintos sistemas operativos.

En este capítulo, el "ordenador cliente" es el ordenador que está ejecutando JBuilder. Éste es el ordenador desde el que se efectúa la depuración. El "ordenador remoto" ejecuta la aplicación que se desea depurar.

Existen dos formas de efectuar la depuración remota. Se puede:

- Abrir un programa de un ordenador remoto desde un ordenador cliente y depurarlo con JBuilder. Si desea más información, consulte [“Apertura y depuración de programas en un equipo remoto” en la página 9-2](#). En este caso se ejecuta el servidor de depuración de JBuilder en el ordenador remoto.
- Depurar por medio de JBuilder un programa que ya se está ejecutando en el ordenador remoto. Si desea más información, consulte [“Depuración de un programa que se ejecuta en un ordenador remoto” en la página 9-6](#). En este caso no es necesario ejecutar el servidor de depuración.

Nota Los ordenadores cliente y remoto deben tener instalado JDK 1.2 o superior (una versión de JDK que acepte el API de depuración JPDA). No es necesario que coincidan las versiones de JDK de los dos ordenadores. Durante la instalación de JBuilder, JDK 1.4 se instala automáticamente en el directorio <jbuilder>/jdk1.4.

También se puede depurar el código que se ejecuta en un proceso independiente, en el mismo ordenador en el que está instalado JBuilder. Para ello, inicie el proceso en el modo de depuración y cree un enlace con JBuilder. Si desea más información, consulte [“Depuración del código local que se ejecuta en un proceso independiente” en la página 9-9](#).

Además, es posible establecer puntos de interrupción interprocesales, ideales para la depuración de aplicaciones cliente/servidor. Si desea más información, consulte [“Depuración con puntos de interrupción interprocesales” en la página 9-10](#).

Para realizar depuraciones remotas de cualquier tipo es necesario utilizar configuraciones de depuración. Si desea más información sobre configuración de ejecución y depuración, consulte [“Definición de las configuraciones de ejecución” en la página 7-8](#) y [“Definición de las configuraciones de ejecución” en la página 7-8](#).

Apertura y depuración de programas en un equipo remoto

En este apartado se explica la forma de abrir un programa de un ordenador remoto desde un ordenador cliente, depurándolo desde aquí con JBuilder. Los pasos son los siguientes:

- 1 Instalar el servidor de depuración en el equipo remoto y ejecutarlo.
- 2 Compilar la aplicación en el ordenador remoto o copiar en él los archivos `.class` de la aplicación.
- 3 Utilizar JBuilder en el ordenador cliente para abrir y depurar la aplicación en el ordenador remoto.

Importante Los archivos de código fuente de la aplicación que se está depurando deben estar disponibles en el ordenador cliente. Los archivos `.class` compilados deben estar a disposición del ordenador remoto. Estos archivos deben coincidir. De lo contrario, podrían producirse resultados imprevisibles, por ejemplo errores, o que el depurador se detenga en una línea de código fuente incorrecta. Siempre que se modifica el código fuente se deben actualizar los archivos `.class` del ordenador remoto.

En primer lugar, instale el servidor de depuración en el equipo remoto y ejecútelo. Si JBuilder ya se encuentra instalado en el ordenador remoto, puede empezar por el cuarto paso.

- 1 Copie el archivo `debugserver.jar` (situado en el directorio <jbuilder>/remote) en el ordenador remoto. Anote el directorio donde lo ha copiado, pues lo necesitará en etapas posteriores.

- 2 Copie el script de shell del servidor de depuración, `DebugServer` (Unix) o el archivo por lotes, `DebugServer.bat` (Windows), en el mismo directorio del equipo remoto.
- 3 Compruebe que JDK 1.2.2 o superior se encuentra instalado en el ordenador remoto.
- 4 Vaya al directorio del ordenador remoto donde se han instalado los archivos del servidor de depuración. Ejecute `DebugServer` con el fin de personalizar las variables de entorno del servidor de depuración remoto.

En los sistemas Unix, utilice el siguiente comando:

```
./DebugServer <debugserver.jar_dir> <jdk_home_dir> [-port portnumber]
[-timeout milliseconds]
```

En el sistema Windows, utilice:

```
DebugServer <debugserver.jar_dir> <jdk_home_dir> [-port portnumber]
[-timeout milliseconds]
```

Donde:

- *debugserver.jar_dir*: El directorio del equipo remoto donde se encuentra el archivo JAR del servidor de depuración. En los sistemas que utilizan Windows es necesario introducir la letra de la unidad.
- *jdk_home_dir*: Directorio inicial del ordenador remoto para la instalación de JDK. En los sistemas que utilizan Windows es necesario introducir la letra de la unidad.
- *-port*: parámetro opcional que ejecuta el servidor de depuración en un puerto distinto del predeterminado, 18699. Este valor sólo se debe modificar si se está utilizando el número de puerto por defecto. Los números de puerto válidos están comprendidos entre el 1025 y el 65535. Este valor debe coincidir con el introducido en el campo Número de puerto en el nodo DepurarRemota del cuadro de diálogo Modificar configuración de ejecución (en el ordenador cliente). Consulte el paso 6, más abajo.
- *-timeout*: parámetro opcional que establece el número de milisegundos para intentar conectar el ordenador remoto al cliente. El proceso se detiene cuando se alcanza este número. El valor por defecto es de 60.000 milésimas de segundo.

Ejemplo de este comando en un entorno Windows:

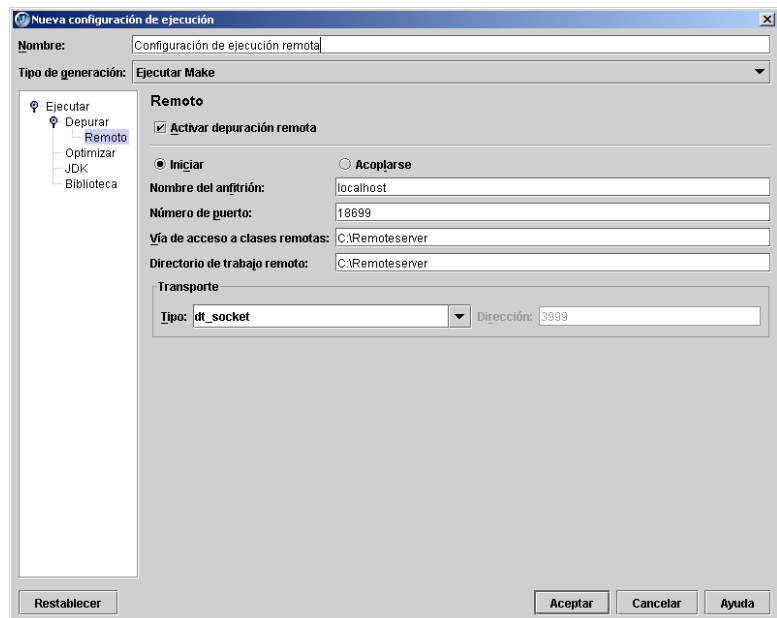
```
DebugServer d:\remote d:\jdk1.3 -port 1234 -timeout 20000
```

- 5 Pulse *Intro* para iniciar el servidor de depuración.

Las funciones de depuración remota de JBuilder en modo de lanzamiento estarán activas si el servidor de depuración se está ejecutando. Cuando se está ejecutando el servidor de depuración en el ordenador remoto es necesario compilar la aplicación y copiar en este ordenador los archivos `.class`. (También se puede compilar la aplicación de forma remota). Después

se utiliza JBuilder, que se ejecuta en el ordenador cliente, para abrir y depurar el programa del ordenador remoto.

- 1 Compile la aplicación. La aplicación se puede compilar utilizando JBuilder en el ordenador cliente, para después copiar los archivos `.class` en el ordenador remoto o transferirlos por FTP. También se puede compilar la aplicación directamente en el ordenador remoto llamando al compilador `javac` con la opción `-g`. (Esto le dice al compilador que tiene que añadir información de depuración al archivo compilado).
- 2 Abra JBuilder en el ordenador cliente.
- 3 Abra el proyecto que va a depurar la aplicación.
- 4 Cree una configuración de depuración (puede formar parte de la configuración de ejecución o ser independiente):
 - a Seleccione Ejecutar|Configuraciones. Se muestra el cuadro de diálogo Propiedades de configuración de ejecución. Para crear una configuración, pulse el botón Nueva. Para modificar una configuración, selecciónela y pulse Modificar.
 - b Si la configuración es nueva, escriba su nombre en el campo Nombre.
 - c Asigne el valor Ninguna a Tipo de generación.
 - d Seleccione Depurar|Remoto.



- 5 Active la casilla de selección Activar depuración remota. Elija la opción Iniciar.

6 Rellene los siguientes campos:

- Nombre del anfitrión: nombre del ordenador remoto. localhost es el valor por defecto. Para averiguar el nombre del anfitrión compruebe la configuración de red en el ordenador remoto.
- Número de puerto: el número de puerto del ordenador remoto con el que el usuario se está comunicando. Utilice el número de puerto por defecto, 18699. Cámbielo únicamente si ya está en uso. Los valores válidos van del 1024 al 65535. Este valor debe coincidir con el parámetro `-port` utilizado para iniciar el servidor de depuración en el ordenador remoto. Consulte el cuarto paso del apartado anterior.
- Vía de acceso a clases remotas: la vía de acceso a clases donde se encuentran los archivos `.class` compilados de la aplicación que se está depurando en el ordenador remoto. Este campo presenta el mismo funcionamiento que otros campos similares: si las clases se encuentran en un paquete, se debe especificar el directorio de raíz y no el directorio que contiene las clases. En los sistemas Windows, especifique la unidad si ésta no es `C:`. Esta vía de acceso a clases remota sólo es aplicable a esta sesión de depuración concreta.
- Directorio de trabajo remoto: el directorio de trabajo del ordenador remoto. En los sistemas Windows, especifique la unidad si ésta no es `C:`. Este directorio de trabajo remoto sólo es aplicable a esta sesión de depuración.

Advertencia

JDK 1.2.2 no acepta el directorio de trabajo. Si el ordenador remoto ejecuta JDK 1.2.2 y se introduce un directorio de trabajo remoto, en la vista Salida, entrada y errores de consola del depurador se mostrará una advertencia.

- Transporte: el tipo de transporte: `dt_shmem` (transporte de memoria compartida, no disponible en los sistemas Unix) o `dt_socket` (transporte de socket). Si desea obtener más información sobre los métodos de transporte, consulte "JPDA: Connection and Invocation Details - Transports" (en inglés) en la página <http://java.sun.com/products/jpda/doc/conninv.html#Transports>.

7 Pulse dos veces sobre Aceptar para cerrar los cuadros de diálogo.**8** Para dar comienzo a la sesión de depuración, elija una de las siguientes opciones:

Comando	Atajo	Descripción
Ejecutar Depurar proyecto	<i>Mayús + F9</i>	Inicia el programa en el depurador utilizando la configuración por defecto o la seleccionada. Ejecuta el programa hasta que se completa o suspende la ejecución en la primera línea de código en la que se requiere que el usuario introduzca un dato o en un punto de interrupción.

Comando	Atajo	Descripción
Ejecutar/Omitir inspección	F8	Interrumpe la ejecución en la primera línea de código ejecutable.
Ejecutar/Inspeccionar	F7	Interrumpe la ejecución en la primera línea de código ejecutable.

Cuando se inicia el depurador, la aplicación que se desea depurar (según lo elegido en Classpath remota) se abre en el ordenador remoto. En JBuilder se muestra el depurador que se ejecuta en el ordenador cliente; no obstante, se están depurando los archivos `.class` que se ejecutan en el ordenador remoto.

Nota Si esa aplicación ya se está ejecutando, el servidor de depuración lanzará una nueva instancia. (Si desea depurar una aplicación que ya se está ejecutando, consulte [“Depuración de un programa que se ejecuta en un ordenador remoto” en la página 9-6](#)).

- 9 Para cerrar la aplicación en el ordenador remoto, detenga el proceso en JBuilder. Para cerrar el servidor de depuración en el ordenador remoto, elija el comando Archivo|Salir del servidor de depuración.

Depuración de un programa que se ejecuta en un ordenador remoto

En este apartado se explica la forma de conectarse a un programa que ya se está ejecutando en un ordenador remoto desde un ordenador cliente y depurarlo con JBuilder. Para ello es necesario:

- 1 Ejecutar la aplicación en el ordenador remoto utilizando las opciones de depuración de la máquina virtual (MV).
- 2 Pasar a y depurar la aplicación que se está ejecutando en el ordenador cliente por medio de JBuilder.

Importante Los archivos de código fuente de la aplicación que se está depurando deben estar disponibles en el ordenador cliente. Los archivos `.class` compilados deben estar a disposición del ordenador remoto. Estos archivos deben coincidir. De lo contrario, podrían producirse resultados imprevisibles, por ejemplo errores, o que el depurador se detenga en una línea de código fuente incorrecta. Siempre que se modifica el código fuente se deben actualizar los archivos `.class` del ordenador remoto.

Si desea seguir un tutorial que explique cómo enlazar con un programa que se está ejecutando, consulte [Capítulo 21, “Tutorial: Depuración remota”](#).

Para iniciar un programa en el ordenador remoto y enlazarse a él:

- 1 Compile la aplicación en el ordenador remoto. La aplicación también se puede compilar con JBuilder en el ordenador cliente, para después copiar los archivos `.class` en el ordenador remoto o transferirlos por FTP.

- 2 Ejecute la aplicación en el ordenador remoto con las siguientes opciones de MV.
 - Si JBuilder está instalado en el ordenador remoto, el programa se puede ejecutar desde JBuilder. Abra el proyecto y modifique la configuración de ejecución (Ejecutar|Configuraciones|Edición). Introduzca los siguientes parámetros en el campo Parámetros de la MV:

```
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdpw:transport=dt_socket,server=y,address=3999,suspend=y
```

- Si JBuilder no está instalado en el ordenador remoto, el programa se debe ejecutar desde la línea de comandos. Añada las siguientes opciones de MV para la línea de comandos de Java:

```
-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdpw:transport=dt_socket,server=y,address=3999,suspend=y
```

Los parámetros *address* y *suspend* son optativos. Se encuentran después del parámetro *server*, separados por una coma. No se pueden colocar espacios entre los parámetros.

- El parámetro *address*, basado en el transporte seleccionado, contiene el número y la dirección del puerto a través del cual se comunica el depurador con el ordenador remoto. Este parámetro simplifica la configuración, ya que elimina la necesidad de modificar continuamente el campo Dirección de la ficha Depurar del cuadro de diálogo Propiedades de ejecución. Si al tipo de transporte se le asigna el valor *dt_socket*, el parámetro *address* contiene el número de puerto. Si se le asigna *dt_shmem*, este parámetro contiene el nombre de dirección unívoco. Si está utilizando XP, no utilice 5000 para el parámetro *address*. Esta dirección está reservada para Universal Plug & Play.
- El parámetro *suspend* indica si el programa se suspende inmediatamente cuando se inicia. Esta configuración se puede desactivar por medio de *suspend=n*. (Si se ha definido *suspend=n* y no hay puntos de interrupción, cuando se inicia el programa se ejecuta hasta el final sin detenerse).

Nota Para ejecutar la aplicación con JDK 1.2x o 1.3x utilice el ejecutable `java` del subdirectorio `bin` del directorio de instalación de JDK, no el archivo `java.exe` del directorio `jre/bin`. Esto permite a la MV de Java cargar el archivo de depurador (`libjdpw.so` en Unix; `jdpw.dll` en Windows), necesario para la depuración. (No es aplicable a JDK 1.4x).

- 3 Abra JBuilder en el ordenador cliente.
- 4 Abra el proyecto de la aplicación que se está ejecutando en el ordenador remoto.
- 5 Cree una configuración de depuración (puede formar parte de la configuración de ejecución o ser independiente):
 - a Seleccione Ejecutar|Configuraciones. Se muestra el cuadro de diálogo Propiedades de configuración de ejecución. Para crear una

configuración, pulse el botón Nueva. Para modificar una configuración, selecciónela y pulse Modificar.

- b** Si la configuración es nueva, escriba su nombre en el campo Nombre.
- c** Asigne el valor Ninguna a Tipo de generación.
- d** Seleccione Depurar|Remoto.

- 6** Active la casilla de selección Activar depuración remota. Seleccione la opción Acoplarse.
- 7** Rellene los siguientes campos:
 - Nombre del anfitrión: nombre del ordenador remoto. localhost es el valor por defecto. Para averiguar el nombre del anfitrión compruebe la configuración de red en el ordenador remoto.
 - Transporte: opciones del método de transporte:
 - Tipo: dt_socket (transporte de socket) o dt_shmem (transporte de memoria compartida, no disponible en los sistemas Unix). Si desea obtener más información sobre los métodos de transporte, consulte "JPDA: Connection and Invocation Details - Transports" (en inglés) en la página <http://java.sun.com/products/jpda/doc/conninv.html#Transports>.
 - Dirección:
 - Si el tipo de transporte se define como dt_socket, el parámetro contiene el número de puerto del ordenador remoto con el que se ha establecido la comunicación. Utilice el número por defecto,

Importante


3999. Cámbielo únicamente si ya está en uso. Este valor debe coincidir con el valor del parámetro *address* de la MV de Java que inicia el programa en el ordenador remoto. Consulte el segundo paso de este apartado.

Si está ejecutando Windows XP, no utilice 5000 como el número de puerto o dirección mediante el que el depurador se comunica con un ordenador remoto. XP reserva este número de puerto para el Universal Plug & Play.

- Si se selecciona *dt_shmem* como tipo de transporte, asigne al parámetro *address* el nombre unívoco del ordenador remoto con el que se ha establecido la comunicación. El valor por defecto es *javadebug*.

8 Pulse dos veces sobre Aceptar para cerrar los cuadros de diálogo.

9 Elija Ejecutar|Omitir inspección o Ejecutar|Inspeccionar para iniciar el depurador.

10 Si el parámetro *suspend* de la MV del ordenador remoto tiene asignado el valor *y* (consulte el segundo paso de este apartado), haga clic en el botón  Reanudar el programa de la barra de herramientas del depurador para continuar con la depuración.

11 Para poner fin a la ejecución de la aplicación, ciérrela en el ordenador remoto.

12 Para desconectarse del ordenador remoto, detenga el proceso en JBuilder.

Nota

Para iniciar y depurar una aplicación en el ordenador remoto, consulte el apartado [“Apertura y depuración de programas en un equipo remoto” en la página 9-2](#).

Depuración del código local que se ejecuta en un proceso independiente

Para depurar el código que se ejecuta en un proceso independiente, en el mismo ordenador en el que está instalado JBuilder, siga las instrucciones anteriores, desde el segundo paso. Utilice los siguientes valores para las opciones de Conectar del nodo Depurar|Remota del cuadro de diálogo Modificar configuración de ejecución.

- Nombre del anfitrión
Configure la opción por defecto, *localhost*.
- Tipo de transporte
Asigne el valor *dt_socket* (transporte de socket) o *dt_shmem* (transporte de memoria compartida, no disponible en los sistemas Unix).
- Dirección de transporte
Si el Tipo de transporte es *dt_socket*, asigne el valor 3999. Si es *dt_shmem*, cámbielo a *javadebug* o a cualquier otro nombre.

Depuración con puntos de interrupción interprocesales

Un punto de interrupción interprocesal detiene el depurador cuando se inspecciona, en un proceso aparte, un método o el método especificado en la clase especificada. Esto permite inspeccionar un proceso servidor desde un proceso cliente, en lugar de tener que configurar puntos de interrupción en el cliente y en el servidor. Por lo general, se configura un punto de interrupción de línea en el cliente y un punto de interrupción interprocesal en el servidor. Si desea seguir un tutorial que explique paso a paso la inspección interprocesal, consulte el [Capítulo 21, “Tutorial: Depuración remota”](#).

Para activar un punto de interrupción interprocesal configurado en un proceso de servidor:

- 1 Inicie el proceso de servidor en el ordenador remoto en modo depuración. Consulte el paso 2 en la sección [“Depuración de un programa que se ejecuta en un ordenador remoto”](#) en la página 9-6.
- 2 En el ordenador cliente y desde JBuilder, conéctese con el servidor que ya está en ejecución en el ordenador remoto. Consulte los pasos 4 a 10 en la sección [“Depuración de un programa que se ejecuta en un ordenador remoto”](#) en la página 9-6.
- 3 Configure un punto de interrupción en el código del cliente y comience a depurar el cliente. Cuando llegue al punto de interrupción, inspeccione el código del servidor. No utilice Omitir inspección. La omisión de inspección no parará en el punto de interrupción interprocesal.

Para configurar un punto de interrupción interprocesal, utilice el cuadro de diálogo Añadir punto de interrupción interprocesal. Para abrir el cuadro de diálogo Añadir punto de interrupción interprocesal, haga una de las siguientes cosas:

- Antes de una sesión de depuración o en su transcurso, seleccione Ejecutar/Añadir punto de interrupción y seleccione Añadir punto de interrupción interprocesal.
- Durante la sesión de depuración, pulse la flecha de lista desplegable situada a la derecha del botón Añadir punto de interrupción; de la barra de herramientas del depurador y seleccione Añadir punto de interrupción interprocesal.
- Durante la sesión de depuración, haga clic con el botón derecho del ratón en una zona vacía de la vista Puntos de interrupción de datos y de código y seleccione Añadir punto de interrupción interprocesal.



Aparece el cuadro de diálogo Añadir punto de interrupción interprocesal.

Para configurar un punto de interrupción interprocesal:

- 1 En el campo Nombre de la clase, escriba el nombre de la clase del lado del servidor que contiene el método en el que desea se detenga el depurador. Pulse el botón de puntos suspensivos para buscar la clase.
 - 2 En el campo Nombre del método, escriba el nombre del método en el que desea se detenga el depurador. Utilice el botón de puntos suspensivos para mostrar el cuadro de diálogo Seleccionar método, en el cual se puede acceder a la lista de métodos disponibles en la clase seleccionada. No es necesario escribir el nombre del método. Si no se especifica un nombre de método, el depurador se detiene en todas las llamadas a métodos en la clase especificada.
- Nota** No se puede seleccionar un método si la clase seleccionada contiene errores de sintaxis o de compilación.
- 3 En el campo Argumentos del método, introduzca una lista de argumentos del método separados por comas. El depurador se parará cuando el nombre del método y la lista de argumentos coincidan. Esto resulta útil con los métodos sobrecargados.
 - Si no se establece ningún argumento, el depurador se detiene en todos los métodos cuyo nombre coincida con el especificado.
 - Si se elige un nombre de método del cuadro de diálogo Seleccionar método, el campo Argumentos del método se rellena automáticamente.
 - 4 Elija qué acciones realizará el depurador. El depurador puede detener la ejecución en el punto de interrupción, mostrar un mensaje o evaluar una expresión. Si desea más información, consulte [“Definición de las acciones de puntos de interrupción” en la página 8-57](#).
 - 5 En el campo Condición, si existe alguna, asígnela al punto de interrupción. Si desea más información, consulte [“Creación de puntos de interrupción condicionales” en la página 8-59](#).

- 6** En el campo Número de pasadas, defina el número de veces que se debe pasar el punto de interrupción con el fin de que se active. Si desea más información, consulte [“Utilización de puntos de interrupción por número de pasadas” en la página 8-60](#).
- 7** Haga clic en Aceptar para cerrar el cuadro de diálogo.
- 8** Fije un punto de interrupción de línea en el cliente, en el método que llama al punto de interrupción interprocesal.
- 9** Cuando se detenga en el punto de interrupción de línea, pulse el botón Inspeccionar de la barra de herramientas del depurador para inspeccionar el método correspondiente del servidor. (Si utiliza Omitir inspección, el depurador no se detendrá).





Creación de JavaBeans con BeansExpress

**Es una función de
JBuilder Developer y
Enterprise.**

BeansExpress es la forma más rápida de crear JavaBeans. Se compone de un conjunto de asistentes, diseñadores visuales y ejemplos de código que permiten desarrollar JavaBeans rápida y fácilmente. BeansExpress también permite modificar JavaBeans ya existentes. Asimismo, es posible convertir clases de Java en JavaBeans.

Definición de JavaBean

Los JavaBeans son colecciones de una o varias clases de Java, por lo general agrupadas en un archivo JAR (recopilatorio Java), que actúan como componentes autónomos y reutilizables. Los JavaBeans pueden ser componentes de interfaz de usuario o componentes no visuales, como módulos de datos o motores de cálculo.

En su forma más simple, los JavaBeans son clases `public` de Java que contienen un constructor sin parámetros. Normalmente, los JavaBeans disponen de propiedades, métodos y sucesos que respetan determinadas convenciones de nomenclatura (también denominadas pautas de diseño).

¿Por qué desarrollar JavaBeans?

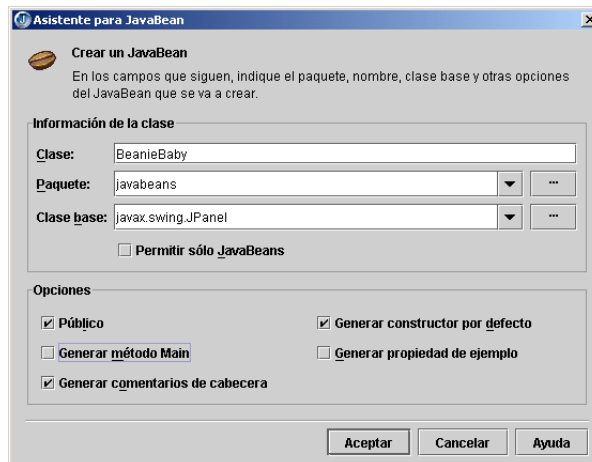
Al igual que otros tipos de componentes, los JavaBeans son fragmentos de código reutilizable que pueden actualizarse con una repercusión mínima en el proceso de prueba de los programas a los que se asocian. No obstante, los JavaBeans presentan ciertas ventajas sobre otros componentes:

- Se trata de componentes multiplataforma desarrollados íntegramente en Java.
- Es posible instalarlos en la paleta de componentes de JBuilder o en otras herramientas de desarrollo en Java y emplearlos en la construcción de programas.
- Pueden distribuirse en archivos .JAR.

Generación de clases bean

Para iniciar la creación de un JavaBean mediante BeansExpress,

- 1 Seleccione Archivo|Nuevo proyecto y cree un proyecto mediante el Asistente para proyectos.
- 2 Para abrir la galería de objetos, elija Archivo|Nuevo o pulse el botón Nuevo de la barra de herramientas principal.
- 3 Seleccione General para que se abra la ficha General y haga doble clic en el icono JavaBean para que se abra el Asistente para JavaBean.



Sugerencia

También se puede iniciar el Asistente para JavaBean haciendo clic en la flecha abajo que se encuentra junto al botón Nuevo de la barra de herramientas, que abre un menú desplegable, y seleccionando JavaBean.

- 4 Especifique el nombre de la nueva clase JavaBean en el campo Clase.

- 5 En el primer campo de texto, indique el paquete al que ha de pertenecer el bean. Por defecto, se toma el nombre del proyecto actual.
- 6 Seleccione la clase que desee ampliar, en el campo Clase base.
Puede optar por emplear la lista desplegable o hacer clic en el botón contiguo para acceder al visualizador de paquetes y especificar en él la clase de Java que desee.
- 7 Elija las restantes opciones según sus preferencias. Ninguna de ellas es obligatoria:
 - a Marque la opción Permitir sólo JavaBeans si desea que JBuilder le avise cuando trate de ampliar una clase de Java que no pueda ser un `JavaBean` válido.
 - b Seleccione Público si desea que la clase sea `public`.
 - c Marque Generar método Main si desea que JBuilder coloque el método `main` en el bean, lo que permite ejecutarlo.
 - d Compruebe Generar comentarios de cabecera si desea comentarios de cabecera Javadoc (Título, Descripción, Autor, etc.) añadido en la parte superior del archivo clase.
 - e Active la opción Generar constructor por defecto si desea que JBuilder cree un constructor sin parámetros.
 - f Marque la opción Generar propiedad de ejemplo si desea que JBuilder añada una propiedad `sample` al bean. Esta opción puede interesarle la primera vez que desarrolle un bean, con el fin de ver cómo JBuilder genera el código de una propiedad. Posteriormente, podrá eliminar esta propiedad o modificarla de modo que tenga alguna utilidad en el bean.
- 8 Pulse Aceptar para cerrar el Asistente para JavaBean.

JBuilder crea un `JavaBean` con el nombre especificado, lo añade al proyecto actual y muestra el código fuente que ha generado. Este es el código generado por JBuilder para los valores mostrados:

```
package myjavabean;

import java.awt.*;
import javax.swing.*;

public class BeanieBaby extends JPanel {
    BorderLayout borderLayout1 = new BorderLayout();

    public BeanieBaby() {
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

```
private void jbInit() throws Exception {  
    this.setLayout(borderLayout1);  
}  
}
```

Si examina el código generado por JBuilder, observará que:

- JBuilder ha asignado al bean el nombre especificado y ha ampliado la clase elegida, que está declarada como `public`.
- La clase cuenta con un constructor sin parámetros.

Aún en su estado más rudimentario, esta clase es un JavaBean válido.

Diseño de la interfaz de usuario de un bean

No todos los JavaBeans disponen de interfaz de usuario, pero si desea que la incluyan, se puede crear por medio del diseñador de interfaces de usuario de JBuilder.

Para crear la interfaz de usuario de un bean:

- 1 En el panel del proyecto, seleccione el archivo de bean que JBuilder ha creado automáticamente.
- 2 Pulse sobre la pestaña Diseño para visualizar el diseñador de interfaces de usuario.
- 3 Mediante el diseñador de interfaces de usuario, elabore la interfaz de usuario del bean.

Si desea obtener más información sobre la creación de interfaces de usuario MIDP en el diseñador, consulte *Diseño de aplicaciones con JBuilder*.

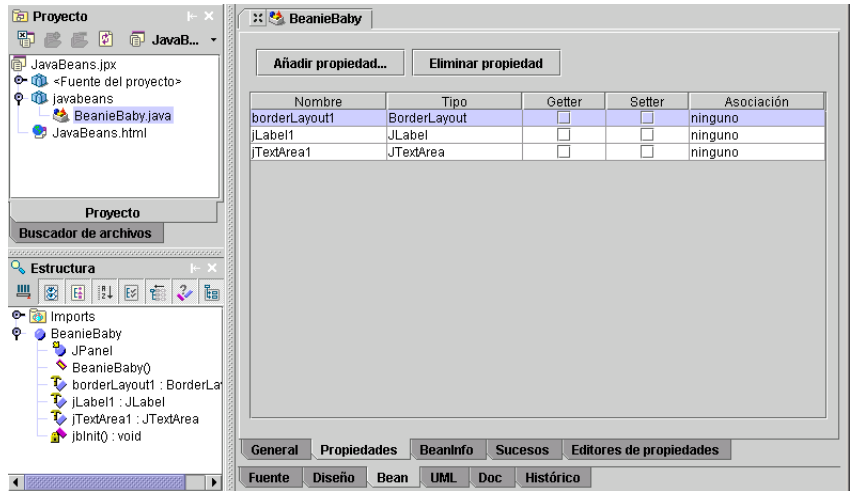
Adición de propiedades a un bean

Las propiedades definen los atributos del bean. Por ejemplo, la propiedad `backgroundColor` describe el color de fondo.

Por lo general, las propiedades de JavaBeans disponen de un método de acceso de escritura y otro de acceso de lectura, también denominados método de obtención y método de asignación, respectivamente. El método de obtención devuelve el valor actual de la propiedad, mientras que el de asignación le asigna un nuevo valor.

Para incorporar una propiedad a un bean:

- 1 Seleccione el componente que desee en el panel del proyecto y haga clic en la pestaña Bean para visualizar los diseñadores BeansExpress.
- 2 Haga clic en la pestaña Propiedades para mostrar el diseñador de propiedades.



- 3 Pulse el botón Añadir propiedad. Aparece el cuadro de diálogo Nueva propiedad.
- 4 Indique el nombre de la propiedad en el cuadro Nombre de la propiedad.
- 5 Indique el tipo en el cuadro Tipo.

Puede escribir uno de los tipos de Java o recurrir al visualizador de paquetes para seleccionar uno, que también puede ser otro JavaBean.

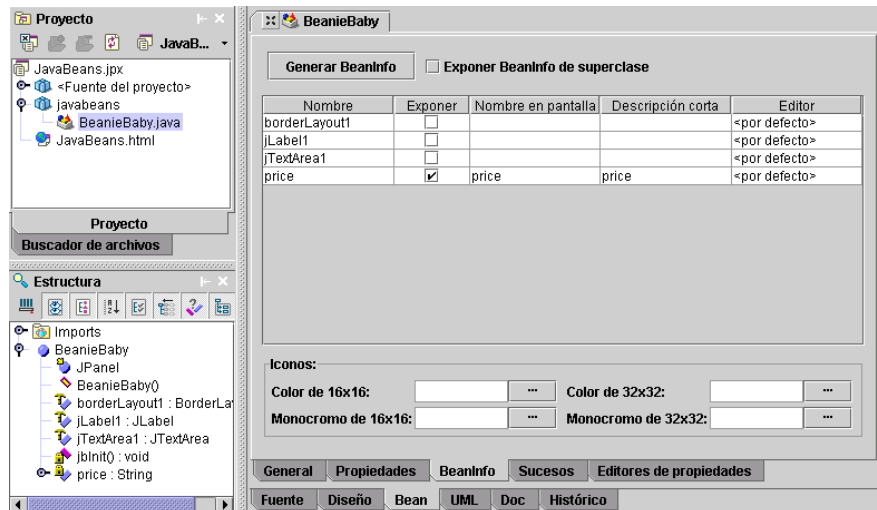
- 6 Deje marcadas las casillas De obtención y De asignación si desea que JBuilder genere los métodos de lectura y escritura del valor de la propiedad.

Si desea que una determinada propiedad sea de sólo lectura, desactive la casilla de selección De asignación.

- 7 Pulse Aceptar.

JBuilder genera el código necesario para la propiedad y añade esta propiedad a la rejilla del diseñador de propiedades. Puede observar que los métodos de acceso de lectura y escritura del bean se han añadido al árbol de componentes. Si hace clic en la pestaña Fuente podrá ver el código generado por JBuilder.

A continuación, se muestra el cuadro de diálogo de la nueva propiedad con los campos obligatorios rellenos:



Los campos Nombre en pantalla y Descripción corta se rellenan automáticamente con los valores por defecto. Este es el código fuente resultante:

```
package myjavabean;

import java.awt.*;
import javax.swing.JPanel;

public class BeanieBaby extends JPanel {
    BorderLayout borderLayout1 = new BorderLayout();
    private float price; // Se ha añadido un campo private
    para el precio

    public BeanieBaby() {
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        this.setLayout(borderLayout1);
    }

    public void setPrice(float price) { // Se ha añadido un método para
    cambiar el
```

```

// precio
    this.price = price;;
}

    public float getPrice() {           // Se ha añadido un método para
obtener el                             // precio

        return price;
    }
}

```

JBUILDER ha añadido un campo `price` a la clase `BeanieBaby`. El campo `price` contiene el valor de la propiedad. Por lo general, los campos de propiedades se declaran `privados`, de forma que sólo puedan acceder a ellos los métodos de obtención y asignación.

También se ha generado un método `setPrice()` que puede cambiar el valor del campo y un método `getPrice()` que puede leer su valor actual.

Una vez instalado el bean en la paleta de componentes de JBUILDER, cuando el usuario lo coloque en el diseñador de interfaces de usuario, la propiedad `price` aparecerá en el Inspector de modo que los usuarios puedan modificar su valor. El código necesario para leer y escribir el valor de `price` se ha incluido en el lugar oportuno.

Modificación de propiedades

Una vez las propiedades son incorporadas a un bean, estas pueden modificarse en cualquier momento mediante el diseñador de propiedades.

Para modificar una propiedad:

- 1 Seleccione el bean mediante el panel del proyecto.
- 2 Haga clic en la pestaña Bean para presentar los diseñadores BeansExpress.
- 3 Haga clic en la pestaña Propiedades.
- 4 Seleccione cualquiera de los campos que aparecen en la rejilla del diseñador de propiedades y efectúe las modificaciones oportunas.

Puede, por ejemplo, cambiar el tipo de la propiedad introduciendo otro.

JBUILDER refleja en el código fuente del bean los cambios realizados en el diseñador de propiedades. También existe la posibilidad de introducir cambios directamente en el código fuente y, siempre que se hayan realizado correctamente, se mostrarán en los diseñadores BeansExpress.

Eliminación de propiedades

Para eliminar una propiedad de un bean:

- 1 En el panel del proyecto, seleccione el bean que contenga la propiedad deseada.
- 2 Haga clic en la pestaña Bean para presentar los diseñadores BeansExpress.
- 3 Haga clic en la pestaña Propiedades.
- 4 En la rejilla del diseñador de propiedades, seleccione la propiedad que desee eliminar.
- 5 Haga clic en Eliminar propiedad.

Tanto el campo de la propiedad como los métodos de acceso se eliminan del código fuente.

Adición de propiedades monitorizables y restringidas

BeansExpress genera el código necesario para crear propiedades monitorizables y restringidas.

Para incorporar una propiedad monitorizable o restringida:

- 1 En el diseñador de propiedades, pulse el botón Añadir propiedad con el fin de mostrar el cuadro de diálogo Nueva propiedad.
- 2 Indique el nombre y el tipo de la propiedad.
- 3 Mediante la lista desplegable Asociación indique si la propiedad ha de ser monitorizable o restringidas.
- 4 Pulse Aceptar.

Si se trata de una propiedad monitorizable, JBuilder añade el correspondiente campo `private` y genera los métodos de lectura y escritura pertinentes.

También crea una instancia de la clase `PropertyChangeSupport`. Por ejemplo, el código generado para una propiedad monitorizable denominada `allTiedUp` es:

```
public class BeanieBaby extends JPanel {
    ...
    private String allTiedUp;
    private transient PropertyChangeSupport propertyChangeListeners = new
        PropertyChangeSupport(this);
    ...
    public void setAllTiedUp(String allTiedUp) {
        String oldAllTiedUp = this.allTiedUp;
        this.allTiedUp = allTiedUp;
        propertyChangeListeners.firePropertyChange("allTiedUp",
            oldAllTiedUp, allTiedUp);
    }
    public String getAllTiedUp() {
```



```

        return allTiedUp;
    }

```

Observe que el método `setAllTiedUp()` incluye código para notificar los cambios del valor de la propiedad a los componentes de monitorización.

JBuilder también genera los métodos de registro a los que llaman los componentes de monitorización de sucesos que desean recibir notificación cuando cambia el valor de la propiedad `allTiedUp`:

```

    public synchronized void removePropertyChangeListener(PropertyChangeListener l) {
        super.removePropertyChangeListener(l);
        propertyChangeListeners.removePropertyChangeListener(l);
    }
    public synchronized void addPropertyChangeListener(PropertyChangeListener l) {
        super.addPropertyChangeListener(l);
        propertyChangeListeners.addPropertyChangeListener(l);
    }

```

El código generado por JBuilder para las propiedades restringidas es similar al anterior, con la diferencia de que los monitores tienen posibilidad de rechazar los cambios del valor de las propiedades.

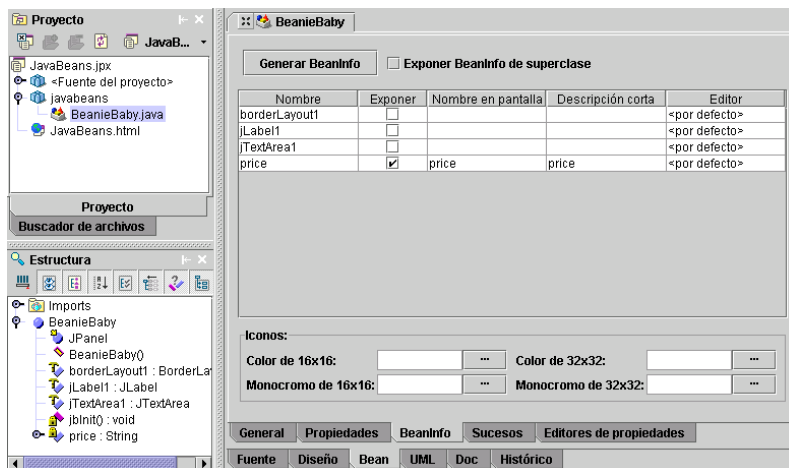
Creación de una clase BeanInfo

La clase `BeanInfo` ofrece la posibilidad de personalizar la presentación de los beans en las herramientas visuales como JBuilder. Por ejemplo, resulta conveniente ocultar algunas de las propiedades con el fin de que no aparezcan en el Inspector de JBuilder. Aunque desde el código fuente seguirá siendo posible acceder a esas propiedades, el usuario no podrá modificarlas en la fase de diseño.

Las clases `BeanInfo` son optativas. Con `BeansExpress` puede generar una clase `BeanInfo` para personalizar un bean. Si piensa utilizar una clase `BeanInfo`, ha de especificar información más detallada en las propiedades que añade al bean.

Especificación de datos BeanInfo de una propiedad

Los datos BeanInfo de una propiedad pueden especificarse en el cuadro de diálogo Nueva propiedad:



Para ocultar una propiedad y que no aparezca en las herramientas de diseño visuales, como el Inspector de JBuilder, cerciórese de que la opción Exponer a través de BeanInfo no esté marcada.

Si desea que la propiedad se muestre con un nombre distinto, introduzca el que desee en el campo Nombre en pantalla.

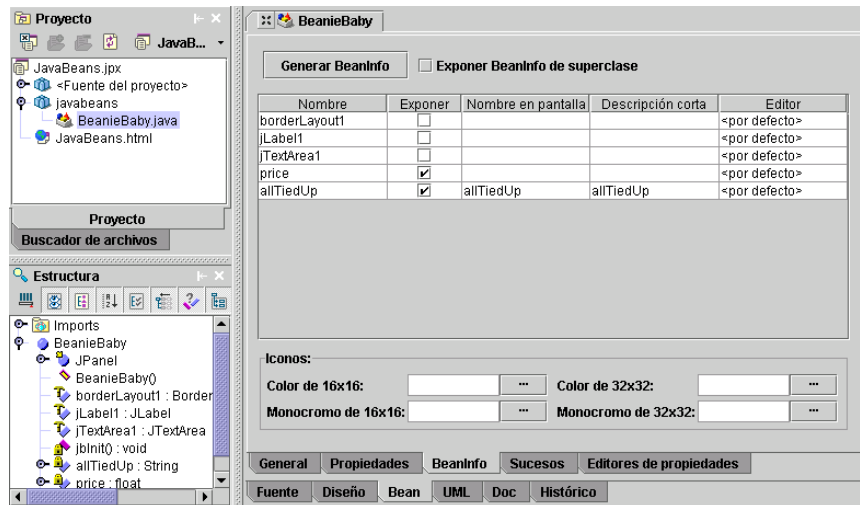
Si desea ofrecer una descripción corta de la propiedad, escribala en el campo Descripción corta. El texto introducido aparece en el Inspector, en forma de ayuda inmediata para la propiedad.

Si ha creado un editor para modificar la propiedad, indique el nombre en el campo Editor. El campo Editor muestra los editores que pueden funcionar en el ámbito actual y tienen capacidad para ese tipo concreto de propiedad.

Utilización del diseñador de BeanInfo

El diseñador de BeanInfo permite modificar los datos BeanInfo de una propiedad, elegir los iconos que han de representar al bean correspondiente en los creadores de aplicaciones como JBuilder y generar automáticamente la clase BeanInfo.

Pulse la pestaña BeanInfo del diseñador de BeansExpress para abrir el Diseñador de BeanInfo.



El diseñador de BeanInfo muestra todas las propiedades añadidas al bean. Si desea rectificar los datos BeanInfo introducidos para una propiedad, puede editar los que desee en la rejilla. El único dato que no puede cambiarse es el nombre de la propiedad.

Si desea elegir un icono para representar al bean, indique las imágenes que desee en el panel Iconos.

Los paneles Iconos permiten elegir distintos iconos para entornos monocromos y en color, y también para diferentes resoluciones de pantalla. Rellene los cuadros según le convenga.

Si la superclase del bean dispone de una clase BeanInfo y desea que los datos de esta última aparezcan en la clase BeanInfo del bean, marque la casilla de selección Exponer BeanInfo de superclase. El código generado contendrá un método `getAdditionalBeanInfo()` que devolverá los datos BeanInfo de la superclase del bean.

Si desea crear una clase BeanInfo para el bean, pulse el botón Generar BeanInfo.

JBuilder creará automáticamente una clase BeanInfo. Los verá aparecer en el panel del proyecto. Para examinar el código generado, seleccione la clase BeanInfo recién creada en el panel del proyecto y aparecerá el código fuente.

Modificación de clases BeanInfo

Puede modificar las clases BeanInfo de los beans mediante BeansExpress.

- 1 Seleccione el bean en el panel del proyecto.
- 2 Haga clic en la pestaña Bean para presentar los diseñadores BeansExpress.

- 3 Pulse sobre la pestaña BeanInfo para visualizar el diseñador de BeanInfo.
- 4 Introduzca los cambios en la rejilla.
- 5 Pulse nuevamente el botón Generar BeanInfo.

Se le avisará de que está a punto de sobrescribir la clase BeanInfo anterior. Si elige Aceptar, la clase se sobrescribirá con los nuevos datos BeanInfo.

Adición de sucesos a un bean

Los JavaBeans pueden:

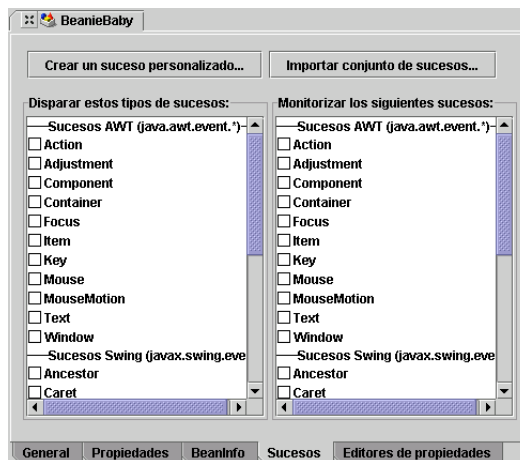
- Generar (o activar) sucesos enviando un objeto de suceso a un objeto monitor.
- Monitorizar sucesos y responder a ellos cuando se produzcan.

BeansExpress genera automáticamente código que permite realizar una o ambas de estas funciones.

Activación de sucesos

Si desea poder enviar objetos de suceso a los componentes monitores de sucesos:

- 1 Seleccione el bean en el panel del proyecto.
- 2 Haga clic en la pestaña Bean para presentar los diseñadores BeansExpress.
- 3 Haga clic en la pestaña Sucesos para mostrar el diseñador de sucesos.



- 4 En la ventana izquierda, seleccione los sucesos que desea el Bean pueda activar.

El diseñador de sucesos añade al bean los métodos de registro de sucesos. Los componentes que deseen recibir notificación de esos sucesos, llamarán a estos métodos. Por ejemplo, si ha seleccionado los sucesos Key, se añadirán los siguientes métodos al bean:

```
package myjavabean;

import java.io.*;
import java.beans.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.util.*;

public class BeanieBaby extends JPanel {
    BorderLayout borderLayout1 = new BorderLayout();
    private float price;
    private transient Vector keyListeners;

    public BeanieBaby() {
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        this.setLayout(borderLayout1);
    }

    public void setPrice(float price) {
        this.price = price;
    }

    public float getPrice() {
        return price;
    }

    public synchronized void removeKeyListener(KeyListener l) {
        super.removeKeyListener(l);
        if(keyListeners != null && keyListeners.contains(l)) {
            Vector v = (Vector) keyListeners.clone();
            v.removeElement(l);
            keyListeners = v;
        }
    }

    public synchronized void addKeyListener(KeyListener l) {
        super.addKeyListener(l);
        Vector v = keyListeners == null ? new Vector(2) : (Vector)
```

```

        keyListeners.clone();
        if(!v.contains(l)) {
            v.addElement(l);
            keyListeners = v;
        }
    }
    ...

}

```

Si un componente desea recibir notificación de los sucesos Key que tengan lugar en el bean, llamará al método `addKeyListener()` de este último y pasará a ser un elemento de `KeyListeners`. Cada vez que se active un suceso Key en el bean, éste se notificará a todos los componentes de monitorización declarados en `KeyListeners`.

La clase también genera métodos de activación de <suceso> que envían un suceso a todos los monitores registrados. Se genera uno de estos sucesos por cada método declarado en la interfaz del Monitor. Por ejemplo, la interfaz `KeyListener` dispone de tres métodos: `keyTyped()`, `keyPressed()` y `keyReleased()`. El diseñador de sucesos añade estos tres métodos de activación de <suceso> a la clase bean:

```

protected void fireKeyPressed(KeyEvent e) {
    if(keyListeners != null) {
        Vector listeners = keyListeners;
        int count = listeners.size();
        for (int i = 0; i < count; i++) {
            ((KeyListener) listeners.elementAt(i)).keyPressed(e);
        }
    }
}

protected void fireKeyReleased(KeyEvent e) {
    if(keyListeners != null) {
        Vector listeners = keyListeners;
        int count = listeners.size();
        for (int i = 0; i < count; i++) {
            ((KeyListener) listeners.elementAt(i)).keyReleased(e);
        }
    }
}

protected void fireKeyTyped(KeyEvent e) {
    if(keyListeners != null) {
        Vector listeners = keyListeners;
        int count = listeners.size();
        for (int i = 0; i < count; i++) {
            ((KeyListener) listeners.elementAt(i)).keyTyped(e);
        }
    }
}

```

```

    }
}

```

Una vez habilitado el bean para generar sucesos, éstos aparecerán en el Inspector de JBuilder cuando el usuario coloque el bean en el diseñador de interfaces de usuario.

Monitorización de sucesos

Los beans también pueden convertirse en monitores de los sucesos que tengan lugar en otros componentes.

Si desea convertir un bean en monitor, seleccione en el diseñador de sucesos el tipo de suceso que desee monitorizar.

En cuanto selecciona un tipo de suceso, el diseñador de sucesos añade la interfaz de monitor correspondiente al bean. Por ejemplo, si ha marcado `java.awt.event.KeyListener`, se añade la frase `implements KeyListener` a la declaración del bean y se implementan los métodos `KeyPressed()`, `KeyReleased()` y `KeyTyped()` con las secciones de código principal vacías.

A continuación, figura un bean con el código generado para implementar la interfaz `KeyListener`:

```

package myBeans;

import java.awt.*;
import javax.swing.JPanel;
import java.beans.*;
import java.awt.event.*;

public class JellyBean extends JPanel implements KeyListener {
    // implements KeyListener
    BorderLayout borderLayout1 = new BorderLayout();

    public JellyBean() {

        try {
            jbInit();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    void jbInit() throws Exception {
        this.setLayout(borderLayout1);
    }

    public void keyTyped(KeyEvent e) {           // Añade un nuevo método
    }

    public void keyPressed(KeyEvent e) {         // Añade un nuevo método

```

```

    }

    public void keyReleased(KeyEvent e) {        // Añade un nuevo método
    }
}

```

Si el bean se ha registrado como monitor en otro componente (llamando al método de registro de sucesos de este último), cuando se produce un suceso del tipo elegido, el componente monitorizado llama a uno de los métodos del bean monitor. Por ejemplo, si se produce un suceso `KeyPressed` en el componente monitorizado, se llama al método `KeyPressed()` del componente monitor. Por tanto, si desea que el componente responda a este suceso, escriba el código oportuno en la sección principal del método `KeyPressed()`, por ejemplo.

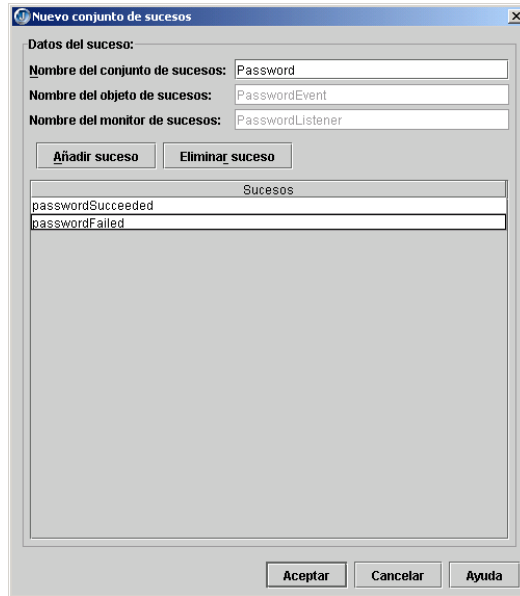
Creación de un conjunto de sucesos personalizado

Puede que, en algún momento, le interese crear un conjunto de sucesos personalizado que describa otros sucesos que puedan tener lugar en un bean determinado. Por ejemplo, si el bean implementa un cuadro de diálogo destinado a la introducción de contraseñas, le interesará activar un suceso cada vez que un usuario introduzca correctamente su contraseña. Posiblemente, también interesará que el bean active un suceso cuando se introduzca una contraseña incorrecta. BeansExpress permite crear un conjunto de sucesos personalizado para tratar estas situaciones.

Para crear un conjunto de sucesos personalizado:

- 1 Seleccione un bean en el panel del proyecto.
- 2 Haga clic en la pestaña Bean para presentar los diseñadores BeansExpress.
- 3 Pulse sobre la pestaña Sucesos.
- 4 Pulse el botón Crear un suceso personalizado.
Aparece el cuadro de diálogo Nuevo conjunto de sucesos.
- 5 Indique el nombre del conjunto de sucesos en el cuadro Nombre del conjunto de sucesos.
Los nombres del objeto de suceso y el monitor de sucesos se generan a partir del nombre del conjunto de sucesos; se muestran en el cuadro de diálogo.
- 6 Seleccione el elemento `dataChanged` y asígnele el nombre del primer suceso del conjunto.

- 7 Si desea añadir otros sucesos, pulse el botón Añadir suceso por cada uno de ellos y sustituya los nombres de los elementos añadidos por los correspondientes nombres de los sucesos:



- 8 Pulse Aceptar.

En el diseñador de sucesos, el nuevo conjunto de sucesos se añade a la lista de tipos de suceso que pueden activarse.

JBuilder genera la clase del objeto de suceso correspondiente al conjunto de sucesos:

```
package myBeans;

import java.util.*;

public class PasswordEvent extends EventObject {
    public PasswordEvent(Object source) {
        super(source);
    }
}
```

JBuilder también genera la interfaz de monitor para el conjunto de sucesos:

```
package myBeans;

import java.util.*;

public interface PasswordListener extends EventListener {
    public void passwordSuccessful(PasswordEvent e);
    public void passwordFailed(PasswordEvent e);
}
```

Creación de editores de propiedades

Los editores de propiedades permiten cambiar los valores de éstas en la fase de diseño. En el Inspector de JBuilder pueden verse distintos tipos de editores de propiedades. Por ejemplo, con ciertas propiedades, al escribir un valor en el Inspector se cambia el valor de la propiedad. Este es el tipo más sencillo de editor de propiedades. En otros casos, aparece un menú de opciones (lista desplegable) que muestra todos los posibles valores de la propiedad. Los editores de propiedades de colores y fuentes son, en realidad, cuadros de diálogo que permiten establecer los valores correspondientes.

Cuando desarrolle sus propios JavaBeans, puede interesarle crear clases de propiedades y los correspondientes editores que puedan modificar sus valores. BeansExpress ofrece la posibilidad de crear editores que presenten listas de opciones.

Para crear un editor de propiedades:

- 1 Haga clic en la pestaña Editores de propiedades de BeansExpress.
- 2 Pulse el botón Crear un editor personalizado.
Aparecerá el cuadro de diálogo Nuevo editor de propiedades.
- 3 Indique el nombre del editor en el cuadro Nombre del editor. Asigne al editor el mismo nombre de la propiedad que va a editar y añádale la palabra Editor. Por ejemplo, si el nombre de la propiedad es `platosPreferidos`, llame al editor `PlatosPreferidosEditor`.
- 4 Seleccione el tipo de editor deseado en la lista desplegable Tipo de editor.
El aspecto del cuadro de diálogo Nueva propiedad cambia en función del tipo de editor seleccionado. En los cuatro apartados siguientes se explica cómo crear un editor de propiedades de cada uno de los cuatro tipos.

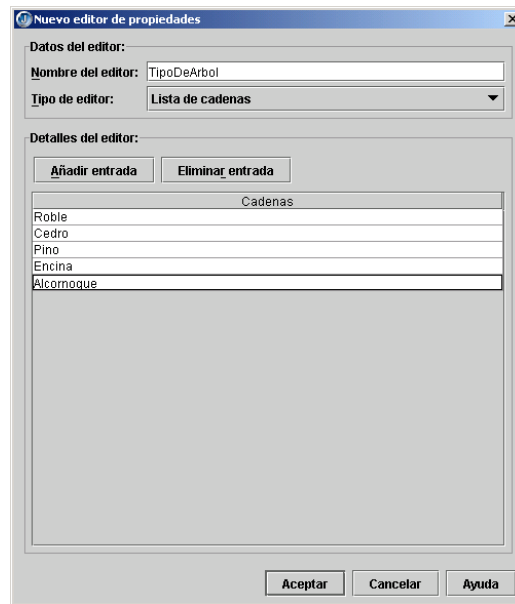
Creación de un editor de lista de cadenas

Una Lista de cadenas es una sencilla lista de cadenas. Aparece en el Inspector como una lista desplegable con las cadenas que se especifiquen. Cuando el usuario selecciona una entrada de la lista, la propiedad que se está modificando adquiere el valor seleccionado.

Para añadir elementos a un editor de listas de cadenas:

- 1 Seleccione Añadir entrada cada vez que desee incluir un nuevo elemento en la lista.
- 2 En cada entrada, introduzca la cadena que desee.

Este es el aspecto que presentaría el cuadro de diálogo Nuevo editor de propiedades:



- 3 Pulse Aceptar y habrá creado otra clase de editor de propiedades.

Para ver el código generado:

- 1 Seleccione la clase del editor de propiedades en el panel del proyecto.
- 2 Pulse sobre la pestaña Fuente.

Creación de un editor de listas de etiquetas de cadena

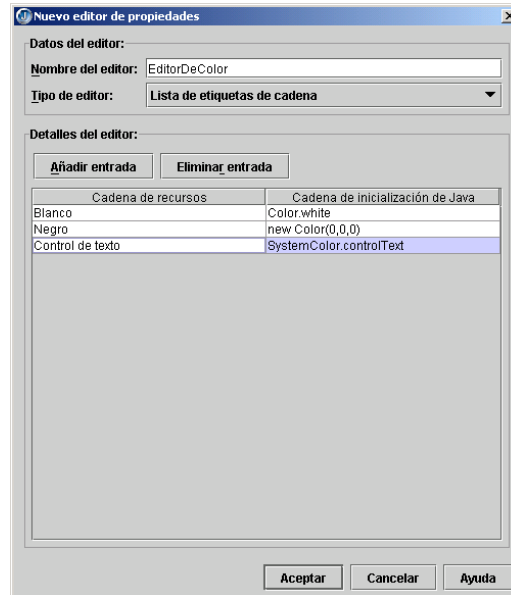
Los editores de propiedades que son editores de lista de etiquetas de cadena también presentan listas de cadenas en el Inspector. Cuando el usuario selecciona una de las entradas, como valor de la propiedad se toma la cadena de inicialización de Java especificada. Las cadenas de inicialización de Java son las que JBuilder emplea en el código fuente generado para las propiedades.

Para añadir elementos a un editor de listas de etiquetas de cadena:

- 1 Seleccione Añadir entrada cada vez que desee incluir un nuevo elemento en la lista.
- 2 En cada entrada, introduzca la cadena que desee presentar en pantalla y la cadena de inicialización asociada a ella.

Si desea incluir una cadena en la lista de cadenas de inicialización, enciérrela entre comillas ("), tal como haría si la estuviese introduciendo en el código fuente.

El cuadro de diálogo presentaría el siguiente aspecto:



3 Pulse Aceptar y habrá creado otra clase de editor de propiedades.

Para ver el código generado:

- 1** Seleccione la clase del editor de propiedades en el panel del proyecto.
- 2** Pulse sobre la pestaña Fuente.

Creación de un editor de listas de etiquetas de enteros

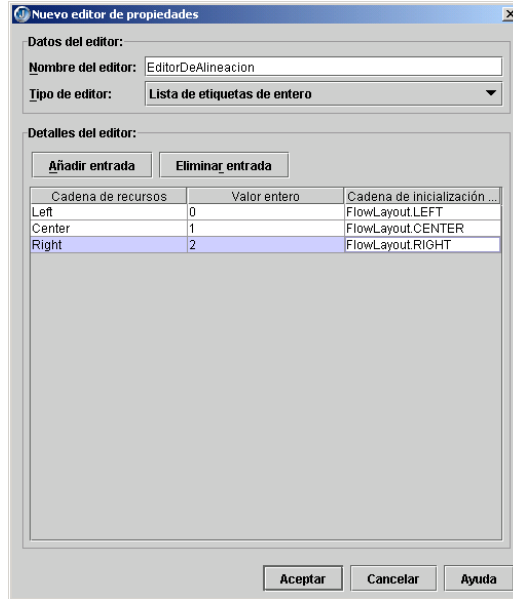
El editor de propiedades de listas de etiquetas de enteros permite editar las propiedades de los enteros. Este editor muestra al usuario una lista de cadenas en el Inspector. Cuando se selecciona un elemento, se usa la cadena de inicialización de Java para configurar la propiedad.

Para añadir elementos a un editor de listas de etiquetas de enteros:

- 1** Seleccione Añadir entrada cada vez que desee incluir un nuevo elemento en la lista.
- 2** En cada entrada, introduzca la cadena que desee presentar en pantalla y la cadena de inicialización asociada a ella.

Si desea incluir una cadena en la lista de cadenas de inicialización, enciérrela entre comillas ("), tal como haría si la estuviese introduciendo en el código fuente.

El cuadro de diálogo presentaría el siguiente aspecto:



- 3 Pulse Aceptar y habrá creado otra clase de editor de propiedades.

Para ver el código generado:

- 1 Seleccione la clase del editor de propiedades en el panel del proyecto.
- 2 Pulse sobre la pestaña Fuente.

Creación de editores de propiedades basados en componentes personalizados

También puede emplear un componente personalizado para editar el valor de una propiedad. Si se selecciona esta opción, se genera un editor de propiedades básico que, para modificar la propiedad, se sirve del editor personalizado que se especifique.

Para especificar un editor de propiedades personalizado:

- 1 Escriba un nombre para la clase del editor que instancie su componente personalizado en el campo Nombre del editor del cuadro de diálogo Nueva propiedad.
- 2 Seleccione Componente editor personalizado en la lista desplegable.
- 3 Especifique el nombre del componente personalizado como el valor del campo Componente editor personalizado.
- 4 Marque la opción Admite paintValue() si desea que el editor personalizado se dibuje a sí mismo.

Para ver el código generado:

- 1 Seleccione la clase del editor de propiedades en el panel del proyecto.
- 2 Pulse sobre la pestaña Fuente.

Admisión de serialización

Serializar un bean es guardar su estado en forma de secuencia de bytes que puede enviarse por una red o almacenarse en un archivo. BeansExpress puede añadir la admisión de serialización a las clases.

Para incorporar la admisión de serialización:

- 1 Seleccione el bean en el panel del proyecto.
- 2 Haga clic en la pestaña Bean para presentar los diseñadores BeansExpress.
- 3 Pulse sobre la pestaña General para visualizar la ficha General.
- 4 Marque la opción Admite serialización.

BeansExpress implementa la interfaz `Serializable` en la clase. Se añaden dos métodos a la clase: `readObject()` y `writeObject()`:

```
void writeObject(ObjectOutputStream oos) throws IOException {
    oos.defaultWriteObject();
}

void readObject(ObjectInputStream ois) throws ClassNotFoundException, IOException {
    ois.defaultReadObject();
}
```

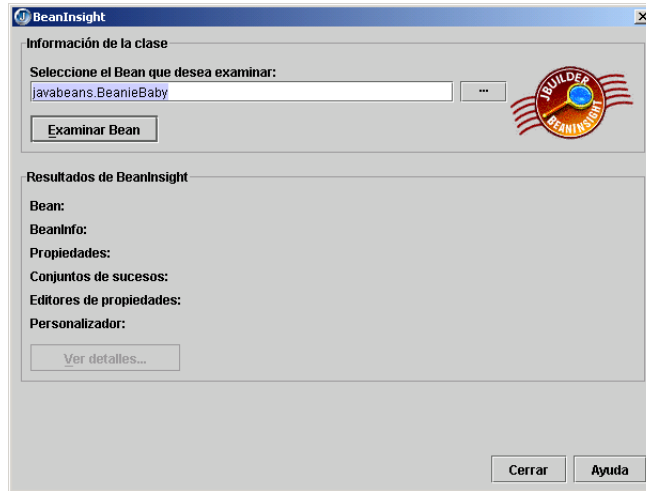
Debe escribir código en ambos para serializar y paralelizar el bean.

Comprobación de la validez de un JavaBean

Una vez se finaliza un bean, puede emplearse BeanInsight con el fin de verificar que el componente recién creado es un JavaBean válido. Si se detectan errores en la clase, BeanInsight informa de ellos. Presenta todas las propiedades, personalizadores y editores de propiedades que encuentra en la clase. También muestra la procedencia de la información de las propiedades, es decir, si se obtiene mediante una clase BeanInfo o mediante introspección.

Para verificar que una clase Java es un JavaBean:

- 1 Elija Herramientas\BeanInsight para visualizar BeanInsight:



- 2 Escriba el nombre del componente que desea examinar o pulse el botón de puntos suspensivos para elegir uno. Si el bean ya está seleccionado en el panel del proyecto cuando aparece BeanInsight, su nombre se muestra en BeanInsight.
- 3 Pulse el botón Examinar Bean para iniciar el proceso de examen.
BeanInsight ofrece un informe del resultado del examen en la sección Resultados de BeanInsight del asistente.
- 4 Si desea consultar todos los detalles del examen de BeanInsight, haga clic en el botón Ver detalles.
- 5 Haga clic en las distintas pestañas para examinar el informe detallado.

Instalación de beans en la paleta de componentes

Una vez se ha desarrollado un componente JavaBean y se ha comprobado su validez, se puede instalar en la paleta de componentes mediante el cuadro de diálogo Propiedades de la paleta. Los archivos de clase del nuevo componente han de encontrarse en la vía de acceso a clases.

Para abrir el cuadro de diálogo Propiedades de la paleta, seleccione Herramientas\Configurar paleta o haga clic con el botón derecho del ratón sobre la paleta de componentes y seleccione Propiedades.

Si desea obtener más información sobre la instalación de componentes, pulse el botón Ayuda del cuadro de diálogo Propiedades de la paleta o, bien, consulte “Cómo añadir un componente a la paleta de componentes” en *Diseño de aplicaciones con JBuilder*.



Presentación de código con UML

Es una función de JBuilder Enterprise.

UML (*Unified Modeling Language*, lenguaje unificado de modelado) es una anotación estándar para el modelado de sistemas orientados a objetos. UML es, básicamente, un lenguaje que describe gráficamente un conjunto de elementos. Sus usos más complejos son los de especificar, presentar, construir y documentar sistemas de software y de otros tipos, así como modelos empresariales. Como los planos de construcción de los edificios, UML proporciona una representación gráfica del diseño de un sistema, que puede ser esencial para la comunicación entre los miembros del equipo y para garantizar la coherencia arquitectónica del sistema.

En la presentación del código, UML constituye una útil herramienta de examen, análisis del desarrollo de las aplicaciones y comunicación del diseño del software. JBuilder utiliza diagramas UML para presentar código y buscar clases y paquetes. Los diagramas UML pueden ayudarle a comprender rápidamente la estructura de un código desconocido, reconocer áreas especialmente complejas, e incrementar su productividad resolviendo problemas con mayor rapidez.

Si desea conocer más a fondo el lenguaje UML, visite estas sedes web:

- Object Management Group, en <http://www.omg.org/>
- Cetus UML links, en http://www.cetus-links.org/oo_uml.html
- UML Central, en http://www.embarcadero.com/support/uml_central.htm
- UML Zone en <http://www.devx.com/uml/>
- UML Dictionary, en <http://softdocwiz.com/UML.htm>

Si desea seguir un tutorial sobre el uso del visualizador UML de JBuilder, consulte el [Capítulo 22, “Tutorial: Visualización de código con el visualizador UML”](#).

Java y UML

Debido a que Java y UML son lenguajes orientados a objetos e independientes de la plataforma, funcionan sin problema alguno de forma conjunta. UML, una valiosa herramienta para la comprensión de Java y las complejas relaciones entre clases y paquetes, ayuda a los desarrolladores a captar el significado de las clases y el paquete completo en el que se encuentran. Sobre todo, puede ayudar a los desarrolladores en Java que se incorporan a un equipo, a familiarizarse rápidamente con la estructura y el diseño del sistema de software.

Términos de Java y UML

Aunque Java y UML utilizan conceptos parecidos, algunos de los términos utilizados para designarlos difieren. UML está diseñado para describir una amplia gama de escenarios distintos, por lo que utiliza términos genéricos para describir las relaciones. En la tabla siguiente se definen los términos propios de Java y sus equivalentes en UML. En esta documentación se utilizan los correspondientes a Java.

Es particularmente importante entender los términos *dependencia* y *asociación*. Las dependencias y las asociaciones son relaciones que pueden tener dos o más clases entre sí. La dependencia se da cuando la implementación de una clase se ve afectada por la implementación de otra. La asociación es un tipo de dependencia en el que un objeto de una clase se puede utilizar para desplazarse a un objeto de otra clase. Se considera que la asociación es una forma más fuerte de dependencia. Si dos clases tienen una relación de dependencia y asociación, el visualizador UML presenta únicamente la segunda.

Tabla 11.1 Términos de Java y UML

Término de Java	Definición de Java	Término de UML	Definición de UML
Herencia	Mecanismo por el cual una clase o interfaz se define como especialización de otra más amplia. Por ejemplo, una subclase (hija, subordinada, descendiente, que amplía,...) hereda la estructura y el comportamiento en lo relativo a campos, métodos, etc., de su superclase (padre, principal, antecesora, ampliada,...). Las clases e interfaces herederas utilizan la palabra clave <code>extends</code> .	Generalización / especialización	Una relación entre un elemento especializado y otro generalizado, en la que el primero incorpora la estructura y el comportamiento del segundo.

Tabla 11.1 Términos de Java y UML (continuación)

Término de Java	Definición de Java	Término de UML	Definición de UML
Dependencia	Relación en la que los cambios realizados en un objeto independiente pueden influir sobre otro que depende de él.	Dependencia	Relación en la que las características semánticas de una entidad dependen de las de otra y están determinadas por ella.
Asociación	Dependencia especializada en la que se almacena una referencia a otra clase.	Relación (asociación)	Relación de estructura que describe los vínculos entre objetos.
Interfaz	Grupo de constantes y declaraciones de métodos que definen la forma de una clase sin implementar los métodos. En la interfaz se establece lo que debe hacer una clase, pero no la forma en que debe hacerlo. Las clases e interfaces que implementan la interfaz utilizan la palabra clave <code>implements</code> .	Realización / interfaz	Conjunto de operaciones que especifican un servicio de una clase o componente. Definen el comportamiento de una abstracción sin implementarlo.
Método	Implementación de una operación definida por una interfaz o una clase.	Operación	Implementación de un servicio que puede solicitar un objeto y puede influir en su comportamiento. Normalmente, en los diagramas UML las operaciones se enumeran debajo de los atributos.
Campo	Variable de instancia o miembro de datos de un objeto.	Atributo	Propiedad de un clasificador, como una clase o una interfaz, que describe los valores que pueden adoptar las instancias de una propiedad.
Propiedad	Información sobre el estado actual de un componente. Las propiedades de los componentes pueden considerarse atributos que tienen un nombre concreto y que puede leer (<code>get</code>) o definir (<code>set</code>) un usuario o un programa. En los diagramas UML existe una propiedad cuando el nombre de un campo coincide con el de un método, este último precedido por "is", "set" o "get". Por ejemplo, un campo llamado <code>parameterRow</code> es una propiedad si tiene un método llamado <code>setParameterRow()</code> .	Atributo	Propiedad de un clasificador, como una clase o una interfaz, que describe los valores que pueden adoptar las instancias de una propiedad.

Tabla 11.1 Términos de Java y UML (continuación)

Término de Java	Definición de Java	Término de UML	Definición de UML
		Valor etiquetado	Una extensión de las propiedades de un elemento UML. Los valores etiquetados constan de una etiqueta, que es el nombre de una propiedad, y un valor. Los valores etiquetados aparecen debajo del nombre de la superclase y todas las subclases de la clase a la que corresponde el diagrama.

JBuilder y UML

En lugar de sustituir las herramientas de UML, JBuilder se centra en la presentación del código y en la adaptación de los diagramas UML al lenguaje Java. Las funciones de UML integradas en JBuilder permiten examinar visualmente paquetes y clases, con el fin de ayudar en el diseño, la comprensión y la solución de problemas en el proceso de desarrollo de aplicaciones.

JBuilder ofrece dos diagramas UML:

- Diagramas limitados a las dependencias de paquetes
- Diagramas combinados de clases

Si desea una descripción más detallada de los elementos de los diagramas de paquetes y clases, consulte [“Glosario de los diagramas UML de JBuilder” en la página 11-9](#).

El visualizador UML de JBuilder proporciona características adicionales como, por ejemplo, el perfeccionamiento (refactoring) del código, la personalización de la representación en UML y la visualización de Javadoc y código fuente.

Importante Los diagramas de clases UML de los archivos fuente pueden ser más detallados que si sólo hay archivos de clase disponibles, ya que los datos privados y los miembros de clase están ocultos. Por ejemplo, JBuilder excluye los campos privados y los miembros de los archivos de clase.

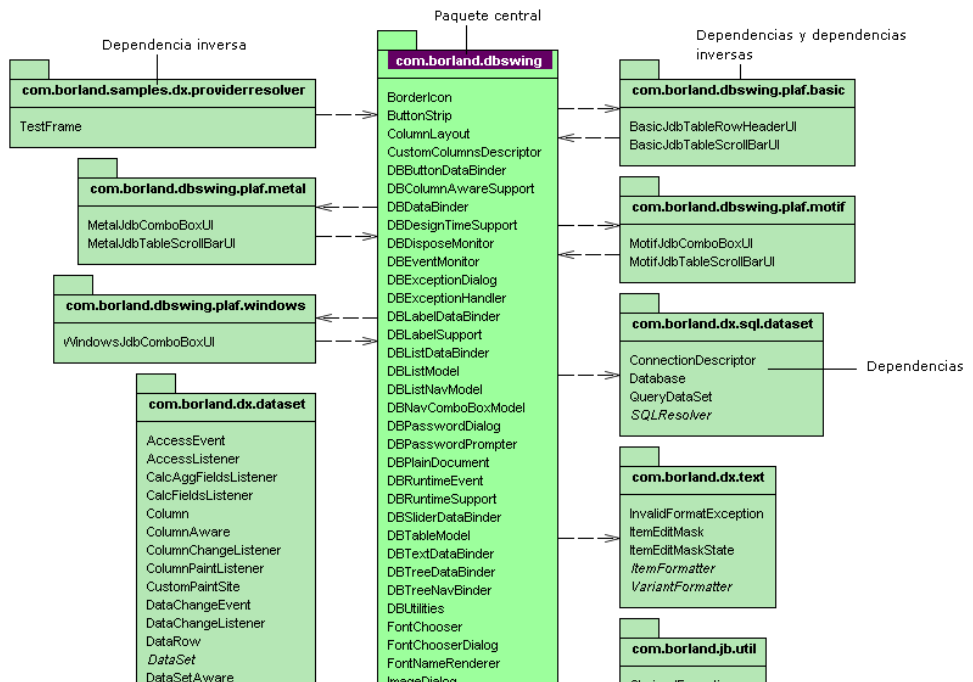
Consulte

- [“Glosario de los diagramas UML de JBuilder” en la página 11-9](#)
- [“Personalización de diagramas UML” en la página 11-20](#)
- [Capítulo 13, “Perfeccionamiento de código”](#)

Diagramas limitados a las dependencias de paquetes

El *diagrama de paquete* se centra alrededor de un paquete central y muestra sólo las dependencias de ese paquete. Las dependencias entre los paquetes dependientes no se muestran. Las dependencias directas e inversas aparecen a la izquierda, a la derecha o a los dos lados del paquete central. En los paquetes dependientes sólo se enumeran las clases de las que depende el paquete central. El paquete central actual se muestra por defecto sobre un fondo verde brillante. Los demás paquetes se muestran por defecto sobre un fondo de un verde más oscuro.

Figura 11.1 Diagrama de paquetes



Aunque en el diagrama UML se muestran únicamente el paquete actual y los importados, se pueden incluir referencias de las clases de las bibliotecas del proyecto. Con el fin de incluir las referencias de biblioteca, active la opción Incluir referencias de archivos de clase de las bibliotecas del proyecto de la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto/Propiedades de proyecto/General).

Consulte

- “Presentación de diagramas de paquetes” en la página 11-15
- “Inclusión de referencias de bibliotecas de proyecto” en la página 11-21

Diagramas combinados de clases

Los *diagramas combinados de clases* del archivo Java fuente o de clase muestran la clase en el centro del diagrama, con las relaciones alrededor. Sólo se muestran las relaciones de las clases incluidas en la vía de acceso del proyecto y las bibliotecas añadidas a éste.

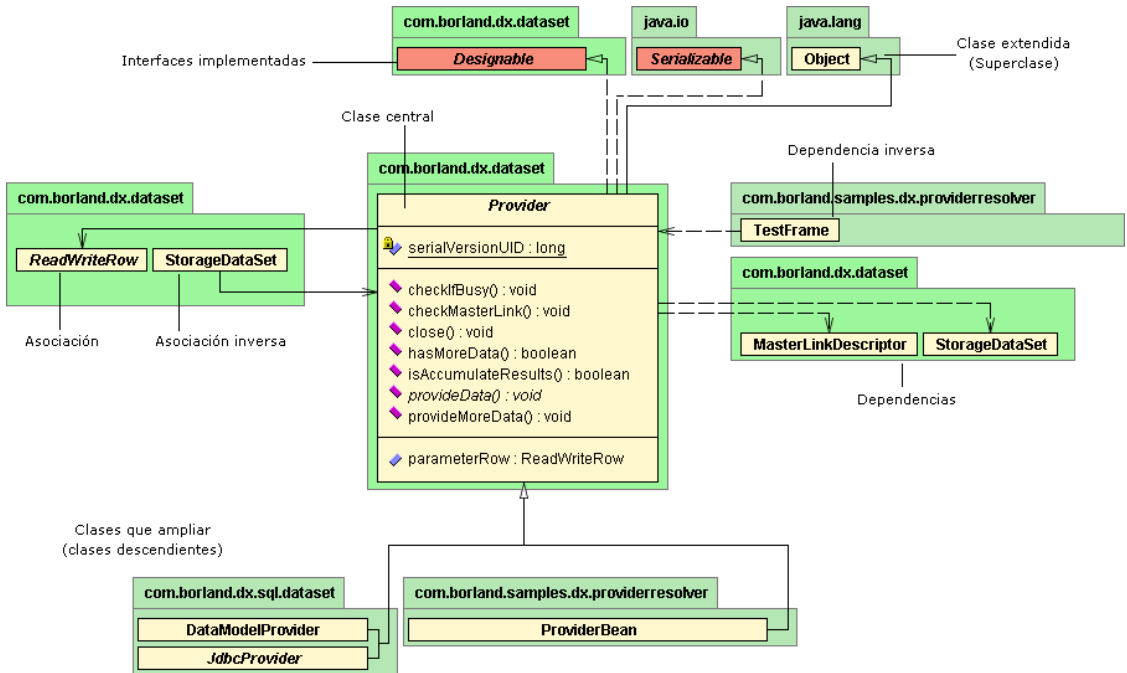
Las relaciones se muestran en cuatro zonas: a la izquierda, a la derecha, arriba y abajo. Las relaciones que se muestran a la izquierda de la clase central son asociaciones, y las que se muestran a la derecha, dependencias. Las clases ampliadas asociadas y las interfaces implementadas se muestran sobre la clase central. Las clases que amplían e implementan se muestran bajo la clase central.

Las asociaciones y las dependencias se organizan en otras tres zonas: superior, central e inferior. En la zona superior se muestran las relaciones inversas, como las clases asociadas o dependientes de la clase central. La zona central contiene las relaciones mixtas. La zona inferior contiene las clases con las que está asociada o de las que depende la clase central.

Los diagramas de clases UML típicos muestran las clases dentro de sus paquetes. Para mejorar la legibilidad, el visualizador UML sólo muestra una agrupación de clases por paquete. Por ejemplo, si la clase central depende de cuatro de las cinco clases del paquete A y la quinta clase tiene una asociación inversa con la central, todo el paquete A se muestra en la parte superior izquierda del diagrama, para indicar la relación más importante contenida en el grupo.

Tenga en cuenta que los diagramas de clases UML de los archivos fuente pueden ser más detallados que si sólo hay archivos de clase disponibles, ya que los datos privados y los miembros de clase están ocultos. Por ejemplo, en el código ofuscado, JBuilder excluye los campos privados y los miembros de los archivos de clase.

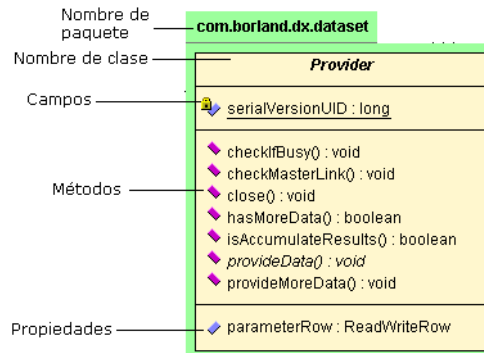
Figura 11.2 Diagrama combinado de clases



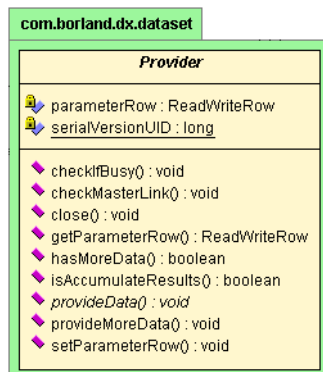
El diagrama UML de clases muestra la clase en el centro, en un rectángulo cuyo color de fondo por defecto es el amarillo. Rodeándola, aparece el paquete, con su nombre en una pestaña en la parte superior. La clase se divide en varias secciones, separadas por líneas horizontales, en el orden siguiente:

- Nombre de la clase en la parte superior
- Campos y propiedades*
- Métodos; funciones de obtención* y definición*
- Propiedades* en la parte inferior

*Por defecto, las propiedades se muestran en la sección inferior del diagrama de clases. La opción **Mostrar propiedades por separado** se configura en la ficha UML del cuadro de diálogo **Preferencias (Herramientas|Preferencias)**. Si esta opción está desactivada, las propiedades se muestran en las secciones adecuadas, con campos y métodos. Consulte [“Configuración de preferencias UML” en la página 11-22](#).

Figura 11.3 Diagrama de clases con las propiedades por separado

Nota Los iconos indican si un campo, método o propiedad es `privado`, `público`, o `protegido`. En el caso de las propiedades se muestra la accesibilidad del método de acceso. Consulte [“Iconos de accesibilidad” en la página 11-11](#).

Figura 11.4 Diagrama de clases sin las propiedades por separado

Aunque en el diagrama UML se muestran únicamente el paquete actual y los importados, se pueden incluir referencias de las clases de las bibliotecas del proyecto. Con el fin de incluir las referencias de biblioteca, active la opción **Incluir referencias de archivos de clase de las bibliotecas del proyecto de la ficha General del cuadro de diálogo Propiedades de proyecto (Project Properties)**.

Consulte

- [“Glosario de los diagramas UML de JBuilder” en la página 11-9](#)
- [“Presentación de diagramas de clases” en la página 11-15](#)
- [“Inclusión de referencias de bibliotecas de proyecto” en la página 11-21](#)
- [“Inclusión de referencias del código generado” en la página 11-22](#)

Glosario de los diagramas UML de JBuilder

El visualizador UML de JBuilder utiliza un subconjunto de los símbolos estándar de los diagramas UML en la representación de las relaciones entre paquetes y clases. En la tabla siguiente se definen los símbolos de UML que utiliza el visualizador UML, así como las carpetas que se muestran en el panel de estructura de JBuilder.

Tabla 11.2 Definiciones de los diagramas UML


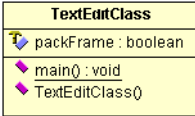
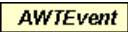


Diagrama	Definición	Descripción del diagrama	Ejemplo de diagrama
Clases ampliadas	Clases de las que otra clase hereda los atributos (campos y propiedades) y métodos. También se denominan superclases, clases padre, clases base, clases principales y clases antecesoras.	Una línea continua con un gran triángulo que apunta de la subclase a la superclase. Aparece en la parte superior del diagrama UML.	
Clases	Estructuras que definen objetos. Las definiciones de clases definen campos y métodos.	Se muestra un cuadro rectangular de fondo amarillo por defecto en el que se incluyen el nombre en la parte superior y los campos, métodos y propiedades en la parte inferior.	
Clases abstractas	Clases de las que no se pueden crear instancias pero son antecesoras de otras clases.	Se muestran en cursiva.	
Clases herederas	Clases que amplían la superclase. También se denominan subclases, clases hijas, clases subordinadas y clases descendientes.	Una línea continua con un gran triángulo que apunta de la subclase a la superclase. Aparece en la parte inferior del diagrama UML.	
Clases de implementación	Clases que implementan la interfaz central.	Una línea discontinua con un gran triángulo que apunta de la clase de implementación a la interfaz heredada. Aparece en la parte inferior del diagrama UML.	

Tabla 11.2 Definiciones de los diagramas UML (continuación)


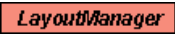

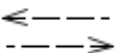
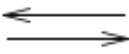
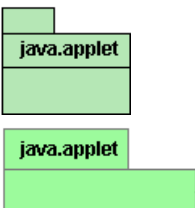
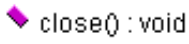

Diagrama	Definición	Descripción del diagrama	Ejemplo de diagrama
Interfaces ampliadas	Interfaces antecesoras heredadas por una descendiente.	Una línea continua con un gran triángulo que apunta de la subinterfaz a la interfaz antecesora. Aparece en la parte superior del diagrama UML.	
Interfaces	Grupos de constantes y declaraciones de métodos que definen la forma de una clase sin implementar los métodos. Las interfaces permiten establecer lo que debe hacer una clase, pero no la forma en que debe hacerlo.	Un rectángulo con el fondo naranja por defecto. El nombre de la interfaz se presenta en cursiva.	
Interfaces implementadas	Interfaces que implementa la clase central.	Una línea discontinua con un gran triángulo que apunta de la clase de implementación a la interfaz implementada. Aparece en la parte superior del diagrama UML.	
Dependencias directas e inversas	Relaciones en las que los cambios realizados en el objeto utilizado pueden influir sobre el que lo utiliza.	Una línea discontinua acabada en punta de flecha.	
Asociaciones directas e inversas	Dependencias especializadas en las que se almacena una referencia a otra clase.	Una línea continua acabada en punta de flecha.	
Paquetes	Conjuntos de clases relacionadas.	Un rectángulo con una pestaña en la parte superior y el nombre del paquete en esta pestaña o bajo ella. El paquete actual se muestra por defecto sobre un fondo verde brillante. Los demás paquetes se muestran por defecto sobre un fondo de un verde más oscuro.	
Métodos	Operaciones definidas en una clase o una interfaz.	Se enumeran bajo los nombres y los campos, e incluyen el tipo de devolución.	

Tabla 11.2 Definiciones de los diagramas UML (continuación)

Diagrama	Definición	Descripción del diagrama	Ejemplo de diagrama
Métodos abstractos	Métodos sin implementación.	Se muestran en cursiva.	 <i>provideData() : void</i>
Redefinir métodos	Métodos de una subclase que tienen una implementación diferente de métodos de la que se define en la superclase.	Se muestran en azul.	 <i>cancelLoad() : void</i>
Miembros/Campos	Variables de instancia o miembros de datos de un objeto.	Se enumeran bajo el nombre de la clase, e incluyen el tipo de devolución.	 <code>contentPane : JPanel</code>
Propiedades	Se habla de propiedades cuando el nombre de un método que coincide con el nombre de un campo va precedido de "is", "get", o "set". Por ejemplo, un nombre de campo <code>parameterRow</code> con un método <code>getParameterRow()</code> constituye una propiedad.	Si se establece así en la ficha UML del cuadro de diálogo Preferencias (Herramientas Preferencias), las propiedades se muestran por separado en la sección inferior del diagrama de clases.	 <code>parameterRow : ReadWriteRow</code>
Estático	Con ámbito de clase.	Los miembros, campos, variables y métodos estáticos se muestran subrayados en el diagrama UML.	 <u><code>serialVersionUID : long</code></u>
Valores etiquetados	Una extensión de las propiedades de un elemento UML.	Se representa con una cadena entre corchetes y muestra más adelante el nombre de la superclase y de las subclases, si las hay, de la clase del diagrama.	{subclases = 2}

Iconos de accesibilidad

En UML se utilizan iconos para representar la accesibilidad de las clases, como `public`, `private`, `protected` y `package`. En los diagramas se pueden utilizar los iconos de accesibilidad de JBuilder o los iconos UML estándar.




Los iconos de visibilidad de JBuilder son los que se muestran en el panel de estructura al visualizar el código fuente. Para obtener más información sobre las definiciones de los iconos, consulte "El panel de estructura y los iconos UML de JBuilder" en *Introducción a JBuilder*. Para utilizar los iconos de JBuilder en los diagramas UML, active la opción Utilizar iconos de visibilidad de la ficha UML del cuadro de diálogo Preferencias (Herramientas|Preferencias). Esta opción se encuentra activada por defecto.

Figura 11.5 Iconos de visibilidad de JBuilder

FlowLayout
<div><div><div>+</div><div>CENTER : int</div></div><div><div>+</div><div>LEADING : int</div></div><div><div>+</div><div>LEFT : int</div></div><div><div>+</div><div>RIGHT : int</div></div><div><div>+</div><div>TRAILING : int</div></div><div><div>+</div><div>align : int</div></div><div><div>+</div><div>newAlign : int</div></div><div><div>+</div><div>currentSerialVersion : int</div></div><div><div>+</div><div>serialVersionOnStream : int</div></div><div><div>+</div><div>serialVersionUID : long</div></div></div>
<div><div><div>+</div><div>addLayoutComponent() : void</div></div><div><div>+</div><div>FlowLayout() : void</div></div><div><div>+</div><div>FlowLayout() : void</div></div><div><div>+</div><div>FlowLayout() : void</div></div><div><div>+</div><div>getAlignment() : int</div></div><div><div>+</div><div>layoutContainer() : void</div></div><div><div>+</div><div>minimumLayoutSize() : Dimension</div></div><div><div>+</div><div>preferredLayoutSize() : Dimension</div></div><div><div>+</div><div>removeLayoutComponent() : void</div></div><div><div>+</div><div>setAlignment() : void</div></div><div><div>+</div><div>toString() : String</div></div><div><div>+</div><div>moveComponents() : void</div></div><div><div>+</div><div>readObject() : void</div></div></div>
<div><div><div>+</div><div>hgap : int</div></div><div><div>+</div><div>vgap : int</div></div></div>

En UML la accesibilidad de las clases se representa por medio de iconos más genéricos, definidos en la tabla siguiente. Con el fin de utilizar los iconos de visibilidad estándar de UML en los diagramas, desactive la opción Utilizar iconos de visibilidad de la ficha UML del cuadro de diálogo Preferencias (Herramientas|Preferencias).

Tabla 11.3 Iconos de accesibilidad

Accesibilidad	iconos UML	Icono de método de JBuilder ¹
public	+	
private	-	
protected	#	

1. Si desea una lista completa de los iconos de UML de JBuilder, busque "iconos" en el índice de ayuda en pantalla.

FlowLayout
<ul style="list-style-type: none"> + CENTER : int + LEADING : int + LEFT : int + RIGHT : int + TRAILING : int + align : int + newAlign : int - currentSerialVersion : int - serialVersionOnStream : int - serialVersionUID : long
<ul style="list-style-type: none"> + addLayoutComponent() : void + FlowLayout() : void + FlowLayout() : void + FlowLayout() : void + getAlignment() : int + layoutContainer() : void + minimumLayoutSize() : Dimension + preferredLayoutSize() : Dimension + removeLayoutComponent() : void + setAlignment() : void + toString() : String - moveComponents() : void - readObject() : void
<ul style="list-style-type: none"> + hgap : int + vgap : int

Presentación de los diagramas UML

JBUILDER proporciona un visualizador UML con el que se puede presentar el código con diagramas UML. El visualizador UML, que se abre desde la pestaña UML del panel de contenido, muestra los diagramas de clases y paquetes estándar de UML. Cuando selecciona la pestaña UML, JBUILDER carga los archivos de clase para determinar sus relaciones, los cuales el visualizador UML utilizará para obtener la información de paquete y clase para los diagramas.

Para obtener un diagrama UML actualizado y exacto, siempre es mejor compilar antes de seleccionar la pestaña UML. El visualizador UML muestra los archivos fuente de Java de forma dinámica aunque no se hayan compilado, pero sólo si se encuentran en la vía de acceso a archivos fuente. En el visualizador de UML aparece un mensaje que indica que es posible que el diagrama UML no sea exacto. Sin embargo, si un archivo no está en la vía de acceso a archivos fuente, es necesario crear antes el archivo `.class`. Se muestra un mensaje en el que se solicita que se compile el proyecto con el fin de crear los archivos de clase para el diagrama UML. Si los archivos de clase son antiguos (por ejemplo, el archivo fuente se ha modificado pero no se ha vuelto a realizar la compilación), en el visualizador de UML aparece un mensaje que indica que es posible que el diagrama UML no sea exacto.

El visualizador UML también acepta la representación en diagramas de dependencias inversas, de clases a JSP (páginas JavaServer). Por ejemplo, un bean generado por el Asistente para JSP conduce a la JSP que lo utiliza. No es necesario que se trate de un bean de JSP; puede ser cualquier clase utilizada por la página JSP.

Por defecto, JBUILDER no incluye referencias de las bibliotecas del proyecto ni del código generado (archivos IIOP y stubs EJB) en los diagramas UML. Si

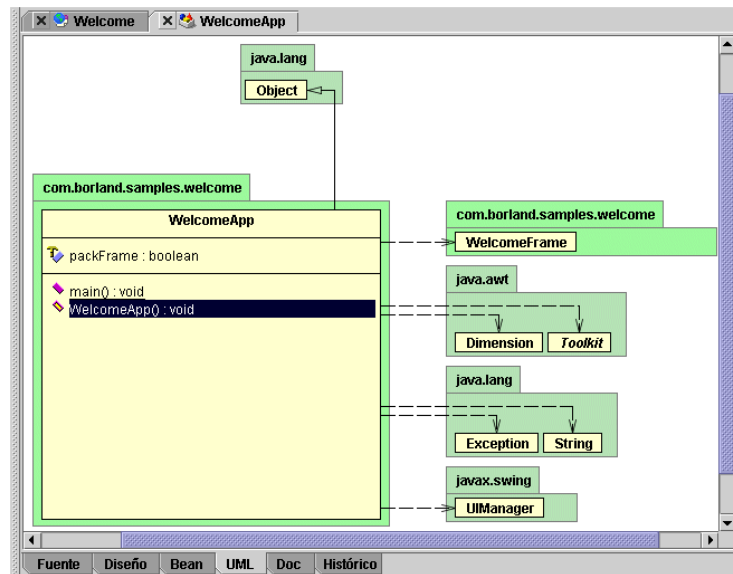
desea incluirlas en los diagramas UML, configure las opciones de la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto | General). Si desea más información sobre opciones del depurador, consulte [“Inclusión de referencias de bibliotecas de proyecto” en la página 11-21](#) y [“Inclusión de referencias del código generado” en la página 11-22](#).

El visualizador UML de JBuilder

El visualizador UML de JBuilder presenta el código Java en diagramas UML. El visualizador UML es una herramienta de representación de clases que utiliza símbolos UML en la representación de las relaciones entre paquetes y clases. El visualizador UML de JBuilder proporciona diversas funciones para personalizar la presentación de los diagramas, desplazarse por ellos y por el código fuente, mostrar las clases internas, el código fuente, Javadoc, crear e imprimir imágenes y para el perfeccionamiento. Para ver un diagrama UML en JBuilder, abra un paquete o una clase y pulse la pestaña UML del panel de contenido.

Nota Si el proyecto es voluminoso, la primera presentación del diagrama UML puede llevar bastante tiempo. Antes de generar los diagramas, JBuilder debe cargar las clases para determinar sus relaciones.

Figura 11.6 Visualizador UML



Presentación de diagramas de paquetes

Para ver un diagrama limitado a las dependencias de paquetes:

- 1 Elija Proyecto|Ejecutar Make del proyecto o Proyecto|Generar el proyecto para llevar a cabo la compilación.
- 2 Haga doble clic en el paquete, en el panel del proyecto, o haga clic con el botón derecho del ratón y elija Abrir.
- 3 Para ver el diagrama del paquete, abra la pestaña UML del panel de contenido.

Nota Si el nodo del paquete no está disponible, elija Proyecto|Propiedades de proyecto|General y habilite la opción Activar la localización y compilación de paquetes fuente.

Presentación de diagramas de clases

Para ver un diagrama combinado de clases:

- 1 Elija Proyecto|Ejecutar Make del proyecto con el fin de realizar la compilación del proyecto o el archivo Java.
- 2 Haga doble clic en un archivo Java, en el panel del proyecto, o haga clic con el botón derecho del ratón y elija Abrir.
- 3 Abra la pestaña UML de la parte inferior del panel de contenido para ver el diagrama UML de clase.

Presentación de las clases internas

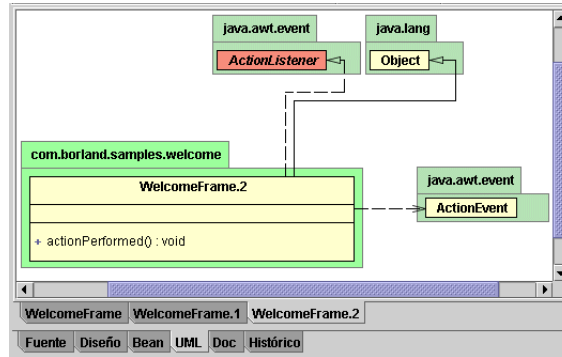
Una clase puede contener varias clases, incluidas las clases internas y las clases internas anónimas. Si ocurre esto, el visualizador UML presenta una interfaz de usuario con una clase por pestaña.

Sólo se muestra el diagrama de una clase interna anónima si el cursor del editor se coloca en ella o si se va a ella desde otro diagrama. Para colocar el cursor en el editor, abra la pestaña Fuente de un archivo fuente abierto, sitúe el cursor en el lugar deseado y abra la pestaña UML.

Las clases internas anónimas seleccionadas se guardan en memoria hasta que se cierra el archivo, de manera que puedan acumularse en forma de pestañas en el visualizador UML. Las dependencias de las clases internas anónimas se incluyen en las clases que las contienen.

El visualizador UML utiliza la posición del cursor en el editor para determinar la clase, el método o el campo seleccionado en el visualizador UML. No obstante, si no se cambia esta posición del cursor en las visitas posteriores al visualizador, se conserva la última selección. En los campos y los métodos, el cursor debe encontrarse entre el primer carácter y el último de la definición, y no al principio de la línea.

Figura 11.7 Presentación de las clases internas



Presentación del código fuente

En los diagramas de clases es posible desplazarse al código fuente y volver al diagrama UML. Haga doble clic en la clase central, en un método, en un campo o en una propiedad para ver su archivo de código fuente. El cursor del editor se coloca en el lugar adecuado. Cuando se sitúa el cursor en una clase, un método, un campo o una propiedad, en el editor, el elemento se resalta en el diagrama UML. En el editor, abra la pestaña UML para volver al diagrama UML.

El visualizador UML tiene una opción de presentación del código en el menú contextual: Ir a código fuente. Seleccione Ir a código fuente para presentar el código en el editor. Esto puede resultar útil para examinar el código fuente de otras clases e interfaces en los diagramas de clases y paquetes.

El visualizador UML también proporciona ayuda inmediata en la que se muestra la lista de argumentos de los métodos. Sitúe el ratón sobre un método para ver su ayuda inmediata. Si se coloca el puntero en el nombre de una clase aparece su nombre completo, que incluye el nombre del paquete.

Visualización de Javadoc

Existen varias formas de acceder al Javadoc de paquetes, interfaces, clases, métodos y campos desde los diagramas UML.

- Seleccione un elemento en el diagrama UML, haga clic con el botón derecho del ratón y elija Ver Javadoc.
- Seleccione un elemento en el diagrama UML y pulse *F1*.
- Seleccione un elemento en el panel de estructura y pulse *F1*.

El visualizador de ayuda de JBuilder muestra el Javadoc, que se genera a partir de los comentarios Javadoc del archivo de código fuente o a partir de la información disponible, como las firmas de los métodos.

Nota Si se ha ejecutado Javadoc con la herramienta **javadoc** o el Asistente para Javadoc, se incluye más información.

Consulte

- [“Visualización de Javadoc” en la página 15-30](#)

Uso del menú contextual

El visualizador UML tiene un menú contextual que permite acceder rápidamente a los comandos habituales. Para abrirlo, haga clic con el botón derecho del ratón en un elemento del visualizador UML. Si desea información sobre estos comandos y su función, consulte la documentación correspondiente:

- Comandos del menú Perfeccionamiento: Buscar referencias, Renombrar, Mover, Cambiar parámetros, Extract Interface, Introduce Superclass. (Consulte el [Capítulo 13, “Perfeccionamiento de código”](#))
- Guardar diagrama: [“Creación de imágenes de diagramas UML” en la página 11-23](#)
- Activar filtrado de clases: [“Filtrado de paquetes y clases” en la página 11-20](#)
- Ir a diagrama: [“Desplazamiento por diagramas” en la página 11-18](#)
- Ir a código fuente: [“Presentación del código fuente” en la página 11-16](#)
- Mostrar Javadoc: [“Visualización de Javadoc” en la página 11-16](#)

Desplazamiento de la vista

Existen varias maneras de desplazar el diagrama UML dentro del visualizador UML:

- Hacer clic y arrastrarlo con el ratón
- Por medio de las teclas Arriba y Abajo
- Con los cursores
- Con las barras de desplazamiento

Se puede utilizar el ratón para mover la vista hacia arriba y hacia abajo. Pulse sobre el fondo del diagrama, haga clic y arrastre el diagrama. Las teclas *RePag* y *AvPag*, así como los cursores arriba y abajo, también pueden desplazar la vista hacia arriba y hacia abajo. La vista también se puede desplazar de forma manual por medio de las barras de desplazamiento.

Sugerencia Para maximizar la vista, seleccione Ver|Maximizar panel de contenido.

Actualización de la vista

Si desea actualizar los diagramas UML después de realizar cambios en el proyecto, utilice uno de estos métodos:



- Genere el proyecto de nuevo (Proyecto|Generar el proyecto).
- Pulse el botón Actualizar de la barra de herramientas del panel de proyecto.

Desplazamiento por diagramas



Haga doble clic en el nombre de una clase o un paquete en el diagrama a fin de ver su diagrama UML. Cuando se selecciona un elemento en el diagrama UML, cambia su color de fondo. Después de realizar una selección, se pueden utilizar las teclas de *flecha* para subir y bajar por el diagrama. Si no hay ningún elemento seleccionado, las teclas *Re Pág* y *Av Pág* desplazan el diagrama arriba y abajo. Si desea ver diagramas UML mostrados previamente, utilice los botones del historial del navegador Inicio, Avanzar y Atrás disponibles en la barra de herramientas principal para desplazarse fácilmente hacia delante y atrás entre los diagramas UML.

También es posible desplazarse seleccionando clases y paquetes en el panel de estructura. Para seleccionar una clase o un paquete, haga clic sobre él. Haga doble clic en una clase para ver su diagrama. Haga clic con el botón derecho en un paquete y elija Abrir para presentar su diagrama.

Existe también un comando de menú en el menú contextual del visualizador UML para visualizar diagramas UML: Ir a diagrama. Haga clic con el botón derecho del ratón en el nombre de un paquete, una clase o una interfaz, en el diagrama UML, y elija Ir a diagrama.

Consulte

- [“UML y el panel de estructura” en la página 11-18](#)

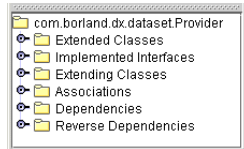
UML y el panel de estructura

El panel de estructura, situado en la esquina inferior izquierda del diagrama UML, proporciona una vista en árbol de las relaciones que contienen las carpetas ampliables por categoría como, por ejemplo, Dependencias y Dependencias inversas. Si alguna de las categorías no está incluida en el diagrama, la carpeta correspondiente no aparece. Estas carpetas permiten desplazarse a otros diagramas, y pueden proporcionar información que no aparece en ellos, ya que reflejan las relaciones independientemente de la configuración de filtrado y cualquier otra restricción. Por ejemplo, aunque se hayan filtrado determinadas clases y paquetes siguen apareciendo en el panel de estructura. Si desea más información sobre el filtrado de diagramas UML, consulte [“Personalización de diagramas UML” en la página 11-20](#). Si

desea ampliar y contraer iconos de carpeta en el panel de estructura, haga doble clic en ellos o pulse el icono conmutador de ampliación.

Los paquetes y las clases que no se muestran en el diagrama aparecen en un color más claro en el panel de estructura. No aparecen en el diagrama si no se filtran o si se han eliminado por ser redundantes y para obtener más claridad.

Figura 11.8 Panel de estructura de los diagramas UML



El panel de estructura también se puede utilizar para seleccionar y dirigirse a clases y paquetes. Si se selecciona una clase, una interfaz o un paquete en el panel de estructura, queda seleccionado en el diagrama. Si se hace doble clic en una clase o un paquete en el panel de estructura se accede a su diagrama UML. Haga doble clic en un paquete y elija Abrir para presentar el diagrama UML de paquetes.

Si desea buscar rápidamente un paquete o una clase en el panel de estructura, pase el foco al árbol y empiece a escribir el nombre deseado. Si desea obtener más información, consulte el tema "Búsqueda en árboles" del capítulo "El entorno de JBuilder" de *Introducción a JBuilder*.

Diagramas de paquetes

Las carpetas de los diagramas de paquetes pueden incluir cualquiera de los siguientes elementos o todos ellos:

- Dependencias
- Dependencias inversas

Si desea ver la definición de estos términos, consulte ["Glosario de los diagramas UML de JBuilder" en la página 11-9](#).

Cuando se abre un paquete dependiente, se muestran todas sus clases y las relaciones que tienen con el paquete central. Esto permite saber cuáles son las clases que originan la dependencia.

Diagramas de clases

Las carpetas de los diagramas de clases pueden incluir cualquiera de los siguientes elementos o todos ellos:

- Clases ampliadas
- Interfaces ampliadas
- Interfaces implementadas
- Clases herederas
- Clases de implementación
- Asociaciones
- Asociaciones inversas

- Dependencias
- Dependencias inversas

Si desea ver la definición de estos términos, consulte [“Glosario de los diagramas UML de JBuilder” en la página 11-9](#).

Personalización de diagramas UML

Aunque no es posible manipular los diagramas UML desplazando o cambiando de tamaño sus elementos, es posible personalizar la presentación en los cuadros de diálogo Propiedades de proyecto y Preferencias. Por ejemplo, se puede filtrar proyecto por proyecto los elementos que aparecen en los diagramas, y también se pueden incluir referencias de bibliotecas de proyecto. También se puede personalizar la presentación general del diagrama UML mediante la configuración del orden, la fuente, los colores y otras opciones.

Definición de las propiedades del proyecto

En el cuadro de diálogo Propiedades de proyecto es posible configurar diversas propiedades de los diagramas UML.

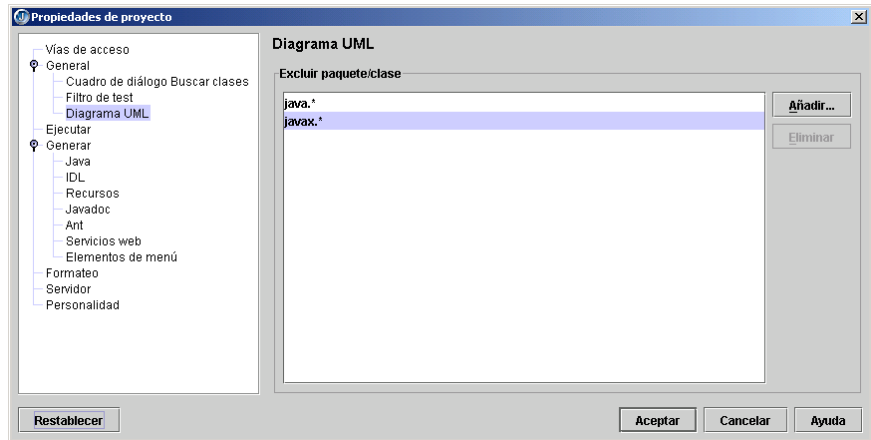
- Filtrado: disponible en la ficha Filtro de diagrama UML
- Referencias de biblioteca: disponible en la ficha General
- Referencias del código generado: disponible en la ficha General

Para abrir el cuadro de diálogo Propiedades de proyecto, elija Proyecto | Propiedades de proyecto o haga clic con el botón derecho del ratón en el archivo del proyecto, en el panel del proyecto, y elija Propiedades.

Filtrado de paquetes y clases

En la ficha Filtros de diagrama UML del cuadro de diálogo Propiedades de proyecto, puede excluir los paquetes y clases de los diagramas UML del proyecto. Seleccione Proyecto | Propiedades de proyecto | General | Diagrama

UML. Después, pulse Añadir si desea añadir clases o paquetes a la lista de exclusiones. Las clases y paquetes de la lista se excluyen del diagrama UML.



Vuelva al diagrama UML y observe que las clases y paquetes de la lista Excluir paquete/clase se excluyen del diagrama, aunque todavía se puede acceder a ellos en el panel de estructura. Para desactivar provisionalmente el filtrado de paquetes y clases aplicado al diagrama UML, haga clic en él con el botón derecho del ratón y desactive la opción Activar filtrado de clases en el menú contextual del visualizador UML.

Importante

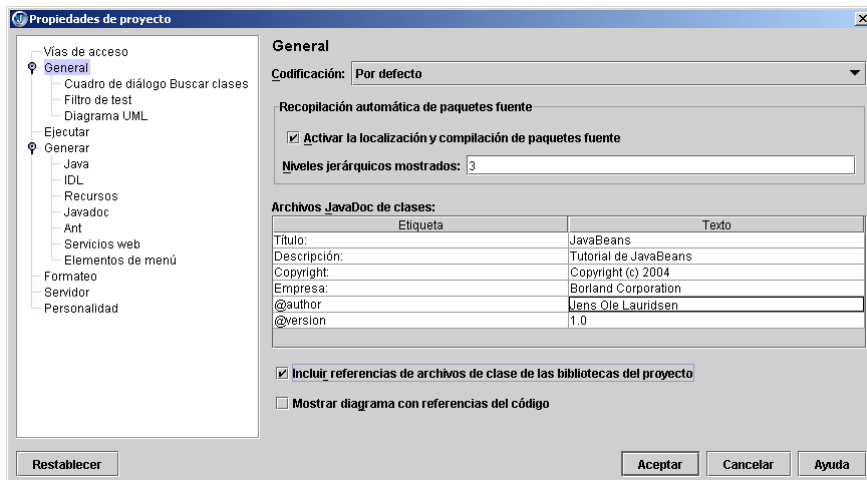
Si está activado el filtrado en el cuadro de diálogo Propiedades de proyecto se filtran todos los diagramas de ese proyecto. La desactivación del filtrado desde el menú contextual de un diagrama no se aplica a los demás. Si se desplaza a otro diagrama del proyecto, el filtrado sigue activado. Cuando se cierra el archivo o el paquete, la configuración recupera los valores definidos para el proyecto.

Inclusión de referencias de bibliotecas de proyecto

Habitualmente, las bibliotecas proporcionan servicios a las aplicaciones que se construyen a partir de ellas, pero no saben nada acerca de sus usuarios. Para mostrar estas relaciones, necesita incluir referencias de las bibliotecas.

Con el fin de incluir las referencias de biblioteca en los diagramas UML, active la opción Incluir referencias de archivos de clase de las bibliotecas del proyecto de la ficha General del cuadro de diálogo Propiedades de proyecto. Por defecto, esta opción está desactivada y se excluyen las dependencias inversas entre las bibliotecas y los proyectos.

Nota Si el proyecto es muy voluminoso, esta opción puede aumentar significativamente el tiempo que tarda JBuilder en cargar las clases y crear el diagrama UML.



Consulte

- [Paso 4: Añadir referencias de bibliotecas](#) en [página 22-9](#) de “[Tutorial: Visualización de código con el visualizador UML](#)”

Inclusión de referencias del código generado

En los diagramas UML también se pueden incluir referencias del código fuente generado como, por ejemplo, archivos IIOP y stubs EJB. Para ello, seleccione la opción **Mostrar diagrama con referencias del código** en la ficha General de **Propiedades de proyecto**.

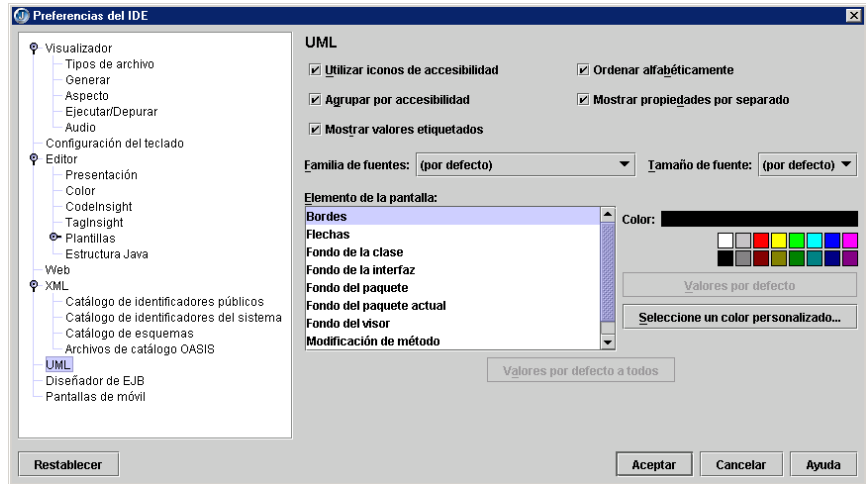
Nota Si el proyecto es muy voluminoso, esta opción puede aumentar significativamente el tiempo que tarda JBuilder en cargar las clases y crear el diagrama UML.

Configuración de preferencias UML

La ficha UML del cuadro de diálogo **Preferencias** (Herramientas|Preferencias) proporciona opciones destinadas a la personalización general de los diagramas en el visualizador UML de JBuilder. Para abrir la ficha UML, seleccione **Herramientas|Preferencias|UML**.

Aquí se pueden modificar los iconos de visibilidad del diagrama UML, el orden, la agrupación, la presentación de propiedades, la presentación del valor de las etiquetas, el tipo y tamaño de letra y el color de los distintos elementos de la pantalla, como por ejemplo los métodos de redefinición,

fondos de las clases y flechas. Si desea más información, pulse el botón Ayuda de la ficha UML.



Consulte

- "Personalización del IDE de JBuilder", en *Introducción a JBuilder*

Creación de imágenes de diagramas UML

El visualizador UML permite guardar diagramas UML como imágenes. Sin embargo, el tamaño de la imagen y el número de colores pueden constituir una limitación. En ese caso se muestra un mensaje de error. JBuilder acepta el formato de imagen PNG (Portable Network Graphics, gráficos portátiles de red).

Para guardar el diagrama UML como imagen, haga clic con el botón derecho del ratón en el visualizador UML y elija Guardar diagrama. Escriba un nombre de archivo en el cuadro de diálogo Guardar diagrama. Se le añade automáticamente la extensión .png.

Impresión de diagramas UML

Si desea obtener una copia impresa del diagrama UML, pulse Imprimir en la barra de herramientas principal o elija el comando Imprimir (Archivo|Imprimir). El comando Configurar página (Archivo|Configurar página) permite definir el encabezado, los márgenes y la orientación. El diagrama se imprime a un tamaño ligeramente inferior al que se muestra en la pantalla. Si el diagrama no cabe en una sola página, se imprime en varias.

Importante

Para que la opción de impresión esté disponible, es necesario desplazar el foco al diagrama UML.

Perfeccionamiento y Buscar referencias

Es posible acceder a las funciones de perfeccionamiento de JBuilder desde el visualizador UML. Existen varias formas de acceder al perfeccionamiento desde el visualizador UML:

- **Cambiar nombre:** haga clic con el botón derecho del ratón en el nombre de un paquete, una clase, un campo, un método o una propiedad, en el diagrama UML, y elija Cambiar nombre en el menú contextual. Seleccione en el diagrama UML el nombre de un paquete, una clase, un campo o un método en el diagrama UML, y pulse *Intro*. Escriba el nuevo nombre en el cuadro de diálogo Cambiar nombre.
- **Desplazar:** haga clic con el botón derecho del ratón en el nombre de una clase, en el diagrama UML, y elija Mover en el menú contextual.
- **Cambiar parámetros:** haga clic con el botón derecho del ratón en el nombre de un método, en el diagrama UML, y elija Cambiar parámetros en el menú contextual.
- **Extraer interfaz:** haga clic con el botón derecho del ratón en la declaración de la clase en el diagrama UML, y elija Extraer interfaz en el menú contextual.
- **Introducir superclase:** haga clic con el botón derecho del ratón en la declaración de la clase en el diagrama UML, y elija Introducir superclase en el menú contextual.

Antes de realizar el perfeccionamiento, puede ser conveniente buscar todos los archivos fuente que utilizan un símbolo seleccionado. Para localizar todas las referencias a un símbolo, selecciónelo en un diagrama UML o en el editor. Haga clic con el botón derecho en el símbolo y elija Buscar referencias.

Consulte

- [Capítulo 13, “Perfeccionamiento de código”](#)
- [“Búsqueda de referencias” en la página 13-4](#)



Comparación de archivos y versiones

JBuilder ofrece muchas posibilidades para comparar archivos, constatar las diferencias entre varios y entre diferentes versiones del mismo archivo, y gestionar, fusionar y restaurar las diferencias. Se puede acceder a estas funciones mediante el cuadro de diálogo Comparar archivos, el cuadro de diálogo Gestionar etiquetas locales y la vista del histórico.

Glosario de gestión de versiones

En este apartado se utilizan algunos términos especializados:

PLAZO	Definición
<i>revisión, versión</i>	Ambos términos se pueden intercambiar en este contexto. Cada vez que se modifica o se revisa un archivo se hace una nueva versión o revisión de ese archivo.
<i>diferencias</i>	Dif es la abreviatura de diferencias. Las diferencias son las áreas de texto que difieren entre dos archivos o entre dos versiones del mismo archivo.
<i>bloque de diferencias</i>	Es una parte que difiere entre dos archivos o dos versiones del mismo archivo. Se aprecian más fácilmente en las vistas de diferencias, en las que los dos archivos o versiones de archivos se fusionan en un archivo aparentemente original y se resaltan las diferencias entre ellos. Observe que los mecanismos de gestión de diferencias de JBuilder evalúan los archivos físicamente, no de forma lógica. Solamente comparan el texto de los archivos o las versiones, no su lógica ni su estructura de código.

PLAZO	Definición
<i>restaurar, restablecer</i>	Consiste en hacer que la versión anterior de un archivo vuelva a ser la versión actual.
<i>control de versiones</i>	Es un modo de gestionar las revisiones que guarda un registro de todos los cambios realizados en todos los archivos sometidos a control de versiones. Los sistemas de control de versiones también incluyen normalmente otras funciones de gestión de código, como la ramificación y el etiquetado de versiones.
<i>copia de seguridad</i>	Es una versión del archivo que se guarda en el directorio de copia de seguridad local que se le ha asignado. JBuilder utiliza las copias de seguridad como versiones anteriores del archivo. El número de copias de seguridad que se desea mantener se establece en la ficha Editor de Herramientas Opciones del editor. Si se alcanza el límite, se conservan las últimas versiones y se vuelven a numerar de forma apropiada.
<i>búfer</i>	Es la versión más moderna que se utiliza en JBuilder. Incluye los cambios no guardados.
<i>repositorio</i>	En el control de versiones, es el lugar donde se guardan las copias maestras del archivo y los históricos de revisiones.
<i>área de trabajo</i>	En el control de versiones, son los archivos locales y la estructura de directorio que se cambian directamente.

Comparación de dos archivos

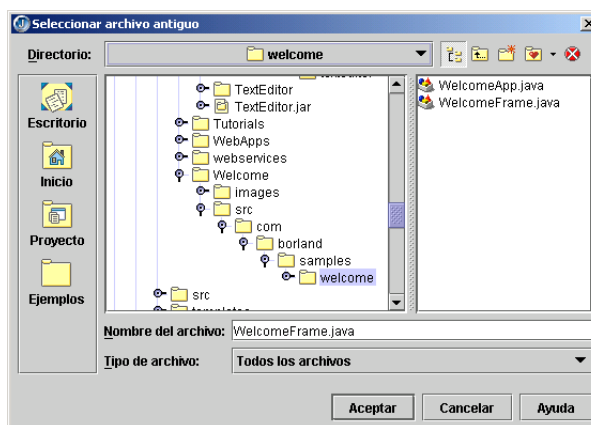
Es una función de JBuilder Developer y Enterprise.

El cuadro de diálogo Comparar archivos de JBuilder permite comparar dos archivos cualesquiera.

Para comparar dos archivos de texto:

- 1 Seleccione Archivo|Comparar archivos.

Aparece el cuadro de diálogo Seleccionar archivo antiguo:

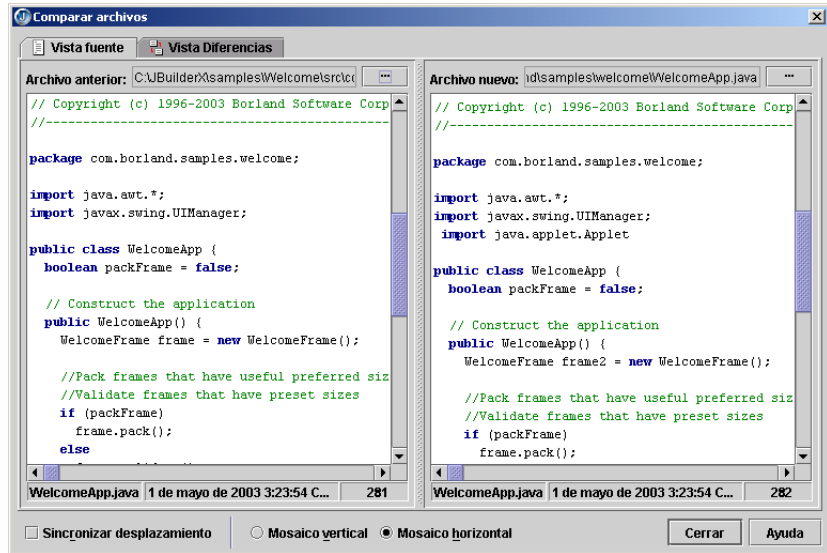


- 2 Seleccione el primer archivo y pulse Aceptar.

Aparece el cuadro de diálogo Seleccionar archivo nuevo.

3 Seleccione el segundo archivo y pulse Aceptar.

Se muestra el cuadro de diálogo Comparar archivos.



La *Vista Fuente* muestra un archivo junto al otro. Para dividir la vista en cualquier dirección, pulse el botón de radio apropiado. Para sincronizar el desplazamiento de los dos paneles, active la casilla de selección Sincronizar desplazamiento.

La Vista Fuente admite funciones básicas de proceso de texto. Esto permite llevar a cabo cambios en los dos búfers. Si se realizan cambios se habilitan las opciones Actualizar archivo y Guardar archivo del menú contextual, que aparece cuando se hace clic con el botón derecho:



La *Vista Diferencias* muestra las diferencias (bloques de diferencias) entre dos archivos. Es posible deshacer bloques de diferencias individuales en el nuevo archivo, de modo que los bloques de código vuelvan a coincidir con el archivo antiguo. El primer archivo seleccionado se muestra como la versión más antigua, y el segundo como la más reciente. Esto significa que si los archivos actual y anterior tienen dos bloques de diferencias en la misma parte del código, JBuilder puede igualarlos sustituyendo el bloque del archivo más moderno por el bloque correspondiente del archivo más antiguo.

Si se modifican los archivos con ayuda del cuadro de diálogo Comparar archivos, cuando se cierra se solicita al usuario que indique si desea guardar los cambios.

Utilización de etiquetas locales en la gestión de revisiones de archivos locales

El cuadro de diálogo Gestionar etiquetas locales (Proyecto|Gestionar etiquetas locales) permite gestionar los archivos fuente del proyecto mediante etiquetas locales. La creación de una etiqueta local aplica la etiqueta a los archivos en los directorios fuente (tal y como se definen en la ficha Fuente de la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto), y a los archivos bajo los nodos Módulo Web para el proyecto activo. Las etiquetas locales ayudan a marcar versiones importantes del proyecto, como una versión estable, o una versión anterior a cambios significativos. Las etiquetas locales permiten deshacer los cambios realizados en los archivos fuente del proyecto al estado en que se encontraban cuando se creó la etiqueta local, y permiten comprobar de forma segura las diferentes modificaciones.

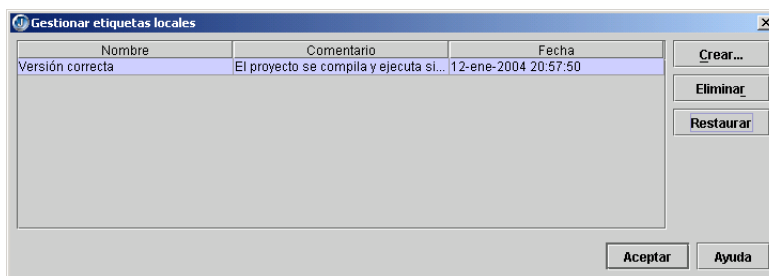
Por ejemplo, si un proyecto es estable, se puede crear una etiqueta local antes de añadir al código una característica posiblemente desestabilizante. Si después de realizar los cambios se decide que es mejor no mantenerlos, se pueden restaurar todos los archivos fuente a su estado estable anterior, mediante la etiqueta local. Aunque las etiquetas locales no están pensadas para sustituir a un sistema de control de versiones (SCV), esta flexibilidad ofrece una alternativa sencilla para poner a prueba cambios temporales en el código fuente.

Las etiquetas locales se implementan mediante el directorio de copia de seguridad local del proyecto. Cuando se restaura a una etiqueta local en concreto, se sobrescriben todos los cambios realizados en los archivos fuente desde que se creó la etiqueta y se eliminan todos los archivos añadidos al proyecto desde ese momento. Las etiquetas locales se muestran en la columna Etiqueta de las fichas Contenido e Información de la vista del histórico. Para más información consulte [“La vista Histórico” en la página 12-6](#).

Para aplicar una etiqueta local al proyecto:

- 1 Seleccione Proyecto|Gestionar etiquetas locales.

Se abre el cuadro de diálogo Gestionar etiquetas locales:



- 2 Haga clic en Crear para que se abra el cuadro de diálogo Crear etiqueta.

- 3 Escriba el nombre y la descripción de la etiqueta y haga clic en Aceptar.

La etiqueta local se añade a la lista de etiquetas existentes del cuadro de diálogo Gestionar etiquetas locales.

Nombre	Comentario	Fecha
Versión correcta	El proyecto se compila y ejecuta si...	12-ene-2004 20:57:50
Pre err. 1581	Previa a la corrección del error n.1...	15-ene-2004 20:56:50

- 4 Pulse Aceptar para cerrar el cuadro de diálogo Gestionar etiquetas locales.

La etiqueta local añadida aparece en la columna Etiqueta de las fichas Contenido e Información de la vista del histórico.

Para deshacer los cambios de una etiqueta local en concreto:

- 1 Seleccione Proyecto|Gestionar etiquetas locales para abrir el cuadro de diálogo Gestionar etiquetas locales.
- 2 Seleccione la etiqueta local de la lista.
- 3 Haga clic en Restaurar.

Se abre un cuadro de diálogo Restaurar "<label_name>" para confirmar la acción.

- 4 Haga clic en Restaurar para confirmar la acción y terminar de deshacer los cambios en los archivos fuente.

Se abre una ventana Restaurar "<label_name>", que muestra el avance mientras todos los archivos fuente modificados vuelven al estado en que estaban cuando se creó la etiqueta local seleccionada.

- 5 Haga clic en Cerrar para cerrar la ventana Restaurar "<label_name>" y haga clic en Aceptar para cerrar el cuadro de diálogo Gestionar etiquetas locales.

Sugerencia Una etiqueta local se puede eliminar en cualquier momento sin que afecte al contenido de los archivos fuente del proyecto. La eliminación de etiquetas inutilizadas del proyecto activo proporciona pequeñas mejoras en el rendimiento.

Precaución Si se está utilizando un sistema de control de versiones junto con el proyecto y se restaura hasta una etiqueta local, se sobrescriben todas las modificaciones en los archivos fuente, incluidas las actualizaciones desde el repositorio realizadas desde la creación de la etiqueta.

La vista Histórico






La vista Histórico permite examinar versiones anteriores de un archivo de cualquier tipo, incluidas las versiones de copias de seguridad, los cambios locales guardados y el búfer del archivo activo. Si el archivo abierto está sometido a control de versiones, en la vista Histórico se ofrecen todos los tipos de revisiones. Si no es así, la vista Histórico permite gestionar las revisiones por medio de las copias de seguridad, los cambios almacenados y el búfer no almacenado.

Como la vista Histórico está en el panel de contenido, sus fichas reflejan el archivo activo actual. Haga doble clic en un archivo, en el panel del proyecto, para activarlo. Pulse la pestaña de un archivo abierto, en el panel de contenido, para activarlo.

Existen cuatro fichas en la vista Histórico. Ofrecen las siguientes funciones:

- La ficha Contenido, disponible en todas las ediciones de JBuilder, muestra versiones actuales y antiguas de los archivos.
- La ficha Diferencias muestra las diferencias entre las versiones seleccionadas del archivo.
- La ficha Información muestra todas las etiquetas y comentarios del archivo activo.
- La ficha Conflictos en la fusión muestra los conflictos de fusión notificados por el sistema de control de versiones.

Todas las fichas, excepto Conflictos en la fusión, cuentan con tablas de revisión que enumeran todas las versiones del archivo activo. Las listas de revisión muestran el número de versión del archivo, el tipo y la fecha de revisión, y otros datos. Si desea ordenar las listas de revisión, pulse el encabezado de la columna que desea utilizar como criterio de ordenación. JBuilder utiliza en las tablas de revisiones los siguientes iconos para distinguir los tipos de control de versiones:

Icono	Descripción:
	La versión del archivo que se encuentra en el búfer. La versión del búfer incluye los cambios no guardados.
	La versión guardada más antigua del archivo.
	Una versión de copia de seguridad del archivo.
	La versión del archivo extraída del repositorio.
	Una versión del archivo procedente de un sistema de control de versiones.

Sugerencia: Cuando el cursor se encuentre en la lista de revisiones, desplácese por ella pulsando la tecla *Intro* o las teclas de desplazamiento (flecha).

Las fichas Contenido y Diferencias cuentan con un visualizador de código fuente. El visualizador de código fuente muestra el código fuente de la versión del archivo seleccionada en la lista de revisiones. Se puede copiar y pegar del visualizador de código fuente al editor, pero no es posible modificar el código directamente en el visualizador. De esta forma se simplifica la recuperación del trabajo antiguo a la vez que se protege la integridad de las versiones anteriores de los archivos.



Actualizar información de revisiones

El botón Actualizar información de revisiones permite actualizar la vista de la lista de revisiones de modo que incluya los cambios realizados por otros usuarios en el archivo del repositorio. Sólo resulta útil si se utiliza el control de versiones. Este botón está disponible en las tres fichas del Histórico.



Restaurar la revisión anterior

El botón Restaurar la revisión anterior hace que la versión seleccionada vuelva a ser la actual. La versión antigua se convierte en la más reciente. Este botón está en las fichas Contenido e Información.

Nota

Cuando se vuelve a una versión anterior se pierden todos los cambios no guardados que se encontraban en el búfer del editor. Si se utiliza un sistema de control de versiones, todas las versiones intermedias se almacenan en el repositorio.



Sincronizar desplazamiento

Sincronizar desplazamiento sincroniza el desplazamiento entre los visualizadores de código fuente de las fichas Contenido y Diferencias del Histórico y el editor. Asocia la línea de texto donde está el cursor con la línea correspondiente de la otra vista. Si en esa parte del archivo no hay texto coincidente, asocia los números de línea. Este botón está disponible en la ficha Contenido y Diferencias.

Sincronizar desplazamiento también es una casilla de selección del cuadro de diálogo Archivo/Comparar archivos.



Comparación inteligente

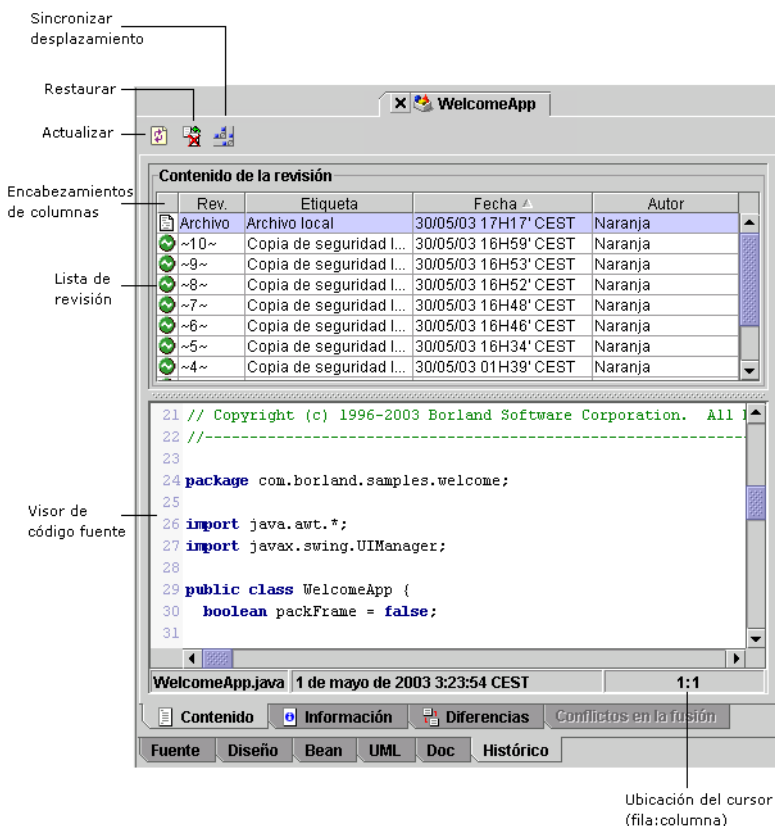
Si la opción Comparación inteligente se encuentra activada, el formato del código y los cambios en los espacios en blanco no se tienen en cuenta cuando se comparan dos versiones de un archivo con JBuilder. Este botón se encuentra en la ficha Diferencias. La Comparación inteligente es una característica de las versiones Developer y Enterprise de JBuilder.

ficha Contenido

La ficha Contenido de la vista del histórico muestra todas las versiones disponibles del archivo activo. La lista de revisiones, en la parte superior, permite ordenar las versiones del archivo por tipo de versión, número de revisión, etiqueta, fecha de cambio y autor. Haga clic en el encabezado que

desee utilizar como criterio de ordenación. Si se pulsa de nuevo se invierte el orden (de ascendente a descendente y viceversa).

En el visualizador de código fuente se muestra el código de la versión seleccionada en la lista de revisiones:



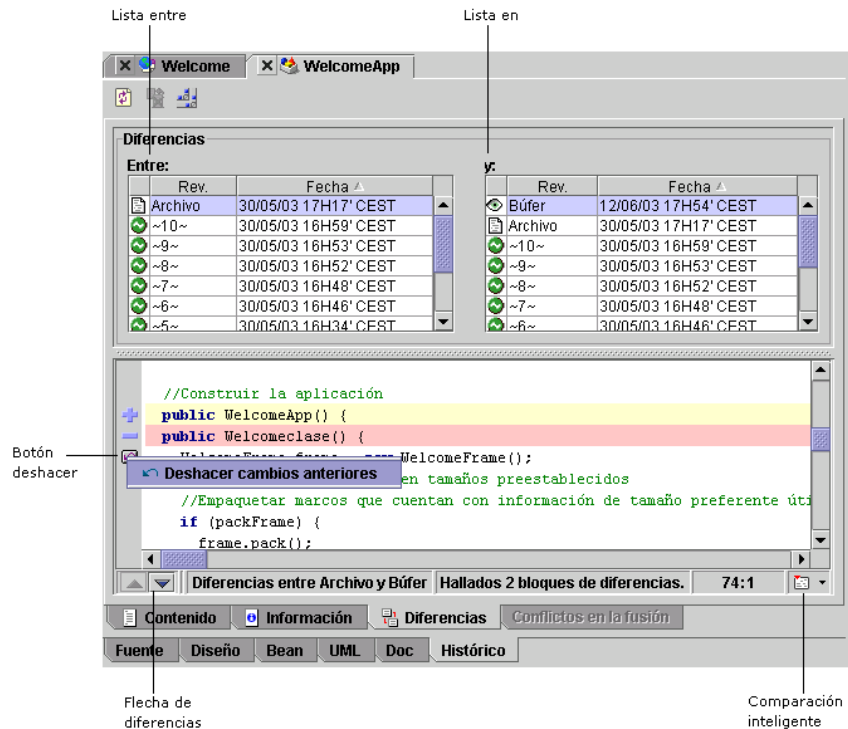
La columna Etiqueta recoge las etiquetas que se han asignado a revisiones específicas del archivo. JBuilder asigna algunas etiquetas automáticamente. Por ejemplo, cuando se guarda un archivo en JBuilder, se crea una nueva versión de copia de seguridad del archivo con la etiqueta "Copia de seguridad local", y a la revisión correspondiente al archivo guardado más reciente se le asigna la etiqueta "Archivo local". Cuando se aplican etiquetas locales o simplemente etiquetas en un sistema de control de versiones (SCV), también aparecen en la columna Etiqueta. Para obtener más información sobre las etiquetas locales, consulte ["Utilización de etiquetas locales en la gestión de revisiones de archivos locales" en la página 12-4](#). Si desea obtener información acerca de las etiquetas SCV, consulte *Team Development using JBuilder* y la documentación de SCV.

Los botones Actualizar información de revisiones, Restaurar la revisión anterior y Sincronizar desplazamiento están habilitados en esta ficha.

Ficha Diferencias

La ficha Diferencias muestra las diferencias entre dos versiones seleccionadas del archivo activo. Las listas de revisiones de la parte superior permiten ordenar las versiones del archivo por tipo de versión, número de revisión y fecha de cambio. Los botones Actualizar información de revisiones y Sincronizar desplazamiento están habilitados en esta ficha.

Existen dos listas de revisiones en la ficha Diferencias. Una se denomina Entre y la otra, En. Seleccione una versión en la lista Entre y otra versión diferente en la lista En. Las diferencias se muestran en el visualizador de código fuente, bajo las listas:



Sugerencia En cada bloque de diferencias, el texto eliminado (con el signo menos) está en color rojo y el texto añadido (con el signo más) está en color verde. Si el búfer está seleccionado como la versión En, pulse el botón Deshacer si desea deshacer la adición.

Utilice las flechas de diferencias para desplazarse entre los bloques de diferencias. Si existen muchos bloques de diferencias debidos a cambios en el formato del código, utilice Comparación inteligente para omitir esos cambios (tales como ubicación y número de líneas en blanco, sangrado diferente o uso de espacios).

- Sugerencia** Cuando el cursor se encuentre en el panel de la vista de código fuente de la ficha Diferencias, puede pulsar *Mayús+F10* para abrir un menú contextual que contiene opciones para activar y desactivar Comparación inteligente, así como desplazarse a bloques de diferencias anteriores y posteriores.
- Nota** Los usuarios del control de versiones pueden ver todos los cambios realizados en el repositorio desde la extracción original. Seleccione la versión original del repositorio en el área Entre y la versión del archivo con el número más alto de revisiones del área En. JBuilder actualiza la versión más reciente con los cambios del repositorio.

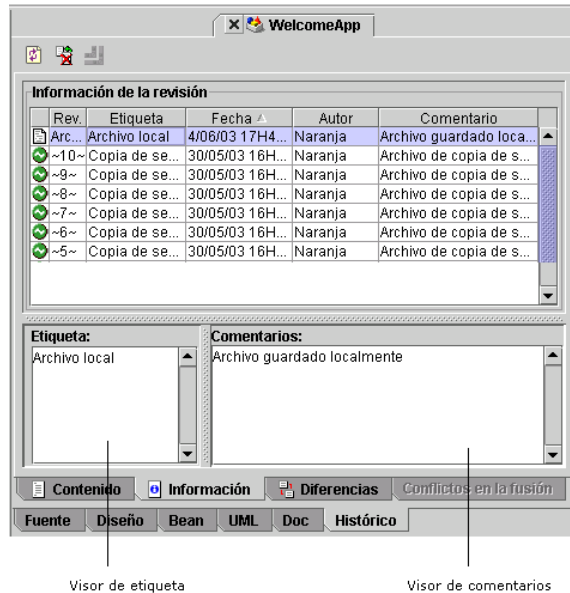
Ficha Información

La ficha Información de la vista Histórico muestra el texto completo de las etiquetas y comentarios de cualquier versión seleccionada del archivo activo. La lista de revisiones, en la parte superior, permite ordenar las versiones del archivo por tipo de versión, número de revisión, etiqueta, fecha de cambio, autor y comentario. Los botones Actualizar información de revisiones y Restaurar la revisión anterior están habilitados en esta ficha.

Al igual que en la ficha Contenido, la columna Etiqueta recoge las etiquetas que se han asignado a revisiones específicas del archivo. JBuilder asigna algunas etiquetas automáticamente. Por ejemplo, cuando se guarda un archivo en JBuilder, se crea una nueva versión de copia de seguridad del archivo con la etiqueta "Copia de seguridad local", y a la revisión correspondiente al archivo guardado más reciente se le asigna la etiqueta "Archivo local". Cuando se aplican etiquetas locales o simplemente etiquetas en un sistema de control de versiones (SCV), también aparecen en la columna Etiqueta. Para obtener más información sobre las etiquetas locales, consulte ["Utilización de etiquetas locales en la gestión de revisiones de archivos locales" en la página 12-4](#). Si desea obtener información acerca de las etiquetas SCV, consulte *Team Development using JBuilder* y la documentación de SCV.

Para utilizar la ficha Información, seleccione una versión en la lista de revisiones. Las etiquetas relacionadas (en caso de haberlas) y cualquier

comentario introducido para describir la revisión se muestran en la parte inferior de la ficha.



La ficha Información es una herramienta de investigación. Permite desplazarse por los comentarios de todas las revisiones de un archivo para obtener una vista general de todos los cambios llevados a cabo.

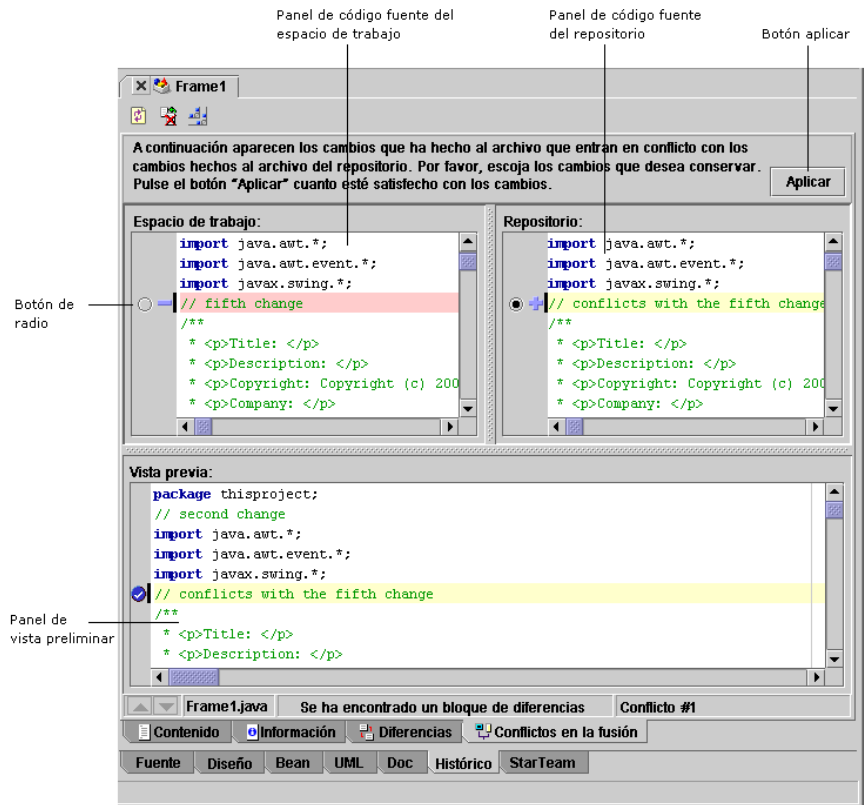
Si desea información adicional sobre estas funciones, pulse la tecla *F1* desde cualquiera de las fichas del histórico. De esta forma aparece la ayuda contextual del panel de contenido. Seleccione Histórico en la tabla de ayuda contextual.

Ficha Conflictos en la fusión

La ficha Conflictos en la fusión permite resolver automáticamente los conflictos de fusión que se pueden producir al trabajar con JBuilder en la integración de StarTeam, Concurrent Versioning System (CVS) o Visual SourceSafe (VSS). La ficha Conflictos en la fusión sólo aparece activada cuando se han detectado conflictos.

Cuando se produce un conflicto de fusión, la ficha Conflictos en la fusión muestra el código fuente del espacio de trabajo junto al código fuente del repositorio, con los bloques de código o texto en conflicto resaltados. Los botones circulares contiguos a los bloques resaltados de código conflictivo permiten seleccionar fácilmente el bloque de código que se desea mantener. El panel de vista preliminar, en la parte inferior de la ficha Conflictos en la

fusión, muestra el aspecto que tendrá el archivo del espacio de trabajo cuando se le hayan aplicado los cambios.



Las líneas de un conflicto que se han seleccionado para mantenerse se marcan en verde y, además, presentan el signo más (+) en el margen. Los cambios que se descartan se marcan en rojo y tienen el signo menos (-) en el margen. Si existen muchos conflictos, se pueden pulsar las flechas de desplazamiento que se encuentran en la esquina izquierda inferior de la ficha para desplazarse hacia delante y atrás entre los pares de diferencias. El panel de vista preliminar, en la parte inferior de la ficha Conflictos en la fusión, muestra el aspecto que tendrá el archivo original cuando se le hayan aplicado los cambios. Por defecto, los cambios en el espacio de trabajo se seleccionan cuando se desea mantenerlos.

Después de haber seleccionado los cambios que desee conservar, pulse Aplicar para almacenarlos en el búfer del editor. De esta forma se actualiza el archivo en el que se está trabajando. La aplicación los cambios no significa que se guarde el archivo o se incorporen (confirman) los cambios en el repositorio. Hasta que se confirman los cambios, se puede utilizar el comando

Deshacer (Edición|Deshacer o *Ctrl+Z*) para restablecer los conflictos. El comando Deshacer no se encuentra disponible desde la vista histórico, por lo que es necesario cambiar a otra vista.

La ficha Conflictos en la fusión sólo aparece activada cuando se producen conflictos de fusión al utilizar la integración de JBuilder de sistemas como Concurrent Versioning System (CVS), Visual SourceSafe (VSS) o StarTeam. Para obtener más información sobre la forma de resolver conflictos de fusión con CVS, VSS o StarTeam, consulte "Trabajo en un proyecto en CVS", "Trabajo en un proyecto en VSS" o "Gestión de archivos en StarTeam" en *Desarrollo en equipo con JBuilder*. Las integraciones de Visual SourceSafe y StarTeam son funciones de las versiones Developer y Enterprise de JBuilder.



Perfeccionamiento de código

**Es una función de
JBuilder Developer y
Enterprise.**

El código evoluciona según los cambios de la tecnología y las demandas del mercado. Es posible que resulte necesario modificar el código creado anteriormente con el fin de hacer posibles las ampliaciones, mejorar el rendimiento, adaptarlo a nuevas necesidades o, simplemente, depurar la base. El término *perfeccionamiento (refactoring)* designa el proceso de realización de alteraciones en el código fuente sin que cambie su comportamiento durante la ejecución a los ojos del usuario.

El perfeccionamiento se puede realizar a pequeña y gran escala, pero incluso los cambios mínimos pueden provocar errores. Con el fin de que el perfeccionamiento resulte eficaz debe ser correcto y completo. Un solo cambio puede desencadenar permutaciones en toda la base de código. El perfeccionamiento gestiona todo el conjunto de permutaciones de forma razonable y coherente: ningún comportamiento debe cambiar si no es con vistas a la mejora del rendimiento o la capacidad de mantenimiento, y no se debe introducir ningún error.

JBuilder ofrece varios tipos de perfeccionamiento, incluidos el perfeccionamiento por cambio de nombre, por desplazamiento, extracción de métodos, optimización de importaciones, y perfeccionamiento orientado a objetos. Se accede a las opciones de perfeccionamiento desde el menú contextual del editor, desde el menú Editor, desde el menú contextual del panel de estructura y desde un menú contextual de los diagramas UML. (UML es una función de JBuilder Enterprise).

Detección de diferencias antes del perfeccionamiento

Antes de realizar el perfeccionamiento se puede consultar, por categoría, todas las ubicaciones del proyecto actual en las que se hace referencia al símbolo de código seleccionado. (Un símbolo de código es un fragmento de código que representa algo, como un nombre de paquete, de clase o de método). También es posible desplazarse a la definición del símbolo.

Importante Para encontrar estas referencias, es necesario compilar el proyecto con las referencias de bibliotecas del proyecto activadas. Para activar esta opción, seleccione Propiedades de proyecto|General. La opción Incluir referencias de archivos de clase de las bibliotecas del proyecto debe encontrarse activada. Esta opción carga todas las relaciones de biblioteca y permite que JBuilder detecte las referencias. no es necesario seleccionar esta opción, ya que podría reducir la velocidad de compilación y perfeccionamiento. No obstante, si esta opción está desactivada, JBuilder no puede detectar todas las diferencias.

Además, el proyecto debe estar actualizado, esto es, la marca horaria de los archivos de clase debe coincidir con la de los archivos fuente. Para garantizar que el proyecto esté actualizado, compílelo por medio del comando Proyecto|Ejecutar Make del proyecto.

Para aprender más acerca del código antes del perfeccionamiento, se utilizan comandos del menú contextual del editor o del menú Buscar. Para que los comandos de menú permitan buscar los distintos tipos de referencias:

- Buscar definición - Determina dónde se define el símbolo seleccionado.
- Buscar método redefinido - Busca el método redefinido por el método seleccionado.
- Buscar referencias locales - Identifica las referencias de un método, campo o clase local del archivo activo.
- Buscar referencias - Busca todos los archivos fuente mediante un símbolo seleccionado.

Búsqueda de definiciones

El comando Buscar definición del menú Buscar y del menú contextual del editor permite localizar la definición de los símbolos. Para buscar la definición de un símbolo:

- 1 Compile el proyecto.
- 2 Seleccione el símbolo en el editor.
- 3 Seleccione Buscar|Buscar definición o, bien, haga clic con el botón derecho en el símbolo y seleccione Buscar definición. (También se puede utilizar el método abreviado de teclado *Ctrl+Intro*). El archivo fuente donde se define el símbolo se abre en el editor.

- Si el símbolo es una instancia de una clase, el cursor se desplaza hasta la definición de la instancia.
- Si el símbolo es un método, la clase en la que se define se abre en el editor, con el cursor en el principio de la firma del método.
- Si el símbolo es una variable que se define en la clase abierta actualmente en el editor, el cursor se desplaza a la definición. Si la variable es *pública* y se define en otra clase, ésta se abre en el editor, con el cursor en la definición.

Importante Para que sea posible buscar definiciones, el proyecto debe estar compilado. La clase que incluye la definición debe encontrarse en la vía de acceso `import` del mismo paquete que el símbolo.

Búsqueda de métodos redefinidos

El comando Buscar método redefinido del menú contextual del editor y el panel de estructura permite buscar el método redefinido mediante el seleccionado. Para buscar el método redefinido:

- 1 Compile el proyecto.
- 2 Abra el archivo fuente que contiene el método declarado.
- 3 Seleccione el símbolo en el editor o en el panel de estructura. Haga clic con el botón derecho y seleccione Encontrar método redefinido.

JBuilder abre la superclase en la que se declara el método redefinido y coloca el cursor en la declaración del método. Puede desplazarse por la cadena de superclases para buscar todos los métodos de superclase redefinidos.

Nota Los métodos redefinidos aparecen en cursiva.

Búsqueda de referencias locales



El comando del menú contextual del editor Buscar referencias locales se utiliza para buscar referencias para métodos, campos o clases locales del archivo activo. Para buscar referencias locales:

- 1 Abra el archivo fuente que contiene el método, el campo o la clase del que desee buscar las referencias locales.
- 2 Seleccione Buscar|Buscar referencias locales, o bien, seleccione el símbolo en el editor, haga clic con el botón derecho y seleccione Buscar referencias locales.

Las referencias se muestran en la pestaña Buscar del panel de mensajes siguiendo el orden en el que se han ido encontrando. JBuilder muestra los resultados en el panel de mensajes. Para facilitar la presentación, las referencias se amplían por defecto.

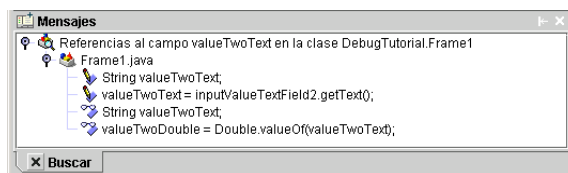
En la siguiente tabla se detallan las categorías de referencias que aparecen en la pestaña Buscar para las referencias locales.

Tabla 13.1 Detalles de Buscar referencias

Icono	Descripción
	Escrituras: Ubicaciones donde se escribe el campo.
	Lecturas: Ubicaciones donde se lee el campo.

En la siguiente imagen se muestra la pestaña Buscar con referencias locales.

Figura 13.1 Presentación de referencias locales













Búsqueda de referencias

Antes de realizar el perfeccionamiento, puede ser conveniente buscar todos los archivos fuente que utilizan un símbolo seleccionado. Para localizar todas las referencias a un símbolo:

- 1 Compile el proyecto.
- 2 Seleccione el símbolo en el editor, en el panel de estructura o en un diagrama de clase UML. (UML es una función de JBuilder Enterprise).
- 3 Seleccione **Buscar** | **Buscar referencias**, o bien, haga clic con el botón derecho en el símbolo y seleccione **Buscar referencias**. (También se puede utilizar el método abreviado de teclado *Ctrl+Mayús+Intro*). Las referencias se muestran en la pestaña **Buscar** del panel de mensajes siguiendo el orden en el que se han ido encontrando. Las referencias a clases y métodos se ordenan por categoría. Las referencias a campos y variables locales se ordenan por nombre de archivo. No es posible buscar las referencias de paquetes ni propiedades.

En la tabla siguiente se detallan, con arreglo a los símbolos de código, las categorías de referencia que aparecen en la pestaña Buscar.

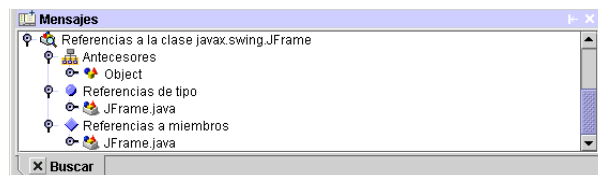
Tabla 13.2 Detalles de Buscar referencias

Símbolo de código	Categoría referencia
Clase, clase interna o interfaz	 Jerarquía de herencia: Clases de las que esta clase hereda o desciende.
	 Referencias de tipo: clases que declaran o crean una instancia del tipo de objeto de la clase.
	 Referencias a tipos descendientes: clases que descienden o utilizan descendientes del tipo de objeto de la clase.
	 Referencias a miembros: miembros de esta clase.
	 Referencias a miembros descendientes: miembros de clases que descienden de esta clase.
Método o constructor	 Declaraciones: ubicaciones donde se declara este método.
	 Usos directos: lugares en clases instanciadas de forma directa que llaman a este método.
	 Usos indirectos: lugares en clases descendientes y antecesoras que, indirectamente, llaman a este método mediante un antecesor o descendiente.
Campo y variable local	 Escrituras: lugares donde se escribe el campo o la variable local.
	 Lecturas: lugares donde se lee el campo o la variable local.

Referencias a clases

Si han aparecido referencias a una clase, haga doble clic en una categoría de referencias, en la pestaña Buscar, para ampliarla. Se enumeran los archivos fuente que contienen referencias a la clase. Pulse uno de estos archivos y pulse la referencia para abrirla directamente en el editor.

Figura 13.2 Presentación de las referencias a clases



Referencias a métodos

Si han aparecido referencias a un método, haga doble clic en una categoría de referencias para ampliarla. Se enumeran los archivos fuente que contienen referencias al método. Elija uno de estos archivos y seleccione la declaración o uso para ir directamente a la referencia en el editor.

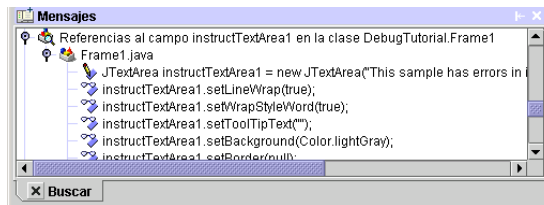
Figura 13.3 Presentación de las referencias a métodos



Referencias a campos y variables locales

Si han aparecido referencias a un campo o una variable local, haga doble clic en un nombre de archivo para mostrar las escrituras y lecturas de este símbolo. Pulse la referencia de lectura o escritura para colocar el cursor en ella, en el editor.

Figura 13.4 Presentación de referencias a campos y variables locales



Perfeccionamiento en JBuilder

Para realizar un proceso de perfeccionamiento en JBuilder, empiece por seleccionar el símbolo o el bloque de código que desea perfeccionar. A continuación, haga clic con el botón derecho del ratón o abra el menú Edición para elegir el tipo de perfeccionamiento. En la mayoría de los procesos de perfeccionamiento, JBuilder proporciona un cuadro de diálogo en el que se pueden introducir detalles de perfeccionamiento y seleccionar si se desea la vista previa del perfeccionamiento. En algunos perfeccionamientos, JBuilder completa automáticamente el proceso. Como regla general, el perfeccionamiento de archivos individuales no muestra salida a menos que haya errores o advertencias.

Se puede tener acceso a las herramientas de perfeccionamiento de JBuilder desde los menús contextuales del editor, del panel de estructura y del diagrama de paquete o clase UML. (UML es una función de JBuilder Enterprise).

Si JBuilder no puede completar un perfeccionamiento, el IDE ofrece mensajes de error y advertencia para explicar el por qué. Las advertencias no detienen el perfeccionamiento. Las advertencias no detienen el perfeccionamiento. Sin embargo, los errores sí. Por ejemplo, un proceso de perfeccionamiento se puede impedir si un archivo es de sólo lectura (aún no se ha extraído) o si ya existe el nombre del símbolo.

Advertencia No se puede perfeccionar en varios proyectos a la vez.

Las herramientas de perfeccionamiento de JBuilder examinan a fondo el proyecto y toman en consideración:

- Limitaciones: JBuilder comprueba la existencia de condiciones que puedan provocar problemas en el perfeccionamiento. Por ejemplo, JBuilder determina si la información necesaria sobre las dependencias no está disponible o es antigua, si un archivo es de sólo lectura y si un archivo de clase no existe.
- Validación JBuilder determina si el nombre nuevo es válido. Por ejemplo, es posible que ya exista o que su sintaxis sea inadecuada.
- Árbol de fuentes: Cuando se perfecciona por desplazamiento una clase y cuando se perfecciona por cambio de nombre un paquete, JBuilder desplaza los directorios o los archivos dentro del árbol de fuentes. También actualiza las sentencias de importación necesarias para las dependencias.

Importante Genere siempre su proyecto antes de perfeccionar. Para seleccionar esta opción para el proyecto, escoja Propiedades de proyecto|Generar. Seleccione la opción Generar siempre antes de perfeccionar. Esta opción está desactivada por defecto. Si selecciona esta opción, el perfeccionamiento es más lento, pero se detectan todos los casos. Si no la selecciona, el perfeccionamiento es más rápido, pero puede haber elementos específicos que no se detecten en el perfeccionamiento.

Perfeccionamiento de EJB

El desarrollo de EJB es una función de JBuilder Enterprise

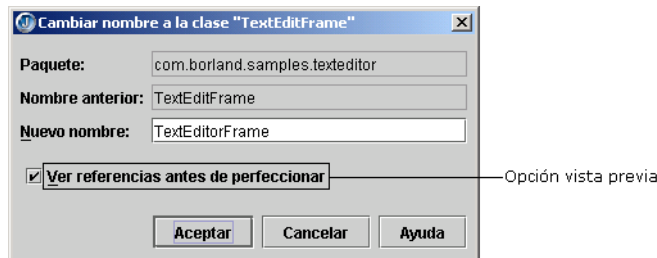
El perfeccionamiento de EJB debe realizarse mediante el Diseñador de EJB. Si se intenta perfeccionar un archivo EJB desde el editor o desde un diagrama UML, aparece la siguiente advertencia:

ADVERTENCIA: está perfeccionado un archivo EJB. Esto puede hacer que tenga que cambiar el código fuente y el descriptor de distribución a mano. Le recomendamos que utilice el diseñador de EJB para la mayoría de los casos de perfeccionamiento.

Si se elige continuar, es necesario actualizar todos los archivos fuente relevantes para realizar el perfeccionamiento.

Presentación previa de los cambios antes del perfeccionamiento

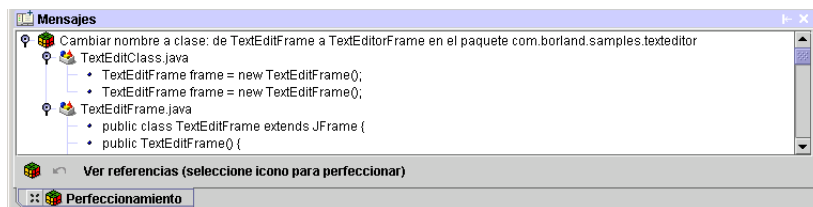
JBuilder permite ver los cambios que se producirán antes de confirmar la mayoría de los tipos de perfeccionamiento. Puede resultar conveniente observar las modificaciones cuando se empiezan a utilizar las herramientas de perfeccionamiento, con el fin de comprender qué cambios se van a realizar. A continuación se muestra la opción de vista previa, Ver referencias antes de perfeccionar.

Figura 13.5 La opción Vista previa

Cuando se elige la opción de vista previa y se pulsa Aceptar en el cuadro de diálogo, los cambios que se van a realizar se muestran en la pestaña Perfeccionamiento del panel de mensajes. Los cambios en potencia, esto es, las líneas de código que cambiarán si se realiza el perfeccionamiento, se muestran según el nombre de archivo, por orden de detección. Amplíe el nodo de archivo y pulse la referencia para dirigirse a ella en el archivo fuente.

Nota Algunas tareas de perfeccionamiento no cuentan con la opción de vista previa. Como regla general, el perfeccionamiento de archivos individuales no muestra salida a menos que haya errores o advertencias.

Antes de perfeccionar, la pestaña Perfeccionamiento en modo vista previa se asemejará a la figura siguiente:

Figura 13.6 La pestaña Perfeccionamiento antes de la acción

La pestaña Perfeccionamiento contiene un botón Perfeccionar y un icono en trancada que indica que el perfeccionamiento no está completo.

Nota Si modifica alguno de los archivos que se muestran en la pestaña perfeccionar antes de completar el perfeccionamiento, JBuilder no permitirá el perfeccionamiento, ya que los archivos saldrían desfasados. La barra de estado de la pestaña Perfeccionamiento muestra el siguiente mensaje: "Cambios en archivos; no se puede perfeccionar".

En la tabla siguiente se detalla el tipo de información de vista previa que se muestra.

Tabla 13.3 Detalles de perfeccionamiento

Símbolo de código	Tipo de perfeccionamiento	Información presentada
Paquete	Cambiar nombre	Archivos fuente que contienen una referencia a clase que va a cambiar.
Clase, clase interna o interfaz	Cambiar nombre	Posición de las líneas del archivo fuente actual donde se declara la clase; incluye constructores. También enumera los lugares del código fuente donde se utiliza la clase.
Interfaz	Extracción de clase	Nombre de la interfaz.
Superclase	Extracción de clase	Nombre de la clase.
Clase	Mover	Lugares del código fuente donde se declara o se importa el paquete actual. Indica si se añade o se borra un paquete de la lista de importaciones. (Se añade la sentencia <code>import</code> a todas las dependencias que tenga la clase en el paquete desde el que se mueve).
Método	Cambiar nombre	Lugares del código fuente donde se declara y se utiliza el método. Indica si se crea un método de reenvío.
Método	Cambiar parámetros	Lugares del código fuente donde se declara y llama al método.
Método	Subir, Bajar	Lugares del código fuente donde se declara y llama al método.
Campo y variable local	Cambiar nombre	Lugares del código fuente donde se declara el símbolo y se realiza la llamada.
Campo	Bajar	Lugares del código fuente donde se declara el símbolo y se realiza la llamada.
Propiedad	Cambiar nombre	Lugares del código fuente donde se declara la propiedad y donde se declaran y llaman los métodos de obtención y definición que la acompañan.

Finalización del perfeccionamiento

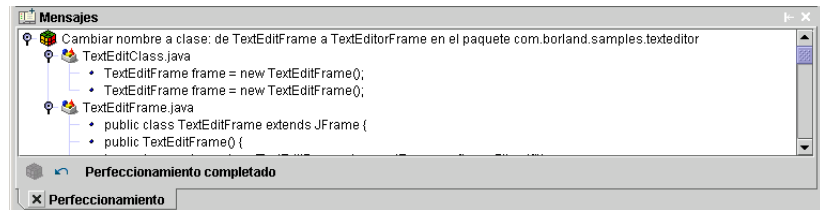


Para completar el perfeccionamiento, pulse el botón Perfeccionar de la pestaña Perfeccionamiento. La barra de estado de la pestaña Perfeccionamiento muestra un mensaje que informa sobre el progreso.

Cuando termina el perfeccionamiento, en la barra de estado aparece el mensaje "Perfeccionamiento completado". El botón Perfeccionar se atenúa y

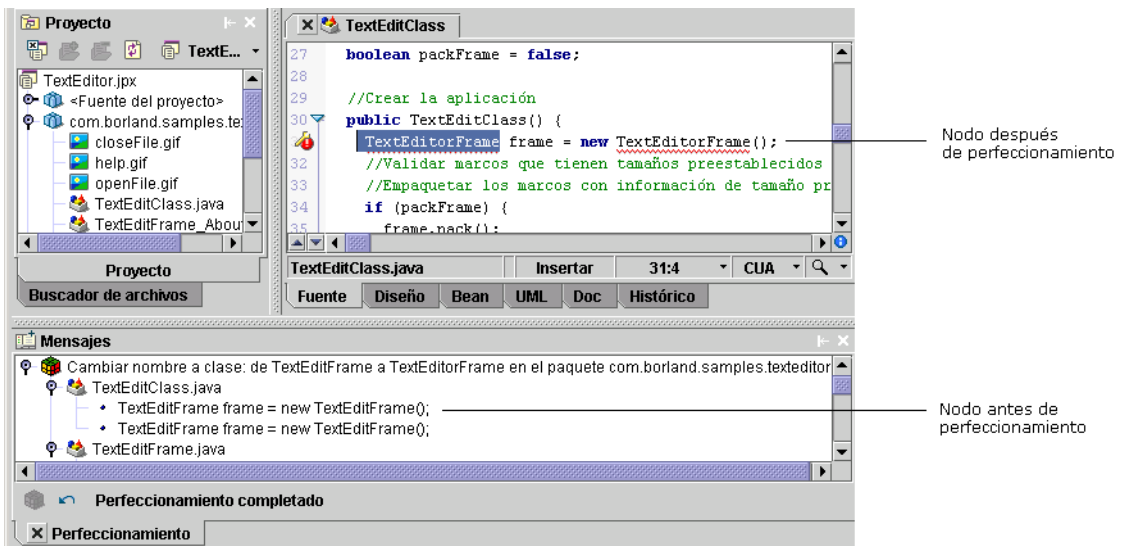
el símbolo de x truncada se cierra en una X, tal y como se muestra en la siguiente imagen.

Figura 13.7 La pestaña Perfeccionamiento después de la acción



Después del perfeccionamiento, el contenido de la pestaña Perfeccionamiento no cambia. Aún se muestran las líneas originales del código fuente, de forma que se pueden comparar los cambios realizados. Pulse una línea del código fuente original para dirigirse al cambio.

Figura 13.8 El archivo fuente y la pestaña Perfeccionamiento después de la acción



Cómo deshacer un perfeccionamiento



Una vez finalizado el perfeccionamiento, la operación se puede deshacer pulsando el botón **Deshacer** de la barra de herramientas **Perfeccionamiento**. Deshaga la operación inmediatamente, antes de realizar más cambios en los archivos. Todos los cambios se deshacen. A continuación se puede rehacer el perfeccionamiento pulsando el botón **Perfeccionar** de la pestaña **Perfeccionamiento**. La opción **Deshacer** está activa mientras la pestaña **Perfeccionamiento** permanece abierta.

Importante Si utiliza un perfeccionamiento que no muestra ninguna salida en la pestaña Perfeccionamiento, puede deshacer los cambios con Modificar|Deshacer.

Almacenamiento de perfeccionamientos

Cuando termine el perfeccionamiento, guarde los archivos del proyecto mediante el comando Archivo|Guardar todo. Si utiliza un sistema de control de versiones, confirme o incorpore los cambios inmediatamente.

Si se intenta cerrar el proyecto sin guardar, JBuilder abre el cuadro de diálogo Guardar archivos modificados, donde se pueden seleccionar los archivos que se desea guardar. Si los archivos no se guardan, el código fuente recupera el estado anterior al perfeccionamiento.

Importante El perfeccionamiento puede provocar cambios en archivos que no se hayan abierto en el editor durante los preparativos. JBuilder guarda automáticamente los cambios en esos archivos. JBuilder realiza estos cambios y guarda los archivos para que el código fuente no se encuentre en un estado incoherente.

Ejecución de perfeccionamiento

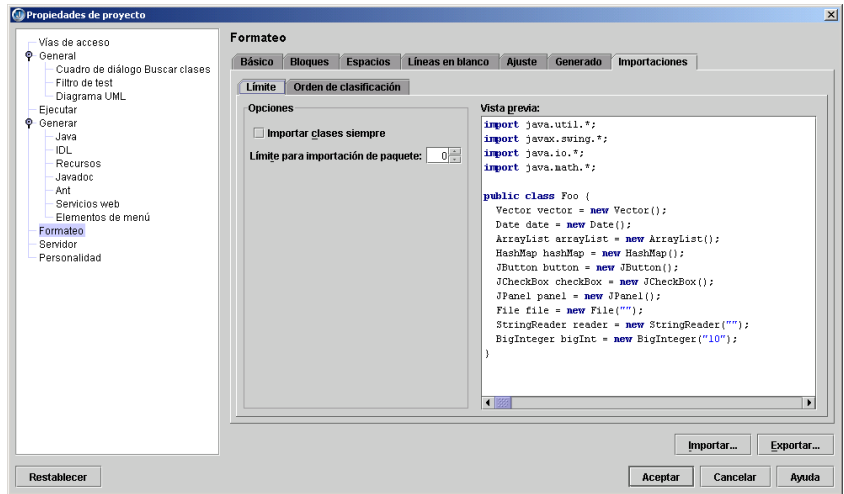
En esta sección se trata el tipo de perfeccionamiento disponible en JBuilder y se presentan instrucciones paso a paso para cada tipo.

Optimizar importaciones

El comando Optimizar importaciones reescribe y reorganiza las sentencias `import` según la configuración de las propiedades del proyecto. También elimina las sentencias `import` que ya no se utilizan. El orden de las importaciones se puede personalizar en la ficha Importaciones de Propiedades de proyecto|Formato.

Para establecer límites y opciones de clasificación para las importaciones:

- 1 Seleccione Propiedades de proyecto|Formateo|Importaciones|Límite. La ficha Límite tiene un aspecto parecido al siguiente:

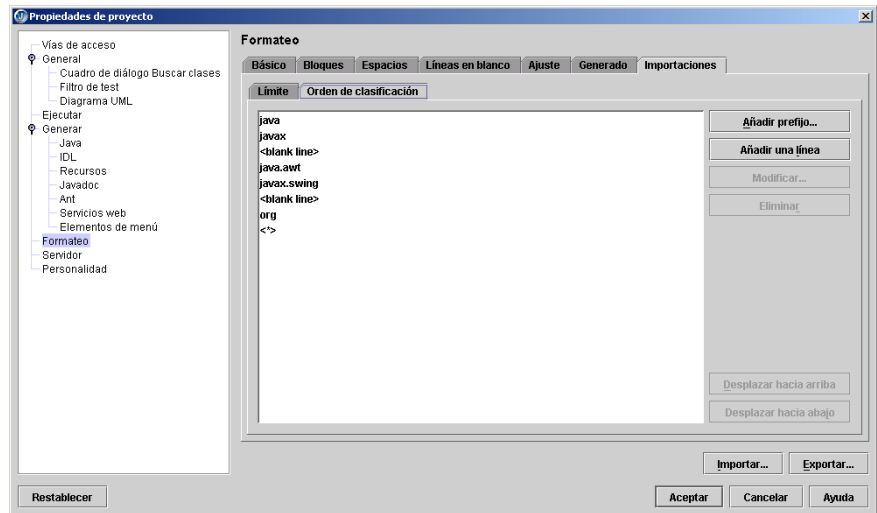


- 2 Utilice la opción Importar clases siempre para determinar si se añaden al código las sentencias de `importación` de paquetes. Elija esta opción si no desea añadir al código sentencias de importación de paquetes. En su lugar, las clases individuales se importan directamente. Cuando se utiliza esta opción se pasa por alto la configuración de Límite para importación de paquete.
- 3 Utilice el Límite para importación de paquete para establecer el número de clases de un paquete antes de reescribir importaciones de clases en una sentencia de paquetes importados.

Hasta este límite, las clases se importan mediante sentencias de `importación` individuales. Cuando se sobrepasa este límite, se importa el paquete completo. Por ejemplo, si se elige 3 en este campo e importa cuatro clases o más de un paquete, se importa el paquete entero.

El cuadro Vista previa muestra los resultados de las diferentes configuraciones de límites de importación.

- 4 Utilice la ficha Orden de clasificación para determinar el modo de clasificación de las importaciones. La ficha Orden de clasificación tiene un aspecto similar a este:



- 5 Para añadir una importación que empiece con un prefijo concreto, pulse el botón Añadir prefijo. Escriba el prefijo en el cuadro de diálogo Añadir prefijo.
- 6 Para insertar una línea de ruptura adicional entre sentencias de importación o grupos de sentencias de importación, seleccione el paquete bajo el que desea insertarla y haga clic en el botón Añadir una línea.
- 7 Seleccione una sentencia de importación de paquetes y pulse el botón Modificar para cambiarla.
- 8 Seleccione una sentencia de importación y pulse Eliminar para eliminarla de la lista.
- 9 Escoja una línea de importación o una línea en blanco y haga clic en Desplazar hacia arriba o en Desplazar hacia abajo para cambiar su orden en la lista.
- 10 Pulse el botón Importar para importar un formato predefinido. Aparece el cuadro de diálogo Importar configuración de formato de código. Elija el formato que desee importar y pulse Aceptar.
- 11 Pulse el botón Exportar para exportar y guardar el formato. Aparece el cuadro de diálogo Exportar la configuración de formato de código. Escriba el nombre del archivo en el que desee guardar la configuración y pulse Aceptar. El tipo de archivo debe ser `.codestyle`. Los archivos de configuración de formato se guardan en el directorio `<.jbuilder>`.

Sugerencia Si desea personalizar el orden de las sentencias de importación para *todos* los nuevos proyectos, seleccione Proyecto|Propiedades por defecto para proyectos y realice las modificaciones en ese cuadro de diálogo.

Optimización de las importaciones

Para optimizar las importaciones:

- 1 Elija Proyecto|Ejecutar Make del proyecto con el fin de realizar la compilación.
- 2 Seleccione Edición|Optimizar importaciones o haga clic en el editor con el botón derecho del ratón y elija Optimizar importaciones. También puede utilizar el método abreviado *Ctrl+I*. Asimismo, puede seleccionar cualquier símbolo del panel de estructura; haga clic con el botón derecho del ratón y seleccione Optimizar importaciones.

Sugerencia Para deshacer la optimización de importaciones elija Edición|Deshacer.

Para optimizar las importaciones desde el panel de proyecto:

- 1 Haga clic con el botón derecho del ratón sobre el paquete del panel de proyecto.
- 2 Seleccione Formatear paquete.
- 3 Active la opción Optimizar importaciones del cuadro de diálogo Formatear código y, a continuación, pulse Aceptar.

Nota Se conservan los comentarios de la sección de importación del código.

Perfeccionamientos por cambio de nombre

Los perfeccionamientos (refactoring) por cambio de nombre consisten en aplicar un nombre nuevo a un paquete, una clase, un método, un campo, una variable local o una propiedad, asegurándose de que todas las referencias a este nombre se gestionan correctamente. Cuando se aplica este tipo de perfeccionamiento a un constructor, el nombre de la clase cambia.

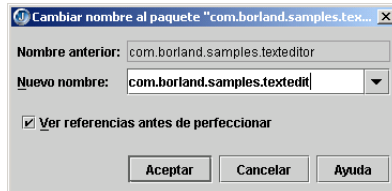
El perfeccionamiento Cambiar nombre no es una simple tarea de buscar y reemplazar, ya que se deben tener en cuenta las referencias y se deben gestionar correctamente. También es necesario reconocer los patrones. Por ejemplo, si este perfeccionamiento se aplica a un nombre de clase, el nuevo nombre se debe reflejar en la declaración de clase, en todas sus instancias y en las referencia a dicha clase.

Cambio de nombre de paquetes

Se pueden perfeccionar paquetes por cambio de nombre desde el editor, el panel de estructura o el menú contextual del diagrama UML. (UML es una función de JBuilder Enterprise). Cuando se aplica el perfeccionamiento Cambiar nombre a un paquete, éste y todo el subconjunto de paquetes adquieren el nuevo nombre del paquete raíz. Se actualizan el paquete y las sentencias de importación de los archivos de clase. El paquete, los subpaquetes y los archivos fuente de clase se desplazan al nuevo directorio fuente, y el antiguo se borra. Para fusionar dos paquetes, asigne al paquete seleccionado el nombre de un paquete existente.

Para perfeccionar por cambio de nombre un paquete:

- 1 Haga clic en el nombre del paquete en un diagrama UML de clase o paquete, en el panel de estructura o en el editor.
- 2 Desde el panel de estructura o un diagrama UML, haga clic con el botón derecho y seleccione Cambiar nombre al paquete. Desde el editor, haga clic con el botón derecho y seleccione Perfeccionamiento|Cambiar nombre al paquete. Se abre el cuadro de diálogo de Cambiar nombre al paquete.



- 3 Escriba el nuevo nombre para el paquete en el campo Nuevo nombre.
- 4 Haga clic en la opción Ver referencias antes de perfeccionar para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.



Se impide el perfeccionamiento del nombre del paquete si ya existe o no es válido.

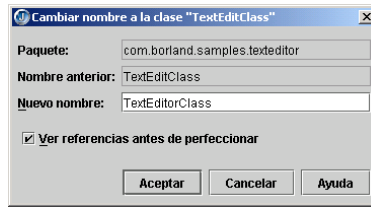
Cambio de nombre de clases, clases internas o interfaces

Puede perfeccionar por cambio de nombre una clase, una clase interna o una interfaz desde el menú Perfeccionar o desde el editor, el panel de estructura o el menú contextual del diagrama UML. (UML es una función de JBuilder Enterprise). El perfeccionamiento por cambio de nombre de una clase pública externa cambia el nombre de todas las declaraciones y usos de la clase y del archivo fuente. Si se selecciona un constructor, el perfeccionamiento por cambio de nombre modifica el nombre de la clase.

Para perfeccionar por cambio de nombre una clase, una clase interna o una interfaz:

- 1 Abra el archivo de clase al que desee cambiar el nombre en el editor o en un diagrama UML.
- 2 Seleccione el nombre de la clase, la clase interna o la interfaz que desee cambiar y escoja Perfeccionar|Cambiar nombre a la clase. (También se puede hacer clic con el botón derecho del ratón y escoger Perfeccionamiento|Cambiar nombre a la clase. Desde un diagrama UML o desde el panel de estructura, haga clic con el botón derecho y seleccione

Cambiar nombre a la clase). Aparece el cuadro de diálogo Cambiar nombre a la clase.



- 3 Escriba un nuevo nombre para la clase en el campo correspondiente.
- 4 Haga clic en la opción Ver referencias antes de perfeccionar para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.



Se impide el perfeccionamiento si el identificador de la clase no es válido. Si el nuevo nombre de archivo fuente ya existe en el paquete actual, no se permite el perfeccionamiento. Si la clase no es pública externa y hay otra clase pública no externa con el nuevo nombre, el cambio no se realiza.

Cambio de nombre de métodos

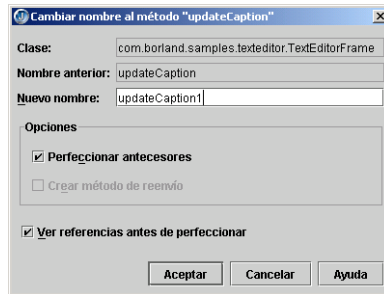
Se pueden perfeccionar por cambio de nombre métodos desde el menú Perfeccionar o desde el editor, el panel de estructura o el menú contextual de un diagrama UML. (UML es una función de JBuilder Enterprise). Cuando se aplica el perfeccionamiento Cambiar nombre a un método, se cambian todas sus declaraciones y usos. Se puede cambiar el nombre del método en la clase seleccionada y las inferiores de la jerarquía o en toda la jerarquía. También se puede crear un método de reenvío, que pasa la llamada del método antiguo al nuevo. Esto permite que la API pública permanezca intacta.

Nota Cuando se cambia el nombre de un método no cambia el nombre de los métodos sobrecargados, es decir, los métodos que tienen el mismo nombre y una firma distinta.

Para perfeccionar por cambio de nombre un método:

- 1 Abra el archivo fuente que contenga el método al que desea cambiar el nombre en el editor o, bien, ábralo como un diagrama de clase UML.
- 2 Seleccione el método al que desee cambiar el nombre y escoja Perfeccionar | Cambiar nombre al método. (También puede hacer clic con el botón derecho y seleccionar Perfeccionamiento | Cambiar nombre al método. Desde un diagrama UML o desde el panel de estructura, haga clic con el botón

derecho y escoja Cambiar nombre al método). Se abre el cuadro de diálogo Cambiar nombre al método.



- 3 Escriba un nuevo nombre para el método en el campo correspondiente.
- 4 Utilice la opción Perfeccionar antecesores (activada por defecto) para cambiar el nombre de los métodos en las clases de las que la actual es heredera. Desactive Perfeccionar antecesores si sólo desea cambiar el nombre al método en esta clase y en sus descendientes. Si lo desea, active la opción Crear método de reenvío.
- 5 Haga clic en la opción Ver referencias antes de perfeccionar para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.



Si este nombre ya existe en el archivo donde se declara, no se permite el perfeccionamiento. Si el nombre existe en otros archivos de la línea directa de herencia se muestra una advertencia. Al utilizar la opción Perfeccionar antecesores también puede aparecer una advertencia si el método existe pero no se encuentra en la vía de acceso a fuentes que se puede modificar. Por ejemplo, si el método existe en una biblioteca no es posible perfeccionarlo, ya que las bibliotecas son de sólo lectura.

Cambio de nombre de variables locales

Sólo se puede perfeccionar por cambio de nombre una variable local desde el menú Perfeccionar o el menú contextual del editor. Cuando se aplica el perfeccionamiento por cambio de nombre a una variable local se modifican el nombre de su declaración y de sus usos. Tenga en cuenta que los parámetros de métodos se consideran variables locales.

Para perfeccionar por cambio de nombre una variable local:

- 1 Abra el archivo fuente en el editor.
- 2 Seleccione la variable local a la que desea cambiar el nombre y escoja Perfeccionar/Cambiar nombre a la variable. (También puede hacer clic con

el botón derecho y seleccionar **Perfeccionamiento** | **Cambiar nombre a la variable**). Se abre el cuadro de diálogo **Cambiar el nombre a la variable**.



- 3 Escriba el nuevo nombre para el paquete en el campo **Nuevo nombre**.
- 4 Haga clic en la opción **Ver referencias antes de perfeccionar** para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga clic en **Aceptar**. Pulse el botón **Perfeccionar** de la pestaña **Perfeccionamiento** para completarlo.



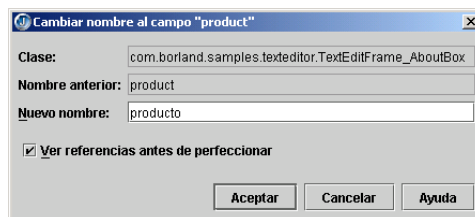
Si este nombre ya existe en la clase donde se declara la variable original, no se permite el perfeccionamiento. La palabra clave `this` se antepone al nuevo nombre si hay un conflicto con un nombre de otra clase.

Cambio de nombre de campos

Se puede perfeccionar por cambio de nombre un campo desde el menú **Perfeccionar** o desde el editor, el panel de estructura o menú contextual de un diagrama UML. (UML es una función de JBuilder Enterprise). Cuando se aplica el perfeccionamiento por cambio de nombre a un campo se modifica el nombre de sus declaraciones y usos.

Para perfeccionar por cambio de nombre un campo:

- 1 Abra el archivo fuente que contenga el campo al que desea cambiar el nombre en el editor o, bien, ábralo como un diagrama de clase UML.
- 2 Seleccione el campo al que desea cambiar el nombre y escoja **Perfeccionar** | **Cambiar nombre al campo**. (También puede hacer clic con el botón derecho y seleccionar **Perfeccionamiento** | **Cambiar nombre al campo**). Desde un diagrama UML o desde el panel de estructura, haga clic con el botón derecho y seleccione **Cambiar nombre al campo**). Se abre el cuadro de diálogo **Cambiar nombre al campo**.



- 3 Escriba un nuevo nombre para el campo en el campo correspondiente.
- 4 Haga clic en la opción **Ver referencias antes de perfeccionar** para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga



clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.

Si este nombre ya existe en la clase donde se declara el campo, no se permite el perfeccionamiento. Si hay conflictos de ámbito entre el nombre nuevo y el antiguo, se antepone al primero la palabra clave `this`. Si el nombre nuevo redefine un campo existente o es redefinido por él, en una clase o una subclase, se muestra una advertencia.

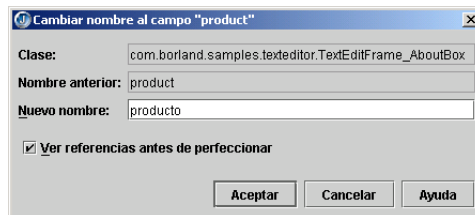
Cambio de nombre de propiedades

Es una función de JBuilder Enterprise.

El perfeccionamiento por cambio de nombre de una propiedad sólo se puede realizar desde un diagrama UML de clase. Cuando se aplica el perfeccionamiento por cambio de nombre a una propiedad, cambia el nombre de todas sus declaraciones, así como sus métodos de obtención y definición.

Para perfeccionar por cambio de nombre una propiedad:

- 1 Abra en el editor el archivo de código fuente que contiene la propiedad cuyo nombre desea cambiar.
- 2 Vuelva al diagrama UML.
- 3 Haga clic con el botón derecho en la propiedad y seleccione Cambiar nombre a la propiedad. Se abre el cuadro de diálogo Cambiar nombre a propiedad.



- 4 Escriba un nuevo nombre de propiedad para el campo en el campo correspondiente.
- 5 Haga clic en la opción Ver referencias antes de perfeccionar para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.



Si este nombre ya existe en la clase donde se declara la propiedad original, no se permite el perfeccionamiento.

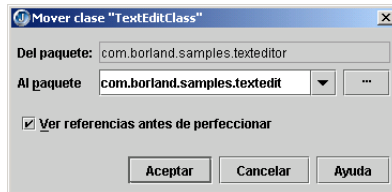
Desplazamiento de una clase a un paquete diferente

En JBuilder es posible aplicar el perfeccionamiento por desplazamiento a las clases. El perfeccionamiento por desplazamiento mueve una clase determinada a otro paquete, a uno ya creado o a uno nuevo. El perfeccionamiento por movimiento sólo es posible en las clases públicas de nivel superior. El paquete al que se desplaza la clase no puede contener un archivo fuente con el mismo nombre.

En el proceso de perfeccionamiento se actualizan la declaración y los usos de la clase. Se actualizan las sentencias de paquete e importación en el archivo fuente de la clase y en todas las clases que contengan una referencia. (Se añade la sentencia `import` a todas las dependencias que tenga la clase en el paquete desde el que se mueve). Debe ser la clase pública de nivel superior.

Se puede desplazar una clase a un nuevo paquete desde el menú Perfeccionar o desde el editor, el panel de estructura o el menú contextual de un diagrama UML. (UML es una función de JBuilder Enterprise). Para desplazar una clase a otro paquete:

- 1 Abra el archivo de clase que desee desplazar en el editor o en un diagrama UML.
- 2 Seleccione el nombre de clase y escoja Perfeccionar|Mover clase. (En el editor, puede hacer clic con el botón derecho y seleccionar Perfeccionamiento|Mover clase. Desde un diagrama UML o el panel de estructura, haga clic con el botón derecho y seleccione Mover clase). Aparece el cuadro de diálogo Mover clase.



- 3 Introduzca el nombre del paquete al que se desplaza la clase en el campo Al paquete.
- 4 Haga clic en la opción Ver referencias antes de perfeccionar para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.



La clase no se traslada si su identificador no es válido o si el nombre del archivo fuente ya existe en el nuevo paquete. En caso necesario, JBuilder añade una sentencia `import` para el antiguo nombre de paquete.

Nota

Si una clase se desplaza a un paquete que no existe, JBuilder lo crea, lo añade al proyecto, crea el nuevo directorio fuente y coloca la clase en él. También actualiza los nombres de paquetes y las sentencias de importación. Asimismo, si el paquete original ya no contiene clases, JBuilder lo elimina del proyecto y borra su directorio fuente.

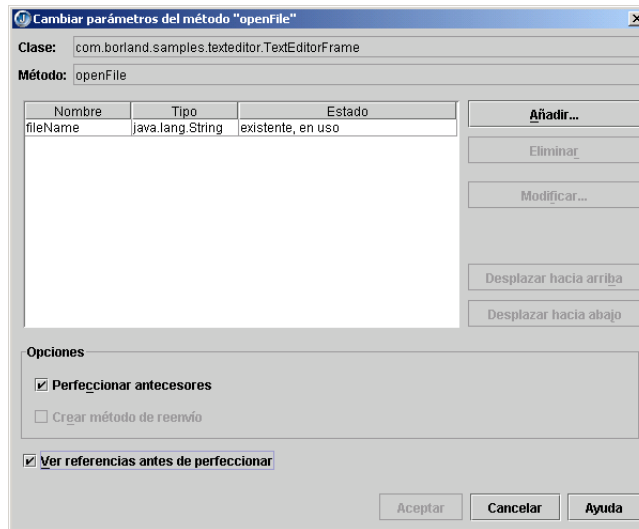
Cambio de parámetros de métodos

Los parámetros de métodos se pueden añadir, eliminar o reordenar desde el menú Perfeccionar o desde el editor, el panel de estructura o el menú contextual de un diagrama UML. (UML es una función de JBuilder Enterprise). Es posible modificar un parámetro recién añadido antes de cerrar el cuadro

de diálogo; sin embargo no se puede modificar un parámetro creado previamente.

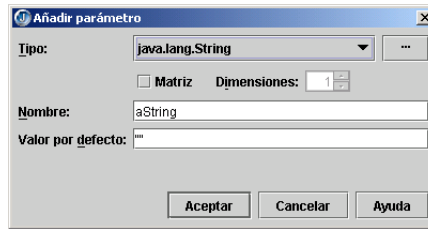
Para cambiar los parámetros de métodos:

- 1 Abra el archivo fuente en el editor o en un diagrama de clase UML.
- 2 Seleccione la firma del método al que desea cambiar los parámetros y escoja **Perfeccionar** | **Cambiar parámetros del método**. (También puede hacer clic con el botón derecho y seleccionar **Perfeccionamiento** | **Cambiar parámetros del método**. Desde un diagrama UML o desde el panel de estructura, haga clic con el botón derecho y escoja **Cambiar parámetros del método**). Se muestra el cuadro de diálogo **Cambiar parámetros del método**. En la lista se muestran los parámetros existentes.



- La columna **Nombre** muestra el nombre del parámetro.
 - La columna **Tipo** muestra el tipo de Java.
 - La columna **Estado** indica si el parámetro es nuevo o ya existente y si está en uso en su código. En los parámetros nuevos, muestra el valor por defecto.
- 3 Pulse el botón **Añadir** para añadir un nuevo parámetro. Se abre el cuadro de diálogo **Añadir parámetro**, donde se elige el tipo de parámetro y se le asignan un nombre y un valor por defecto. Si el nuevo parámetro es una matriz, active la casilla de selección **Matriz**. Escriba el número de

elementos de la matriz en el campo Dimensiones. Haga clic en Aceptar para cerrar el cuadro de diálogo y añadir los nuevos parámetros.



- 4 Seleccione el parámetro y haga clic en Modificar para modificarlo. Se abre el cuadro de diálogo Modificar parámetro, donde se pueden cambiar el nombre, el tipo y el valor por defecto.
- 5 Seleccione un parámetro y pulse el botón Eliminar para eliminarlo. No puede eliminar parámetros ya existentes que se encuentren en uso.
- 6 Utilice los botones Desplazar hacia arriba y Desplazar hacia abajo para reordenar los parámetros de métodos.
- 7 La opción Perfeccionar antecesores (activada por defecto) perfecciona los métodos en las clases de las que la actual es heredera. Desactive Perfeccionar antecesores si desea perfeccionar el método sólo en esta clase y en sus descendientes. Si lo desea, active la opción Crear método de reenvío.
- 8 Haga clic en la opción Ver referencias antes de perfeccionar para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.



Si la firma del nuevo método ya existe en el archivo donde se declara, no se permite el perfeccionamiento. Si la firma existe en otros archivos de la línea directa de herencia, se muestra una advertencia. Si se utiliza la opción Perfeccionar antecesores también puede aparecer una advertencia si el mismo método existe pero no se encuentra en la vía de acceso a fuentes que se puede modificar. Por ejemplo, si el método existe en una biblioteca no es posible perfeccionarlo, ya que las bibliotecas son de sólo lectura.

Tenga en cuenta que el perfeccionamiento se impide si el nombre o el tipo del nuevo parámetro no es un identificador de Java válido.

Extracción de métodos

Utilice el perfeccionamiento por extracción de métodos para convertir un fragmento de código seleccionado en un método. JBuilder extrae el código del método actual, determina los parámetros necesarios, genera las variables locales que procedan y determina el tipo de devolución. Introduce una llamada al nuevo método en el lugar donde se encontraba el fragmento de código. Se encuentra disponible desde el menú Perfeccionar y el menú contextual del editor.

Por ejemplo, el siguiente método se encuentra en `Frame1.java` del `DebugTutorial` disponible en la carpeta `samples/Tutorials` de la instalación de `JBUILDER`. Este método añade dos valores, `valueOneDouble` y `valueTwoDouble`.

```
public void addValues(Double valueOneDouble, Double valueTwoDouble) {
    double valueOneDoubleResult = valueOneDouble.doubleValue();
    double valueTwoDoubleResult = valueTwoDouble.doubleValue();
    addResult = (valueOneDoubleResult + valueTwoDoubleResult);
    addStringResult = Double.toString(addResult);
    addResultDisplay.setText(addStringResult);
}
```

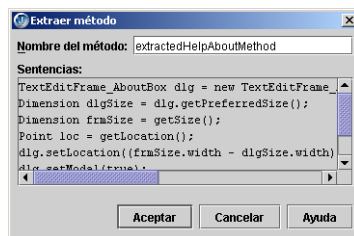
Si se seleccionan las tres últimas líneas del método y se escoge **Perfeccionar! Extraer método**, las líneas seleccionadas de código se extraen al nuevo método y se llaman desde el método original, del siguiente modo (observe que el nuevo método es `private`):

```
public void addValues(Double valueOneDouble, Double valueTwoDouble) {
    double valueOneDoubleResult = valueOneDouble.doubleValue();
    double valueTwoDoubleResult = valueTwoDouble.doubleValue();
    extractedAddValuesMethod(valueOneDoubleResult, valueTwoDoubleResult);
}

private void extractedAddValuesMethod(double valueOneDoubleResult,
                                     double valueTwoDoubleResult) {
    addResult = (valueOneDoubleResult + valueTwoDoubleResult);
    addStringResult = Double.toString(addResult);
    addResultDisplay.setText(addStringResult);
}
```

Para extraer un método:

- 1 Abra el archivo fuente en el editor y seleccione el bloque de código que desea convertir en un método.
- 2 Seleccione **Perfeccionar! Extraer método**. (También puede hacer clic con el botón derecho y seleccionar **Perfeccionamiento! Extraer método**). Se abre el cuadro de diálogo **Extraer método**. El código seleccionado aparece en el campo **Sentencias**.



- 3 Escriba un nuevo nombre para el método en el campo correspondiente.
- 4 Pulse **Aceptar** para completar el proceso de perfeccionamiento.

Importante

No hay ninguna opción de vista previa para este perfeccionamiento. Para deshacer, utilice inmediatamente **Edición! Deshacer**.

JBuilder extrae el código del método actual, determina los parámetros necesarios, genera las variables locales que procedan y determina el tipo de devolución. Introduce una llamada al nuevo método en el lugar donde se encontraba el fragmento de código. JBuilder no le permitirá perfeccionar si más de una variable se encuentra escrita en el bloque o se lee después del bloque.

Nota Si no se selecciona el conjunto de sentencias en su totalidad, JBuilder intentará ampliar la selección hasta llegar a la expresión o sentencia incluyente más cercana .

Introducción de variables

El perfeccionamiento por introducción de variables permite sustituir el resultado de una expresión compleja o una parte de ésta por un nombre de variable temporal. Este nombre debe explicar la finalidad de la expresión o la subexpresión. También se conoce como "variable explicativa". Sólo se puede acceder a este perfeccionamiento desde el menú contextual del editor y desde el menú Perfeccionamiento.

Por ejemplo, el siguiente método `actionPerformed()` antes del perfeccionamiento configura el color de fondo y el texto de un botón:

```
void jButton1_actionPerformed(ActionEvent e) {  
    jButton1.setBackground(new Color(255,0,0));  
    jButton1.setText("Hello World");  
}
```

Si se selecciona la cadena `Hello World` y, a continuación, se escoge **Perfeccionar!Introducir variable**, el código tiene la siguiente apariencia:

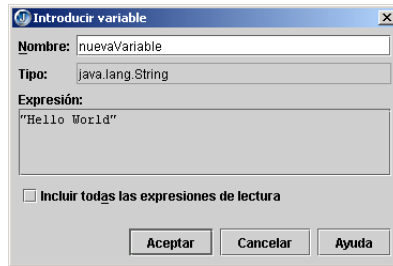
```
void jButton1_actionPerformed(ActionEvent e) {  
    jButton1.setBackground(new Color(255,0,0));  
    final String newVariableString = "Hello World";  
    jButton1.setText(newVariableString);  
}
```

Observe la nueva variable temporal `final` en el cuerpo del método. La variable recién generada ha sustituido a la expresión original.

Para introducir una variable:

- 1 Abra el archivo fuente en el editor y seleccione la expresión compleja que desea sustituir por una variable temporal.
- 2 Seleccione **Perfeccionar!Introducir variable**. (También puede hacer clic con el botón derecho y seleccionar **Perfeccionamiento!Introducir variable**). Se

abre el cuadro de diálogo Introducir variable. La expresión seleccionada aparece en el campo Expresión.



- 3 Escriba un nombre para la nueva variable en el campo correspondiente.
- 4 Active la opción Incluir todas las expresiones de lectura para reemplazar todas las lecturas de esa expresión. Si esta opción está desactivada, sólo se reemplaza la selección actual.
- 5 Haga clic en Aceptar para cerrar el cuadro de diálogo. El perfeccionamiento se completa automáticamente.

Importante

No hay ninguna opción de vista previa para este perfeccionamiento. Para deshacer, utilice inmediatamente Edición|Deshacer.

Introducción de campos

El perfeccionamiento por introducción de campo permite sustituir el resultado de una expresión compleja o una parte de ésta por un nombre de campo. Si se usa este perfeccionamiento, el nuevo campo se crea con la visibilidad y los modificadores seleccionados. Se encuentra disponible desde el menú Perfeccionar y el menú contextual del editor.

Por ejemplo, el siguiente método `jbInit()` antes del perfeccionamiento configura el tamaño de un applet y le añade un botón:

```
private void jbInit() throws Exception {
    this.setSize(new Dimension(400.300, 300));
    this.add(new Button("button"));
}
```

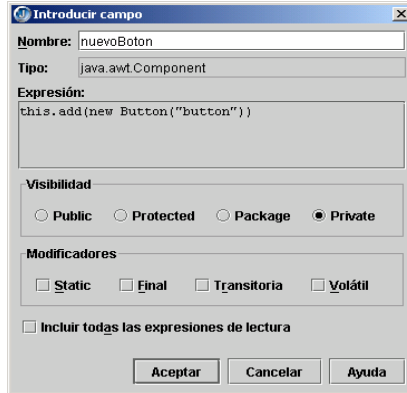
Si se selecciona el fragmento de código `new(Button("button"))` y, a continuación, se escoge Perfeccionar|Introducir campo, el código tiene la siguiente apariencia:

```
private Button newFieldButton = new Button("button");
private void jbInit() throws Exception {
    this.setSize(new Dimension(400.300, 300));
    this.add(newFieldButton);
}
```

Observe cómo la línea de código `this.add` llama al campo `newFieldButton` que se define inmediatamente antes del método.

Para introducir un campo:

- 1 Abra el archivo fuente en el editor y seleccione la expresión compleja que desea sustituir por un campo temporal.
- 2 Seleccione **Perfeccionar** | **Introducir campo**. (También puede hacer clic con el botón derecho y seleccionar **Perfeccionamiento** | **Introducir campo**). Se abre el cuadro de diálogo **Introducir campo**. La expresión seleccionada aparece en el campo **Expresión**.



- 3 Escriba el nombre del nuevo campo en el campo **Nombre**.
- 4 Escoja la visibilidad del nuevo campo mediante las opciones de visibilidad. Por defecto es `private`.
- 5 Seleccione el modificador, si lo hay, mediante las opciones del modificador. Se puede seleccionar más de un modificador.
- 6 Active la opción **Incluir todas las expresiones de lectura** para reemplazar todas las lecturas de esa expresión. Si esta opción está desactivada, sólo se reemplaza la selección actual.
- 7 Haga clic en **Aceptar** para cerrar el cuadro de diálogo. El perfeccionamiento se completa automáticamente. El nuevo campo se inserta en el código en la ubicación correcta.

Importante No hay ninguna opción de vista previa para este perfeccionamiento. Para deshacer, utilice inmediatamente **Edición** | **Deshacer**.

Perfeccionamiento con sentencia `try/catch`

Consiste en añadir una sentencia `try/catch` alrededor de un bloque de código seleccionado. JBuilder detecta todas las expresiones activas del bloque y añade un bloque para cada una de ellas. Sólo se puede acceder a este perfeccionamiento desde el menú contextual del editor y desde el menú **Perfeccionamiento**.

Por ejemplo, el siguiente método se encuentra en `Frame1.java` del `DebugTutorial` disponible en la carpeta `samples/Tutorials` de la instalación de `JBuilder`. Este método sustrae dos valores, `valueOneDouble` y `valueTwoDouble`.

```
public void subtractValues(Double valueOneDouble, Double valueTwoDouble) {
    double valueOneDoubleResult = valueOneDouble.doubleValue();
    double valueTwoDoubleResult = valueTwoDouble.doubleValue();
    subtractResult = (valueOneDoubleResult - valueTwoDoubleResult);
    subtractStringResult = Double.toString(subtractResult);
    subtractResultDisplay.setText(subtractStringResult);
}
```

Si se seleccionan las tres últimas líneas del método y se escoge **Perfeccionar** | **Insertar en bloque try/catch**, el código tiene la siguiente apariencia:

```
public void subtractValues(Double valueOneDouble, Double valueTwoDouble) {
    double valueOneDoubleResult = valueOneDouble.doubleValue();
    double valueTwoDoubleResult = valueTwoDouble.doubleValue();
    try {
        subtractResult = (valueOneDoubleResult - valueTwoDoubleResult);
        subtractStringResult = Double.toString(subtractResult);
        subtractResultDisplay.setText(subtractStringResult);
    }
    catch (Exception ex) {
    }
}
```

Para englobar un bloque en una sentencia `try/catch`:

- 1 Seleccione el bloque de código en el editor.
- 2 Seleccione **Perfeccionar** | **Insertar en bloque try/catch**. También puede hacer clic con el botón derecho y seleccionar **Perfeccionamiento** | **Insertar en bloque try/catch**.

Importante No hay ninguna opción de vista previa para este perfeccionamiento. Para deshacer, utilice inmediatamente **Edición** | **Deshacer**.

El código queda englobado en una sentencia `try/catch`: si no se ha seleccionado un bloque de sentencias válido, en la pestaña **Perfeccionamiento** aparece un error y no se realiza la acción.

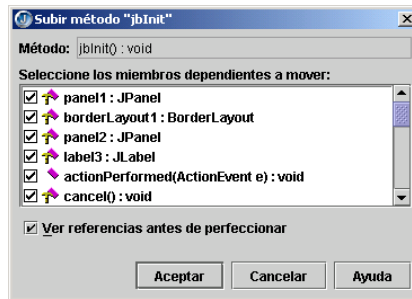
Subida de métodos

El perfeccionamiento **Subir método** se utiliza para mover el método seleccionado a una superclase de su clase actual. En la superclase se añaden sentencias `import` cuando sea necesario.

Se encuentra disponible desde el menú **Perfeccionar** y el menú contextual del editor.

Para subir un método:

- 1 Abra en el editor el archivo fuente que contenga el método que desea subir.
- 2 Seleccione el método y escoja Perfeccionar|Subir método. También puede hacer clic con el botón derecho del ratón y seleccionar Perfeccionamiento|Subir método. Se abre el cuadro de diálogo Subir método.



- 3 Seleccione los métodos dependientes que se van a mover en la lista. Seleccione los métodos dependientes a mover. Utilice las teclas de flecha para desplazarse entre los elementos de la lista; use *Mayús+teclas de flecha* para seleccionar varios elementos.
- 4 Haga clic en la opción Ver referencias antes de perfeccionar para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.



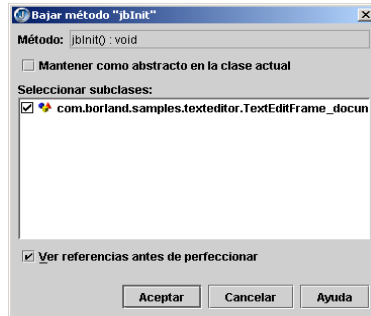
Bajada de métodos

El perfeccionamiento Bajar método se utiliza para mover el método seleccionado a una o más subclases de la clase actual. Se encuentra disponible desde el menú Perfeccionar y el menú contextual del editor.

Para bajar un método:

- 1 Abra en el editor el archivo fuente que contenga el método que desea bajar.

- 2 Seleccione el método y escoja Perfeccionar|Bajar método. También puede hacer clic con el botón derecho y seleccionar Perfeccionamiento|Bajar método. Se abre el cuadro de diálogo Bajar método.



- 3 Seleccione la opción Mantener como abstracto en la clase actual para mantener el método como un método `abstract` en la superclase.
- 4 Seleccione las subclases para bajar el método en la lista Seleccionar subclases. Observe que la subclase ya debe estar creada en el proyecto; este cuadro de diálogo no crea nuevos archivos de clase.
- 5 Haga clic en la opción Ver referencias antes de perfeccionar para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.



Subida de campos

El perfeccionamiento Subir campo se utiliza para mover un campo a una superclase. Si el campo se utiliza en otra subclase, cambia la visibilidad. Se encuentra disponible desde el menú Perfeccionar y el menú contextual del editor.

Para subir un campo:

- 1 Abra en el editor el archivo fuente que contenga en campo que desea subir.
- 2 Seleccione el campo y escoja Perfeccionar|Subir campo. También puede hacer clic con el botón derecho y seleccionar Perfeccionamiento|Subir campo. Aparece el cuadro de diálogo Subir campo.



- 3 Haga clic en la opción Ver referencias antes de perfeccionar para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga



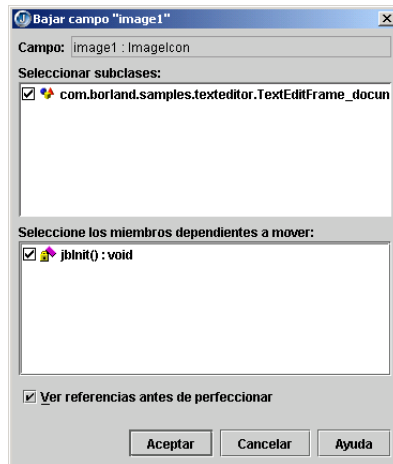
clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.

Bajada de campos

El perfeccionamiento Bajar campo se utiliza para mover un campo de una superclase a una subclase. Este perfeccionamiento se puede usar si ya no se necesita un campo en la superclase, pero sí en la subclase. Se encuentra disponible desde el menú Perfeccionar y el menú contextual del editor.

Para bajar un campo:

- 1 Abra en el editor el archivo fuente que contenga el campo que desea bajar.
- 2 Seleccione el campo y escoja Perfeccionar|Bajar campo. También puede hacer clic con el botón derecho del ratón y seleccionar Perfeccionamiento|Bajar campo. Se abre el cuadro de diálogo Bajar campo.



- 3 El nombre aparece en el campo Campo. Es de sólo lectura.
- 4 En la lista Seleccionar subclases, seleccione las subclases a las que desee bajar el campo.
- 5 Seleccione los miembros dependientes que desea mover a la subclase seleccionada en la lista Seleccione los miembros dependientes a mover.
- 6 Haga clic en la opción Ver referencias antes de perfeccionar para ver los cambios antes de terminar el perfeccionamiento y, a continuación, haga clic en Aceptar. Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.



Aparece una advertencia si otras subclases o clases externas utilizan este método. Una vez que se muestra la advertencia, continúa el perfeccionamiento.

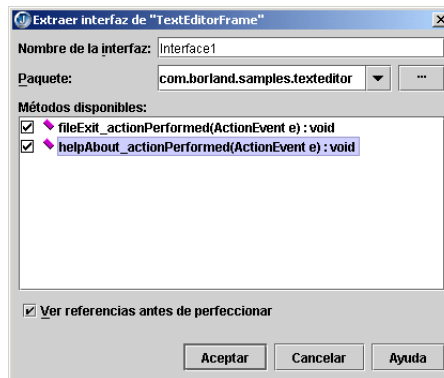
Extracción de interfaces

El perfeccionamiento por Extraer interfaz se utiliza para extraer el subconjunto seleccionado de una clase a una nueva interfaz. Este perfeccionamiento se usa si varios clientes utilizan el mismo subconjunto de una interfaz o si dos clases tienen en común parte de sus interfaces. JBuilder añade sentencias `implements` a las clases que implementan la interfaz. Una vez que se ha extraído la interfaz, se pueden mover métodos y campos a la nueva interfaz mediante los comandos Bajar método y Bajar campo.

Se puede acceder a este perfeccionamiento desde el menú Perfeccionar o desde los menús contextuales del editor y del diagrama UML. (UML es una función de JBuilder Enterprise).

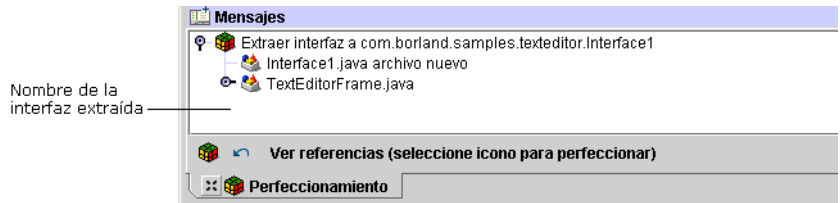
Para extraer una interfaz:

- 1 Abra el archivo de clase en el editor, seleccione la declaración de clase y escoja Perfeccionar|Extraer interfaz de. También puede hacer clic con el botón derecho del ratón y seleccionar Perfeccionamiento|Extraer interfaz de. Desde un diagrama UML, haga clic con el botón derecho y seleccione Extraer interfaz de. Aparece el cuadro de diálogo Extraer interfaz de clase.



- 2 Introduzca el nombre del nuevo interfaz en el campo Interfaz:
- 3 Si la interfaz va a estar en otro paquete, seleccione el nombre del paquete en el campo Paquete. Pulse el botón de puntos suspensivos (...) para desplazarse hasta el paquete.
- 4 Seleccione los métodos que desea extraer a la nueva interfaz en la lista desplegable Métodos disponibles.
- 5 Active la opción Ver referencias antes de perfeccionar si desea examinar los cambios antes de realizar el perfeccionamiento, y seguidamente pulse

Aceptar. Observe que el nombre de la interfaz aparece en la pestaña de vista previa, tal y como se muestra a continuación:



- 6 Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.

Para ver la nueva interfaz, haga clic en su nombre en la pestaña Perfeccionamiento. La nueva interfaz se abre en el editor.

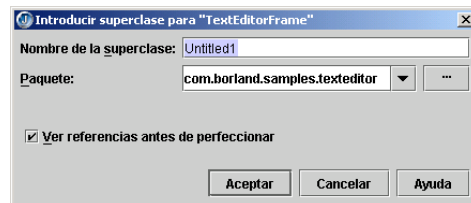
Introducción de superclases

El perfeccionamiento Introducir superclase se utiliza para crear una superclase. Las características comunes de dos o más clases se mueven a la superclase. Este perfeccionamiento se usa cuando dos o más clases tienen características parecidas. Una vez creada la superclase, se pueden mover métodos y campos a la nueva interfaz mediante los comandos Subir método y Subir campo.

Se puede acceder a este perfeccionamiento desde el menú Perfeccionar o desde los menús contextuales del editor y del diagrama UML. (UML es una función de JBuilder Enterprise).

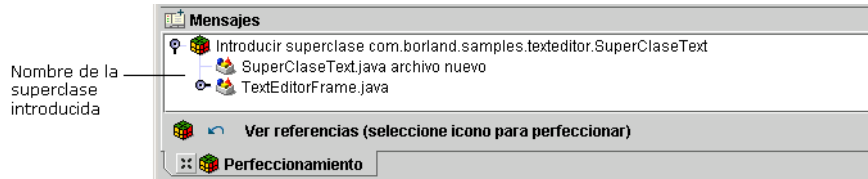
Para introducir una superclase:

- 1 Abra la clase en el editor, seleccione la declaración de clase y escoja Perfeccionar|Introducir superclase. También puede hacer clic con el botón derecho del ratón y seleccionar Perfeccionamiento|Introducir superclase. Se abre el cuadro de diálogo Introducir superclase para.



- 2 Escriba el nombre de la superclase en el campo Nombre de la superclase.
- 3 Si la superclase va a estar en otro paquete, seleccione el nombre de ese paquete en el campo Paquete. Pulse el botón de puntos suspensivos (...) para desplazarse hasta el paquete.
- 4 Active la opción Ver referencias antes de perfeccionar si desea examinar los cambios antes de realizar el perfeccionamiento, y seguidamente pulse

Aceptar. Observe que el nombre de la nueva superclase aparece en la pestaña de vista previa, tal y como se muestra a continuación:



Pulse el botón Perfeccionar de la pestaña Perfeccionamiento para completarlo.

Para ver la nueva superclase, haga clic en el nombre en la pestaña Perfeccionamiento. La nueva superclase se abre en el editor.



Test de módulos

La comprobación de módulos es una característica de todas las ediciones de JBuilder. La compatibilidad con Cactus es una característica de JBuilder Enterprise. Los asistentes para montajes para tests son funciones de JBuilder Enterprise.

El test de módulos consiste en escribir pruebas para partes pequeñas de código, definidas a discreción del desarrollador como, por ejemplo, un método y, a continuación, ejecutarlas y analizar sus resultados. Cuando un desarrollador realiza tests de módulos como parte del proceso de desarrollo, debe escribir varias pruebas pequeñas, que se pueden repetir, y ejecutarlas con regularidad. Las ventajas son la creación de un código más fiable y la detección temprana de *regresiones* cuando se modifica el código. Las regresiones son errores que aparecen en un código que funcionaba anteriormente. Muchas metodologías recomiendan la ejecución de pruebas de módulos como parte del proceso de generación de proyectos. Si el resultado de los tests de módulos es negativo, se considera que el proceso de generación es erróneo.

JUnit

JUnit es un entorno de código abierto para el test de módulos, creado por Erich Gamma y Kent Beck. JUnit proporciona diversas herramientas para el test de módulos, entre las que se encuentran dos clases, `junit.framework.TestCase` y `junit.framework.TestSuite`, que se utilizan como base para escribir comprobaciones parciales. JUnit proporciona también tres ejecutores de tests distintos: TextUI, SwingUI y AwtUI. Dos de ellos, TextUI y SwingUI, están disponibles en el IDE de JBuilder. Si desea más información sobre JUnit, visite

<http://www.junit.org>. También hay documentación disponible sobre JUnit en el directorio <jbuilder>/thirdparty/<junit>/doc.

JBuilder integra el entorno de test de módulos de JUnit. Esto significa que se pueden crear y ejecutar pruebas de JUnit desde el IDE de JBuilder. Además de las eficaces funciones de comprobación de módulos de JUnit y Cactus, JBuilder incluye asistentes para la creación de tests, conjuntos de tests y montajes para tests, así como un ejecutor llamado JTestRunner que combina elementos de texto e interfaz gráfica en la salida y se integra directamente en el IDE de JBuilder.

Consulte

- “Detección de tests” en la página 14-3
- “Creación de tests y conjuntos de tests JUnit” en la página 14-5
- “Ejecución de tests” en la página 14-14

Cactus

La compatibilidad con Cactus es una característica de JBuilder Enterprise. El Asistente para cliente de prueba EJB es una función de JBuilder Enterprise.

Cactus extiende JUnit para suministrar test de módulos del código Java de la parte del servidor. Esto lo hace redirigiendo la prueba a un proxy en el lado del servidor. Si desea obtener más información acerca de Cactus, visite <http://jakarta.apache.org/cactus/index.html>. La documentación de Cactus también está disponible en el directorio <jbuilder>/thirdparty/<jakarta-cactus>/doc.

JBuilder ofrece varias funciones que facilitan el test de Cactus. El Asistente para la configuración de Cactus permite configurar el proyecto para que admita el test Cactus. El Asistente para cliente de prueba EJB puede generar un test Cactus para su Enterprise JavaBean (EJB). La integración de Cactus de JBuilder en el entorno permite ejecutar los tests Cactus dentro del IDE de JBuilder si hay disponible un servidor y un proyecto debidamente configurados.

Consulte

- “Utilización de Cactus” en la página 14-11
- “Asistente para la configuración de Cactus” en la página 14-12
- “Ejecución y prueba de un bean enterprise” en *Desarrollo de aplicaciones con Enterprise JavaBeans*

Funciones de test de módulos de JBuilder

Las funciones de comprobación de módulos de JBuilder integran JUnit y Cactus en el IDE de JBuilder y proporcionan herramientas para la creación de tests de módulos y su organización en conjuntos, así como para la ejecución, análisis y depuración de los tests. JBuilder proporciona un conjunto de montajes para tests con el objeto de realizar tareas comunes a varios tests. JBuilderTestRunner de JBuilder ofrece una forma de ejecución de tests que combina la salida por texto e interfaz gráfica. JBuilder incluye las siguientes funciones de test de módulos:

Éstas son funciones de todas las ediciones de JBuilder:

- Asistente para tests
- Asistente para conjuntos de tests
- Ejecución de tests
- JBuilderTestRunner
- Compatibilidad con JUnit TextUI
- Compatibilidad con JUnit SwingUI
- Test del filtro del seguimiento de la pila
- Depuración de tests

A continuación se enumeran características de JBuilder Enterprise:

- Asistente para cliente de prueba EJB
- Comprobación de Cactus
- Montaje JDBC
- Montaje JNDI
- Tests por comparación
- Asistente para montajes personalizados
- Recopilador de tests de JUnit

Detección de tests

Por defecto, JBuilder identifica automáticamente como tests las clases que son ampliaciones de `junit.framework.TestCase` o `junit.framework.TestSuite`. Si una clase se identifica como test, cuando se pulsa con el botón derecho del ratón el nombre del archivo fuente en el panel de proyecto o la pestaña en la que figura su nombre, si este archivo fuente está abierto en el editor, aparece un menú contextual que contiene las opciones Ejecutar test y Depurar test. La opción Optimizar test está también disponible si Borland Optimizeit está instalado adecuadamente.

El recopilador de test de JUnit proporciona un método alternativo de identificación de tests.

Recopilador de tests de JUnit

El Recopilador de tests de JUnit es una función de JBuilder Enterprise.

El recopilador de test de JUnit es una función de JBuilder que proporciona una interfaz gráfica de usuario (GUI) para la clase `PackageTestSuite`. Esto resulta útil para la detección de tests, para que no necesite mantener una lista de todas las clases de tests. JUnit Test Collector está disponible en el cuadro de diálogo Configuración de ejecución para el tipo Test de configuración de ejecución. Se activa seleccionando el botón circular Paquete de la ficha Ejecutar de este cuadro de diálogo.

Para añadir una configuración de ejecución tipo Test que utilice el recopilador de tests de JUnit:

- 1 Seleccione ProyectoPropiedades de proyecto.
- 2 Abra la ficha Ejecutar del cuadro de diálogo Propiedades de proyecto.
- 3 Pulse el botón Nuevo.
- 4 Seleccione la ficha Ejecutar del cuadro de diálogo Nueva configuración de ejecución.
- 5 Asigne Test a Tipo.
- 6 Pulse el botón circular Paquete. Esto activa el recopilador de tests de JUnit y desactiva el modo de detección de tests por defecto.
- 7 Especifique si desea o no que el analizador de tests incluya los subpaquetes activando o desactivando Incluir subpaquetes.
- 8 Especifique las cadenas con las que empiezan o terminan los nombres de clase de tests en los campos El nombre comienza con y El nombre finaliza con. Al hacerlo, se restringen los tests que encuentra el analizador de tests a aquellos que contengan dichas cadenas. Este paso es optativo.
- 9 Pulse Aceptar para guardar la configuración de ejecución y cierre el cuadro de diálogo Nueva configuración de ejecución.
- 10 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

Los tests que coincidan con el filtro que se estableció en el cuadro de diálogo Propiedades de configuración de ejecución, ahora se identifican correctamente como tests. Esto significa que las opciones Ejecutar test y Depurar test aparecen en el menú contextual si pulsa con el botón derecho del ratón uno de los tests que coinciden en el panel de proyecto. La opción Optimizar test también está disponible si Borland Optimizeit está instalado adecuadamente.

Consulte

- [“Definición de las configuraciones de ejecución” en la página 7-8](#)

Creación de tests y conjuntos de tests JUnit

Un test es una instancia de `junit.framework.TestCase`. Los tests contienen uno o más métodos que ejecutan una o más partes de una clase de la aplicación que se comprueba. También contiene los métodos `setUp()` y `tearDown()`. El método `setUp()` se utiliza para llevar a cabo la configuración necesaria antes de ejecutar cada método de test. El método `tearDown()` se utiliza para limpiar y liberar recursos después de haber ejecutado cada método de test. Cuando se ejecutan tests de JUnit se crea una instancia de la clase de test para cada método. Los métodos `setUp()` y `tearDown()` se ejecutan una vez por instancia. Por ejemplo, en un test llamado `MyTestCase` que contiene los métodos `testMethod1()` y `testMethod2()`, el orden de ejecución es el siguiente:

- 1 El ejecutor de tests crea dos instancias de `MyTestCase`.
- 2 Se llama al método `setUp()`.
- 3 Se llama al método `testMethod1()`.
- 4 Se llama al método `tearDown()`.
- 5 Se llama al método `setUp()`.
- 6 Se llama al método `testMethod2()`.
- 7 Se llama al método `tearDown()`.

Tanto un test como un conjunto de tests amplían `TestCase`. La diferencia entre los tests y los conjuntos es que los primeros contienen métodos de test individuales, mientras que los segundos se utilizan para organizar conjuntos en un grupo lógico y ejecutarlos juntos. Los conjuntos pueden contener llamadas a cualquier número de tests o a otros conjuntos de tests.

Un objetivo importante de la comprobación de tests es crear tests que se repitan. Si los tests se repiten siempre dan el mismo resultado cuando el software sometido al test funciona correctamente. Si uno de los métodos que se comprueba contiene errores, el resultado del test es negativo. Si se ejecutan tests de módulos cada vez que se realizan cambios en el software, se verifica con más facilidad que no se han introducido errores ni regresiones.

El desarrollador decide el número de tests. Algunos desarrolladores escriben tests para todos los métodos públicos del código. No es necesario alcanzar este nivel de seguimiento para ofrecer cierta protección contra las regresiones. Al principio puede resultar conveniente concentrarse en la escritura de pruebas para las partes más críticas del software o las que presentan más probabilidades de fallo.

Si el código es correcto, los métodos de test devuelven un resultado esperado. De lo contrario, proporcionan información útil para la localización del origen del fallo. El Asistente para tests crea estructuras de métodos de tests; el desarrollador decide qué resultados son significativos y aporta la implementación.

Consulte

- [Capítulo 23, “Tutorial: Creación y ejecución de tests y de conjuntos de tests”](#)

El Asistente para tests

El Asistente para tests se emplea para crear clases de test que amplían `TestCase` y contienen definiciones de métodos con las secciones principales vacías, que se deben llenar con arreglo a la clase que se desea comprobar. Para abrir el Asistente para tests seleccione `Archivo|Nuevo|Probar`, seleccione `Test` y haga clic en `Aceptar`. El Asistente para tests crea pruebas en el directorio fuente del proyecto especificado en la ficha `Vías de acceso` del cuadro de diálogo `Propiedades de proyecto`. Si desea ver o modificar el directorio fuente de tests, elija `Proyecto|Propiedades de proyecto`, abra la ficha `Vías de acceso` y pulse la pestaña `Fuente`.

El Asistente para tests permite elegir la clase y los métodos que se deben comprobar, utilizar montajes para tests predefinidos y crear una configuración de ejecución para el test. Si desea más información sobre la interfaz del Asistente para tests pulse el botón `Ayuda`.

Consulte

- [Capítulo 23, “Tutorial: Creación y ejecución de tests y de conjuntos de tests”](#)
- [“Montajes para tests predefinidos” en la página 14-8](#)

Adición de código a los tests

El Asistente para tests crea la estructura de los tests, pero el desarrollador debe introducir el código. El Asistente para tests señala las zonas del código que se deben rellenar con comentarios `@todo` de Javadoc. Estos comentarios se muestran en el panel de estructura, en el nodo `"Por Hacer"` del árbol. Para completar los tests, se debe añadir código a todos los métodos.

A continuación se presenta un ejemplo de un método de test sencillo:

```
public void testSum() {  
    assertEquals( 2, sum(1,1) );  
}
```

Los métodos de test deben ser `public` y `void`, y no deben llevar argumentos.

Cuando se añade código de test a los métodos es necesario tener la forma de averiguar si el resultado del test ha sido positivo o negativo. Es posible crear un método de test que examine varias condiciones y devuelva un fallo si no se cumplen. La forma más normal de conseguirlo consiste en llamar a uno de los métodos de orden en `junit.framework.Assert`, por ejemplo:

- `assertEquals()`: ordena que los argumentos que se le pasan tengan el valor equal.

- `assertTrue()`: ordena que el valor de la expresión booleana que se le pasa sea `true`.
- `assertNotNull()`: ordena que el argumento que se le pasa tenga un valor distinto de `null`.

En `junit.framework.Assert` hay varias versiones sobrecargadas de estos métodos. Las diversas firmas de los métodos toman argumentos de distintos tipos, lo que los hace más flexibles. Si la condición no se cumple, estos métodos desencadenan un fallo del test, del que informa el ejecutor de tests. Si un método de test concluye sin desencadenar ningún fallo, el ejecutor de tests comunica que el resultado es correcto. Los métodos de orden se pueden llamar directamente desde el test, porque `TestCase` es una subclase de `Assert`.

Sugerencia

Si desea ver los métodos de `junit.framework.Assert`, abra un test en el editor; en el panel de estructura haga doble clic en `TestCase`, la clase antecesora y, a continuación, haga doble clic en `Assert`, su clase antecesora.

También es posible escribir un método de test que lance una excepción. A continuación se propone un ejemplo.

```
public void testException() throws Exception {
    throw new Exception("ouch!");
}
```

Cuando un test arroja una excepción que no gestiona, el ejecutor de tests informa que el método ha fallado.

Si desea más información sobre la escritura de test con JUnit, consulte el artículo de Kent Beck y Erich Gamma, "JUnit Test Infected: Programmers Love Writing Tests", (en inglés) en el sitio web de JUnit.

El Asistente para conjuntos de tests

El Asistente para conjuntos de tests se utiliza para crear conjuntos de tests que se ejecutan por lotes. Para abrir el Asistente para conjuntos de tests, seleccione `Archivo|Nuevo|Probar`, seleccione `Conjunto de tests` y haga clic en `Aceptar`.

En este asistente se pueden elegir los tests que se deben incluir en el conjunto, así como crear una configuración de ejecución. Si desea más información sobre la interfaz del Asistente para conjuntos de tests, pulse el botón `Ayuda`.

Consulte

- [Capítulo 23, "Tutorial: Creación y ejecución de tests y de conjuntos de tests"](#)

El Asistente para clientes de prueba EJB

El Asistente para cliente de prueba EJB es una función de JBuilder Enterprise.

El Asistente para clientes de prueba EJB, disponible en la ficha Enterprise de la galería de objetos, permite crear tres tipos diferentes de clientes de prueba para probar sus Enterprise JavaBeans (EJB). De estos tipos, dos, cliente de prueba JUnit y cliente de prueba Cactus JUnit, están diseñados para el test de módulos.

Sugerencia

Aunque es posible crear un test que pruebe EJB mediante el Asistente para tests, es mejor utilizar el Asistente para clientes de prueba EJB al probar EJB. Esto se debe a que el Asistente para clientes de prueba EJB genera más código específico de EJB.

Consulte

- "Ejecución y prueba de un bean enterprise" en *Desarrollo de aplicaciones con Enterprise JavaBeans*

Montajes para tests predefinidos

Los montajes para tests predefinidos son características de JBuilder Enterprise.

Los montajes para tests son clases de utilidades que se pueden emplear en los tests con el fin de realizar tareas rutinarias de creación de entornos. Un ejemplo es la gestión de las conexiones de base de datos con datos utilizados con fines de comprobación.

El Asistente para tests de JBuilder puede instalar automáticamente los montajes para tests si cuentan con un constructor que tome el argumento `Object` y contenga los métodos `setUp()` y `tearDown()`. He aquí un ejemplo básico de montaje válido:

```
public class CustomFixture1 {

    public CustomFixture1(Object obj) {
        // aquí va el código
    }

    public void setUp() {
        // aquí va el código
    }

    public void tearDown() {
        // aquí va el código
    }

}
```

Con el fin de instalar un montaje de este tipo en un test nuevo, selecciónelo en el tercer paso del Asistente para tests. El test crea una instancia del montaje, y se llama a sus métodos `setUp()` y `tearDown()`.

JBuilder proporciona estos tres montajes predefinidos para la realización de tareas comunes:

- Montaje JDBC
- Montaje JNDI
- Montaje para comparación

También es posible crear montajes para tests personalizados con el Asistente para montajes de tests.

Montaje JDBC

El montaje JDBC es una característica de JBuilder Enterprise.

El montaje JDBC, `com.borland.jbuilder.unittest.JdbcFixture`, se puede utilizar en las pruebas para gestionar conexiones JDBC. Los métodos del test pueden utilizar el método `getConnection()` con el fin de obtener una conexión JDBC. Para indicar una conexión JDBC, utilice los métodos `setUrl()` y `setDriver()`. El método `runSqlFile()` se utiliza para ejecutar archivos de script SQL.

La forma más sencilla de crear un montaje JDBC consiste en utilizar el Asistente para montajes para JDBC. El Asistente para montajes para JDBC crea una clase que amplía `JdbcFixture`. La ampliación de `JdbcFixture` permite especificar la conexión JDBC necesaria para utilizar y proporcionar otras funciones, en caso necesario. He aquí un resumen de los métodos más utilizados en `JdbcFixture`:

- `dumpResultSet()` vuelca en `Writer` los valores de un conjunto de resultados. Toma como parámetros `java.sql.ResultSet` y `java.io.Writer`.
- `getConnection()` devuelve un objeto `java.sql.Connection` que define la conexión JDBC.
- `runSqlBuffer()` ejecuta una sentencia SQL contenida en `StringBuffer`.
- `runSqlFile()` lee un script SQL de un archivo y lo ejecuta. Toma como parámetros una cadena `String` que indica la posición del archivo y un valor booleano.
- `setDriver()` configura la propiedad `Driver` de la conexión JDBC. Toma una cadena `String` como parámetro.
- `setUrl()` configura la propiedad `URL` de la conexión JDBC. Toma una cadena `String` como parámetro.
- `setUsername()` establece el nombre de usuario para el acceso a la conexión JDBC. Toma una cadena `String` como parámetro.
- `setPassword()` establece la contraseña para el acceso a la conexión JDBC. Toma una cadena `String` como parámetro.

Sugerencia

Los montajes JDBC creados por medio del asistente amplían `JdbcFixture`. La estructura de la clase heredada se puede presentar en el panel de estructura, haciendo doble clic sobre el nodo de la clase heredada en el panel de estructura con el montaje JDBC abierto en el editor. Otro método consiste en

hacer clic con el botón derecho del ratón en el nombre de la clase antecesora y seleccionar Buscar definición en el menú contextual.

Si desea más información sobre la interfaz del Asistente para montajes para JDBC, pulse el botón Ayuda.

Consulte

- [Capítulo 24, “Tutorial: Utilización de montajes para tests”](#)

Montaje JNDI

El montaje JNDI es una característica de JBuilder Enterprise.

Los montajes JNDI son clases que facilitan la realización de consultas JNDI. Se pueden crear con un asistente. El Asistente para montajes para JNDI se encuentra en la ficha Test de la galería de objetos. Si desea más información sobre la interfaz del Asistente para montajes para JNDI, pulse el botón Ayuda.

Montaje para comparación

El montaje de comparación es una característica de JBuilder Enterprise.

Los montajes para comparación se utilizan para registrar el resultado de un test y compararlo con el de test anteriores y posteriores. Los montajes para comparación son clases que amplían `com.borland.jbuilder.unittest.TestRecorder`. La clase `TestRecorder` es una ampliación de `java.io.Writer`, por lo que los montajes para comparación se pueden utilizar siempre que se requiera un `Writer`. Los montajes para comparación se pueden generar por medio del asistente que se encuentra en la ficha Test de la galería de objetos.

`TestRecorder` contiene cuatro constantes que establecen el modo de registro:

- `UPDATE` - El montaje de comparación compara la nueva salida con un archivo de salida existente, o lo crea si no existe y graba la salida en él.
- `COMPARE` - El montaje de comparación siempre compara la nueva salida con la ya existente.
- `RECORD` - El montaje de comparación graba todas las salidas, sobrescribiendo cualquiera ya existente en el archivo de salida.
- `OFF` - El montaje de comparación está desactivado.

Tenga en cuenta que si se añaden o se borran cadenas de un test o un conjunto tras el registro de los resultados, es necesario reinicializar el archivo de datos. Si las pruebas han cambiado, utilice `RECORD` en lugar de `UPDATE` o borre el archivo de datos. Se trata de un archivo binario que se encuentra en el mismo directorio que los archivos de código fuente de prueba. Lleva el mismo nombre que el test.

He aquí un resumen de los métodos más utilizados en los montajes para comparación:

- `print()` imprime una cadena que se le pasa como parámetro.
- `println()` imprime una cadena que se le pasa como parámetro, con un salto de línea.

- `compareObject()` llama al método `equals()` de un objeto y compara un objeto que se le pasa con otro registrado anteriormente por medio de `recordObject()`.
- `recordObject()` registra un objeto que más adelante se comparará con otro por medio de `compareObject()`.

Si desea más información sobre la interfaz del Asistente para montajes para comparación, pulse el botón Ayuda.

Consulte

- [Capítulo 24, "Tutorial: Utilización de montajes para tests"](#)

Creación de un montaje para tests personalizado

El Asistente para montajes personalizados es una función de JBuilder Enterprise.

También es posible escribir montajes personalizados con el fin de realizar tareas habituales en los tests. Los tests podrán compartir así el montaje personalizado. El Asistente para montajes personalizados resulta útil para generar la estructura de montajes para tests o crear un englobador para el código ya escrito. La estructura de los montajes para tests personalizados incluye los métodos `setUp()` y `tearDown()`. El Asistente para montajes personalizados se encuentra en la ficha Test de la galería de objetos. Si desea más información sobre la interfaz del Asistente para montajes personalizados, pulse el botón Ayuda.

Utilización de Cactus

La compatibilidad con Cactus es una característica de JBuilder Enterprise.

Cactus extiende JUnit para suministrar test de módulos del código Java de la parte del servidor. Resulta útil comprobar los Enterprise JavaBeans (EJB) junto con las aplicaciones web. JBuilder proporciona funciones que facilitan la comprobación de Cactus.

- Asistente para la configuración de Cactus - Configura el proyecto para que utilice Cactus y se pueden ejecutar los tests Cactus en el IDE de JBuilder.
- Asistente para clientes de prueba EJB - Ayuda a crear el cliente de prueba Cactus para sus EJB.

El objetivo principal de Cactus de JBuilder es facilitar la comprobación de EJB con Cactus. La comprobación de los EJB con Cactus se describe con más detalle en "Ejecución y comprobación de enterprise beans" en *Desarrollo de aplicaciones con Enterprise JavaBeans*.

También puede utilizar Cactus para comprobar otros tipos de código Java en el servidor. Incluso si su objetivo no es comprobar sus EJB, puede utilizar el Asistente para la configuración de Cactus para configurar el proyecto para la comprobación de Cactus y facilitar la distribución adecuada de los archivos necesarios.

Asistente para la configuración de Cactus

El Asistente para la configuración de Cactus configura el proyecto para que pueda utilizar Cactus. Esto hace que sea posible ejecutar pruebas con Cactus desde el IDE de JBuilder. El asistente se abre mediante la opción de menú Asistentes|Configuración de Cactus.

Para configurar su proyecto para Cactus:

- 1 Seleccione Asistentes|Configuración de Cactus. Se abre el Asistente para configuración de Cactus.
- 2 Seleccione la aplicación Web a la que el asistente capacitará para realizar pruebas con Cactus. Se puede utilizar un módulo web ya creado, o, bien, pulsar el botón Nuevo para abrir el Asistente para módulos web y crear un módulo web.
- 3 Seleccione las opciones de registro para los archivos de registro de Cactus. Especifique las ubicaciones para los archivos de registro de cliente y servidor de Cactus, o bien, desactive la opción Activar registro si no desea utilizar archivos de registro.
- 4 Pulse Siguiente.
- 5 Seleccione los recopilatorios que se van a distribuir en el servidor y redistribuir antes de cada prueba. Así, se mantienen sincronizados los recopilatorios con el proyecto. Si cualquiera de los recopilatorios se muestra en rojo con un signo de exclamación antes del nombre, significa que el archivo físico ya no existe. Esto se debe probablemente a que el recopilatorio aún no se ha generado. No hay ningún problema en seleccionar alguno de estos recopilatorios, siempre que recuerde generar el recopilatorio antes de ejecutar las pruebas con Cactus.
- 6 Seleccione una configuración de ejecución de tipo Servidor. Puede crear una mediante el botón Nuevo.
- 7 Seleccione una configuración de ejecución de tipo Test. Puede crear una mediante el botón Nuevo.
- 8 Pulse el botón Finalizar. El proyecto ha quedado configurado para su uso con Cactus.

Consulte

- "Configuración del proyecto para probar un EJB con Cactus" en *Desarrollo de aplicaciones con Enterprise JavaBeans*

Creación de un test Cactus para los Enterprise JavaBean

Puede que desee utilizar Cactus para comprobar sus Enterprise JavaBeans (EJB). JBuilder incorpora el Asistente para clientes de prueba EJB, que puede generar tres tipos diferentes de clientes de prueba EJB. Uno de ellos es un cliente de prueba Cactus. Para abrir el Asistente para clientes de prueba EJB:

- 1 Seleccione Archivo|Nuevo.
- 2 Seleccione la ficha Enterprise/EJB de la galería de objetos.
- 3 Seleccione Cliente de prueba EJB y pulse Aceptar.

Nota El Asistente para clientes de prueba EJB no está disponible si el proyecto no se ha configurado correctamente para utilizar un servidor compatible con los servicios EJB.

La comprobación de EJB está fuera del ámbito de este capítulo. Este tema se trata en "Ejecución y comprobación de enterprise beans", en *Desarrollo de aplicaciones con Enterprise JavaBeans*.

Consulte

- "Ejecución y prueba de un bean enterprise" en *Desarrollo de aplicaciones con Enterprise JavaBeans*
- "Configuración del servidor de aplicaciones de destino" en *Desarrollo de aplicaciones para servidores J2EE*

Ejecución de test Cactus

La ejecución de test Cactus es más complicada que la ejecución de otro tipo de tests de módulos, ya que es necesario comprobar que cuenta con un servidor configurado correctamente, los descriptores de distribución adecuados y las configuraciones de ejecución Test y Servidor apropiadas. Además, la diferencia principal entre la ejecución de test Cactus y otro tipo de tests JUnit es que en el caso de un test Cactus, es necesario que inicie primero el servidor.

Una vez que la configuración es correcta y ya se ha iniciado el servidor, la ejecución de tests Cactus dentro del IDE de JBuilder es parecida a la ejecución de otros tests JUnit. Si su proyecto está configurado correctamente, todo lo que necesita para ejecutar los tests Cactus es:

- 1 Inicie el servidor con la configuración de ejecución Servidor.
- 2 Haga clic con el botón derecho del ratón sobre el archivo del test Cactus del panel de proyecto.
- 3 En el menú contextual, seleccione Ejecutar test utilizando <configuración de prueba>. El test se ejecuta en el ejecutor de tests especificado en la configuración de ejecución Test.

Todo lo relacionado con la configuración del servidor y los descriptores de distribución necesarios están fuera del ámbito de este capítulo. Esos temas

se tratan con más detalle en *Desarrollo de aplicaciones con Enterprise JavaBeans* y *Guía del desarrollador de aplicaciones web*.

Consulte

- "Ejecución y prueba de un bean enterprise" en *Desarrollo de aplicaciones con Enterprise JavaBeans*
- "Configuración del servidor de aplicaciones de destino" en *Desarrollo de aplicaciones para servidores J2EE*
- "Las aplicaciones web y los archivos WAR" en *Guía del desarrollador de aplicaciones web*
- ["Ejecución de tests" en la página 14-14](#)

Ejecución de tests

Existen tres ejecutores de tests distintos destinados a la ejecución de tests. El ejecutor de tests por defecto de JBuilder se llama JBuilderTestRunner. Si lo prefiere, puede utilizar los ejecutores de tests de JUnit, TextUI y SwingUI. El ejecutor de test que desee utilizar se especifica en la configuración de ejecución tipo Test. Para seleccionar un ejecutor de tests:

- 1 Elija Proyecto/Propiedades de proyecto en el menú.
- 2 Seleccione la ficha Ejecutar.
- 3 Seleccione una configuración de ejecución de tipo Test existente y pulse Modificar, o pulse Nuevo si todavía no existe ninguna configuración de ejecución Test. Se muestra el cuadro de diálogo Configuración de ejecución.
- 4 Escriba el nombre de la configuración de ejecución, si es necesario.
- 5 Seleccione la ficha Ejecutar del cuadro de diálogo Configuración de ejecución.
- 6 Asigne el valor Test al tipo de configuración de ejecución.
- 7 Seleccione un elemento de la lista desplegable Ejecutores de tests.
- 8 Pulse de nuevo Aceptar para cerrar el cuadro de diálogo Configuración de ejecución.
- 9 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

En los apartados siguientes se describe la ejecución de tests con los distintos ejecutores de tests disponibles:

Consulte

- ["Definición de las configuraciones de ejecución" en la página 7-8](#)

JBTestRunner

JBTestRunner proporciona una combinación de salida de texto e indicaciones por interfaz gráfica para informar sobre el estado de los tests. JBTestRunner muestra la jerarquía actual de los tests y los conjuntos, así como de los métodos de test que contienen. Para desplazarse a un método de test basta con hacer clic sobre él en el árbol. El cursor del editor se coloca en el método de test. JBTestRunner es el ejecutor de tests por defecto de JBuilder.

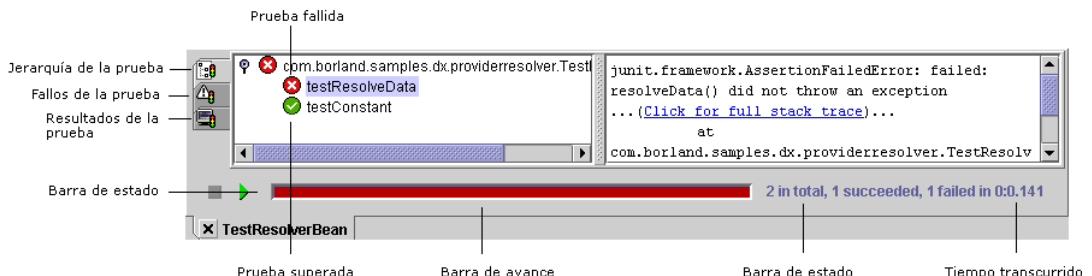
Para ejecutar un test, haga clic con el botón derecho del ratón en un test o un conjunto, en el panel del proyecto, y elija Ejecutar test en el menú contextual. Si no se ha cambiado el ejecutor de tests por defecto en el cuadro de diálogo Propiedades de proyecto, se utiliza JBTestRunner. Si ha cambiado el ejecutor de tests y desea volver a JBTestRunner, siga las instrucciones para la selección de ejecutores de tests que se facilitan en el apartado anterior.

Cuando se ejecutan pruebas, el resultado se muestra en la ficha JBTestRunner del panel de mensajes. En esta ficha hay tres vistas: Fallos del test, Jerarquía del test y Resultados del test.

Cuando se ejecutan los tests, JBTestRunner muestra una barra de progreso que indica el porcentaje de tests realizados. Esta barra es de color verde excepto si falla algún test, en cuyo caso adquiere el color rojo. JBTestRunner también muestra en el árbol de jerarquía de comparación iconos con una marca de verificación verde cuando el resultado es positivo, e iconos con una X roja cuando se presentan fallos o errores. Si ocurre un fallo o un error, JBTestRunner muestra una pila en la parte derecha de la ficha Fallos del test o Jerarquía del test, si el nodo de este fallo o error se encuentra seleccionado a la izquierda. Si se pulsa una línea de la pila, el punto en el que ha fallado una orden se resalta en el editor.

A medida que se ejecutan los tests, la barra de estado de JBTestRunner indica el número de tests efectuados, así como el número de éxitos, fallos y errores. También muestra el tiempo transcurrido desde el inicio de las pruebas, incluida la carga. Este tiempo se actualiza cada vez que se completa un test.

Si se hace clic en un nodo del árbol Fallos del test o Jerarquía del test, aparece un menú contextual que contiene las opciones Ejecutar selección y Depurar selección. Estas opciones permiten ejecutar y depurar tests específicos cuando se investiga un fallo.



Jerarquía del test



La vista Jerarquía del test aparece por defecto, excepto si el test ha fallado. Esta vista muestra un árbol con los tests, así como los conjuntos y los métodos de test. Junto a los nodos del árbol hay iconos que indican el estado del test correspondiente. Una marca de verificación verde indica que el test ha sido correcto. Una X de color rojo indica que el test ha fallado. Esta vista se actualiza de forma dinámica a medida que se ejecutan los tests. Cuando se pulsa un nodo de este árbol, su resultado aparece en el panel derecho de la vista de mensajes, y en el editor se resalta la línea de código que ha provocado el fallo, en caso de un test con fallos, o la primera línea, en caso de test correctos.

Fallos del test



La vista Fallos del test aparece cuando se pulsa la pestaña central de la parte izquierda de la ficha JUnitRunner. Se muestra por defecto la vista Fallos del test si el test falla. En el panel de la izquierda aparece una línea por cada test con fallos. Cuando se pulsa la línea correspondiente a un fallo, éste se resalta en el editor, y aparece más información sobre él en el panel de la derecha. Si no ha fallado ningún test, esta vista estará vacía.

Resultados del test



La vista Resultados del test aparece cuando se pulsa la pestaña inferior de la parte izquierda de la ficha JUnitRunner. Esta vista muestra la salida generada por el test, que incluye las excepciones.

JUnit TextUI

TextUI de JUnit es un ejecutor de tests sencillo que presenta la salida en texto. JUnit se ha integrado en JBuilder de forma que cuando se realizan tests con TextUI por medio del IDE de JBuilder, basta con pulsar una línea de salida que indique el fallo de un test, en la vista de mensajes, para que se abra el editor de JBuilder con la línea que ha provocado el fallo resaltada.

JUnit SwingUI

SwingUI de JUnit es un ejecutor de tests que proporciona una interfaz gráfica en la que se indica el estado de los tests y mensajes de texto en los que se señalan los fallos. Aunque es posible ejecutar tests con SwingUI desde el IDE de JBuilder, no es posible pulsar una línea de texto que indique un fallo y pasar directamente a esta línea en el editor, como ocurre con JUnitRunner y con JUnit TextUI. La ventaja de SwingUI consiste en que se puede revisar la jerarquía del test y volver a ejecutar los métodos uno a uno.

Configuraciones de ejecución para pruebas

Las configuraciones de ejecución para pruebas contienen los parámetros de MV, si procede, el ejecutor de tests que se va a utilizar y, de forma optativa, la clase, el paquete y el grupo filtrado de tests que se va a ejecutar. Para definir las propiedades de una configuración de ejecución para ejecutar tests, dirijase a `ProyectoPropiedades` de `proyectoEjecutar`, seleccione la configuración de ejecución que desee, pulse el botón `Nuevo`, `Copiar` o `Modificar`, seleccione la ficha `Ejecutar` del cuadro de diálogo `Configuraciones de ejecución` y asigne `Test` a `Tipo`. Aquí se pueden establecer los parámetros de máquina virtual que se utilizan para la ejecución de los tests, así como seleccionar un ejecutor de tests. También se puede indicar la clase o el paquete que contiene los tests. Si se elige el botón circular `Paquete`, la función `Recopilador` de tests de `JUnit` se habilita y permite filtrar los tests del paquete. Se pueden definir configuraciones de ejecución adicionales en la ficha `Ejecutar` del cuadro de diálogo `Propiedades de proyecto` o en `Ejecutar! Configuraciones`. Los Asistentes para tests y conjuntos de tests también permiten definir configuraciones de ejecución.

Sugerencia También es posible ejecutar tests con la configuración para la ejecución de aplicaciones; para ello, llame al método `main()` del ejecutor de tests `TextUI`. Esto puede resultar útil si se desea escribir un script que llame al ejecutor de tests con argumentos de línea de comandos.

Consulte

- “Definición de las configuraciones de ejecución” en la página 7-8

Definición del filtro de seguimiento de la pila de tests

Los filtros de seguimiento de pila de tests permiten especificar los paquetes y clases que se deben excluir de los seguimientos de pila al ejecutar los tests de módulos mediante `JBTestRunner`. Las líneas del seguimiento de la pila de los paquetes y clases excluidos no aparecen. Esto le permite concentrarse en la información de seguimiento de la pila que le resulte útil.

Para especificar un filtro de seguimiento de la pila de la unidad en test:

- 1 Seleccione `ProyectoPropiedades` de `proyectoGeneral` `Filtro` de comprobación del módulo.
- 2 Utilice los botones `Añadir` y `Eliminar` para especificar los paquetes y las clases que se van a excluir.
- 3 Pulse `Aceptar`.

El filtro de seguimiento de la pila de la unidad en test excluye por defecto los siguientes paquetes y clases:

- `junit.framework.*`
- `java.lang.reflect.Method`
- `com.borland.jbuilder.unittest.JBTestRunner`

- `sun.reflect.NativeMethodAccessorImpl`
- `sun.reflect.DelegatingMethodAccessorImpl`

Utilice los botones **Añadir** y **Eliminar** de la ficha **General** Filtro de comprobación del módulo del cuadro de diálogo **Propiedades de proyecto** para modificar esta lista.

Depuración de tests

La depuración de tests de módulos es parecida a la depuración de otro código mediante el depurador de JBuilder. La única diferencia radica en que al depurar un test, aparecen las pestañas **Jerarquía del test** y **Fallos del test** de JBuilder además de la interfaz del usuario normal del depurador. Para depurar un test, haga clic con el botón derecho del ratón en cualquier test del panel de proyecto, y elija **Depurar test** en el menú contextual.

Sugerencia Cuando se ejecutan tests con JBuilder o TextUI y se pulsa un error del Resultados del test, la línea de código que ha provocado el fallo se muestra resaltada en el editor. Si se vuelve a hacer clic en el margen izquierdo de la ventana del editor, junto a la línea de código resaltada, se coloca un punto de interrupción en ella. De esta forma se puede realizar la depuración cómodamente hasta llegar al punto de interrupción.

Sugerencia Es posible definir un punto de interrupción en la excepción lanzada en el fallo de un test. Para ello, elija **Ejecutar** | **Añadir punto de interrupción** | **Añadir punto de interrupción por excepción** y escriba `junit.framework.AssertionFailedError` en Nombre de la clase.

Sugerencia Durante la ejecución de tests mediante JBuilder, cuando se hace clic con el botón derecho del ratón en un nodo de test de la vista **Jerarquía del test** o en un nodo de test fallido de la vista **Fallos del test** se abre un menú contextual que permite depurar el test correspondiente.

Consulte

- [“JBuilder” en la página 14-15](#)
- [Capítulo 8, “Depuración de programas en Java”](#)



Capítulo 15

Creación de Javadoc a partir de archivos fuente

Javadoc es una herramienta creada por Sun Microsystems principalmente para generar documentación en archivos con formato HTML. La documentación en HTML se crea a partir de comentarios a nivel de clases y método que se introducen en los archivos fuente. Los comentarios deben tener un formato acorde con la norma Javadoc. Si desea información más completa acerca de la herramienta Javadoc, puede dirigirse a la página principal de esta herramienta en la página web de Sun, <http://www.java.sun.com/j2se/javadoc/>.

JBUILDER incorpora varias funciones para la creación de Javadoc. El asistente crea un nodo de documentación con propiedades para la ejecución del Javadoc. Este nodo aparece en el panel del proyecto. Se puede crear Javadoc siempre que se cree un proyecto, utilizando las propiedades adecuadas. (Estas funciones pertenecen a las ediciones Developer y Enterprise de JBUILDER).

JBUILDER también incluye las siguientes funciones relacionadas con Javadoc:

- Generación automática de comentarios y parámetros basados en la firma de la clase, la interfaz, el método, el campo o el constructor
- JavadocInsight para introducir etiquetas Javadoc
- Capacidad de añadir y ver automáticamente etiquetas `@todo`
- Capacidad de informar acerca de y corregir conflictos de comentarios Javadoc
- Recopilación de documentación con el Creador de compiladores
- Generación de Javadoc sobre la marcha

- Posibilidad de crear etiquetas personalizadas (ediciones Developer y Enterprise de JBuilder)
- Un visualizador para los documentos Javadoc que se hayan generado (ediciones Developer y Enterprise de JBuilder)

Adición de comentarios Javadoc a los archivos fuente

Los comentarios Javadoc incluyen texto descriptivo y etiquetas Javadoc. Se pueden añadir comentarios Javadoc de clase e interfaz y así como comentarios de método, constructor y campo. La herramienta Javadoc extrae los comentarios Javadoc de los archivos fuente y los coloca en los archivos de documentación con formato HTML.

Un comentario Javadoc empieza con un símbolo de comienzo de comentario (`/**`) y termina con un símbolo de final de comentario (`*/`). Cada comentario consta de una descripción seguida de una o más etiquetas. Si lo desea, puede utilizar el formato HTML en los comentarios Javadoc. Al escribir sus comentarios, siga los siguientes pasos:

- Alinee el símbolo de comienzo de comentario (`/**`) de tal manera que coincida con el código al que hace referencia.
- Comience las siguientes líneas del comentario con `*` (asterisco). Alinee también estas líneas.
- Escriba el texto con la descripción en la línea siguiente al símbolo de comienzo de comentario (`/**`).
- Introduzca un espacio en blanco delante del texto o de la etiqueta.
- Introduzca una línea de comentario en blanco entre el texto y la lista de etiquetas.

A continuación se muestra un ejemplo de comentario Javadoc para un método:

```
/**
 * Sets this check box's label to the string argument.
 *
 * @param label a string to set as the new label, or null for no label.
 */
```

En el archivo HTML generado, este comentario aparece de la siguiente manera:

Sets this check box's label to the string argument.

Parameters:

label - a string to set as the new label, or null for no label.

Observe cómo Javadoc convierte la etiqueta `@param` en un título. También añade el guión que separa el nombre del parámetro de su descripción. Además, muestra el nombre del parámetro utilizando una fuente para códigos.

Al escribir la descripción del comentario, escriba la primera frase a modo de resumen. Debería ser una descripción concisa y completa del elemento. La herramienta Javadoc copia la primera frase del comentario en la clase, interfaz o tabla resumen miembro.

Nota La herramienta Javadoc hereda los comentarios para métodos que implementan o que redefinen otros métodos. En estos casos, no es necesario duplicar estos comentarios.

Para obtener más información sobre la creación de comentarios Javadoc, consulte “How to Write Doc Comments for the Javadoc Tool” (en inglés) en <http://www.java.sun.com/j2se/javadoc/writingdoccomments/index.html>.

Consulte

- “Generación automática de comentarios y etiquetas Javadoc” en la [página 15-7](#)
- “Javadoc Tags” (equipos Windows) en <http://java.sun.com/j2se/1.4/docs/tooldocs/win32/javadoc.html#javadoctags>
- “Javadoc Tags” (equipos Solaris) en <http://java.sun.com/j2se/1.4/docs/tooldocs/solaris/javadoc.html#javadoctags>

Colocación de los comentarios Javadoc

Puede añadir comentarios Javadoc para clases, interfaces, métodos, campos y constructores. Coloque los comentarios de clase e interfaz en la parte superior del archivo, después de las sentencias `import` y justo antes de la sentencia de declaración `class` o `interface`. Por ejemplo, el comentario de clase para `com.borland.internetbeans.PageProducer.java` es:

```
/**
 * Generates markup text from a template file, replacing
 * identified spans with dynamic content from Ix
 * components.
 * @author Borland Software Corporation
 * @versión 1.0
 */
public class PageProducer implements Binder, Renderable, Cloneable,
    Serializable
...
}
```

Se pueden personalizar comentarios a nivel de clase por defecto en la tabla Archivos Javadoc de clases de la ficha Propiedades de proyecto/General. Los comentarios que introduzca aquí se añaden a todas las clases o interfaces que cree con un asistente de JBuilder.

Los comentarios para métodos, campos y constructores se colocan inmediatamente antes de la firma del método en el archivo fuente de clases. A continuación se muestra un ejemplo de comentario Javadoc para un método:

Ejemplo de comentarios Javadoc de métodos

```
/**
 * Subtracts Value One and Value Two and displays result.
 *
 * @param valueOneDouble The minuend.
 * @param valueTwoDouble The subtrahend.
 */
public void subtractValues(Double valueOneDouble, Double valueTwoDouble) {
    double valueOneDoubleResult = valueOneDouble.doubleValue();
    double valueTwoDoubleResult = valueTwoDouble.doubleValue();
    subtractResult = (valueOneDoubleResult - valueTwoDoubleResult);
    subtractStringResult = Double.toString(subtractResult);
    subtractResultDisplay.setText(subtractStringResult);
}
```

Etiquetas Javadoc

En los comentarios Javadoc se pueden utilizar las siguientes etiquetas. Algunas de ellas, como `@param` y `@return` se incluyen automáticamente en la ejecución de Javadoc que se inicia en JBuilder. El asistente utilizado para crear Javadoc permite seleccionar otras etiquetas en la ficha Indique opciones de línea de comandos para el doclet . También puede introducir otras etiquetas en el campo Opciones adicionales de la misma ficha, con lo que se obliga al asistente a incluir esas etiquetas de comentarios en los archivos creados.

En la siguiente tabla se recogen y se describen las etiquetas Javadoc. Indica el doclet de Javadoc para el que se van a procesar las etiquetas. Por ejemplo, no todas las etiquetas las va a procesar el doclet JDK 1.1. También indica a qué parte del archivo de clases se puede aplicar la etiqueta. Para obtener más información sobre las etiquetas individuales, consulte .

- **Usuarios de Windows: "Javadoc Tags"** en <http://java.sun.com/j2se/1.4/docs/tooldocs/win32/javadoc.html#javadocTags>

- **Usuarios de Solaris: “Javadoc Tags”** en <http://java.sun.com/j2se/1.4/docs/tooldocs/solaris/javadoc.html#javadoctags>

Tabla 15.1 Etiquetas Javadoc

Etiqueta	Descripción	Doclet JDK 1.1:	Std doclet	Tipo de etiqueta
@authorcode <i>nombre</i>	Añade a los documentos creados una entrada Author con el <i>nombre</i> especificado si la opción @author se ha seleccionado en la ficha Indique opciones de línea de comandos para el doclet del asistente utilizado para crear Javadoc.	X	X	Aspectos generales, paquete, clase, interfaz
{@docRoot}	Es la vía de acceso relativa desde cualquier ficha al directorio raíz o destino de los documentos generados. Este asistente no define automáticamente esta opción; es necesario introducirla en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos.		X	Aspectos generales, paquete, clase, interfaz, campo
@version <i>número-de-versión</i>	Añade a los documentos creados un subtítulo Versión con el <i>número de versión</i> especificado si la opción @version se utiliza en la ficha Indique opciones de línea de comandos para el doclet del asistente.	X	X	Aspectos generales, paquete, clase, interfaz
@param <i>descripción-del-nombre-del-parámetro</i>	Añade un parámetro al subtítulo Parameters. Incluido automáticamente en documentos generados.	X	X	métodos:Constructor; Constructor (método)
@return <i>descripción</i>	Añade un subtítulo Returns con el texto de la <i>descripción</i> . Incluido automáticamente en documentos generados.	X	X	métodos:Constructor; Constructor (método)
@deprecated <i>texto-desaconsejado</i>	Añade un comentario que indica que se ha desaconsejado la API y no debe volver a utilizarse, aunque todavía funcione. Esta opción se puede configurar en la ficha Indique opciones de línea de comandos para el doclet del asistente.	X	X	Paquete, clase, interfaz, campo, constructor, método
@exception <i>descripción del nombre-de-clase</i>	Es un sinónimo de @throws. Incluido automáticamente en documentos generados.	X	X	métodos:Constructor; Constructor (método)

Tabla 15.1 Etiquetas Javadoc (continuación)

Etiqueta	Descripción	Doclet JDK 1.1:	Std doclet	Tipo de etiqueta
@throws <i>descripción del nombre-de-clase</i>	Es un sinónimo de @exception. Añade un subtítulo Throws a los documentos generados con la excepción <i>nombre de clase</i> que se puede lanzar. Incluido automáticamente en documentos generados.		X	métodos:Constructor; Constructor (método)
@see <i>referencia</i>	Añade un subtítulo See Also a los documentos generados. Incluido automáticamente en documentos generados.	X	X	Aspectos generales, paquete, clase, interfaz, campo, constructor, método
@since <i>texto-since</i>	Añade un título Since con el <i>texto</i> especificado a los documentos generados. Este asistente no define automáticamente esta opción; es necesario introducirla en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos.	X	X	Aspectos generales, paquete, clase, interfaz, campo, constructor, método
@serial <i>descripción-del-campo</i>	Describe un campo serializable por defecto. Este asistente no define automáticamente esta opción; es necesario introducirla en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos.		X	Campo
@serialField <i>nombre-del-campo</i> <i>tipo-de-campo</i> <i>descripción-del-campo</i>	Documenta un componente <code>ObjectStreamField</code> de un miembro de la clase serializable <code>serialPersistentFields</code> . Este asistente no define automáticamente esta opción; es necesario introducirla en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos.		X	Campo
@serialData <i>descripción-de-datos</i>	Documenta el tipo y el orden de los datos de forma serializada. Este asistente no define automáticamente esta opción; es necesario introducirla en el campo Opciones adicionales de la ficha Especificar opciones doclet de línea de comandos.		X	métodos:Constructor; Constructor (método)
{@link} <i>paquete.clase#etiqueta del miembro</i>	Inserta un enlace en línea con la <i>etiqueta</i> como texto visible. Incluido automáticamente en documentos generados.		X	Aspectos generales, paquete, clase, interfaz, campo, constructor, método

Tabla 15.1 Etiquetas Javadoc (continuación)

Etiqueta	Descripción	Doclet JDK 1.1:	Std doclet	Tipo de etiqueta
<code>{@linkPlain <i>etiqueta</i> paquete.clase#miembro}</code>	Igual que <code>{@link}</code> , con la diferencia de que la etiqueta del enlace se muestra en texto sin formato y no en texto de código. Resulta de gran utilidad cuando la etiqueta está en texto sin formato.		X (desde la versión 1.4)	Aspectos generales, paquete, clase, interfaz, campo, constructor, método
<code>{@value}</code>	Cuando se usa en el comentario doc de un campo estático muestra el valor de la constante. Éstos son los valores que se muestran en la ficha Valores de constante de campo.		X (desde la versión 1.4)	Campo
<code>{@inheritDoc}</code>	Hereda (copia) la documentación de la clase heredable o la interfaz implementable más cercana en el comentario doc actual, en la ubicación de esta etiqueta. De este modo se pueden escribir comentarios más generales en los niveles superiores del árbol de herencias, así como escribir en torno al texto copiado.		X (desde la versión 1.4)	Método (en el bloque de descripción principal o en los argumentos de texto de las etiquetas <code>@return</code> , <code>@param</code> y <code>@throws</code>)

Generación automática de comentarios y etiquetas Javadoc

JBuilder puede generar automáticamente comentarios y etiquetas Javadoc, con un cuadro de diálogo o con una plantilla de código.

El cuadro de diálogo Javadoc en la generación y modificación de comentarios y etiquetas

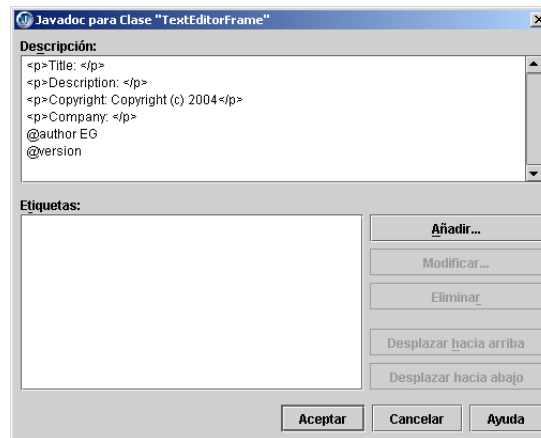
El cuadro de diálogo Javadoc permite introducir en el código el comentario Javadoc y las etiquetas necesarias. Este cuadro de diálogo genera automáticamente las etiquetas correctas por el orden adecuado y reduce la posibilidad de que ocurran conflictos Javadoc.

Nota Para modificar los comentarios Javadoc ya creados, seleccione el símbolo de código en el editor, haga clic con el botón derecho y seleccione Edición Javadoc. Una vez generado Javadoc, si a continuación se cambia o se perfecciona la firma del código, Javadoc no cambia automáticamente. Es necesario abrir de nuevo el cuadro de diálogo Modificar Javadoc o cambiarlo manualmente.

Para añadir comentarios y etiquetas mediante el cuadro de diálogo:

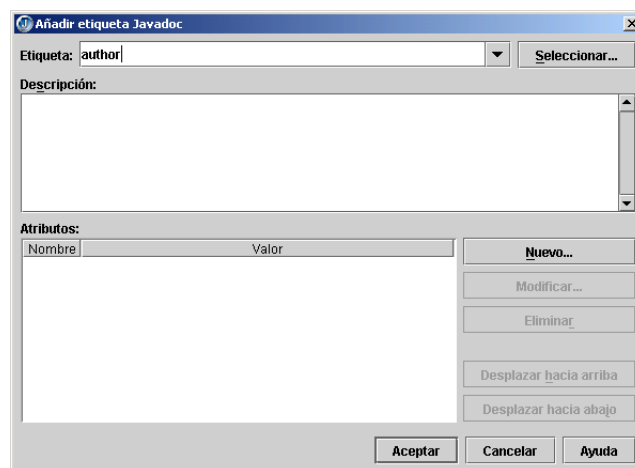
- 1 Coloque el cursor en una declaración de clase o firma de método en el editor o el panel de estructura. Haga clic con el botón derecho y seleccione

NuevolJavadoc. Aparece el cuadro de diálogo Javadoc para Clase, en el que se añaden y modifican comentarios y etiquetas:



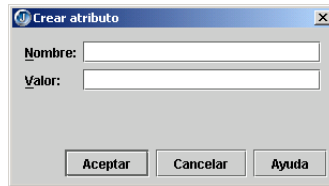
La descripción de Javadoc aparece en el cuadro de texto Descripción. En el caso de una clase, es la cabecera estándar Javadoc. En el caso de un método, es el nombre del método. Las etiquetas necesarias para el símbolo de código seleccionado se recogen en la lista Etiquetas, junto con su tipo Java.

- 2 Para añadir o modificar texto descriptivo para el símbolo de código seleccionado, escriba o modifique el código con formato HTML en el cuadro de texto Descripción.
- 3 Si desea añadir una etiqueta, pulse el botón Añadir. Se muestra el cuadro de diálogo Añadir etiqueta Javadoc.



- 4 Seleccione la etiqueta que desea añadir en la lista desplegable Etiqueta. Pulse el botón Seleccionar para escoger entre las etiquetas utilizadas previamente si hay demasiadas en la lista desplegable. Si la etiqueta todavía no se ha utilizado, sólo hay que escribirla en el campo Etiqueta.

- 5 Escriba texto descriptivo en formato HTML en el campo Descripción.
- 6 Para añadir atributos (por ejemplo, para Javadoc de estilo XDoclet y EJBGen), pulse el botón Nuevo. Se abre el cuadro de diálogo Crear atributo:



Escriba el nombre y el valor del atributo en los campos Nombre y Valor. Si desea obtener más información acerca de Javadoc de estilo XDoclet y EJBGen, consulte:

- “Using XDoclet” en <http://www.xdoclet.com/using.html>
- “EJBGen Reference” en http://edocs.bea.com/wls/docs81/ejb/EJBGen_reference.html

Haga clic en Aceptar tres veces para cerrar los cuadros de diálogo y crear los comentarios Javadoc.

Las plantillas de código en la generación de comentarios y etiquetas

JBuilder también puede generar automáticamente comentarios Javadoc desde una plantilla de código. En el caso de los comentarios de clases, coloque el cursor en una línea antes de la declaración de clase o interfaz. Escriba el símbolo de principio de comentario (/**) y pulse *Intro* para insertar en el código la siguiente plantilla:

```
/**
 * <p>Título: </p>
 * <p>Descripción: </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Empresa: </p>
 * @author not attributable
 * @versión 1.0
 */
```

Si el proyecto utiliza JDK 1.4 se pueden emplear las etiquetas siguientes, aunque no se generan automáticamente:

```
* @inheritDoc
* @value
* @linkPlain
```

Los campos se pueden rellenar y borrar según sea necesario.

Nota Para un proyecto nuevo, la plantilla (para clases e interfaces) se puede rellenar en la ficha Especificar configuración general del proyecto del Asistente para proyectos a la vez que se crea el proyecto. Estos valores se pueden cambiar en cualquier momento en la ficha Propiedades de proyecto General. El archivo `javadocClass.template` utiliza los valores como entrada en el directorio `.jbuilder`.

Si el cursor se encuentra delante de un método, campo o firma del constructor, al escribir `/**` se inserta la plantilla que se indica a continuación. Tenga en cuenta que solamente aparecen en la plantilla ampliada de comentario las etiquetas utilizadas en la firma.

```
/**
 *
 * @param
 * @throws
 * @returns
 */
```

JBUILDER completa la etiqueta rellenando el nombre del parámetro o la excepción. Por ejemplo, en el caso de la siguiente firma de método:

```
public void addValues(Double valueOneDouble, Double valueTwoDouble)
```

al escribir `/**` se crea el siguiente comentario Javadoc:

```
/**
 *
 * @param valueOneDouble Double
 * @param valueTwoDouble Double
 */
```

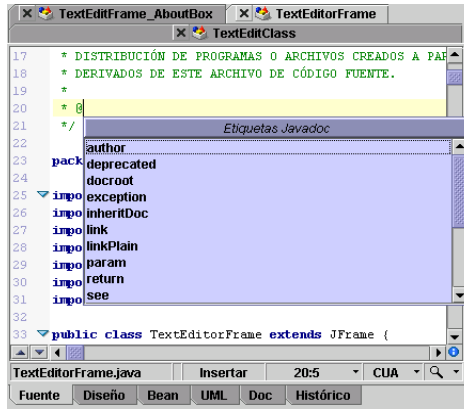
Es posible personalizar el color de toda la plantilla de comentarios Javadoc, de forma que se muestre en un color distinto al de otras etiquetas de comentario. Para ello, diríjase a Herramientas|Preferencias|Editor|Color. Elija el elemento de pantalla de Javadoc y, a continuación, seleccione el color y otros atributos que desee aplicar al comentario. La selección se muestra en la parte inferior del cuadro de diálogo. Si desea obtener más información, consulte el tema de la Ayuda online, ficha Color (Herramientas|Preferencias).

JavadocInsight

JavadocInsight resulta de gran utilidad a la hora de introducir etiquetas Javadoc estándar o personalizadas en los comentarios Javadoc. La ventana JavadocInsight recoge las etiquetas Javadoc y le permite elegir la que desee. La utilización de JavadocInsight evita los errores de formato en los comentarios Javadoc. Para abrir JavadocInsight:

- Escriba `@` en un bloque de comentarios Javadoc.
- Abra MemberInsight en un bloque de comentarios Javadoc (pulse *Ctrl+Espacio*).
- Asigne una combinación de teclas específica en la configuración de teclado actual.

Figura 15.1 Ventana JavadocInsight

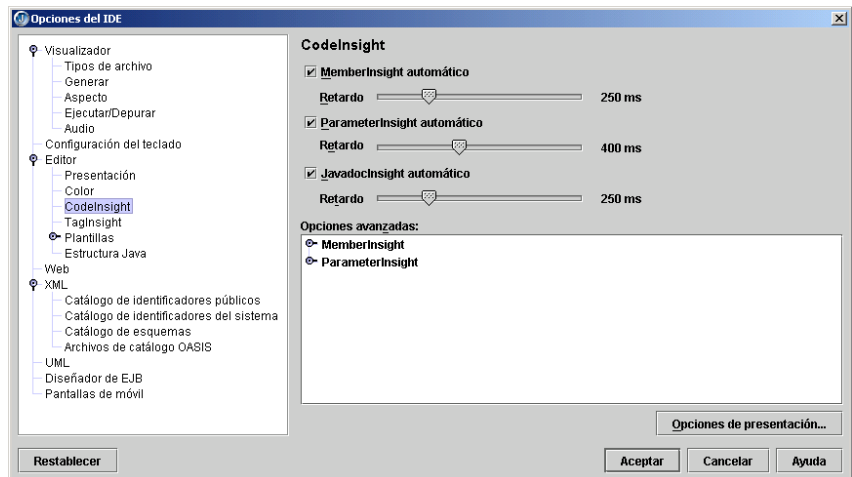


La ventana JavadocInsight permite personalizar:

- La duración de la pausa antes de que se abra la ventana
- La fuente y el tamaño de letra de las etiquetas Javadoc
- El color con el que se muestra cada etiqueta Javadoc, incluidas las personalizadas

Para personalizar la duración de la pausa antes de que se abra la ventana:

- 1 Seleccione Herramientas|Preferencias|Editor|CodeInsight. Aparece la ficha CodeInsight:



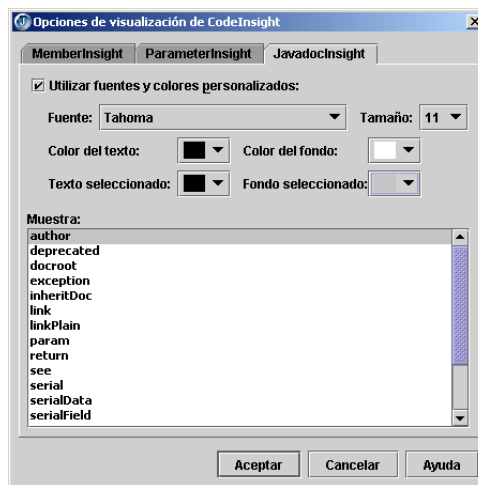
- 2 Ajuste la barra deslizante JavadocInsight automático en el intervalo de espera que desee establecer para la presentación de la ventana emergente de JavadocInsight.
- 3 Haga clic en Aceptar para cerrar el cuadro de diálogo.

Para desactivar JavadocInsight:

- 1 Seleccione Herramientas|Preferencias|Editor|CodeInsight.
- 2 Desactive la opción JavadocInsight automático.
- 3 Haga clic en Aceptar para cerrar el cuadro de diálogo.

Para personalizar la fuente, el tamaño de letra y el color de una etiquetas Javadoc (o de todas ellas):

- 1 Seleccione Herramientas|Preferencias|Editor|CodeInsight.
- 2 Pulse el botón Opciones de presentación, situado en la parte inferior del cuadro de diálogo, para abrir el cuadro de diálogo Opciones de visualización de CodeInsight. Seleccione la pestaña JavadocInsight para abrir la ficha JavadocInsight.



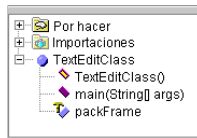
- 3 Para personalizar la fuente y el color de las etiquetas Javadoc que se muestran en la ventana JavadocInsight, active la opción Utilizar fuentes y colores personalizados. La configuración se muestra en la lista Ejemplo, situada en la parte inferior del cuadro de diálogo. Cuando se activa esta opción se habilitan otras opciones del cuadro de diálogo. Si esta opción se encuentra desactivada, se utiliza la selección por defecto.
- 4 Si desea definir una fuente para todas las etiquetas, elija el tipo de letra en la lista desplegable Fuente, y el tamaño en la lista desplegable Tamaño de fuente. Esta selección se aplica a todas las etiquetas que aparecen en la ventana JavadocInsight; la fuente y el tamaño no se pueden definir por separado.
- 5 Para aplicar un color al texto de todas las etiquetas, abra la lista desplegable Primer plano y elija el que desee.
- 6 Para aplicar un color de fondo a todas las etiquetas, abra la lista desplegable Fondo y elija el que desee.

- 7 Para aplicar un color de texto personalizado a una sola etiqueta, selecciónela en la lista Ejemplo y elija el que desee en el cuadro Texto seleccionado.
- 8 Para aplicar un color de fondo personalizado a una sola etiqueta, selecciónela en la lista Ejemplo y elija el que desee en el cuadro Fondo seleccionado.
- 9 Haga clic en Aceptar para cerrar el cuadro de diálogo.
- 10 Haga clic en Aceptar para cerrar el cuadro de diálogo Herramientas|Preferencias.

Etiquetas @todo Javadoc

Las etiquetas @todo de Javadoc resultan útiles para recordar que es necesario hacer algo en una zona del código. Estas etiquetas se colocan dentro de los comentarios Javadoc. Estas etiquetas @todo aparecen en el panel de estructura de JBuilder en la carpeta Por Hacer.

Figura 15.2 Carpeta Por hacer del panel de estructura



Para añadir etiquetas @todo al código, abra JavadocInsight y seleccione todo en la lista.

Algunos asistentes de JBuilder generan etiquetas @todo para recordar que es necesario añadir código al stub generado.

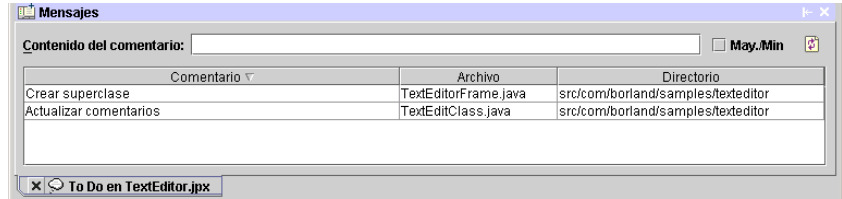
Nota Se puede personalizar la presentación de las etiquetas @todo de la documentación generada en la ficha Herramientas|Preferencias|Editor|Plantillas|Java.

Visualización de etiquetas @todo

Se puede utilizar el comando de menú contextual Ver|Todos del panel del proyecto para ver todas las etiquetas @todo de todos los archivos del proyecto o grupo de proyectos. Para cada etiqueta @todo aparece el comentario, el nombre de archivo y la ubicación del directorio. La presentación se puede filtrar por comentario, y se puede ordenar por nombre de archivo o por directorio. Se puede hacer clic en un comentario en la presentación para dirigirse directamente a ese comentario en el código fuente.

Para ver las etiquetas @todo en su proyecto o grupo de proyectos:

- 1 Haga clic con el botón derecho en el proyecto o grupo de proyectos en el panel del proyecto y seleccione VeriTodos. Todas las etiquetas @todo aparecen en la pestaña Todo del panel de mensajes:



- 2 Para filtrar las etiquetas @todo, escriba el comentario que desea filtrar en el campo Contenido del comentario de la parte superior del panel. Si el filtro distingue entre mayúsculas y minúsculas, active la opción May./Min. Pulse Intro. Sólo aquellos comentarios de etiquetas @todo que cumplen con los criterios del filtro se recogen en la vista.
- 3 Para ordenar las etiquetas @todo, haga clic en la cabecera de la columna por la que desea ordenarlas. Por ejemplo, para ordenar por nombre de archivo, haga clic en la palabra Archivo de la cabecera de la columna.
- 4 Para actualizar la presentación, pulse el botón Actualizar de la parte superior derecha del panel. Si se pulsa este botón aparecen los comentarios @todo que se acaban de añadir, se borran los eliminados y se actualizan los modificados.
- 5 Para dirigirse a un comentario en el código fuente, haga clic en el comentario en la lista @todo. (Si se han modificado los comentarios, haga clic primero en Actualizar).

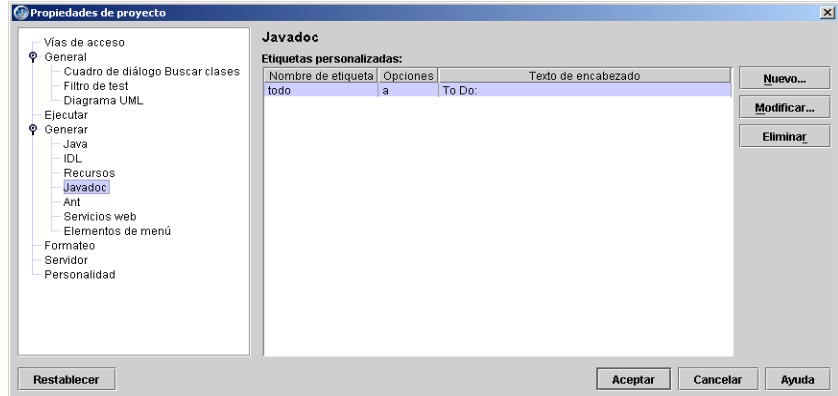
Creación de etiquetas Javadoc personalizadas

Es una función de
JBuilder Developer y
Enterprise.

La ficha Propiedades de proyecto|GenerarJavadoc permite añadir etiquetas personalizadas para Javadoc. Este cuadro de diálogo también se utiliza para añadir texto de cabecera a las etiquetas personalizadas.

Para añadir una etiqueta personalizada y el texto de encabezamiento:

Abra el cuadro de diálogo Propiedades de proyecto y seleccione Generar Javadoc. La ficha Javadoc tendrá este aspecto:



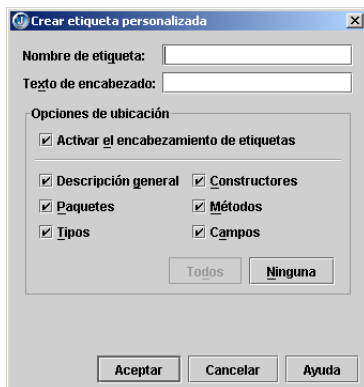
Nota

La etiqueta `todo` es la etiqueta establecida por defecto, aunque es posible personalizar el encabezamiento y la forma en la que se muestra. Aunque se desactive esta etiqueta en la documentación generada, JBuilder sigue mostrándola en el panel de estructura durante la modificación.

El campo Nombre de etiqueta muestra las etiquetas personalizadas que se han definido. El campo Opciones muestra los códigos de las opciones de la etiqueta, y el campo Texto de encabezamiento muestra el texto que aparece como encabezado en el archivo representado. Los códigos de las opciones se enumeran en la tabla siguiente:

Código	Descripción
X	No se ha establecido ninguna opción.
a	Se han establecido todas las opciones.
o	Se ha establecido la opción Descripción general, lo que significa que esta etiqueta se puede colocar en la sección Descripción general del archivo fuente.
p	Se ha establecido la opción Paquetes, lo que significa que esta etiqueta se puede colocar en la sección Paquetes del archivo fuente.
t	Se ha establecido la opción Tipos, lo que significa que esta etiqueta se puede colocar en la sección Tipos del archivo fuente.
.c	Se ha establecido la opción Constructores, lo que significa que esta etiqueta se puede colocar en la sección Constructores del archivo fuente.
m	Se ha establecido la opción Métodos, lo que significa que esta etiqueta se puede colocar en la sección Métodos del archivo fuente.
f	Se ha establecido la opción Campos, lo que significa que esta etiqueta se puede colocar en la sección Campos del archivo de código fuente.

- 1 Si desea añadir una etiqueta personalizada, pulse el botón Nueva. A continuación, se abre el cuadro de diálogo Crear etiqueta personalizada.



- 2 Escriba el nombre de la etiqueta en el campo Nombre de etiqueta y el texto de encabezamiento que se debe asociar a esta etiqueta en el campo Texto de encabezamiento. El texto correspondiente a los bloques de Javadoc asociados a esta etiqueta aparece bajo este encabezamiento. Por ejemplo, si se añade el nombre de etiqueta `test` y el texto de encabezamiento `Javadoc Test Tag`, y, a continuación, se utiliza la etiqueta personalizada `test` en el código, aparece la siguiente salida en el archivo HTML de la documentación Javadoc:

addValues

```
public void addValues(java.lang.Double valueOneDouble,
                     java.lang.Double valueTwoDouble)
```

Method that adds Value One and Value Two and displays result.

Parameters:

valueOneDouble - First value entered.

valueTwoDouble - Second value entered.

Javadoc Test Tag

Testtag

Texto de
encabezado

Nota Las etiquetas personalizadas se muestran en JavadocInsight.

- 3 Para activar el encabezamiento, seleccione la opción Activar el encabezamiento de etiquetas. De esta forma también se muestra el texto Javadoc asociado bajo el encabezamiento en el archivo Javadoc representado.
- 4 Elija en el archivo fuente los lugares en los que desee activar la etiqueta de encabezamiento. Los encabezamientos se pueden activar en todas las ubicaciones o sólo en las que se desee. Pulse Todos para activar todas las ubicaciones, y Ninguna para desactivarlas.

- 5 Haga clic en Aceptar para cerrar el cuadro de diálogo. La nueva etiqueta se muestra en la ficha Javadoc.
- 6 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto. Para modificar una etiqueta, selecciónela en la ficha GenerarJavadoc y pulse Edición. Realice los cambios que desee en el cuadro de diálogo Modificar etiqueta personalizada. Para eliminar una etiqueta, selecciónela en la ficha GenerarJavadoc y pulse Eliminar.

Conflictos en comentarios Javadoc

Los conflictos Javadoc tienen lugar cuando las etiquetas de un comentario Javadoc no coinciden con la firma del método o, bien, si no se ofrece ningún argumento en etiquetas como `@param`. Por ejemplo, si la firma del método contiene dos parámetros y el comentario solamente uno, se produce un conflicto.

En JBuilder, los conflictos Javadoc se registran en la parte superior del panel de estructura, en la carpeta `Conflictos Javadoc`. Amplíe la carpeta y haga clic en el conflicto si desea ir a la firma del método donde ha tenido lugar el conflicto.

Figura 15.3 Conflictos Javadoc en el panel de estructura



Nota No se informa de los conflictos Javadoc hasta que se hayan solventado todos los errores de sintaxis de la carpeta Errores.

Para corregir conflictos en comentarios Javadoc, haga clic con el botón derecho en el conflicto en el panel de estructura y seleccione Corregir conflictos Javadoc. Si falta alguna etiqueta o argumento, se añade automáticamente. (Esta función pertenece a las ediciones Developer y Enterprise de JBuilder).

Generación del nodo de documentación

Es una función de JBuilder Developer y Enterprise.

El Asistente para Javadoc de JBuilder genera un nodo de documentación en el panel de proyecto. En este nodo se almacenan las propiedades para la ejecución de Javadoc. Entre las propiedades se incluyen el formato de la documentación Javadoc, qué paquetes se documentan y qué archivos de salida se generan para esos paquetes. Si desea cambiar las propiedades Javadoc después de crear el nodo, pulse con el botón derecho del ratón y, a continuación, seleccione Propiedades.

Durante la creación del nodo, tiene la opción de crear también archivos Javadoc cada vez que se genera un proyecto. También puede crear Javadoc solamente cuando lo necesite si pulsa con el botón derecho del ratón sobre el nodo y selecciona Ejecutar Make.

Para abrir el Asistente para Javadoc, seleccione Archivo|Nuevo. En la ficha Generar de la galería de objetos, haga doble clic sobre el icono Javadoc. También puede seleccionar Asistentes|Javadoc.

El asistente consta de cuatro pasos. Las opciones del asistente incluyen:

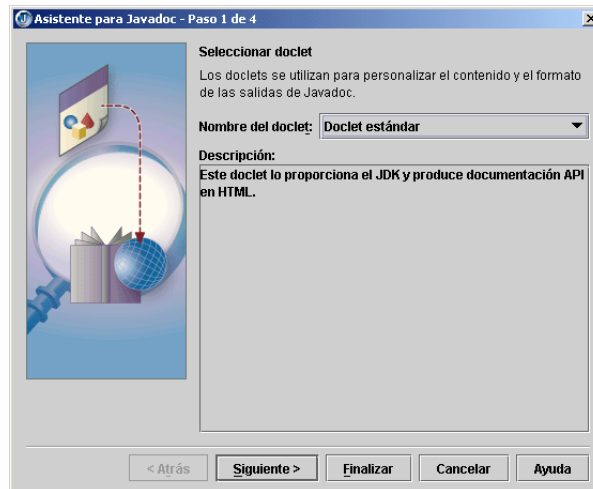
- El formato de los archivos generados por Javadoc
- El nombre del nodo de documentación que crea el asistente
- El directorio de salida
- Cuándo ejecutar Javadoc
- Los paquetes y el ámbito documentados
- Qué archivos de salida se generan y qué etiquetas se procesan

Selección del formato de la documentación

Es una función de
JBuilder Developer y
Enterprise.

En el primer paso del asistente se elige el formato de la documentación Javadoc. Estos archivos están controlados por un doclet, una clase Java que especifica los contenidos y el formato de los archivos de salida.

Figura 15.4 Paso 1 de selección de doclet



Para seleccionar el formato de los archivos de salida:

- 1 Si desea crear archivos de salida Javadoc con formato JDK 1.1, seleccione la opción Doclet JDK 1.1 de la lista desplegable Nombre del doclet. Este doclet crea la documentación con formato HTML, pero no incluye el nivel adicional de detalles que proporciona la opción Doclet estándar. Esta opción corresponde a la etiqueta Javadoc **-1,1**. Tenga en cuenta que este

doclet no se encuentra disponible cuando se utiliza con JDK 1.4. Si se selecciona este doclet y se utiliza JDK 1.4 en el proyecto, en la pestaña Generar del panel de mensajes aparece el siguiente mensaje cuando se intenta generar el Javadoc:

```
Doclet JDK 1.1: Imposible hallar el doclet
com.sun.tools.doclets.oneone.OneOne.
en la vía de acceso a clases. Elija un doclet disponible en el JDK.
```

- 2** Si desea crear archivos de salida con formato JDK 1.3 ó JDK 1.4, seleccione la opción Doclet estándar en la lista desplegable. Este doclet crea la documentación con formato HTML e incluye más características que la opción que permite crear archivos con formato JDK 1.1, entre las que se encuentran:

- Tablas de campos y métodos para una clase o interfaz
- Descripciones de nivel de paquetes
- Listas de campos y métodos heredados
- Listas de clases internas
- Un índice por letra
- Comentarios sobre la etiqueta @use
- Barra de desplazamiento ampliable

Si desea un ejemplo de archivos de salida estándar, seleccione Ayuda Referencia de Java en la barra del menú principal de JBuilder. Aparece entonces la documentación de referencia de la API de JDK de Sun. Utiliza el doclet JDK 1,4 para dar formato a la documentación.

Los dos doclets utilizan convenciones de nomenclatura HTML y estructuras de directorio diferentes. Cuando aparece Javadoc, la ficha Documentación busca los archivos formateados mediante la opción Doclet estándar. Si no encuentra este tipo de archivo, JBuilder busca a continuación archivos formateados mediante el doclet JDK 1.1. Si desea más información, consulte [“Visualización de Javadoc” en la página 15-30](#).

Nota La opción Nombre del doclet se corresponde con la opción **-doclet** de Javadoc. El asistente para Javadoc configura de forma explícita la opción **-docletpath**.

Nota Se pueden añadir doclets adicionales mediante la API de OpenTools. Consulte el ejemplo de Doclet en la carpeta `samples/OpenToolsAPI/wizards/doclet` de la instalación de JBuilder.

Selección de opciones para generar documentación

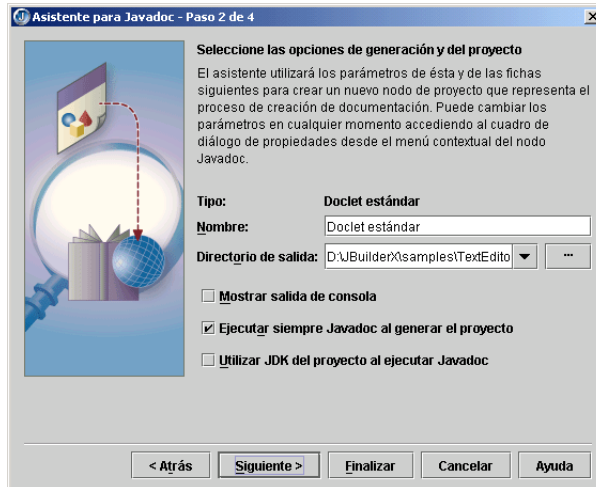
Es una función de JBuilder Developer y Enterprise.

En el segundo paso del asistente, se seleccionan:

- El nombre del nodo de documentación que genera el asistente
- El directorio de salida
- Cómo se presenta la documentación Javadoc

- Las opciones de generación de Javadoc

Figura 15.5 Paso 2 de selección de opciones de generación y del proyecto



Para seleccionar las opciones de generación y del proyecto:

- 1 Escriba el nombre del nodo de documentación en el campo Nombre. Este nombre aparece en el panel de proyecto. Por defecto, aparece el tipo de doclet que se seleccionó en el paso anterior. Puede cambiarlo por cualquier otro nombre descriptivo.
- 2 Escriba el directorio de salida de la documentación en el campo Directorio de salida. Esta es la vía de salida para los archivos generados. El asistente utiliza la primera vía de acceso a directorios configurada en la pestaña Propiedades de proyecto/Vías de acceso/Documentación. Si no ha configurado una vía de acceso a la documentación, el asistente le sugiere un directorio `doc` en el directorio del proyecto. JBuilder lo crea durante la generación de Javadoc.

Pulse el botón de puntos suspensivos (...) para desplazarse hasta un nuevo directorio. Si no hay ninguno, JBuilder se encarga de crearlo. También puede seleccionar en la lista desplegable una vía de acceso que se haya utilizado anteriormente. Estas vías de acceso corresponderían a otras vías de acceso a la documentación configuradas en su proyecto.

Un solo proyecto puede tener varias vías de acceso Javadoc, con lo que, por ejemplo, paquetes diferentes pueden utilizar distintas opciones Javadoc. Dos proyectos no deberían compartir la misma vía de acceso. Cada nodo tiene su propia vía de acceso.

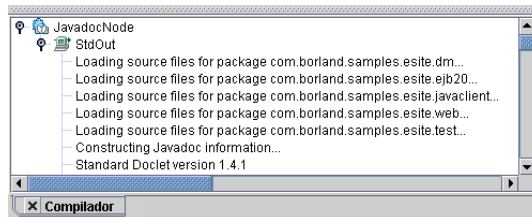
Esta opción corresponde a la opción Javadoc **-d**. El asistente configura las opciones Javadoc **-sourcepath**, **-classpath**, y **-bootclasspath**, basándose en las configuraciones del proyecto.

Nota

Por motivos de mantenimiento, la documentación Javadoc deben mantenerse en su propio directorio, en lugar de colocarlos en los

directorios de salida o fuente. Para más información consulte [“Mantenimiento de Javadoc” en la página 15-32](#).

- 3 Seleccione la opción **Mostrar salida de consola** para mostrar la salida de Javadoc y doclet en la pestaña **Generar**. (Tenga en cuenta que la salida de Javadoc se muestra con mensajes del compilador y otros elementos de salida del proceso de generación). Si la salida es un error o una advertencia, puede pulsar el nombre del archivo para dirigirse con el editor al número de línea correspondiente de ese archivo. Esta opción corresponde a la opción Javadoc **-verbose**. La salida de la generación de Javadoc presenta el siguiente aspecto:



Nota

La generación de Javadoc se detiene si se produce algún error. Los errores aparecen en el panel de la pestaña **Generar**, independientemente de que la opción **Mostrar salida de consola** se encuentre activada.

- 4 Seleccione la opción **Ejecutar siempre Javadoc al generar el proyecto** para generar archivos Javadoc cada vez que cree un proyecto. Puede desactivar esta opción al desarrollar el proyecto, ya que así se reduce de forma significativa el tiempo de compilación.

Si no elige esta opción, puede crear los archivos Javadoc en cualquier momento; solamente tiene que pulsar dos veces con el botón derecho en el nodo de documentación del panel de proyecto y seleccionar **Ejecutar Make**.

- 5 Seleccione **Utilizar JDK del proyecto** al ejecutar Javadoc para que se utilice la versión del JDK especificado en la ficha **Propiedades de proyecto/Vías de acceso**. De lo contrario, Javadoc se ejecutaría utilizando el JDK que hospeda a JBuilder.

Nota

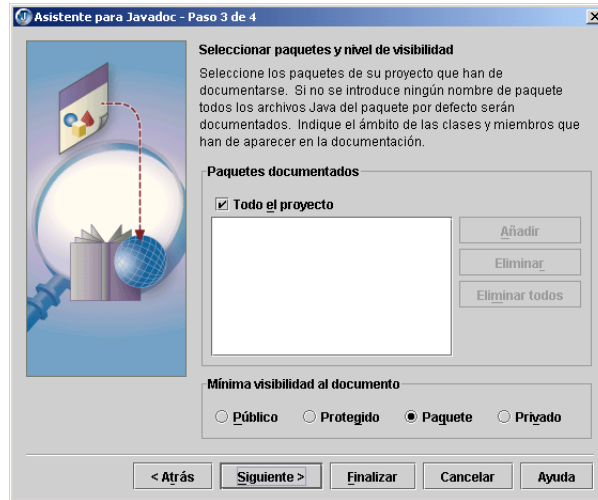
El Asistente para Javadoc utiliza el valor de la opción **Codificación** configurado en **Propiedades de proyecto/General**.

Selección de paquetes que se han de documentar

Es una función de JBuilder Developer y Enterprise.

En este paso del asistente se seleccionan los paquetes que se han de documentar y qué ámbito de clases y miembros han de aparecer en la documentación.

Figura 15.6 Paso 3 de selección de paquetes y nivel de visibilidad



Para seleccionar el paquete y el nivel de visibilidad:

- 1 Marque la opción **Todo el proyecto** para que se documenten todos los paquetes del proyecto. Esta opción se activa por defecto, para que se documenten todos los paquetes. Desactívela si desea seleccionar paquetes que se han de documentar de forma individualizada.
- Nota** Por defecto, siempre se documentan los archivos fuente del paquete.
- 2 Desactive la opción **Todo el proyecto** si desea que los paquetes se documenten de forma individualizada. Los paquetes del proyecto aparecen en la lista **Paquetes que se han de documentar**.
 - Seleccione el botón **Añadir** para poder añadir los paquetes a la lista. Aparece entonces el cuadro de diálogo **Seleccionar paquetes** que se han de documentar, en el que puede elegir de forma individualizada los paquetes para el proyecto.
 - Seleccione un paquete y, a continuación, pulse el botón **Eliminar** si desea eliminar un solo paquete de la lista.
 - Seleccione el botón **Eliminar todos** si lo que desea es eliminar todos los paquetes de la lista.
 - 3 Seleccione una de las opciones **Mínima visibilidad al documento** si desea elegir el ámbito de clases y miembros que se van a incluir en la documentación:
 - **Público**: incluye solamente clases y miembros públicos en la documentación. Esta opción se corresponde con la opción Javadoc **-public**.
 - **Protegido**: incluye solamente clases y miembros protegidos y públicos en la documentación. Esta opción se corresponde con la opción Javadoc **-protected**.

- **Paquete:** incluye solamente clases y miembros protegidos, públicos y de paquete en la documentación. Esta opción se corresponde con la opción Javadoc **-package**.
- **Privado:** incluye todas las clases y miembros en la documentación, excepto aquellos con visibilidad privada. Esta opción se corresponde con la opción Javadoc **-private**.

Nota

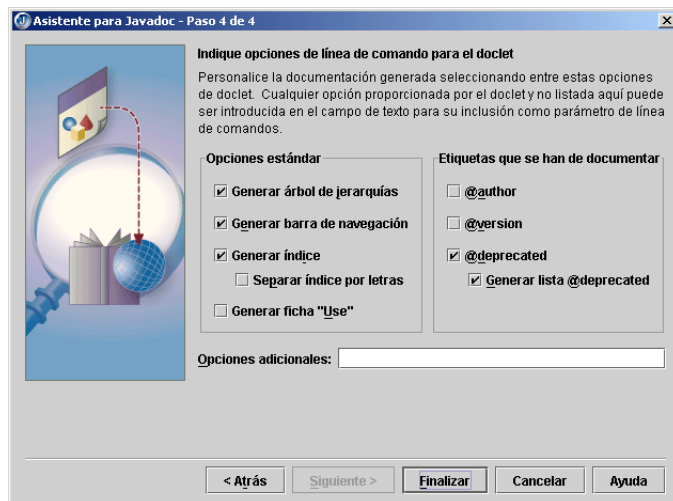
Si un archivo fuente no tiene elementos con comentarios Javadoc que cumplan los requisitos mínimos de visibilidad, la herramienta Javadoc no crea un archivo HTML para esa clase.

Especificación de opciones de línea de comandos para el doclet

Es una función de
JBuilder Developer y
Enterprise.

En el último paso del asistente se determina qué archivos de salida se generan y qué etiquetas se procesan. Todas las selecciones de esta ficha son optativas, ninguna es necesaria para que se genere Javadoc.

Figura 15.7 Paso 4 de especificación de opciones de línea de comandos para el doclet

**Nota**

En la salida de doclet de JDK 1.1 sólo están disponibles las opciones Generar árbol de jerarquías y Generar índice de la lista Opciones estándar.

Para especificar opciones de línea de comandos:

- 1 Seleccione la opción Generar árbol de jerarquías para crear el árbol de jerarquías para todas las clases de todos los paquetes. Un árbol de jerarquías es una lista de las jerarquías de todos los paquetes, clases e interfaces de la documentación. En el caso del doclet estándar, el árbol de jerarquías se crea a nivel de paquetes. El árbol de jerarquía se guarda en el archivo `overview-tree.html`, en el directorio raíz de la vía de acceso a la documentación.

- 2 Elija la opción Generar barra de navegación para crear una barra de navegación en la parte superior de cada archivo de salida HTML. Esta barra incluye enlaces a la clase y al paquete anterior y posterior, los aspectos generales de todos los paquetes de la documentación, el archivo del árbol, el índice y el tema de ayuda generado por Javadoc.
- 3 Seleccione la opción Generar índice si desea crear una entrada de índice para cada método, campo, paquete, clase e interfaz. El índice se guarda en el archivo `index-all.html`, en el directorio raíz de la vía de acceso a la documentación. Si se desactiva esta opción, se correspondería con la opción Javadoc **-noindex**.

Para la documentación Javadoc en JDK 1,4, se pueden crear enlaces a las entradas de índice por letras. Seleccione la opción Separar índice por letras. Esta opción se corresponde con la opción Javadoc **-splitindex**. Esta opción no se tiene en cuenta en el caso del doclet JDK 1.1, ya que este doclet siempre crea un índice independiente para cada letra.

- 4 Seleccione la opción "Uso" de la ficha Generar para generar una página Uso para cada paquete y otra para cada clase e interfaz. El archivo Use del paquete tiene el nombre `package-use.html`; el archivo Use de la clase tiene el nombre `class-use/classname.html`. En esta ficha se describe qué paquetes, clases, métodos, constructores y campos utilizan cada parte de la clase, interfaz o paquete. Esta opción no se tiene en cuenta para el tipo de doclet JDK 1.1.
- 5 Seleccione la opción `@author` si desea generar la documentación para las etiquetas `@author` del código fuente. Esta opción permite añadir el nombre del autor a los archivos generados Javadoc. En una misma etiqueta se pueden incluir uno o varios nombres.
- 6 Elija la opción `@version` para crear la documentación para las etiquetas `@version` del código fuente. Esta opción permite añadir el número de versión del código a los archivos generados Javadoc. Se puede aplicar tanto a una clase como a un elemento dentro de ella.
- 7 Seleccione la opción `@deprecated` si desea crear la documentación para las etiquetas `@deprecated` de su código fuente. Esta opción permite añadir un comentario que indica que el elemento API que se especifica se eliminará en una versión posterior de la API.
- 8 Elija la opción Generar lista `@deprecated` si desea crear una lista de elementos `@deprecated`. Si esta opción está desactivada, se correspondería con la opción Javadoc **-nodeprecatedlist**.
- 9 Puede introducir más opciones en el campo Opciones adicionales. Estas opciones se añaden a la línea de comandos anterior a la lista de paquetes o archivos. Las opciones que configure en este campo sustituyen a las opciones configuradas en el asistente. Observe que si especifica la opción **-locale**, siempre aparece como la primera opción de la línea de comandos, como requiere Javadoc.
- 10 Haga clic en Finalizar para crear el nodo de documentación.

En la siguiente tabla se enumeran las opciones que no se pueden configurar en el asistente. Estas opciones se pueden configurar en el campo Opciones adicionales. La tabla indica para qué doclet Javadoc se van a procesar las opciones. Por ejemplo, no todas las opciones las procesa el doclet JDK 1.1. Si desea más información acerca de las opciones Javadoc, consulte:

- **Usuarios de Windows: "Javadoc options"** en <http://java.sun.com/j2se/1.4/docs/tooldocs/windows/javadoc.html#javadocoptions>
- **Usuarios de Solaris: "Javadoc options"** en <http://java.sun.com/j2se/1.4/docs/tooldocs/solaris/javadoc.html#javadocoptions>

Tabla 15.2 Opciones que no se configuran en el asistente de Javadoc

Opción	Descripción	Doclet JDK 1.1:	Std doclet
-overview <i>vía de acceso\nombre del archivo</i>	Especifica el archivo que contiene el texto para la documentación de aspectos generales.	X	X
-help	Muestra la ayuda Javadoc, que contiene una relación de las opciones Javadoc y de línea de comandos.	X	X
-Jflag	Pasa el <i>flag</i> directamente al JDK de tiempo de ejecución que ejecuta Javadoc. No incluya un espacio entre la <i>J</i> y el <i>flag</i> . Utilice esta opción para aumentar la memoria de Javadoc, por ejemplo, <code>-J-Xms64m</code> . (El flag <code>-Xms</code> establece el tamaño de la memoria inicial. Puede utilizarlo junto con el flag <code>-Xmx</code> si desea aumentar la memoria disponible).	X	X
-locale <i>idioma_país_variante</i>	Especifica la lengua que utiliza Javadoc al crear la documentación. Javadoc selecciona los archivos de recursos de la lengua elegida para los mensajes tales como las cadenas de la barra de navegación, los títulos de las listas y tablas, los contenidos de los archivos de ayuda y los comentarios de la hoja de estilos. También especifica la clasificación de las listas alfabéticas.	X	X
-doctitle <i>title</i>	Define el título que se tiene que colocar en la parte superior del archivo resumen de aspectos generales, por debajo de la barra de navegación superior.		X
-windowtitle <i>title</i>	Especifica qué título hay que colocar en la etiqueta <code><title></code> .		X
-header <i>encabezado</i>	Define el texto de encabezado que hay que colocar en la parte superior de cada archivo HTML generado, a la derecha de la barra de navegación superior.		X
-footer <i>pie de página</i>	Especifica qué texto de pie de página se coloca en la parte inferior de cada archivo HTML generado, a la derecha de la barra de navegación inferior.		X
-bottom <i>texto</i>	Define el texto que se coloca en la parte inferior de cada archivo HTML generado, debajo de la barra de navegación inferior.		X

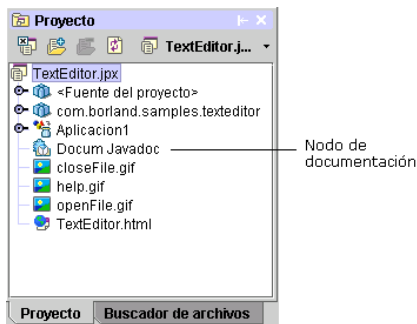
Tabla 15.2 Opciones que no se configuran en el asistente de Javadoc (continuación)

Opción	Descripción	Doclet JDK 1.1:	Std doclet
-link <i>URL de la documentación externa</i>	Crea enlaces con la documentación Javadoc existente para las clases externas referenciadas.		X
-linkoffline <i>ubicación de la lista de paquetes en la URL de documentación externa</i>	Crea enlaces con la documentación Javadoc existente para las clases externas referenciadas, en el caso de que el archivo de lista de paquetes no exista en la ubicación <i>extdocURL</i> . (Consulte -link).		X
-group <i>cabeceradegrupo modelodepaquete:modelo de paquete:...</i>	Separa los paquetes de la lista de aspectos generales en los grupos especificados.		X
-nosince	No incluye comentarios en las etiquetas <code>@since</code> .		
-nohelp	No incluye el enlace Ayuda de las barras de navegación superior e inferior.		X
-helpfile <i>vía de acceso\nombre del archivo</i>	Especifica la vía de acceso a un archivo de ayuda sustitutivo para el enlace Ayuda de la barra de navegación.		X
-stylesheetfile <i>vía de acceso\nombre del archivo</i>	Define la vía de acceso a un archivo de hoja de estilos sustitutivo.		X
-serialwarn	Genera advertencias del compilador para las etiquetas <code>@serial</code> que falten.		X
-charset <i>nombre</i>	Define el conjunto de caracteres HTML para la documentación generada.		X
-doencoding <i>nombre</i>	Especifica la codificación de la documentación generada.		X

Creación de archivos de salida

**Es una función de
JBuilder Developer y
Enterprise.**

Una vez que haya configurado todas las opciones en el asistente y que haya pulsado el botón Finalizar, se crea el nodo de documentación en el panel de proyecto.

Figura 15.8 Nodo de documentación en el panel de proyecto

Los archivos HTML no están disponibles hasta que no se generan. Para generar los archivos de salida puede hacer lo siguiente:

- El proyecto se genera si se ha configurado la opción Ejecutar siempre Javadoc al generar el proyecto de la ficha Seleccione las opciones de generación y del proyecto del asistente.
- Pulse con el botón derecho del ratón en el nodo de documentación y seleccione Ejecutar Make. Esta opción sólo crea Javadoc, no crea el proyecto. Para borrar archivos de salida en el directorio configurado, seleccione Limpiar. Seleccione Generar de nuevo si desea limpiar y, después, generar.

La información aparece en la ficha Compilador del panel de mensajes, si se ha activado la opción Mostrar salida de consola de la ficha Seleccione las opciones de generación y del proyecto en el asistente.

Javadoc genera un conjunto de archivos con formato HTML, basados en las propiedades seleccionadas y que se colocan en el directorio de salida seleccionado, que normalmente es el directorio `doc`. Los archivos de salida para la opción Doclet estándar incluyen:

- Un archivo `classname.html` para cada clase o interfaz del proyecto, que contiene la documentación para la clase o interfaz.
- Un archivo `package-summary.html` para cada paquete del proyecto, que enumera las clases y paquetes, y que ofrece una descripción general.
- Un archivo `overview-summary.html` para el conjunto completo de paquetes, que es la página principal de la documentación. Este archivo solamente se crea si su proyecto contiene dos o más paquetes y ha utilizado la opción **-overview** del campo Opciones adicionales, en la ficha Indique opciones de línea de comandos para el doclet del asistente utilizado para crear Javadoc.
- Un archivo `overview-tree.html` para la jerarquía de clases del conjunto entero de paquetes.
- Un archivo `package-tree.html` para la jerarquía de clases de cada paquete.
- Un archivo `package-use.html` para cada paquete, clase e interfaz, que determina qué paquetes, clases, métodos, constructores y campos utiliza cada parte del paquete, clase o interfaz en cuestión. Para crear este archivo, tiene que activar la opción `@use` de la ficha Indique opciones de línea de comandos para el doclet del asistente.
- Un archivo `deprecated-list.html` para todos los nombres desaconsejados. Para crear este archivo, tiene que activar la opción `@use` de la ficha Indique opciones de línea de comandos para el doclet del asistente.
- Un archivo `serialized-form.html` con la información acerca de las clases serializables y exteriorizables. Para que se genere este archivo, hay que introducir las etiquetas `@serial`, `@serialField` y `@serialData` en el campo Opciones adicionales de la ficha Indique opciones de línea de comandos para el doclet del asistente.

- Un archivo `index-*.html` que recoge todos los nombres de clases, interfaces, constructores, campos y métodos por orden alfabético. Para crear este archivo, debe activar la opción Crear índice de la ficha Especificar opciones doclet de línea de comandos del asistente.

Javadoc también crea archivos de apoyo, que incluyen:

- Un archivo `help-doc.html`, que describe cómo se utiliza la barra de navegación.
- Un archivo `index.html`, que genera los marcos HTML.
- Un número de archivos `*-frame.html`, que recogen los paquetes, clases e interfaces que se utilizan para mostrar los marcos HTML.
- Un archivo `package-list`, que es un archivo de texto utilizado por las opciones `-link`. No se puede acceder a él a través de ningún enlace.
- Un archivo `stylesheet.css` que controla la apariencia de los archivos HTML, y que se basa en el doclet seleccionado en la ficha Seleccionar doclet del asistente.

Si desea más información acerca de los archivos generados, consulte "Generated Files" en:

- **Usuarios de Windows:** <http://java.sun.com/j2se/1,4/docs/tooldocs/win32/javadoc.html#generatedfiles>
- **Usuarios de Solaris:** <http://java.sun.com/j2se/1,4/docs/tooldocs/solaris/javadoc.html#generatedfiles>

Creación de archivos adicionales

**Es una función de
JBuilder Developer y
Enterprise.**

Javadoc genera de forma automática numerosos archivos de salida para los archivos fuente `.java`. También se pueden crear archivos adicionales de salida, como archivos descriptivos de paquetes o archivos de comentarios de aspectos generales, a partir de archivos auxiliares.

Archivos de paquetes

Cada paquete puede tener su propio comentario de documentación en su propio archivo fuente. Javadoc fusiona este archivo de comentarios en la ficha resumen de paquetes que genera para cada paquete del proyecto. Este archivo debe tener el nombre `package.html`, y debe estar colocado en el directorio de paquetes del árbol fuente, junto con los archivos de clase del paquete. Javadoc buscará automáticamente este nombre de archivo en esa ubicación.

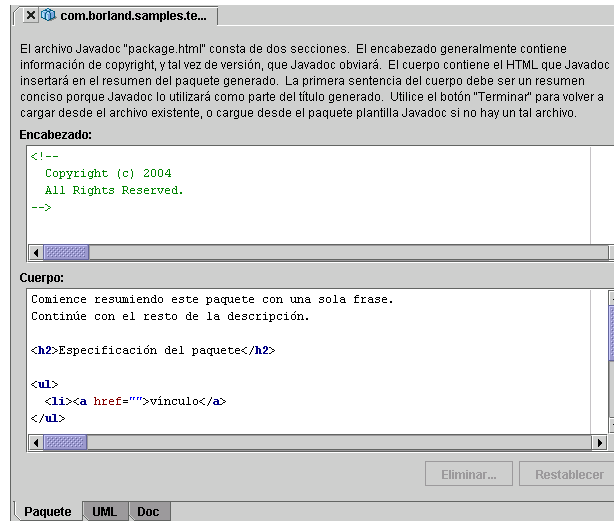
El archivo `package.html` debe contener un solo comentario de documentación en formato HTML. No incluya las etiquetas Javadoc de comienzo y final de comentario (`/** y */`). No ponga ningún título ni ningún otro texto entre la etiqueta `<body>` y la primera frase. La primera frase debe ser un resumen.

JBuilder ofrece un editor de archivos de paquetes que permite crear, modificar o eliminar fácilmente el archivo `package.html` para los paquetes individuales del

proyecto. El editor de archivos de paquetes coloca el archivo en la ubicación correcta para que Javadoc lo pueda procesar.

Para utilizar este editor:

- 1 Abra el proyecto. En el panel de proyecto, seleccione el paquete para el que desee crear un archivo de paquetes.
- 2 Haga doble clic en el paquete. El editor de archivos de paquetes aparece en la pestaña Paquete del panel de contenido.



- 3 Escriba el texto de encabezado del archivo `package.html` en la sección Encabezado del editor. Esta sección normalmente contiene la información de la versión y el copyright. Javadoc no procesa texto en esta sección, sino que lo deja en una etiqueta de comentario en el archivo `package.html`.
- 4 Escriba el texto del cuerpo en la sección Cuerpo. La primera frase debe ser una breve frase resumen del paquete. A continuación, escriba una descripción completa del paquete. Puede utilizar la plantilla para introducir enlaces a otros paquetes y/o a la documentación relacionada.

Al generar los archivos de salida, la frase resumen aparece en la parte superior del archivo de paquetes. El resto de la descripción del paquete sigue a la lista Resumen de clase. Tenga en cuenta que puede crear archivos `package.html` individuales para cada uno de los paquetes del proyecto.

Nota El texto por defecto del encabezamiento y el cuerpo se almacena en el archivo `javadocPackage.template`, que se crea en el directorio `.jbuilder` la primera vez que se utiliza.

Si desea información adicional acerca de los archivos de paquete, consulte "Package Comment Files" en:

- Usuarios de Windows:

<http://java.sun.com/j2se/1,4/docs/tooldocs/win32/javadoc.html#packagecomment>

- Usuarios de Solaris:

```
http://java.sun.com/j2se/1,4/docs/tooldocs/solaris/  
javadoc.html#packagecomment
```

Archivos de comentarios de aspectos generales

Cada proyecto puede tener su propio archivo de comentarios de aspectos generales en su propio archivo fuente. Javadoc fusiona este archivo de comentarios en la ficha de aspectos generales que crea para cada uno de los paquetes del proyecto. Puede ponerle al archivo el nombre que desee, normalmente es `overview.html`. También lo puede colocar donde quiera, aunque normalmente se coloca en la parte superior del árbol de fuentes.

El archivo `overview.html` debe contener un solo comentario de documentación, en formato HTML. No incluya las etiquetas Javadoc de comienzo y final de comentario (`/** y */`). No ponga ningún título ni ningún otro texto entre la etiqueta `<body>` y la primera frase. La primera frase debe ser un resumen.

Si desea información adicional acerca del archivo de comentarios de aspectos generales, consulte "Overview Comment Files" en:

- Usuarios de Windows:

```
http://java.sun.com/j2se/1,4/docs/tooldocs/win32/  
javadoc.html#overviewcomment
```

- Usuarios de Solaris:

```
http://java.sun.com/j2se/1,4/docs/tooldocs/solaris/  
javadoc.html#overviewcomment
```

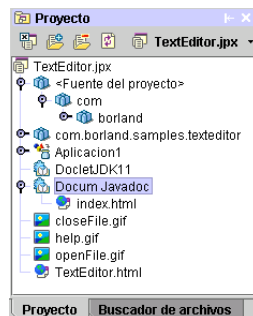
Visualización de Javadoc

**Es una función de
JBuilder Developer y
Enterprise.**

Existen varias formas de visualizar Javadoc una vez que se ha generado.

Si desea visualizar el Javadoc de todo el proyecto, amplíe el nodo de documentación y haga doble clic en `index.html` (para la información en la que se ha utilizado el Doclet estándar), o bien, `index-1.html` (para la información en la que se ha utilizado el Doclet JDK 1.1).

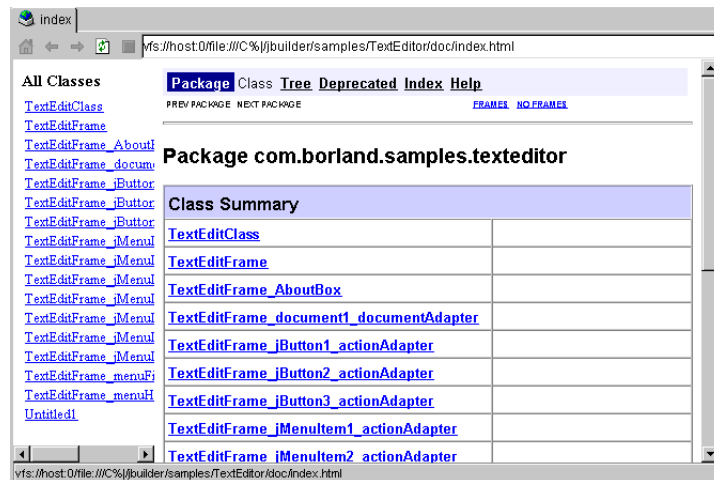
Figura 15.9 Nodo de documentación ampliados



Se abre el archivo de índice en el panel Ver del Visualizador de aplicaciones.

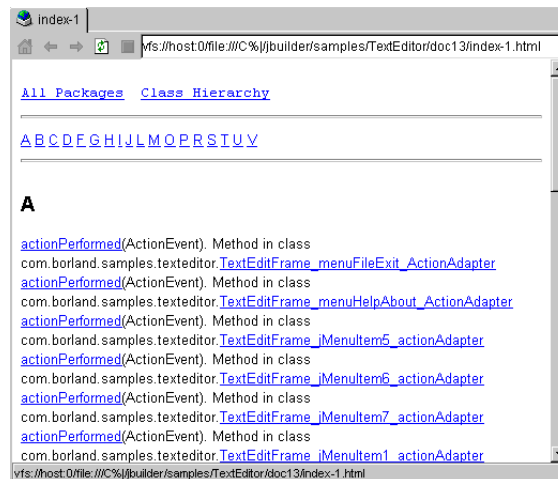
- Para la documentación en Doclet estándar, los paquetes y clases aparecen en el marco izquierdo. El marco derecho muestra las tablas resumen.

Figura 15.10 Salida de archivos de índices desde el Doclet estándar



- Para la documentación en Doclet JDK 1.1, aparece un índice alfabético.

Figura 15.11 Salidas de archivos de índices desde el Doclet JDK 1.1



También puede visualizar el Javadoc de un solo archivo, seleccionando ese archivo en el panel del proyecto y, a continuación, la pestaña Documentación. Para ver el Javadoc de un solo archivo desde un diagrama de clase UML, pulse con el botón derecho del ratón en el nombre de la clase del diagrama y seleccione Ver Javadoc. Aparece entonces el archivo HTML en el Visualizador de ayuda.

En cualquiera de estas formas de visualizar Javadoc, puede utilizar la barra de navegación superior o inferior y los enlaces en las tablas resumen para desplazarse por la documentación.

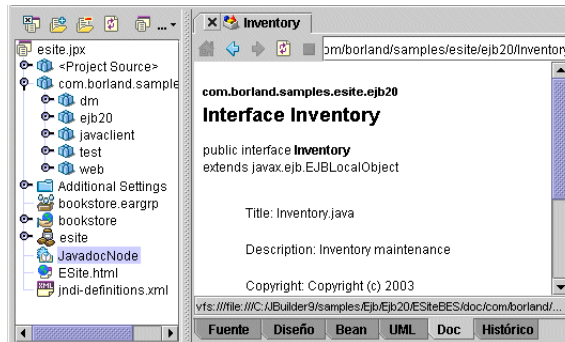
Apariencia de Javadoc en JBuilder

Si selecciona la pestaña Documentación para mostrar Javadoc de un archivo fuente abierto en el editor, JBuilder busca en primer lugar archivos con formato HTML o archivos en JDK 1.1, utilizando la vía de acceso a la documentación que se define en la pestaña Propiedades de proyecto/Vías de acceso/Documentación.

- Si solamente existen archivos de salida con formato JDK 1.1, son éstos los que aparecen.
- Si existen archivos tanto con formato JDK 1.1 como con formato estándar, aparece el archivo del doclet estándar.

Si no hay archivos de ninguno de estos dos tipos, JBuilder muestra el Javadoc "sobre la marcha", es decir, muestra el Javadoc que se genera directamente a partir de los comentarios del archivo fuente. Esto permite actualizar Javadoc de tal manera que siempre se muestre para los archivos fuente, incluso si todavía no ha creado Javadoc. En esta vista no hay enlaces disponibles.

Figura 15.12 Información Javadoc "sobre la marcha"



Mantenimiento de Javadoc

Es una función de JBuilder Developer y Enterprise.

La ventaja principal de crear Javadoc en su propio directorio de salida, como un directorio `doc`, es que así puede mantenerlo fácilmente. Para eliminar los archivos de salida del directorio de salida de la documentación, pulse el nodo de la documentación con el botón derecho del ratón y seleccione Limpiar. Se eliminan los archivos de salida y los archivos CSS. Sin embargo, no se borra la estructura de directorios. Si selecciona Generar de nuevo, en primer lugar se limpiará el directorio y, a continuación, se generarán de nuevo los archivos de salida.

Cambio de propiedades para el nodo de documentación

**Es una función de
JBuilder Developer y
Enterprise.**

Una vez que se ha creado el nodo de la documentación, puede cambiar las propiedades del nodo, incluidos el nombre del nodo, el directorio de salida y el momento de creación de Javadoc. Para cambiar las propiedades, pulse con el botón derecho del ratón en el nodo de documentación del panel de proyecto y seleccione Propiedades. En el cuadro de diálogo Propiedades, seleccione la pestaña Nodo para cambiar las propiedades del nodo. Seleccione la pestaña Javadoc si desea cambiar los paquetes que tienen que documentarse. Si desea cambiar las opciones de doclet, seleccione la pestaña Doclet.

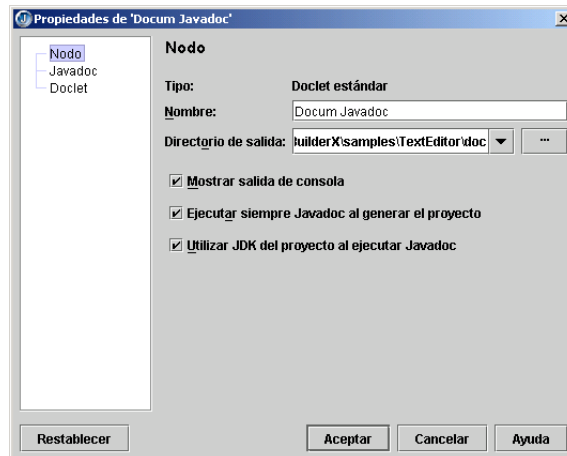
Cambio de las propiedades del nodo

**Es una función de
JBuilder Developer y
Enterprise.**

Si desea cambiar las propiedades del nodo, utilice la pestaña Nodo del cuadro de diálogo Propiedades. Es posible cambiar las siguientes opciones:

- Nombre del nodo
- Directorio de salida
- Apariencia de la consola de salida
- Cuándo crear Javadoc
- Qué JDK utilizar (el del proyecto o uno albergado en JBuilder)

La ficha Nodo tendrá este aspecto:



Si desea más información, consulte [“Selección de opciones para generar documentación” en la página 15-19.](#)

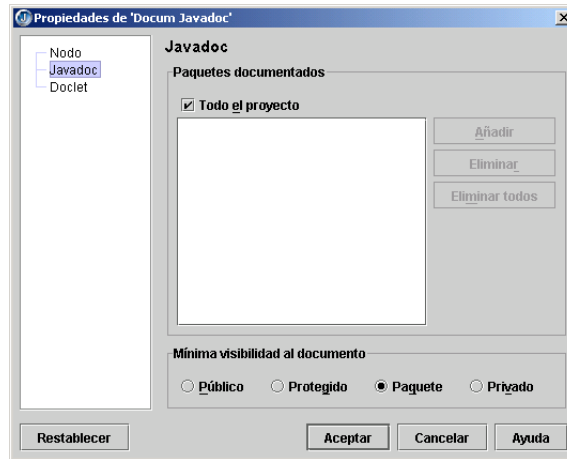
Modificación de propiedades Javadoc

**Es una función de
JBuilder Developer y
Enterprise.**

Utilice la pestaña Javadoc del cuadro de diálogo Propiedades para modificar los paquetes que se han de documentar. Es posible cambiar las siguientes opciones:

- Paquetes que se han de documentar.
- Visibilidad más baja documentado.

La ficha Javadoc tendrá este aspecto:



Si desea más información, consulte [“Selección de paquetes que se han de documentar” en la página 15-21](#).

Modificación de propiedades doclet

**Es una función de
JBuilder Developer y
Enterprise.**

Utilice la pestaña Doclet del cuadro de diálogo Propiedades si desea modificar las opciones y etiquetas documentados. Puede elegir:

- Si se genera el archivo del árbol de jerarquías
- Si aparece la barra de navegación en la parte superior de cada archivo generado
- Si se crea un índice
- Las etiquetas que se han documentado

La ficha Doclet tendrá este aspecto:

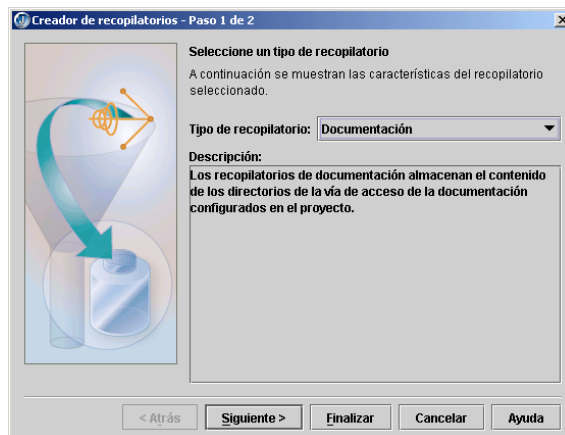


Si desea más información, consulte [“Especificación de opciones de línea de comandos para el doclet”](#) en la página 15-23.

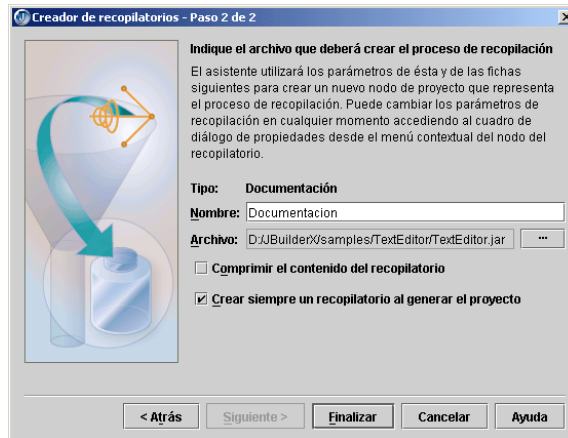
Creación de un archivo recopilatorio de documentación

Después de la ejecución definitiva de Javadoc, puede utilizar el Creador de recopilatorios para crear un archivo de documentación JAR. En este tipo de archivo JAR se encuentran todos los archivos de los directorios de la vía de acceso a la documentación y, normalmente, suele ser el directorio `doc`. Para crear un recopilatorio de documentación:

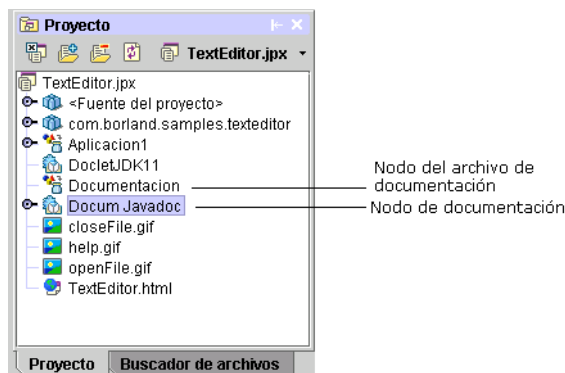
- 1 Seleccione Asistentes\Creador de recopilatorios.
- 2 En la ficha Seleccione un tipo de recopilatorio del Creador de recopilatorios, seleccione Documentación en la lista desplegable Tipo de recopilatorio. El Creador de recopilatorios tiene la siguiente apariencia:



- 3 Pulse Siguiente para pasar al siguiente paso. Se muestra la ficha Indique el archivo que deberá crear el proceso de recopilación.



- 4 Escriba un nombre en el campo Nombre para el nodo de documentación. Este nodo aparece en el panel de proyecto al pulsar Finalizar.
- 5 Introduzca el nombre del archivo JAR en el campo Archivo: este archivo debe tener la extensión .jar . Por defecto, se encuentra en el directorio raíz del proyecto.
- 6 Si desea comprimir el recopilatorio, seleccione Comprimir el contenido del recopilatorio. Esta opción se utiliza para reducir al mínimo el tamaño del archivo JAR.
- 7 Seleccione Crear siempre un recopilatorio al generar el proyecto para que se cree un archivo JAR cada vez que seleccione Ejecutar Make o Generar.
- 8 Seleccione Finalizar para cerrar el asistente y crear el nodo de recopilatorio de documentación en el panel del proyecto. El panel del proyecto presentará este aspecto:



- 9 Seleccione Proyecto|Ejecutar Make del proyecto para ejecutarlo y crear el archivo JAR.

El archivo JAR de documentación se coloca en el directorio especificado en el Creador de recopilatorios.

También tiene la posibilidad de crear un recopilatorio fuente para los archivos fuente del proyecto, si selecciona Tipo de recopilatorio fuente en la ficha Seleccione un tipo de recopilatorio del Creador de recopilatorios.

Nota Se puede añadir documentación a cualquier archivo JAR seleccionando la opción Incluir la documentación del proyecto en el recopilatorio (en la ficha del asistente Indique el archivo que deberá crear el proceso de recopilación) al crear un recopilatorio de la mayoría de tipos de recopilatorio del Asistente para recopilatorios.

Si desea obtener más información acerca del uso de la vista Personalizar, consulte [“Utilización del Creador de recopilatorios” en la página 16-16.](#)

Creación de un doclet personalizado

**Es una función de
JBuilder Developer y
Enterprise.**

Puede crear un doclet personalizado si amplía el Asistente para Javadoc con la API de OpenTools. Si desea un ejemplo de doclet personalizado, abra el archivo `Doclet.jpx` del proyecto en la carpeta `samples/OpenToolsAPI/wizards/doclet` de su instalación de JBuilder. Los doclets personalizados no necesitan generar archivos de salida. Por ejemplo, las etiquetas personalizadas se pueden utilizar junto con la función de análisis de archivos fuente de la herramienta Javadoc. El doclet puede crear entonces archivos XML o archivos Java adicionales. El doclet personalizado debe situarse en el directorio `<jbuilder>/doclet`.

Para obtener más información acerca de la API de OpenTools, consulte "Conceptos básicos de JBuilder OpenTools" en *“Developing OpenTools”* (en inglés).



Distribución de programas en Java

La distribución de programas Java consiste en reunir los diferentes archivos de clase de Java, los archivos de imágenes y demás archivos necesarios para el programa, para copiarlos en una dirección de un ordenador cliente o servidor con el fin de que se puedan ejecutar. Los archivos se pueden entregar por separado, o todos juntos en un recopilatorio comprimido o sin comprimir.

Nota Todas las referencias a "programas" Java de este documento se refieren a aplicaciones Java, applets, JavaBean o Enterprise Java Beans.

JBuilder proporciona el Creador de recopilatorios que ayuda a distribuir un programa. Además, el Creador de ejecutables nativos, disponible en JBuilder Developer y Enterprise, agrupa un archivo JAR de aplicaciones con englobadores de ejecutables nativos para conseguir una distribución más rápida. Los archivos JAR también se pueden crear desde la línea de comandos mediante la herramienta **jar** de Sun que se incluye en el JDK.

El Creador de recopilatorios de JBuilder simplifica la distribución reuniendo automáticamente las clases, los recursos y bibliotecas que necesita el programa y distribuyendo los archivos en un archivo ZIP o JAR comprimido o sin comprimir. También crea el `descriptor` del archivo JAR.

Importante Si hay clases o recursos a los que se llama como parámetros durante la ejecución (por ejemplo, `className(String)`), el Creador de recopilatorios no los puede distinguir si no se añaden al proyecto expresamente antes de crear el recopilatorio. De otro modo, es necesario añadirlos manualmente al recopilatorio más adelante.

El Constructor de recopilatorios de JBuilder crea también un nodo de archivos del proyecto, que permite un fácil acceso al archivo recopilatorio y al archivo descriptor. El archivo recopilatorio puede crearse o modificarse en cualquier fase del desarrollo. También se puede ver el contenido del archivo recopilatorio y del archivo descriptor. Para más información consulte ["Utilización del Creador de recopilatorios" en la página 16-16](#).

El primer paso para la distribución de cualquier programa es identificar qué contenidos (clases, recursos y dependencias) del proyecto y de biblioteca, se van a incluir en el archivo recopilatorio. La inclusión de todas las clases, recursos y dependencias genera un archivo recopilatorio muy voluminoso. Sin embargo, no es necesario proporcionar al usuario final otros archivos, ya que el archivo de revisiones contiene todo lo necesario para ejecutar el programa. Si se excluyen clases, recursos o dependencias parcial o totalmente, será necesario proporcionarlos por separado al usuario final.

La distribución es un tema complejo y avanzado que se debe estudiar con el fin de comprender todas sus implicaciones. El Creador de recopilatorios de JBuilder reduce esta complejidad y ayuda a crear un archivo de revisiones con todos los requisitos necesarios para la distribución.

Consulte

- "Trail: Jar Files" (en inglés) en el Java Tutorial de <http://java.sun.com/docs/books/tutorial/jar/index.html>
- "Using JAR Files: The Basics" (en inglés) en <http://java.sun.com/docs/books/tutorial/jar/basics/index.html>
- Paso 16 del tutorial de JBuilder "Creación de un editor de texto de Java", en *Diseño de aplicaciones con JBuilder*

Nota En este documento se da por supuesto que ya entiende la diferencia entre un applet (un programa que se ejecuta dentro de otro contexto, normalmente un navegador Web) y una aplicación (una aplicación autónoma que contiene un método `main()`). Para obtener más información sobre los applets, los problemas con navegadores y la compatibilidad con JDK, consulte la sección "Problemas con los navegadores" incluida en *Guía del desarrollador de aplicaciones web*.

Distribución de archivos recopilatorios de Java (JAR)

Los programas Java pueden consistir en muchos archivos .class, además de varios archivos de recursos, propiedades y documentación. Un programa voluminoso puede tener cientos o incluso miles de estos archivos. Cuando los programas están completos y listos para su distribución, es necesario contar con una herramienta cómoda para agrupar los archivos de clase y de otro tipo en un solo conjunto de distribución.

Los archivos se pueden distribuir por separado, o unidos en un archivo recopilatorio para facilitar la entrega. También se puede entregar un programa voluminoso en varios archivos recopilatorios que representen bibliotecas y

programas principales. Los archivos recopilatorios comprimidos tienen la ventaja de que el tiempo de descarga de las applets es más corto, y los archivos ocupan menos espacio en el servidor o el sistema de destino. Su desventaja es que son un poco más lentos durante la ejecución.

Una forma de distribuir programas de Java consiste en utilizar archivos JAR comprimidos o sin comprimir. Un archivo JAR contiene archivos de clase y recursos (en una estructura de directorios apropiada para paquetes), un archivo descriptor y, de forma potencial, archivos de firmas, tal y como se definen en la Manifest Specification. El archivo descriptor posibilita las funciones más avanzadas de los archivos JAR, como el sellado de paquetes, las ediciones de paquetes y la firma electrónica.

Los archivos JAR (`.jar`) son fundamentalmente archivos ZIP con una extensión distinta y con determinadas reglas sobre la estructura interna de directorio. JavaSoft utilizaba el formato de archivo ZIP de PKWARE como base del formato de archivos JAR.

Nota Sólo los navegadores JDK 1.1 o posteriores aceptan los archivos JAR. Si está desarrollando un applet para un navegador JDK 1.0.2 debe utilizar un archivo comprimido ZIP.

El archivo HTML desde el que se carga el applet no procede de este recopilatorio. Es un archivo distinto del servidor. No obstante, la especificación JavaBeans indica que los archivos HTML de documentación de un bean se pueden colocar en un recopilatorio.

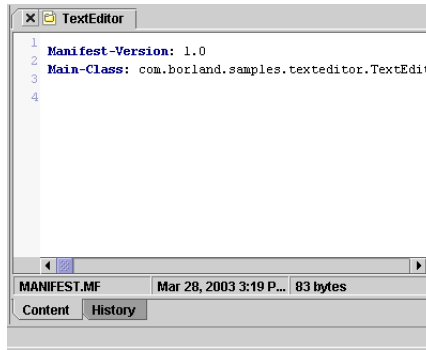
Conceptos básicos acerca del archivo descriptor

El archivo descriptor de un archivo JAR es un archivo de texto que incluye información acerca de algunas o todas las clases contenidas en el archivo JAR en cuestión. En Java 2, también contiene información sobre cuál de las clases del archivo JAR es la ejecutable.

El archivo descriptor de todo archivo JAR debe llamarse `manifest.mf` y debe encontrarse en el directorio `META-INF` del archivo JAR.

El archivo descriptor por defecto generado por el Creador de recopilatorios incluye los encabezados en la parte superior del archivo:

<code>Manifest-Version: 1.0</code>	Informa de que las entradas del descriptor toman la forma de pares "cabecera:valor" y que se trata de la versión 1.0 de la especificación del descriptor.
<code>Main-Class: class-name</code>	Este encabezado se utiliza para los tipos de archivo de aplicación. Indica qué clase ejecuta la aplicación.



La cabecera **Main-Class** permite ejecutar aplicaciones desde el archivo JAR por medio de la opción **-jar** de las herramientas de Java, que inicia la aplicación de Java desde una línea de comandos.

Hay otros encabezados que se pueden añadir al archivo descriptor, que proporcionan al archivo JAR funciones adicionales que permiten su uso con varias finalidades.

Consulte

- "Understanding the Manifest" (en inglés) en <http://java.sun.com/docs/books/tutorial/jar/basics/manifest.html>.
- "Special purpose manifest headers" (en inglés) en <http://java.sun.com/docs/books/tutorial/jar/basics/manifest.html#special-purpose>
- "JAR Manifest" (en inglés) en [http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html#JAR Manifest](http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html#JAR%20Manifest)
- "JAR Manifest - Main Attributes" (en inglés) en [http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html#Main Attributes](http://java.sun.com/j2se/1.4/docs/guide/jar/jar.html#Main%20Attributes)
- "Command-line tools" (en inglés) en <http://java.sun.com/products/js2e/1.4/docs/tooldocs/tools.html#basic>
- "JAR Guide" (en inglés), en <http://java.sun.com/j2se/1.4/docs/guide/jar/jarGuide.html>

Estrategias de distribución

Existen dos estrategias básicas para distribuir su aplicación:

- Entregar las bibliotecas de libre distribución con el archivo JAR e incluirlas en `CLASSPATH` durante la ejecución, en vez de colocar las clases necesarias de estas bibliotecas dentro del archivo JAR. Ésta es la manera más fácil de efectuar una distribución y la que crea el archivo JAR más pequeño. Ésta es la mejor elección si se entregan varias aplicaciones o applets al mismo lugar y se desea que compartan las bibliotecas.

Si desea información sobre los elementos de libre distribución y los de uso exclusivo, según las condiciones de la licencia de JBuilder, consulte los archivos `< jbuilder>/license.html` y `</jbuilder>/redist/deploy.html`.

- Crear un archivo JAR con la ayuda de la herramienta **Jar** de Sun, disponible en el JDK, o del Creador de compiladores de JBuilder. El Creador de compiladores ofrece muchas opciones para reunir las clases, recursos y bibliotecas que el programa necesita. Las opciones elegidas dependen de los requisitos de la distribución (incluidas las consideraciones de espacio), de si el programa es un applet o una aplicación autónoma y de la instalación que hagan los usuarios del programa. Consulte [“Utilización del Creador de compiladores” en la página 16-16](#).

El proceso de distribución de JBuilder se puede resumir en unos pocos pasos básicos:

- 1 Cree y compile el código en JBuilder.
- 2 Cree el archivo JAR con la ayuda de la herramienta **jar** de Sun o con el Creador de compiladores de JBuilder.
- 3 Cree un procedimiento de instalación.
- 4 Entregue el archivo JAR, todos los archivos JAR de libre distribución que sean necesarios y los archivos de instalación.

Consulte

- La herramienta **jar** de Sun <http://java.sun.com/products/js2e/1.4/docs/tooldocs/tools.html#basic>
- [“Distribución rápida” en la página 16-8](#) para obtener más información acerca del proceso de distribución
- [“Utilización del Creador de compiladores” en la página 16-16](#) para saber más sobre la creación de archivos JAR

Aspectos relativos a la distribución

Es importante tener claras las siguientes cuestiones para determinar cuál es la estrategia de distribución idónea:

- ¿Está todo lo necesario en la vía de acceso a clases?
- ¿El programa depende de funciones JDK 1.1 o Java 2 (JDK 1.2 y superior)?
- ¿Tiene el usuario bibliotecas Java distintas del JDK instaladas en su ordenador?
- ¿Se trata de un applet o de una aplicación?
- ¿Existen en el servidor limitaciones de espacio en disco o de tiempo de descarga de archivos?

¿Está todo lo necesario en la vía de acceso a clases?

Los problemas de distribución se deben, sobre todo, a que el archivo JAR o la vía de acceso a clases no contiene todo lo necesario. Si no se han añadido las clases necesarias de una biblioteca determinada al archivo JAR, se debe comprobar que las bibliotecas de libre distribución se encuentran especificadas en la opción **-classpath** de la línea de comandos de Java o en la variable de entorno `CLASSPATH`. Es recomendable utilizar la opción **-classpath**, ya que las vías de acceso a clases se pueden definir independientemente para una aplicación sin que esto influya sobre las otras, y las demás aplicaciones no pueden modificar su valor.

Sugerencia JBuilder indica la vía de acceso a clases cuando se ejecuta desde el IDE. Si lo refleja en la línea de comandos, el programa funcionará.

La forma en que se define la variable de entorno `CLASSPATH` depende del sistema operativo que se esté utilizando.

Consulte

- “Definición de la variable de entorno `CLASSPATH` para herramientas de línea de comandos” en la página A-4
- “Setting the class path” (en inglés) en <http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html>

¿El programa depende de funciones JDK 1.1 o Java 2 (JDK 1.2 y superior)?

Si está desarrollando un applet, puede tener ciertas complicaciones si algunos usuarios utilizan navegadores de Internet que no estén actualizados para aceptar aplicaciones escritas con características de JDK 1.1.x o posteriores, como Swing.

Los navegadores que cumplen las especificaciones de JDK 1.0.2 no aceptan archivos recopilatorios JAR. Por lo tanto, si ha escrito un applet compatible con JDK 1.0.2 y desea distribuirlo, asegúrese de crear un recopilatorio en formato ZIP.

Consulte

- “Uso de applets” en la *Guía del desarrollador de aplicaciones web*

¿Tiene el usuario bibliotecas Java instaladas en su ordenador?

Si su programa utiliza componentes que dependen de bibliotecas que no son JDK, necesita proporcionárselos al usuario en el archivo JAR. Los archivos de libre distribución de JBuilder se encuentran en el directorio `<jbuilder>/redist`. Todos los archivos de libre distribución del JDK se encuentran en los

`directorios <jbuilder>/<jdk>/lib/ y <jbuilder>/<jdk>/jre/lib/.<jdk>` representa el nombre del directorio de JDK.

Nota Al realizar la distribución con el Creador de compiladores de JBuilder, puede crear estas bibliotecas seleccionando la opción Incluir clases necesarias y todos los recursos del paso 4 del asistente. Esta opción asegura que las clases requeridas, así como todos los recursos (incluidos los de las bibliotecas de terceros), están incluidos en el archivo JAR del programa. El Creador de compiladores nunca incluye JDK en el archivo compilador. Presupone que las clases JDK ya existen en el ordenador de destino en forma de JDK instalado, entorno de ejecución Java o Plug-in Java, o que se proporcionarán en la instalación. Consulte [“Utilización del Creador de compiladores” en la página 16-16](#).

Si puede presuponer que todos los usuarios tienen ya estos compiladores en su entorno, ya sea porque los hayan instalado o porque se les hayan suministrado en algún tipo de proceso de instalación, no es necesario que las aplicaciones y applets que les suministre contengan estos paquetes.

Si no está seguro de que el usuario cuenta con estas bibliotecas, debe proporcionárselas. Esto es especialmente importante en el caso de distribución de applets. Cuando se distribuye un applet, es necesario copiar en el servidor tanto estas bibliotecas como los otros recursos que se puedan necesitar.

Importante En JDK 1.1.x, las clases Swing/JFC **no** se entregan como parte del JDK. Si escribe programas que utilizan cualquiera de estos JDK, debe descargar y entregar `swingall.jar`, que contiene estos archivos.

Consulte

- [“Redistribución de las clases que se suministran con JBuilder” en la página 16-13](#)
- [“Utilización del Creador de compiladores” en la página 16-16](#)

¿Se trata de un applet o de una aplicación?

Las aplicaciones no se distribuyen de la misma forma que las applets. En lo relativo a la distribución, ésta es la principal diferencia:

- En el caso de las aplicaciones, el usuario debe utilizar el ejecutable de Java (del Entorno de ejecución de Java) para ejecutar las clases o los archivos JAR que se le han suministrado, ya sea directamente desde el servidor o después de haberlas instalado en el ordenador. Si el usuario no tiene los archivos JRE necesarios, se deben incluir en el conjunto de distribución.
- En el caso de las applets, se presupone que el usuario utiliza un navegador de Internet o un visualizador de applet, y que es necesario que haya una página HTML que contenga la etiqueta `<APPLET>` que hace referencia a las clases que se quieren ejecutar. En este caso, se debe comprobar que el navegador del usuario acepta la versión del JDK utilizado para desarrollar

Importante

las applets y, en caso contrario, proporcionarle el Plug-in de Java, de Sun, para el navegador.

Si tiene intención de utilizar el Plug-in de Java, de Sun, lea antes toda la documentación pertinente al tema. Hay varias versiones del Plug-in de Java y del Convertidor HTML. Ya que las versiones de plug-in son incompatibles entre sí, sólo se puede tener instalada una versión al mismo tiempo. Sólo es recomendable utilizar el Plug-in Java en entornos controlados, por ejemplo en una intranet, en los que se sabe qué versión de navegador se utiliza y qué JDK acepta.

Consulte

- "Uso de applets" en la *Guía del desarrollador de aplicaciones web*
- Si desea más información, consulte el Plug-in Java, que se encuentra en <http://java.sun.com/products/plugin/>

Tiempo de descarga de archivos

Una de las primeras preguntas a las que debe contestar un desarrollador es si desea guardar el programa en un recopilatorio. Los factores que influyen en mayor medida sobre esta decisión son el tiempo de descarga y el tamaño del programa, especialmente para las applets.

Una de las ventajas del uso del Creador de recopilatorios es que sólo se incluyen los archivos necesarios. El Creador de recopilatorios identifica todas las clases que necesita el proyecto y las reúne en un archivo recopilatorio. Esto mejora la eficacia del tiempo de recuperación de archivos, utilización del disco y consumo de sockets de red, cuando la aplicación o applet se inicia desde un navegador.

Mientras se familiariza con las clases Java y sus dependencias, es posible que deba crear archivos JAR más voluminosos para asegurarse de que está incluido todo lo necesario. A medida que adquiera más experiencia podrá reducir el tamaño de los archivos JAR, ya que le bastará con incluir en ellos y en el proyecto las clases y dependencias necesarias.

Distribución rápida

Los pasos de distribución varían dependiendo de lo que se vaya a distribuir. Estos pasos describen el modo de distribuir aplicaciones, applets y JavaBeans.

Consulte

- ["Utilización del Creador de recopilatorios" en la página 16-16](#)

Aplicaciones



- 1 Añada todos los archivos de recursos y clases cargadas dinámicamente que necesite la aplicación. Si no tiene activada la recopilación automática de paquetes fuente, añádalos al proyecto mediante el botón Añadir archivos/paquetes/clases; de la barra de herramientas del panel del proyecto.

Optativo: utilice el asistente para extracción de recursos, disponible en las ediciones Developer y Enterprise de JBuilder, para desplazar las cadenas a un conjunto de recursos. Esto es optativo, pero facilita la internacionalización de las aplicaciones.

- 2 Compile la aplicación.
- 3 Utilice el Asistente para Javadoc de JBuilder, disponible en las ediciones Developer y Enterprise, para crear la documentación adecuada o, bien, utilice otro medio de creación de documentación para los clientes desarrolladores.
- 4 Cree un archivo JAR mediante la herramienta **jar** de Sun, el Creador de compiladores o el Creador de ejecutables nativos de JBuilder.
- 5 Copie el archivo JAR en el servidor de destino o en los directorios de instalación.
- 6 Prepare un procedimiento de instalación que cree las carpetas y subcarpetas necesarias en el ordenador del usuario final y coloque los archivos en estas carpetas. Este procedimiento también debe modificar la variable de entorno `CLASSPATH` de la forma necesaria en el entorno del usuario final, para especificar la `CLASSPATH` (vía de acceso a clases) correcta para localizar las clases de Java.

Nota Si está creando un ejecutable o un tipo de compilador JAR ejecutable con el Creador de compiladores o con el Creador de ejecutables nativos, puede personalizar el procedimiento de instalación modificando el archivo de configuración que inicia el ejecutable.

Nota Si es necesario que los usuarios dispongan de determinadas clases de Java o compiladores instalados en el ordenador, es posible que deba proporcionarles una forma cómoda de descargar estos archivos y añadirlos a la *vía de acceso a clases*.

- 7 Indique a los usuarios cómo iniciar la aplicación (por ejemplo, haciendo doble clic en un icono que ha suministrado, o ejecutando un script de shell en la ventana del terminal).

Consulte

- [“Recopilación automática de paquetes fuente” en la página 6-38](#)
- Paso 16 del tutorial de JBuilder “Creación de un editor de texto de Java”, en *Diseño de aplicaciones con JBuilder*

Applets

Dado que la aceptación de JDK es variable, la creación y la distribución de applets puede ser bastante complicada. Los siguientes pasos se pueden utilizar como directrices generales. Si desea obtener más información sobre los applets y navegadores, consulte *Guía del desarrollador de aplicaciones web*.



- 1 Añada al proyecto todos los archivos de recurso y clases de carga dinámica que necesita la aplicación, por medio del botón Añadir archivos/paquetes/Clases; de la barra de herramientas del panel de proyecto.

Optativo: utilice el asistente para extracción de recursos, disponible en las ediciones Developer y Enterprise de JBuilder, para desplazar las cadenas a un conjunto de recursos. Esto es optativo, pero facilita la internacionalización de aplicaciones.

- 2 Compile el applet.
- 3 Cree un archivo JAR o ZIP con la herramienta **jar** de Sun, una utilidad ZIP o el Creador de recopilatorios de JBuilder.
- 4 Añada el atributo `archive` con el nombre del archivo JAR dentro de las etiquetas `<applet>` en el archivo HTML del applet. Si tiene varios archivos JAR, inclúyalos separados por comas: `archive="file1.jar, file2.jar, file3.jar"`. Algunos navegadores anteriores no admiten listas múltiples y solamente aceptan archivos ZIP.
- 5 Compruebe la precisión de los valores `codebase` y `code`.

El atributo `codebase` indica la ubicación de las clases en relación con la ubicación del archivo HTML.

- Si al atributo `codebase` se le asigna un valor de `" "`, `"."`, o `"./"`, las clases deben encontrarse en el mismo directorio que el archivo HTML.
- Si las clases pertenecen a un paquete, deben encontrarse en un subdirectorio del directorio del archivo HTML que coincida con la estructura del paquete.
- Cuando las clases se encuentran en el mismo directorio que el archivo HTML, el atributo `codebase` es optativo, ya que es el predeterminado.
- Si las clases se encuentran en un directorio diferente que el archivo HTML, el atributo `codebase` es necesario y debe especificar la ubicación del directorio de los archivos de la clase en relación con el directorio del archivo HTML.

El valor `code` debe ser el nombre de clase completo del applet: `<nombre de paquete> + <nombre de clase>`.

- 6 Inicie el navegador web y abra su archivo local HTML para la prueba inicial.
- 7 Copie el recopilatorio o el archivo de distribución configurado en el servidor de destino.

- 8 Compruebe que todas las mayúsculas y minúsculas de los nombres de clases, paquetes, archivos recopilatorios y directorios de la etiqueta `<applet>` coinciden exactamente con los nombres del servidor.
- 9 Elimine la vía de acceso a clases para que no queden bibliotecas ni JDK en su vía. Debe suponer que los usuarios no cuentan con ninguna de estas en sus vías de acceso a clases.
- 10 Pruebe por medio del servidor web: dirija el navegador a la URL que representa el archivo HTML del servidor de destino.

Nota

Asegúrese de copiar el archivo HTML adecuado en la carpeta de destino y de colocar el archivo JAR en la misma carpeta. Los proyectos de JBuilder pueden contener varios archivos HTML. Si hay más de un archivo HTML, elija el que contenga la pestaña `<applet>`.

Optativo: si es necesario que los usuarios dispongan de determinadas clases de Java o recopilatorios instalados en el ordenador, es posible que deba proporcionarles una forma cómoda de descargar estos archivos y añadirlos a la vía de acceso a clases local.

Importante

Los navegadores que cumplen las especificaciones de JDK 1.0.2 no aceptan archivos recopilatorios JAR. Asegúrese de que crea en su lugar un archivo ZIP.

Consulte

- Paso 7 del tutorial de JBuilder, "Creación de un applet", en *Introducción a JBuilder*
- Si desea obtener más información sobre las applets, consulte "Las applets" de la *Guía del desarrollador de aplicaciones web* Guía del desarrollador de aplicaciones web

JavaBeans



- 1 Añada todos los archivos de recursos y clases cargadas dinámicamente que necesite la aplicación. Si no tiene activada la recopilación automática de paquetes fuente, añádalos al proyecto mediante el botón Añadir archivos/paquetes/clases; de la barra de heramientas del panel del proyecto.

Optativo: utilice el asistente para extracción de recursos, disponible en las ediciones Developer y Enterprise de JBuilder, para desplazar las cadenas a un conjunto de recursos. Esto es optativo, pero facilita la internacionalización de los beans.

- 2 Compile los beans.
- 3 Utilice el Asistente para Javadoc de JBuilder, disponible en las ediciones Developer y Enterprise, para crear la documentación adecuada, o bien, utilice otro medio de creación de documentación para los clientes desarrolladores.

- 4 Cree un archivo JAR con la herramienta **jar** de Sun, una utilidad ZIP o el Creador de compiladores de JBuilder. La documentación se puede incluir en el compilador si se trata de una versión de desarrollo, u omitirla si es una versión redistribuible. Si desea incluir la documentación en el compilador, asegúrese de que se trata de un nodo de proyecto añadiéndolo al proyecto antes de la distribución.
- 5 Suministre a sus clientes los archivos .jar.

Nota Si el bean necesita alguna biblioteca de JBuilder, consulte la sección [“Redistribución de las clases que se suministran con JBuilder” en la página 16-13.](#)

Consulte

- [“Recopilación automática de paquetes fuente” en la página 6-38](#)

Sugerencias de distribución

Consejos básicos para que el proceso de distribución resulte satisfactorio:

- Incluya las imágenes en un subdirectorio (generalmente denominado `images`) que esté asociado a las clases que las utilizan. Haga lo mismo con todos los demás archivos de recursos.
- Utilice sólo vías de acceso relativas (por ejemplo, `images/logo.gif`) para acceder a los archivos de imagen o de otro tipo que utilice la aplicación o el applet.
- Utilice paquetes, por pequeño que sea el applet o la aplicación.

Configuración del entorno de trabajo

La clave para combinar el desarrollo y la distribución consiste en gestionar la colocación de los archivos. Normalmente, lo mejor es conservar los archivos de código fuente destinados a la distribución en un nivel distinto del dedicado a las herramientas de desarrollo. Esto ayuda a prevenir la eliminación o modificación accidental de elementos insustituibles.

El proceso de distribución le puede resultar más fácil si configura un entorno de trabajo que refleje las condiciones reales que tendrá el applet o la aplicación una vez distribuida, con lo que el desarrollo se acerca un poco más a la distribución instantánea. Comience el proyecto con el Asistente de JBuilder porque se encarga de colocar todo en su lugar. Una vez que el proyecto se ha creado y se está ejecutando, puede crear un archivo JAR con la ayuda de la herramienta **jar** de Sun o del Creador de compiladores de JBuilder. Después de crear el compilador, puede probar el programa en la línea de comandos. Asegúrese de comprobar sus applets con todos y cada uno de los navegadores que necesita utilizar.

Sugerencia Si se está creando un applet, se puede utilizar una copia de la página HTML que se encuentra en el proyecto local, en vez de emplear una página

experimental, más sencilla. Esto aumenta el realismo de la prueba externa. Cuando esté satisfecho, envíe todo el directorio a la página Web.

Distribución en Internet

Si distribuye el programa en una dirección remota por medio de FTP (como un proveedor de Internet), el procedimiento básico de la distribución sigue siendo el mismo. No obstante, debe utilizar una utilidad de FTP para transferir los archivos, siguiendo las instrucciones del proveedor del espacio Web.

Importante No olvide transferir los archivos recopilatorios y de clase como archivos binarios. Si la transferencia a la página de Internet se efectúa de forma incorrecta se provocan excepciones `java.lang.ClassFormatError`.

A menudo, la estructura de directorios de la página, vista desde FTP, no es la misma que la de la URL a la que acceden los usuarios. Pregunte a su proveedor de Internet dónde está el directorio raíz de su página y cómo puede transferir archivos hasta ella por medio de FTP. La mayoría de los programas de FTP, tanto comerciales como de shareware, permiten crear directorios además de copiar archivos, por lo que todos los pasos detallados anteriormente deberían poder realizarse, aunque con distintos mecanismos de transferencia de archivos.

Consulte

- “Distribución de aplicaciones web” en *Guía del desarrollador de aplicaciones web*

Distribución de aplicaciones distribuidas

Cuando se distribuyen aplicaciones distribuidas, el Creador de recopilatorios coloca los stubs y esqueletos en un archivo JAR. Se debe instalar el ORB en cada máquina que ejecute un programa CORBA servidor, cliente o de nivel medio. Si utiliza el ORB VisiBroker, consulte el apartado sobre distribución en la documentación de Borland Enterprise Server o en la del servidor de aplicaciones.

Redistribución de las clases que se suministran con JBuilder

Los archivos JAR de libre distribución de JBuilder se encuentran en el directorio `<jbuilder>/redist/`. Los archivos recopilatorios de libre distribución del JDK se encuentran en el directorio `<jbuilder>/<jdk>/lib/`.

Si se crea el recopilatorio con el Creador de recopilatorios de JBuilder, se detectan todos los archivos de recursos, clases y bibliotecas necesarios. El Creador de recopilatorios nunca incluye JDK en el archivo recopilatorio. Presupone que las clases JDK ya existen en el ordenador de destino en forma de JDK instalado, entorno de ejecución Java o Plug-in Java, o que se proporcionarán en la instalación.

Importante No obstante, en Java 1.1.1, las clases Swing/JFC no forman parte del núcleo del JDK y el Constructor de compiladores no las detecta. Si escribe programas que utilizan estos JDK, debe descargar y entregar `swingall.jar`.

Cuando se distribuyen applets no es necesario entregar las clases JRE de JDK, porque el navegador las facilita durante la ejecución. No obstante, es necesario asegurarse de que la versión del JDK que utiliza el applet coincide con la versión que utiliza el navegador. En una intranet, puede utilizar el plug-in Java de Sun para que le proporcione el JDK actual.

Nota Normalmente, cuando se ejecuta un applet, en `CLASSPATH` sólo están las clases de Java. Si hay una biblioteca de terceros en `CLASSPATH` en forma de clases o como compilador, y algunas de estas clases también se distribuyen en el applet o en el archivo JAR, es preferible la copia de `CLASSPATH`, ya que el cargador de clases System puede cargarla. El primer elemento de la lista es suficiente para la MV, que no toma en consideración el segundo.

Si desea información sobre los elementos de libre distribución y los de uso exclusivo, según las condiciones de la licencia de JBuilder, consulte los archivos `<jbuilder>/license.txt` y `<jbuilder>/redist/deploy.txt`.

Consulte

- Para obtener más información sobre los applets, los problemas con navegadores y la compatibilidad con JDK, consulte la sección "Problemas con los navegadores" incluida en *Guía del desarrollador de aplicaciones web*.
- Java Runtime Environment - "Notes for Developers" (en inglés), en <http://java.sun.com/j2se/1.4/runtime.html>.

- "JAR Files Trail" (en inglés) en el tutorial de Java en <http://java.sun.com/docs/books/tutorial/jar/>
- "JAR Guide" (en inglés), en <http://java.sun.com/j2se/1.4/docs/guide/jar/jarGuide.html>

Información adicional de distribución

Puede encontrar información adicional en las siguientes URL:

- Java Runtime Environment - "Notes for Developers" (en inglés), en <http://java.sun.com/j2se/1.4/runtime.html>
- "Trail: Writing Applets", en el Java Tutorial, que discute las consideraciones básicas de las applets como la seguridad, en <http://java.sun.com/docs/books/tutorial/applet/index.html>
- "Trail: Security in Java 2 SDK 1.2", en el Java Tutorial, que discute las API de seguridad general en <http://java.sun.com/docs/books/tutorial/security1.2/index.html>
- "JAR Files Trail" (en inglés) en el tutorial de Java en <http://java.sun.com/docs/books/tutorial/jar/>
- "Understanding the Manifest" (en inglés) en <http://java.sun.com/docs/books/tutorial/jar/basics/manifest.html>
- "JAR Guide" (en inglés), en <http://java.sun.com/j2se/1.4/docs/guide/jar/jarGuide.html>
- Guías y tutoriales para desarrolladores de Sun en <http://developer.java.sun.com/developer/onlineTraining/index.html>
- "Writing advanced applications" (en inglés); se resumen los problemas y soluciones relacionados con la coexistencia de varias versiones de la plataforma Java en el sistema; <http://developer.java.sun.com/developer/onlineTraining/Programming/JDCBook/version.html>
- "The Extension Mechanism Trail" (en inglés) del tutorial de Java, en <http://java.sun.com/docs/books/tutorial/ext/index.html>. Explicación de las nuevas clases de ampliación de Java 1.2 que ponen las API personalizadas a disposición de todas las aplicaciones que se ejecutan en la plataforma Java. El mecanismo de ampliación permite al entorno de ejecución detectar y cargar las clases de ampliación sin necesidad de mencionar las clases de ampliación en la vía de acceso a clases.

Utilización del Creador de recopilatorios

Las funciones disponibles varían según la edición de JBuilder.

El Creador de recopilatorios agiliza el proceso de creación de la distribución. Busca en el proyecto clases y recursos y da la oportunidad de personalizar el contenido del archivo JAR antes de recompilar los archivos en un archivo JAR con el archivo descriptor adecuado.

El Creador de recopilatorios y los recursos

El Creador de recopilatorios reconoce automáticamente determinados tipos de archivos como recursos, tal y como se especifica en la ficha Proyecto|Propiedades de proyecto|Generar|Recursos. JBuilder, al compilar, copia recursos tales como archivos de imagen, sonido y propiedades de la vía de acceso a fuentes en la vía de salida. La vía de salida, que se encuentra en la ficha Vías de acceso del cuadro de diálogo Propiedades de proyecto, contiene los archivos `.class` que JBuilder crea al compilar un programa. Para más información consulte [“Cómo construye JBuilder las vías de acceso” en la página 4-9](#).

Importante

Si existe código que llama a un nombre de clase o recurso como parámetro, el compilador convierte este parámetro en `classForName(String)` y la llamada se realiza durante la ejecución. El Creador de recopilatorios no puede determinar qué recursos son necesarios en este caso. Es necesario añadir los recursos al proyecto de forma manual antes de ejecutar el proceso de archivo, con el fin de indicar al Creador de recopilatorios que debe incluirlos.

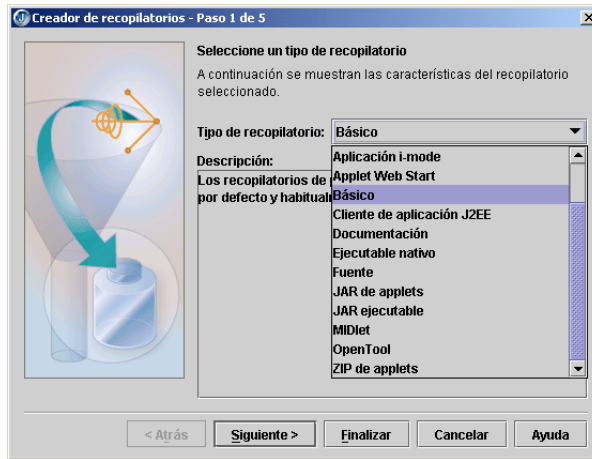
Si desea ver una lista de la configuración por defecto según las extensiones de archivo, consulte la ficha Recursos de la ficha Generar. Puede cambiar la configuración por defecto de JBuilder y especificar los tipos de archivos individuales o de extensiones de archivos que desea que se copien en la vía de salida al compilar. Consulte [“Copia selectiva de los recursos” en la página 6-42](#).

Si en el proyecto hay tipos de archivos que JBuilder no reconoce, puede añadirlos como archivos de recursos genéricos y, a continuación, especificar que se copien en la vía de salida. Si desea más información, consulte [“Adición de tipos de archivos no reconocidos como archivos de recursos genéricos” en la página 6-44](#).

Seleccione un tipo de recopilatorio

El primer paso del Creador de recopilatorios permite seleccionar el tipo de recopilatorio que se desea crear: Applet JAR, Applet ZIP, Aplicación, Básico, Documentación, JAR ejecutable y Ejecutable nativo, además de otros tipos. Según la opción elegida, se definen diferentes valores por defecto y se encuentran disponibles diferentes opciones en los pasos del asistente. Si

desea obtener más información sobre otros tipos de recopilatorios disponibles, seleccione el botón de ayuda del asistente.



Para empezar a crear un recopilatorio:

- 1 Elija Asistentes\Creador de recopilatorios para abrir el Creador de recopilatorios: el Creador de recopilatorios está disponible en la ficha Generar de la galería de objetos (Archivo\Nuevo\Generar).
- 2 Seleccione un tipo de recopilatorio en la lista desplegable Tipo de recopilatorio. Si desea obtener más información sobre los tipos de recopilatorios, seleccione el botón de ayuda del asistente.
- 3 Haga clic en Siguiente para avanzar al paso siguiente del asistente.

Indique el archivo que va a crear el proceso de recopilación

El nombre de este paso y las opciones disponibles cambian según el tipo de recopilatorio seleccionado. Si se selecciona el tipo de recopilatorio Aplicación Web Start o Applet Web Start, característica de las ediciones Developer y Enterprise de JBuilder, el nombre del paso es Seleccione aplicación web. Si se selecciona el tipo de recopilatorio JAR ejecutable, el nombre del paso es Especifique el recopilatorio que se va a utilizar.

Este paso del Creador de recopilatorios permite hacer lo siguiente:

- Definir un nombre para el archivo y el nodo de recopilatorio.
- Seleccionar la compresión del recopilatorio.
- Decidir la frecuencia con que se genera el recopilatorio.

En el caso de los recopilatorios de tipo documentación o fuente, este es el último paso.

- Si el tipo de recopilatorio es Aplicación Web Start o Applet Web Start, es necesario que seleccione también una aplicación web o WebApp definida en el proyecto.

Para poder acceder al recopilatorio Web Start, el JAR debe estar en un directorio de aplicación web.

- Si el tipo de recopilatorio es JAR ejecutable, es necesario que seleccione el recopilatorio fuente para el ejecutable que desee crear.

Aunque este paso cambia ligeramente según el tipo de recopilatorio que se seleccione, para la mayoría de los tipos de recopilatorios puede seguir estos pasos básicos:

1 Asigne un nombre al nodo de recopilatorio en el campo Nombre.

El nodo de recopilatorio se muestra en el panel de proyecto tras finalizar el asistente, pero el recopilatorio no se crea realmente hasta que se ejecute el make o se vuelva a generar el nodo de recopilatorio. Puede hacer clic con el botón derecho del ratón en el nodo y crearlo o volver a generarlo en cualquier momento, así como restablecer sus propiedades. Si desea más información sobre el nodo de recopilatorios, consulte [“Los nodos de recopilatorios” en la página 16-29](#).

2 Escriba el nombre completo del archivo y de la vía de acceso del recopilatorio que genera el Creador de recopilatorios.

Pulse el botón de puntos suspensivos (...) para desplazarse hasta un nuevo directorio.

En el caso del tipo de recopilatorio JAR ejecutable, seleccione un recopilatorio fuente existente para el ejecutable que desea crear.

Nota Sólo los navegadores JDK 1.1 o posteriores aceptan los archivos JAR. Si está desarrollando un applet para un navegador JDK 1.0.2 debe utilizar un archivo comprimido ZIP.

3 Tipos de recopilatorios Aplicación Web Start o Applet Web Start (JBuilder Developer o Enterprise): seleccione una aplicación web (WebApp) definida en el proyecto.

4 Active o desactive la opción Comprimir el contenido del recopilatorio. Normalmente, se suele dejar esta opción desactivada, de modo que el recopilatorio no se comprime y el tiempo necesario para cargarlo se reduce.

Si el recopilatorio es un applet, esta opción aparece seleccionada por defecto. Debido a que la compresión hace que los archivos recopilatorios se vuelvan más pequeños, esto posibilita que los applets se descarguen en Internet o en las intranets de forma más rápida.

5 Active o desactive la opción Crear siempre un recopilatorio al generar el proyecto.

Esta opción determina la frecuencia con que se crea el archivo recopilatorio. Esta opción se encuentra activada por defecto para todos los tipos de recopilatorio. Si está activada, el archivo recopilatorio se volverá a crear cada vez que se cree (make) o genere (build) el proyecto.

- 6 Pulse Siguiente para pasar al siguiente paso o bien, pulse Finalizar si este es el último paso.

Si este es el último paso, genere el recopilatorio tal y como se describe en [“Generación de archivos recopilatorios” en la página 16-28](#).

El tercer paso del asistente, si lo hay, varía según el tipo de recopilatorio que se esté creando:

- La mayoría de tipos de archivos JAR
- Cliente de aplicación J2EE
- Adaptador de recursos (RAR)

Consulte

- [“Los nodos de recopilatorios” en la página 16-29](#)
- Pulse el botón Ayuda en el asistente

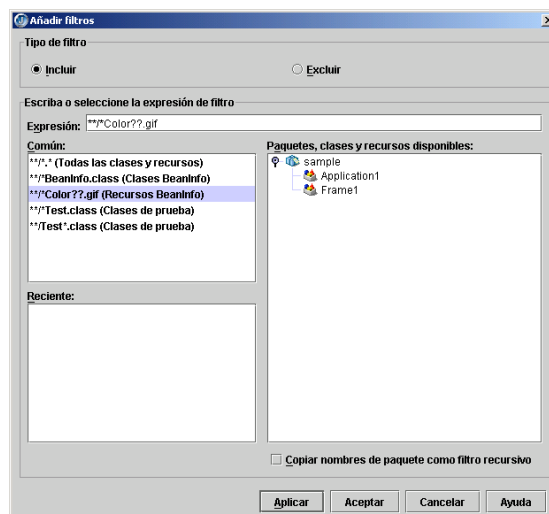
Indique los contenidos del recopilatorio

En este paso del Creador de recopilatorios se eligen las partes del proyecto que se van a incluir en el recopilatorio. Se puede aceptar la configuración por defecto, que incluye todos los elementos del proyecto, o se pueden elegir paquetes, clases, archivos y recursos determinados. También es posible crear filtros personalizados que incluyan o excluyan determinados elementos.

Si desea incluir las clases de dependencias en el archivo recopilatorio, active Incluir dependencias de clases, en la parte inferior de esta ficha del asistente.

Añadir filtros

El botón Añadir filtros abre el cuadro de diálogo del mismo nombre, donde se pueden definir los filtros que debe usar el Creador de recopilatorios.



Especificación de filtros

Los filtros pueden asegurar que los elementos se introducen en el recopilatorio (filtros de tipo Incluir) o asegurar que se mantienen fuera de él (filtros de tipo Excluir.) Los filtros del Creador de recopilatorios admiten comodines, y permiten crear patrones específicos de filtrado.

- 1 Seleccione Incluir o Excluir para el tipo de filtro.
- 2 Escriba una expresión de filtro o seleccione una de las comunes en la lista que aparece a continuación.

La sintaxis de un filtro es un patrón de coincidencia de archivos que admite *, ** y ? como comodines:

- * - coincide con 0 o más caracteres
- ** - coincide con todos los directorios a partir de ese punto
- ? - coincide con un solo carácter

Todos los filtros se aplican de forma relativa, en la vía de salida de archivos generados del proyecto de JBuilder.

Ejemplos de filtros típicos:

<code>**/*Test*.class</code>	Filtro para todas las clases de prueba del proyecto
<code>**/*BeanInfo.class</code>	Filtro para todas las clases BeanInfo
<code>**/*Color?.gif</code>	Filtro para todas las imágenes BeanInfo
<code>com/mycompany/ejb/interfaces/*.class</code>	Filtro para las interfaces, stubs y esqueletos de EJB
<code>com/mycompany/ejb/server/*.class</code>	Filtro para clases EJB sólo de servidor

Los filtros sólo funcionan correctamente si se observan las convenciones de escritura de código correspondientes. Por ejemplo, el primer filtro mencionado arriba sólo funciona correctamente si todos los nombres de las clases de prueba y sólo ellos empiezan por "Test".

En los ejemplos anteriores no se ha indicado si los filtros son de exclusión o de inclusión. Según la situación, un filtro determinado se puede utilizar para incluir o para excluir archivos. En el Paso 3 del Creador de recopilatorios los filtros cuentan con iconos que distinguen los filtros Incluir de los filtros Excluir.

Importante

La exclusión tiene preferencia sobre la inclusión. Nunca se incluyen los archivos que coinciden con una pauta de exclusión, aunque también coincidan con un filtro de inclusión. Una vez que se excluye, no se puede volver a incluir algo con coincida con el patrón de exclusión. Por ejemplo, si se excluyen todos los archivos cuyo nombre acaba en "Test", pero se incluyen las clases que coincidan con el patrón `**/*.class`, las clases que coincidan con `**/*Test.class` no se incluyen con los otros archivos `.class`.

Copiar nombres de paquete como filtro recursivo

Para copiar nombres de paquetes como expresiones de filtro recursivo, seleccione un paquete de la lista Paquetes, clases y recursos disponibles y haga clic la casilla de selección en la parte inferior del cuadro de diálogo Añadir filtros. Esta casilla de selección hace que el campo Expresión alterne entre `/*.*` y `/**/*.`

Por ejemplo, si se selecciona un paquete llamado `com.borland` aparecerá `com.borland/*.*` o `com.borland/**/*.` pegado en la expresión, según se haya activado o desactivado esta opción.

- Si elige `com.borland/*.*`, sólo las clases de `com.borland` coinciden con el filtro.
- Si se selecciona `com.borland/**/*.`, las clases de `com.borland` y de cualquier otro paquete que haya bajo `com.borland`, coinciden con el filtro. Todos los paquetes se procesan recursivamente a partir de ese punto.

Pulse el botón Aplicar para mantener abierto el cuadro de diálogo y añadir otro filtro. A medida que se añaden filtros, se van añadiendo a la lista Recientes, en la parte inferior del cuadro de diálogo, donde se pueden volver a seleccionar. Pulse Aceptar cuando termine.

Modificación y eliminación de filtros

Para modificar un filtro, selecciónelo en la lista Filtros y archivos necesarios de la ficha Indique los contenidos del recopilatorio del Creador de recopilatorios. Pulse el botón Edición para abrir el cuadro de diálogo Modificar filtro. Este cuadro de diálogo funciona exactamente igual que el cuadro de diálogo Añadir filtros.

Si desea eliminar un filtro, selecciónelo en la lista Filtros y archivos necesarios y pulse Eliminar.

Adición de archivos

Para especificar los archivos que se deben incluir explícitamente en el recopilatorio, pulse el botón Añadir archivos. Esto abre un buscador de archivos que admite selección múltiple. Seleccione el archivo o archivos que desee incluir. Pulse Aceptar cuando haya terminado.

El botón Añadir archivos permite incluir archivos de cualquier ubicación, y definir la vía de acceso y el nombre de destino. Esto significa que el archivo `foo/goo/abc.txt` se puede incluir en el recopilatorio como `bz/def.txt`, por ejemplo.

Haga clic en Siguiente para avanzar al paso siguiente del asistente.

Seleccione los archivos descriptores de la distribución

El tipo de recopilatorio Cliente de aplicación J2EE es una característica de las ediciones Developer y Enterprise de JBuilder.

Si se selecciona el tipo de recopilatorio Cliente de aplicación J2EE, en este paso del Creador de recopilatorios se eligen los archivos estándar del descriptor de distribución (`application-client.xml`) y los archivos específicos que se incluyen en el directorio `META-INF` del recopilatorio. El Cliente de aplicación J2EE debe disponer de un archivo `application-client.xml` y puede tener otros archivos dependientes del servidor.

Para indicar los archivos del descriptor de distribución para un recopilatorio Cliente de aplicación J2EE:

- 1 Pulse el botón Crear descriptor.

Aparece un cuadro de diálogo de navegación, con un nuevo archivo descriptor de distribución `application-client.xml` por defecto.

- 2 Seleccione el directorio superior adecuado y acepte el nombre de archivo por defecto o, bien, escriba otro nombre de archivo para crear un archivo descriptor de distribución vacío.

Más tarde puede modificar este archivo antes de crear el recopilatorio.

- 3 Seleccione el botón Añadir para buscar los archivos del descriptor de distribución y añadirlos a su recopilatorio.
- 4 Haga clic en Siguiente para continuar hasta el siguiente paso del asistente, en el que se especifica el contenido del recopilatorio.

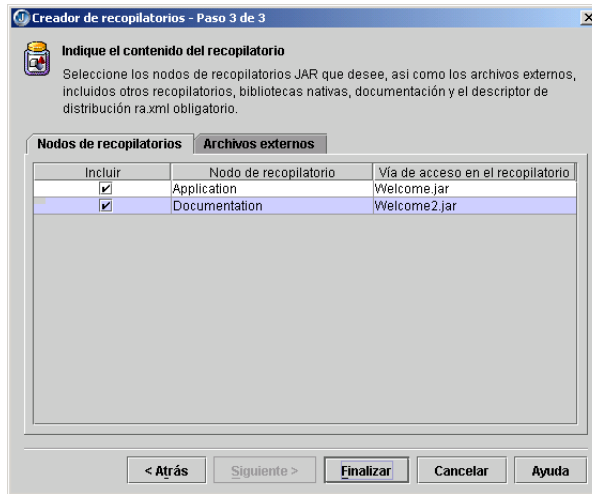
Para obtener más información sobre este paso, pulse el botón Ayuda del asistente.

Especificación del contenido de un recopilatorio de adaptador de recursos

El tipo de recopilatorio Adaptador de recursos (RAR) es una característica de las ediciones Developer y Enterprise de JBuilder.

Si selecciona un tipo de recopilatorio RAR (Adaptador de recursos), en este paso del Creador de recopilatorios ha de elegir los nodos de recopilatorios JAR del proyecto, además de los archivos externos, que desee añadir al recopilatorio. Los archivos externos pueden incluir otros recopilatorios, más

bibliotecas y documentación, y el descriptor de distribución `ra.xml` necesario. El archivo `ra.xml` ya debe estar creado.



Para definir el contenido para un recopilatorio RAR (Adaptador de recursos):

- 1 En la ficha Nodos de recopilatorios, marque todos los nodos de recopilatorios JAR que desee incluir en el recopilatorio.
- 2 Seleccione todos los archivos que desee añadir al recopilatorio, tales como otros recopilatorios, bibliotecas, documentación y el descriptor de distribución `ra.xml` necesario. Seleccione el botón Añadir para buscar los archivos.
- 3 Pulse Finalizar para cerrar el asistente.
- 4 Genere el recopilatorio tal y como se describe en [“Generación de archivos recopilatorios” en la página 16-28](#).

Indique cómo tratar las dependencias de las bibliotecas

En este paso del Creador de recopilatorios se determina cómo tratar las dependencias entre bibliotecas. Las bibliotecas utilizadas en el proyecto se muestran en una lista, y es posible elegir una estrategia de distribución para cada una.

Nota El Creador de recopilatorios nunca incluye JDK en el archivo recopilatorio. Presupone que las clases JDK ya existen en el ordenador de destino en forma de JDK instalado, entorno de ejecución Java o Plug-in Java, o que se proporcionarán en la instalación.

Importante En JDK 1.1.x, las clases Swing/JFC **no** se entregan como parte del JDK. Si escribe programas que utilizan cualquiera de los JDK, debe descargar y entregar `swingall.jar`, que contiene estos archivos.

Para definir las dependencias entre bibliotecas:

- 1 Seleccione una biblioteca de la lista.
- 2 Elija una de las siguientes opciones:
 - Nunca incluir clases ni recursos.
 - Incluir clases necesarias y recursos conocidos.
 - Incluir clases necesarias y todos los recursos.
 - Incluir siempre todas las clases y recursos.

Normalmente, la opción Incluir clases necesarias y todos los recursos es una buena elección para la distribución de bibliotecas. Si desea obtener más información sobre estas opciones, pulse el botón de ayuda del asistente.

- 3 Seleccione otra biblioteca de la lista y una opción de biblioteca.
- 4 Haga clic en Siguiente para avanzar al paso siguiente del asistente.

Importante Recuerde que la Configuración de biblioteca para cada biblioteca se define por separado. Compruebe la configuración en la columna de la derecha y asegúrese de que coincide con lo que desea hacer para cada biblioteca de la columna de la izquierda.

Nota Si distribuye clases del paquete DataStore (`com.borland.datastore`) o el paquete VisiBroker, aparecerá un mensaje que recuerda al programador que la distribución de estos paquetes requiere una licencia de distribución independiente. Si ya dispone de dicha licencia y no desea ver esta advertencia de nuevo en el proyecto, active la casilla "No advertir nuevamente en este proyecto".

Configurar opciones del descriptor del recopilatorio

En este paso del Creador de recopilatorios, se indica a JBuilder el modo de crear un archivo descriptor. En la mayoría de los casos, la opción por defecto, Crear un descriptor, resulta suficiente.

Para configurar las opciones del archivo descriptor del recopilatorio:

- 1 Acepte la opción por defecto, Incluir un descriptor en el recopilatorio, si desea que se incluya un archivo descriptor.
- 2 Seleccione una de estas opciones si desea que se incluya un archivo descriptor:
 - Crear un descriptor.
 - Crear un descriptor y guardar una copia en el archivo.
 - Usar el archivo especificado como archivo descriptor.

Si desea obtener más información sobre estas opciones, pulse el botón de ayuda del asistente.

- 3 Pulse Siguiente para continuar o Finalizar si éste es el último paso del asistente. Si este es el último paso, genere el recopilatorio tal y como se describe en [“Generación de archivos recopilatorios” en la página 16-28](#).

Consulte

- [“Conceptos básicos acerca del archivo descriptor” en la página 16-3](#)

Seleccione un método para indicar la clase principal de la aplicación

En este paso se configura la clase principal de la aplicación. La clase principal es la encargada de ejecutar la aplicación. Contiene el método `public static void main(String[] args)`. Se puede hacer que JBuilder utilice la clase principal que se definió en la configuración de ejecución que desea utilizar, o se puede especificar una clase principal. Si se especifica aquí la clase principal, se utiliza siempre para la aplicación en este recopilatorio, sin tener en cuenta la clase principal establecida en cualquiera de las configuraciones de ejecución con que se ejecuta.

Para configurar la clase principal del recopilatorio:

1 Elija una de las siguientes opciones:

- Determinar la clase principal para configuraciones de ejecución

Esta opción utiliza la clase principal de la configuración de ejecución que se seleccionó en la lista. La lista desplegable incluye todas las configuraciones de ejecución del tipo Aplicación del proyecto.

Seleccione, o bien una configuración de ejecución del proyecto, o <Por defecto>.

<Por defecto> utiliza la configuración de ejecución por defecto del proyecto. Si no existe ninguna o la que hay no es una configuración de ejecución de tipo Aplicación, se utiliza la primera configuración de ejecución de tipo Aplicación de la lista de configuraciones de ejecución.

Nota

Si crea un tipo de recopilatorio ejecutable nativo o JAR ejecutable y especifica una configuración de ejecución en este paso del asistente, se incluyen la clase principal, los parámetros de aplicación y los parámetros de la MV, de esta configuración de ejecución, en el archivo de configuración utilizado para iniciar el ejecutable. Puede personalizar la configuración en el último paso del asistente.

- Usar la siguiente clase

Esta opción utiliza la clase principal especificada y no otra.

Pulse el botón de puntos suspensivos y busque y seleccione una clase.

2 Pulse Siguiente para continuar o Finalizar si éste es el último paso del asistente. Si este es el último paso, genere el recopilatorio tal y como se describe en [“Generación de archivos recopilatorios” en la página 16-28](#).

Para obtener más información sobre este paso, pulse el botón Ayuda del asistente.

- Precaución** Si no se especifica una clase principal, las siguientes tareas no funcionarán:
- Lanzamiento de ejecutables nativos
 - Utilización de `java -jar <jarname>` desde la línea de comandos
 - Doble clic en un recopilatorio JAR

Consulte

- [“Definición de las configuraciones de ejecución” en la página 7-8](#)

Determinar qué ejecutables se van a generar

**Es una función de
JBuilder Developer y
Enterprise.**

El Creador de recopilatorios puede agrupar automáticamente el archivo JAR de la aplicación con englobadores de ejecutables nativos para facilitar su distribución a varias plataformas.

Esta paso se encuentra disponible si ha seleccionado Ejecutable nativo o JAR ejecutable como el tipo de recopilatorio en el Creador de recopilatorios, y también si utiliza el Creador de ejecutables nativos (Asistentes\Creador de ejecutables nativos).

El archivo ejecutable contiene

- El iniciador de ejecutables
- El archivo de configuración para el iniciador de ejecutables
- El archivo JAR con las clases y recursos Java en un solo archivo

El Creador de recopilatorios configura el comentario JAR en el archivo de configuración, y añade el programa de inicio al principio del archivo.

Para crear un archivo ejecutable:

- 1 Seleccione los ejecutables que desea crear. Pulse el botón de puntos suspensivos para cambiar el nombre y/o guardar el archivo en una ubicación diferente.
- 2 Seleccione Siguiente para establecer las opciones de la configuración de ejecución del ejecutable.

Importante Debe especificar una clase principal en el paso anterior o, de lo contrario, no podrá ejecutarlo.

Para obtener más información sobre este paso, pulse el botón Ayuda del asistente.

Ejecución de ejecutables

Observe que el JDK **no** se agrupa con el archivo JAR. Con el fin de que el ejecutable se pueda utilizar, debe haber un JDK adecuado instalado en el equipo del usuario y en la vía de acceso del usuario. Los ejecutables específicos de la plataforma buscan el JDK instalado en la siguiente ubicación:

- Windows: registro.

- Linux/Solaris: variable de entorno `JAVA_HOME` y la vía de acceso del usuario.
- Mac OS X: ubicación predefinida para el JDK.

Nota

Puede redefinir este comportamiento por defecto si especifica la ubicación del JDK en un archivo de configuración personalizado. El archivo ejecutable aparece entonces en la ubicación especificada. Si desea obtener más información sobre los archivos de configuración, consulte el siguiente paso.

Si crea el ejecutable en la plataforma Windows y desea trasladarlo a otra plataforma, puede que necesite cambiar los permisos para convertirlo en ejecutable.

Si selecciona la opción Mac OS X, se crea una aplicación que solamente se puede iniciar desde la línea de comandos. Para crear una aplicación que se pueda iniciar desde el Buscador, los desarrolladores de Mac necesitan crear una agrupación de aplicaciones. Consulte la documentación para el desarrollador de Mac OS X de Apple con el fin de obtener más información acerca de las agrupaciones y paquetes de aplicaciones.

Configuración de las opciones de depuración

Este paso del asistente se encuentra disponible si está creando un JAR ejecutable o un tipo de compilador ejecutable nativo. JBuilder crea automáticamente una configuración ejecutable para iniciar el archivo ejecutable, basándose en la configuración de ejecución o clase principal especificada en el paso del Creador de compiladores. Seleccione un método para indicar la clase principal de la aplicación. Si selecciona una configuración de ejecución en ese paso del asistente, el Creador de compiladores o de ejecutables nativos incluye la clase principal, los parámetros de la aplicación y los parámetros de la MV del archivo de configuración que inicia el ejecutable. Para obtener más información acerca de capturas, consulte [“Definición de las configuraciones de ejecución” en la página 7-8](#).

Si desea personalizar la configuración del ejecutable, puede modificar la configuración que crea JBuilder o crear su propia configuración. Para obtener más información sobre la creación de archivos de configuración, consulte [“Creación de ejecutables con el Creador de ejecutables nativos” en la página 16-31](#).

Para crear una configuración ejecutable para el compilador:

1 Elija una de las siguientes opciones:

- Crear configuración ejecutable

Crea un archivo `<application-name>.config` utilizando las opciones por defecto que se derivan de la información introducida en la fichas anteriores del asistente.

- Crear una Configuración ejecutable y guardar una copia en el archivo

Crea el archivo de configuración por defecto y guarda una copia en otro lugar. Se puede personalizar la copia en cualquier momento y utilizar esa en su lugar.

- Redefinir la Configuración ejecutable por el archivo especificado

El archivo de configuración por defecto también se crea, pero se redefine automáticamente por el archivo de configuración que se especifique aquí. El ejecutable busca un archivo de configuración fuera antes de buscar uno dentro de sí mismo.

Nota Si elige guardar una copia o redefinir la configuración, el archivo de configuración se añade al proyecto.

2 Pulse Finalizar para cerrar el asistente.

3 Genere el recopilatorio tal y como se describe en [“Generación de archivos recopilatorios” en la página 16-28](#).

Además de generar el archivo JAR y las configuraciones de ejecución y ejecutables, el asistente añade automáticamente el ejecutable a la vía de acceso.

Para obtener más información sobre este paso, pulse el botón Ayuda del asistente.

Consulte

- [“Definición de las configuraciones de ejecución” en la página 7-8](#)
- [“Creación de ejecutables con el Creador de ejecutables nativos” en la página 16-31](#), para obtener más información acerca de la creación y personalización de archivos de configuración ejecutables.

Generación de archivos recopilatorios

Al salir del Creador de recopilatorios y del Creador de ejecutables nativos, aparece automáticamente un nodo de recopilatorios en el panel de proyecto. Sin embargo, el archivo recopilatorio no se crea hasta que no se ha generado el nodo de recopilatorios.

El momento de la generación del nodo de recopilatorios viene determinado por la opción elegida en el paso Especifique el archivo que se va a crear del Creador de recopilatorios y del Creador de ejecutables nativos: la opción Crear siempre un recopilatorio al generar el proyecto.

- Si esta opción está activada, el archivo de revisiones se genera cada vez que se elige Proyecto|Ejecutar Make de proyecto o Proyecto|Generar el proyecto.
- Si esta opción está desactivada, puede crear el archivo de revisiones haciendo clic con el botón derecho en el nodo del recopilatorio en el panel de proyecto y eligiendo Ejecutar Make o Generar.

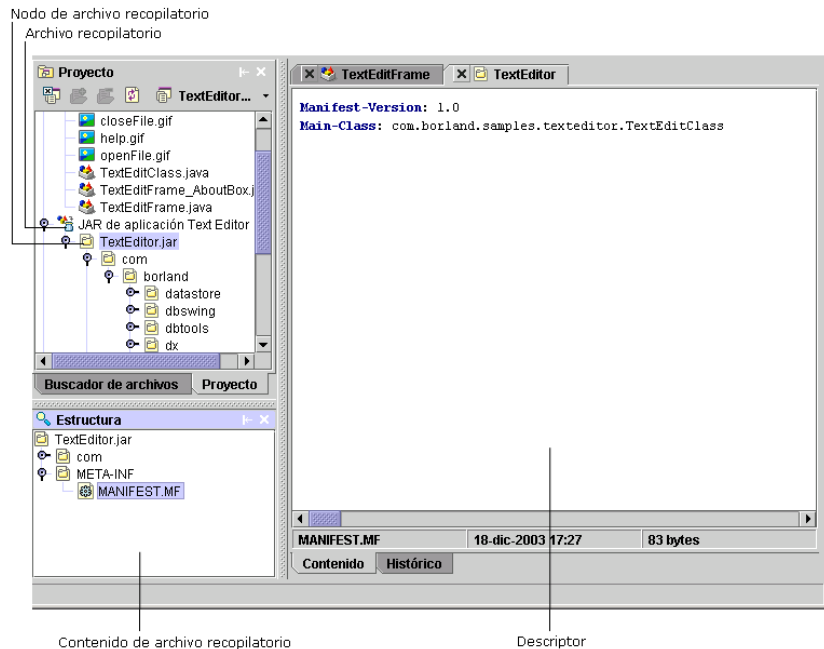
Los nodos de recopilatorios

Con la ayuda del Creador de recopilatorios o del Creador de ejecutables nativos puede crear varios archivos recopilatorios con opciones diferentes para probar varias distribuciones. Primero, utilice el Creador de recopilatorios para incluir diferentes clases, dependencias y recursos en diferentes combinaciones. Luego podrá comparar el contenido de cada archivo de revisiones y descubrir la mejor estrategia para lograr sus objetivos de tamaño, tiempo de descarga e instalación.

El archivo recopilatorio puede crearse o modificarse en cualquier fase del desarrollo. También se puede ver el contenido del archivo recopilatorio y del archivo descriptor.

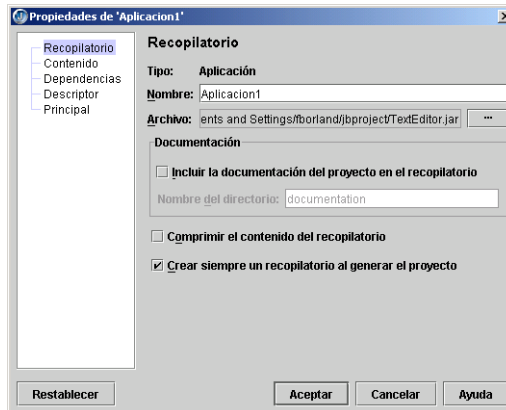
Presentación del archivo recopilatorio y del archivo descriptor

Para ver el archivo de revisiones y el archivo descriptor, expanda el nodo de recopilatorios en el panel de proyecto. Haga doble clic en el archivo de revisiones en el panel de proyecto para mostrar su contenido en el panel de la estructura y el archivo descriptor en el panel de contenido. Haga doble clic en otros archivos del panel de estructura para abrirlos como archivos de sólo lectura en el editor.



Modificación de las propiedades de los nodos de recopilatorios

Durante el desarrollo puede modificar las propiedades del nodo de recopilatorios para cambiar el contenido del archivo de revisiones resultante. Para cambiar las propiedades, haga clic con el botón derecho en el nodo de recopilatorios y elija Propiedades. El cuadro de diálogo Propiedades muestra las fichas correspondientes a los pasos del Creador de recopilatorios y del Creador de ejecutables nativos.



Eliminación, borrado y asignación de nombres a recopilatorios

Después de crear varias ediciones de recopilatorios para su proyecto, puede que desee eliminar, borrar o cambiarle el nombre al que no le interesa. Existen varios modos de llevar esto a cabo, según lo que desee hacer.

Si se elimina el nodo de recopilatorio no se suprime el archivo JAR, pero se elimina a ambos del proyecto.

Si desea eliminar del proyecto el nodo de recopilatorios y su contenido, haga lo siguiente:

- Haga clic con el botón derecho en el nodo de recopilatorio del panel de proyecto y, a continuación, seleccione la opción Eliminar del proyecto.
- Seleccione el nodo de recopilatorio del panel de proyecto y haga clic en el botón Eliminar del proyecto de la barra de herramientas de ese panel.
- Seleccione el nodo de recopilatorios del panel del proyecto y, a continuación, seleccione Proyecto|Eliminar del proyecto.

También se puede eliminar el archivo recopilatorio del proyecto. Esto resulta útil si desea modificar las propiedades del nodo de recopilatorio y crear un archivo recopilatorio nuevo para el proyecto. Pero tenga en cuenta que si ha seleccionado la opción Crear siempre un recopilatorio al generar el proyecto,

el archivo recopilatorio se crea de nuevo al generar el nuevo proyecto. Para ver si esta opción está activada, haga clic con el botón derecho del ratón en el nodo de recopilatorio y, a continuación, seleccione Propiedades. Esta opción de la ficha Recopilatorio está activada por defecto. Después de eliminar el archivo recopilatorio, restablezca las propiedades del recopilatorio, haga clic con el botón derecho en el nodo de recopilatorios y seleccione Ejecutar Make para recrear el recopilatorio con la nueva configuración.

Para borrar el archivo recopilatorio, realice una de las siguientes acciones:

- Pulse el botón derecho sobre el nodo de recopilatorios del panel de proyecto y seleccione Limpiar.
- Expanda el nodo del recopilatorio, haga clic con el botón derecho del ratón sobre el archivo JAR, y escoja Borrar <filename.jar>.

Por último, puede cambiar el nombre de los nodos de recopilatorios y de los archivos.

Para cambiar el nombre de los archivos y nodos de recopilatorios:

- Seleccione el nodo de recopilatorios o el archivo recopilatorio en el panel de proyecto y, a continuación, ProyectoCambiar nombre a.

Creación de ejecutables con el Creador de ejecutables nativos

**Es una función de
JBuilder Developer y
Enterprise.**

El Creador de ejecutables nativos agrupa automáticamente un archivo JAR de aplicaciones con englobadores nativos para Windows (GUI y consola), Linux, Solaris y Mac OS X.

El Creador de ejecutables nativos, disponible en el menú Asistentes y en la ficha Generar de la galería de objetos, es una forma de acceso abreviado a este tipo de recopilatorio del Creador de ejecutables. Si desea más detalles acerca del Creador de recopilatorios, consulte [“Utilización del Creador de recopilatorios” en la página 16-16](#).

El Creador de ejecutables nativos incluye estos pasos, que son los mismos que los de los tipos JAR ejecutable y Ejecutable nativo del Creador de recopilatorios:

- Definir el archivo que se ha de crear
- Especificar las partes del proyecto que se van a recopilar
- Determinar las dependencias entre bibliotecas
- Configurar las opciones del archivo descriptor del recopilatorio
- Seleccionar un método para indicar la clase principal de la aplicación
- Determinar los archivos ejecutables que se van a crear
- Configuración de las opciones de depuración

Importante

Los ejecutables nativos **tienen** que tener una clase principal especificada.

Una vez que ha terminado de utilizar el asistente, pulse con el botón derecho del ratón en el nodo de archivos ejecutables nativos del panel de proyecto y, a

continuación, seleccione Ejecutar Make para crear los ejecutables y el archivo JAR. Amplíe el nodo para ver el archivo JAR creado y los ejecutables. Si desea modificar las propiedades de este nodo, pulse con el botón derecho del ratón y seleccione Propiedades.

Nota El JDK no se agrupa con el archivo JAR. Con el fin de que el ejecutable se pueda utilizar, debe haber un JDK adecuado instalado en el equipo del usuario y en la vía de acceso del usuario.

Consulte

- [“Los nodos de recopilatorios” en la página 16-29](#)

Personalización de archivos de configuración ejecutables

Si está creando ejecutables con el Creador de compiladores o con el Creador de ejecutables nativos, puede redefinir el archivo de configuración por defecto que hayan creado estos asistentes. Para ello, cree o modifique un archivo de configuración ejecutable.

Los archivos de configuración proporcionan flexibilidad y personalización en el inicio de aplicaciones y utilidades. Por ejemplo, se pueden utilizar para pasar parámetros a la Máquina Virtual de Java (MV), pasar parámetros a OpenTools, depurar OpenTools y personalizar el inicio de las aplicaciones.

El *programa de inicio* de la aplicación contiene los ejecutables adecuados y/o los iconos de escritorio y se utiliza para iniciar la aplicación. Antes de iniciarla, busca un *archivo de configuración* para obtener instrucciones adicionales. Un archivo de configuración es un archivo de texto con una lista de directivas que distinguen entre mayúsculas y minúsculas y que tienen que ejecutarse antes de iniciar el ejecutable.

Una vez encontrado el archivo de configuración, el programa de inicio procesa cada línea de texto de forma secuencial antes de que se cargue la MV de Java. Si el programa de inicio no encuentra un archivo de configuración, se abre como un archivo y busca el archivo de configuración que tiene almacenado como un comentario zip. El programa de inicio declara un error y termina en cualquiera de las siguientes circunstancias:

- No puede leer el archivo de configuración.
- Una línea contiene una directiva desconocida.
- Falta la directiva `mainclass`.
- La MV Java no se inicia.

También se declara un error si se omite la directiva `javapath` y no se puede determinar una ubicación por defecto de la MV de Java.

Las directivas del archivo de configuración pueden especificar la clase principal, añadir archivos JAR, pasar parámetros a la MV, añadir una entrada a la vía de acceso a clases Java, etc. Los archivos de configuración también pueden incluir otros archivos de configuración. Por ejemplo, el programa de inicio de JBuilder se refiere al archivo `jbuilder.config`, que a su vez utiliza la

directiva `include` para referirse a `jdk.config`, tal y como se muestra en el siguiente ejemplo.

Ejemplo de archivo de configuración

```
# Leer la definición del JDK compartido
include jdk.config

# Ajustar esta MV para que ofrezca espacio suficiente para trabajar con
grandes
# applications
vmparam -Xms32m
vmparam -Xmx128m

# Colocar el Englobador AWT ligero en la vía de acceso de inicio
addbootpath ../lib/lawt.jar
addbootpath ../lib/TabbedPaneFix.jar

# Añadir todos los archivos JAR ubicados en el directorio parche, lib y
lib/ext
addjars ../patch
addjars ../lib
addjars ../lib/ext

# Incluir la API del Servlet 2.3 de Tomcat 4 en la vía de acceso a clases
del IDE
addpath ../jakarta-tomcat-4.0.3/common/lib/servlet.jar

# Activar la integración shell
socket 8888

# Añadir todos los archivos de configuración ubicados en el directorio
lib/ext
includedir ../lib/ext

# JBuilder necesita tener acceso al entorno
exportenv

# Iniciar JBuilder mediante la clase principal
mainclass com.borland.jbuilder.JBuilder
```

Al crear ejecutables nativos para sus aplicaciones en el Creador de compiladores o en el Creador de ejecutables nativos de JBuilder, puede personalizar el funcionamiento de inicio de la aplicación con un archivo de configuración en la ficha Ejecutables. Por ejemplo, puede pasar determinados parámetros de ejecución y de la MV a su aplicación antes de que se inicie.

Consulte

- [“Creación de ejecutables con el Creador de ejecutables nativos” en la página 16-31](#)
- [“Utilización del Creador de compiladores” en la página 16-16](#)

Inicio de la MV

El comando de la línea de comandos para iniciar la MV tiene la siguiente forma:

```
<javapath> [-Xbootclasspath/p:<bootpath>]  
[-classpath <classpath>] <vmparams> <mainclass> {params}
```

Los elementos entre <corchetes angulares> representan valores derivados del archivo de configuración, y los que están entre [corchetes] representan las partes opcionales de la línea de comandos que se omiten si los elementos relevantes de la vía de acceso no están definidos en el archivo de configuración. El componente {params} es por defecto un duplicado de los parámetros de la línea de comandos del programa de inicio, pero algunas directivas del archivo de configuración pueden alterarlo.

Requisitos del archivo de configuración

El archivo de configuración debe cumplir determinadas especificaciones para que se pueda analizar correctamente.

Tipo de archivo y ubicación

El archivo de configuración debe ser un archivo de texto sin formato. Debe encontrarse en el mismo directorio que el programa de inicio, y tener el mismo nombre pero con la extensión `.config`. Por ejemplo, `bcj.exe`, ubicado en el directorio `bin` de JBuilder, tiene un archivo de configuración denominado `bcj.config`. Si el programa de inicio no encuentra el archivo de configuración, se abre como un archivo y busca el archivo `.config` que tiene almacenado.

Líneas en blanco y comentarios

Las líneas en blanco y las que empiezan con el carácter `#` se omiten para permitir que el archivo de configuración se estructure y se documente.

Convenciones de las vías de acceso

Todas las vías de acceso del archivo de configuración pueden ser relativas o absolutas. Las vías de acceso relativas son relativas para el directorio que contiene el archivo de configuración y el programa de inicio. Puede utilizar `..` en la vía de acceso para desplazarse hasta el directorio superior. Todos los separadores de las vías de acceso deben ser barras diagonales, sin tener en cuenta el separador de vías de acceso estándar de la plataforma local.

Directivas

En el archivo de configuración se pueden especificar las siguientes directivas. Algunas son necesarias, y otras optativas.

`javapath`

La directiva `javapath` proporciona la ubicación y el nombre exactos del intérprete Java. Si este programa de inicio es un ejecutable o una biblioteca compartida depende de dicho programa. Por ejemplo, un programa de inicio Win32 necesita una de las siguientes:

```
javapath ../jdk14/bin/java.exe
javapath ../jdk14/jre/bin/client/jvm.dll
```

Si no se encuentra una directiva `javapath`, el programa de inicio intenta determinar una ubicación por defecto de la MV de Java de acuerdo con el modo adecuado de esa plataforma.

Los ejecutables específicos de la plataforma buscan el JDK instalado en la siguiente ubicación:

- Windows: registro.
- Linux/Solaris: variable de entorno de `JAVA_HOME` y en la vía de acceso del usuario.
- Mac OS X: ubicación predefinida para el JDK.

Nota

Puede redefinir este comportamiento por defecto si especifica la ubicación del JDK en un archivo de configuración personalizado. El archivo ejecutable aparece entonces en la ubicación especificada. Si desea obtener más información sobre los archivos de configuración, consulte el siguiente paso.

La directiva `javapath` es necesaria si el JDK no está instalado en la ubicación habitual de la plataforma.

mainclass

La directiva `mainclass`, la cual es necesaria, ofrece el nombre completo de la clase que se utiliza para iniciar la aplicación. Por ejemplo:

```
mainclass com.borland.jbuilder.JBuilder
```

addpath

La directiva `addpath` añade una vía de acceso única a la vía de acceso a clases utilizada para iniciar la aplicación. Por ejemplo:

```
addpath ../lib/jbuilder.jar
```

Compruebe cómo se aplican las siguientes reglas adicionales:

- Las vías de acceso que se refieren a un directorio o a un archivo que no existe no se añaden.
- Las vías de acceso que ya están en la vía de acceso a clases, la vía de acceso de inicio y la vía de acceso a excluir no se añaden.
- Las vías de acceso que contengan espacios se colocan automáticamente entre comillas al generar la línea de comandos.

addjars

La directiva `addjars` añade todos los archivos JAR del directorio especificado a la vía de acceso a clases utilizada para iniciar la aplicación. Por ejemplo:

```
addjars ../lib
```

Las mismas reglas que se aplican a la directiva `addpath` se aplican a cada una de las vías de acceso añadidas como resultado de la directiva `addjars`.

addbootpath

La directiva `addbootpath` añade una única vía de acceso a la vía de acceso de inicio utilizada para iniciar la MV de Java. Por ejemplo:

```
addbootpath ../lib/lawt.jar
```

Compruebe cómo se aplican las siguientes reglas adicionales:

- Las vías de acceso que se refieren a un directorio o a un archivo que no existe no se añaden.
- Las vías de acceso que ya se encuentren en la vía de acceso a clases se eliminan de la vía de acceso a clases y se añaden a continuación a la vía de acceso de inicio.
- Las vías de acceso que ya están en la vía de acceso de inicio y la vía de acceso a `excluir` no se añaden.
- Las vías de acceso que contengan espacios se colocan automáticamente entre comillas al generar la línea de comandos.

addbootjars

La directiva `addbootjars` añade todos los archivos JAR del directorio especificado a la vía de acceso de inicio utilizada para iniciar la aplicación. Por ejemplo:

```
addbootjars ../lib
```

Las mismas reglas que se aplican a la directiva *addbootpath* se aplican a cada una de las vías de acceso añadidas como resultado de la directiva *addbootjars*.

addskippath

La directiva `addskippath` define una única vía de acceso que no se debe añadir nunca a las vías de acceso a clases o de inicio utilizadas para iniciar la MV de Java. Esto resulta de gran utilidad para eliminar las vías de acceso individuales que, de otro modo, añadirían las directivas `addjars` o `addbootjars`. Por ejemplo:

```
addskippath ../lib/dbswing.jar
```

Compruebe cómo se aplican las siguientes reglas adicionales:

- La vía de acceso se elimina de la vía de acceso a clases si ya se ha añadido.
- La vía de acceso se elimina de la vía de acceso de inicio si ya se ha añadido.

vmparam

La directiva `vmparam` proporciona parámetros que se pasan directamente a la MV de Java cuando se inicia. Por ejemplo, la siguiente directiva establece los tamaños mínimos y máximos de memoria dinámica en 8MB y 128MB respectivamente:

```
vmparam -Xms8m -Xmx128m
```

Los efectos de esta directiva son acumulativos. Cada uno de los sucesos siguientes se añade al conjunto de parámetros que se pasan a la MV con espacios insertados automáticamente entre los parámetros.

include

La directiva `include` hace que se analice el contenido del archivo antes de continuar con el archivo de configuración actual. Esta directiva no se debe utilizar para anidar archivos de configuración en un número arbitrario de niveles. El programa de inicio declara un error y finaliza si no se puede leer el nombre del archivo.

```
include jdk.config
```

Al igual que todas las vías de acceso del archivo de configuración, la vía de acceso puede ser relativa o absoluta. Consulte [“Convenciones de las vías de acceso” en la página 16-34](#).

includedir

La directiva `includedir` hace que todos los archivos con la extensión `.config` del directorio especificado se procesen como con la directiva `include`.

```
includedir ../lib/ext
```

Al igual que todas las vías de acceso del archivo de configuración, la vía de acceso puede ser relativa o absoluta. Consulte [“Convenciones de las vías de acceso” en la página 16-34](#).

copyenv

La directiva `copyenv` hace que se exponga el contenido de una variable de entorno, definiendo la variable de entorno Java correspondiente.

```
copyenv PROMPT
```

La variable Java tiene un prefijo para evitar conflictos de espacios de nombres, por lo que esta directiva define una variable de entorno Java de nombre `borland.copyenv.PROMPT`, cuyo valor se deriva originariamente de la variable de entorno `PROMPT`.

exportenv

La directiva `exportenv` hace que se exponga el contenido de todas las variables de entorno del sistema, escribiéndolas en un archivo temporal. Cada llamada del programa de inicio crea un archivo único utilizando el formato de archivo de Java `.properties` para representar una colección completa de pares nombre/valor para todas las variables de entorno.

```
exportenv
```

La variable de entorno Java `borland.exportenv` está configurada para contener el nombre del archivo en el que se ha escrito el entorno. Una vez que se cierra la MV de Java, el programa de inicio se encarga de eliminar el archivo temporal.

addparam

La directiva `addparam` añade un nuevo parámetro o conjunto de parámetros al conjunto existente de parámetros de aplicaciones.

```
addparam <param>
```

clearparams

La directiva `clearparams` descarta el conjunto existente de parámetros de aplicaciones. Esta directiva se utiliza normalmente junto con las siguientes directivas `addparam`.

```
clearparams
```

restartcode

La directiva `restartcode` especifica un código de salida para el proceso Java que se debe interpretar como una solicitud para reiniciar el proceso de inicio. Esta directiva es utilizada normalmente por una aplicación que necesita realizar cambios en su configuración, por lo que el programa de inicio debe volver a leer los archivos de configuración como si se iniciase por primera vez.

```
restartcode 22
```

Precaución Esta directiva debe utilizarse con precaución porque puede conducir fácilmente a un ciclo infinito de inicios de máquinas virtuales. Por motivos de seguridad, el código de salida de reinicio 0 nunca se interpreta como una solicitud de reinicio, incluso si se encuentra una directiva explícita `restartcode 0`.



Internacionalización de programas con JBuilder

Es una función de JBuilder Developer y Enterprise.

En este capítulo se tratan los aspectos relacionados con el diseño de aplicaciones de Java destinadas a cumplir las necesidades del público mundial. No hay motivos para limitar el uso de las applets y aplicaciones a un país determinado, cuando, con un poco más de esfuerzo, se podrían utilizar en todo el mundo. Las funciones especiales de JBuilder facilitan el aprovechamiento de la capacidad de internacionalización de Java, que permite personalizar las aplicaciones para los países e idiomas deseados sin necesidad de efectuar trabajosos cambios en el código.

Este capítulo trata sobre funciones especiales de JBuilder y no se ha escrito con el ánimo de examinar a fondo las funciones de internacionalización de Java. Por tanto, si desea más información antes de empezar, consulte la documentación de Java. Antes de emprender la explicación sobre las funciones de internacionalización de JBuilder, repase el siguiente apartado, que trata sobre los términos específicos de la internacionalización.

Términos y definiciones de internacionalización

■ Internacionalización (i18n)	La internacionalización es el proceso de diseño o conversión de programas para su uso en más de una versión localizada. A causa de su longitud, en inglés esta palabra se abrevia a menudo como "i18n", donde 18 representa el número de letras que hay entre la 'i' y la 'n' de "internacionalización".
■ Versión localizada	"Versión localizada" define un conjunto de convenciones culturales específicas para la presentación, el formato y la ordenación de datos. En Java, las versiones localizadas se especifican por medio de un objeto <code>Locale</code> , que es simplemente un contenedor para cadenas que identifican un idioma y un país determinados.
■ Inclusión en recursos	La inclusión en recursos es la parte del proceso de internacionalización que consiste en aislar los recursos específicos de una versión localizada del código fuente en módulos que se pueden añadir o extraer de la aplicación de forma independiente. Algunos ejemplos de recursos específicos de versiones localizadas son el texto que se muestra al usuario, las reglas empresariales y la lógica de la aplicación. Java suministra una serie de clases <code>ResourceBundle</code> destinadas a la inclusión en recursos de cadenas y objetos en programas Java.
■ Localización (l10n)	La localización es la adaptación de los recursos de un programa a un país y un idioma determinados. Observe que la internacionalización generaliza los programas para todos los países e idiomas, mientras que la localización los especializa para zonas concretas. A causa de su longitud, en inglés esta palabra se abrevia a menudo como "l10n", donde 10 representa el número de letras que hay entre la 'l' y la 'n' de "localización".
■ Codificación nativa	La codificación nativa, también conocida como página de códigos y como conjunto de caracteres, determina la redefinición de valores numéricos a caracteres simbólicos dentro de cada sistema operativo. Como las codificaciones nativas varían según los distintos sistemas operativos (y en ocasiones dentro del mismo sistema operativo), un archivo que contenga caracteres de un sistema puede presentar caracteres completamente distintos en otro sistema que utilice una codificación nativa distinta.
■ Unicode	Unicode es una norma de codificación de caracteres que mantiene <code>The Unicode Consortium</code> (http://www.unicode.org). Esta norma define una correspondencia de caracteres para la mayoría de los idiomas del mundo. Se puede especificar cualquier carácter de Unicode en el código fuente de Java por medio de su secuencia de escape de Unicode, <code>\uNNNN</code> , donde <code>NNNN</code> es el valor hexadecimal del carácter en el conjunto de caracteres Unicode. Los caracteres y cadenas se procesan siempre en la máquina virtual de Java como valores Unicode de 16 bits.

Funciones de internacionalización de JBuilder

Entre las funciones de JBuilder hay varias destinadas a facilitar la internacionalización de las applets y aplicaciones Java. En los siguientes apartados se tratan estas funciones:

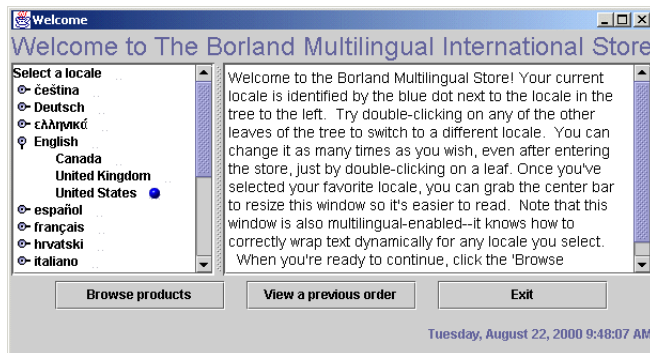
- Aplicación de ejemplo multilingüe
- Eliminación de cadenas no modificables (hard-coded) incluidas en el código
- Funciones de internacionalización de dbSwing
- Componentes que identifican la versión localizada
- Los componentes de JBuilder muestran todos los caracteres Unicode
- Funciones de internacionalización del diseñador de interfaz de usuario
- Unicode en el Depurador IDE
- Elección de una codificación nativa para el compilador

Aplicación de ejemplo multilingüe

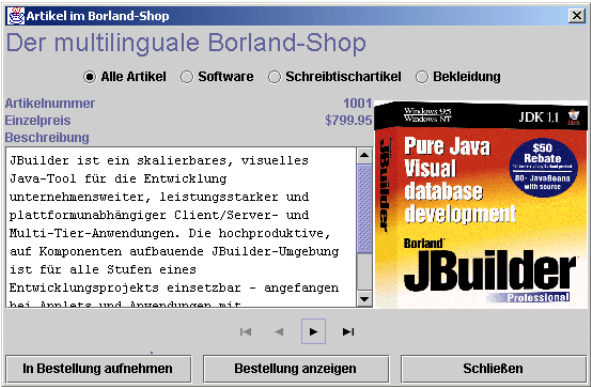
JBuilder incluye una amplia aplicación multilingüe de anotación de pedidos, que demuestra detalladamente muchos de los conceptos importantes de la internacionalización. Este ejemplo muestra también muchas otras funciones importantes de JBuilder, como es la construcción de aplicaciones con los componentes de JBuilder, la creación de JavaBeans internacionalizados y el uso de la arquitectura DataExpress.

El proyecto `IntlDemo.jpr` se encuentra en el directorio `samples/dbswing/MultiLingual` de la instalación de JBuilder. Encontrará más información en el archivo de documentación `IntlDemo.html` y en el código fuente del ejemplo suministrado. El ejemplo `IntlDemo` acepta e incluye traducciones para 15 versiones localizadas distintas.

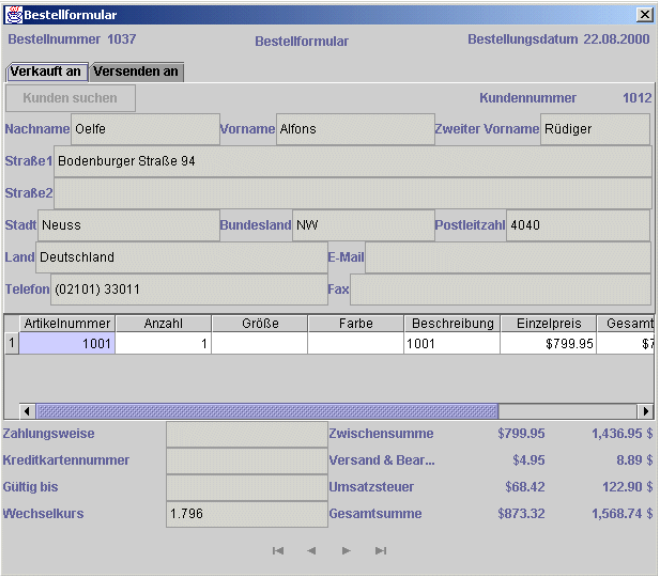
El JavaBean `LocaleChooser` del Multilingual International Store de Borland, permite cambiar el idioma de la aplicación en tiempo de ejecución. Con ello se adapta automáticamente la interfaz gráfica de usuario al idioma y las convenciones de la versión localizada elegida.



ProductFrame permite a los usuarios ver imágenes de productos de Borland Store y descripciones escritas en su propio idioma. Observe que el tamaño de los botones y etiquetas se ajusta automáticamente para las traducciones al italiano y al alemán que se muestran a continuación.



OrderFrame presenta la dirección del cliente y el importe del pedido en el formato adecuado para la versión localizada del usuario. A continuación se muestra OrderFrame en alemán:



Eliminación de cadenas no modificables (hard-coded) incluidas en el código

Un error de diseño frecuente que impide que las aplicaciones y applets se localicen con facilidad es la presencia en el código fuente de cadenas no modificables (hard-coded) que se muestran en la interfaz gráfica de usuario de la aplicación o el applet.

Si bien es posible incluir estas cadenas en los recursos después de haber finalizado y probado el código fuente, es mejor realizar esta tarea como parte del proceso de diseño de la interfaz gráfica de usuario.

La inclusión en recursos de la interfaz gráfica de usuario a medida que se escribe el código tiene dos ventajas:

- No es necesario volver y examinar todas las cadenas no modificables (hard-coded) del código fuente para comprobar cuáles se deben incluir en los recursos. Además de ser un proceso muy trabajoso, en ocasiones al desarrollador (o a otra persona que esté menos familiarizada con el código) le resulta difícil determinar qué cadenas se deben incluir en los recursos.
- Incluir las cadenas en recursos durante una fase temprana puede ayudar a descubrir los diseños de interfaz gráfica de usuario no internacionalizados al principio del proceso de desarrollo, con lo que se evita el esfuerzo de tener que volver a escribirlo más adelante.

JBuilder proporciona dos herramientas para beneficiarse de este método con muy poco esfuerzo: el Asistente para extracción de recursos y el cuadro de diálogo Propiedad localizable.

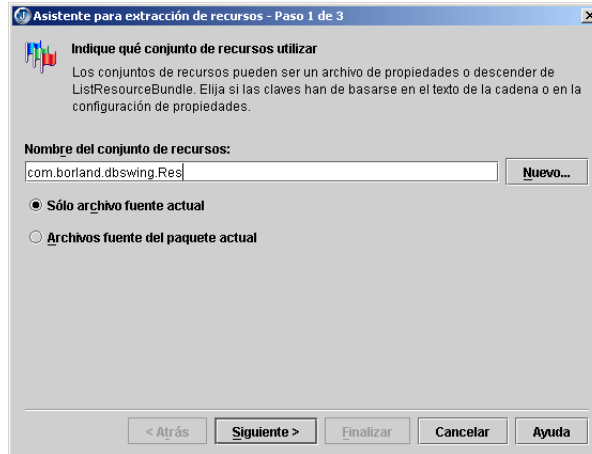
Utilización del Asistente para extracción de recursos

El Asistente para extracción de recursos examina el código fuente y permite extraer las cadenas no modificables (hard-coded) y los caracteres aislados, tales como las teclas aceleradoras, incluidos en el código de forma rápida y sencilla y llevarlos a clases `ResourceBundle` de Java. Este asistente funciona con todos los archivos Java, no sólo con el código fuente generado con JBuilder.

Los `ResourceBundles` (conjuntos de recursos) son archivos especializados que contienen una serie de cadenas traducibles. (También pueden contener otros tipos de datos, aunque es menos frecuente). Una clave de recurso unívoca identifica cada cadena traducible del `ResourceBundle`. La cadena incrustada en el código de la aplicación es sustituida por una referencia al conjunto de recursos y a la clave del recurso. Esta separación entre la lógica de la aplicación y los elementos traducibles se conoce como *extracción de recursos*. A continuación, estos archivos de recursos separados se envían a los traductores.

Para colocar las cadenas en una clase `ResourceBundle`:

- 1 En el panel del proyecto, haga doble clic en el archivo de código fuente que desea examinar; de esta forma se abre en el editor.
- 2 Seleccione Asistentes|Recursos de cadenas para abrir el Asistente para extracción de recursos:

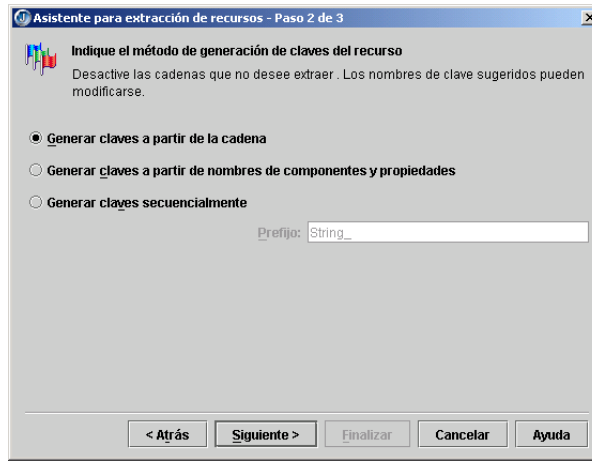


- 3 Indique el nombre del conjunto de recursos que utiliza. JBuilder sugiere un nombre por defecto. Puede aceptarlo o cambiarlo. Por defecto, el objeto `ResourceBundle` que se crea es del tipo `ListResourceBundle`. Si desea crear un conjunto de recursos `PropertyResourceBundle` pulse Nuevo, elija `PropertyResourceBundle` en la lista desplegable Tipo y pulse Aceptar en el cuadro de diálogo que aparezca.

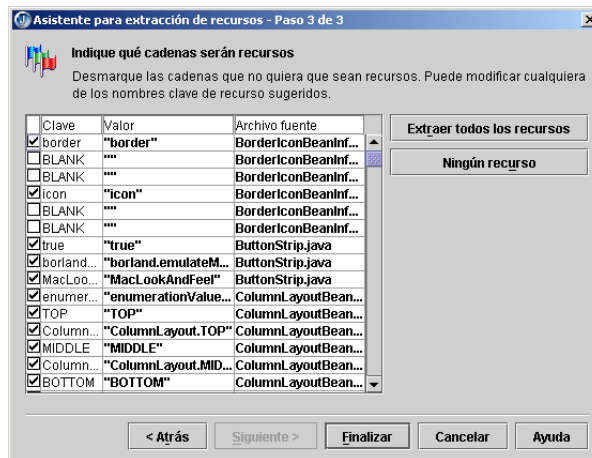
Los `PropertyResourceBundles` son archivos de texto con extensión `.properties` y tienen la misma ubicación que los archivos de clase del código fuente. Los `ListResourceBundles` se proporcionan como archivos fuente de Java. Tanto los `ListResourceBundles` nuevos como los modificados necesitan volver a compilarse para su distribución ya que están implementados como código fuente Java. No es necesario recompilar los `PropertyResourceBundles` cuando se modifican o añaden traducciones a la aplicación. Los `ListResourceBundles` proporcionan un rendimiento mucho mayor que los `PropertyResourceBundles`.

- 4 Si desea reestructurar únicamente el código del archivo que se encuentra en el editor, elija Sólo archivo fuente actual. Si desea extraer el código de

todos los archivos del paquete en el que se encuentra el archivo abierto, elija la opción Archivos fuente del paquete actual y pulse Siguiente.



- 5 Indique de qué forma desea que se generen las claves. Las cadenas se identifican por medio de claves. Por ejemplo, si se desea extraer el código de la cadena "Open File" y se activa la opción Generar clave a partir de la cadena, la clave pasa a ser `Open_File`. La misma cadena se puede convertir en `jbutton1.ToolTipText` si se elige Generar claves a partir de los nombres de componentes y propiedades. Si se elige la opción Generar claves secuencialmente, la clave podría ser `String_10` si es la décima cadena del archivo. Si se elige la tercera opción también se puede añadir un prefijo al nombre de la clave. El prefijo por defecto es `String_`.
- 6 Pulse Siguiente para abrir la última ficha del Asistente para extracción de recursos:



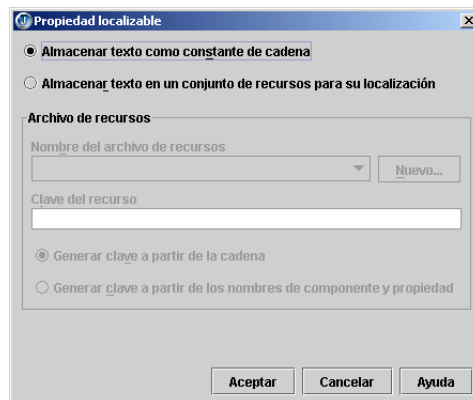
Para ordenar la columna Clave o Valor, haga clic en la cabecera correspondiente. Si se vuelve a pulsar la cabecera, la columna se ordena en sentido inverso.

Si se hace clic en una cadena que aparece en el Asistente para extracción de recursos, la línea en la que aparece se resalta en el código fuente. Esta función permite examinar el contexto de la cadena en el código.

- 7 Desactive las cadenas de las que no desee extraer recursos.
- 8 Pulse el botón Finalizar.

Utilización del cuadro de diálogo Propiedad localizable

El cuadro de diálogo Propiedad localizable permite definir cadenas visibles mientras crea o personaliza componentes en la interfaz de usuario. Basta con hacer clic con el botón derecho del ratón en cualquier propiedad de texto, como la etiqueta de un control `ButtonControl` y seleccionar el comando Conjunto de recursos para presentar el cuadro de diálogo Propiedad localizable.



Este cuadro de diálogo presenta opciones parecidas a las del Asistente para extracción de recursos, pero sólo incluye las que afectan a la propiedad seleccionada. Active la opción Almacenar texto en conjunto de recursos para localización y elija la forma de generar las claves, si no lo ha hecho aún o desea efectuar algún cambio. Como resulta tan rápido y cómodo efectuar la extracción de estos recursos desde el Inspector, se puede convertir con facilidad en parte integrante de la personalización de los componentes de la aplicación.

Funciones de internacionalización de dbSwing

dbSwing es una característica de JBuilder Enterprise.

La arquitectura dbSwing incluye varias decisiones de diseño que facilitan la internacionalización de aplicaciones y applets:

- Las funciones desaconsejadas (deprecated), no internacionalizadas, que se utilizan en JDK 1.0 se evitan en dbSwing.
- Todos los mensajes en dbSwing se guardan en clases `ResourceBundle`, con lo que se permite a las aplicaciones que se construyen con estos componentes mostrar el texto en el idioma correspondiente a la versión localizada del usuario final.
- Todos los componentes de dbSwing gestionan aspectos internacionales, como el formato de datos y el sistema de ordenación acordes con la versión localizada.

La mayoría de los componentes dbSwing de JBuilder incluye la propiedad `textWithMnemonic`. Esta propiedad acepta un carácter acelerador que se especifica en la cadena utilizada para mostrar el texto del componente. El diseño Swing (de donde derivan muchos componentes dbSwing) consiste en guardar el texto y los caracteres mnemotécnicos en propiedades independientes. Esto dificulta la localización ya que los traductores suelen disponer de poco contexto en el que basar la traducción de las cadenas. Los componentes dbSwing guardan el carácter mnemotécnico incrustado en el texto, el traductor puede escoger el mnemotécnico correcto. Si esta función se utiliza adecuadamente, puede proporcionar contexto durante la traducción.

El componente `IntlSwingSupport` de JBuilder admite la internacionalización de Swing en doce versiones distintas. Cuando se instancia `IntlSwingSupport` actualiza automáticamente los recursos internos localizables de Swing en función del idioma actual. `IntlSwingSupport` sólo debe ser instanciado una vez en una aplicación y al inicio de la aplicación, antes de que se presenten los componentes Swing.

Importante

`IntlSwingSupport` no es necesario si la aplicación se ejecuta en JDK 1.3 o posterior. Estos JDK posteriores ofrecen recursos de localización parecidos a los del componente `IntlSwingSupport`.

Para inicializar `IntlSwingSupport` en un idioma distinto del establecido por defecto (por ejemplo, en una aplicación multilingüe), defina la propiedad `locale` del componente `IntlSwingSupport` para el país destino. Por ejemplo:

```
new IntlSwingSupport();
int response = JOptionPane.showConfirmDialog(frame, localizedMessageString,
    localizedTitleString, JOptionPane.OK_CANCEL_OPTION);
```

Nota

`IntlSwingSupport` sirve para suministrar apoyo internacional a algunos componentes Swing, pero no para los de dbSwing. Todos los componentes dbSwing ya están completamente internacionalizados.

En JDK 1.2, los únicos componentes Swing con cadenas de texto visibles y traducibles son `JFileChooser`, `JColorChooser` y `JOptionPane`. Si desea obtener más información sobre las versiones internacionales, consulte la documentación de Sun acerca de la clase `Local`.

Componentes que identifican la versión localizada

Además de estar completamente incluidos en los recursos, muchos componentes de JBuilder cuentan también con un comportamiento práctico acorde con la versión localizada. Por ejemplo, los datos de cadena que se cargan en la columna `Column` de una `JdbTable` por medio de componentes `DataExpress` para conjuntos de datos `DataSet` se ordenan automáticamente según el sistema de ordenación por defecto de la versión localizada que ejecuta el usuario. Del mismo modo, la fecha, la hora y los valores numéricos adquieren automáticamente el formato acorde con la versión localizada del usuario.

Por defecto, los objetos heredan la versión localizada de sus contenedores. Por tanto, la configuración localizada de un `Conjunto de datos` es la que utilizan por defecto las `columnas de ese conjunto de datos`. También existe la posibilidad de elegir una configuración localizada distinta para cada objeto `Column` del `DataSet`. Esto puede resultar útil si, por ejemplo, cada `columna` contiene datos que se deben ordenar con arreglo a las normas de una versión localizada distinta. Si desea más información sobre las formas de ordenación acordes con la versión localizada, consulte la documentación de la API de JDK sobre la clase `Collator`.

Si desea más información sobre el formato de tipos de datos acorde con la versión localizada en Java, consulte la documentación de la API de JDK sobre las clases `DateFormat`, `NumberFormat` y `MessageFormat`.

Los componentes de JBuilder muestran todos los caracteres Unicode

Las arquitecturas de componentes que dependen únicamente de los controles visuales homólogos nativos para la presentación de caracteres, sólo pueden mostrar el conjunto de caracteres que acepta el homólogo nativo. Dado que los componentes de JBuilder utilizan Java para mostrar caracteres en vez de homólogos nativos, pueden presentar cualquier carácter de Unicode del que se tenga una fuente instalada, independientemente de que ese carácter exista o no en la página de códigos por defecto del sistema operativo.

Para mostrar los caracteres Unicode de una nueva fuente:

- 1 Instale la fuente deseada en el sistema operativo.
- 2 Modifique el archivo `font.properties` de JDK de su versión localizada e indique que la fuente correspondiente a ese carácter está ahora disponible.

Si desea ver las instrucciones sobre la forma llevar a cabo esta operación, consulte el tema “Archivo `fonts.properties`” de la documentación sobre la internacionalización de JDK.

Funciones de internacionalización del diseñador de interfaz de usuario

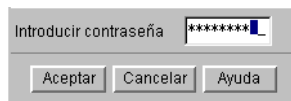
El diseñador de interfaces de usuario de JBuilder es una valiosa herramienta para la creación y verificación de interfaces gráficas de usuario internacionalizadas. Cuando se añaden a la interfaz gráfica de usuario elementos de texto que se pueden traducir, es posible colocarlos inmediatamente en conjuntos de recursos. El Inspector lee automáticamente las cadenas de los conjuntos de recursos y vuelve a escribirlas. Además, una vez incluido en los recursos todo el texto de la interfaz gráfica de usuario y recibido el conjunto de recursos localizado del traductor, se puede utilizar el diseñador para construir y comprobar rápidamente la interfaz de usuario internacionalizada.

El Inspector presenta descripciones breves, acordes con la versión localizada, sobre las propiedades de cada JavaBean, tal y como se describe en la sección de internacionalización de la especificación JavaBeans.

El Inspector permite utilizar secuencias de escape de caracteres Unicode para indicar caracteres que no se pueden introducir directamente desde el teclado con la configuración de país del sistema operativo. Cuando se desea insertar un carácter de Unicode dentro de una propiedad de cadena que se está modificando, basta con introducir el valor hexadecimal de la secuencia de escape de Unicode correspondiente a ese carácter, entre corchetes angulares. Por ejemplo, para insertar el carácter japonés de la palabra “montaña” en la etiqueta de un botón, se debe escribir “<5C71>”. Si tiene fuentes japonesas instaladas en el sistema y el archivo `font.properties` de JDK está configurado correctamente, el carácter se muestra como etiqueta del botón, y la secuencia de escape de Unicode “\u5C71” aparece en el código fuente.

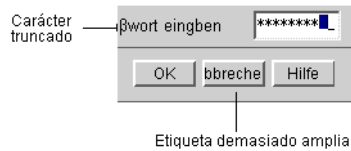
El diseñador de interfaces de usuario proporciona un excelente apoyo a los gestores dinámicos de diseño, un importante requisito para el diseño de interfaces gráficas del usuario internacionalizadas. Es difícil construir una sola interfaz gráfica de usuario que pueda aceptar muchos idiomas, pero la compatibilidad del diseñador visual con los gestores dinámicos de diseño AWT de Java facilita enormemente esta tarea. Cuando se diseña una interfaz gráfica de usuario con intención de localizarla a más de un idioma, es muy importante *utilizar siempre un gestor dinámico de diseño*.

Observe, por ejemplo, el siguiente cuadro de diálogo, que contiene los botones Aceptar, Cancelar y Ayuda:



La presentación es la adecuada para las etiquetas en inglés, pero cuando se traducen al alemán, el texto de las etiquetas es demasiado largo para

ajustarse al tamaño fijo del botón. Éste es un problema que se da con frecuencia cuando se intenta localizar una IU no internacionalizada.



La solución consiste en utilizar uno o más gestores dinámicos de diseño AWT para permitir que el tamaño de los botones se ajuste al de su etiqueta. Éstas son las versiones inglesa y alemana, internacionalizadas, del mismo cuadro de diálogo, escrito con un panel con `GridLayout` dinámico para los botones e incrustados dentro de un cuadro de diálogo `BorderLayout`.



Para obtener más información sobre la creación de diseños dinámicos mediante el diseñador de interfaces de usuario, consulte la sección “Gestores de diseño” en *Diseño de aplicaciones con JBuilder*.

La aplicación de ejemplo internacional multilingüe también muestra algunas técnicas avanzadas para actualizar el diseño de `Marcos` de una aplicación durante la ejecución.

Unicode en el Depurador IDE

El depurador de JBuilder permite ver y editar caracteres de Unicode, incluso en el caso de que el sistema operativo no los acepte. Cuando se examinan los valores del panel de punto de observación del depurador, es necesario ampliar el valor del árbol que se desea inspeccionar hasta ver su tipo Java primitivo. Por defecto, el depurador intenta presentar el carácter de Unicode, dando por supuesto que el sistema operativo lo acepta.

Si desea ver el equivalente Unicode de un carácter, haga clic con el botón derecho en su valor y seleccione la opción `Mostrar valor hexadecimal` para ver la secuencia de escape de Unicode correspondiente al carácter. También se puede cambiar el valor seleccionando `Modificar valor` e introduciendo otra secuencia de escape de Unicode en el cuadro de diálogo `Cambiar valor de los datos`.

Elección de una codificación nativa para el compilador

Los compiladores JBuilder y **javac** pueden compilar código fuente codificado con codificaciones nativas (también denominadas `páginas de códigos locales`), que es el formato de almacenamiento que utilizan la mayoría de los editores, como el editor de JBuilder.

El IDE y el compilador aceptan todas las codificaciones nativas de JDK. Todos los compiladores de JBuilder seleccionan automáticamente la codificación nativa correspondiente al país configurado en el sistema operativo. También se puede elegir otra codificación de JDK para compilar archivos de código fuente escritos con una codificación nativa distinta.

Puede especificar un nombre nativo de codificación, para controlar cómo interpreta el compilador los caracteres que no pertenecen al conjunto de caracteres ASCII. Esto se puede hacer en todo el proyecto o con la opción de codificación del compilador, desde la línea de comandos. Si no define un valor en esta opción, se utiliza el convertidor por defecto de codificación nativa de la plataforma.

Definición de la opción de codificación

Para definir la opción de codificación desde el IDE:

- 1 Elija Proyecto|Propiedades de proyecto para abrir el cuadro de diálogo homónimo.
- 2 Seleccione General para que se abra la ficha General.
- 3 Seleccione un nombre de codificación en la lista desplegable Codificación.
- 4 Pulse Aceptar.

Para definir la opción de codificación en la línea de comandos:

- 1 Utilice la opción **-encodingdebcj** seguida del nombre de codificación.
- 2 Utilice la opción **-encodingdebmj** seguida del nombre de codificación.

El nombre de codificación `default` equivale a no especificar una opción de codificación. En su lugar, se utiliza la codificación por defecto del entorno del usuario.

Para obtener una descripción de cada codificación, consulte la JDK Internationalization Specification (Especificación de internacionalización del JDK): Character Set Conversion (Conversión de conjuntos de caracteres): Supported Encodings (Codificaciones admitidas) en

<http://java.sun.com/j2se/1.4/docs/guide/intl/encoding.doc.html>. Las descripciones siguientes complementan este apartado:

Unicode	Unicode, con bytes más o menos significativos al principio, según indique la marca de orden de bytes.
----------------	---

UnicodeBig	Unicode con bytes más significativos al principio.
UnicodeLittle	Unicode con bytes menos significativos al principio.

Adición y redefinición de codificaciones

Para añadir codificaciones a la lista desplegable Codificación de la ficha Generar del cuadro de diálogo Propiedades de proyecto (Proyecto | Propiedades de proyecto):

- 1 Abra el archivo `user.properties` de la carpeta `<.jbuilder>`.
- 2 Añada codificaciones, como en el ejemplo siguiente:

```
compiler.java;encodings.add.1=ISO8859_2
compiler.java;encodings.add.2=ISO8859_3
```

- 3 Guarde y cierre el archivo.
- 4 Reinicie JBuilder.

Para sustituir las codificaciones de la lista desplegable Codificación:

- 1 Abra el archivo `user.properties` de la carpeta `<.jbuilder>`.
- 2 Redefina las codificaciones, como en el ejemplo siguiente:

```
compiler.java;encodings.override.1=ISO8859_2
compiler.java;encodings.override.2=ISO8859_3
```

- 3 Guarde y cierre el archivo.
- 4 Reinicie JBuilder.

Otros asuntos relacionados con las codificaciones nativas

Los entornos no basados en Unicode representan los caracteres utilizando distintos sistemas de codificación. En el mundo de los PC se denominan páginas de códigos; Java las conoce como codificaciones nativas. Cuando se trasladan datos de un sistema de codificación a otro, es necesario realizar una conversión. Dado que cada sistema puede contar con un conjunto diferente de caracteres extendidos, se necesita una conversión para impedir que se pierda información.

Si lo prefiere, puede emplear la utilidad **native2ascii** de Java para convertir los caracteres de codificación nativa a secuencias de escape de Unicode (por ejemplo, `\uNNNN`). Los archivos convertidos se pueden compilar en cualquier sistema, sin necesidad de efectuar más conversiones ni especificar una codificación.

La mayoría de los editores de textos, como el editor de JBuilder, almacenan el texto con la codificación nativa. Por ejemplo, en la versión japonesa de Windows se utiliza el formato Shift-JIS, mientras que en la versión estadounidense se utiliza la página de códigos 1252 de Windows. A partir de

JDK 1.1, **javac** también puede compilar código fuente “con codificación nativa”. La codificación puede especificarse utilizando el modificador “encoding”. Si no se utiliza el modificador “encoding”, el compilador utiliza la codificación que corresponde al entorno del usuario.

Al contrario que en el caso de Unicode, el código fuente escrito con una codificación nativa no puede llevarse directamente a sistemas que utilizan otras codificaciones. Por ejemplo, si el código fuente se ha codificado con Shift-JIS (una codificación japonesa) y utiliza el compilador en un entorno de Windows en español, debe especificar la codificación Shift-JIS para que el compilador pueda leer correctamente el código fuente.

Formato Unicode de 16 bits

Unicode es un sistema universal de representación de caracteres con números de 16 bits. El conjunto de caracteres Unicode de 16 bits puede implementarse directamente o puede representarse de forma indirecta dentro del conjunto de caracteres ASCII de 7 bits, utilizando el carácter de escape `\u` seguido de cuatro dígitos hexadecimales.

Cuando todos los sistemas operativos principales admitan directamente Unicode, se reemplazará el sistema establecido actualmente, que requiere una conversión entre las distintas codificaciones nativas que tengan valores de caracteres conflictivos. Java es uno de los primeros entornos que utilizan la norma Unicode, que es el conjunto de caracteres interno del entorno Java.

Utilización de Unicode mediante ASCII y ‘`\u`’

Actualmente, la mayoría de los editores de texto de Windows, incluido el editor de JBuilder, almacenan y procesan el texto como caracteres de 7 u 8 bits, en lugar de caracteres de 16 bits de Unicode. El conjunto de caracteres ASCII utiliza una codificación de 7 bits que contiene las 26 letras del alfabeto inglés y algunos símbolos. Casi todas las codificaciones nativas tienen ASCII como un subconjunto y lo representan de la misma forma: los primeros 127 caracteres de la codificación componen el conjunto de caracteres ASCII. El conjunto de caracteres ASCII puede considerarse un subconjunto de Unicode.

Para permitir a los usuarios especificar caracteres de Unicode en su código fuente sin utilizar un editor preparado para Unicode, la especificación de Java permite utilizar el carácter escape `\u` de Unicode en los archivos ASCII. Esta utilización permite representar caracteres extendidos con una combinación de caracteres ASCII. Esta forma de representar Unicode utiliza 6 caracteres para representar un solo carácter no perteneciente a ASCII. Para introducir un carácter ASCII normal, basta con pulsar en el teclado la tecla que corresponde al carácter; para introducir uno de otro conjunto de caracteres, se escribe la secuencia de escape de Unicode que representa al carácter.

En esta representación de 7 bits de Unicode, los caracteres no pertenecientes al conjunto de caracteres ASCII se representan con la forma `\uNNNN`, donde NNNN son los 4 dígitos hexadecimales del carácter Unicode. Por ejemplo, el carácter Unicode “f latina minúscula con gancho”, una letra 'f' cursiva, que se representa en Unicode con el número hexadecimal 0192, puede introducirse escribiendo “\u0192”.

Unicode, tanto en el formato de 16 bits como el de 7 bits, está en formato universal; el código fuente en Unicode se puede trasladar directamente a todas las plataformas, en todos los idiomas.

JBuilder en el mundo

JBuilder está disponible en varios idiomas, entre los que se encuentran el inglés, el alemán, el francés, el español y el japonés. Las versiones localizadas generalmente incluyen traducciones de la documentación, la IU y los componentes. Se pueden encargar versiones localizadas de JBuilder en las delegaciones de ventas de Borland de los siguientes países. Para encontrar enlaces a las sedes web internacionales de Borland, diríjase a Borland Worldwide en <http://www.borland.com/bww/>.

Asistencia internacional en línea

Visite el grupo de noticias sobre aplicaciones multilingües en la sede Web de Borland en

<news://newsgroups.borland.com/borland.public.jbuilder.multi-lingual-apps>. Este grupo de noticias se dedica a asuntos de internacionalización y localización de JBuilder, y está moderado activamente por nuestros ingenieros de servicio técnico, así como por los ingenieros de Investigación y desarrollo y Control de calidad del grupo de internacionalización de JBuilder.



Tutorial: Creación de un proyecto con un archivo de generación Ant

Este tutorial explica cómo trabajar con archivos de generación Ant para generar proyectos. Ant es una herramienta de generación basada en Java que construye proyectos según se especifica en los archivos de generación XML. Los archivos de generación definen los tipos y tareas de generación. Por ejemplo, un archivo de generación podría contener tipos de generación separados para generar un proyecto y crear Javadoc. Puede ejecutar tipos de generación individuales o el tipo por defecto del proyecto utilizando el archivo de generación Ant.

JBUILDER reconoce automáticamente los archivos de generación Ant llamados `build.xml` y muestra estos nodos con un icono Ant en lugar del icono XML habitual. También puede utilizar el Asistente para Ant para importar archivos Ant, sea cual sea su nombre. Los tipos del archivo `build.xml` se muestran como nodos descendientes.

En este tutorial, se completan las siguientes tareas:

- Creación de un proyecto y una aplicación
- Creación de un archivo de generación Ant
- Ejecución de tipos de generación Ant
- Ejecución del tipo Ant por defecto

Paso 1: Crear un proyecto y una aplicación

- Cómo introducir un error en un archivo fuente Java y examinar los mensajes de error Ant
- Cómo añadir un tipo al menú Proyecto
- Modificación de propiedades Ant
- Cómo añadir bibliotecas Ant personalizadas

Consulte

- [“Generación con archivos Ant” en la página 6-12](#)
- El proyecto Jakarta en Apache: <http://jakarta.apache.org/ant>
- La documentación de Ant en el archivo ZIP Ant del directorio [extras](#) de JBuilder

El apartado Accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder por parte de personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte [“Convenciones de la documentación” en la página 1-5](#).

Paso 1: Crear un proyecto y una aplicación

En este paso, utilizará los asistentes de JBuilder para crear un proyecto y una aplicación.

- 1 Elija **Archivo|Nuevo proyecto** para iniciar el Asistente para proyectos.
- 2 Introduzca `AntProject` en el campo Nombre, y pulse **Finalizar** para cerrar el asistente y crear el proyecto.
- 3 Seleccione **Archivo|Nuevo|General** para abrir la galería de objetos y haga doble clic en el icono **Aplicación** de la ficha **General** para abrir el Asistente para aplicaciones.
- 4 Acepte los valores por defecto y haga clic en **Finalizar** para cerrar el asistente.

Paso 2: Crear el archivo de generación Ant

Ahora que ya tiene un proyecto, se crea un archivo de generación Ant que se utiliza para generar el proyecto. JBuilder identifica automáticamente los archivos llamados `build.xml` como archivos de generación Ant y muestra iconos Ant para estos nodos en el panel del proyecto.

Primero, cree el archivo de generación Ant.

- 1 Seleccione **Archivo|Nuevo archivo** o haga clic con el botón derecho del ratón en el nodo del proyecto `.jpx` en el panel del proyecto y seleccione **Nuevo|Archivo**.

- 2 Realice los siguientes cambios en el cuadro de diálogo Crear archivo:
 - a Escriba `build` en el campo Nombre.
 - b Escoja XML como la extensión de archivo en la lista desplegable Tipo.
 - c Cambie la vía de acceso en el campo Directorio en la raíz del proyecto. El archivo `build.xml` se guarda en la raíz del proyecto. Por ejemplo, en Windows sería algo parecido a lo siguiente: `C:\Documents and Settings\username\jbproject\AntProject\.`
 - d Asegúrese de que la opción Añadir archivo al proyecto se encuentra activada.
- 3 Haga clic en Aceptar para cerrar el cuadro de diálogo. El nuevo archivo se abrirá en el editor y se añadirá al proyecto.
- 4 Escriba el siguiente texto o cópielo y péguelo en el editor:

```
<?xml version="1.0"?>
<!DOCTYPE project>
<project name="AntProject" default="dist" basedir=".">
  <property name="src" value="src"/>
  <property name="build" value="build"/>
  <property name="dist" value="dist"/>

  <target name="init">
    <tstamp/>
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init">
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile">
    <mkdir dir="${dist}/lib"/>
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean">
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>

</project>
```

Nota

Se puede utilizar TagInsight de XML para facilitar la entrada automática de elementos y atributos. Para abrir TagInsight, escriba un corchete de apertura (`<`) y seleccione un elemento en la lista emergente. Para añadir un atributo, escriba un espacio después del nombre del elemento y seleccione un atributo en la lista TagInsight de XML. Si desea obtener más información, consulte "TagInsight" en *Introducción a JBuilder*. Para obtener más información acerca de XML y el editor, consulte "XML en el editor" en la *Guía del desarrollador de XML*.

- 5 Guarde el proyecto.

6 Examine el archivo `build.xml` para comprender su función:

<pre><project name="AntProject" default="dist" basedir="."></pre>	Incluye el nombre de proyecto, el objetivo por defecto a ejecutar si no se ejecuta ninguno de los otros tipos individuales, y la ubicación del directorio base.
<pre><property name="" value=""/></pre>	Los tipos y tareas Ant están habitualmente "enlazados a la propiedad". Las propiedades se utilizan también para pasar parámetros a las tareas sin, por ello, redefinir las propiedades ya existentes en el archivo de generación.
<pre><target name="init"></pre>	Crea un directorio <code>build</code> para las clases compiladas.
<pre><target name="compile" depends="init"></pre>	Inicia primero el objetivo <code>init</code> , compila los archivos fuente Java y coloca los archivos <code>.class</code> generados en el directorio <code>build</code> .
<pre><target name="dist" depends="compile"></pre>	Inicia primero el objetivo <code>compile</code> , a continuación crea un directorio <code>dist/lib/</code> y, por último, crea un archivo JAR en dicho directorio.
<pre><target name="clean"></pre>	Elimina los directorios <code>build</code> y <code>dist</code> .

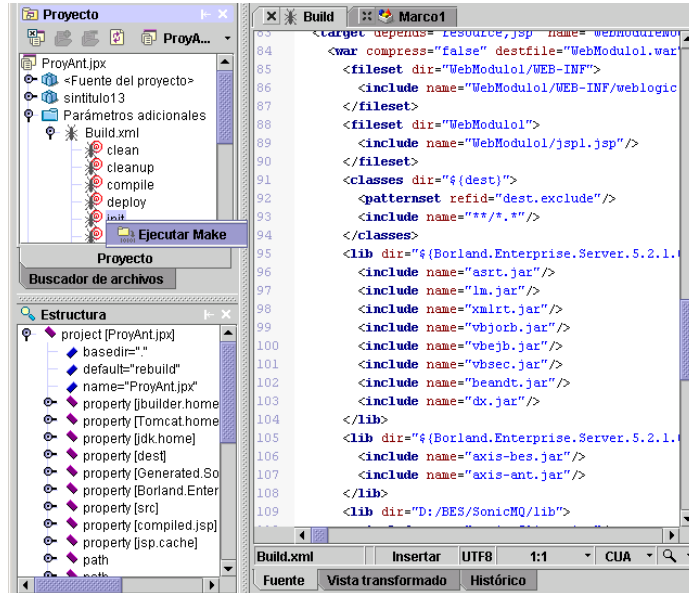
Paso 3: Ejecutar tipos de generación individuales

A continuación, ejecutará dos tipos del archivo `build.xml`. Primero, ejecutará el `init` target para crear el directorio `build` para los archivos `.class` compilados. Finalmente, utilizará el objetivo `clean` para eliminar la salida y el directorio `build`.



- 1 Pulse el botón Actualizar en la barra de herramientas del panel del proyecto y amplíe el nodo Ant en el panel del proyecto para mostrar los nodos dependientes, que son los tipos Ant. Observe que los tipos Ant aparecen en orden alfabético en el panel del proyecto.

- 2 Haga clic con el botón derecho sobre el tipo Ant init del panel del proyecto y seleccione Ejecutar Make. Este tipo crea el directorio build, donde se almacenarán los archivos .class compilados.



- 3 Examine los mensajes de salida de un nodo Ant en el panel de mensajes. El objetivo init creó con éxito el directorio build.

```
StdOut
Archivo de generación: build.xml
init:
[mkdir] Directorio creado: C:\Documents and Settings\ktaylor\
        jproject\AntProject\build
        GENERACIÓN CORRECTA
Tiempo total: 2 segundos
```

- 4 Haga clic con el botón derecho sobre el tipo Ant compile del panel del proyecto y seleccione Ejecutar Make. Este objetivo compila los archivos fuente .java, genera los archivos .class, y los coloca en el directorio build creado por el objetivo init. Examine el resultado obtenido en el panel de mensajes.
- 5 Haga clic con el botón derecho sobre el objetivo Ant clean y seleccione Ejecutar Make para eliminar toda la salida generada, incluido el directorio build.

Paso 4: Ejecutar el objetivo por defecto

Si selecciona el nodo Ant y realiza un Ejecutar Make sobre él, JBuilder ejecuta el objetivo Ant por defecto, en este caso, el objetivo dist. El objetivo por defecto se especifica en el elemento `<project>`. El objetivo dist crea `dist/lib/` y genera un archivo JAR en ese directorio. Observe que el objetivo dist depende de compile, que, a su vez, depende de init. Así que cuando el objetivo dist se ejecuta, ejecuta el objetivo compile, que, a su vez, ejecuta el objetivo init. Debido a estas dependencias, el orden de ejecución de los tipos es: init, compile, dist.

A continuación, se ejecutará el tipo por defecto del archivo de generación.

- 1 Haga clic con el botón derecho sobre el nodo Ant `build.xml` en el panel del proyecto y seleccione Ejecutar Make para ejecutar el tipo de generación por defecto, dist.
- 2 Mire en el panel de mensajes para ver los resultados de la generación:

```
StdOut
Archivo de generación: build.xml

init:
[mkdir] Directorio creado: C:\Documents and Settings\ktaylor\
      jbproject\AntProject\build

compile:
[javac] Compilando 2 archivos fuente en C:\Documents and Settings\ktaylor\
      jbproject\AntProject\build
[javac] analizando C:\Documents and Settings\ktaylor\
      jbproject\AntProject\src\antproject\Application1.java
[javac] analizando C:\Documents and Settings\ktaylor\
      jbproject\AntProject\src\antproject\Framel.java

dist:
[mkdir] Directorio creado: C:\Documents and Settings\ktaylor\
      jbproject\AntProject\dist\lib
[jar] Generando jar: C:\Documents and Settings\ktaylor\
      jbproject\AntProject\dist\lib\MyProject-20030902.jar

GENERACIÓN CORRECTA
Tiempo total: 2 segundos
```

Debido a que borré anteriormente las clases y su directorio con el tipo clean, el tipo init vuelve a crear el directorio build. El tipo compile compila una vez más los archivos `.class`. Finalmente, el tipo dist crea el directorio `dist/lib/` y genera un archivo JAR en él.

Paso 5: Tratamiento de errores con Ant

En este paso, introducirá un error en un archivo fuente Java, ejecutará Make del archivo de generación Ant, y utilizará los mensajes de error para buscarlo en el archivo fuente.

- 1 Abra `Application1.java` en el editor.
- 2 Busque el método `main()` y añada etiquetas de comentario como se muestra a continuación para presentar un error:

```
// public static void main(String[] args) {
```

- 3 Haga clic con el botón derecho en `build.xml` y elija Ejecutar Make.
- 4 Examine el panel de mensajes y observe que aparece un nodo `StdErr` que muestra mensajes de error. En este ejemplo, añadir etiquetas de comentario delante del método `main()` produce varios errores:

```
StdErr
"Application1.java":      [javac] C:\Documents and Settings\ktaylor\jbproject
                          AntProject\src\antproject\Application1.java:43: illegal start of
                          type at line 43
                          [javac]      try {
                          [javac] ^
"Application1.java":      [javac] C:\Documents and Settings\ktaylor\jbproject
                          AntProject\src\antproject\Application1.java:49: <identifier>
                          expected at line 49
                          [javac] new Application1();
                          [javac] ^
"Application1.java":      [javac] C:\Documents and Settings\ktaylor\jbproject
                          AntProject\src\antproject\Application1.java:51: 'class' or 'interface'
                          expected at line 51
                          [javac] }
                          [javac] ^
"Application1.java":      [javac] C:\Documents and Settings\ktaylor\jbproject
                          AntProject\src\antproject\Application1.java:51: 'class' or 'interface'
                          expected at line 51
                          [javac] }
                          [javac] ^
```

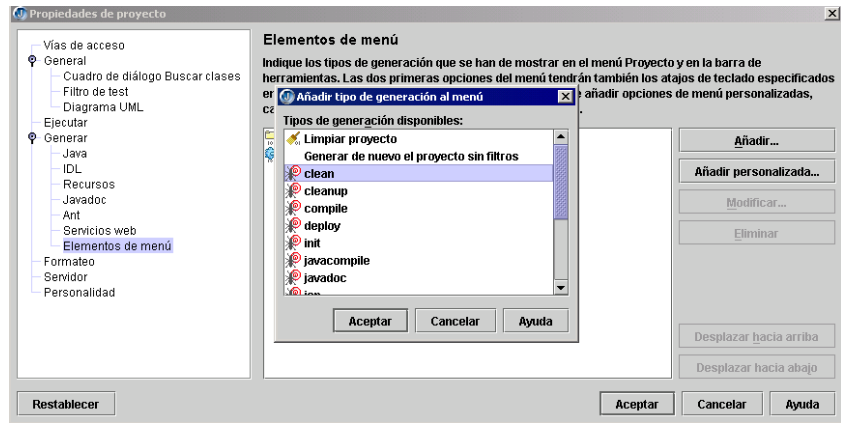
- 5 Seleccione un mensaje de error del panel para resaltarlo en el editor. Haga doble clic en el error para desplazar el cursor a la línea de código, en el editor. Recuerde que el número de línea del error no tiene por qué ser el origen del error.
- 6 Elimine las etiquetas de comentario del método `main()` antes de pasar al siguiente paso.

Paso 6: Añadir un tipo de generación al menú Proyecto

En este paso, se añade el tipo Ant clean al menú Proyecto y se reorganizan los tipos de generación. Si desea más información sobre cómo configurar el menú, consulte [“Configuración del menú Proyecto” en la página 6-34](#).

- 1 Seleccione `Proyecto|Propiedades de proyecto|Generar|Elementos de menú` para abrir el cuadro de diálogo `Propiedades de proyecto`.

- 2 Pulse el botón Añadir para abrir el cuadro de diálogo Añadir tipo de generación al menú.



- 3 Seleccione clean como tipo Ant en el archivo `build.xml`, **no** el tipo JBuilder, Limpiar proyecto.
- 4 Pulse Aceptar para añadir al menú Proyecto el tipo de generación clean.
- 5 Pulse el botón Desplazar hacia arriba para subir el tipo Ant clean en la lista, debajo de Ejecutar Make del proyecto. Ahora, el tipo clean será el segundo elemento en el menú Proyecto.

Sugerencia

Los dos primeros elementos del menú Proyecto tienen asignadas teclas que pueden personalizarse en el Editor de configuración de teclado (Herramientas|Preferencias|Configuración de teclado).

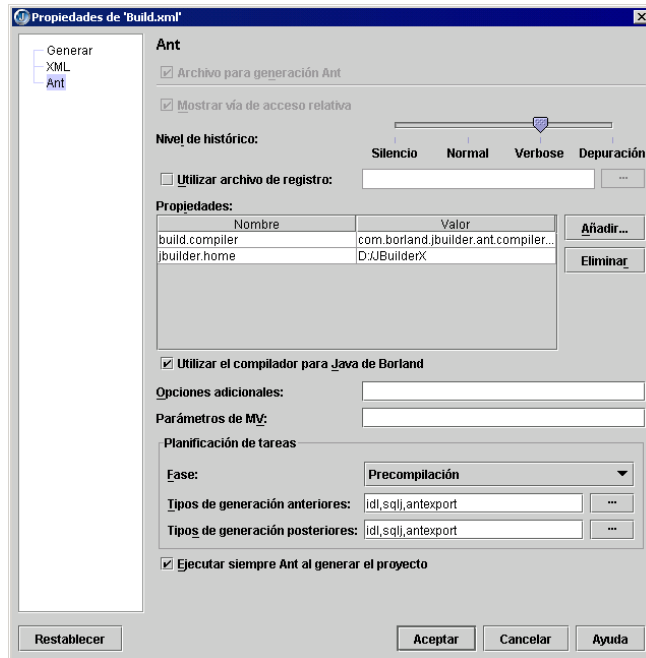
- 6 Haga clic en Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.
- 7 Seleccione Proyecto|Limpiar para limpiar el proyecto. Observe en el panel de mensajes que el tipo clean se ha ejecutado.

Paso 7: Configuración de las propiedades de Ant

Puede que haya casos en los que quiera cambiar las propiedades Ant del fichero de generación sin sobreescribirlo. Puede realizar esto pasando parámetros en el cuadro de diálogo Propiedades Ant.

- 1 Haga clic con el botón derecho del ratón en el nodo `build.xml` de Ant en el panel del proyecto y seleccione Propiedades.

- 2 Seleccione Ant en el árbol y cambie la opción Nivel de histórico a Verbose, lo que proporciona más información al panel de mensajes.



- 3 Pulse el botón Añadir, a la derecha de la lista Propiedades.
- 4 Seleccione `build` en la lista desplegable Nombre y escriba `test` el campo Valor.



- 5 Pulse dos veces sobre Aceptar para cerrar ambos cuadros de diálogo.
Ahora, cuando ejecute el objetivo Ant compile, se creará un directorio `test` y los archivos de clase se crearán en este directorio en vez de en `build`.
- 6 Haga clic con el botón derecho sobre el tipo Ant compile y seleccione Ejecutar Make.
- 7 Examine la salida en el panel de mensajes y observe que hay más información, porque se utiliza la opción verbose. Los resultados le indican que se creó el directorio `test` cuando se ejecutó el tipo `init`, en lugar del directorio `build`.

A continuación, se configura una opción para generar siempre el proyecto con Ant cuando utilice el comando Crear proyecto o Generar el proyecto. En primer lugar, ha de limpiar el proyecto.

- 1 Seleccione Proyecto|Limpiar. Como puede ver en el panel de mensajes, se ha borrado toda la salida de Ant, incluyendo los archivos de clases y el directorio `test`.
- 2 Haga clic con el botón derecho del ratón en `AntProject.jpx` en el panel del proyecto y pulse Limpiar. Esto elimina las clases del directorio `classes` que generó el sistema de generación de JBuilder. Si mira en el administrador de archivos de su sistema operativo, verá que las clases y el directorio `classes` generado por JBuilder han sido eliminados.
- 3 Haga doble clic sobre el nodo `build.xml` y seleccione Propiedades.
- 4 Abra Ant en el árbol y active la opción Ejecutar siempre Ant al generar el proyecto en la ficha Ant y pulse Aceptar para cerrar el cuadro de diálogo. A partir de ahora, cuando elija Proyecto|Ejecutar Make del proyecto, Ant se ejecutará como parte del proceso de generación de JBuilder.
- 5 Seleccione Proyecto|Ejecutar Make del proyecto para generarlo.

Ant ejecuta el tipo Ant por defecto y JBuilder genera con Ejecutar Make. Los mensajes de Ant mostrados en el panel de mensajes le informan de que se han creado nuevos directorios, se han compilado las clases y se ha creado un JAR. Mire en el administrador de archivos de su sistema operativo para comprobar si JBuilder generó también los archivos de clases en el directorio `classes` cuando se ejecutó Make.

Cómo añadir tareas Ant personalizadas a su proyecto

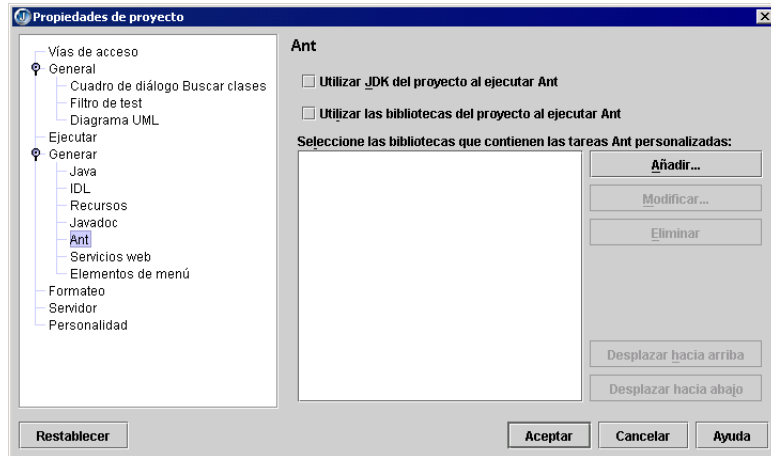
Si el proyecto utiliza bibliotecas, es necesario especificarlas en la ficha Ant del cuadro de diálogo Propiedades de proyecto, para que Ant las pueda encontrar al generar. Puede haber casos en los que tenga bibliotecas personalizadas que contengan tareas de generación Ant hechas a medida. También se pueden añadir estas bibliotecas al proyecto en la ficha Ant. Además, si desea utilizar una versión distinta de Ant, puede añadir una biblioteca con los JAR de Ant. Si no especifica ningún JAR de Ant, JBuilder utiliza el Ant que se suministra en el directorio `lib` de JBuilder.

Consulte

- [“Adición de bibliotecas” en la página 6-20](#)

Puede añadir estas bibliotecas a su proyecto en la ficha Ant de Propiedades de proyecto de la siguiente manera:

- 1 Seleccione Proyecto|Propiedades de proyecto|Generar|Ant.



- 2 Marque la opción Utilizar las bibliotecas del proyecto al ejecutar Ant si desea que JBuilder añada automáticamente las bibliotecas del proyecto. Si se selecciona esta opción, Ant sabe dónde encontrar las bibliotecas del proyecto al generar.
- 3 Pulse el botón Añadir para abrir el cuadro de diálogo Seleccionar bibliotecas.
- 4 Seleccione una biblioteca de la lista o pulse el botón Nueva para crearla con el Asistente para bibliotecas. Pulse Aceptar para cerrar el cuadro de diálogo. Seleccione una biblioteca y añada la biblioteca a la lista de Bibliotecas que contienen tareas Ant personalizadas.
- 5 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.

Ahora que ha completado el tutorial, consulte [“Generación con archivos Ant” en la página 6-12](#) para conocer más detalles sobre la ejecución de archivos de generación Ant en JBuilder.



Tutorial: La vista Histórico

En este tutorial se explica el uso de la vista Histórico. La vista Histórico está disponible en todas las ediciones de JBuilder, aunque las funciones pueden variar de una a otra. Este tutorial hace referencia a todas las funciones disponibles, organizadas por ediciones. Los usuarios de todas las ediciones de JBuilder pueden completar las dos primeras lecciones. Los usuarios de las versiones Developer y Enterprise de JBuilder pueden completar las tres primeras. Los usuarios de JBuilder Enterprise pueden completar las cuatro lecciones. Los cambios realizados en este tutorial son extremadamente sencillos, con el fin de que pueda centrarse más en las funciones del Histórico que en el código.

La terminología que describe la gestión de las versiones varía según la herramienta. En este tutorial se utilizan términos genéricos para las utilidades de gestión de versiones de JBuilder independientes de la herramienta. Si desea una explicación adicional de esos términos, consulte el [“Glosario de gestión de versiones” en la página 12-1](#).

El uso de los sistemas de control de versiones se analiza en los casos necesarios. El tutorial se puede utilizar con o sin sistema de control de versiones (los pasos que se aplican sólo a los usuarios de sistemas de control de versiones se indican claramente). JBuilder integra Concurrent Versions System (CVS), Microsoft Visual SourceSafe (sólo ediciones Developer y Enterprise de JBuilder), Borland StarTeam (sólo ediciones Developer y Enterprise de JBuilder) y Rational ClearCase (sólo JBuilder Enterprise).

Nota Los pasos de este tutorial muestran cómo se utiliza la vista del histórico y cómo se realiza un seguimiento y se gestionan las versiones de archivos con o sin un sistema de control de versiones. El tutorial no proporciona información completa sobre cómo gestionar las revisiones de archivos con JBuilder. Si desea obtener más información acerca de la gestión de revisiones de archivos en JBuilder sin un sistema de control de versiones, consulte [Capítulo 12, “Comparación de archivos y versiones”](#). Si desea obtener más información acerca del uso de sistemas de control de versiones con JBuilder, consulte *Desarrollo en equipo con JBuilder* Desarrollo en equipo con JBuilder. El apartado Accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder por parte de personas con discapacidades. Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte [“Convenciones de la documentación” en la página 1-5](#).

Paso 1: Generación de varias versiones de un archivo

En primer lugar es necesario generar varias versiones diferentes de un archivo, de manera que haya algo que ver en la vista Histórico. A lo largo de este tutorial, trabajará con archivos del proyecto de ejemplo Welcome.

Para realizar cambios en un archivo fuente:

- 1 Abra el proyecto de ejemplo Welcome.

El ejemplo, `Welcome.jpx`, está situado en el subdirectorio `samples/welcome` del directorio de instalación de JBuilder.

Nota El siguiente paso sólo es aplicable a usuarios de sistemas de control de versiones.

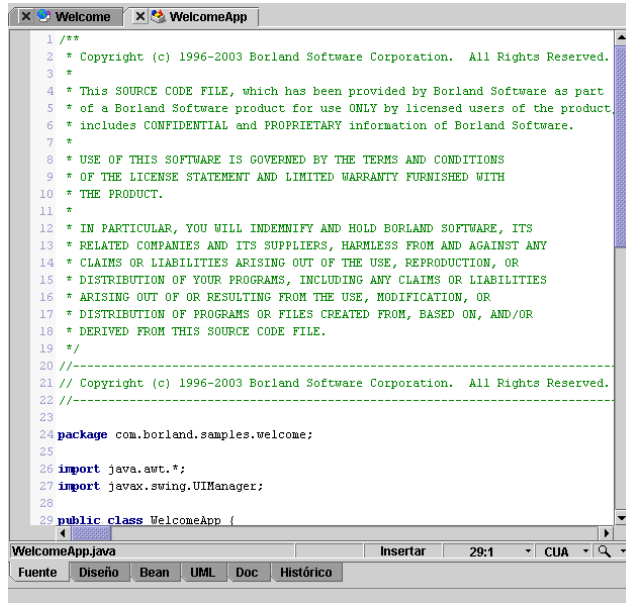
- 2 Si utiliza un sistema de control de versiones: coloque el proyecto bajo el control de versiones.

Nota Si no tiene los derechos de acceso necesarios para colocar un proyecto en el sistema de control de versiones pero desea conocer las funciones de control de versiones que ofrece la vista Histórico, puede utilizar la instalación de CVS incluida en JBuilder. CVS permite crear un repositorio local controlado completamente por el usuario. Si desea más información sobre el control de versiones en JBuilder, consulte *Desarrollo en equipo con JBuilder*. Si desea obtener información específica sobre CVS, consulte "CVS en JBuilder" en *Desarrollo en equipo con JBuilder*.

- 3 Haga doble clic en `WelcomeApp.java`, en el panel del proyecto, si aún no está abierto y activo en el panel de contenido.

4 Observe la barra de estado del panel de contenido.

Los dos números de la barra de estado indican la línea y la columna de la posición actual del cursor.



```

1  /**
2   * Copyright (c) 1996-2003 Borland Software Corporation. All Rights Reserved.
3   *
4   * This SOURCE CODE FILE, which has been provided by Borland Software as part
5   * of a Borland Software product for use ONLY by licensed users of the product,
6   * includes CONFIDENTIAL and PROPRIETARY information of Borland Software.
7   *
8   * USE OF THIS SOFTWARE IS GOVERNED BY THE TERMS AND CONDITIONS
9   * OF THE LICENSE STATEMENT AND LIMITED WARRANTY FURNISHED WITH
10  * THE PRODUCT.
11  *
12  * IN PARTICULAR, YOU WILL INDEMNIFY AND HOLD BORLAND SOFTWARE, ITS
13  * RELATED COMPANIES AND ITS SUPPLIERS, HARMLESS FROM AND AGAINST ANY
14  * CLAIMS OR LIABILITIES ARISING OUT OF THE USE, REPRODUCTION, OR
15  * DISTRIBUTION OF YOUR PROGRAMS, INCLUDING ANY CLAIMS OR LIABILITIES
16  * ARISING OUT OF OR RESULTING FROM THE USE, MODIFICATION, OR
17  * DISTRIBUTION OF PROGRAMS OR FILES CREATED FROM, BASED ON, AND/OR
18  * DERIVED FROM THIS SOURCE CODE FILE.
19  */
20  //-----
21  // Copyright (c) 1996-2003 Borland Software Corporation. All Rights Reserved.
22  //-----
23
24  package com.borland.samples.welcome;
25
26  import java.awt.*;
27  import javax.swing.UIManager;
28
29  public class WelcomeApp {

```

5 Ponga el cursor al final de la línea 27. El texto de esta línea es `import javax.swing.UIManager;`.

6 Pulse *Intro* al final de la línea 27 para añadir una línea por debajo.

7 Escriba `import java.applet.*;` en la línea 28.



8 Seleccione Archivo|Guardar archivo o pulse el icono Guardar archivo para guardar el archivo.

Esto genera una versión de copia de seguridad que se verá posteriormente. Las versiones de copia de seguridad se mantienen en el subdirectorio `bak` del directorio del presente proyecto. Por defecto, JBuilder mantiene las diez versiones de copia de seguridad más recientes. Para cambiar el nivel de copia de seguridad (el número de versiones de copia de seguridad que mantiene JBuilder), seleccione Herramientas|Opciones del editor, abra la ficha Editor si no se había abierto automáticamente y ajuste el control de graduación Nivel de copia de seguridad. Se puede tener un máximo de 90 copias de seguridad almacenadas en el directorio `bak`.

- 9** Cambie `import java.applet.*;` a `import java.applet.Applet;` y guarde el archivo.

Nota El siguiente paso sólo es aplicable a usuarios de sistemas de control de versiones.

- 10** Si utiliza un sistema de control de versiones: confirme el archivo y escriba el comentario `import.java.applet.*` cambiado por `import.java.Applet`.

Esto genera un tipo de control de versiones de la versión que se verá posteriormente.

- 11** Vaya a la línea 31, en la que se puede leer `boolean packFrame = false;`, y cambie `false` por `true`.

- 12** Guarde el archivo.

Nota Va a realizar otro cambio, pero *no guarde el archivo* hasta que se le indique.

- 13** Sitúe el cursor en la intersección de la línea 37 y la columna 23, donde se puede leer `WelcomeFrame frame = new WelcomeFrame();`, y cambie el nombre del objeto `frame` por `frame2`.

El código fuente tiene ahora este aspecto:

```
package com.borland.samples.welcome;

import java.awt.*;
import javax.swing.UIManager;
import java.applet.Applet;

public class WelcomeApp {
    boolean packFrame = true;

    // Crear la aplicación
    public WelcomeApp() {
        WelcomeFrame frame2 = new WelcomeFrame();
    }
}
```

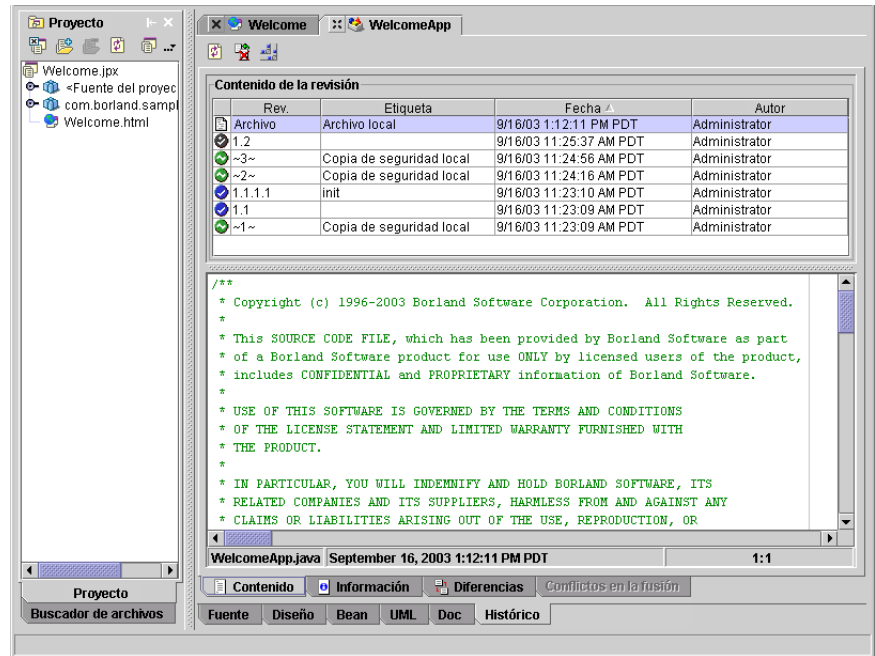
Nota JBuilder puede informar de errores cuando se cambia el nombre del objeto en el paso anterior. En este tutorial, no haga caso de los errores.

Paso 2: La ficha Contenido

La ficha Contenido muestra todas las revisiones del archivo seleccionado en el panel de contenido, permite ordenar los archivos de diversas formas y proporciona acceso a los botones Actualizar, Restaurar y Sincronizar desplazamiento.

1 Elija la pestaña del archivo Histórico.

Se muestra la ficha Contenido:



Nota La sección Contenido de la revisión, en la ficha Contenido, puede diferir de la ficha mostrada aquí, según se utilice o no el control de versiones.

La lista de revisiones de la parte superior presenta las versiones del archivo ordenadas por número de revisión, con las revisiones más recientes en la parte superior de la lista. El código fuente de la versión mas reciente se presenta en el visualizador de código fuente, debajo de la lista de revisiones.

Observe la información de la barra de estado: nombre del archivo, fecha y hora, y coordenadas de línea y columna.

2 Haga clic en la versión de un archivo para verla en el visualizador de código fuente.

JBuilder recupera la versión solicitada del lugar donde este almacenada, en el propio sistema de archivos o en el control de versiones.

- 3 Pulse el encabezado de la columna de iconos para ordenar las versiones por tipo de revisión.

Observe que se emplean iconos diferentes para los distintos tipos de versiones: copias de seguridad locales, control de versiones (si procede) y la última versión guardada:

Revisión extraída del sistema de control de versiones (tipo VCS)

Última revisión guardada (tipo universal)

Contenido de la revisión				
	Rev.	Etiqueta	Fecha ▲	Autor
📁	Archivo	Archivo local	16/06/03 11H28' CEST	Naranja
🔄	1.2		16/06/03 11H27' CEST	Naranja
🌿	~3~	Copia de seguridad local	16/06/03 11H27' CEST	Naranja
🌿	~2~	Copia de seguridad local	13/06/03 16H40' CEST	Naranja
🔗	1.1.1.1	init	13/06/03 16H40' CEST	Naranja
🔗	1.1		13/06/03 16H34' CEST	Naranja

Otras versiones del sistema de control de versiones (tipo VCS)

Copia de seguridad local (tipo universal)

- 4 Pulse el encabezado de la columna Fecha para ordenar las versiones de forma descendente por fecha de modificación.

- 5 Seleccione el elemento superior de la lista (~1~ Local Backup).

Esta versión aparece en el panel de código fuente debajo de la lista de revisiones.



- 6 Pulse el botón Sincronizar desplazamiento.

- 7 Desplácese hasta la línea 30, que tiene la sentencia `boolean packFrame`.

- 8 Seleccione la sentencia completa y cópiela:

The screenshot shows the JBuilder IDE interface. At the top, there are two tabs: 'Welcome' and 'WelcomeApp'. Below the tabs is the 'Contenido de la revisión' panel, which displays a table of revisions. The table has columns for 'Rev.', 'Etiqueta', 'Fecha', and 'Autor'. The revisions are listed in descending order of date. The top revision is '~1~' (Local Backup) with a date of 13/06/03 16H40' CEST. Below this is '~2~' (Local Backup) with a date of 16/06/03 11H27' CEST. The bottom revision is '1.1' with a date of 13/06/03 16H34' CEST. Below the table is the source code editor, which shows the code for the 'WelcomeApp' class. The code is as follows:

```

29 public class WelcomeApp {
30     boolean packFrame = false;
31
32     // Construct the application
33     public WelcomeApp() {
34         Welcome frame = new WelcomeFrame();
35
36         //Pack frames that have useful preferred size info, e.g. from their layout
37         //Validate frames that have preset sizes
38         if (packFrame)
39             frame.pack();
40     }
41 }

```

At the bottom of the IDE, there is a status bar showing the file name 'WelcomeApp.java~1~', the date and time '13 de junio de 2003 16:40:05 CEST', and the line number '30:29'. Below the status bar are several tabs: 'Contenido', 'Información', 'Diferencias', and 'Conflictos en la fusión'. At the very bottom, there is a row of buttons: 'Fuente', 'Diseño', 'Bean', 'UML', 'Doc', and 'Histórico'.

Elija Edición|Copiar o utilice el atajo de teclado. En la mayoría de las configuraciones de teclado, el método abreviado para copiar es *Ctrl + C*.

9 Seleccione la pestaña Fuente para volver al editor.

El texto que se presenta en el editor es el bloque que aparece en la ficha Contenido. Aunque la ventana de JBuilder sea pequeña, no es necesario desplazarse hacia abajo en el editor hasta encontrar la línea 31. Esto se debe a que se ha pulsado el botón Sincronizar desplazamiento.

10 En el editor, seleccione la línea 31 (la sentencia `boolean packFrame` en la última versión) y péguela en el texto copiado de la ficha Contenido de la vista Histórico.

Elija Edición|Copiar o utilice el teclado. En la mayoría de las configuraciones de teclado, el método abreviado para pegar es *Ctrl + V*.

11 Guarde el archivo.

La ficha Contenido de la ficha Histórico se puede utilizar para ver todas las versiones almacenadas de un archivo. Cuenta con los botones Restaurar y Sincronizar desplazamiento. Con el fin de proteger las versiones anteriores de los archivos, esta ficha las abre en modo de sólo lectura, y admite las operaciones de copiar y pegar texto de versiones anteriores al código fuente que se maneja actualmente.

Para obtener mas información sobre la vista del historial y las fichas Historial, consulte el [Capítulo 12, “Comparación de archivos y versiones”](#).

Paso 3: La ficha Diferencias

La ficha Diferencias muestra las diferencias entre dos versiones cualesquiera del archivo activo y proporciona acceso al botón Actualizar. Vamos a probarlo.

En este paso del tutorial aprenderá a:

- Ordenar revisiones en la sección Diferencias
- Deshacer cambios en una revisión seleccionada
- Utilizar Comparación inteligente para filtrar los cambios sin importancia al comparar versiones de archivos

Ordenación de las revisiones

Para ordenar la información de las revisiones:

- 1** Seleccione la pestaña Diferencias en el panel histórico.
- 2** Pulse el encabezado Rev. del área Entre para ordenar las versiones por número de revisión.
- 3** Pulse el encabezado de la columna de iconos del área En si desea ordenar las versiones por tipos de gestión de revisiones.

Observe que las áreas Entre y En se ordenan de forma independiente.

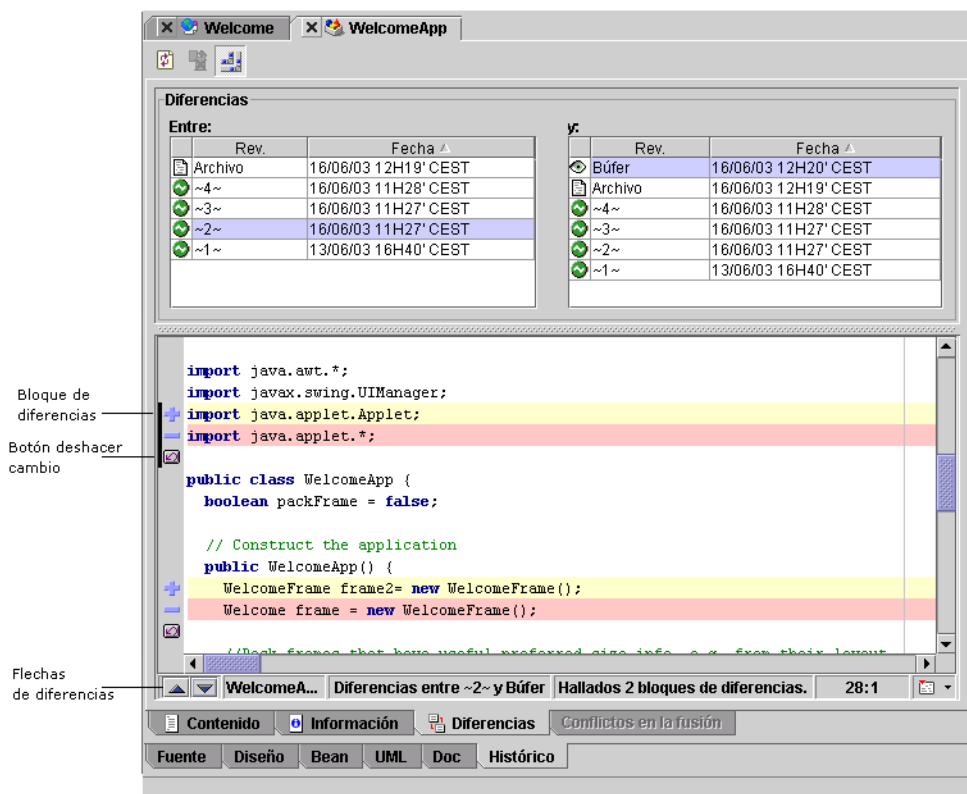
4 Pulse el encabezado Rev. del área En.

Cómo deshacer cambios

Ahora va a ver las diferencias entre dos versiones de un archivo.

1 Seleccione ~2~ en el área Entre y Buffer en el área y.

JBuilder recupera las diferencias y las muestra en el visualizador de código fuente:



En el área de estado, en la parte inferior, la ficha Diferencias indica el número de bloques de diferencias que hay en la comparación.

Si utiliza un sistema de control de versiones: Para ver todos los cambios realizados en el repositorio desde el comienzo de la sesión de trabajo, seleccione la versión original del repositorio en el área Entre y la versión del archivo con el número de revisión más alto en el área En.



2 Pulse el botón Actualizar.

Esta acción recupera las versiones nuevas del repositorio. No hay ninguna; por tanto, no cambia la lista de revisiones.

- 3 Para desplazarse de un bloque de código con diferencias a otro, pulse las flechas de diferencias de la izquierda de la barra de estado, situada en la parte inferior de la ficha Diferencias.

Como alternativa al uso de las flechas de diferencias para desplazarse entre los bloques, puede pulsar *Mayús+F10* para abrir un menú contextual (el cursor debe estar en el panel de la vista de código fuente) y seleccionar Anterior bloque de diferencias o Siguiente bloque de diferencias.

- 4 Desplácese al segundo bloque de diferencias y pulse el botón Deshacer cambio.

Aparece un menú contextual con una sola opción, Deshacer cambios anteriores.

- 5 Elija Deshacer cambios anteriores.

Ahora, el contenido de la línea es `WelcomeFrame frame = new WelcomeFrame();`.

`WelcomeFrame frame2 = new WelcomeFrame();` desaparece y se resuelve el bloque de diferencias.

Comparación inteligente

Es una función de
JBuilder Developer y
Enterprise.

La función Comparación inteligente permite excluir las diferencias debidas al formato del código y a cambios en los espacios en blanco cuando se comparan dos versiones de un archivo. Para observar cómo funciona Comparación inteligente:

- 1 Haga clic en la pestaña de la vista del archivo fuente.
- 2 Abra la ficha Formateo del cuadro de diálogo Propiedades de proyecto (Projecto|Propiedades de proyecto|Formateo).
- 3 En la ficha Formato, abra la pestaña Básico y pulse el botón Importar para abrir el cuadro de diálogo Importar configuración de formateo de código.
- 4 Seleccione el archivo de configuración `Expanded.codestyle` y pulse Aceptar.


El formato de código ampliado especificado por el archivo de configuración `Expanded.codestyle` aumenta significativamente la cantidad de espacios en blanco en los archivos.

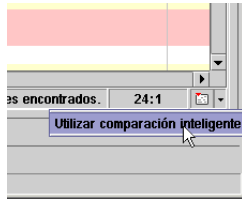
- 5 Pulse Aceptar para cerrar el cuadro de diálogo Propiedades de proyecto.
- 6 Seleccione Edición|Formatear todo para aplicar los cambios de formato a `WelcomeApp.java`.


Observe cómo cambia el formato del código en el archivo.

- 7 Vuelva al panel histórico y seleccione la pestaña Diferencias.

Observe el incremento de bloques de diferencias debido al cambio de formato. Los bloques de diferencias adicionales se deben simplemente a las líneas y los espacios añadidos.

-  **8** Pulse la flecha abajo situada junto al botón Comparación inteligente (ubicado en la esquina superior derecha de la ficha Diferencias) y seleccione la opción Utilizar Comparación inteligente.



-  El botón Comparación inteligente cambia, lo cual indica que la función de comparación inteligente está activada. Cuando se activa Comparación inteligente, todos los bloques, excepto el de las diferencias originales, desaparecen. La función de comparación inteligente puede resultar especialmente útil cuando varios desarrolladores comparten archivos y éstos presentan formatos adaptados a las preferencias de cada desarrollador, o bien se utiliza un formato global para guardar la coherencia.

La ficha Diferencias de la vista Histórico permite ver las diferencias entre todas las versiones disponibles de un archivo, ordenar las versiones de diversas formas, resolver las diferencias entre el búfer y otra versión, y desplazarse sin dificultad por las diferencias entre dos versiones.

Para obtener mas información sobre la vista del historial y las fichas Historial, consulte el [Capítulo 12, “Comparación de archivos y versiones”](#).

Consulte

- "Aplicación de formato al código" en *Introducción a JBuilder* si desea obtener más información sobre el modo de especificación y aplicación de preferencias de formato de código.

Paso 4: La ficha Información

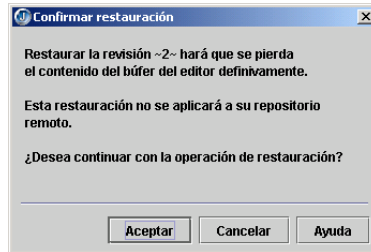
La ficha Información proporciona acceso a los comentarios y a las etiquetas de todas las revisiones de un archivo. Sus seis columnas se pueden utilizar como criterio de ordenación. Esta ficha también cuenta con los botones Restaurar y Actualizar.

- 1** Abra la pestaña Información.
Debajo de la lista de revisiones aparecen la etiqueta y el comentario de la versión del archivo seleccionada en la lista.
- 2** Haga clic en una versión para ver su etiqueta y su comentario.
- 3** Pulse el encabezado de la columna de comentarios de la lista de versiones para ordenarla por comentarios.
- 4** Seleccione la versión número ~2~ de la copia de seguridad local.



5 Pulse el botón Restaurar.

Aparece el cuadro de diálogo Confirmar restauración.



Si se realiza la restauración desde una versión anterior, ésta pasa a ser la actual. Si el archivo está bajo el control de versiones, o si la capacidad de copia de seguridad es suficientemente alta, se conservan todas la versiones intermedias.

Si utiliza un sistema de control de versiones: El archivo restaurado se debe incorporar en el control de versiones, como cualquier otro cambio realizado en los archivos.

6 Pulse Aceptar.

De esta forma, el contenido del búfer del editor del archivo vuelve al estado de la copia de seguridad local ~2~.

7 Haga clic en la pestaña de la vista del archivo fuente.

Tenga en cuenta que la sentencia `import` (y el resto del contenido) ha vuelto al estado que tenía después de que se guardara el archivo por primera vez en este tutorial.

Ha utilizado las fichas Contenido, Información y Diferencias de la vista Histórico. Ya sabe cómo ver diferentes revisiones de un archivo, buscar y ver sus comentarios y etiquetas, y examinar las diferencias entre las diversas versiones de copia de seguridad, búfer, repositorio y espacio de trabajo de los archivos.

Para obtener mas información sobre la vista del historial y las fichas Historial, consulte el [Capítulo 12, “Comparación de archivos y versiones”](#).

Consulte

- El tutorial CVS en JBuilder en *Desarrollo en equipo con JBuilder* para ver un ejemplo de cómo utilizar la ficha Conflictos en la fusión.



Tutorial: Compilación, ejecución y depuración

Este tutorial es una característica de las ediciones Developer y Enterprise de JBuilder.

En este tutorial aprenderemos, paso a paso, a encontrar y resolver errores de sintaxis, de compilación y de ejecución en un archivo de ejemplo incluido en JBuilder.

- Los errores de sintaxis afloran antes de la compilación. Los errores de sintaxis corresponden a código que no cumple los requisitos sintácticos del lenguaje Java.
- Los errores de compilación son los generados por el compilador: la sintaxis puede ser correcta, pero el compilador no puede compilar el código porque faltan variables o clases o porque hay sentencias incompletas. La verdadera causa del error podría encontrarse en una o más líneas antes o después del número de línea especificado en el mensaje de error.
- Los errores en tiempo de ejecución suceden cuando su programa puede ser compilado satisfactoriamente pero genera excepciones en tiempo de ejecución o se bloquea cuando se le ejecuta. Las sentencias del programa son sintácticamente correctas, pero provocan errores cuando se ejecutan.

El tutorial emplea el proyecto de ejemplo incluido en la carpeta `<jbuilder>/samples/Tutorials/DebugTutorial`. Se trata de una sencilla calculadora. Se han introducido errores en el programa con el fin de que no se compile ni se ejecute correctamente. Para que llegue a ejecutarse es preciso seguir este tutorial, encontrando y solucionando los errores.

Importante

Es posible que los números de línea mencionados en este tutorial no coincidan con los que se muestran en JBuilder.

Si desea información más detallada sobre la compilación, ejecución y depuración, consulte los siguientes apartados:

- [Capítulo 6, “Generación de programas en Java”](#)
- [Capítulo 5, “Compilación de programas en Java”](#)
- [Capítulo 7, “Ejecución de programas en JBuilder”](#)
- [Capítulo 8, “Depuración de programas en Java”](#)

El apartado Accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder por parte de personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte [“Convenciones de la documentación” en la página 1-5](#).

Paso 1: Abrir el proyecto de ejemplo

En este paso, abriremos el archivo de proyectos y uno de los archivos del proyecto. Veremos que hay errores de sintaxis en uno de los archivos.

Para abrir el proyecto de ejemplo:

- 1 Seleccione Archivo|Abrir proyecto. Se abre el cuadro de diálogo Abrir proyecto.
- 2 Desplácese a la carpeta `samples/Tutorials/DebugTutorial` de la instalación de JBuilder.
- 3 Haga doble clic sobre `DebugTutorial.jpx`. El proyecto se abre en el panel del proyecto. Amplíe el nodo de paquetes `DebugTutorial` para ver las dos clases de la aplicación:
 - `Application1.java`: El archivo ejecutable, que contiene el método `main()`.
 - `Frame1.java`: Archivo que contiene el marco, los componentes y los métodos del programa.
- 4 Haga doble clic sobre `Frame1.java`. De este modo se abre el archivo en el editor y en el panel de estructura aparece su estructura.

Fíjese en la carpeta Errores del panel de estructura. En el Paso 2 del tutorial encontraremos y solucionaremos estos errores.

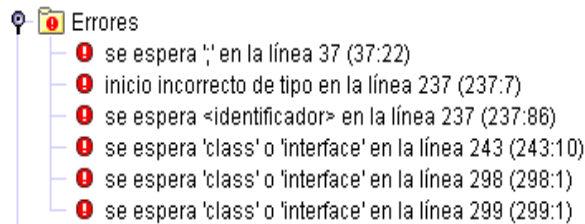
Paso 2: Solucionar errores de sintaxis

Los errores de sintaxis son situaciones en las que no se cumplen los requisitos sintácticos del lenguaje Java. JBuilder detecta estos errores antes de la compilación. Aparecen en una lista en la carpeta Errores del panel de estructura. Si intenta compilar el programa sin haber solucionado estos errores de sintaxis, JBuilder mostrará los errores en el panel de mensajes. El programa no puede compilarse hasta que se hayan solucionado estos errores.

En este paso, encontraremos los errores de sintaxis del programa de ejemplo y los solucionaremos. Para obtener más información acerca de los mensajes de error de JBuilder, consulte el tema "Mensajes de error del compilador".

Para encontrar y solucionar errores de sintaxis:

- 1 Examine la carpeta Errores del panel de estructura, si no está ya ampliada.



Aparecen seis errores. El primer error indica que falta un punto y coma en la columna 22, línea 37.

- 2 Haga clic sobre el primer error del panel de estructura. JBuilder desplaza el cursor a la línea de código correspondiente, en el editor. Se resalta la columna en la que ha tenido lugar el error. Observe el icono de error del margen.



Sugerencia

La barra de estado del panel de contenido muestra el número de línea y de columna, además del modo inserción y la configuración de teclado actual. Las flechas de la barra de estado abren las opciones del editor.



- 3 Inserte un punto y coma al final de la línea. Se ha solucionado el error, y éste desaparece del panel de estructura.
- 4 Haga clic sobre el siguiente error del panel de estructura. JBuilder desplaza el cursor a la línea de código correspondiente, en el editor. Este error es un poco más complicado de descifrar. El mensaje indica que se esperaba la presencia de un identificador de tipo en este punto del programa.

Observe que la línea de código empieza con la palabra clave `else` y que la línea siguiente se compone solamente de una llave de cierre. Si ha leído las líneas de código anteriores, habrá visto el principio de una sentencia `if`. En Java, las sentencias `if` deben incluir llaves de apertura y cierre.

Sin embargo, si observa la línea que contiene la sentencia `if`, verá que falta la llave de apertura.

- 5 Añada la llave de apertura al final de la línea. La línea de código completa será la siguiente:

```
if (valueOneIsOdd) {
```

Observe cómo la característica de coincidencia de llaves del editor muestra la llave de cierre coincidente.

Los otros dos errores de sintaxis desaparecen del panel de estructura. En este momento aparecen errores del compilador en la carpeta Errores. En el siguiente paso se solucionan estos errores.



- 6 Haga clic en el botón Guardar todo en la barra de herramientas principal.

Algunas veces hay que actuar como un detective para corregir los errores de sintaxis. A menudo, al solucionar un error de sintaxis se solucionan varios de los que aparecen en el panel de estructura. En nuestro caso, por ejemplo, el tercer error de sintaxis era: `se espera 'clase' o 'interfaz'` en la línea 227. Debido a que la llave de cierre no tiene una llave de apertura correspondiente, JBuilder espera encontrar una declaración de clase tras el cierre del método actual. Sin embargo, al insertar la llave de apertura, JBuilder detectó que esta llave tenía su pareja y que la siguiente línea de código no contenía ningún error.

Sugerencia

Puede encontrar las llaves correspondientes moviendo el cursor hasta ellas. La llave correspondiente aparece resaltada.

En el siguiente paso, podrá encontrar y corregir errores que impedirían que este programa se compilara.

Paso 3: Solucionar errores de compilación

En este paso del tutorial, podrá buscar y solucionar errores que podrían impedir la compilación de su programa. Estos errores se muestran en la carpeta Errores del panel de estructura.

Para encontrar y solucionar errores de compilación:

- 1 Haga clic sobre el primer error de la carpeta Errores:

```
no se puede resolver el símbolo: constructor Double() en  
la clase java.lang.Double en la línea 38 (38:27)
```

JBuilder coloca el cursor en la línea de código correspondiente:

```
Double valueOneDouble = new Double();
```

El mensaje de error indica que la clase Java `java.lang.Double` no contiene un constructor sin parámetros. La sentencia resaltada intenta crear un objeto `Double` sin parámetro. Si presta atención a los constructores de la clase `java.lang.Double`, verá que todos ellos requieren un parámetro.

Además, si se fija en unas líneas más adelante, verá que el objeto `Double`, `valueTwoDouble`, está construido con un valor inicial de `0.0`.

Sugerencia

Coloque el cursor entre los paréntesis y pulse *Ctrl+Mayús+Espacio* para ver ParameterInsight, la ventana emergente de JBuilder que muestra el tipo de parámetro necesario. También puede hacer clic con el botón derecho sobre el método `Double()` y seleccionar **Buscar definición** para abrir el código fuente en el editor.

- 2 Escriba `0.0` entre los paréntesis. Ahora, la sentencia tiene la siguiente forma:

```
Double valueOneDouble = new Double(0.0);
```

Sugerencia

La barra de estado del panel de contenido muestra la palabra `Modified`, lo cual indica que ha realizado cambios en el archivo.



- 3 Haga clic en el botón **Guardar todo** en la barra de herramientas.
- 4 Haga clic sobre el siguiente error del panel de estructura:

```
no se puede resolver el símbolo: variable subtractresultDisplay en
la clase DebugTutorial.Frame1 en la línea 243 (243:5)
```

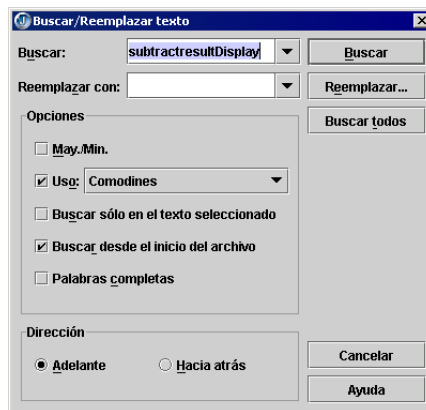
Este error indica que no se ha definido la variable `subtractresultDisplay`.

- 5 Seleccione **Buscar|Buscar** para abrir el cuadro de diálogo **Buscar/Reemplazar texto**.

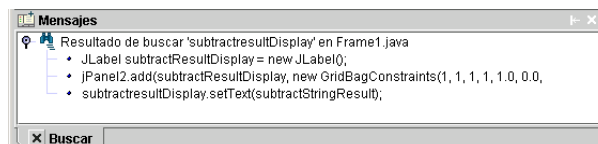
Sugerencia

Si el comando **Buscar** aparece atenuado, haga clic en el editor y vuelva a seleccionar **Buscar|Buscar**.

- 6 Escriba `subtractresultDisplay` en el campo **Buscar**. Asegúrese de que la opción **May./Min.** está desactivada. Haga clic sobre la opción **Buscar** desde el inicio del archivo para empezar la búsqueda desde ese punto.



- 7 Haga clic en **Buscar todos**. Los resultados aparecen en la pestaña **Buscar** del panel de mensajes.



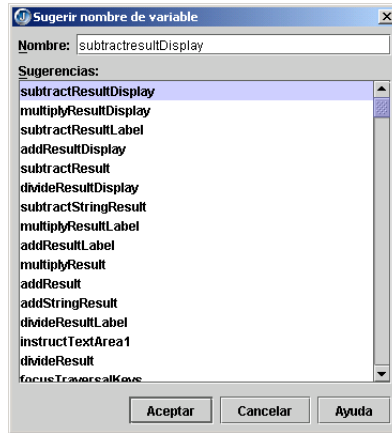
Paso 3: Solucionar errores de compilación

Observe que dos de las tres referencias a esta etiqueta son `subtractResultDisplay`; hay una **R** mayúscula en la palabra `Result`. La distinción entre mayúsculas y minúsculas es fundamental en Java: `subtractresultDisplay` no es lo mismo que `subtractResultDisplay`.

- 8 Haga doble clic sobre la referencia incorrecta en la pestaña Buscar (la última de la lista) para desplazar el cursor hasta la referencia del editor.



- 9 Pulse el icono ErrorInsight en el margen y elija Sugerir corrección para abrir el cuadro de diálogo Sugerir nombre de variable.



- 10 Asegúrese de que está seleccionado `subtractResultDisplay` en la parte superior de la lista y pulse Aceptar. El nombre de la variable `subtractResultDisplay` (observe la **R** mayúscula) se introduce en el código, y el error se elimina de la carpeta Errores.
- 11 Haga clic en la X de la pestaña Buscar para cerrarla. (También puede hacer clic con el botón derecho del ratón en la pestaña y seleccionar Eliminar la pestaña "Buscar").

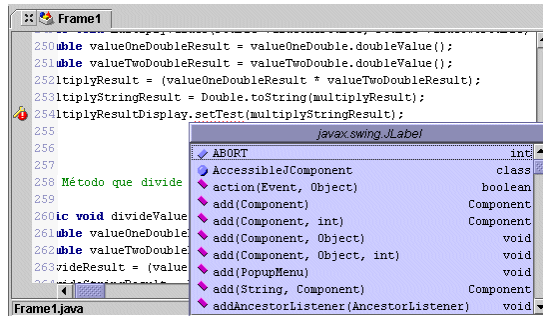
Use CodeInsight para solucionar los demás errores:

- 1 Haga clic sobre los restantes errores de la carpeta. Este error indica que no existe un método `setTest()` en `javax.swing.JLabel`.

```
no se puede resolver el símbolo: no se encontró  
setTest(java.lang.String) en  
la clase javax.swing.JLabel en la línea 254 (254:27)
```

JBuilder coloca el cursor en la línea de código correspondiente.

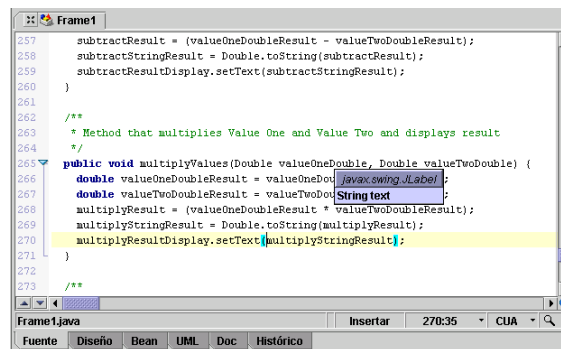
- 2 Sitúe el cursor detrás del punto (.) y pulse **Ctrl+Espacio**. De este modo se abre la ventana emergente CodeInsight que muestra las funciones miembro disponibles.



Nota

Si no aparece la ventana emergente, consulte "Configuración de teclado para las emulaciones del editor" (Ayuda/Configuraciones de teclado) para ver una lista de teclas abreviadas de CodeInsight.

- Desplácese por la ventana utilizando las teclas de desplazamiento. Los elementos resaltados en negrita están en esta clase. Los elementos tachados son desaconsejados. Los elementos atenuados son heredados, pero pueden utilizarse.
- Busque `setText` escribiendo `setText` o desplazándose por la lista. Una vez seleccionado, haga doble clic sobre él o pulse **Intro**. El método `setText()` queda insertado en el editor tras el punto, reemplazando al nombre incorrecto de método `setTest`. Una ayuda inmediata muestra el tipo de parámetros más lógico para el método.



- 3 Haga clic en el botón Guardar todo en la barra de herramientas.

En el paso siguiente se examina la configuración de ejecución y se ejecuta el programa.

Paso 4: Ejecutar el programa

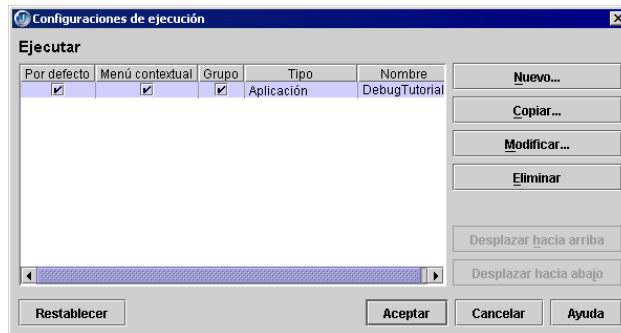
En este paso del tutorial se examina la configuración de ejecución del programa, y a continuación, se ejecuta el programa.

Las configuraciones de ejecución son parámetros de ejecución preestablecidos. La utilización de parámetros preestablecidos ahorra mucho tiempo durante la ejecución y la depuración, porque así sólo se definen una vez. Estas configuraciones preestablecidas permiten, cada vez que ejecute o depure la aplicación, simplemente seleccionar la configuración deseada.

Para obtener más información acerca de capturas, consulte [“Definición de las configuraciones de ejecución” en la página 7-8](#).

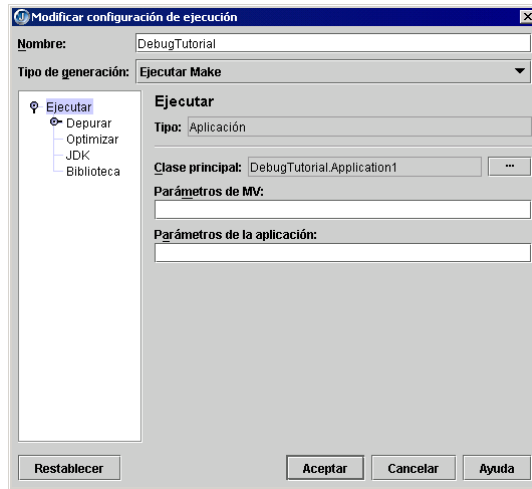
Para verificar la configuración para la ejecución de esta aplicación:

- 1 Seleccione Ejecutar!Configuraciones. Se muestra el cuadro de diálogo Propiedades de configuración de ejecución.



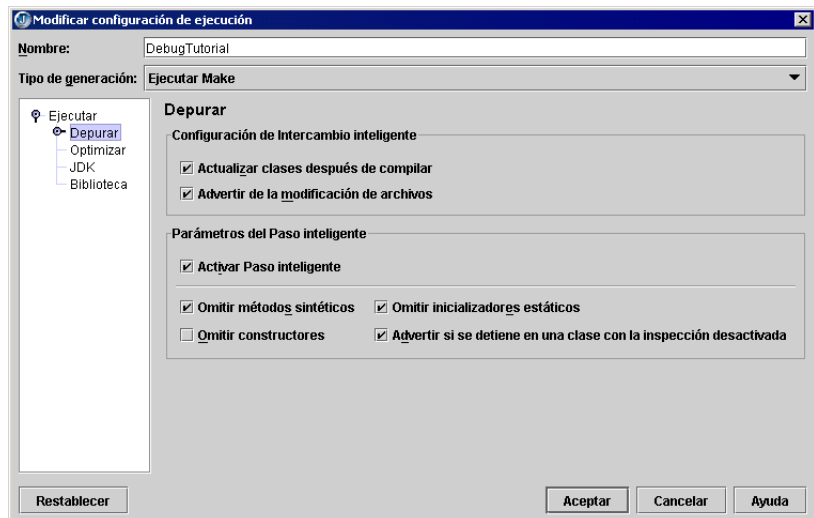
Existe una configuración predeterminada: DebugTutorial. Es una configuración de aplicación, lo que significa que para su ejecución se utiliza un ejecutor de aplicaciones. Se trata de la configuración por defecto, que se utiliza cuando se ejecuta el programa. También se muestra en el menú contextual cuando se hace clic con el botón derecho del ratón en el archivo ejecutable, `Application1.java`, en el panel del proyecto.

- 2 Pulse el botón Modificar. Se muestra el cuadro de diálogo Modificar configuración de ejecución.



Observe que el valor de Tipo es aplicación y que se utiliza el ejecutor de aplicaciones. El valor de Clase principal es `DebugTutorial.Application1`.

- 3 Abra el nodo Depurar para mostrar las propiedades de depuración de la configuración de ejecución. Aparece la ficha Depurar.



Observe que Intercambio inteligente (JBuilder Developer y Enterprise) y Paso inteligente se encuentran activados.

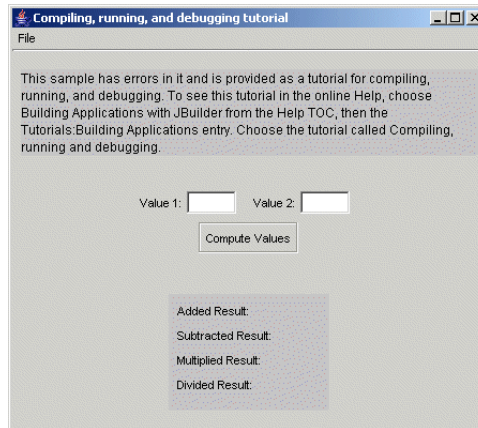
- 4 Pulse dos veces sobre Aceptar para cerrar los cuadros de diálogo.

Ejecución del programa

Ejecute el programa:



- 1 Haga clic en el botón Ejecutar en la barra de herramientas. El programa se ejecuta utilizando la configuración por defecto, DebugTutorial. La salida del compilador aparece en la ficha Application1, en el panel de mensajes. Aparece la interfaz de usuario del programa.



- 2 Introduzca numeros enteros en los campos de entrada Value 1 y Value 2 del programa. Pulse el botón Compute values. Los valores se calculan y aparecen en pantalla. Sin embargo, si observa con atención los resultados obtenidos, verá que hay algunos errores de ejecución; el programa se compila y ejecuta, pero da resultados incorrectos. En los siguientes pasos encontraremos y solucionaremos estos errores.
- 3 Elija File|Exit para cerrar la aplicación.
- 4 Haga clic sobre la X en la pestaña Aplicación1 del panel de mensajes para cerrarlo.

En el siguiente paso, podrá buscar y corregir un error de ejecución.

Paso 5: Corregir el método subtractValues()

En este paso del tutorial, encontraremos y solucionaremos uno de tres errores de ejecución. Para encontrar este error, nos serviremos del depurador. Se aprende a:

- Poner en marcha y detener el depurador.
- Crear una ventana flotante para una de las vistas del depurador.
- Definir un punto de interrupción.
- Inspeccionar un método y omitir su inspección.
- Seguir el rastro de un hilo.

- Definir un punto de observación `this`, un punto de observación de objeto y un punto de observación de una variable local.
- Utilizar el cuadro de diálogo Evaluar/Modificar.

En el paso anterior, se ejecutó el programa. Al introducir valores en los campos de introducción de datos Value 1 y Value 2 y pulsar Compute Values para obtener los valores de la suma, resta, multiplicación y división, quizá se diera cuenta de que el valor de la resta no era correcto. Por ejemplo, si introduce 4 en el campo Value 1 y 3 en el campo Value 2, el resultado de la resta es 0.0 en lugar de 1.0.

Para encontrar este error, utilizaremos el depurador. En primer lugar, fijaremos un punto de interrupción e iniciaremos el depurador.

- 1 Utilice el cuadro de diálogo Buscar/Reemplazar texto (Buscar|Buscar) para encontrar la línea de código que llama al método `addValues()`. Este es el primer método al que se llama cuando se pulsa el botón Compute Values. Escriba `addValues` en el campo Buscar del cuadro de diálogo para encontrar la llamada al método. Pulse el botón Buscar.
- 2 Haga clic en el margen gris del editor que está a la izquierda de la línea de código. En esta línea se fija un punto de interrupción. El círculo rojo indica que no se ha verificado el punto de interrupción.

```

207 //
208 void computeValuesButton_actionPerformed(ActionEvent e) {
209     valueOneText = inputValueTextField1.getText();
210     valueOneDouble = Double.valueOf(valueOneText);
211     valueTwoText = inputValueTextField2.getText();
212     valueTwoDouble = Double.valueOf(valueTwoText);
213     addValues(valueOneDouble, valueTwoDouble);
214     subtractValues(valueOneDouble, valueTwoDouble);
215     multiplyValues(valueOneDouble, valueTwoDouble);
216     divideValues(valueOneDouble, valueTwoDouble);
217     oddEven(valueOneDouble);
218     if (valueOneIsOdd) {
219         valueOneOddEvenLabel.setText("The first value enter");

```



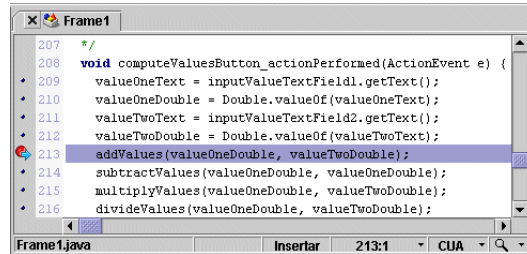
- 3 Haga clic en el botón Depurar programa de la barra de herramientas. JBuilder inicia el depurador MV, utilizando la configuración de ejecución de DebugTutorial.

El programa se está ejecutando y espera la entrada de datos (esto puede tardar unos segundos).


- 4 Escriba 4 en el campo Value 1 y 3 en el campo Value 2. Pulse Compute Values. Antes de que pueda examinar los resultados, el depurador toma el control. El programa queda minimizado y en el panel de mensajes aparece el depurador. En el editor aparecen ahora unos puntos azules junto a las

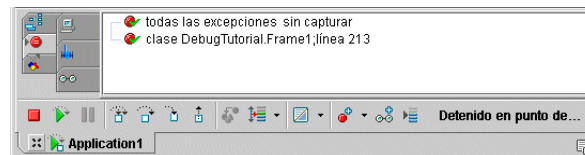
Paso 5: Corregir el método subtractValues()

líneas de código ejecutables, que indican dónde pueden fijarse puntos de interrupción válidos. La flecha indica el punto de ejecución (en este caso, la línea con punto de interrupción es también el punto de ejecución).




Si desea información más detallada acerca de la interfaz de usuario del depurador, consulte el [Capítulo 8, “Depuración de programas en Java”](#).

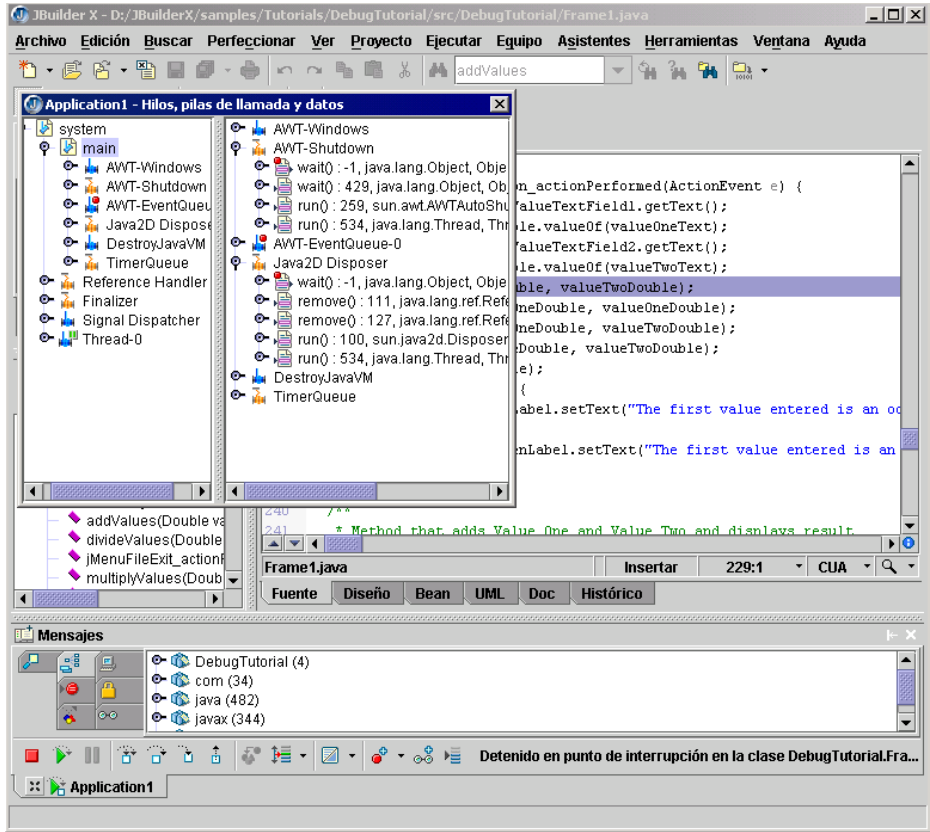
-  **5** Haga clic en la pestaña Puntos de interrupción situada en la parte izquierda del depurador para ir a la vista Puntos de interrupción de datos y código. En pantalla aparecen el punto de interrupción por defecto y el punto de interrupción que acaba de fijar. La barra de estado del depurador muestra un mensaje que indica que el programa se ha detenido en el punto de interrupción que se ha fijado en el editor.



El siguiente paso será inspeccionar el hilo que se está ejecutando paso a paso. Esto nos permitirá ver desde dónde se llama a los métodos y fijar puntos de observación sobre esos métodos.

-  **1** Abra la vista Hilos, pilas de llamadas y datos. Observe que la vista está dividida, lo cual permite ver, en el panel derecho, el contenido del elemento seleccionado en el panel izquierdo.
- 2** Haga clic con el botón derecho en un área vacía del panel izquierdo de la vista y seleccione Ventana flotante. La vista se convierte en una ventana flotante e inicialmente aparece en la parte superior izquierda de la pantalla. Puede cambiar las dimensiones de la ventana y moverla. Transformar una vista en ventana nos permite tener más de una vista del depurador

simultáneamente. (Observe que todas las vistas pueden transformarse en ventanas flotantes, excepto las vistas Consola, Salida, Entrada y Errores.)



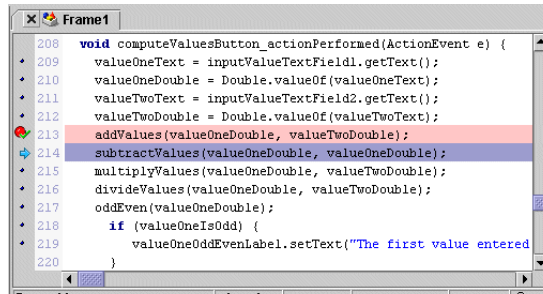
3 Desplácese por el editor para ver simultáneamente el punto de interrupción y la ventana flotante.



4 Para desplazarse a `subtractValues()`, que es el método que va a examinar, pulse el botón Omitir inspección de la barra de herramientas del depurador. De este modo se omite la inspección de la llamada al método `addValues()`, quedando el punto de ejecución en la llamada al método `subtractValues()`. El icono para el punto de interrupción ha cambiado a un punto rojo con una

Paso 5: Corregir el método subtractValues()

marca de comprobación verde que indica que el punto de interrupción es válido.

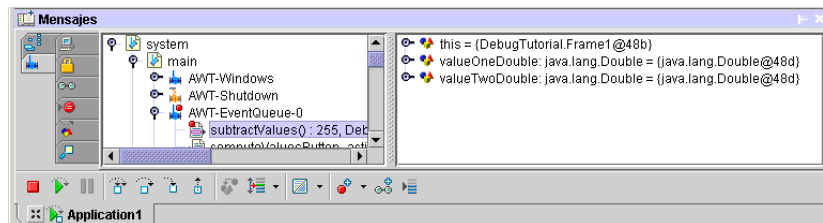


- 5 Pulse el botón Inspeccionar para inspeccionar el método `subtractValues()`. En estos momentos, el método `subtractValues()` aparece resaltado en el panel izquierdo de la vista flotante Hilos, pilas de llamadas y datos.
- 6 Haga clic con el botón derecho en un área vacía del panel izquierdo de la vista flotante Hilos, pilas de llamadas y datos y desactive la selección Ventana flotante para cerrarla. La ventana flotante vuelve a mostrarse como vista del depurador.

Sugerencia

Si desea devolver a las pestañas del depurador su orden por defecto, haga clic con el botón derecho en un área vacía de la vista y seleccione.

- 7 Abra la vista Hilos, pilas de llamadas y datos. Observe que el método `subtractValues()` aparece seleccionado en la parte izquierda de la vista y ampliado en el panel de la derecha.



Sugerencia



Puede pulsar el botón Mostrar marco de la pila actual para mostrar el hilo que se está inspeccionando.

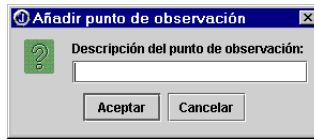
El siguiente paso será fijar puntos de observación sobre objetos y variables. Esto nos permitirá examinar los valores de los datos.

- 1 Cree un punto de observación sobre el objeto `this` haciendo clic con el botón derecho sobre el objeto `this` en la lista desplegada:

```
this = {DebuggerTutorial.Frame1@3c6}
```

Seleccione el comando Crear punto de observación 'this'. Un punto de observación sobre el objeto `this` permite inspeccionar la instanciación actual de la clase.

- Se muestra el cuadro de diálogo Añadir punto de observación con el campo Descripción del punto de observación. Pulse Aceptar.



No es necesario introducir una descripción del punto de observación. Si se introduce una descripción, aparecerá en la misma línea que la expresión con punto de observación de la vista Puntos de observación de datos. Una descripción puede facilitar la localización de los puntos de observación en la vista.

- Vuelva a hacer clic con el botón derecho sobre el objeto `this`:

```
this = {DebuggerTutorial.Frame1@3c6}
```

Seleccione el comando Crear punto de observación de objetos. Se muestra el cuadro de diálogo Añadir punto de observación. Pulse Aceptar.

- Haga clic con el botón derecho sobre el objeto `valueOneDouble` de la lista desplegada (en el panel derecho), con el fin de crear un punto de observación sobre el primer valor que se pasa al método `subtractValues()`:

```
valueOneDouble:java.lang.Double={java.lang.Double@3c7}
```

Seleccione el comando Crear punto de observación de variable local. Se muestra el cuadro de diálogo Añadir punto de observación. Pulse Aceptar.

Sugerencia

Si desea presentar el valor del objeto `valueOneDouble` en forma de cadena, haga clic en él con el botón derecho del ratón y elija el comando `Mostrar toString()`:

```
valueOneDouble:java.lang.Double="4.0"
```

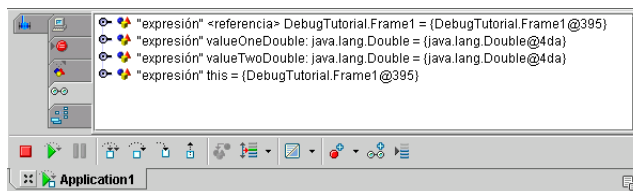
- Haga clic con el botón derecho sobre el objeto `valueTwoDouble` de la lista desplegada, con el fin de crear un punto de observación sobre el segundo valor que se pasa al método:

```
valueTwoDouble:java.lang.Double={java.lang.Double@3c7}
```

Seleccione el comando Crear punto de observación de variable local. Se muestra el cuadro de diálogo Añadir punto de observación. Pulse Aceptar.



- Abra la vista Puntos de observación de datos.



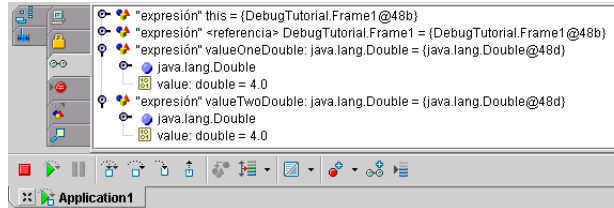
- Amplíe los dos primeros puntos de observación: el punto de observación `this` y el punto de observación `<reference>`. En este caso, ambos puntos de observación proporcionan los mismos datos, ya que son puntos de

observación idénticos. Observe que en esta vista se pueden ver todos los datos de los objetos (excepto los datos estáticos). Los elementos atenuados son heredados. Contraiga estos dos puntos de observación. Los otros dos puntos de observación (puntos de observación de variable local) observan los valores de `valueOneDouble` y `valueTwoDouble`.



8 Pulse el botón Inspeccionar una sola vez para continuar inspeccionando el método `subtractValues()`.

9 Amplíe los puntos de observación sobre `valueOneDouble` y `valueTwoDouble`.



Los dos valores son iguales. Sin embargo, no introdujo dos valores iguales en los dos campos de introducción de datos del programa.

Sugerencia

En la vista Puntos de observación de datos se puede ir rápidamente al método en el que se define la variable del punto de observación seleccionado. (Debe tratarse de un punto de observación de variable de ámbito.) Para ello, haga clic con el botón derecho del ratón en el punto de observación y seleccione Ir a punto de observación.

10 Fije un punto de observación sobre `subtractStringResult`, el resultado de la resta. Este valor de tipo `String` aparece en la etiqueta de salida. Para fijar el punto de observación, haga clic en el botón Añadir punto de observación de la barra de herramientas del depurador y escriba `subtractStringResult` en el campo Expresión. Pulse Aceptar. Quizá tenga que desplazarse por la vista Puntos de observación de datos para ver el punto de observación.



11 Pulse el botón Inspeccionar tres veces para pasar a la siguiente línea en el editor:

```
subtractResultDisplay.setText(subtractStringResult)
```

En la vista Puntos de observación de datos, el valor de `subtractStringResult` es 0.0 en lugar de 1.0, como sería de esperar.

Nota

También puede utilizar el cuadro de diálogo Evaluar/Modificar para examinar el valor de `subtractStringResult`. Para ello, seleccione Ejecutar! Evaluar/Modificar. Escriba `subtractStringResult` en el campo de introducción de datos Expresión y haga clic en Evaluar. En el campo Resultado aparece el resultado de la evaluación. Observe que la imagen en pantalla es igual que si se ampliara el punto de observación. Haga clic en Cerrar para cerrar el cuadro de diálogo.

12 Haga clic sobre el método dos veces más. El punto de ejecución vuelve a la línea desde la que se llama al siguiente método, `multiplyValues()`.

13 Fíjese en la llamada al método `subtractValues()`, la línea situada antes del punto de ejecución. Observe que `valueOneDouble` se pasa dos veces, en lugar de pasar `valueOneDouble` y `valueTwoDouble`. Cambie el segundo parámetro a `valueTwoDouble`.

Almacenamiento de archivos y ejecución del programa

Nota



En las versiones Developer y Enterprise de JBuilder no es necesario salir del depurador cuando se realizan los cambios. El botón Intercambio inteligente de la barra de herramientas del depurador modifica los archivos cambiados y actualiza las clases compiladas. Para hacer esto y continuar con el tutorial:

1 Haga clic en el botón Reanudar el programa de la barra de herramientas del depurador.

Nota

Si se abre el cuadro de diálogo Archivos modificados, elija la opción Compilar y actualizar las clases compiladas y pulse Aceptar.

2 Pulse tres veces Omitir inspección para definir el punto de ejecución en la llamada al método `divideValues()`.

3 Siga con el tutorial por el paso 8, en la siguiente sección.

Para guardar los cambios y ejecutar el programa sin utilizar Paso inteligente:



1 Haga clic en el botón Guardar todo en la barra de herramientas.



2 Haga clic en el botón Reanudar el programa de la barra de herramientas del depurador.



3 Haga clic en el botón Ejecutar en la barra de herramientas. Introduzca los valores. Se ejecuta el programa. Al introducir los valores y pulsar el botón Compute Values, el valor de la resta que ahora aparece es correcto. Sin embargo, si observa detenidamente los demás resultados, verá que el resultado de la división también es incorrecto. Prosiga con el Paso 6 para encontrar y solucionar el error.

4 Salga del programa. Elimine las pestañas del panel de mensajes. Para ello, haga clic con el botón derecho en la pestaña Application1 y seleccione Eliminar "Application1".

En el siguiente paso, se busca y corrige otro error de ejecución.

Paso 6: Corregir el método divideValues()

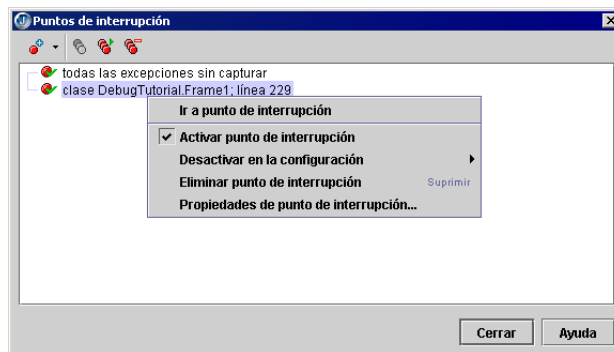
En este paso del tutorial, encontraremos y solucionaremos otro de los tres errores de ejecución. Fijaremos un punto de interrupción, inspeccionaremos un método y aprenderemos a utilizar la ayuda inmediata y ExpressionInsight para localizar errores.

En el paso anterior, se encontró y solucionó un error en la llamada al método `subtractValues()`. En este momento, al volver a ejecutar el programa, quizá se dé cuenta de que el resultado de la división también es incorrecto. Por ejemplo, si escribe 4 en el campo Value 1 y 2 en el campo Value 2, el resultado de la división es 8.0 en lugar de 2.0.

En primer lugar fijaremos un punto de interrupción, inspeccionaremos el método dudoso y utilizaremos ExpressionInsight y la ayuda inmediata.

- 1 Seleccione Ejecutar!Puntos de interrupción para eliminar el punto de interrupción fijado en el Paso 5. En el cuadro de diálogo Puntos de interrupción, haga clic con el botón derecho en este punto de interrupción:

`clase DebugTutorial.Frame1; línea 222; (sin verificar)`



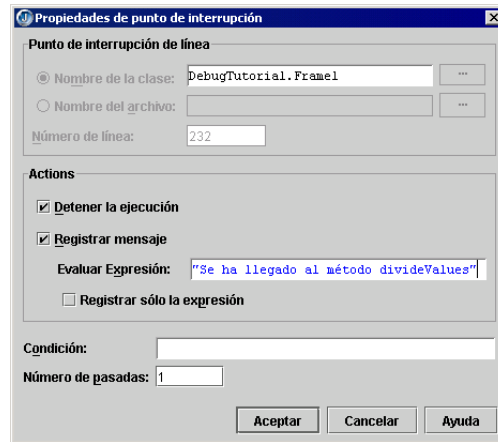
Seleccione Eliminar punto de interrupción y cierre el cuadro de diálogo pulsando Cerrar.

- 2 Utilice el cuadro de diálogo Buscar/Reemplazar texto para localizar la llamada al método `divideValues()`.
- 3 Fije un punto de interrupción en esta línea. Haga clic con el botón derecho en una línea con punto de interrupción, y elija Propiedades de punto de interrupción para abrir el cuadro de diálogo Propiedades de punto de interrupción.
- 4 Seleccione la opción Registrar mensaje. En el campo de introducción de datos Evaluar expresión escriba:

`"Se ha llegado al método divideValues"`

El mensaje quedará escrito en la vista Salida, entrada y errores de consola cuando se llegue al punto de interrupción especificado. Si además se

selecciona la opción Detener la ejecución, el programa se detendrá. El cuadro de diálogo tendrá un aspecto parecido a éste:



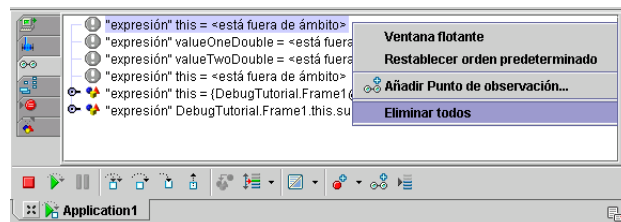
Haga clic en Aceptar para cerrar el cuadro de diálogo.

- 5 Haga clic en el botón Depurar de la barra de herramientas principal.
- 6 Introduzca 4 en la caja de entrada de Value 1 y 2 en la caja de entrada de Value 2 cuando aparezca la interfaz de usuario del programa. Pulse Compute Values. Recuerde que antes de que el usuario pueda examinar los resultados, el depurador toma el control y se muestra en el panel de mensajes.
- 7 Abra la vista Salida, entrada y errores de consola. Aparece el mensaje:

```
Alcanzado punto de interrupción en la clase DebugTutorial.Frame1 en la
línea 216
Expresión del histórico: "Se ha llegado al método divideValues" =
"divideValues
    método conseguido"
```

Durante el ciclo de desarrollo, puede utilizar esta característica en lugar de añadir sentencias `println` en el código. Esta función también se puede utilizar para evaluar expresiones.

- 8 Abra la vista Puntos de observación de datos y código. Observe que la mayoría de los puntos de observación ya están fuera de ámbito. Haga clic con el botón derecho del ratón en una zona vacía de la vista y seleccione Eliminar todos.





9 Pulse el botón Inspeccionar para inspeccionar el método `divideValues()`.

10 Pulse el botón tres veces más para superar la línea:

```
divideResult = (valueOneDoubleResult * valueTwoDoubleResult)
```

11 Coloque el cursor sobre la variable `divideResult` en el editor. En pantalla aparece una ayuda inmediata que muestra el valor de `divideResult`. Observe que el valor es incorrecto. Conforme a los datos introducidos, el resultado debería ser 2.0. Sin embargo es 8.0.

Nota

Si se ha utilizado Paso inteligente en el paso anterior de este tutorial, es posible que el valor sea distinto. (El resultado es incorrecto independientemente de los valores introducidos).

```

258  * Method that divides Value One and Value Two and displays result
259  */
260  public void divideValues(Double valueOneDouble, Double valueTwoDouble) {
261      double valueOneDoubleResult = valueOneDouble.doubleValue();
262      double valueTwoDoubleResult = valueTwoDouble.doubleValue();
263      divideResult = (valueOneDoubleResult * valueTwoDoubleResult);
264      divideStringResult = Double.toString(divideResult);
265      divideResultDisplay.setText(divideStringResult);
266  }
267
268  /*
  
```

También puede pulsar la tecla *Ctrl* y al mismo tiempo hacer clic con el botón derecho del ratón para abrir *ExpressionInsight*, cuando el cursor se encuentre sobre una expresión. Esta ventana emergente muestra el nombre, tipo y valor de la expresión. Si la expresión es un objeto, puede examinar los miembros del objeto, y también puede utilizar el menú emergente para fijar puntos de observación, modificar valores y cambiar los valores iniciales de presentación. Por ejemplo, coloque el cursor sobre `divideResultDisplay`. Pulse *Ctrl* junto con el botón derecho del ratón. Aparecerán los miembros del objeto `JLabel`. Desplácese por la lista: los elementos atenuados son heredados.

```

258  * Method that divides Value One and Value Two and displays result
259  */
260  public void divideValues(Double valueOneDouble, Double valueTwoDouble) {
261      double valueOneDoubleResult = valueOneDouble.doubleValue();
262      double valueTwoDoubleResult = valueTwoDouble.doubleValue();
263      divideResult = (valueOneDoubleResult * valueTwoDoubleResult);
264      divideStringResult = Double.toString(divideResult);
265      divideResultDisplay.setText(divideStringResult);
266  }
267
268  /*
  
```

Haga clic en el editor para cerrar la ventana *ExpressionInsight*. La ventana también se cerrará automáticamente al mover el cursor.

12 Lea detenidamente la línea del código fuente inmediatamente anterior al punto de ejecución:

```
divideResult = (valueOneDoubleResult * valueTwoDoubleResult)
```

¿Detecta el error? El método `divideResult()` multiplica los valores en lugar de dividirlos.

13 Para solucionar el error, cambie el operador `*` por `/`.

Almacenamiento de archivos y ejecución del programa

Guarde los cambios y ejecute el programa:

- 1** Haga clic en el botón Reanudar el programa de la barra de herramientas del depurador.
- 2** Haga clic en el botón Guardar todo en la barra de herramientas.
- 3** Elimine el punto de interrupción si todavía está en el archivo.
- 4** Haga clic en el botón Ejecutar en la barra de herramientas. Escriba valores en los campos Valor 1 y Valor 2. El programa se ejecuta y el valor dividido ya es correcto. Sin embargo, si observa detenidamente los demás resultados, verá el último error. Si en el campo Value 1 escribe un número impar, el programa informa erróneamente de que el valor es par. Si escribe un valor par, el programa dice que es impar.
- 5** Salga del programa. Elimine la pestaña Application1 del panel de mensajes.

En el paso siguiente, se busca y corrige el último error de ejecución de este tutorial.

Paso 7: Corregir el método oddEven()

En este paso del tutorial, encontraremos el último de los tres errores de ejecución. Se va a inspeccionar y omitir la inspección de un método, fijar un punto de observación y modificar un valor booleano sobre la marcha para probar una teoría.

En el Paso 6, se solucionó un error en el método `divideValues()`. Ahora, al volver a ejecutar el programa, observará que la sentencia que indica si el primer valor es par o impar es incorrecta.

Por ejemplo, si en el campo Value 1 escribe 4, el programa informa que es un número impar. Sin embargo, si escribe 3, el programa dice que el valor es par. En este paso, encontraremos y solucionaremos este error.

Encontraremos este error utilizando el cuadro de diálogo Evaluar/Modificar para evaluar el método que determina si el número es par o impar. A continuación, fijaremos un punto de observación en el resultado dado por el método para ver si se muestra en pantalla correctamente.

- 1 Utilice el cuadro de diálogo Buscar/Reemplazar texto para localizar la llamada al método `oddEven()` en `Frame1.java`. Observe que el nombre de la variable también contiene el texto `OddEven`. Para encontrar el método, puede activar la opción May./Min. en el cuadro de diálogo o buscar:
`oddEven(`

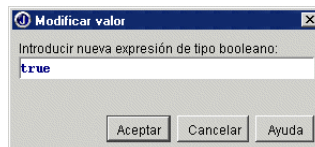
- 2 Fije un punto de interrupción en esta línea:

```
oddEven(valueOneDouble);
```

- 3 Pulse el botón Depurar.
- 4 Introduzca 3 en la caja de entrada de Value 1 y 4 en la caja de entrada de Value 2 cuando aparezca la interfaz de usuario del programa. Pulse el botón Compute Values. El foco vuelve al depurador.
- 5 Abra la vista Puntos de observación de datos. Fije un punto de observación sobre `valueOneIsOdd`. Tenga en cuenta que su valor es `true`, ya que se inicializó como `true`.
- 6 Pulse el botón Inspeccionar de la barra de herramientas del depurador. Cuando se inspecciona el método `oddEven()`, el valor de `valueOneIsOdd` sigue siendo `true`.
- 7 Haga clic en Inspeccionar tres veces más para avanzar en la inspección del método. Este método determina si el valor es par o impar. Al ir inspeccionando, el valor de `valueOneIsOdd` sigue siendo verdadero, `false`. ¿Es correcto? ¿El resultado de $(3 \text{ módulo } 2)$ es igual a cero? En realidad no es igual a cero, sino a uno, lo que indica que el valor es un número impar. A `valueOneIsOdd` se le debe asignar el valor `true`.

- 8 Para comprobar esta teoría, haga clic con el botón derecho sobre `valueOneIsOdd` en la vista Puntos de observación de datos y seleccione Modificar valor. Se muestra el cuadro de diálogo Modificar valor.

Escriba `true` y pulse Aceptar. A `valueOneIsOdd` se le debe asignar el valor `true`. Se acaba de cambiar el valor devuelto del método, de `false` a `true`.



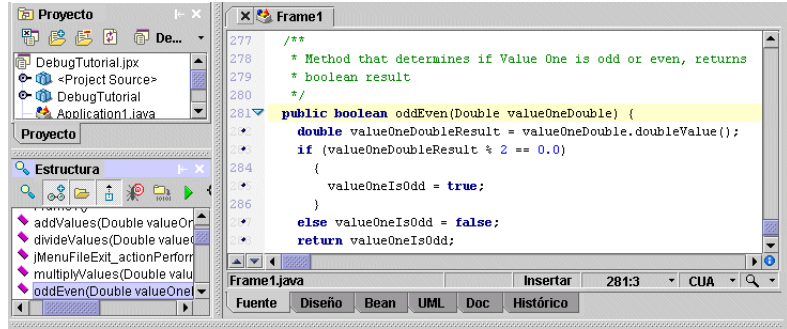
Haga clic en Aceptar para cerrar el cuadro de diálogo.



- 9 Haga clic en Abandonar la inspección para salir del método y volver al punto en el cual se realizó la llamada y a continuación seleccione Inspeccionar para inspeccionar la sentencia `if` de la siguiente línea de código.
- 10 Examine el contenido de la sentencia `if`. Es muy sencillo:

```
Si valueOneIsOdd es true, imprima el mensaje que dice que  
el número es impar. Sin embargo, si el valor es false, imprima  
el mensaje que indica que el número es par.
```

- 11 Pulse el botón Inspeccionar dos veces más. El punto de ejecución se desplaza a la sentencia `if`, donde pone: "Si el valor de `valueOneIsOdd` es true, imprima el mensaje que indica que el número es impar".
- 12 Haga clic sobre el método `oddEven()` en el panel de estructura para llegar a la ubicación del método en el editor. (Quizá tenga que desplazarse por el panel de estructura para localizar el método).



- 13 Examine la operación módulo y sus resultados. ¿Están asignados correctamente los resultados `true/false`? Si observa detenidamente, se dará cuenta de que los valores asignados `true` y `false` en realidad están trastocados. El código indica que si el módulo es igual a cero, el valor devuelto es `false` y el número es impar. Si el módulo no es igual a cero, el valor devuelto es `false` y el número es par. En realidad, estas sentencias se deberían invertir, dado que si el módulo es igual a cero, el número es par. El código debe ser el siguiente:

```

if (valueOneDoubleResult % 2 == 0.0)
{
    valueOneIsOdd = false;
}
else valueOneIsOdd = true;

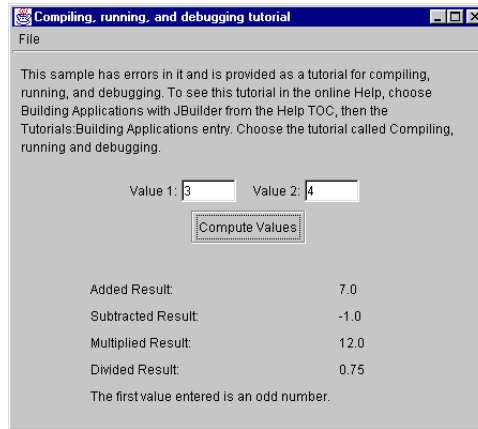
```

- 14 Intercambie los valores `true` y `false`.

Guarde los cambios y ejecute el programa:

- 1 Guarde los archivos.
- 2 Haga clic en el botón Reanudar el programa de la barra de herramientas del depurador.
- 3 Ejecute la aplicación.

- 4 Escriba 3 en el cuadro Value 1 y 4 en el cuadro Value 2. Pulse el botón Compute Values. El resultado es correcto. Ahora el programa afirma que el Value 1 es un número impar.



- 5 Salga del programa seleccionando File|Exit. Elimine la pestaña Application1.

En el siguiente paso, veremos qué sucede cuando se genera una excepción durante la ejecución.

Paso 8: Buscar excepciones producidas durante la ejecución

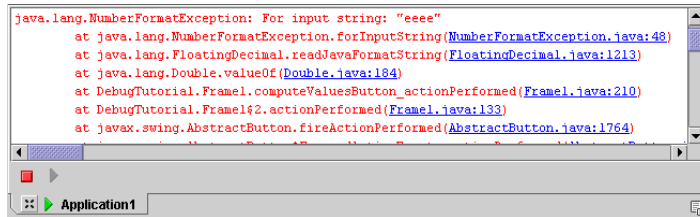
En este paso del tutorial, veremos lo que sucede cuando se genera una excepción durante la ejecución. El programa de ejemplo no cuenta con tratamiento de errores. Por ejemplo, si en los campos Value 1 y Value 2 se escribe un valor no numérico, el programa genera un seguimiento de la pila de excepciones producidas durante la ejecución. No informará acerca de que el valor no tiene el formato esperado ni facilitará información acerca de cuáles son los valores válidos.

Para ver un seguimiento de la pila de excepciones producidas durante la ejecución:

- 1 Ejecute la aplicación.
- 2 Escriba `eeee` en el campo Valor 1. Escriba 3 en el campo Valor 2. Pulse Compute Values.

- 3 Minimice el programa para ver el panel de mensajes. Vaya a la parte superior del panel.

La pestaña Application1 contiene un seguimiento de la pila NumberFormatException. Esto es un seguimiento de cómo el programa ha llegado a esta excepción.



```

java.lang.NumberFormatException: For input string: "eeee"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
    at java.lang.Float.parseFloat(Float.java:1213)
    at java.lang.Double.parseDouble(Double.java:184)
    at DebugTutorial.Frame1.computeValuesButton_actionPerformed(Frame1.java:210)
    at DebugTutorial.Frame1.actionPerformed(Frame1.java:133)
    at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:1764)

```

- 4 Seleccione el primer nombre de clase subrayado del seguimiento de la pila para ver dónde se generó la excepción. En este caso, seleccione NumberFormatException.

JBuilder abre el código fuente de `java.lang.Float.parseFloat` y resalta la línea del código donde se encuentra la excepción. Puede hacer clic en otras clases del seguimiento de la pila para examinar los pasos que llevaron al programa a esta excepción.

El tratamiento de esta excepción excede del ámbito de este tutorial. Para volver a ejecutar el programa sin la excepción, cierre el programa y ejecútelo de nuevo escribiendo valores numéricos.

¡Enhorabuena!. Ha llegado al final de este tutorial. Hemos encontrado y solucionado errores de sintaxis, errores de compilación y errores de ejecución con el depurador integrado de JBuilder. También hemos visto un ejemplo de un seguimiento de la pila de excepciones producidas durante la ejecución.

Si desea información más detallada sobre la compilación, ejecución y depuración, consulte los siguientes apartados:

- [Capítulo 6, “Generación de programas en Java”](#)
- [Capítulo 5, “Compilación de programas en Java”](#)
- [Capítulo 7, “Ejecución de programas en JBuilder”](#)
- [Capítulo 8, “Depuración de programas en Java”](#)



Tutorial: Depuración remota

Este tutorial es una característica de las ediciones Developer y Enterprise de JBuilder.

En este tutorial aprenderá paso a paso a:

- Utilizar las funciones de depuración remota para conectarse a un programa que ya se está ejecutando en un ordenador remoto.
- Depurar utilizando la inspección interprocesal.
- Utilizar configuraciones predefinidas para depurar tanto procesos de cliente como de servidor.

El tutorial emplea el proyecto de ejemplo incluido en la carpeta `<jbuilder>\samples\RMI`. Se trata de una aplicación RMI creada en JBuilder. Antes de ejecutar este tutorial, asegúrese de haber instalado la carpeta de ejemplos.

En este tutorial se presupone lo siguiente:

- El lector está utilizando Windows.
- El lector está familiarizado con la compilación, la ejecución y la depuración. Si no es así, consulte el [Capítulo 20, “Tutorial: Compilación, ejecución y depuración”](#). También puede leer los siguientes capítulos:
 - [Capítulo 6, “Generación de programas en Java”](#)
 - [Capítulo 5, “Compilación de programas en Java”](#)
 - [Capítulo 7, “Ejecución de programas en JBuilder”](#)
 - [Capítulo 8, “Depuración de programas en Java”](#)
- El lector está familiarizado con los procesos cliente/servidor en JBuilder.
- Ha leído el capítulo [Capítulo 9, “Depuración remota”](#).

Paso 1: Abrir el proyecto de ejemplo

- Se está familiarizado con las ventanas de DOS y con la ejecución de comandos desde la línea de comandos.

Para ejecutar este tutorial, necesita:

- Dos equipos conectados en red. JBuilder debe estar instalado en uno de ellos y el JDK 1.3 o posterior en el otro. En este tutorial, llamaremos equipo "cliente" al que tiene JBuilder. Aquel donde se encuentra el JDK se conocerá como "remoto". Este ordenador ejecutará el servidor.
- El nombre del host o la dirección IP del ordenador remoto. Este ID suele asignarlo el administrador de la red.
- Una forma de transferir archivos desde el ordenador cliente hasta el remoto.

Nota Para ejecutar el ejemplo sin depurarlo, siga las instrucciones del archivo HTML del proyecto, `SimpleRMI.html`.

El apartado Accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder por parte de personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte [“Convenciones de la documentación” en la página 1-5](#).

Paso 1: Abrir el proyecto de ejemplo

Este tutorial emplea el proyecto de ejemplo incluido en la carpeta `samples\RMI` de la instalación de JBuilder. Antes de ejecutar este tutorial, asegúrese de haber instalado la carpeta de ejemplos.

En este paso se abrirá el archivo de proyecto. Para abrir el proyecto de ejemplo:

- 1 Seleccione **Archivo** > **Abrir proyecto**. Se abre el cuadro de diálogo **Abrir proyecto**.
- 2 Desplácese a la carpeta `<jbuilder>\samples\RMI`.
- 3 Haga doble clic en `SimpleRMI.jpx`. El proyecto se abre en el panel del proyecto. Los archivos del proyecto se enumeran en el panel del proyecto. Este proyecto consta de seis archivos:
 - `SimpleRMI.html` - El archivo HTML que proporciona una descripción general del proyecto. Este archivo incluye instrucciones para la creación y ejecución de una aplicación RMI en JBuilder.
 - `SimpleRMI.policy` - El archivo de normas de seguridad. El archivo especifica los derechos que tiene el servidor RMI para monitorizar y aceptar peticiones del cliente en una red.
 - `SimpleRMIClient.java` - La clase cliente que se conecta al objeto servidor.
 - `SimpleRMIImpl.java` - La clase que implementa la interfaz del servidor RMI.

- SimpleRMIInterface.java - La interfaz RMI.
- SimpleRMIServer.java - La clase servidor que crea una instancia de la clase Impl.

En el Paso 2, se definirán las configuraciones de ejecución y depuración del cliente y del servidor.

Paso 2: Definir las configuraciones de ejecución y depuración

En este paso se definirán las configuraciones de ejecución y depuración para el cliente y el servidor. Si desea más información sobre configuración de ejecución y depuración, consulte [“Definición de las configuraciones de ejecución” en la página 7-8](#) y [“Configuración de las opciones de depuración” en la página 8-81](#).

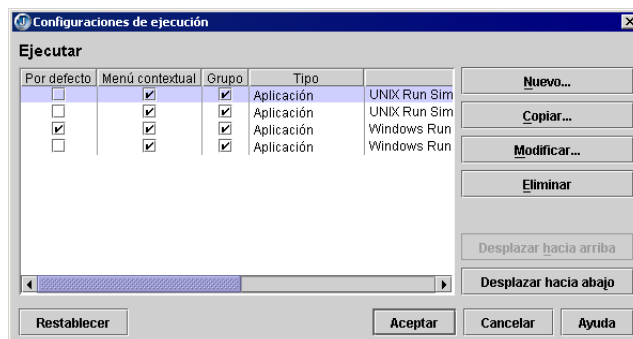
Para definir las configuraciones de este tutorial, se utilizan las fichas del cuadro de diálogo incluidas en la siguiente tabla.

Tabla 21.1 Fichas del cuadro de diálogo para definir las configuraciones de depuración y ejecución servidor y cliente

Nodo Configuraciones de ejecución	Se aplica a	Descripción
Nodo Ejecutar	Servidor (ejecuta en el ordenador remoto)	Configura los parámetros de ejecución para el servidor RMI.
Nodo DepurarRemota	Servidor (ejecuta en el ordenador remoto)	Configura cómo se asocia el servidor del ordenador del cliente al proceso del servidor remoto.
Nodo Ejecutar	Cliente (ejecuta en el ordenador con JBuilder)	Configura los parámetros de ejecución del servidor RMI.

Para definir las configuraciones de ejecución del servidor:

- 1 Seleccione EjecutarConfiguraciones. Se muestra el cuadro de diálogo Configuraciones de ejecución.



- 2 Elija la configuración denominada `Windows Run SimpleRMIserver`.
- 3 Pulse **Modificar** para abrir el cuadro de diálogo **Modificar configuración de ejecución**. Tenga en cuenta que el campo **Parámetros de MV** recoge los parámetros de la MV de Java, en este caso, la ubicación de los archivos de clase del servidor y el archivo de normas de seguridad.
- 4 Compruebe que el argumento `codebase` de los **Parámetros de MV** indique la ubicación de los archivos de clase de servidor. En una típica instalación de Windows, éste será la carpeta `classes` de la carpeta
`<jbuilder>\samples\RMI:`

Nota

Hace falta incluir la última barra invertida del argumento, después de `classes`.

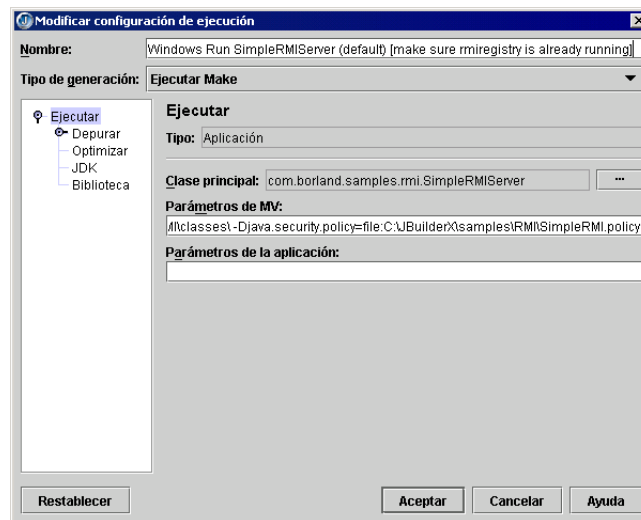
- 5 Compruebe que el argumento de normas de seguridad del campo **Parámetros de MV** indica la ubicación del archivo de normas de seguridad. El archivo de normas delimita los derechos que tiene el servidor RMI para monitorizar y aceptar peticiones del cliente RMI en una red. En una instalación típica de Windows, se tratará de la carpeta
`<jbuilder>\samples\RMI.`

`-Djava.security.policy=file:C:\<jbuilder>\samples\RMI\SimpleRMI.policy`

- 6 Cerciórese de que la clase principal sea:

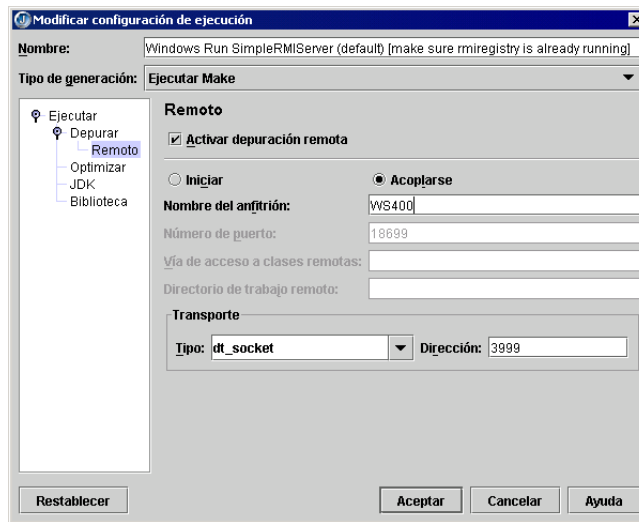
`com.borland.samples.rmi.SimpleRMIserver`

- 7 Cuando haya terminado, el nodo **Ejecutar** del servidor debería tener un aspecto parecido a éste:



Para definir la configuración de la depuración remota del servidor:

- 1 Haga clic en el nodo Depurar\Remoto.
- 2 Haga clic en la opción Activar depuración remota y luego en la opción Acoplarse.
- 3 En el campo Nombre del anfitrión, escriba el nombre del ordenador donde se ejecutará el servidor.
- 4 Deje el tipo de transporte en `dt_socket`.
- 5 Escriba la dirección del ordenador remoto en el campo Dirección. Volverá a utilizar este número cuando ejecute el servidor en el ordenador remoto (Paso 5 del tutorial). Para los objetivos de este tutorial, déjelo como `3999`.
- 6 Cuando haya terminado, el nodo Depurar\Remoto del servidor debería tener un aspecto parecido a éste:



- 7 Pulse Aceptar para cerrar el cuadro de diálogo Modificar configuración de ejecución.

A continuación, se configurará la ejecución del cliente.

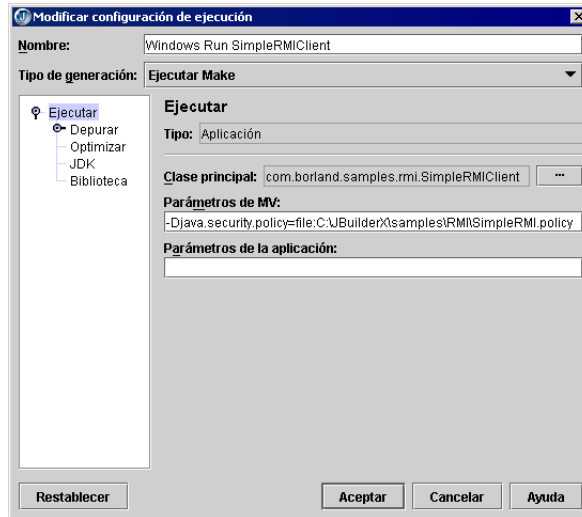
- 1 En el cuadro de diálogo Configuraciones de ejecución, seleccione la configuración de nombre `Windows Run SimpleRMIClient`.
- 2 Pulse Modificar para abrir el cuadro de diálogo Modificar configuración de ejecución. El campo Parámetros de MV recoge los parámetros de la MV de Java, en este caso, la ubicación del archivo de normas de seguridad.
- 3 Compruebe que el argumento del campo Parámetros de MV indica la ubicación del archivo de normas de seguridad. En una instalación típica de Windows, se tratará de la carpeta `<jbuilder>\samples\RMI`.

`-Djava.security.policy=file:C:\<jbuilder>\samples\RMI\SimpleRMI.policy`

- 4 Cerciórese de que la clase principal sea:

```
com.borland.samples.rmi.SimpleRMIClient
```

- 5 En el campo Parámetros de la aplicación, escriba el nombre del ordenador remoto. Éste es el nombre que escribió en el campo Nombre de anfitrión del nodo DepurarRemota del cuadro de diálogo Modificar configuración de ejecución para el servidor (consulte el apartado anterior).
- 6 Cuando haya terminado, la ficha Ejecutar del servidor debería tener un aspecto parecido a éste:



- 7 Pulse de nuevo Aceptar para cerrar el cuadro de diálogo Modificar configuración de ejecución.
- 8 Pulse Aceptar de nuevo para cerrar el cuadro de diálogo Modificar configuraciones de ejecución.

En el próximo paso, se definirán los puntos de interrupción en el cliente y el servidor.

Paso 3: Definir los puntos de interrupción

En este paso se definirá un punto de interrupción en el proceso del cliente y un punto de interrupción interprocesal en el proceso del servidor. El primer punto de interrupción hará que el cliente se detenga cuando esté a punto de llamarse al punto de interrupción interprocesal. El punto de interrupción interprocesal detendrá al servidor. Esta técnica le permite inspeccionar procesos del servidor desde un proceso del cliente.

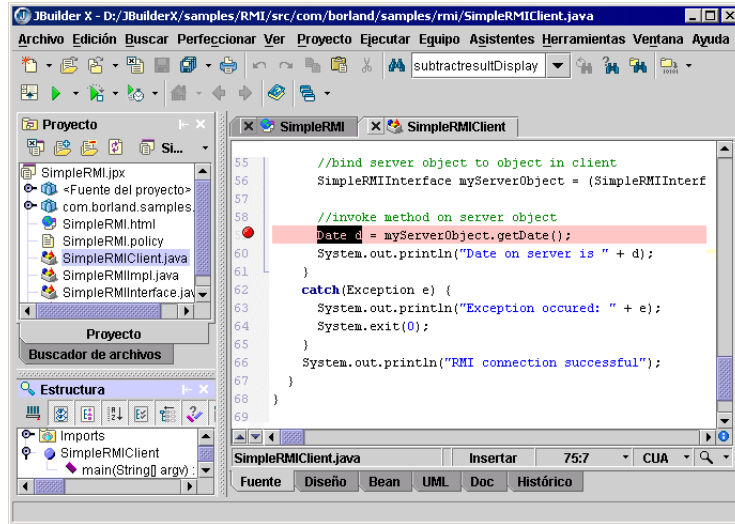
Antes de empezar, compile el proyecto (Proyecto|Ejecutar Make) para eliminar los errores del compilador del panel de estructura.

Para definir un punto de interrupción en una línea en el proceso del cliente:

- 1 Haga doble clic sobre `SimpleRMIClient.java` en el panel del proyecto. El archivo se abre en el editor.
- 2 Utilice Buscar|Encontrar para encontrar la cadena `Date d`. El cursor se posicionará en la siguiente línea:

```
Date d = myServerObject.getDate();
```

- 3 Haga clic sobre el margen, en el área gris a la izquierda de la línea de código, para definir un punto de interrupción en la línea.

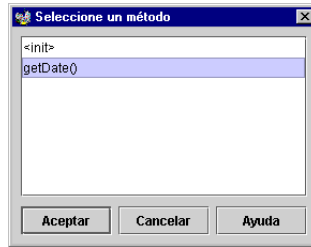


Para definir un punto de interrupción interprocesal en el proceso del servidor:

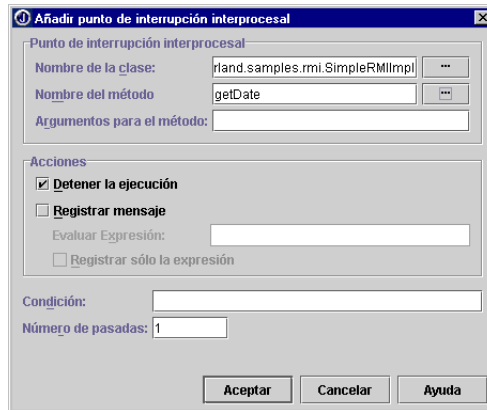
- 1 Seleccione Ejecutar|Añadir punto de interrupción|Añadir punto de interrupción interprocesal. Aparece el cuadro de diálogo Añadir punto de interrupción interprocesal.
- 2 Pulse el botón de puntos suspensivos (...) a la derecha del campo Nombre de la clase.
- 3 En el campo Buscar del cuadro de diálogo Seleccionar clases, escriba `SimpleRMIServerImpl`.
- 4 Pulse Aceptar para cerrar el cuadro de diálogo cuando haya seleccionado la clase.
- 5 Pulse el botón de puntos suspensivos (...) a la derecha del campo Nombre del método.

Paso 3: Definir los puntos de interrupción

- 6 Seleccione `getDate()` en el cuadro de diálogo Seleccione un método.



- 7 Haga clic en Aceptar para cerrar el cuadro de diálogo.
- 8 La opción Acción del cuadro de diálogo Añadir punto de interrupción interprocesal debe mostrar Detener la ejecución.
- 9 El cuadro de diálogo Añadir punto de interrupción interprocesal debe tener este aspecto:



- 10 Haga clic en Aceptar para cerrar el cuadro de diálogo.

En el próximo paso, compilaremos el servidor y copiaremos los archivos de clase del servidor al ordenador remoto.

Paso 4: Compilar el servidor y copiar los archivos de clase del servidor al equipo remoto

En este paso se compilará el servidor y se copiarán sus archivos de clase en el equipo remoto.

Seleccione Proyecto|Ejecutar Make de "SimpleRMI.jpj" para compilar los archivos del servidor en JBuilder. La barra de estado indica cuándo ha sido generado el proyecto.

Observe que en el panel del proyecto aparece un icono que indica que `SimpleRMIImpl.java` puede ampliarse. El compilador de RMI ha creado la clase stub `SimpleRMIImpl_Stub.java`. No modifique este archivo, ya que se genera automáticamente.

Abra una ventana de DOS y desplácese a la carpeta `<jbuilder>/samples/RMI`. Debe incluir un subdirectorio `classes`. El subdirectorio `classes` contiene una estructura de carpetas idéntica a la del paquete. Los archivos `.class` de servidor se almacenan en la carpeta `classes/com/borland/samples/rmi`. La carpeta `classes` también incluye las carpetas `dependency` `cache` y `Generated Source`.

Los archivos de clase del servidor han de copiarse al equipo remoto. En el caso de este ejemplo, puede copiarse toda la carpeta `RMI` al equipo remoto, con el mismo nombre, `RMI`. Para ello, puede proceder de una de las siguientes maneras:

- Copie los archivos en una red y posteriormente al equipo remoto.
- Copie los archivos a un disquete y de allí llévelos al equipo remoto.
- Envíe los archivos al equipo remoto por FTP.

Importante A partir de este punto, si actualiza los archivos fuente en el equipo cliente (donde se ejecuta JBuilder), deberá copiar los archivos `.class` de nuevo en el equipo remoto. De no hacerlo así, los archivos fuente y los compilados diferirán, produciendo errores.

En el siguiente paso, se iniciarán el registro de RMI y el servidor en el equipo remoto.

Paso 5: Iniciar el registro de RMI y el servidor en el equipo remoto

En este paso se iniciará el registro de RMI en el equipo remoto, así como el servidor, éste en modo depuración. Es imprescindible conocer los valores de RMI, así como la configuración de depuración Java de la línea de comandos que pone en marcha el servidor.

Para iniciar el registro de RMI en el equipo remoto:

- 1 Abra la ventana 4DOS o 4NT.
- 2 Cambie a la carpeta `<jdk>\bin`.
- 3 Inicie el registro de RMI mediante el siguiente comando:

```
start rmiregistry
```

El registro de RMI se pone en marcha en un proceso independiente. Si el registro no se inicia, puede que no haya memoria suficiente. Cierre las demás aplicaciones y la ventana de DOS e inténtelo de nuevo.

Para iniciar el servidor en el equipo remoto:

- 1 Abra una ventana de comandos. La línea de comandos de Java cuenta con más de 256 caracteres, por lo que no podrá ejecutarla en una ventana estándar de 4DOS o 4NT. Abra la ventana de comandos desde el menú Inicio: pulse sobre Ejecutar y escriba `command`. En el caso de NT, escriba `cmd`.
- 2 Cerciórese de que la carpeta `<jdk>\bin` está en el PATH.
- 3 Desplácese al directorio raíz del ejemplo RMI.
- 4 Escriba el siguiente comando en la ventana. Este comando iniciará el servidor en modo depuración y detendrá su ejecución. Este comando puede colocarse en un archivo `.bat` o un script del shell. Si lo hace así, cerciórese de no incluir retornos de carro.

```
java -Xdebug -Xnoagent -Djava.compiler=NONE  
-Djava.rmi.server.codebase=file:\rmi\classes\  
-Djava.security.policy=file:\rmi\SimpleRMI.policy  
-Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=y  
-classpath d:\rmi\classes\ com.borland.samples.rmi.SimpleRMIServer
```

La línea de comandos que el usuario introduce para ejecutar el servidor utiliza los argumentos para RMI y para el depurador. Sigue una descripción de cada uno de los parámetros.

Tabla 21.2 Argumentos de línea de comandos para RMI y el depurador

Parámetro	Descripción
java	El comando que inicia la máquina virtual de Java.
-Xdebug	Ejecuta la MV en modo depuración.
-Xnoagent	No utiliza el agente de depuración.
-Djava.compiler=NONE	No utiliza ningún JIT.
-Djava.rmi.server.codebase=file:\rmi\classes\	Indica la ubicación de los archivos de clase del servidor.
-Djava.security.policy=file:\rmi\SimpleRMI.policy	Indica la ubicación del archivo de normas de seguridad de java.
-Xrunjdp:transport=dt_socket,server=y,address=3999,suspend=y	Opciones del depurador, donde: <ul style="list-style-type: none"> ■ transport: El método de transporte. Debe coincidir con el valor seleccionado en la ficha Depurar de Propiedades para la ejecución en el servidor - consulte el Paso 2 de este tutorial. ■ server: Ejecutar la MV en modo servidor. ■ address: El número del puerto a través del cual el depurador se comunica con el equipo remoto. Debe coincidir con el valor seleccionado en la ficha Depurar de Propiedades para la ejecución en el servidor - consulte el Paso 2 de este tutorial. ■ suspend: Indica si se ha de suspender el programa inmediatamente tras su inicio.
-classpath d:\rmi\classes\	La vía de acceso a las clases.
com.borland.samples.rmi.SimpleRMIServer	El archivo ejecutable del servidor (incluye el nombre del paquete).

En el siguiente paso se empleará el depurador para conectarse con el servidor en ejecución e inspeccionar el método `getDate()` del servidor, donde se fijó el punto de interrupción interprocesal.

Paso 6: Iniciar el proceso de servidor remoto, el cliente en modo depuración e inspeccionar el punto de interrupción interprocesal

En este paso se indica cómo conectarse al proceso de servidor remoto, iniciar el cliente en modo depuración en JBuilder y, a continuación, se inspecciona el punto de interrupción interprocesal. Una vez comenzada la inspección, JBuilder permite desplazarse entre el cliente y el servidor. Se seguirán los siguientes pasos:

- Establecer una conexión con el proceso del servidor remoto en el equipo remoto.
- Iniciar el cliente, en el ordenador cliente, en modo depuración.
- Inspeccionar el punto de interrupción interprocesal en el servidor que se está ejecutando en el equipo remoto.

Para establecer una conexión con el proceso del servidor:



- 1 Pulse sobre la flecha situada a la derecha del botón Depurar programa en la barra de herramientas principal.
- 2 Seleccione la configuración `Windows Run SimpleRMIServer`.

Nota

No es preciso iniciar el registro de RMI en el equipo cliente. Ya se está ejecutando en el equipo remoto.

- 3 El depurador establece una conexión con el proceso remoto en suspensión. (El proceso remoto está en suspensión porque el proceso del servidor se ha iniciado en el equipo remoto con el argumento `suspend=y` en el paso 5.)

El nombre del ordenador remoto y la dirección se muestran en la pestaña de sesión del depurador, situada en la parte inferior de la ventana del Visualizador de aplicaciones de JBuilder.



- 4 Haga clic en el botón Reanudar el programa de la barra de herramientas del depurador.

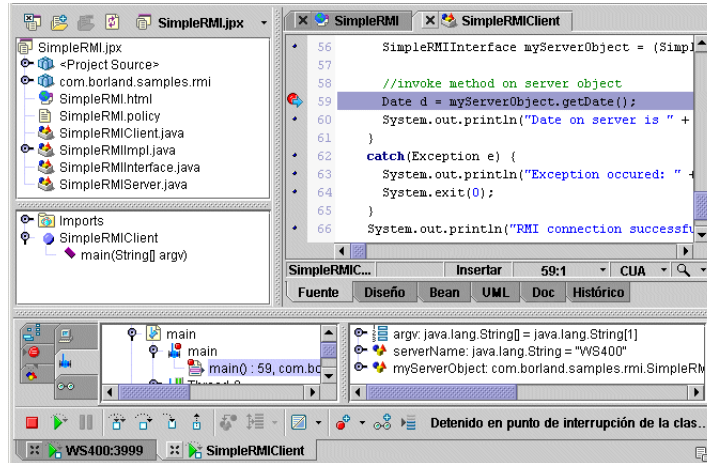
En el ordenador remoto se muestra el mensaje `SimpleRMIImpl ready`.

Para iniciar el cliente en modo depuración, en el equipo cliente (donde se ejecuta JBuilder):



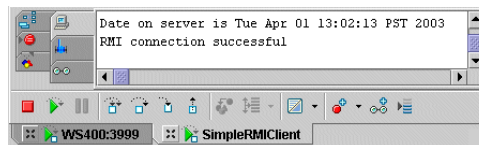
- 1 Pulse sobre la flecha situada a la derecha del botón Depurar programa en la barra de herramientas principal.
- 2 Seleccione la configuración `Windows Run SimpleRMIClient`.

- 3 El depurador inicia una nueva sesión y detiene la ejecución en la llamada al método `getDate()` del servidor. (Se estableció un punto de interrupción en esta línea en el Paso 2.)



Para inspeccionar el punto de interrupción interprocesal:

- 1 Seleccione sobre la pestaña de sesión del depurador para el proceso SimpleRMIClient.
- 2 Pulse el botón Inspeccionar de la barra de herramientas del depurador para inspeccionar el método con punto de interrupción del servidor. Si utiliza Omitir inspección, el depurador no se detendrá.
- 3 Pulse el botón Inspeccionar dos veces más. En el ordenador remoto se muestra el mensaje SimpleRMIIImpl getDate().
- 4 Siga pulsando Inspeccionar en la pestaña para la sesión de depuración SimpleRMIClient hasta que el cliente realice la ejecución por completo. El proceso SimpleRMIClient tendrá el siguiente aspecto en el depurador:




Este será el resultado en el servidor que se ejecuta en el equipo remoto:

```
SimpleRMIIImpl ready
SimpleRMIIImpl.getDate()
```

- 5 Para salir del servidor del ordenador remoto, pulse `Ctrl + C` desde la ventana de comandos. Para cerrar RMIRRegistry, pulse el botón de cierre de la ventana de RMIRRegistry.

Durante la puesta en marcha del servidor o el cliente en modo depuración, en JBuilder, puede presentarse alguno de los siguientes mensajes de error:

Tabla 21.1 Mensajes de error de RMI cliente/servidor

Mensajes de error	Descripción
<code>connection refused</code>	Puede que el registro de RMI no se esté ejecutando en el ordenador remoto. Detenga todos los procesos y ejecute el registro de RMI en el ordenador remoto, escribiendo <code>start rmiregistry</code> en la línea de comandos. (La carpeta <code><jdk>\bin</code> debe estar incluida en PATH.) Reinicie el servidor remoto y comience el proceso de depuración de nuevo.
<code>Java exception:</code> <code>java.rmi.NotBoundException</code> <code>SimpleRMIImpl</code>	No ha iniciado aún el proceso de depuración del servidor. Haga clic en el botón Reanudar el programa  de la barra de herramientas del depurador. Vuelva a iniciar el cliente en modo depuración.

¡Enhorabuena! Ha finalizado este tutorial. Ha ejecutado un servidor RMI en un equipo remoto valiéndose de configuraciones de ejecución preestablecidas. Con lo cual, ha depurado un programa mediante las funciones de depuración remota de JBuilder.



Tutorial: Visualización de código con el visualizador UML

En este tutorial se utilizan funciones de JBuilder Enterprise

En este tutorial se le muestra paso a paso cómo utilizar las características UML de JBuilder para poder navegar por su código y analizarlo. El UML resulta de gran ayuda para examinar código, analizar el desarrollo de una aplicación y para transmitir el diseño del software. JBuilder utiliza diagramas UML para presentar código y buscar clases y paquetes. Los diagramas UML pueden ayudarle a comprender rápidamente la estructura de un código desconocido, reconocer áreas especialmente complejas, e incrementar su productividad resolviendo problemas con mayor rapidez.

Si desea más información sobre las funciones UML de JBuilder, consulte el [Capítulo 11, “Presentación de código con UML”](#). Para conocer las definiciones de los términos de UML, consulte [“Términos de Java y UML” en la página 11-2](#).

En este tutorial, realizará tareas como:

- Visualización de un diagrama de paquete UML
- Visualización de un diagrama de clase UML
- Cómo añadir referencias de biblioteca
- Filtrado de diagramas UML

El tutorial utiliza el proyecto de ejemplo situado en la carpeta `samples/DataExpress/ProviderResolver` de la instalación de JBuilder. Antes de ejecutar este tutorial, asegúrese de haber instalado la carpeta de `ejemplos`.

Los usuarios con acceso de sólo lectura a los ejemplos de JBuilder deben copiar el directorio samples en un directorio para el que tengan permiso de lectura y escritura.

Este ejemplo muestra cómo generar un proveedor y un almacenador de DataExpress personalizados. Utiliza una sencilla aplicación que muestra los datos proporcionados por el bean `ProviderBean` al conjunto de bases de datos `TableDataSet` en una tabla `JdbTable`. Incluye también una barra de herramientas `JdbNavToolBar` cuyo botón Guardar puede pulsarse para guardar los cambios en el archivo de texto, que contiene datos de ejemplo, vía `ResolverBean`. Si desea más información acerca del ejemplo, consulte el archivo de notas del proyecto en el ejemplo: `ProviderResolver.html`. El ejemplo incluye los siguientes archivos:

- `ProviderBean.java`: proporciona datos, a partir de un sencillo archivo de texto no delimitado, a un `TableDataSet`.
- `ResolverBean.java`: reemplaza los datos en el archivo de texto original.
- `TestApp.java`: una sencilla aplicación que muestra los datos proporcionados por el bean `ProviderBean` al conjunto de bases de datos `TableDataSet` en una tabla `JdbTable`. Incluye también una barra de herramientas `JdbNavToolBar` cuyo botón Guardar puede pulsarse para guardar los cambios en el archivo de texto vía `ResolverBean`.
- `TestFrame.java`: la aplicación interfaz de usuario.
- `data.txt`: el archivo de texto que contiene los datos del ejemplo.
- `DataLayout.java`: una interfaz que describe la estructura de `data.txt`.

Consulte

- [“Términos de Java y UML” en la página 11-2](#)
- [“Glosario de los diagramas UML de JBuilder” en la página 11-9](#)

Si desea obtener información útil para ver e imprimir los tutoriales, consulte el apartado Tutoriales, en Sugerencias de JBuilder. El apartado Accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder por parte de personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte [“Convenciones de la documentación” en la página 1-5](#).

Paso 1: Compilación del ejemplo

En este paso, compilará el proyecto. Siempre es mejor compilar antes de seleccionar la pestaña UML, de esta forma el diagrama será exacto y estará actualizado. Cuando selecciona la pestaña UML, JBuilder carga los archivos de clase para determinar sus relaciones, los cuales el visualizador UML utilizará para obtener la información de paquete y clase para los diagramas.

Lo primero que hay que hacer es abrir el ejemplo:

- 1 Seleccione Archivos|Abrir proyecto y busque el ejemplo `ProviderResolver`:
<jbuilder>/samples/DataExpress/ProviderResolver/ProviderResolver.jpx
- 2 Seleccione Proyecto|Ejecutar Make del proyecto para compilar el proyecto.

Importante

Antes de ver el diagrama UML de un proyecto o archivo, siempre debe compilar el proyecto para ver el diagrama completo. JBuilder carga entonces las clases compiladas para generar los diagramas. Seleccione Proyecto|Ejecutar Make del proyecto antes de ir a la pestaña UML.

Paso 2: Visualización de un diagrama de paquete UML

Ahora que el proyecto está compilado, JBuilder puede crear los diagramas UML a partir de los archivos de clase generados. Comience por mirar el diagrama de paquete UML.

- 1 Haga doble clic en el nodo del paquete

`com.borland.samples.dx.providerresolver` en el panel del proyecto para abrirlo en el panel de contenido. Por defecto, se abre con la pestaña Paquete, que muestra el resumen del paquete, activada.

Nota

Si el nodo del paquete no está disponible, configure la opción Activar la localización y compilación de paquetes fuente en la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto|Propiedades de proyecto|General).

- 2 Seleccione la pestaña UML para ver el diagrama del paquete UML. Cuando selecciona la pestaña UML, JBuilder carga las clases para determinar sus relaciones y genera el diagrama UML.



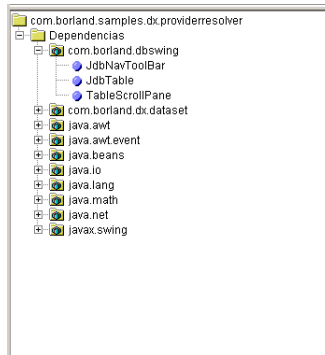
Paso 2: Visualización de un diagrama de paquete UML

El diagrama representa cada paquete como un rectángulo con una pestaña y su nombre completo encima. El paquete actual, `com.borland.samples.dx.providerresolver`, está en el centro, con sus dependencias con otros paquetes a cada lado. Las dependencias en el diagrama UML se representan mediante una línea discontinua que va desde el paquete central hasta el paquete del que depende. Este paquete central, que tiene un fondo verde brillante, lista todas las clases que contiene. Los paquetes exteriores, que tienen un fondo verde más oscuro, sólo listan las clases de las que `com.borland.samples.dx.providerresolver` es dependiente.

- 3 Examine el panel de estructura del lado inferior izquierdo del diagrama UML. Hay dos posibles carpetas que pueden aparecer en el panel de estructura para mostrar diagramas de paquetes: Dependencias y Dependencias inversas. Este paquete sólo tiene dependencias, por lo que sólo hay una carpeta.

Nota Para conocer más acerca de las definiciones de las carpetas del panel de estructura, consulte [“Glosario de los diagramas UML de JBuilder” en la página 11-9](#).

- 4 Abra la carpeta Dependencias para ver los paquetes, clases e interfaces de los que el paquete central es dependiente. Como verá más adelante, puede utilizar el panel de estructura para navegar a otros diagramas UML.



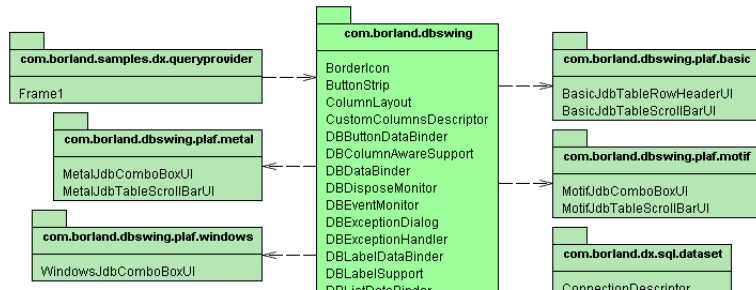
Nota Si desea consultar las definiciones de los iconos del panel de estructura, busque "iconos" en el índice de la ayuda en pantalla (Ayuda | Temas de ayuda).

En los siguientes pasos, navegue hasta un paquete que también tiene dependencias inversas: `com.borland.dbswing`.

- 1 Desplácese hasta el paquete `com.borland.dbswing` que está en la parte superior izquierda del diagrama. Existen dos formas de hacer esto:
 - Haga doble clic sobre el nombre del paquete en el diagrama UML.
 - Abra una carpeta en el panel de estructura, haga clic con el botón derecho sobre el nombre del paquete y seleccione Abrir.

Puede tardar un poco en cargarse. Ahora, el paquete `com.borland.dbswing` es el paquete central del diagrama UML. Hay líneas discontinuas que apuntan en ambas direcciones en este diagrama. El paquete `com.borland.dbswing` tiene *dependencias* y *dependencias inversas*. Si desea ver la definición de estos términos, consulte [“Glosario de los diagramas UML de JBuilder” en la página 11-9](#).

- 2 Mire al paquete del lado superior izquierdo, `com.borland.samples.dx.providerresolver`. Tiene una línea discontinua que apunta hacia el paquete central, y no desde el paquete central. Esto significa que es una dependencia inversa. La clase `TestFrame` del paquete `com.borland.samples.dx.providerresolver` utiliza componentes `dbSwing` para el interfaz de usuario de la aplicación, tales como `JdbNavToolBar` y `JdbTable`. Puesto que `TestFrame` tiene dependencia de `com.borland.dbswing`, entonces `com.borland.dbswing` tiene una dependencia inversa de `com.borland.samples.dx.providerresolver`.

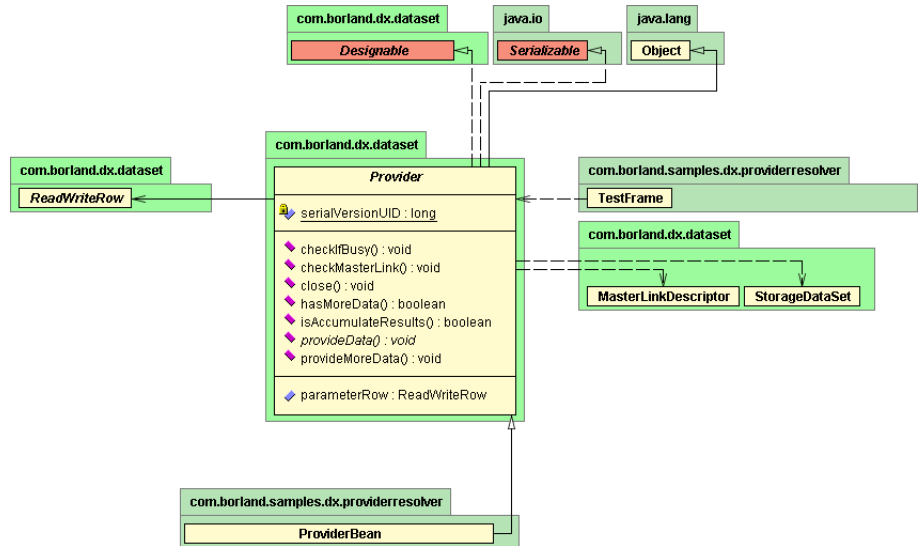


- 3 Mire las carpetas del panel de estructura. Hay dos para este paquete: Dependencias y Dependencias inversas.
- 4 Haga doble clic en la carpeta Dependencias inversas del panel de estructura y amplíe el nodo del paquete `com.borland.samples.dx.providerresolver` para ver que `TestFrame` también aparece listado como una dependencia inversa de `com.borland.dbswing`.
- 5 Haga doble clic sobre `TestFrame` en el diagrama UML o en el panel de estructura para ver los componentes de `dbSwing` que utiliza, así como otros componentes de otros paquetes. Ahora, está viendo un diagrama de clase UML con `TestFrame` como la clase central y los componentes listados directamente bajo el nombre de la clase.
- 6 Seleccione Ver/Maximizar panel de contenido para ampliar el visualizador UML y ocultar los paneles de proyecto y estructura.

Paso 3: Visualización de un diagrama de clase UML

En este paso, navegará hasta otro diagrama de clase, en este caso una clase abstracta llamada `Provider`. En los diagramas UML, las clases abstractas aparecen en cursiva.

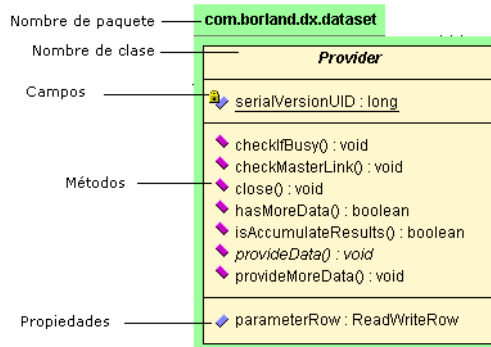
- 1 Desplácese a la derecha y haga doble clic sobre la clase `Provider` ubicada en el paquete `com.borland.dx.dataset`, el segundo a la derecha en el diagrama UML `TestFrame.java`. Ahora, `Provider` aparece como la clase central de un diagrama de clase UML. `Provider` aparece resaltado para indicar que está seleccionado.



Y todos los diagramas del paquete actual, `com.borland.dx.dataset`, se muestran por defecto con fondo verde brillante, mientras que los otros paquetes lo hacen con verde más oscuro.

El diagrama de clase UML muestra la clase en el centro del diagrama. Rodeándola, aparece el paquete, con su nombre en una pestaña en la

parte superior. La propia clase está dividida en varias secciones separadas por líneas horizontales en el orden siguiente:



- Nombre de la clase (arriba): las clases abstractas aparecen en cursiva.
- Campos y propiedades *: los campos estáticos aparecen subrayados.
- Métodos y obtención* y asignación*: los métodos abstractos aparecen en cursiva, los estáticos subrayados.
- Propiedades* (opcional) (abajo).

*Por defecto, las propiedades se muestran en la sección inferior. La opción Mostrar propiedades por separado se configura en la ficha UML del cuadro de diálogo Preferencias (Herramientas|Preferencias). Si esta opción está desactivada, las propiedades se muestran en las secciones adecuadas, con cursiva, los estáticos subrayados. Consulte [“Configuración de preferencias UML” en la página 11-22](#).

Nota

Los iconos indican si un campo, método o propiedad es `privado`, `público`, o `protegido`. Para obtener información sobre las definiciones de los iconos, consulte "El panel de estructura y los iconos UML de JBuilder" en *Introducción a JBuilder*. Puede también cambiar estos iconos por iconos genéricos de UML. Consulte [“Configuración de preferencias UML” en la página 11-22](#).

He aquí otras cosas a observar en el diagrama:

- Las dependencias y las dependencias inversas se encuentran a la derecha, indicadas con líneas discontinuas.
- Las asociaciones están a la izquierda indicadas con líneas sólidas.
- Las clases ampliadas (superclases) y las interfaces implementadas, están arriba. Las relaciones ampliadas se representan con una línea continua con un gran triángulo. Las relaciones implementadas utilizan una línea discontinua con un gran triángulo.
- Las clases ampliables están abajo. Las relaciones ampliadas se representan con una línea continua con un gran triángulo.
- Las interfaces se representan por defecto con rectángulos naranjas con su nombre en cursiva.
- Por defecto, las clases se representan con rectángulos amarillos.

Si desea ver la definición de estos términos, consulte [“Glosario de los diagramas UML de JBuilder” en la página 11-9](#).

- 2 Mire los métodos listados en la tercera sección de la clase central `Provider`. Uno de los métodos, `provideData()`, es un método abstracto. Por lo tanto, su nombre aparece en cursiva.
- 3 Haga doble clic sobre el nombre del método `provideData()` en el diagrama UML para leer los comentarios en el código fuente. Este método abstracto, heredado por la clase `ProviderBean` en la parte baja del diagrama, proporciona los datos para un `DataSet`. Además, este método está también en la clase abstracta `Provider`.
- 4 Seleccione la pestaña UML para volver al diagrama de la clase `Provider`. Las clases que amplían se muestran bajo la clase central. `ProviderBean`, que proporciona los datos que se han de leer en una `TableDataSet`, deriva de (hereda de) la clase abstracta `Provider`.
- 5 Haga clic sobre la clase `ProviderBean` en la parte inferior del diagrama y seleccione Ir a código fuente para examinar algunos de sus métodos. Observe que hereda el método `provideData()` del método abstract `provideData()` de `Provider` del que se deriva.
- 6 Seleccione la pestaña UML para ver el diagrama de clase `ProviderBean`. La herencia se representa en los diagramas UML como una línea continua con un gran triángulo que apunta a la superclase. Debido a que `ProviderBean` hereda de y se deriva de `Provider`, hay una línea continua con un gran triángulo que apunta desde `ProviderBean` hacia `Provider` en la parte superior del diagrama.
- 7 Posicione el puntero sobre el método `provideData()` de la clase `ProviderBean`. Una herramienta de ayuda inmediata muestra el nombre del método con sus parámetros y tipo devuelto. Esta es una forma práctica de aprender más sobre un método sin abandonar el diagrama.

```
provideData(StorageDataSet, boolean): void
```

- 8 Haga doble clic sobre la clase `Provider` en la parte superior del diagrama o haga clic con el botón derecho y seleccione Ir a el diagrama para volver a su diagrama de clase.

Ahora, mire la última sección del diagrama de la clase central `Provider`. Por defecto, las propiedades se listan de forma separada. Puede cambiar la apariencia de las propiedades en la ficha UML del cuadro de diálogo Preferencias (Herramientas|Preferencias). Una propiedad existe cuando un nombre de método precedido de "is", "get", o "set" coincide con un nombre de campo. Por ejemplo, en este diagrama, `parameterRow` es un campo con métodos get y set. Por lo tanto, `parameterRow` es una propiedad.

Importante

Los diagramas de clases UML de los archivos fuente pueden ser más detallados que si sólo hay archivos de clase disponibles, ya que los datos privados y los miembros de clase están ocultos. Por ejemplo, en el código ofuscado, JBuilder excluye los campos privados y los miembros de los archivos de clase.

- 1 Haga clic con el botón derecho en el campo `parameterRow`, en la parte inferior de la clase `Provider`, en el diagrama UML, y seleccione Ir a código fuente.
- 2 Seleccione de nuevo Ver/Maximizar panel de contenido para desactivar el comando de menú Maximizar panel de contenido y volver a la vista normal, que incluye el panel de estructura.
- 3 Examine los métodos del panel de estructura y busque cualquier método "is", "get", o "set". Hay dos métodos con el mismo nombre que el campo `parameterRow`: `getParameterRow()` y `setParameterRow(ReadWriteRow)`.

Paso 4: Añadir referencias de bibliotecas

En alguna ocasión, puede querer incluir referencias desde bibliotecas del proyecto para ver un diagrama completo de todas las relaciones. Por defecto, los diagramas UML de JBuilder no muestran las referencias de bibliotecas del proyecto. Habitualmente, las bibliotecas proporcionan servicios a las aplicaciones que se construyen a partir de ellas, pero no saben nada acerca de sus usuarios. Para mostrar estas relaciones, necesita incluir referencias de las bibliotecas.

Cuando seleccione la opción Incluir referencias de archivos de clase de las bibliotecas del proyecto en la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto/Propiedades de proyecto/General), se incluirán también referencias de las clases de la biblioteca del proyecto en el diagrama UML. Después de seleccionar esta opción, JBuilder actualiza automáticamente el diagrama para incluir las referencias. Para obtener más información sobre esta opción, consulte ["Inclusión de referencias de bibliotecas de proyecto" en la página 11-21](#).

Precaución

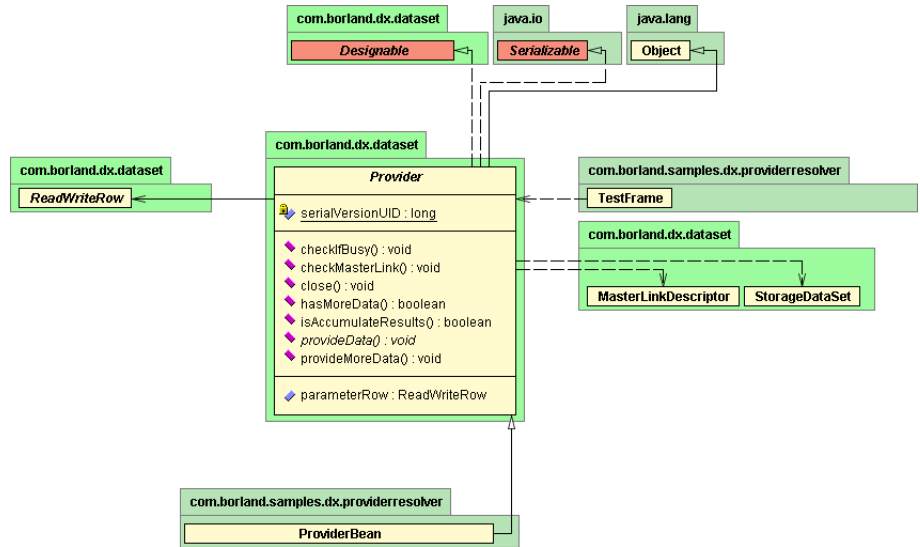
Si las bibliotecas son grandes, el diagrama UML puede tardar cierto tiempo en cargarse. JBuilder debe cargar primero todas las clases y la información de las dependencias para determinar las relaciones.

Antes de seleccionar la opción Incluir referencias de los archivos de clase de las bibliotecas del proyecto, revise el diagrama de la clase `Provider`. Después, cambie las propiedades del proyecto y observe cómo cambia el diagrama `Provider`.

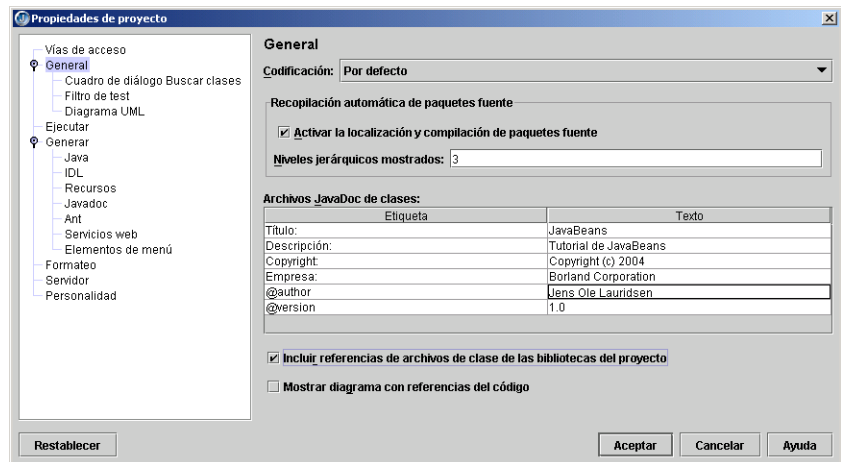
- 1 Seleccione la pestaña UML para volver al diagrama de la clase `Provider`. Hay un paquete en el lado izquierdo que contiene la clase `ReadWriteRow` y un paquete en la parte inferior del diagrama que contiene la clase

Paso 4: Añadir referencias de bibliotecas

`ProviderBean`. Cuando añada las referencias de las bibliotecas, se añadirán más paquetes y clases al diagrama en estas áreas.

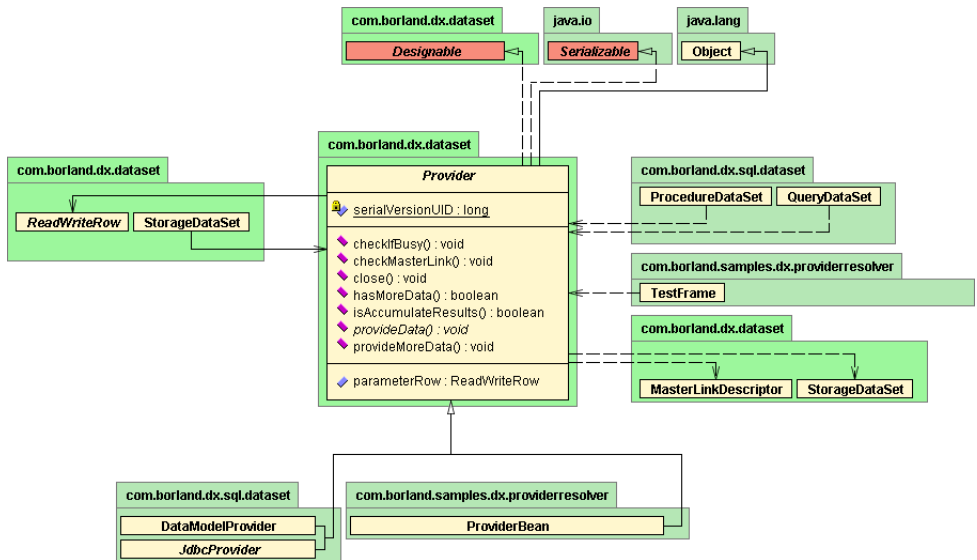


- 2 Seleccione Proyecto|Propiedades de proyecto|General para abrir la ficha General del cuadro de diálogo Propiedades de proyecto.
- 3 Seleccione la opción Incluir referencias de los archivos de clase de las bibliotecas del proyecto en la parte inferior de la ficha General.



- 4 Haga clic en Aceptar para cerrar el cuadro de diálogo. El diagrama UML se actualiza y ahora muestra referencias adicionales de las bibliotecas del proyecto.
- 5 Revise el diagrama UML y observe los paquetes y clases adicionales añadidos a la izquierda y en la parte inferior del mismo. En el lado izquierdo, se ha añadido `StorageDataSet` al paquete `com.borland.dx.dataset`.

En la parte inferior se ha añadido otro paquete `com.borland.dx.sql.dataset` con dos clases. Estas clases en la parte inferior del diagrama se derivan de `Provider`. Las clases del paquete `com.borland.dx.sql.dataset` no se utilizan directamente por las clases del proyecto, pero, puesto que derivan de `Provider`, guardan relación con el mismo.



- 6 Pulse y arrastre el diagrama sin soltar el botón del ratón para trasladarlo al centro. Mire los dos paquetes `com.borland.dx.dataset` a izquierda y derecha. Observe que `StorageDataSet` aparece en dos sitios. `Provider` tiene dos relaciones con `StorageDataSet`: una asociación inversa y una dependencia. Aparece también en las carpetas apropiadas del panel de estructura.

Ahora, dedique un momento a examinar las asociaciones generadas en un diagrama de clase UML. Las asociaciones aparecen en el lado izquierdo de un diagrama de clase. `Provider` tiene dos asociaciones: una *asociación* y una *asociación inversa*. Estas asociaciones aparecen en las carpetas adecuadas del panel de estructura. Si desea ver la definición de estos términos, consulte ["Glosario de los diagramas UML de JBuilder"](#) en la página 11-9.

- 1 Abra la carpeta Asociaciones del panel de estructura del diagrama de la clase `Provider` y amplíe el paquete. Aquí puede ver que `ReadWriteRow` es una asociación.
- 2 Seleccione `ReadWriteRow` en el panel de estructura para resaltarlo en el diagrama. Las asociaciones en el diagrama se representan con una línea continua que apunta desde la clase central a la asociación. `Provider` tiene una asociación con `ReadWriteRow`, puesto que tiene la siguiente referencia a `ReadWriteRow`:

```
private ReadWriteRow parameterRow;
```

- 3 Abra la carpeta Asociaciones inversas en el panel de estructura y amplíe el paquete. Aquí puede ver que `StorageDataSet` es una asociación inversa.
- 4 Seleccione `StorageDataSet` en el panel de estructura para resaltarlo en el diagrama. Una asociación inversa se representa con una línea continua que apunta desde la asociación a la clase central. `StorageDataSet` tiene una referencia a `Provider`:

```
private Provider provider;
```
- 5 Haga clic con el botón derecho en `StorageDataSet` en el diagrama y seleccione Ir a código fuente para navegar directamente al código fuente.
- 6 Realice una búsqueda en el código fuente de `provider` (Buscar|Buscar) y seleccione Buscar todos. Examine los resultados de la búsqueda en el panel de mensajes y observe que hay muchas referencias a la clase `Provider`, así como a los métodos `getProvider()` y `setProvider()`.
- 7 Haga clic con el botón derecho sobre la pestaña Buscar en el panel de mensajes y seleccione Eliminar la pestaña "Buscar" para cerrarlo.

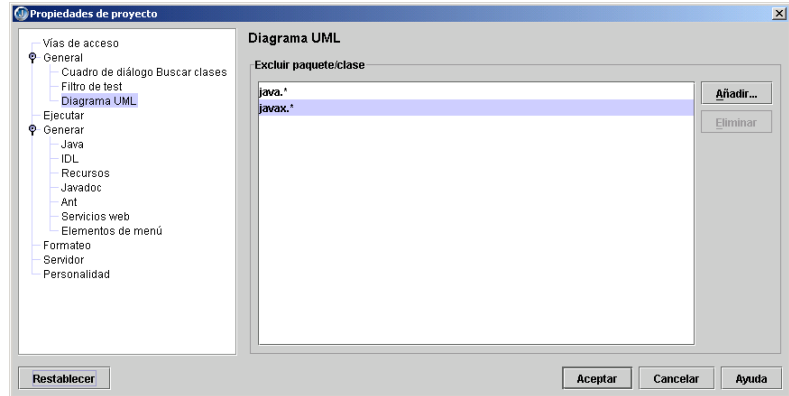
Paso 5: Filtrado de diagramas UML

En algunos casos, puede que desee eliminar paquetes y clases de sus diagramas UML con el fin de simplificarlos. Para ello, ha de abrir el cuadro de diálogo Propiedades de proyecto. Si desea más información, consulte [“Definición de las propiedades del proyecto” en la página 11-20](#).

El filtrado de paquetes y clases se configura en la ficha Filtros del diagrama UML del cuadro de diálogo Propiedades de proyecto. El filtrado determina qué clases y paquetes se excluyen de los diagramas UML del proyecto. Una vez que haya configurado los filtros, puede activar y desactivar el filtrado desde el menú contextual del visualizador UML.

- 1 Seleccione la pestaña del archivo del paquete `com.borland.samples.dx.providerresolver` en la parte superior del panel de contenido. Verá que ahora hay muchos paquetes `java` y `javax` en el diagrama. Estos paquetes aparecen también en el panel de estructura de la carpeta Dependencias.
- 2 Seleccione `Proyecto|Propiedades de proyecto|General|Diagrama UML` para abrir el Diagrama UML del cuadro de diálogo Propiedades de proyecto.
- 3 Excluya los paquetes `java` y `javax` del diagrama del paquete `com.borland.samples.dx.providerresolver` de la siguiente manera:
 - a Seleccione el botón Añadir para abrir el cuadro de diálogo Seleccionar paquete/clase.
 - b Pulse sobre la pestaña Examinar.
 - c Seleccione el paquete `java` y utilice `Ctrl+Clic` para seleccionar también el paquete `javax`.

- d Pulse Aceptar para cerrar el cuadro de diálogo Seleccione clase/ Paquete. Observe que estos dos paquetes aparecen ahora en la lista Excluir paquete/clase.



- e Haga clic en Aceptar para cerrar el cuadro de diálogo Filtro de diagrama UML.
- 4 Vuelva a revisar el diagrama y notará que los paquetes `java` y `javax` se han eliminado del diagrama. Sólo permanecen en el diagrama los paquetes `borland`.
 - 5 Si examina el panel de estructura, comprobará que el filtro no elimina los paquetes `java` y `javax` packages del panel de estructura, aunque los paquetes se muestran en un color más suave para indicar que no se encuentran en el diagrama.
 - 6 Haga clic con el botón derecho en el visualizador UML y observe que Activar filtrado de clases está seleccionado en el menú contextual.
 - 7 Desactive el filtrado quitando la marca de Activar filtrado de clases. Los paquetes reaparecen en el diagrama.

Importante

Si configura el filtrado en el cuadro de diálogo Propiedades de proyecto, todos los diagramas del proyecto se filtran. Deshabilitar el filtrado en un diagrama no lo desactiva para todos. Si se desplaza a otro diagrama del proyecto, el filtrado sigue activado. Cuando se cierra el archivo o el paquete, la configuración recupera los valores definidos para el proyecto.

Enhorabuena. Ha finalizado este tutorial de UML. Hay otras muchas características disponibles en el visualizador UML de JBuilder, tales como:

- Perfeccionamiento de código.
- Guardar e imprimir diagramas UML.
- Visualización de Javadoc.
- Personalización de diagramas UML.

Consulte

- [Capítulo 13, “Perfeccionamiento de código”](#)
- [“Creación de imágenes de diagramas UML” en la página 11-23](#)
- [“Impresión de diagramas UML” en la página 11-23](#)
- [“Visualización de Javadoc” en la página 15-30](#)
- [“Configuración de preferencias UML” en la página 11-22](#)



Tutorial: Creación y ejecución de tests y de conjuntos de tests

La comprobación de módulos es una característica de todas las ediciones de JBuilder.

Este tutorial le enseñará cómo crear un test y un conjunto de test para analizar el código creado. Este tutorial utiliza el ejemplo `ProviderResolver` de `<jbuilder>/samples/DataExpress` como aplicación en test. Antes de ejecutar este tutorial, asegúrese de haber instalado la carpeta de `ejemplos`.

Nota Capítulo 22, “Tutorial: Visualización de código con el visualizador UML” también se utiliza el ejemplo `ProviderResolver`. Si ha previsto ejecutar ambos tutoriales, es recomendable que ejecute primero este último, o que haga una copia del ejemplo `ProviderResolver` antes de ejecutar éste, porque las modificaciones hechas en este tutorial pueden cambiar algunos de los diagramas descritos en el tutorial de UML.

Este tutorial asume que usted está familiarizado con Java, JUnit y con el IDE (entorno integrado de desarrollo) de JBuilder. Para obtener más información acerca de capturas, consulte *Procedimientos iniciales con Java*. Si desea más información sobre JUnit, visite del sitio web de JUnit, <http://www.junit.org>. Para obtener más información sobre el IDE de JBuilder IDE, consulte “El entorno de JBuilder” en *Creación de aplicaciones con JBuilder*.

Nota La opción elegida en su configuración de ejecución actual en Tipos de generación debe ser Ejecutar Make o Generar de nuevo para que todos los pasos de este tutorial funcionen correctamente. Ejecutar Make es el valor por defecto. Para obtener más información acerca de configuraciones de ejecución, consulte “Definición de las configuraciones de ejecución” en la [página 7-8](#).

Si desea obtener información útil para ver e imprimir los tutoriales, consulte el apartado Tutoriales, en Sugerencias de JBuilder. El apartado Accesibilidad en las Sugerencias de JBuilder contiene sugerencias sobre la utilización de las funciones de JBuilder para mejorar la facilidad de uso de JBuilder por parte de personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte [“Convenciones de la documentación” en la página 1-5](#).

Paso 1: Apertura de proyectos

En este paso, se abre el ejemplo ProviderResolver. Para cumplir con los propósitos de este tutorial, la aplicación ProviderResolver será la aplicación bajo prueba.

- 1 Seleccione ArchivosAbrir proyecto para presentar el cuadro de diálogo homónimo.
- 2 Haga clic en el icono de la carpeta Ejemplos.
- 3 Abra el nodo DataExpress y, a continuación, abra el nodo ProviderResolver en el árbol.
- 4 Seleccione `ProviderResolver.jpx` y pulse Aceptar.

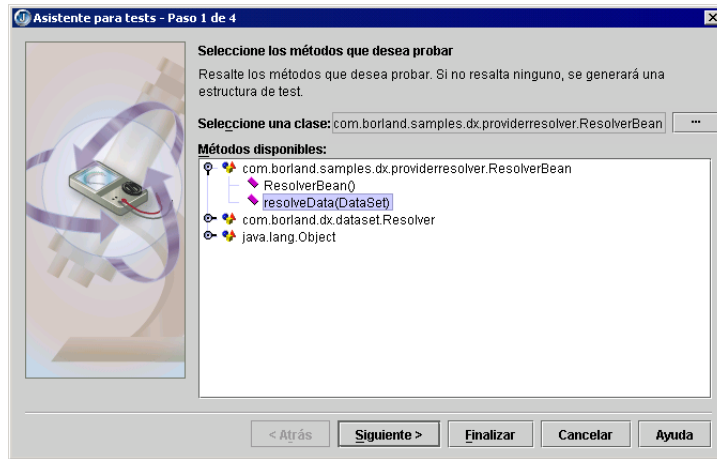
El ejemplo ProviderResolver está abierto.

Paso 2: Creación de montajes para tests

Este paso crea el montaje de un test utilizando el Asistente para tests. La clase del montaje para tests contendrá un método de test para uno de los métodos de la clase `ResolverBean`. Posteriormente añadirá un segundo método de test. Cuando escriba test en el mundo real, querrá realizar test más exhaustivos, pero para los propósitos de este tutorial, sólo implementará dos métodos de test.

- 1 Seleccione ProyectoGenerar de nuevo proyecto "ProviderResolver.jpx". Esto hace que los métodos de las clases del proyecto estén disponibles para el Asistente para tests.
- 2 Haga doble clic en `ResolverBean.java` en el panel del proyecto para abrirlo en el editor.
- 3 Para mostrar la galería de objetos, seleccione ArchivosNuevo.
- 4 Seleccione Test en la ficha Test de la galería de objetos y pulse Aceptar. Se muestra el Asistente para tests con `ResolverBean` seleccionada como la clase a comprobar.

- 5 Seleccione el método `resolveData(DataSet)` de `ResolverBean`. Éste es el aspecto del Asistente para tests:



- 6 Pulse el botón Finalizar. `TestResolverBean.java`, el nuevo test, se abre en el editor y se coloca en el paquete `com.borland.samples.dx.providerresolver`.

Paso 3: Implementación de un método de test que lanza una excepción esperada

A veces es útil escribir un test para verificar que se lanza una excepción esperada. Su código del test debería ser lo suficientemente exacto como para determinar si la excepción lanzada es la misma que la esperada. El test debería fallar si se lanza otra excepción.

En este paso y en el siguiente, rellenará el montaje para el test escribiendo código de test. Este paso implementa el método `testResolveData()` en `ProbarResolverBean`. Para implementar el método:

- 1 Sustituya el cuerpo del método `testResolveData()` por el siguiente código:

```
resolverBean = new ResolverBean();
DataSet dataSetView1 = new StorageDataSet();

try {
    resolverBean.resolveData(dataSetView1);
    fail("fallido: resolveData() did not throw an exception");
}
catch(DataSetException e){
    System.out.println("TestResolverBean.testResolveData(): correcto ");
}
```



- 2 Pulse el botón Guardar todo de la barra de herramientas, o bien seleccione `ArchivolGuardar todo`.

- 3 Abra el nodo `com.borland.samples.dx.providerresolver` en el panel del proyecto.
- 4 Para ejecutar el test, haga doble clic en `TestResolverBean.java` en el panel del proyecto y seleccione en el menú contextual Ejecutar test utilizando "TestResolverBean". Cuando termina la ejecución de la comprobación, la barra de progreso del ejecutor de tests aparece en verde. Junto a los nombres de las clases de test y los tests, en el ejecutor de tests, se muestra una marca de verificación que indica que la comprobación se ha realizado correctamente.

El código del test lanza una excepción `DataSetException`. Esto se debe a que el conjunto de datos no está abierto. Cuando la excepción `DataSetException` esperada es lanzada, ésta se captura y el test pasa porque no hay comunicación de fallo o error. Si se lanza otra clase de excepción, el test falla con un seguimiento de la pila.

Añadió también otra línea de código del test a continuación de la línea del bloque `try` que se espera que lance una excepción. Si no se lanza ninguna excepción, esta línea se ejecuta. La llamada a `fail()` en esta línea de código hace que el test falle y la cadena que se pasa explica el fallo. Por supuesto, esta nueva línea de código sólo puede ejecutarse si no se lanza una excepción.

Visualización de la salida de fallo del test

Para mostrar el aspecto que tiene un test fallido en el ejecutor de tests:

- 1 Anteponga signos de comentario (`//`) a la siguiente línea de código en el bloque `try`:

```
resolverBean.resolveData(dataSetView1);
```



- 2 Pulse el botón Guardar todo de la barra de herramientas, o bien seleccione ArchívolGuardar todo.

- 3 Para ejecutar el test, haga doble clic en `TestResolverBean.java` en el panel del proyecto y seleccione en el menú contextual Ejecutar test utilizando "TestResolverBean". La barra de avance se vuelve roja para indicar que ha habido, al menos, un fallo en el test. Se muestra la pestaña Fallos del test. Se lista un fallo para el método `testResolveData()`. Esto se indica mediante un icono X rojo.



- 4 Haga clic en el nodo `testResolveData()` de la ficha Fallos del test. Se muestran los detalles de fallo del test.

Observe cómo la cadena que se pasa al método `fail()` ha sido escogida cuidadosamente para proporcionar información útil en caso de fallo. Este es un buen objetivo a tener en cuenta cuando escriba sus tests.

Corrección del test para que pase

Ha precedido de signos de comentario una línea de código necesaria para que el test pase. El propósito de esto era ver el aspecto que tiene un fallo en el ejecutor de tests. Para hacer que el test pase de nuevo:

- 1 Quite los signos de comentario de la línea de código que llama a

```
resolverBean.resolveData(dataSetView1).
```



- 2 Pulse el botón Guardar todo de la barra de herramientas, o bien seleccione ArchívolGuardar todo.

Si el test se vuelve a ejecutar, el resultado será correcto. Hágalo ahora si lo desea.

Paso 4: Creación de un segundo método de test

En este paso escribirá un método para probar el valor de una de las constantes definidas en la interfaz `DataLayout`, que es implementada por `ResolverBean`. Esta test verifica que el valor de la constante se asigna correctamente. Para ello:

- 1 Añada el siguiente método a la clase `TestResolverBean`:

```
public void testConstant() {
    resolverBean = new ResolverBean();
    assertEquals(6, resolverBean.COLUMN_COUNT);
}
```



- 2 Pulse el botón Guardar todo de la barra de herramientas o bien seleccione ArchívolGuardar todo.

El código que acaba de añadir comprueba el valor de la constante `COLUMN_COUNT` de la interfaz `DataLayout` para asegurarse de que concuerda con el valor esperado. Para ahorrar tiempo en este tutorial, sólo comprobamos esta constante, pero esto debe darle una idea de algunos posibles tests que podría escribir para verificar valores esperados. Si ejecuta el test ahora, debería pasar, puesto que el valor se ha configurado correctamente.

Paso 5: Creación de un conjunto de tests

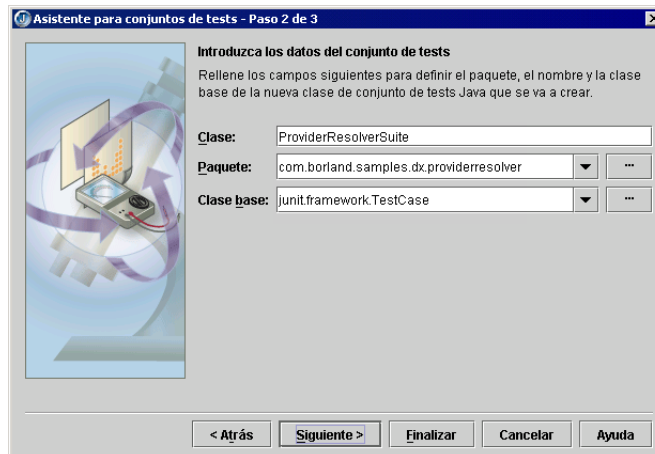
Un conjunto de tests es una colección de ellas que deberían ejecutarse en grupo. En este paso, creará un conjunto de tests utilizando el Asistente para conjuntos de tests. Este conjunto de tests llamará a `TestResolverBean`. Normalmente, un conjunto de tests llama a más de uno, pero, para ahorrar tiempo en este tutorial, sólo se ha creado un test. Si dispusiera de más de un test, el proceso para crear el conjunto que llama a varios test es el mismo.

- 1 Seleccione ArchívolNuevo.

- 2 Seleccione Conjunto de tests en la ficha Test de la galería de objetos y haga clic en Aceptar.
- 3 Seleccione `TestResolverBean.java` como test a incluir en el conjunto. Use el botón Añadir si es necesario. Si tiene otros test, puede también añadirlos en este momento. Éste es el aspecto del Asistente para tests:



- 4 Pulse Siguiente.
- 5 Escriba `ProviderResolverSuite` en Nombre de clase. Ahora el Asistente para conjuntos de tests tiene este aspecto:



- 6 Pulse el botón Finalizar. Se añade un archivo `ProviderResolverSuite.java` a su proyecto.
- 7 Haga doble clic sobre `ProviderResolverSuite.java` en el paquete `com.borland.samples.dx.providerresolver` en el panel del proyecto para abrirlo. Examine la siguiente línea de código del método `suite()`:

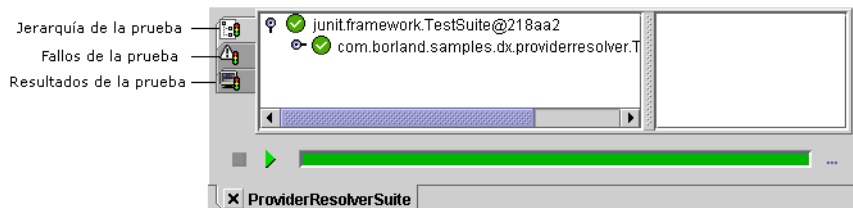
```
suite.addTestSuite(
    com.borland.samples.dx.providerresolver.TestResolverBean.class);
```

Si posteriormente desea añadir otros tests a este conjunto, deberá escribir una línea de código similar a ésta por cada uno de ellos, sustituyendo el nombre de clase del nuevo test por `TestResolverBean`.

Paso 6: Ejecución de tests

En este paso ejecutará el conjunto de tests que acaba de crear. El proceso para ejecutar un conjunto de tests es el mismo que para ejecutar un test excepto por el hecho de que cuando se ejecuta un conjunto, automáticamente se ejecutan todos los tests que forman parte de él. Para ejecutar su conjunto de tests:

- 1 Haga doble clic en `ProviderResolverSuite.java` en el panel del proyecto y seleccione Ejecutar test utilizando "TestResolverBean" en el menú contextual. El test se ejecuta.
- 2 Examine la salida. JBTTestRunner tiene el siguiente aspecto:



La pestaña superior situada a la izquierda de la ficha JBTTestRunner en la Vista de mensajes es la vista Jerarquía del test. Observe que el árbol que se muestra en esta vista indica que todas los tests han pasado. Todos los iconos de los tests tienen una marca de comprobación verde. Bajo el nodo para `ProviderResolverSuite`, verá un subnodo para su test. Si tenía otros test, habrá un subnodo por cada uno de ellos. Expanda el nodo del test para ver los test individualmente. Haga clic en un test o nodo individual para ver sus resultados.

- 3 Pulse sobre la pestaña Fallos del test. Actualmente no hay ninguna salida en esta vista. Si hubiera habido fallos, los listaría y mostraría la salida generada por sus aserciones fallidas.

- 4 Pulse sobre la pestaña Resultados del test. Esta pestaña lista los resultados de los tests. El resultado del test que escribió en este tutorial es:

```
TestResolverBean.testResolveData(): correcto
```

Encima de todo esto, la pestaña Resultados del test muestra también el comando que se utilizó para ejecutar los tests.

Puede también depurar los tests haciendo clic con el botón derecho sobre ellos en el panel del proyecto y seleccionando Depurar test en el menú contextual. El depurador de tests funciona igual que el depurador normal, por lo que no se explica en este tutorial. La única diferencia radica en que al depurar un test, aparecen las pestañas Jerarquía del test y Fallos del test de JUnitRunner además de la interfaz del usuario normal del depurador. Si desea más información sobre el depurador, consulte el [Capítulo 8, “Depuración de programas en Java”](#).

¡Enhorabuena! Ha completado el tutorial sobre creación y ejecución de tests y conjuntos de tests. Para obtener más información sobre el test de módulos, consulte el [Capítulo 14, “Test de módulos”](#).



Tutorial: Utilización de montajes para tests

Los montajes para tests predefinidos son características de JBuilder Enterprise.

Este tutorial le muestra cómo utilizar el Asistente para montajes para JDBC y el Asistente para montajes para tests que le permitirán crear montajes para sus tests. Los montajes son código compartido que puede ser utilizado por múltiples clases de comparaciones para realizar tareas rutinarias. Este tutorial le muestra también cómo utilizar el Montaje de comparación y el montaje JDBC juntos en un test.

Este tutorial asume que usted está familiarizado con Java, JUnit, JDataStore y con el IDE (entorno integrado de desarrollo) de JBuilder. Para obtener más información acerca de capturas, consulte *Procedimientos iniciales con Java*. Si desea más información sobre JUnit, visite del sitio web de JUnit, <http://www.junit.org>. Consulte la *Guía del Programador de JDataStore*, para obtener más información. Para obtener más información sobre el IDE de JBuilder IDE, consulte "El entorno de JBuilder" en *Creación de aplicaciones con JBuilder*.

Nota

La opción elegida en su configuración de ejecución actual en Tipos de generación debe ser Ejecutar Make o Generar de nuevo para que todos los pasos de este tutorial funcionen correctamente. Ejecutar Make es el valor por defecto. Para obtener más información acerca de configuraciones de ejecución, consulte "[Definición de las configuraciones de ejecución](#)" en la [página 7-8](#).

Si desea obtener información útil para ver e imprimir los tutoriales, consulte el apartado Tutoriales, en Sugerecias de JBuilder. El apartado Accesibilidad en las Sugerecias de JBuilder contiene sugerencias sobre la utilización de las

funciones de JBuilder para mejorar la facilidad de uso de JBuilder por parte de personas con discapacidades.

Para obtener información sobre las convenciones utilizadas en este tutorial y en otra documentación de JBuilder, consulte [“Convenciones de la documentación” en la página 1-5](#).

Paso 1: Creación de proyectos

- 1 Seleccione Archivo|Nuevo proyecto para abrir el Asistente para proyectos.
- 2 En el campo Nombre, introduzca un nombre de Proyecto, como `fixturestutorial`.
- 3 Pulse Finalizar para cerrar el asistente para Proyectos y crear el proyecto. No se necesita hacer ningún cambio en los valores por defecto en los Pasos 2 y 3 del asistente.

Se crea un nuevo proyecto.

Paso 2: Creación de un módulo de datos

En este paso, creará una clase `DataModule` que se utilizará para el test. En la mayoría de los casos, cuando se escriben tests reales ya se tiene un código que se desea comprobar. En este tutorial se va a crear un código de ejemplo para comprobarlo a continuación.

Para crear el módulo de datos:

- 1 Seleccione Archivo|Nuevo.
- 2 Seleccione Módulo de datos en la ficha General de la galería de objetos. Pulse Aceptar. Se abre el Asistente para módulos de datos.
- 3 Acepte los valores por defecto para Paquete y Nombre de clase, desactive la opción Llamar al Modelador de datos y pulse Aceptar. Se crea un módulo de datos.
- 4 Añada la siguiente sentencia de importación a la parte superior del archivo `DataModule1.java`, justo después de la línea `package fixturestutorial;`

```
import com.borland.dx.sql.dataset.*;
```
- 5 Añada la siguiente línea de código justo después de la línea `private static DataModule1 myDM;`

```
Database databasel = new Database();
```
- 6 Añada la siguiente línea de código al método `jbInit()`. `<unidad>` y `<jbuilder>` son la unidad real y el directorio donde se encuentra instalado JBuilder:

```
databasel.setConnection(new ConnectionDescriptor  
("jdbc:borland:dslocal:<drive>:\\<jbuilder>\\samples\\JDataStore\\
```

```
datastores\\employee.jds", "user", "", false,
"com.borland.datastore.jdbc.DataStoreDriver"));
```

7 Añada el siguiente método al módulo de datos:

```
public Database getDatabase1() {
    return database1;
}
```

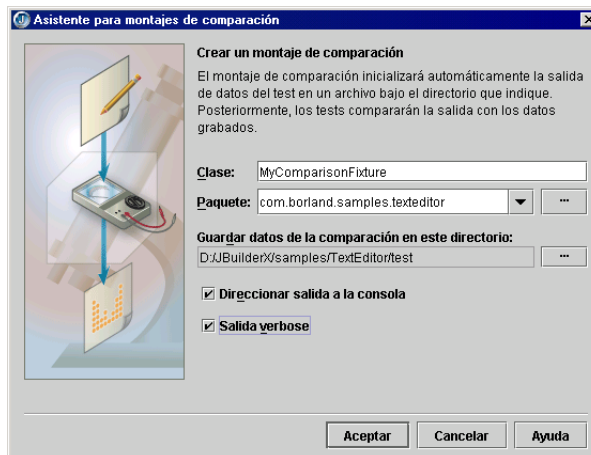
8 Seleccione Proyecto|Generar de nuevo el proyecto "fixturetutorial.jpj".

Ha finalizado el módulo de datos. En los pasos siguientes se crearán montajes para tests y un test, que se utilizarán para comprobar este módulo de datos.

Paso 3: Creación de un montaje de comparación

El Asistente para montajes para tests genera un montaje que es de utilidad para grabar los resultados de los tests y compararlos con resultados de tests previos. Un Montaje de comparación amplía `com.borland.jbuilder.unittest.TestRecorder`. Para crear un Montaje de comparación:

- 1 Seleccione Archivo|Nuevo|Test.
- 2 Seleccione Montaje de comparación y pulse Aceptar. Se abre el Asistente para montajes de comparación.
- 3 Escriba `MyComparisonFixture` como nombre de clase.
- 4 Acepte el valor por defecto en el campo Paquete.
- 5 Acepte el valor por defecto para la ubicación del directorio de datos de comparación.
- 6 Marque Direcccionar salida a la consola y Salida verbose. Éste es el aspecto del Asistente para montajes de comparación:



- 7 Pulse Aceptar. Se crea una clase de montaje de comparación llamada `MyComparisonFixture`. Expanda el nodo `fixturestutorial` del panel del proyecto para poder verlo.

Nota

Si el nodo del paquete no está disponible, configure la opción Activar la localización y compilación de paquetes fuente en la ficha General del cuadro de diálogo Propiedades de proyecto (Proyecto|Propiedades de proyecto).

- 8 Haga doble clic sobre `MyComparisonFixture.java` en el panel de proyecto para abrirlo en el editor (está en el paquete `fixturestutorial`). Anote la siguiente línea de código:

```
super.setMode(UPDATE);
```

Esta línea de código configura el modo de salida del montaje de comparación. He aquí los posibles valores de las constantes pasadas a `setMode()`:

- `UPDATE` - El montaje de comparación compara la nueva salida con un archivo de salida existente, o lo crea si no existe y graba la salida en él.
- `COMPARE` - El montaje de comparación siempre compara la nueva salida con la ya existente.
- `RECORD` - El montaje de comparación graba todas las salidas, sobrescribiendo cualquiera ya existente en el archivo de salida.
- `OFF` - El montaje de comparación está desactivado.

Sugerencia

Si un archivo de salida contiene datos incorrectos, defina el modo de salida en `RECORD` después de resolver el problema. Cuando haya registrado la salida deseada, vuelva a definir el modo en `UPDATE`.

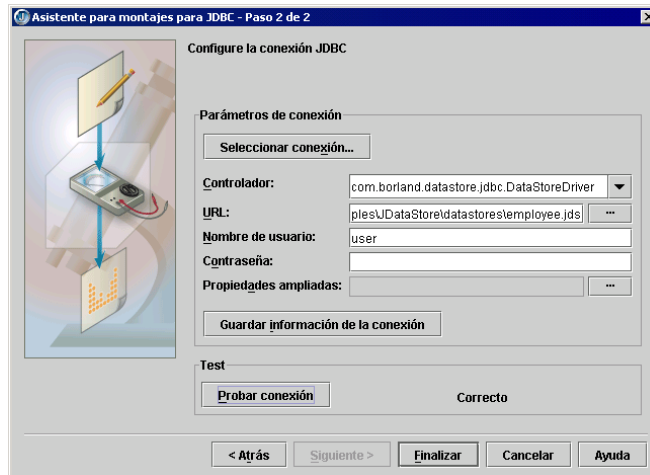
Paso 4: Creación de un montaje JDBC

El Asistente para montajes JDBC genera un montaje que se utiliza para administrar conexiones con fuentes de datos JDBC.

Para crear un montaje JDBC:

- 1 Seleccione Archivo|Nuevo|Test.
- 2 Seleccione Montaje JDBC y pulse Aceptar. Se abre el Asistente para montajes JDBC.
- 3 Acepte los valores por defecto para Paquete, Clase, Clase base y pulse en Siguiente.
- 4 Seleccione el siguiente controlador:
`com.borland.datastore.jdbc.DataStoreDriver`
- 5 Escriba o navegue hasta la siguiente URL:
`jdbc:borland:dslocal:<unidad>\<jbuilder>\samples\JDataStore\datastores\employee.jds` (<unidad> y <jbuilder> se sustituyen por la ubicación real de JBuilder).

- 6 Escriba `user` en Nombre de usuario.
- 7 Haga clic en el botón Probar conexión. Debería haber un mensaje indicando que la acción se ha realizado con éxito a la derecha del botón Probar conexión. Éste es el aspecto del Asistente para montajes para JDBC:



Si la conexión falla, puede deberse a que aún no haya introducido la información de licencia correcta de JDataStore en el Administrador de licencias de JDataStore. El Administrador de licencias de JDataStore esta disponible en el menú Archivo del Explorador de JDataStore.

- 8 Pulse el botón Finalizar. Se crea una clase de montaje para JDBC llamada `JdbcFixture1`.
- 9 Haga doble clic en `JdbcFixture1.java`, en el panel del proyecto, para abrirlo en el editor. Observe los métodos `setUp()` y `tearDown()` en este montaje. En el paso siguiente, utilizará estos métodos para ejecutar scripts SQL para gestionar datos que podrían utilizarse en sus tests.

Paso 5: Modificación del montaje JDBC para ejecutar scripts SQL

En este paso modificará los métodos `setUp()` y `tearDown()` del método del montaje JDBC para hacer que ejecuten scripts SQL que generen automáticamente datos de test antes de que se ejecuten los tests y borren los datos cuando hayan finalizado. Para ello:

- 1 Asegúrese de que `JdbcFixture1.java` está abierto en el editor.
- 2 Añada las variables `String` que aparecen en **negrita** al montaje:

```
public class JdbcFixture1 extends
com.borland.jbuilder.unittest.JdbcFixture {
```

```
String createSQL =
"create table TESTTABLE (i int, j int);" +
"insert into TESTTABLE values(1, 2);" +
"insert into TESTTABLE values(2, 3);" +
"insert into TESTTABLE values(3, 4);" +
"insert into TESTTABLE values(4, 5);";

String deleteSQL =
"drop table TESTTABLE;";
```

Estas variables `String` contienen sentencias SQL que se utilizarán para administrar los datos de test.

3 Añada el código que aparece en **negrita** al método `setUp()`:

```
public void setUp() {
    super.setUp();

    Connection con = getConnection();
    if(con != null)
        runSqlBuffer(new StringBuffer(createSQL), true);
```

Este código consigue una conexión con `JDataStore` y ejecuta el script SQL para crear la tabla de tests.

4 Añada el código que aparece en **negrita** al método `tearDown()`:

```
Connection con = getConnection();
if(con != null)
    runSqlBuffer(new StringBuffer(deleteSQL), true);
super.tearDown();
```

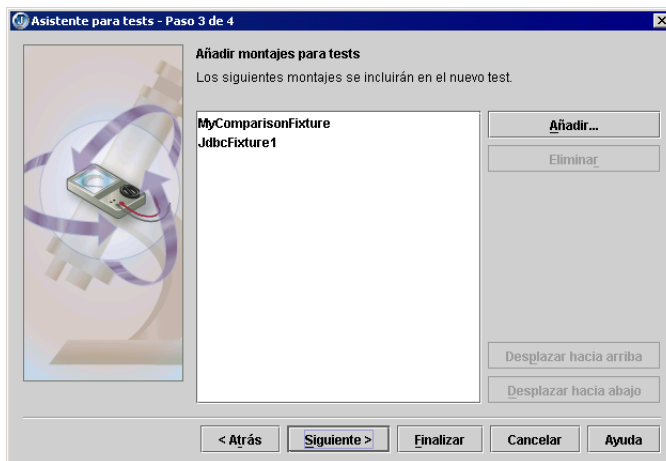
Este código consigue una conexión con `JDataStore` y ejecuta el script SQL para eliminar la tabla de tests.

Paso 6: Creación de un test utilizando montajes para tests

En este caso utilizará el Asistente para tests para incluir el montaje de comparación y el montaje JDBC en un tests.

- 1 Seleccione Proyecto|Generar de nuevo el proyecto "fixturestutorial.jpj". Esto hace que los métodos de las clases del proyecto estén disponibles para el Asistente para tests.
- 2 Haga doble clic en `DataModule1.java` para abrirlo en el editor.
- 3 Seleccione Archivo|Nuevo|Test.
- 4 Seleccione Test y pulse Aceptar. Se abre el Asistente para tests.
- 5 Acepte `fixturestutorial.DataModule1` como la clase a probar. No seleccione ningún método.
- 6 Pulse Siguiente.

- 7 Acepte los detalles de clase por defecto en el Paso 2 del asistente y pulse en Siguiente.
- 8 Utilice el botón Añadir para añadir `MyComparisonFixture` y `JdbcFixture1` a la lista de montajes para tests seleccionados, si no lo están ya. Éste es el aspecto del Asistente para tests:



- 9 Pulse el botón Finalizar. Se añade un nuevo test al proyecto llamado `TestDataModule1`. Abra el nodo `fixturestutorial` en el panel de proyecto para verlo.

Paso 7: Implementación del test

En este paso se escribe el código que llama a los dos montajes para tests, con el fin de utilizarlos en un test.

- 1 Haga doble clic en `TestDataModule1.java` en el panel de proyecto para abrirlo en el editor. El test crea una instancia para los montajes y llama a sus métodos `setUp()` y `tearDown()`.
- 2 Añada la siguiente línea de código a las sentencias `import` en la parte superior de `TestDataModule1.java`:

```
import java.awt.*;
```

- 3 Añada el siguiente método al cuerpo del archivo `TestDataModule1.java`:

```
public void testQuery() throws Exception{
    DataModule1 dm = new DataModule1();
    Connection con = dm.getDatabase1().getJdbcConnection();
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM TESTTABLE");
    jdbcFixture1.dumpResultSet(rs, myComparisonFixture);
    dm.getDatabase1().closeConnection();
}
```

Este método hace lo siguiente:

- Crea una instancia `DataModule1`.
- Establece una conexión utilizando el método `getJdbcConnection()` del objeto `Database` de `DataExpress` utilizado en `DataModule1`.
- Llama al método `Connection.createStatement()` con el fin de prepararse para la ejecución de una consulta en SQL.
- Ejecuta la consulta, almacenando el resultado en un objeto `ResultSet`.
- Utiliza el método `dumpResultSet()` del montaje JDBC para volcar el conjunto de resultados al montaje de comparación. El método `dumpResultSet()` pasa un `ResultSet` y un `Writer` como parámetros. Un Montaje para comparación puede utilizarse como el `Writer` ya que lo amplía `Writer`.
- Llama al método `closeConnection()` del objeto `Database` con el fin de garantizar que la conexión con la fuente de datos está cerrada.

Paso 8: Adición de una biblioteca necesaria

Para que sea posible ejecutar el test es necesario añadir al proyecto la biblioteca `JDataStore`. Para ello:

- 1 Seleccione `Proyecto1Propiedades` de proyecto.
- 2 Seleccione la pestaña `Bibliotecas necesarias` de la ficha `Vías de acceso` del cuadro de diálogo `Propiedades de proyecto`.
- 3 Pulse el botón `Añadir`.
- 4 Seleccione en la lista la biblioteca `JDataStore` y pulse `Aceptar`.
- 5 Pulse `Aceptar` para cerrar el cuadro de diálogo `Propiedades de proyecto`.

Paso 9: Ejecución del test

En este paso ejecutará el test.

- 1 Haga clic con el botón derecho en `TestDataModule1.java` en el panel del proyecto y seleccione `Ejecutar test` utilizando `"TestDataModule1"` en el menú. El test se ejecuta. Cuando se ejecuta el test sucede lo siguiente:
 - El ejecutor del test crea una instancia `TestDataModule1`.
 - Se llama al `TestDataModule1.setUp()` que, como respuesta, llama a los métodos `setUp()` de los dos montajes en el orden adecuado.
 - Se llama al método `testQuery()`. La salida del montaje de comparación se graba en un archivo de datos en el mismo directorio fuente en que se encuentra `TestDataModule1.java`, el subdirectorio `test/fixturestutorial` del directorio del proyecto.

- Se llama al método `tearDown()` que, como respuesta, llama a los métodos `tearDown()` de los dos montajes en el orden adecuado.

¡Enhorabuena! Ha finalizado este tutorial. Para obtener más información sobre el test de módulos, consulte el [Capítulo 14, “Test de módulos”](#).



Las herramientas de línea de comandos

JBuilder cuenta con las siguientes herramientas de línea de comandos:

- Interfaz de la línea de comandos de JBuilder
- Make de Borland para Java (**bmj**)
- El compilador de Borland para Java (**bcj**)

bmj y bcj son características de JBuilder Developer y Enterprise

Herramientas de desarrollo del SDK

El JDK incluye las siguientes herramientas de línea de comandos:

- **javac** - compila el lenguaje de programación Java.
- **java** - inicia las aplicaciones Java.
- **jar** - gestiona los archivos recopilatorios Java (JAR).
- **javadoc** - extrae los comentarios del código y genera documentación HTML a partir de ellos.
- **appletviewer** - permite ejecutar applets sin utilizar un visualizador de Web.
- **native2ascii** - convierte archivos de caracteres de codificación nativa en archivos con secuencias de escape de Unicode.

Consulte

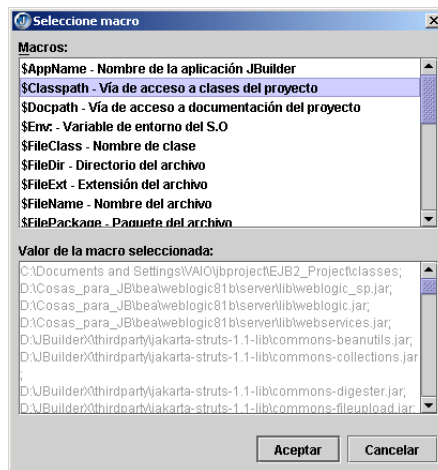
- "Herramientas básicas" en "Herramientas y utilidades del SDK de Java 2"

Utilización de macros de la línea de comandos

Las macros de la línea de comandos ayudan a construir vías de acceso y parámetros mediante términos predefinidos. Por ejemplo, en lugar de escribir la vía de acceso a clases completa del proyecto o configurar un archivo batch para hacerlo en cada proyecto, se puede escribir `$Classpath`, donde se invoca a la vía de acceso a clases del proyecto, y JBuilder inserta automáticamente la vía de acceso completa en la línea de comandos. Y, como estas macros las mantiene JBuilder, no es necesario escribir un archivo batch para cada proyecto individual, se puede simplemente usar la macro para cada proyecto y dejar que JBuilder haga el trabajo.

En el cuadro de diálogo Configuración de herramientas se pueden insertar macros de la lista. En cualquier lugar, escriba manualmente las macros donde se necesiten. Para buscar la lista completa de macros de la línea de comandos disponibles:

- 1 Seleccione Herramientas\Configurar herramientas.
- 2 Pulse el botón Añadir o Modificar del cuadro de diálogo Configuración de herramientas.
- 3 Pulse el botón Insertar macro en el cuadro de diálogo Añadir herramienta o Modificar herramienta.
- 4 Aparece el cuadro de diálogo Seleccione macro:



- 5 Seleccione la macro que le interese para ver su valor en el área de texto que se encuentra más abajo.
- 6 Anote las macros que le interesen.
- 7 Haga clic en Cancelar en cada uno de los cuadros de diálogo, a menos que desee aprovechar la oportunidad de configurar aquí las herramientas de la línea de comandos.

Determinadas macros requieren la introducción de valores. Estas macros tienen dos puntos, como `$Env:`, la variable de entorno del sistema operativo, y `$Property:`, la variable de la propiedad del sistema. Escriba valores después de los dos puntos. Los valores válidos para la variable de la propiedad del sistema se recogen en el cuadro Acerca de de JBuilder de la ficha Información, incluidos los valores pertenecientes a herramientas de terceros. Seleccione Ayuda/Acerca de la Información para verlos.

Las macros de la línea de comandos se pueden introducir manualmente en vías de acceso y campos de parámetros en los cuadros de diálogo Añadir configuración de ejecución o Modificar configuración de ejecución, así como en el Asistente para proyectos y en el Asistente para aplicaciones.

Definición de la vía de acceso a clases para herramientas de línea de comandos

La vía de acceso a clases indica a las herramientas de Java dónde se encuentran las clases que no forman parte de la plataforma Java. Es posible definir la vía en las clases que contienen la opción **-classpath**, y también se puede definir la variable de entorno `CLASSPATH` descrita a continuación. La opción **-classpath** redefine provisionalmente la variable de entorno `CLASSPATH` para la sesión de línea de comandos actual. Es recomendable utilizar **-classpath**, ya que se puede definir para una aplicación sin que influya sobre las otras.

Los directorios de classpath se muestran separados por dos puntos en UNIX y por punto y coma en Windows. Debe incluir siempre las clases de sistema al final de la vía de acceso. La vía de acceso a clases se utiliza también para buscar los archivos fuente si no se ha especificado ninguna vía de acceso a ellos.

Para obtener más información sobre las vías de acceso a clases, consulte "Configuración de la vía de acceso a clases".

Si desea obtener más información sobre el IDE de JBuilder y las vías de acceso a clases, consulte ["Cómo construye JBuilder las vías de acceso" en la página 4-9](#) y ["Localización de los archivos" en la página 4-12](#).

Opción -classpath

La opción **-classpath** permite definir provisionalmente la vía de acceso a clases.

UNIX La opción **-classpath** tiene la siguiente forma:

```
% jdkTool -classpath path1:path2
```

Windows La opción **-classpath** tiene la siguiente forma:

```
C:>jdkTool -classpath path1;path2
```

Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos

La opción **-classpath** de línea de comandos sólo modifica temporalmente la vía de acceso a clases y no interfiere con otras aplicaciones. No obstante, se puede configurar de forma permanente la variable de entorno `CLASSPATH`.

Para obtener más información sobre la opción **-classpath** y la variable de entorno `CLASSPATH`, consulte "Configuración de la vía de acceso a clases".

Unix: variable de entorno CLASSPATH

Para visualizar la variable `CLASSPATH`,

- 1 Abra una ventana shell de línea de comandos.
- 2 Presente el valor actual de la variable de entorno `CLASSPATH` mediante el siguiente formato de línea de comando:

- en el shell csh:

```
env
```

- en el shell sh:

```
CLASSPATH
```

Para definir la variable de entorno `CLASSPATH`:

- 1 Abra una ventana shell de línea de comandos.
- 2 Defina la variable de entorno `CLASSPATH` con el siguiente formato de línea de comandos:

- en el shell csh:

```
setenv CLASSPATH path1:path2
```

- en el shell sh:

```
CLASSPATH = path1:path2  
export CLASSPATH
```

Para eliminar la variable `CLASSPATH`,

- 1 Abra una ventana shell de línea de comandos.
- 2 Borre la variable de entorno de `CLASSPATH` con el siguiente formato de línea de comando:

- en el shell csh:

```
unsetenv CLASSPATH
```

- en el shell sh:

```
unset CLASSPATH
```

Windows: variable de entorno CLASSPATH

Para visualizar la variable `CLASSPATH` actual, utilice el comando `set`.

```
C:> set
```

Para definir la variable de entorno `CLASSPATH`,

- 1 Abra una ventana DOS.
- 2 Modifique la variable de entorno `CLASSPATH` con el comando `set`.

```
set CLASSPATH=path1;path2 ...
```

Las vías de acceso deben comenzar con la letra de la unidad, por ejemplo `C:\.`

Si desea borrar la vía de acceso, puede eliminar los valores de la variable `CLASSPATH` de la siguiente manera:

```
C:> set CLASSPATH=
```

Este comando elimina los valores de la variable `CLASSPATH` sólo para la sesión DOS actual. Para que los valores de `CLASSPATH` sean los correctos, deberá modificar o borrar los valores de la configuración de inicio.

Si la variable `CLASSPATH` queda establecida al iniciar el sistema, se deberá buscar en sitios diferentes según el sistema operativo.

- Windows NT: Para que se abra el cuadro Propiedades del sistema, seleccione Inicio|Configuración|Panel de control|Sistema. Seleccione la pestaña Entorno y modifique la variable `CLASSPATH` en la sección Variables de usuario.
- Windows 2000: Para que se abra el cuadro Propiedades del sistema, seleccione Inicio|Configuración|Panel de control|Sistema. Haga clic sobre la pestaña Avanzado y pulse el botón Variables de entorno para modificar la variable `CLASSPATH` en la sección Variables de usuario. Si no está conectado en el equipo local como administrador, sólo puede cambiar variables de usuario.
- Windows XP: Para que se abra el cuadro Propiedades del sistema, seleccione Inicio|Configuración|Panel de control|Sistema. Pulse la pestaña Avanzadas y, a continuación, el botón Variables de entorno. Si no está conectado en el equipo local como administrador, sólo puede cambiar variables de usuario.

Interfaz de la línea de comandos de JBuilder

JBuilder cuenta con una interfaz de línea de comandos que incluye argumentos como:

- Creación de proyectos
- Visualización de información de la configuración
- Visualización del administrador de licencias
- Desactivación de la pantalla de presentación
- Activación del modo de depuración con comentarios para autores de OpenTools

Nota Estos argumentos varían según la edición de JBuilder.

JBuilder ejecuta su propio programa de inicio, que es un ejecutable. El ejecutable puede pasar argumentos a JBuilder.

Acceso a una lista de opciones

Para acceder a la lista de argumentos disponibles en su versión de JBuilder, abra una ventana de línea de comandos, sitúese en el directorio `bin` de JBuilder y escriba `jbuilder -help`.

```
/<jbuilder>/bin> jbuilder -help
```

JBuilder presenta una lista de argumentos disponibles:

Argumentos de línea de comandos disponibles:

- build: Generar proyectos JBuilder **(una función de las ediciones Developer y Enterprise)**
- help: Mostrar la ayuda en la línea de comandos
- info: Mostrar información de configuración
- license: Mostrar el administrador de licencias
- nosplash: Desactivar pantalla de inicio
- verbose: Mostrar el diagnóstico de carga de OpenTools

Si desea obtener más información sobre cada argumento, escriba `jbuilder -help <nombre_del_argumento>`, tal y como aparece en este ejemplo.

```
/<jbuilder>/bin> jbuilder -help info
```

JBuilder devuelve esta información:

```
-info
Muestra la información sobre la configuración del sistema durante el
arranque.
```

También es posible presentar una lista de argumentos tal y como se muestra en este ejemplo:

```
/<jbuilder>/bin> jbuilder -help info nosplash verbose
```

Sintaxis

```
jbuilder [argumentos]
```

Opciones

Es una función de JBuilder Developer y Enterprise.

-build<args>

Genera uno o más proyectos JBuilder suministrados como argumentos. Las configuraciones se toman directamente de los archivos de proyecto correspondientes. Se pueden definir destinos, que se ejecutan en el orden por el que aparecen. Si no se indica ninguno, Ejecutar Make es el destino por defecto.

Entre los argumentos **--build**, los proyectos se distinguen de los destinos por la extensión .jpx o .jpr. Se supone que todos los argumentos que terminen por .jpx o .jpr son proyectos. Todos los argumentos que no tengan la extensión .jpx o .jpr son nombres de tipos.

Nota Si un proyecto falla al completarse, los proyectos restantes no se generan.

El comando tiene esta forma:

```
jbuilder -build <project1.jpx> [ [<target1> <target2> ...]
                               [<project2.jpx> [<target3> ...] ] ... ]
```

Por ejemplo:

```
jbuilder -build myproject.jpx rebuild
```

```
jbuilder -build myproject.jpx clean make myotherproject.jpx
```

Nota Si el proyecto no se encuentra en el directorio actual, incluya la vía de acceso completa en el archivo de proyecto. Por ejemplo, si los proyectos están en el directorio /usuario/nombreusuario/ la vía de acceso completa sería:

```
jbuilder -build /usuario/nombreusuario/myproject.jpx clean make
/usuario/nombreusuario/myotherproject.jpx
```

El proceso de generación desde la línea de comandos presenta los errores y las advertencias de texto que pueden redirigirse a otro proceso o archivo. Por ejemplo, en las plataformas Windows, puede redirigirse a un archivo de texto del siguiente modo:

```
jbuilder -build myproject.jpx > myproject.txt
```

El proceso de generación desde la línea de comandos devuelve un código de resultado que indica que si hay fallos o si es correcto. Si la generación es correcta, se devuelve un valor cero. Si no lo es, se devuelve un valor distinto de cero. El valor real declarado depende del error.

En este ejemplo, el archivo .BAT de Windows genera el proyecto y refleja un mensaje de error o de éxito:

```
rem This has to be in a .BAT file:
jbuilder -build myproject.jpx
```

```
if not errorlevel 0 echo ERROR
if errorlevel 0 echo SUCCESS
rem End of .BAT file
```

Además, el proceso de generación desde la línea de comandos permite automatizar los procesos con otras herramientas de la línea de comandos. Por ejemplo, puede ejecutar tareas más complicadas, como generar otro proyecto si la generación es correcta y, a continuación, copiar los archivos o cancelar el archivo por lotes si falla la generación.

Consulte la documentación de su sistema operativo si desea obtener más información sobre los scripts o programas por lotes que admite su sistema.

-help<args>

Enumera los argumentos de línea de comando de JBuilder disponibles.

Cuando se utiliza sin argumentos, **-help** muestra una lista de argumentos de línea de comandos reconocidos con breves descripciones. Cuando se llama a help con uno o varios argumentos se obtiene una descripción más detallada de los comandos especificados y sus argumentos.

```
-help <argumento1> <argumento2> <argumento3>
```

Los argumentos han de escribirse sin guión inicial.

-info

Muestra la información sobre la configuración del sistema durante el arranque.

-license

Muestra el administrador de licencias en lugar de iniciar JBuilder.

-nosplash

Desactiva la pantalla de presentación que se muestra cuando se lanza JBuilder.

-verbose <args>

Muestra el diagnóstico de carga de OpenTools.

Make de Borland para Java (bmj)

Es una función de
JBuilder Developer y
Enterprise.

Sintaxis

```
bmj [options] rootClasses
```

Descripción

La utilidad Borland Make para Java (**bmj**), que es el compilador por defecto del IDE y un compilador de línea de comandos, usa el compilador estándar **javac** para compilar el código fuente Java a bytecodes Java y comprueba las dependencias con el objeto de determinar qué archivos necesitan realmente una nueva compilación. **bmj** genera el programa Java en forma de archivos `.class` que contienen bytecodes que, a su vez, constituyen el código máquina de la máquina virtual Java. La compilación de un archivo fuente produce un archivo `.class` para cada declaración de clase o de interfaz. Cuando se ejecuta el programa de Java resultante en una plataforma concreta, como por ejemplo Windows NT, el intérprete de Java de dicha plataforma ejecuta los bytecodes de los archivos `.class`.

bmj busca archivos de dependencia en la vía de acceso a clases. Si se especifica un conjunto de archivos fuente, es posible que no se recompilen todos ellos. Por ejemplo, podría determinarse que los archivos de clase están actualizados si se han guardado pero no se han editado desde la última compilación. Es posible forzar la recompilación por medio de la opción **-rebuild**.

Para verificar un conjunto (o gráfico) de clases interdependientes, basta con llamar a **bmj** en la clase raíz (o en varias clases raíz, si no está una debajo de otra). Es posible especificar las clases raíz utilizando nombres de clase, de paquete, nombres de fuentes que declaren clases o una combinación de éstos.

Es posible que necesite definir la variable de entorno `CLASSPATH` de la línea de comandos, de forma que las clases necesarias estén disponibles.

Para ver la sintaxis y la lista de opciones de la línea de comandos, introduzca el comando `bmj` sin argumentos que se encuentra en el directorio `<jbuilder>/bin`. La mayoría de estas opciones también se pueden especificar en el IDE de JBuilder en Propiedades de proyecto (Proyecto/Propiedades de proyecto/Generar). Haga clic en el botón Ayuda para obtener más información.

Nota Si desea utilizar la versión anterior de **bmj** de JBuilder 8, utilice el comando `oldbmj`.

Consulte

- [“Comprobación inteligente de dependencias” en la página 5-2](#)
- [“Compilación desde la línea de comandos” en la página 5-11](#)
- [“Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos” en la página A-4](#)

- "Configuración de la vía de acceso a las clases" en la documentación de las herramientas de Java
- [“El compilador de Borland para Java \(bcj\)” en la página A-16](#)

Opciones

Nota Los directorios de las vías se muestran separados por dos puntos en UNIX y por punto y coma en Windows. Los siguientes ejemplos representan la plataforma UNIX.

-classpath viadeAcceso

La vía de acceso que se utiliza para encontrar archivos de clases y de dependencia. Redefine el valor por defecto o la variable de entorno `CLASSPATH`. Debe incluir siempre la vía de salida al principio de la vía. La vía de salida es el directorio raíz de la jerarquía de archivos de clase. La vía de acceso a clases se utiliza también para buscar los archivos fuente si no se ha especificado ninguna vía de acceso a ellos.

Por ejemplo:

```
bmj -classpath jbproject/testing/classes/test3:  
jbproject/project1/classes tester.java
```

-d directorio

El directorio raíz de la jerarquía de archivos de clase (destino). También se le denomina vía de salida.

Por ejemplo, la siguiente sentencia:

```
bmj -d jbproject/project1/classes tester.java
```

hace que los archivos de las clases definidas en el archivo fuente `tester.java` se guarden en el directorio `jbproject/project1/classes/test/test3`, suponiendo que `tester.java` contenga la siguiente sentencia de paquete: `package test.test3;`

El archivo de dependencias actualizado, `test.test3.dep2`, se guarda en `jbproject/project1/classes/package cache`.

Los archivos se leen desde la vía de acceso a clases y se escriben en el directorio de destino. El directorio de destino puede ser parte de la vía de acceso a clases. El destino por defecto de los archivos de clase coincide con la estructura de paquetes de los archivos origen y se inicia desde el directorio raíz de archivos fuente. El destino por defecto de los archivos de dependencia coincide con la estructura de paquetes y se inicia en el directorio actual.

-deprecation

Presenta todas las clases, métodos, propiedades, variables y sucesos desaconsejados empleados en la API.

Si se produce una advertencia, durante la compilación, de que se han utilizado API desaconsejadas, puede ver estas últimas activando esta opción.

-nombre de codificación

Puede especificar un nombre de codificación de archivo (o un nombre de ficha de códigos) para controlar cómo interpreta el compilador los caracteres que no pertenecen al conjunto de caracteres ASCII. Por omisión se utiliza el convertidor de codificación nativa por defecto de la plataforma. Si desea más información, consulte [“Elección de una codificación nativa para el compilador” en la página 17-13](#).

Por ejemplo, la siguiente sentencia:

```
bmj -encoding EUC_JP tester.java
```

compila `tester.java` y todos los archivos `.java` importados directamente que no tengan un archivo `.class`. Se considera que todos los archivos de código fuente están codificados en el conjunto de caracteres EUC_JP, que es el que se utiliza normalmente para los entornos UNIX japoneses. Puede especificar cualquier codificación que admita la plataforma Java 2. Puede obtener una lista de las codificaciones válidas en Para más información acerca de la internacionalización y las codificaciones válidas, consulte "native2ascii" en "SDK Development Tools".

-exclude classname

Excluye todas las llamadas a los métodos `static void` en el archivo `.class` seleccionado de una compilación. Esto también excluye la evaluación de los parámetros pasados a esos métodos.

Por ejemplo, la exclusión de la clase A elimina todas las llamadas a los métodos `static void` de A desde las OTRAS clases.

-g

Genera toda la información de depuración en el archivo de clase, incluidas las variables locales. Por defecto, sólo se generan números de línea y la información del archivo fuente.

-g:none

No genera ninguna información de depuración.

-g:{lista de palabras clave}

Genera algunos tipos de información de depuración. La lista de palabras clave es una lista de palabras separadas por comas, como, por ejemplo: `bmj -g:lines,vars`

Entre las palabras clave se incluyen:

- **lines**: información de depuración del número de línea
- **vars**: información de depuración de la variable local
- **source**: información de depuración del archivo fuente

-nocompile

Verifica si las clases están actualizadas, pero no compila ninguna. Es útil para verificar rápidamente si una clase o un paquete están actualizados. Se detiene en el primer archivo que deba volver a compilarse y comunica que "La clase <clase> necesita volver a compilarse debido a <motivo>".

-nowarn

Compila sin mostrar advertencias.

-obfuscate

El camuflaje hace que los programas sean menos vulnerables a las manipulaciones. Cuando se descompila el código confuso o camuflado, el código fuente generado representa los símbolos privados mediante nombres distintos.

-quiet

Compila sin mostrar ningún mensaje.

-rebuild

Compila las clases raíz especificadas y sus archivos importados, independientemente de si se han modificado.

Consulte

- [“El comando Generar de nuevo” en la página 6-5](#)

-versión fuente

Habilita la compilación de código fuente que contiene órdenes.

- En la versión 1.4, esta opción de línea de comandos activa la palabra clave `assert` y la función de órdenes de JDK 1.4.
- En la versión 1.3 el compilador no acepta las órdenes.
- Si no se utiliza la opción `-source`, el comportamiento por defecto del compilador es el de la versión 1.3.

Consulte

- [“Programming With Assertions” en la Guía de características de JDK](#)

-sourcepath vía de acceso

Vía de acceso en la que se buscarán los archivos fuente. Si no se ha especificado ninguna vía de acceso a archivos fuente, se utilizará la vía de acceso a clases para buscar estos archivos.

Al igual que la vía de acceso a clases, la vía de acceso a archivos fuente debe situarse en el directorio raíz del árbol de paquetes y no directamente en el directorio de los archivos fuente.

Por ejemplo, para crear `tester.java`, que contiene la sentencia de paquete `test.test3` y que se encuentra en `jbproject/project1/src/test/test3`, debe definir la vía de acceso de archivos fuente como `jbproject/project1/src/` y no como `jbproject/project1/src/test/test3`.

A continuación, escriba lo siguiente:

```
bmj -sourcepath jbproject/project1/src
jbproject/project1/src/test/test3/tester.java
-d jbproject/project1/classes
```

-sync

Borra del directorio de salida, los archivos de clase para los que no se disponía de archivos fuente antes de la compilación. Se debe especificar el directorio de vía de salida por medio de la opción **-d**.

Esta opción puede resultar útil para evitar que el compilador encuentre en el directorio de archivos generados un archivo de clase que no se pueda compilar desde el código fuente. (Probablemente ha eliminado el archivo fuente o ha renombrado una de las clases declaradas en un archivo fuente.)

-versión tipo

Restringe los archivos de clase de modo que funcionen con una versión de MV determinada.

bmj admite:

- 1.1 – Genera archivos de clase para ejecutar en 1.1 y MV en Java 2 SDK. Si selecciona ésta como la MV destino, cualquier MV puede cargar sus archivos de clase.
- 1.2 - Genera archivos de clase que pueden ejecutarse **únicamente** en las MV del SDK de Java 2, versiones 1.2 y posteriores, pero **no** en las MV 1.1. Ésta es la opción por defecto.
- 1.3 - Genera archivos de clase que pueden ejecutarse **únicamente** en las MV del SDK de Java 2, versiones 1.3 y posteriores, pero **no** en las MV 1.1 ni 1.2.
- 1.4 - Genera archivos de clase que se ejecutan **únicamente** en máquinas virtuales de la versión 1.4 y posteriores del SDK de Java 2, pero **no** se ejecutan en 1.1, 1.2 ni 1.3.

-verbose

Esta opción proporciona más información sobre la compilación, como por ejemplo, los archivos de clase que se cargan y su ubicación en la classpath. Se presenta la siguiente información:

- La vía de acceso a clases, de archivos fuente y el directorio de archivos generados que se utilizan.
- Qué archivos fuente se compilan.
- Qué archivos de clases se cargan.

- Qué clases se generan.
- Qué archivos de dependencia se generan.

Opciones de compilación cruzada

bmj acepta la compilación cruzada, que consiste en compilar las clases con un inicio y clases de extensión de una plataforma Java distinta.

Utilice **-bootclasspath** y **-extdirs** para realizar una compilación cruzada.

-bootclasspath bootclasspath

Compilación cruzada con las clases de arranque especificadas. Se pueden utilizar directorios, recopilatorios JAR y ZIP.

-extdirs directorios

Compilación cruzada con los directorios de extensión determinada. Se buscan archivos de clase para todos los archivos JAR de los directorios elegidos.

Especificadores para las clases raíz

Las clases raíz se especifican de la siguiente forma:

```
{[-s] {source.java} | -p {package} | -c {class}}
```

-s nombredearchivofuente

Indica que las clases raíz especificadas son las definidas en los archivos fuente indicados. Ésta es la interpretación por defecto.

Por ejemplo, la siguiente sentencia:

```
bmj -sourcepath jbproject/project1/src  
-s jbproject/project1/src/tester.java
```

es igual que

```
bmj -sourcepath jbproject/project1/src  
jbproject/project1/src/tester.java
```

Si enumera algunos paquetes con la opción **-p** antes de enumerar los archivos fuente, debe especificar la opción **-s**. Si enumera los archivos fuente antes que los paquetes y las clases, se da por supuesto el uso de la opción **-s** por lo que no es necesario especificarla.

-p nombre del paquete

Nombre de los paquetes que se deben compilar.

Por ejemplo, la siguiente sentencia:

```
bmj -sourcepath jbproject/project1/src -p test.test3
```

ejecuta Make sobre todas las clases del paquete `test.test3` y todas las clases importadas.

-c nombredeclase

Los nombres de las clases que se van a procesar con Make.

Por ejemplo, la siguiente sentencia:

```
bmj -sourcepath jbpproject/project1/src
-c test.test3.testster
```

ejecuta Make sobre la clase `testster` del paquete `test.test3` y todas las clases importadas.

Otro ejemplo es la sentencia siguiente:

```
bmj -sourcepath jbpproject/project1/src tester.java -p package1 package2
-s jbpproject/project1/src/*.java
```

Crea el archivo fuente `tester.java`, los paquetes `package1` y `package2` y todos los archivos java del directorio `jbpproject/project1/src`.

El primer nombre de archivo fuente (`tester.java`) va antes que la opción **-p** (paquete), por lo que no hace falta especificar de modo explícito la opción **-s**, ya que **-s** está implícita. Sin embargo, si después de especificar la opción **-p** desea especificar otro nombre de archivo fuente, tiene que utilizar explícitamente la opción **-s**.

Opciones de MV

-Joption

Pasa opciones al método de inicio de `java` al que llama **bmj**. Por ejemplo, **-J-Xms48m** configura la memoria de inicio en 48 megabytes. Es una convención común para **-J** pasar opciones a la MV subyacente que ejecuta aplicaciones escritas en Java.

Observe que `CLASSPATH`, **-classpath**, **-bootclasspath** y **-extdirs** no especifican las clases utilizadas para ejecutar **bmj**. Si la ejecución es necesaria, utilice la opción **-J** para pasar opciones al método de inicio de **bmj**.

El compilador de Borland para Java (bcj)

Es una función de
JBuilder Developer y
Enterprise.

Sintaxis

```
bcj [ opciones ] {archivo.java}
```

Descripción

Como en el caso de JBuilder 9, el compilador de Borland para Java (**bcj**) utiliza **javac** para ofrecer compatibilidad con versiones anteriores. **bcj** también proporciona opciones adicionales del compilador, como **-exclude** para excluir clases y **-source 1.4** para activar asserts de JDK 1.4 en JDK 1.3. **bcj** compila código fuente Java para crear bytecodes Java desde la línea de comandos. **bcj** genera el programa Java en forma de archivos `.class` que contienen bytecodes que, a su vez, constituyen el código máquina de la máquina virtual Java. La compilación de un archivo fuente produce un archivo `.class` para cada declaración de clase o de interfaz. Cuando se ejecuta el programa de Java resultante en una plataforma concreta, como por ejemplo Windows NT, el intérprete de Java de dicha plataforma ejecuta los bytecodes de los archivos `.class`.

bcj compila el archivo `.java` especificado y todos los archivos indicados en la línea de comandos. **bcj** compila el archivo `.java` especificado, independientemente de que el archivo `.class` correspondiente esté o no desfasado. Un archivo `.class` desfasado es aquél que no se ha generado al compilar la versión actual del archivo fuente `.java` correspondiente. Los archivos `.java` importados que ya tienen archivos `.class` no vuelven a compilarse, aunque sus archivos `.class` no estén actualizados; después de utilizar **bcj**, algunas clases importadas pueden seguir teniendo archivos `.class` desfasados.

bcj no comprueba las dependencias entre archivos. Para obtener más información sobre **bmj** y la comprobación inteligente de dependencias, consulte [“Make de Borland para Java \(bmj\)” en la página A-9](#) y [“Comprobación inteligente de dependencias” en la página 5-2](#).

Para ver la sintaxis y la lista de opciones de la línea de comandos, introduzca el comando `bcj` sin argumentos que se encuentra en `<jbuilder>/bin`.

En ocasiones se debe utilizar la opción **-classpath** o definir la variable de entorno `CLASSPATH` para la línea de comandos, con el fin de encontrar las clases necesarias.

Nota Si desea utilizar la versión anterior de **bcj** de JBuilder 8, que no utiliza **javac**, emplee el comando `oldbcj`.

Consulte

- [“Compilación desde la línea de comandos” en la página 5-11](#)
- [“Definición de la variable de entorno CLASSPATH para herramientas de línea de comandos” en la página A-4](#)

- "Configuración de la vía de acceso a las clases" en la documentación de las herramientas de Java

Opciones

Nota Los directorios de las vías se muestran separados por dos puntos en UNIX y por punto y coma en Windows. Los siguientes ejemplos representan la plataforma UNIX.

-classpath viadeAcceso

Vía de acceso en la que se buscarán las clases. Redefine el valor por defecto o la variable de entorno `CLASSPATH`. Debe incluir siempre la vía de salida al principio de la vía. La vía de acceso a clases se utiliza también para buscar los archivos fuente si no se ha especificado ninguna vía de acceso a ellos. Por ejemplo:

```
bcj -classpath jbproject/testing/classes/test3:
jbproject/project1/classes tester.java
```

-d directorio

El directorio raíz de la jerarquía de archivos de clase (destino). También se denomina "vía de salida".

Por ejemplo, la siguiente sentencia:

```
bcj -d jbproject/project1/classes tester.java
```

hace que los archivos de las clases definidas en el archivo fuente `tester.java` se guarden en el directorio `jbproject/project1/classes/test/test3`, suponiendo que `tester.java` contenga la siguiente sentencia de paquete: `package test.test3;`

Los archivos se leen desde la vía de acceso a clases y se escriben en el directorio de destino. El directorio de destino puede ser parte de la vía de acceso a clases. El destino por defecto coincide con la estructura del paquete en los archivos fuente y se inicia desde el directorio raíz de archivos fuente.

-deprecation

Presenta todas las clases, métodos, propiedades, variables y sucesos desaconsejados empleados en la API.

Si se produce una advertencia, durante la compilación, de que se han utilizado API desaconsejadas, puede ver estas últimas activando esta opción.

-nombre de codificación

Puede especificar un nombre de codificación de archivo (o un nombre de ficha de códigos) para controlar cómo interpreta el compilador los caracteres que no pertenecen al conjunto de caracteres ASCII. Por omisión se utiliza el convertidor de codificación nativa por defecto de la plataforma. Para más

información, consulte el tema [“Elección de una codificación nativa para el compilador” en la página 17-13.](#)

Por ejemplo, la siguiente sentencia:

```
bcj -encoding EUC_JP tester.java
```

compila `tester.java`. Se considera que todos los archivos de código fuente están codificados en el conjunto de caracteres EUC_JP, que es el que se utiliza normalmente para los entornos UNIX japoneses. Puede especificar cualquier codificación que admita la plataforma Java 2. Para obtener más información acerca de la internacionalización y codificaciones válidas, consulte "native2ascii" en "SDK Development Tools".

-exclude classname

Excluye todas las llamadas a los métodos `static void` en el archivo `.class` seleccionado de una compilación. Esto también excluye la evaluación de los parámetros pasados a esos métodos.

Por ejemplo, la exclusión de la clase A elimina todas las llamadas a los métodos `static void` de A desde las OTRAS clases.

-g

Genera toda la información de depuración en el archivo de clase, incluidas las variables locales. Por defecto, sólo se generan números de línea y la información del archivo fuente.

-g:none

No genera ninguna información de depuración

-g:{lista de palabras clave}

Genera algunos tipos de información de depuración. La lista de palabras clave es una lista de palabras separadas por comas, como, por ejemplo: `bcj`

```
-g:lines,source
```

Entre las palabras clave se incluyen:

- **lines:** información de depuración del número de línea
- **vars:** información de depuración de la variable local
- **source:** información de depuración del archivo fuente

-nowarn

Compila sin mostrar advertencias.

-obfuscate

El camuflaje hace que los programas sean menos vulnerables a las manipulaciones. Cuando se descompila el código confuso o camuflado, el código fuente generado representa los símbolos privados mediante nombres distintos.

-quiet

Compila sin mostrar ningún mensaje.

-versión fuente

Habilita la compilación de código fuente que contiene órdenes.

- En la versión 1.4, esta opción de línea de comandos activa la palabra clave `assert` y la función de órdenes de JDK 1.4.
- En la versión 1.3 el compilador no acepta las órdenes.
- Si no se utiliza la opción `-source`, el comportamiento por defecto del compilador es el de la versión 1.3.

Consulte

- “Programming With Assertions” en la Guía de características de JDK

-sourcepath vía de acceso

Vía de acceso en la que se buscarán los archivos fuente. Si no se ha especificado ninguna vía de acceso a archivos fuente, se utilizará la vía de acceso a clases para buscar estos archivos.

Al igual que la vía de acceso a clases, la vía de acceso a archivos fuente debe situarse en el directorio raíz del árbol de paquetes y no directamente en el directorio de los archivos fuente.

Por ejemplo, para compilar `tester.java`, que contiene la sentencia de paquete `paquete test.test3` y que se encuentra en `jbproject/project1/src/test/test3`, debe definir la vía de acceso de archivos fuente como `jbproject/project1/src` y no como `jbproject/project1/src/test/test3`.

A continuación, escriba lo siguiente:

```
bcj -sourcepath jbproject/project1/src
    jbproject/project1/src/test/test3/tester.java
    -d jbproject/project1/classes
```

-versión tipo

Restringe los archivos de clase, de modo que funcionen sólo con una versión de MV determinada.

bcj admite:

- 1.1 – Genera archivos de clase para ejecutar en 1.1 y MV en Java 2 SDK. Si selecciona ésta como la MV destino, cualquier MV puede cargar sus archivos de clase.
- 1.2 - Genera archivos de clase que pueden ejecutarse **únicamente** en las MV del SDK de Java 2, versiones 1.2 y posteriores, pero **no** en las MV 1.1. Ésta es la opción por defecto.
- 1.3 - Genera archivos de clase que pueden ejecutarse **únicamente** en las MV del SDK de Java 2, versiones 1.3 y posteriores, pero **no** en las MV 1.1 ni 1.2.

- 1.4 - Genera archivos de clase que se ejecutan **únicamente** en máquinas virtuales de la versión 1.4 y posteriores del SDK de Java 2, pero **no** se ejecutan en 1.1, 1.2 ni 1.3.

-verbose

Esta opción proporciona más información sobre la compilación, como por ejemplo, los archivos de clase que se cargan y su ubicación en la classpath, además de la siguiente información:

- Qué archivos fuente se compilan.
- Qué clases se están cargando.
- Qué clases se generan.

Opciones de compilación cruzada

bcj acepta la compilación cruzada, que consiste en compilar las clases con un inicio y clases de extensión de una plataforma Java distinta.

Utilice **-bootclasspath** y **-extdirs** para realizar una compilación cruzada.

-bootclasspath bootclasspath

Compilación cruzada con las clases de inicio especificadas. Se pueden utilizar directorios, recopilatorios JAR y ZIP.

-extdirs directorios

Compilación cruzada con los directorios de extensión determinada. Se buscan archivos de clase para todos los archivos JAR de los directorios elegidos.

Opciones de MV

-Joption

Pasa opciones al programa de inicio de `java` al que llama **bcj**. Por ejemplo, **-J-Xms48m** configura la memoria de inicio en 48 megabytes. Es una convención común para **-J** pasar opciones a la MV subyacente que ejecuta aplicaciones escritas en Java.

Observe que `CLASSPATH`, **-classpath**, **-bootclasspath** y **-extdirs** no especifican las clases utilizadas para ejecutar **bcj**. Si debe hacerlo, utilice la opción **-J** con el fin de pasar opciones al método de inicio de **bcj**.

Índice

Símbolos

@author 15-4
@deprecated 15-4
@docRoot 15-4
@etiquetas, lista de 15-4
@exception 15-4
@link 15-4
@param 15-4
@return 15-4
@see 15-4
@serial 15-4
@serialData 15-4
@serialField 15-4
@since 15-4
@throws 15-4
@version 15-4

A

abrir archivos de proyecto 2-11

actualizar

 árbol de fuentes 13-6

 botón de información de revisiones (vista del histórico) 12-7

 clases después de compilar 8-76, 8-78

adición de bibliotecas 4-2

admitir serialización (opción) 10-22

advertencias

 perfeccionamiento 13-6

añadir

 al proyecto 2-14

 directorio de navegación al proyecto 2-21

 macros de línea de comandos 25-2

 sucesos a JavaBeans 10-12, 10-15, 10-16

 un JDK 2-25

Ant 6-12, 6-19

 añadir

 bibliotecas 6-20

 tipos al menú Proyecto 6-34

 archivos de generación 6-12

Consulte también archivos de generación Ant

 asignar valores a propiedades 6-23, 6-29

 asistentes 6-13

 Exportar a Ant 6-25

 configurar JDK 6-20

 exportar proyectos 6-25

 exportar proyectos EJB 6-28

 generar 6-12

 importación de proyectos Ant 6-25

 opciones 6-25

 tipos 6-12

 tutorial 18-1

 utilizar make de línea de comandos bmj 6-23

aplicaciones 6-1, 14-1

 compilar 5-1

 depurar 8-1

 distribuidas

 depuración

 distribuir 16-1, 16-9, 16-13

 ejecutar 7-1

 generar 6-1

 web, ejecutar 7-7

applets

 distribuir 16-1, 16-10

 ejecutar 7-1

 versiones JDK 16-6

appletviewer

 herramientas de línea de comandos 25-1

archivos

 abrir

 de nuevo 2-12

 fuera del proyecto 2-20

 añadir al archivo comprimido 16-21

 cambiar nombre 2-20

 cambio 2-10

 código fuente de stub 8-45

 comparar 12-1

 de clase 4-10

 cómo los encuentra JBuilder 4-13

 ubicación de los directorios 4-7

 de configuración 16-32

 Creador de ejecutables nativos 16-31

 JAR ejecutable 16-27

 de generación Ant 6-12

 añadir al proyecto 6-13

 crear 6-15

 desplazamiento 6-18

 editar 6-15

 especificar vías de acceso 6-16

 de grupos de proyectos 3-1

 de paquetes, documentación Javadoc 15-28

 de proyecto 2-1

 establecer vías de acceso

 guardar 2-11

 presentación 4-13

 vías de acceso 4-12

 de salida

 estándar para Javadoc 15-18

 JDK 1.1 para Javadoc 15-18

- descriptor 16-3
 - editar 16-24
 - ver en JBuilder 16-29
- ejecutables
 - crear 16-26
 - ejecutar 16-26
- ejecutar en el proyecto 7-6
- fuelle, path 4-9
- generados para Javadoc 15-26
- JAR 16-4
 - añadir al proyecto 4-1
 - archivo descriptor 16-3
 - crear con el Constructor de
 - recopilatorios 16-4
 - ejecución desde la línea de comandos 7-17, 16-24
- Java
 - ubicación de los directorios 4-7
- mover en panel de proyecto 2-14
- recopilatorios
 - borrar 16-30
 - cambiar nombre 16-30
 - crear archivos ejecutables 16-26
 - crear para distribuir 16-2
 - crear para la documentación 15-35
 - definir opciones de la configuración de
 - ejecución 16-27
 - descriptor 16-3
 - eliminar 16-30
 - JAR 16-4
 - Java 16-4
 - tipos aceptados por el Creador de
 - recopilatorios 16-16
 - ver contenido 16-29
 - ZIP 16-4
- ubicación 4-12
- ver archivos de proyecto 4-13
- ver en JBuilder 2-9
- ZIP 16-2
 - añadir al proyecto 4-1
 - crear con el Constructor de
 - recopilatorios 16-16
- de generación
 - Consulte también* archivos de generación
 - Ant
- área de trabajo
 - en control de versiones 12-1
- arrastrar y soltar archivos 2-14
- asignar valores a propiedades
 - en JavaBeans 10-4, 10-8
 - proyectos 2-23
- Asistencia a desarrolladores 1-7
- asistentes
 - conjuntos de tests 14-7
- creador de ejecutables nativos 16-31
- creador de compiladores 16-16
- exportar a Ant 6-25
- Javadoc 15-17
- montajes
 - de comparación 14-10
 - JDBC 14-9
 - JNDI 14-10
 - personalizado 14-11
- para
 - Ant 6-13
 - cliente de prueba EJB 14-8, 14-13
 - configuración de Cactus 14-12
 - conjuntos de tests 14-7
 - conjuntos de tests,tutorial 23-1
 - extracción de recursos 17-5
 - grupos de proyectos 3-1
 - JavaBean 10-2
 - tareas externas de generación 6-30
 - tests 14-6
 - tests, tutorial 23-1
- para Javadoc 15-1
 - ámbito de documentación 15-21
 - archivos de salida estándar 15-18
 - archivos de salida JDK 1.1 15-18
 - cambiar propiedades para el nodo 15-33
 - dar formato a los archivos de salida 15-18
 - directorio de salida 15-19
 - nombre del nodo de documentación 15-19
 - opciones de generación 15-19
 - opciones de línea de comandos 15-23
- para proyectos
 - crear nuevos proyectos 2-2
 - para código existente 2-7
 - Paso 1 Definir raíz, tipo y plantilla 2-3
 - Paso 2 Establecer vías de acceso,
 - bibliotecas, JDK 2-4
 - Paso 3 Opciones generales del proyecto 2-5
- recursos de cadenas 17-5
- tarea externa de generación 6-30
- Assert (clase) 14-6
- assert (palabra clave)
 - activar 2-5, 25-12, 25-19
- assertEquals() 14-6
- assertNotNull() 14-6
- assertTrue() 14-6
- atributos
 - EJBGen para Javadoc 15-7
 - XDoclet para Javadoc 15-7
- ayuda inmediata 11-16
- depurador 8-32
- diagramas UML 11-16

B

bajar

- campos 13-30
- métodos 13-28

barras

- de estado
 - depurador 8-29
 - progreso de la generación 7-3, 7-7
- de herramientas
 - depurador 8-29

BeanInfo (clases) 10-9

- cambiar 10-11
- crear 10-9
- generar automáticamente 10-10

BeanInsight 10-22

BeansExpress 10-1

- asignar valores a propiedades 10-4, 10-8
- borrar propiedades 10-8
- cambiar
 - clases BeanInfo 10-11
 - propiedades 10-7
- crear JavaBeans 10-2

bibliotecas 4-1

- adición y configuración 4-2
- añadir
 - al proyecto 2-4
 - bibliotecas Ant 6-20
 - proyectos, como necesarias 3-4, 4-5
- de libre distribución 16-6
- definición 4-1
- distribuir 16-6
- editar 4-4
- establecer vías de acceso 2-27
- incluir referencias en diagramas UML 11-21, 11-22
- mostrar listas 4-5
- necesarias 4-1, 4-5
 - añadir proyectos 3-4, 4-5

bloques de diferencias 12-1

- definición 12-1

Borland

- asistencia
 - a desarrolladores 1-7
 - técnica 1-7
- contacto 1-7
- correo electrónico 1-9
- grupos de noticias 1-8
- informar sobre errores 1-9
- recursos en línea 1-8
- World Wide Web 1-8

botones

- de comparación inteligente (vista del histórico) 12-7

- perfeccionar 13-7, 13-9

buscar

- definición 13-2
- método redefinido 13-3
- referencias 13-4
 - del visor UML 11-24
 - locales 13-3
- una llamada a un método 8-42

C

Cactus 14-2, 14-11

- configurar el proyecto 14-12
- ejecutar pruebas 14-13
- probar un EJB 14-8

cadena de inicialización de Java 10-19

cambiar

- clases BeanInfo 10-11
- código durante la depuración 8-76
- de archivo 2-10
- el JDK 2-25
- expresiones 8-74
- nombre

- campos 13-14, 13-18
- clases 13-14, 13-15
- métodos 13-14, 13-16
- paquetes 13-14
- propiedades 13-14, 13-19
- variables locales 13-14, 13-17
- valores de las variables 8-75

campos 11-9, 13-2

- bajar a subclase 13-30

buscar

- definición 13-2
- referencias a 13-3, 13-4

diagramas UML 11-9

- introducir en el código 13-25

Javadoc

- definir en el Asistente para proyectos 2-5
- subir a superclase 13-29

carpeta Filtros de paquetes 6-39

clases

- actualizar después de compilar 8-76, 8-78

buscar

- definición 13-2
- referencias a 13-3, 13-4

diagramas UML 11-6, 11-9

- documentación 15-2

ejecutable

- especificar en archivo descriptor 16-24

Java

- colecciones 10-1

PackageTestSuite 14-4

- redistribución en la distribución 16-13

TestSuite 14-1, 14-3

- codificaciones
 - añadir y redefinir 17-14
 - nativas 17-13, 17-14
 - definir 2-5, 17-2
- códigos 11-16
 - LegacyJ
 - depurar 8-32
 - modificar
 - durante la depuración 8-76
 - mostrar desde un diagrama UML 11-16
 - presentar *Consulte* UML
 - SQLJ 6-32
 - depurar 8-32
 - generar 6-32
- comando
 - nueva vista de directorio 2-21
- comentarios
 - información de revisiones 12-10
 - Javadoc 15-2
 - atributos EJBGGen 15-7
 - atributos XDoclet 15-7
 - colocación 15-3
 - conflictos 15-17
 - conflictos, corregir 15-17
 - editar 15-7
 - etiquetas @todo 15-13
 - generar automáticamente etiquetas 15-7, 15-9
 - JavadocInsight 15-10
 - usar etiquetas 15-4
 - y perfeccionamiento 15-7
 - Javadoc, añadir
 - para campos 15-3
 - para clases 15-3
 - para interfaces 15-3
 - para métodos 15-3
- cómo abrir de nuevo proyectos y archivos 2-12
- comparación inteligente, activar 12-7
- comparar archivos 12-1
 - excluir cambios de formato 12-7
 - sincronizar desplazamiento 12-2
 - vista de diferencias 12-2
 - vista de fuente 12-2
- compiladores
 - de Borland para Java, *Consulte también* bcj 25-9, 25-16
 - bcj 5-11, 25-16
 - bmj 5-11, 25-9
 - de Borland para Java 25-16
 - de línea de comando 5-11
 - bmj 25-9
 - definir en el IDE 5-9
 - definir opciones 5-7
 - errores 5-5
 - javac 5-9
 - javac del proyecto 5-9
 - Make de Borland 5-9
 - modificar 5-9
- compilar 5-1
 - archivos fuente 5-3
 - barra de estado 7-3, 7-7
 - comando
 - Generar de nuevo 6-5
 - Limpiar 6-5
 - cómo encuentra JBuilder los archivos 4-13
 - comprobación inteligente de dependencias 5-2
 - configurar vía de salida 5-10
 - copiar recursos en la vía de salida 6-42
 - definir opciones 5-7
 - definir un compilador diferente 5-9
 - descripción general 5-1
 - desde la línea de comandos 5-11
 - Ejecutar (comando) 5-5
 - Ejecutar Make 6-4
 - errores 5-5
 - al abrir proyectos 5-6
 - excluir paquetes 6-39
 - generar 5-1
 - Consulte también* generar con archivos Ant 18-1
 - grupos de proyectos 6-9
 - modificar compiladores 5-9
 - parámetros de todo el proyecto 5-7
 - programas
 - con información de depuración 8-4
 - en Java 5-1
 - proyectos en un grupo de proyectos 5-11
 - tutorial 20-1
- completar perfeccionamiento 13-9
- componentes
 - multiplataforma (entornos) 10-2
 - reutilizables 10-1
- comprobación
 - de dependencias 5-2, 5-7
 - de unidades 14-1
 - asistente para conjuntos de tests 14-7
 - asistente para tests 14-6
 - Cactus 14-2, 14-11
 - clase TestSuite 14-1, 14-3
 - código del servidor 14-2, 14-11
 - configuraciones de ejecución 14-4, 14-17
 - crear tests 14-5
 - depuración de tests 14-18
 - detección de tests 14-3
 - directorío fuente 14-6
 - EJB 14-8, 14-11, 14-13
 - ejecutar pruebas 14-14
 - filtro de seguimiento de pila 14-17

- JBTestRunner 14-15
- JUnit 14-1
- JUnit SwingUI 14-16
- JUnit TextUI 14-16
- lista de funciones 14-3
- métodos de test 14-6
- montaje de comparación 14-10
- montaje JDBC 14-9
- montaje JNDI 14-10
- montaje personalizado 14-11
- montajes para tests 14-8
- objetivos 14-5
- opción de menú Depurar test 14-3
- opción de menú Ejecutar test 14-3
- recopilador de tests de JUnit 14-4
- TestCase (clase) 14-1, 14-3
- tutorial 23-1, 24-1
- inteligente de dependencias 5-2
- comprobar 14-1
 - Consulte* test de módulos
 - código del servidor 14-11
 - EJB 14-11
- configuraciones 25-6
 - de ejecución
 - clase principal, determinar 7-10
 - crear 7-15
 - definir 7-8
 - depurar 8-4
 - gestionar 7-11
 - para comprobación de módulos 14-17
 - test 14-4
 - tipo de generación 7-12
 - ubicaciones del menú de ejecución 7-1
 - de inspección para clases sin archivo
 - fuente 8-45
 - de JBuilder 6-34
 - Menú de Proyecto 6-34, 6-36
 - mostrar con argumentos de la línea de comandos 25-6
 - tipos de ejecución 7-14
- configurar
 - bibliotecas 4-2
 - JDK 2-25
 - en JBuilder 2-27
- conflictos 8-38
 - de fusión
 - almacenar 12-11
 - en la fusión (ficha) 12-11
- conjuntos
 - de sucesos, crear personalizado 10-16
 - de tests 14-5
 - Consulte* test de módulos
- Contenido (ficha) 12-7
- control de versiones

- (tipos), iconos 12-6
- comparar archivos 12-1
- conflictos de fusión 12-11
- etiquetas locales 12-4
- Histórico (vista) 12-6
- convenciones
 - de la documentación 1-5
 - plataformas 1-6
 - de nomenclatura para paquetes 4-9
- correspondencia
 - directorio-paquete 5-7
 - paquete-directorio 5-7
- creador de ejecutables nativos 16-31
- creador de recopilatorios
 - crear
 - archivos ejecutables 16-26
 - archivos recopilatorios 16-4
 - recopilatorio de documentación 15-35
 - definir
 - opciones de la configuración de ejecución 16-27
 - distribuir archivos 16-16
 - tipos de recopilatorios 16-16
- crear
 - archivos de proyecto 2-17
 - BeanInfo (clases) 10-9
 - configuraciones de ejecución 7-8, 7-15
 - conjuntos de eventos personalizados 10-16
 - doclet personalizado 15-37
 - documentación 15-1
 - editores de propiedades 10-18
 - JavaBeans 10-1, 10-2
 - proyectos a partir de archivos 2-7
 - tests 14-6
- cuadros de diálogo
 - archivos modificados 8-78
 - nuevo editor de propiedades 10-18
 - propiedad localizable 17-8
 - propiedades de la paleta 10-23

D

- Datos de BeanInfo 10-10
 - cambiar 10-10
- definiciones
 - de las configuraciones para la ejecución 7-8
 - duplicadas de clases 5-7
- dependencias 11-9
 - diagramas UML 11-9
 - excluir de recopilatorios 16-19
- depuración
 - en proceso independiente 9-9
 - remota 9-1
 - abrir programa remoto 9-2

- depurar un programa en un ordenador remoto 9-2, 9-6
 - enlazar con un programa en ejecución 9-6
 - tutorial 21-1
- depurador 8-1
 - ayuda inmediata 8-32
 - barra de estado 8-29
 - barra de herramientas 8-29
 - definición de los intervalos de actualización 8-82
 - ejecución bajo su control 8-7
 - ExpressionInsight 8-31
 - interfaz de usuario 8-9
 - personalizar
 - opciones de 8-81
 - presentación de 8-80
 - teclas de método abreviado 8-30
 - vistas *Consulte* vistas del depurador y tests de módulos 14-18
- depurar 8-1
 - aplicación web 8-6
 - aplicaciones distribuidas
 - Consulte* depuración remota
 - archivo de clase 8-6
 - borrar puntos de observación 8-74
 - cambiar valores de datos 8-64
 - código
 - LegacyJ 8-32
 - local que se ejecuta en un proceso independiente 9-9
 - SQLJ 8-32
 - compilar programas con información de depuración 8-4
 - comprobación de tests 8-6, 14-3, 14-18
 - con JDK 1.2.2 8-3
 - configuraciones de ejecución 8-4
 - controlar
 - ejecución del programa 8-33
 - inspección de clases 8-43
 - crear puntos de interrupción 8-46, 8-47
 - definir puntos de ejecución 8-35
 - descripción general 8-3
 - desde la línea de comandos 8-1
 - desplazarse a través del código 8-38
 - detectar conflictos 8-38
 - edición de puntos de observación 8-74
 - ejecutar
 - hasta el final del método 8-41
 - hasta posición del cursor 8-41
 - hasta punto de ejecución 8-41
 - elegir hilo para inspección 8-37
 - errores lógicos 8-3
 - examen de los valores de datos del programa 8-62
 - excepciones de ejecución 8-2
 - finalización de la sesión 8-8
 - fuentes distintas de Java 8-32
 - gestionar hilos 8-36
 - iniciar sesión con -classic 8-7
 - inspección de clases con archivo fuente no disponible 8-45
 - inspeccionar
 - llamadas a métodos 8-39
 - salir de métodos 8-39
 - interrupción prolongada de un hilo 8-37
 - interrupción y ejecución del depurador 8-33
 - ir a puntos de observación de variables de ámbito en el editor 8-72
 - Java Process Debugging Architecture (JPDA)-Arquitectura Java de depuración de procesos) 8-3
 - JSP 8-32
 - la opción -classic 2-25
 - modificar
 - código 8-76
 - valores 8-75
 - mostrar
 - hilo actual 8-36
 - marco de pila superior 8-37
 - objetos como cadenas 8-66, 8-67
 - variables estáticas y locales 8-62
 - observación de expresiones 8-70
 - paso inteligente 8-40
 - pausar la ejecución del programa 8-8
 - proyecto 8-6
 - puntos de
 - ejecución 8-34
 - interrupción interprocesal 9-10
 - interrupción y configuración de Inspección desactivada 8-46
 - reinicio del programa 8-34
 - remoto
 - sesiones 8-9
 - de inicio 8-6
 - tutorial 20-1
 - visualizar llamadas a métodos 8-42
- descendientes 13-4
- desde la línea de comandos 25-6
 - argumentos de JBuilder 5-12, 25-6
 - cambio a IDE durante la compilación 5-12
 - compilar 5-11
 - ejecutar archivos JAR 7-17
 - ejecutar programas distribuidos 7-17
 - generar proyectos 5-12
 - herramientas 25-1
 - Interfaz de la línea de comandos de JBuilder 25-6
 - usar macros de línea de comandos 25-2

- desplazamiento
 - archivos de generación Ant 6-18
 - diagramas UML 11-18
- diagramas UML 11-4, 11-13
 - Consulte también* UML
 - ayuda inmediata 11-16
 - buscar referencias 11-24
 - carpetas del panel de estructura 11-9
 - definición 11-9
 - diagramas de clases 11-6, 11-15
 - diagramas de paquetes 11-5, 11-15
 - filtrar 11-20
 - guardar como imágenes 11-23
 - iconos de accesibilidad 11-11
 - impresión 11-23
 - incluir
 - referencias de biblioteca 11-21
 - referencias del código generado 11-22
 - perfeccionar código fuente 11-24
 - personalizar 11-20
 - presentar clases y paquetes 11-13
 - valores etiquetados 11-11
 - ver clases internas 11-15
- diferencias 12-1
 - comparar dos archivos cualesquiera 12-2
 - definición 12-1
- directorios
 - de navegación 2-21
 - añadir 2-21
 - de trabajo 4-12
 - fuentes
 - test 14-6
- diseñador de BeanInfo 10-10
- diseño
 - interfaces de usuario JavaBean 10-4
- distribución 16-5
 - applets escritas con JDK 1.1.x o Java 2 16-6
 - applets y aplicaciones 16-7
 - bibliotecas 16-6
 - bibliotecas de CLASSPATH 16-6
 - redistribución de clases 16-13
 - tiempo de descarga de archivos 16-8
- distribuir
 - aplicaciones 16-1, 16-9
 - como compilatorios 16-2
 - distribuidas 16-13
 - applets 16-10
 - en Internet 16-13
 - JavaBeans 16-11
 - sugerencias 16-12
- doclet para Javadoc 15-18, 15-37
- documentación 11-16
 - de campo 15-2
 - de interfaz 15-2

- de Javadoc 15-1
 - ámbito de documentación 15-21
 - añadir comentarios 15-2
 - archivos de comentarios de aspectos
 - generales 15-28
 - archivos de detalles de paquetes 15-28
 - archivos de salida estándar 15-18
 - archivos de salida JDK 1.1 15-18
 - archivos generados 15-26
 - dar formato a los archivos de salida 15-18
 - directorio de salida 15-19
 - generar archivos de paquetes 15-28
 - generar archivos de salida 15-26
 - mantener 15-32
 - nombre del nodo 15-19
 - opciones de generación 15-19
 - opciones de línea de comandos 15-23
 - ver desde el panel de proyecto 15-30
 - ver desde un diagrama UML 11-16
 - ver en el Visor de ayuda 15-30
 - ver en pestaña Documentación 15-30, 15-32
 - ver un solo archivo 15-32
- de método 15-2
- Consulte*
 - comentarios Javadoc
- documentación de Javadoc 11-16, 15-1

E

- editar
 - JDK del proyecto 2-24
- editor
 - y ejecutores de test 14-15
- editores
 - de propiedades 10-10
 - cadena de inicialización de Java 10-19
 - componente personalizado 10-21
 - crear 10-18
 - Lista de cadenas 10-18
 - Lista de etiquetas de cadena 10-19
 - optimizar importaciones 13-11
- EJB
 - comprobación de unidades 14-11
 - exportar a Ant 6-28
 - perfeccionamiento 13-7
 - tests con Cactus 14-13
- ejecución del programa
 - controlar 8-33
- ejecutables
 - archivos de configuración 16-32
 - lanzar desde los archivos de configuración 16-27, 16-31
 - personalizar archivos de configuración 16-27, 16-31

- ejecutar
 - aplicaciones 7-1
 - desde la línea de comandos 7-17
 - web 7-7
 - applets 7-1
 - archivos individuales 7-6
 - bajo control del depurador 8-7
 - comprobación de unidades 14-3, 14-15
 - mensajes de error 7-2
 - OpenTools 7-5
 - proyectos 7-1, 7-3
 - proyectos agrupados 7-4
 - tests Cactus 14-13
 - tutorial 20-1
 - Ejecutar (comando)
 - generar 5-5
 - Ant 6-22
 - ejecutores de test 14-14
 - definir 14-17
 - disponibles en JBuilder 14-1
 - JJUnitRunner 14-15
 - JJUnit AwtUI 14-1
 - JJUnit SwingUI 14-16
 - JJUnit TextUI 14-16
 - y editor 14-15
 - errores 5-5
 - errores de sintaxis 5-5
 - lógicos 8-3
 - mensajes de error 5-5
 - tiempo de ejecución 8-2
 - tipos 8-2
 - establecer vías de acceso
 - archivos fuente 4-9
 - bibliotecas necesarias 2-27
 - CLASSPATH 25-4
 - copia de seguridad 4-12
 - doc 4-11
 - herramientas de línea de comandos 25-3
 - JDK 2-24
 - Consulte también* JDK
 - output 4-10
 - vía de acceso a clases 4-10
 - vía de búsqueda 4-11
 - vías de acceso de proyecto 2-4
 - etiquetas
 - @todo 15-13
 - en tests 14-6
 - presentación 15-13
 - de proyectos
 - todo, ver 15-13
 - Javadoc 15-4
 - @author 15-4
 - @deprecated 15-4
 - @docRoot 15-4
 - @exception 15-4
 - @link 15-4
 - @param 15-4
 - @return 15-4
 - @see 15-4
 - @serial 15-4
 - @serialData 15-4
 - @serialField 15-4
 - @since 15-4
 - @throws 15-4
 - @version 15-4
 - creación personalizada 15-14
 - crear 15-14
 - editar 15-14
 - generar automáticamente 15-7, 15-9
 - introducir en archivos fuente 15-4
 - locales 12-4
 - para Javadoc 15-2
 - revisión 12-10
 - todo 15-13
 - todo, ver 15-13
 - evaluación de expresiones 8-74
 - excepciones
 - comprobación de unidades 14-6
 - exponer BeanInfo de superclase (opción) 10-10
 - exportar a Propiedades Ant 6-29
 - expresiones
 - evaluar y modificar 8-74
 - observar 8-70
 - ExpressionInsight 8-31
 - extraer
 - interface 13-31
 - method 13-22
- ## F
-
- fases
 - compilar 6-3
 - de generación 6-3
 - compilar 6-3
 - definición 6-1
 - distribuir 6-3
 - generar de nuevo 6-3
 - limpiar 6-3
 - make 6-3
 - package 6-3
 - postcompilar 6-3
 - precompilar 6-3
 - distribuir 6-3
 - ejecutar make 6-3
 - empaquetar 6-3
 - generar de nuevo 6-3
 - sin filtros 6-41
 - limpiar 6-3
 - postcompilar 6-3

- precompilar 6-3
- fecha de revisión de archivo 12-10
- filtrar
 - excluir paquetes de la generación 6-39
 - omitir al generar 6-41
- filtros 6-39
 - creación para recopilatorio 16-19
 - de seguimiento de pila, tests de módulos 14-17
 - diagramas UML 11-20
 - excluir paquetes de la generación 6-39
 - modificación para recopilatorio 16-21
 - omitir al generar 6-41
 - seguimiento de pila de tests de módulos 14-17
- fuentes
 - convenciones de la documentación 1-5
 - distinta de Java, depuración 8-32
 - inteligente 8-32
 - mostrar fuentes internacionales 17-10
- funciones acordes con la versión localizada 17-10
- fusionar
 - paquetes 13-14

G

- generar 6-1
 - Ant 6-12
 - antes de perfeccionar 13-6
 - archivos SQLj 6-32
 - asignar valores a propiedades 6-7
 - atributos
 - EJBGen para etiquetas Javadoc 15-7
 - XDoclet para etiquetas Javadoc 15-7
 - comandos
 - ejecutar y Ant 6-22
 - generar de nuevo 6-5
 - limpiar 6-5
 - compilar 5-1
 - Consulte también* compilar
 - con referencias de bibliotecas de proyecto 13-2
 - configurar preferencias 6-8
 - copiar recursos en la vía de salida 6-42
 - creación de tareas externas de
 - generación 6-30
 - definir opciones 5-9
 - descripción general 6-1, 6-2
 - desde la línea de comandos 25-6
 - Ejecutar (comando) 5-5
 - Ejecutar Make 6-4
 - en segundo plano 6-8
 - etiquetas Javadoc 15-7, 15-9
 - excluir paquetes 6-39
 - fases 6-2, 6-3
 - generar de nuevo sin filtros 6-41
 - grupos de proyectos 6-9
 - JavaBeans 10-2

- Javadoc 15-1, 15-26
- mensajes
 - Ant 6-19
 - de generación 6-31
- programas en Java 6-1
- proyectos 6-1, 6-19
 - Ant 6-12, 6-19
 - comandos
 - Ejecutar y Ant 6-22
 - desde la línea de comandos 5-12, 25-6
 - ejecutar (comando) 5-5
 - recopilación automática de paquetes
 - fuentes 6-38
 - tareas de generación 6-2
 - externas 6-30
 - términos definidos 6-2
 - texto para etiquetas Javadoc 15-7
 - tipos 6-2
- gestión
 - de revisiones
 - etiquetas locales 12-4
 - Histórico (vista) 12-6
 - de versiones
 - restaurar las versiones anteriores de
 - archivos 12-7
- glosario de términos de gestión de versiones 12-1
- gráficos 11-23
 - imágenes PNG 11-23
- grupos
 - de noticias 1-8
 - Borland y JBuilder 1-8
 - public 1-8
 - Usenet 1-8
 - de proyectos 3-1
 - añadir
 - proyectos 3-1, 3-3
 - añadir tipos al menú Proyecto 6-36
 - compilar 6-9
 - configuraciones de ejecución 7-11
 - configurar menú Grupo de proyectos 6-36
 - crear 3-1
 - dependencias entre proyectos 3-4
 - desplazamiento 3-4
 - ejecutar 7-4
 - eliminar proyectos 3-3
 - generar 6-9
 - orden de generación de proyectos 3-1
 - reordenar 3-4
 - ventajas 3-1
- guardar varios proyectos 2-29

H

- herramientas
 - appletviewer 25-1

- de línea de comandos 25-1
 - bcj 25-16
 - bmj 25-9
 - definir CLASSPATH 25-4
 - establecer vías de acceso 25-3
 - jar 25-1
 - java 25-1
 - javac 25-1
 - javadoc 25-1
 - JBuilder 25-6
 - native2ascii 25-1
- Interfaz de la línea de comandos de JBuilder 25-6
 - jar 25-1
 - java 25-1
 - javac 25-1
 - javadoc 25-1
 - native2ascii 25-1
- hilos
 - detectar conflictos 8-38
 - elegir para inspección 8-37
 - gestionar 8-36
 - interrupción prolongada 8-37
 - mostrar el actual 8-36
 - mostrar marco de pila superior 8-37
 - panel dividido 8-36
- Histórico (vista) 12-6
 - conflictos
 - en la fusión (ficha) 12-11
 - Contenido (ficha) 12-7
 - descripción general 12-6
 - iconos 12-6
 - Información (ficha) 12-10
 - tutorial 19-1

I

- iconos
 - control de versiones (tipos) 12-6
 - de visibilidad, diagramas UML 11-11
 - especificar JavaBean 10-10
 - Histórico (vista) 12-6
 - JJUnitRunner 14-15
 - lista de revisiones 12-6
 - UML 11-11
- imágenes UML 11-23
 - guardar diagramas 11-23
 - imprimir diagramas 11-23
- inclusión en recursos
 - cadena 17-5
 - definición 17-2
- información
 - de depuración 8-4
 - de revisión, ver 12-10
 - ficha 12-10

- informar sobre errores 1-9
- Informe de limitaciones 13-6
- insertar en sentencia try/catch 13-26
- inspección 8-38
 - inteligente 8-39, 8-40
 - llamadas a métodos 8-39
 - salir de métodos 8-39
- inspeccionar clases, activar y desactivar 8-43
- instalar
 - JavaBeans en la paleta de componentes 10-23
- Intercambio inteligente 8-76
 - y la MV de destino 8-76
- interfaces 11-9
 - de la línea de comandos
 - JBuilder 25-6
 - de usuario
 - diseñar JavaBean 10-4
 - diagramas UML 11-9
 - extraer 13-31
- interfaz
 - de la línea de comandos 25-6
- internacionalización 17-3, 17-16
 - características del depurador 17-12
 - controles homólogos no nativos 17-10
 - dbSwing 17-9
 - definición 17-2
 - funciones
 - de compilador 17-13
 - de JBuilder 17-3
 - del diseñador visual 17-11
 - opciones de codificación 17-13
 - programas 17-1
 - términos y definiciones 17-2
- Internet
 - distribuir aplicaciones 16-13

J

- Java 11-2
 - y UML 11-2
- Java Process Debugging Architecture (JPDA - Arquitectura Java de depuración de procesos) y depurar 8-3
- JavaBeans 10-2
 - añadir sucesos 10-12, 10-15, 10-16
 - cambiar propiedades 10-7
 - crear 10-1, 10-2
 - definición 10-1
 - diseñar una interfaz de usuario 10-4
 - distribuir 16-11
 - eliminar propiedades 10-8
 - establecer propiedades 10-4, 10-8
 - instalar 10-23
 - mostrar valores de las propiedades 10-10
 - personalizar 10-9

- serialización 10-22
- validez 10-22
- válidos, comprobar 10-22
- ventajas 10-2
- javac 25-1
 - compilar con 5-9
- Javadoc 15-1
 - Asistente 15-17
 - comentarios 15-2
 - Etiquetas Javadoc 15-4
 - JavadocInsight 15-10
 - presentación 15-30
- JavadocInsight 15-10
- JBTestRunner 14-15
 - Fallos del test 14-15
 - iconos 14-15
 - Jerarquía del test 14-15
 - Resultados del test 14-15
 - tutorial 23-1
 - y editor 14-15
- JBuilder
 - grupos de noticias 1-8
 - informar sobre errores 1-9
 - versiones internacionales 17-16
- JDK
 - cambiar y configurar 2-25
 - configuración en JBuilder 2-27
 - configurar proyectos Ant 6-20
 - definir en el Asistente para proyectos 2-4
 - depurar con -classic 2-25
 - editar 2-24
- JDK 1.2.2 y depurar 8-3
- Jerarquía del test (ficha) 14-15
- JSP
 - depurar 8-32
- JUnit 14-1
 - Assert (clase) 14-6
 - assertEquals() 14-6
 - assertNotNull() 14-6
 - assertTrue() 14-6
 - AwtUI 14-1
 - clase TestSuite 14-1, 14-3
 - ejecutores de test 14-1
 - integración en JBuilder 14-1
 - probar un EJB 14-8
 - setUp() 14-5
 - SwingUI 14-1, 14-16
 - tearDown() 14-5
 - TestCase (clase) 14-1, 14-3
 - TextUI 14-1, 14-16

L

- lista de revisiones
- actualizar 12-7

- iconos 12-6
- ordenar 12-7
- llamadas a métodos
 - buscar 8-42
 - evaluar 8-75
 - presentación 8-42
- localización 17-1
 - automática de archivos fuente 6-44
 - añadir tipos de archivo no reconocidos como recursos 6-44
 - de fuentes 6-38
 - de paquetes 6-38
 - definición 17-2

M

- macros, línea de comandos 25-2
 - añadir a vías de acceso y campos de parámetros 25-2
 - insert 25-2
- Make de Borland para Java 25-9
 - Consulte también* bmj
- make de línea de comandos bmj 5-11, 25-9
 - utilizar con Ant 6-23
- Make de Borland para Java
 - Consulte también* bmj
- mantener Javadoc 15-32
- mensajes de error 5-5
 - depurador 8-2
 - en tiempo de compilación 7-2
 - perfeccionamiento 13-6
- menú
 - de Proyecto 6-34
- menús 6-34
 - configurar el menú Proyecto 6-34, 6-36
 - de generación 5-4
 - añadir tipos 6-34, 6-36
 - de Proyecto
 - añadir tipos 6-34
 - configurar 6-34, 6-36
- métodos 11-9
 - bajar a subclase 13-28
 - buscar definición 13-2
 - buscar redefinido 13-3
 - buscar referencias a 13-3, 13-4
 - de test
 - requisitos 14-6
 - diagramas UML 11-9
 - ejecución hasta el final 8-41
 - redefinido
 - buscar en el código 13-3
 - subir a superclase 13-27
- modificar
 - parámetros de métodos 13-20
 - valores de datos en el depurador 8-64

- montajes 24-1
 - de comparación 14-10
 - asistente 14-10
 - tutorial 24-1
- JDBC 14-9
 - asistente 14-9
 - tutorial 24-1
- JNDI 14-10
 - para tests 14-8
 - ejemplo 14-8
 - montaje de comparación 14-10
 - montaje JDBC 14-9
 - montaje JNDI 14-10
 - personalizada 14-11
 - predefinidos 14-8
 - personalizado 14-11
 - asistente 14-11
- mostrar
 - archivos
 - de proyecto 4-13
 - variables estáticas y locales 8-62
- MV 5-7
 - definir MV destino 5-7
 - depurar con -classic 2-25

N

- nodo de documentación 15-17
 - ampliar 15-30
 - Asistente para Javadoc 15-17
 - cambiar propiedades 15-33, 15-34
 - propiedades para 15-17
- nodo de ejecutables nativos
 - asignar valores a propiedades 16-31
- nodo de recopilatorios
 - borrar 16-30
 - cambiar nombre 16-30
 - eliminar 16-30
 - generar 16-28
 - propiedades 16-30
 - restablecer propiedades 16-29
 - ver contenido 16-29
- nodo Fuente del proyecto 6-38, 6-39
 - filtrar paquetes 6-39

O

- objetos
 - presentar como cadenas 8-66, 8-67
- observación de expresiones 8-70
- opciones
 - classic para depurar 8-7
 - de codificación 17-13
 - definir 17-13
 - por defecto 17-13

- de codificación por defecto 17-13
- de la configuración de ejecución
 - definir en el archivo recopilatorio 16-27
- de línea de comandos 5-12, 25-6
 - bcj 25-16
 - bmj 25-9
 - en el Asistente para Javadoc 15-23
 - JBuilder 25-6
 - para Javadoc 15-23
- de recopilación automática de paquetes
 - fuentes 2-5
- de UML
 - filtrado de paquetes y clases 11-20
 - incluir referencias de biblioteca 11-21
 - incluir referencias del código
 - generado 11-22
 - personalizar IDE 11-22
 - propiedades de proyecto 11-20
- del IDE
 - UML 11-22
- MV destino 5-7
- OpenTools
 - activar el modo de depuración verbose con argumentos de la línea de comando 25-6
 - ejecutar 7-5
- optimizar importaciones 13-11
- ordenar la lista de revisiones 12-7
- organizar proyectos 2-14

P

- páginas de códigos 17-14
- palabras clave
 - assert 2-5, 25-12, 25-19
- paleta de componentes
 - instalar JavaBeans 10-23
- paneles
 - de contenido
 - Histórico (vista) 12-6
 - de estructura
 - carpeta Errores 5-5
 - Carpeta Por hacer 15-13
 - conflictos Javadoc 15-17
 - UML 11-18
 - de mensajes
 - mostrar etiquetas todo 15-13
 - del proyecto 2-1
 - 2-14
 - ver recopilatorios 2-10, 2-29
- pantalla de inicio, desactivar 25-6
- paquetes 4-6
 - activar la localización de paquetes fuente 6-38
 - convenciones de nomenclatura 4-9
 - diagramas UML 11-5, 11-9
 - excluir del proceso de generación 6-39

- filtrar 6-39
- localización de paquetes fuente 6-38
- recopilación automática de paquetes
 - fuelle 6-38
- referencias a clases 4-8
- para tests
 - tutorial 24-1
- parámetros de métodos
 - modificar 13-20
- paso inteligente 8-39, 8-40
 - opciones 8-40
- pausar la ejecución del programa 8-8
- pautas de diseño 10-1
 - JavaBean 10-1
- perfeccionamiento 13-1
 - Actualización del árbol de fuentes 13-6
 - advertencias 13-6
 - bajar campos 13-30
 - bajar métodos 13-28
 - cambiar de nombre 13-14
 - Consulte* perfeccionamiento por cambio de nombre
 - cambiar parámetros de métodos 13-20
 - completar 13-9
 - desde el visor UML 11-24
 - deshacer 13-10
 - desplazar 13-19
 - Consulte* perfeccionamiento por desplazamiento
 - Detección de referencias 13-2
 - EJB 13-7
 - extraer
 - interfaces 13-31
 - método 13-22
 - generar antes 13-6
 - guardar 13-11
 - herramientas de JBuilder 13-6
 - Informe de limitaciones 13-6
 - insertar en sentencia try/catch 13-26
 - introducir
 - campos 13-25
 - superclases 13-32
 - variable 13-24
 - mensajes de error 13-6
 - optimizar importaciones 13-11
 - por cambio de nombre 13-14
 - campos 13-14, 13-18
 - clases 13-14, 13-15
 - métodos 13-14, 13-16
 - paquetes 13-14
 - propiedades 13-14, 13-19
 - variables locales 13-14, 13-17
 - por desplazamiento 13-19
 - subir
 - campos 13-29
 - métodos 13-27
 - validación 13-6
 - vista previa de los cambios 13-7
 - y Javadoc 15-7
- personalizar
 - JavaBeans 10-9
- pestaña Perfeccionar 13-7, 13-9
- plantillas
 - de proyecto 2-1
 - proyecto
 - definir 2-3
- posición del cursor, ejecutar 8-41
- presentación
 - versiones de archivo anteriores 12-6
- presentar código 11-1
 - Consulte también* UML
- probar directorio fuente 14-6
- programas 6-1
 - compilar 5-1
 - de inicio
 - archivos de configuración 16-32
 - ejecutables 16-32
 - definición 14-1
 - depurar 8-1
 - distribuir 16-1
 - ejecutar 8-33
 - generar 6-1
 - interrupción 8-33
- PropertyChangeSupport (clase) 10-8
- propiedades 5-7, 11-9
 - Ant 6-23
 - componentes JavaBean 10-4, 10-8
 - Creador de ejecutables nativos 16-31
 - de proyecto 11-20
 - configurar para generar 5-7
 - configurar vía de salida 5-10
 - configurar para detectar referencias 13-2
 - definición para UML 11-20
 - directorio fuente para los tests 14-6
 - diagramas UML 11-9
 - exportar a Ant 6-29
 - generar 5-7, 6-7
 - modificar 10-18
 - monitorizables
 - definir para JavaBeans 10-8
 - nodo de compilatorios 16-29
 - ocultar 10-10
 - personalizar 10-10
 - definir para JavaBeans 10-8
 - tareas externas de generación 6-32
- proyectos 2-1
 - abrir 2-11

- abrir de nuevo 2-12
- agrupados
 - ejecutar 7-4
- añadir
 - a grupos de proyectos 3-1
 - archivos o paquetes 2-14
 - archivos ZIP o JAR 4-1
 - archivos, paquetes 2-17
 - como bibliotecas necesarias 3-4, 4-5
 - directorio de navegación 2-21
 - tipos al menú Proyecto 6-34
- Ant 6-12
- asignar valores a propiedades 2-23
- cambiar
 - de uno a otro 2-29
 - nombre 2-20
- cambio entre compiladores 5-9
- cerrar 2-11
- cómo añadir carpetas 2-15
- compilar 5-1
- configurar
 - el menú Proyecto 6-34
 - para Cactus 14-12
- crear 2-2
 - con el asistente Proyecto para código existente 2-7
 - y añadir archivos 2-17
- crear con Asistente para proyectos
- definir parámetros generales 2-5
- ejecutar 7-1, 7-3
- eliminar
 - archivos 2-20
 - carpetas y archivos 2-19
 - clases y paquetes 2-19
- exportar a Ant 6-25
- guardar 2-11
 - varios 2-29
- importación de proyectos Ant 6-25
- uso de varios proyectos 2-28
- ver archivos 2-9
- ver recopilatorios 2-10
- pruebas de regresiones 14-1
- puntos de ejecución 8-34
 - definir 8-35
- puntos de interrupción 8-46
 - acciones 8-57
 - activar 8-60
 - buscar 8-61
 - campo 8-53
 - clase 8-51
 - class 8-51
 - condicional 8-59
 - definir 8-59
 - definir 8-47

- desactivar 8-60
- ejecutar hasta 8-41
- eliminar 8-61
- exception 8-49
- interprocesal 8-54, 9-10
- línea 8-48
- method 8-52
- método 8-52
- número de pasadas 8-60
- para depurar tests 14-18
- por excepción 8-49
- propiedades 8-56
- sobreescribir Inspección desactivada 8-46
- puntos de observación 8-70
 - borrar 8-74
 - de objetos 8-73
 - de variables 8-71
 - de variables con nombre 8-71
 - de variables de ámbito 8-72
 - ir a en el editor 8-72
 - editar 8-74

R

- readObject() 10-22
- recopilación automática de paquetes fuente 6-38
 - niveles jerárquicos mostrados 6-38
 - Nodo Fuente del proyecto 6-38
- Recopilador de tests de JUnit 14-4
- recopilar
 - archivo descriptor 16-3
 - archivos fuente 16-16
 - con el Creador de ejecutables nativos 16-31
 - con el Creador de recopilatorios 16-4
 - Creador de recopilatorios 16-16
 - documentación 16-16
 - excluir dependencias 16-19
 - proyectos para distribución 16-2
- recopilatorios
 - añadir archivos 16-21
 - Creador de recopilatorios 16-16
 - crear filtros 16-19
 - de documentación, creación 15-35
 - excluir dependencias 16-19
 - incluir tipos de archivo no reconocidos como recursos 6-44
 - modificar filtros 16-21
 - ver desde el panel de proyecto 2-29
 - ver en visualizador 2-10
- recursos 6-42
 - añadir tipos de archivos no reconocidos 6-44
 - copiar en vía de salida 6-42
 - recopilación automática de paquetes fuente 6-38
 - tipo de archivo de recursos genérico 6-44

- redistribución de clases 16-13
- referencias
 - a clases 4-8
 - buscar método redefinido 13-3
 - de bibliotecas 2-5
 - de escrituras 13-4
 - de lecturas 13-4
 - de miembros 13-4
 - de miembros descendientes 13-4
 - de usos directos 13-4
 - de usos indirectos 13-4
 - detectar 13-2
 - locales, buscar 13-3
- repositorio, definido 12-1
- revisión
 - etiqueta 12-10
 - fecha 12-10

S

- sentencias
 - import 4-8
 - optimizar 13-11
- serialización
 - admitir 10-22
- servidor de depuración 9-1
 - abrir programa remoto 9-2
 - enlazar con un programa remoto 9-6
- setUp() 14-5
- Sincronizar botón de desplazamiento (vista del histórico) 12-7
- sincronizar desplazamiento en visualizadores de código fuente 12-7
- sistema de generación 6-1
- stubs
 - archivos fuente 8-45
- subir
 - campos 13-29
 - métodos 13-27
- sucesos
 - añadir a JavaBeans 10-12, 10-15, 10-16
- superclases 10-10
 - introducir 13-32
- SwingUI 14-16

T

- TagInsight
 - edición de archivos de generación Ant 6-15
- tareas de generación 6-30
 - asignar valores a propiedades 6-32
 - Asistente 6-30
 - configuración de las propiedades Ant 6-23
 - tareas externas de generación 6-31
- tareas externas de generación 6-30

- tearDown() 14-5
- test del servidor 14-2, 14-11
- TestCase (clase) 14-1, 14-3
 - ampliar 14-5
 - setUp() 14-5
 - tearDown() 14-5
- TestRecorder (clase) 14-10
- tests 14-5
 - Consulte* test de módulos
 - ejecutar desde main() 14-17
 - etiquetas @todo 14-6
 - orden de ejecución 14-5
 - setUp() 14-5
 - tearDown() 14-5
- TextUI 14-16
- tiempo de ejecución
 - errores 8-2
 - excepciones 8-2
- tipos 6-2
 - Ant 6-12
 - configuración de ejecución 7-12
 - configuración de las propiedades Ant 6-23
 - configurar propiedades de tareas externas 6-32
 - de archivo
 - de recursos genérico 6-44
 - de configuraciones de ejecución 7-14
 - de ejecutor 7-14
 - de generación 5-5
 - Ant 6-12
 - configuración de ejecución 7-12
 - de referencias 13-4
 - antecedentes 13-4
 - de antecedentes 13-4
 - de declaraciones 13-4
 - de miembros 13-4
 - de miembros descendientes 13-4
 - de tipo 13-4
 - de tipo descendientes 13-4
 - declaraciones 13-4
 - descendientes 13-4
 - escrituras 13-4
 - lecturas 13-4
 - usos directos 13-4
 - usos indirectos 13-4
- definición 6-2
- generar 5-5

- tipos de archivo 6-44
- archivo de recursos genérico 6-44
- trasladar
- clases a paquetes 13-19
- try/catch, insertar bloque de código 13-26
- tutoriales 18-1
- compilación, ejecución y depuración 20-1
- comprobación de unidades 23-1

- depuración remota 21-1
- generación con archivos Ant 18-1
- montajes para tests 24-1
- UML 22-1
- utilizar la vista del histórico 19-1

U

- ubicación de documentos 4-11

- UML 11-1

- definición 11-1
 - descripción general 11-1
 - términos 11-2
 - tutorial 22-1
 - y Java 11-2

- Unicode 17-12

- ASCII de 7 bits 17-15
 - definición 17-2
 - formato de 16 bits 17-15
 - introducir 17-11
 - mostrar 17-10

- Usenet, grupos de noticias 1-8

V

- valores

- de datos
 - cambiar en el depurador 8-64
 - del programa 8-62
 - examinar durante la depuración 8-62

- valores de datos

- examinar durante la depuración 8-62

- variables

- buscar definición 13-2
 - buscar referencias a 13-4
 - de entorno 25-4
 - introducir en el código 13-24
 - modificar valores 8-75

- varios proyectos 2-28

- cambiar de uno a otro 2-29
 - guardar 2-29

- ver

- archivos 2-9
 - de proyecto 4-13
 - documentación de Javadoc
 - del panel del proyecto 15-30
 - desde un diagrama UML 11-16, 15-30
 - en el Visor de ayuda 15-30
 - en la pestaña Doc 15-30
 - para archivo individual 15-30
 - para todo el proyecto 15-30
 - documentación del API 11-16
 - visualizador UML 11-16

- versiones

- anteriores de archivos, restaurar 12-7

- internacionales de JBuilder 17-16

- localizada

- definición 17-2

- vías

- de acceso 4-1, 4-9
 - compilación, ejecución y depuración 4-13
 - definir 25-3
 - ubicación de archivos de clase 4-7
 - ubicación de archivos Java 4-7
 - de acceso a archivos fuente
 - archivos de clase 4-7
 - Archivos Java 4-7
 - de acceso a clases 4-10
 - variable de entorno 25-4
 - de acceso a copias de seguridad 4-12
 - de acceso de proyecto 2-4
 - definir 2-4
 - de archivos generados
 - definir 5-10
 - de búsqueda 4-11
 - de salida 4-10
 - de acceso

- Consulte* establecer vías de acceso

- vías de acceso

- configurar vía de salida 5-10

- vistas

- Clases cargadas y datos estáticos 8-24
 - Clases con inspección desactivada 8-12
 - del depurador 8-10
 - Clases cargadas y datos estáticos 8-24
 - Clases con inspección desactivada 8-12
 - Hilos, pilas de llamada y datos 8-16
 - Monitores de sincronización 8-27
 - Personalizar 8-28
 - Puntos de interrupción de datos y código 8-14
 - Puntos de observación de datos 8-21
 - Salida, entrada y errores de consola 8-11
 - Hilos, pilas de llamada y datos 8-16
 - panel dividido 8-36
 - Monitores de sincronización 8-27
 - Personalizar 8-28
 - previa de cambios en el perfeccionamiento 13-7
 - Puntos de interrupción de datos y código 8-14
 - Puntos de observación de datos 8-21
 - Salida, entrada y errores de consola 8-11

- visualizadores

- de código fuente
 - sincronizar desplazamiento 12-7
 - UML 11-13
 - ayuda inmediata 11-16
 - definición 11-13
 - desplazamiento por diagramas 11-18

- menú contextual 11-17
- perfeccionar código fuente 11-24
- tutorial 22-1
- ver código 11-16
- ver Javadoc 11-16
- ver recopilatorios 2-10
- visualizar
 - variable
 - estática 8-62
 - local 8-62
- Volver al botón de la revisión anterior (vista del histórico) 12-7

W

- writeObject() 10-22

