Tutorial 2 ASIR Java

Jose Luis Pérez Morillas y Juan Antonio Martínez Sánchez

Índice de Contenidos:

1. Paradigma Orientado a Objetos. Fundamentos y origen de JAVA

La programación orientada a objetos es un paradigma de programación que usa objetos en sus interacciones para diseñar aplicaciones y programas informáticos. Esta basada en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamientos.

Su uso se popularizó a principios de la década de 1991 en la actualidad, existe una gran variedad de lenguajes que soportan la orientación a objetos pero nos vamos a centrar en JAVA.

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, lo que quiere decir que el código que es ejcutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, pero vamos a volver atrás en el tiempo para ver sus orígenes.

Java se creó como una herramienta de programación para ser usada en un proyecto denominado Green Project en Sun Microsystems en el año 1991. El nombre viene de las iniciales de sus diseñadores.

El lenguaje Java se creó con cinco con cinco objetivos principales:

1. Debería usar el paradigma de la programación orientada a objetos.

2. Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.

3. Debería incluir por defecto soporte para trabajo en red.

4. Debería diseñarse para ejecutar código en sistemas remotos de forma segura.

5. Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

infografía:

https://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29#Filosof.C3.ADa

https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos#V.C3.A9ase_tambi.C3.A9n

2. Elementos del Lenguaje. Tipos de datos, variables y operadores.

Los elementos del lenguaje java son los siguientes:

- **Tipos de datos:**
  Dentro de los tipos de datos en java, podemos diferenciar dos clases: los tipos primitivos y los tipos objetos.
  1. Los tipos primitivos se caracterizan porque no tienen métodos. Hay 8 tipos primitivos: byte, short, int, long, float, double, char y boolean. Los mas comunes son int, double y boolean.
  2. Los tipos objeto necesita una invocación para ser creados y poseen metodos. En esta clase se encuentran los tipos de la biblioteca de Java (string), los tipos definidos por el programador, los arrays y los tipo wrapper (boolean, integer, byte).

- **Variables:**
  Las variables son un mecanismo del lenguaje java que nos permite guardar datos en memoria para poder hacer uso de ellos dentro de nuestro programa.
  Por ejemplo podemos definir que la variable “numero” sea 2, para despues sumarle 8. El resultado de la variable sería 10:

  ```
  numero = 2
  numero = numero + 8
  ```

- **Operadores:**
  Los operadores en java sirven para generar un resultado en función de los valores tomados.
  Los mas comunes son los siguientes:
  1. Operadores aritméticos:
     + Suma. Los operandos pueden ser enteros o reales
     - Resta. Los operandos pueden ser enteros o reales
     * Multiplicación. Los operandos pueden ser enteros o reales
     / División. Los operandos pueden ser enteros o reales. Si ambos son enteros el resultado es entero. En cualquier otro caso el resultado es real.
     % Resto de la división. Los operandos pueden ser de tipo entero o real.
  2. Operadores relacionales
     < Menor que
     > Mayor que
     <= Menor o igual
     >= Mayor o igual
     != Distinto
     == Igual
  3. Operadores logicos.
     && And
     || Or
     ! Not

infografía:

http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/introduccion/primero.htm

http://puntocomnoesunlenguaje.blogspot.com.es/2012/04/operadores.html

3. Sentencias de Control

Un lenguaje de programación utiliza sentencias de control para hacer que el flujo de ejecución avance o se bifurque en función de los cambios de estado en el programa. Las sentencias de control se clasifican en los grupos: Selección, Iteración y Salto

Selección: Java admite dos sentencias de selección: if y switch. Controlan el flujo de ejecución en función de condiciones conocidas durante el tiempo de ejecución.

If: es una sentencia de bifurcación condicional. Sintaxis:

```
if (condición)
      {sentencia1;}
      else {sentencia2;}
```

Switch: Es una sentencia de bifurcación múltiple. Sintaxis

```
switch(expresión){
case valor1:
sentencias
break;
case valorN:
sentencias
break;
default}
```

Iteración: Las sentencias while, do-while y for permiten crear lo que se conoce como bucles.

```
While (condición){
      cuerpo
      }
```

```
do {
      cuerpo
      }while(condición;
```

```
for (inicialización; condición; iteración){
      cuerpo}
```

Salto: Las sentencias break, continue y return transfieren el flujo de ejecución a otra parte del programa.

Break se usa para finaliza switch y un bucle.

Continue fuerza en un bucle la ejecución de una nueva iteración descartando el procesamiento de la actual iteración.

Return se utiliza para salir explícitamente de un método, es decir hace que el control de flujo se vuelva al método llamante.

Infografía: http://lsc.fie.umich.mx/~pedro/metodoprogra/sentenciasdecontrol.pdf

4. Clase y Objetos en Java. Estructura básica

La estructura de un programa en java esta compuesto por los siguientes elementos

```
Import  java.io.*;       //LIBRERIAS

public class saludo // la clase debe llamarse como el programa

{

    public static void main ( String [ ] ar)    //METODO PPAL

    {


    } // cierro método

} //cierro clase
```

Después del nombre de la clase va el método principal.

- public static void main (String[]ar): donde el interprete JAVA irá a comenzar la ejecución del programa
- Los paréntesis luego de la palabra main indican la declaración de un método. Una clase puede contener uno o más métodos.
- Los programas que se ejecutan de forma independiente y autónoma, deben contener el método ”main()”.
- La palabra reservada ”void” indica que el método main no devuelve nada.
- El método main debe aceptar un array de objetos tipo String. Por acuerdo se referencia como ”args”, O ar.
- Las palabra static es atributo del método; si declaro un método como static, su contenido será el mismo en cualquier objeto que crease e instanciase.

Una clase en java es un paquete o fragmento de código Java que permite crear al menos una instancia (objeto). Por instancia entendemos una entidad existente en la memoria del ordenador que tiene unos atributos y operaciones específicas.

Infografía:

http://aprenderaprogramar.es/index.php?option=com_content&view=article&id=525:concepto-y-definicion-de-clase-en-java-objetos-del-mundo-real-y-abstractos-ejemplos-y-ejercicio-cu00644b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

http://fundamentosdelogica.jimdo.com/instituto-tecnologico-metropolitano-itm-politecnico-jic/laboratorio-logica-de-programacion-java/estructura-b%C3%A1sica-de-java/

5. Sistema de Excepciones en Java. Exception.

Java incluye en el propio lenguaje la gestión de errores. El mejor momento para detectar los errores es durante la compilación. Sin embargo prácticamente sólo los errores de sintaxis son detectados durante ese periodo. El resto de problemas surgen durante la ejecución de los programas.

En java una Exception es un cierto tipo de error o una condición anormal que se ha producido durante la ejecución de un programa. Algunas excepciones son fatalas y provoca que se deba finalizar la ejecución del programa. En este caso conviene terminar ordenadamente y con un mensaje explicando el tipo de error que se ha producido, otra deben dar al usuario la oportunidad de corregir el error.

Un buen programa debe gestionar correctamente todas o la mayor parte de los errores que se producen.

Java ya tiene en su lenguaje una serie de excepciones estándar integradas a las que podremos llamar en caso de necesitarlo.

Para lanzar una Exception:

1. Se crea un objeto Exception de la clase adecuada.

2. Se lanza la excepción con la sentencia throw seguida del objeto Exception creado.

Para Capturar una Exception:

1. Gestionar la excepción con una construcción del tipo try{...} cathc{...}

2. Re-lanzar la excepción hacia un método anterior, declarando su método, también lanza dicha excepción, utilizando para ello la construcción.

Bloques try y catch

En el caso de las excepciones que java obliga a tenerlas en cuenta habrá que utilizar los bloques try, catch y finally. El bloque try está “vigilado” si se produce una situación anormal se lanza la excepción y salta del bloque try y pasa al bloque cath, y el bloque finally es opcional, si se incluye sus sentencias se ejecutan siempre, sea cual sea la excepción que se produzca o si no se produce ninguna.

Ejemplo de un método que debe controla una IOException relacionada con la lectura de ficheros y una MyEception propia:

```
voi metodo1(){
      try {
// Código que puede lanzar las excepciones IOException y MyException
} catch (IOException e1) {// Se ocupa de IOException simplemente dando aviso
System.out.println(e1.getMessage());
} catch (MyException e2) {
// Se ocupa de MyException dando un aviso y finalizando la función
```

```
System.out.println(e2.getMessage()); return;
} finally { // Sentencias que se ejecutarán en cualquier caso
...
}
...
} // Fin del metodo1
```

Infografía: http://www4.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/Java/Java2.pdf

6. Estructuras estáticas: Vectores y Matrices.

En programación, una matriz o vector (llamados en inglés arrays) es una zona de almacenamiento continuo, que contiene una serie de elementos del mismo tipo, los elementos de la matriz. Desde el punto de vista lógico una matriz se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos dimensiones).

Para usar la clase Vector hemos de poner al principo del archivo del código fuente la siguiente sentencia import

import java.util.*;

Cuando creamos un vector u objeto de la clase Vector, podemos especificar su dimensión inicial, y cuanto crecerá si rebasamos dicha dimensión.

Vector vector=new Vector(20, 5);

Tenemos un vector con una dimensión inicial de 20 elementos. Si rebasamos dicha dimensión y guardamos 21 elementos la dimensión del vector crece a 25.

Podemos añadir un elemento a continuación del último elemento del vector, mediante la función miembro addElement.

v.addElement("uno");

Si queremos repreentar una matriz en Java hay que crear un array bidimensional. Por ejemplo para declarar una matriz de 3x3 haríamos lo siguiente:

int matriz[ ] [ ]= new int[3][3];

Infografía:

http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/colecciones/vector.htm

7. Clase Object

Es la raíz de toda jerarquía de clases de Java. Todas las clases de Java derivan de Object.

La clase Object tiene método para cualquier objeto que son heredados por cualquier clase. Entre ellos se pueden citar los siguientes

1. Métodos que pueden ser redefinidos por el programador:

clone() Crea un objeto a partir de otro objeto de la misma clase. El método original heredado de Object lanza una CloneNotSupporteException. Si se desea poder clonar una clase hay que implementar la interface Cloneable y redefinir el método clone().

Equals() Indica si dos objetos son o no iguales. Devuelve True si son iguales, tanto si son referencias al mismo objeto como si son objetos distintos con iguales valores de las variables miembro.

ToStrgin() Devuelve un String que contiene una representación del objeto como cadena de caracteres, por ejemplo para imprimirlo o exportarlo.

2. Métodos que no pueden ser redefinidos:

getClass() Devuelve un objeto de la clase Class, al cual se le puede aplicar métodos para determinar el nombre de la clase, su super-clase, las interfaces implementadas, etc... Se puede crear un objeto de la misma clase que otro sin saber de que clase es.

Notify(), notifyAll() y wait() Son métodos relacionados con las threads.

Infografía: http://www4.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/Java/Java2.pdf

8. Clase System

La clase system se utiliza para acceder a datos que contiene el sistema, como por ejemplo los dispositivos de entrada y salida conectados o la hora y la fecha del sistema.

Los metodos que contiene esta clase son los siguientes:

static void arraycopy(Object,int,Object,int,int)

static long currentTimeMillis()

static void exit(int)

static void gc()

static Properties getProperties()

static String getPropertie(String)

static String getPropertie(String,String)

static SecurityManager getSecurityManager()

static native int identityHashCode(Object)

static void load(String)

static void loadLibrary(String)

static void runFinalization()

static void runFinalizersOnExit(boolean)

static void setErr(PrintStream)

static void setIn(InputStream)

static void setOut(PrintStream)

static void setProperties(Properties)

static void setSecurityManager(SecurityManager)

Los dispositivos de entrada/salida probablemente sean uno de los aspectos más utilizado de esta clase.

La clase System proporciona automáticamente cuando comienza la ejecución de un programa, un stream para leer del dispositivo estándar de entrada (el teclado, generalmente) un stream para presentar información en el dispositivo estándar de salida (normalmente, la pantalla) y otro stream donde presentar mensajes de error, que es el dispositivo estándar de error (normalmente, la pantalla).

Los tres streams de entrada/salida están controlados por esta clase y se referencian como:

System.in → entrada estándar

System.out → salida estándar

System.err → salida de error estándar

Infografía:

http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/archivos/teclado.htm

http://dis.um.es/~lopezquesada/documentos/IES_1415/IAW/curso/UT3/ActividadesAlumnos/java7/paginas/pag8.html

9. Herencia en Java

La herencia permite que se pueden definir nuevas clases basadas en clases existentes, lo cual facilita re-utilizar código previamente desarrollado. Si una clase deriva de otra (extends) hereda todas sus variables y métodos. La clase derivada puede añadir nuevas variables y métodos y/o redefinir las variables y métodos heredados. En Java, a diferencia de otros lenguajes orientados a objetos, una clase sólo puede derivar de una única clase, con lo cual no es posible realizar herencia múltiple en base a clases. Sin embargo es posible "simular" la herencia múltiple en base a las interfaces.

Ejemplo de herencia en java:

```
//Código de la clase Persona
public class Persona {
      private String nombre;
      private String apellidos;
      private Int edad;
//Constructor
public Persona (String nombre, String apellidos, int edad) {
      this.nombre = nombre;
      this.apellidos = apellidos;
      this.edad = edad;
//Métodos
public String getNombre() {return nombre;}
public String getApellidos () {return apellidos; }
public Int getEdad () {return edad;}
}//Cierre de la clase
```

```
//Código de la clase profesor de la clase Persona
public class Profesor extends Persona {
      //Campos especificos de la sublcase.
      Private String IdProfesor;
//Constructor de la sublcase: Incluimos como parámetros al menos los del constructor del padre
public Profesor (String nombre, String apellidos, Int edad){
      super(nombre, apellidos, edad);
      Idprofesor = "Unknown"; } //Cierre del contructor
// Metodos especificos de la sublcase
```

```
public void setIdProsor (String IdProfesor) {this.IdProfesor = IdProfesor;}

public String getIdProfesor () {return IdProfesor;}

public void mostrarNombreApellidosYCarnet(){

      System.out.println ("Profesor de nombre: " + getNombre() + " "
      +getApellidos() +" con Id de profesor: " + getIdProfesor() ); }

}//Cierre de la clase
```

```
// Códgp de test

public class TestHerencia1 {

      public static void main (String[] Args){

            Profesor profesor1 = new Profesor (“Juan”. “Bla Bla”. 42);

            profesor1.setIdProfesor(“Prof 22-387-11”);

            profesor1.mostrarNombreApellidosYCarnet();}

//Cierre de la clase
```

Infografía: http://www4.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/Java/Java2.pdf

http://aprenderaprogramar.es/index.php?option=com_content&view=article&id=653:ejemplo-de-herencia-en-java-uso-de-palabras-clave-extends-y-super-constructores-con-herencia-cu00686b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

10. Gestión de cadenas en Java

Una cadena es una secuencia de caracteres. Las cadenas son una parte fundamental de la mayoría de los programas, así pues Java tiene varias características incorporadas que facilitan la manipulación de cadenas. Java tiene una clase incorporada en el paquete java.lang que encapsula las estructuras de datos de una cadena.

El paquete java.lang contiene dos clases de cadenas: String y StringBuffer. La clase String se utiliza cuando se trabaja con cadenas que no pueden cambiar. Por otro lado, StringBuffer, se utiliza cuando se quiere manipular el contenido de una cadena. El entorno de desarrollo Java proporciona dos clases para manipular y almacenar datos del tipo carácter: String, para cadenas constantes, y StringBuffer, para cadenas que pueden cambiar.

Muchos Strings se crean a partir de cadenas literales. Cuando el compilador encuentra una serie de caracteres entre comillas (" y "), crea un objeto String cuyo valor es el propio texto. El esquema general es el siguiente: String nombre="cadena"; Cuando el compilador encuentra la siguente cadena, crea un objeto String cuyo valor es Hola Mundo.

"Hola Mundo"

o también

String s = "Hola Mundo"

También se pueden crear objetos String como se haría con cualquier otro objeto Java: utilizando new.

String s = new String("Hola Mundo.");

El constructor anterior es equivalente pero es mucho más eficiente el primer método ya que, el segundo método crea dos objetos String en vez de sólo uno.

Infografía:

http://www.oocities.org/collegepark/quad/8901/cap06.htm

11. Clase String

La clase String esta orientadas a manejar cadena de caracteres constantes, es decir, que no pueden cambiar, pertenece al package java.lang, y por lo tanto no hay que importarla.

Para concatenar objetos del tipo String se usa el operador de concatenación (+).

Los métodos de String se pueden utilizar directamente sobre literales, es decir cadenas entre comillas por ejemplo “Hola”.lenght(), java crea siempre un objeto String al encontrar una cadena entre comillas, para crear objeto con la clase String usamos:

String cadena1 = “Hola”; // El sistema más efizac de crear String

String cadena2 = new String (“Hola”); // También para crear con un constructor

El primero de los métodos expuestos es el más eficiente, por que como al encontrar un texto entre comillas se crea automáticamente, en la práctica al usar el “new” se llama dos veces al constructor.

Algunos métodos de la clase string:

Métodos de String	Función que realizan
String(...)	Constructores para crear Strings a partir de arrays de bytes o de caracteres (ver documentación on-line)
String(String str) y String(StringBuffer sb)	Costructores a partir de un objeto String o StringBuffer
charAt(int)	Devuelve el carácter en la posición especificada
getChars(int, int, char[], int)	Copia los caracteres indicados en la posición indicada de un array de caracteres
indexOf(String, [int])	Devuelve la posición en la que aparece por primera vez un String en otro String, a partir de una posición dada (opcional)
lastIndexOf(String, [int])	Devuelve la última vez que un String aparece en otro empezando en una posición y hacia el principio
length()	Devuelve el número de caracteres de la cadena
replace(char, char)	Sustituye un carácter por otro en un String
startsWith(String)	Indica si un String comienza con otro String o no
substring(int, int)	Devuelve un String extraído de otro
toLowerCase()	Convierte en minúsculas (puede tener en cuenta el locale)
toUpperCase()	Convierte en mayúsculas (puede tener en cuenta el locale)
trim()	Elimina los espacios en blanco al comienzo y final de la cadena
valueOf()	Devuelve la representación como String de sus argumento. Admite Object, arrays de caracteres y los tipos primitivos

Un punto a tener en cuenta es que hay métodos, tales como System.out.printIn() que exigen que su argumento sea un objeto de la clase String y si no lo es habrá que utilizar un método que lo convierta a String.

Infografía: http://www4.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/Java/Java2.pdf

12. Clase StringBuffer

La clase StringBuffer permite modificar el String en lugar de crear uno nuevo en cada paso intermedio.

Contiene metodos similares e igual nombre que String pero son independientes.

Los constructores son StringBuffer() y StringBuffer (String), y los metodos principales son los siguientes:

-setCharAt(int, char)

-StringBuffer insert (int offset, valor)

-StringBuffer append (valor) donde valor puede ser:

-int, char, boolean, float, double, long

-Object

-String

-char [] str

Aqui podemos ver un ejemplo de uso de StringBuffer:

public static String escribirRaiz (int i) { StringBuffer buf = new StringBuffer();

buf.append(“sqrt(“).append(i).append(‘)’); buf.append(“ = “).append(Math.sqrt(i)); return buf.toString(); }

Infografía:

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/La_clase_StringBuffer

13. Clase Math

La clase java.lang.Math deriva de Object. La clase Math proporciona métodos para realizar las operaciones matemáticas más habituales. Proporciona además las constantes E y PI.

Algunos de los métodos más usados:

Métodos	Significado
abs()	Valor absoluto
acos()	Arcocoseno
asin()	Arcoseno
atan()	Arcotangente entre -PI/2 y PI/2
atan2(,)	Arcotangente entre -PI y PI
ceil()	Entero más cercano en dirección a infinito
floor()	Entero más cercano en dirección a -infinito
round()	Entero más cercano al argumento
rint(double)	Devuelve el entero más próximo
IEEEremainder(double , double)	Calcula el resto de la división
cos(double)	Calcula el coseno
sin(double)	Calcula el seno
tan(double)	Calcula la tangente
exp()	Calcula la función exponencial
log()	Calcula el logaritmo natural (base e)
max(,)	Máximo de dos argumentos
min(,)	Mínimo de dos argumentos
random()	Número aleatorio entre 0.0 y 1.0
power(,)	Devuelve el primer argumento elevado al segundo
sqrt()	Devuelve la raíz cuadrada
toDegrees(double)	Pasa de radianes a grados (Java 2)
toRadians()	Pasa de grados a radianes (Java 2)

Infografía: http://www4.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/Java/Java2.pdf

14. Clase Arrays

Un array es un medio de guardar un conjunto de objetos de la misma clase. Se accede a cada elemento individual del array mediante un número entero denominado índice. 0 es el índice del primer elemento y n-1 es el índice del último elemento, siendo n, la dimensión del array.

Para crear un array de tres objetos de la clase Rectangulo se escribe

-Declarar

```
Rectangulo[] rectangulos;
```

-Crear el array

```
rectangulos=new Rectangulo[3];
```

-Inicializar los elementos del array

```
rectangulos[0]=new Rectangulo(10, 20, 30, 40);
rectangulos[1]=new Rectangulo(30, 40);
rectangulos[2]=new Rectangulo(50, 80);
```

-O bien, en una sola línea

```
Rectangulo[] rectangulos={new Rectangulo(10, 20, 30, 40),
        new Rectangulo(30, 40), new Rectangulo(50, 80)};
```

-Usar el array

Para calcular y mostrar el área de los rectángulos escribimos

```
for(int i=0; i<rectangulos.length; i++){
        System.out.println(rectangulos[i].calcularArea());
}
```

Infografía:

http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/arays.htm

15. Clase para tipos de primitivos o simples:String, Integer, Double.... java.lang.*

Java trabaja con dos tipos de entidades: Primitivo y objetos verdaderos. Java no utiliza los objetos para sustituir a la mayoría de los tipos primitivos (los números, boléanos y caracteres), sobre todo para mayor eficiencia. La manipulación de los tipos primitivos independientes es más eficiente. Se puede agrupar los tipos primitivos, por ejemplo podemos crear una clase cuya finalidad sea englobar un entero de esta forma podremos trabajar con nuestro tipos primitivos de siempre como si fueran objetos.

El paquete java.lan contiene las siguientes clases envolventes:

Integer → Entero de 4 bytes con rango de 2* 10^9

Byte → Entero de 1 byte con rango de -128 a 127

Float → Decimal simple de 4 bytes

Character → Carácter simple de 2 bytes

Void → Vacio

Long → Entero de 8 bytes

Short → Entero de 2 bytes con rango de -32768 a 32767

Double → Decimal doble de 8 bytes

Boolean → Valor true o false 1 byte

Las clases envolventes son un complemento de los tipos primitivos. Cada tipo primitivo tiene su correspondiente clase envolvente en el paquete java.lang:

byte → Byte

int → Integer

float → Float

boolean → Boolean

short → Short

long → Long

double → Double

char → Character

infografía: http://profesores.fi-b.unam.mx/laloea/pamn/htm/docs/ClasesEnvolventesYTiposPrimitivos.pdf

http://aprenderaprogramar.es/index.php?option=com_content&view=article&id=419:tipos-de-datos-java-tipos-primitivos-int-boolean-y-objeto-string-array-o-arreglo-variables-cu00621b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

16. Introducción a la jerarquía de colecciones en JAVA: Clase ArrayList y Clase Vector.

Un objeto ArrayList será una lista redimensionable en la que tendremos disponibles los métodos más habituales para operar con listas.

Como métodos para operar con listas podemos señalar: añadir un objeto en una posición determinada, añadir un objeto al final de la lista, recuperar un objeto situado en determinada posición, etc. Los objetos de un ArrayList tienen un orden, que es el orden en el que se insertaron en la lista.

Un aspecto a tener muy presente: hablamos de colecciones de objetos. Por tanto, un ArrayList no puede ser una lista de enteros como tipo primitivo (int) pero sí de objetos Integer. La sintaxis que emplearemos con ArrayList es la siguiente:

-**(Declaración del objeto ArrayList)** : private ArrayList<TipoDeObjetosEnLaColección> NombreDelArrayList;

-**(Creación de un objeto):** NombreDeObjeto = new ArrayList<TipodeObjetosEnLaColección>();

-**(Uso del método reemplazar objeto existente):** NombreDelArrayList.set (int índice, E elemento);

-**(Uso del método añadir al final):** NombreDelArrayList.add (objeto_a_añadir);

-**(Uso del método obtener el número de objetos en la lista):** NombreDelArrayList.size();

-**(Uso del método extraer un objeto de cierta posición):** NombreDelArrayList.get (posición);

Infografía:

http://aprenderaprogramar.es/index.php?option=com_content&view=article&id=631:clase-arraylist-del-api-java-metodos-add-size-etc-concepto-de-clase-generica-o-parametrizada-cu00665b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

17. Introducción al tratamiento de ficheros en Java. E/S java.io.*

El package java.io contiene las clases necesarias para la comunicación con el exterior. Dentro de este package existe dos familias de jerarquías distintas para la entrada/salida de datos.

La diferencia principal consiste en que una opera con bytes y la otra con caracteres (unicode de dos bytes), en general para el mismo fin hay dos clases que manejan bytes, una de entrada y otra de salida y otras dos que manejan los caracteres.

Desde Java 1.0 la entrada y salida de datos del programa se podía hacer con clases derivadas de InputStream y OutputStream. Estas clases tienen los métodos básicos read() y write() que manejan bytes y que no se suelen utilizar directamente.

En java 1.1 aparecieron dos nueva familias de clases derivadas de Reader y Writer, que manejan caracteres en vez de bytes. Estas clases resultan más prácticas para las aplicaciones en las que se maneja texto.

Se recomienda utilizar siempre que sea posible las clases Reader y Writer, dejando las de java 1.0 para cuando sean imprescindibles. Algunas tareas como la serialización y la compresión necesitan las clases InputStream y OutputStream.

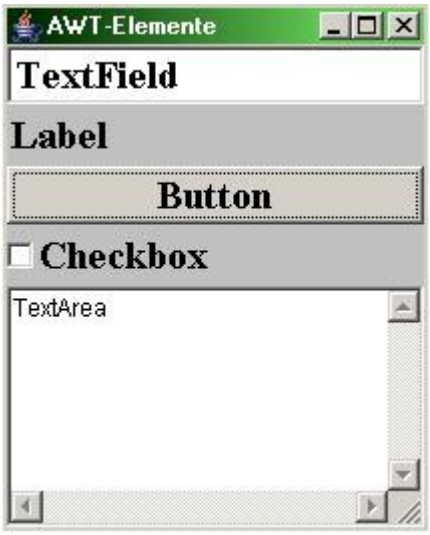
Las clases de java.io siguen una nomenclatura sistemática que permite deducir su función a partir de las palabras que componen el nombre:

Palabra	Significado
InputStream, OutputStream	Lectura/Escritura de bytes
Reader, Writer	Lectura/Escritura de caracteres
File	Archivos
String, CharArray, ByteArray, StringBuffer	Memoria (a través del tipo primitivo indicado)
Piped	Tubo de datos
Buffered	Buffer
Filter	Filtro
Data	Intercambio de datos en formato propio de Java
Object	Persistencia de objetos
Print	Imprimir

Infografía: http://www4.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/Java/Java2.pdf

18. Introducción a la programación de escritorio. AWT/SWING.

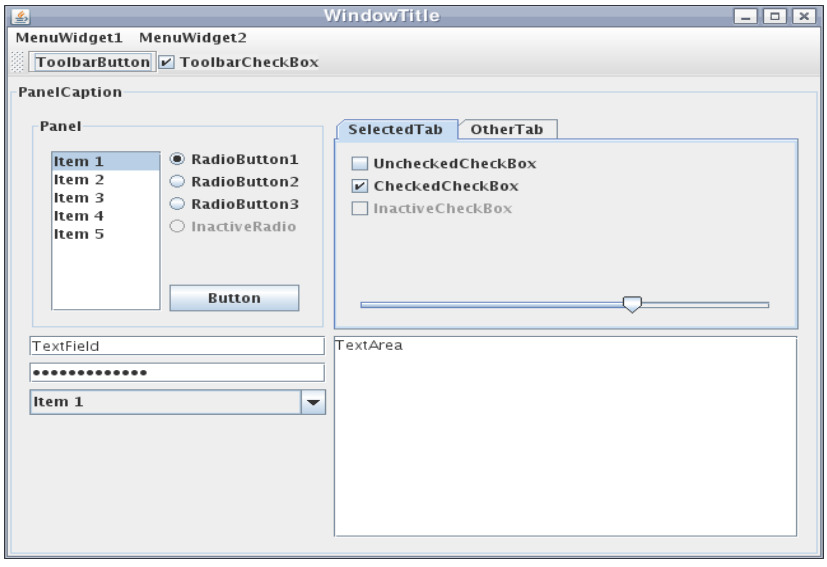
La **Abstract Window Toolkit** (AWT, en español Kit de Herramientas de Ventana Abstracta) es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java.



Algunos desarrolladores de aplicaciones prefieren este modelo porque suministra un alto grado de fidelidad al kit de herramientas nativo subyacente y mejor integración con las aplicaciones nativas. En otras palabras, un programa GUI escrito usando AWT parece como una aplicación nativa Microsoft Windows cuando se ejecuta en Windows, pero el mismo programa parece una aplicación nativa Apple Macintosh cuando se ejecuta en un Mac, etc. Sin embargo, algunos desarrolladores de aplicaciones desprecian este modelo porque prefieren que sus aplicaciones se vean exactamente igual en todas las plataformas.

Swing es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

Es un framework MVC para desarrollar interfaces gráficas para Java con independencia de la plataforma.



Las ventajas que nos ofrece Swing son::

-El diseño en Java puro posee menos limitaciones de plataforma.

-El desarrollo de componentes Swing es más activo.

-Los componentes de Swing soportan más características.

Infografía:

https://es.wikipedia.org/wiki/Swing_%28biblioteca_gr%C3%A1fica%29

https://es.wikipedia.org/wiki/Abstract_Window_Toolkit

19. Herramientas CASE para el desarrollo de artefactos SW en Java.

Las herramientas CASE (programas que te asisten a la hora de programar) más usadas para java son las siguientes:

NetBeans (http://netbeans.org/)

Herramienta muy buena con características como desarrollo intuitivo gratis y opensource. Principalmente para desarrollo de escritorio Web Mobile y enterprise con compatibilidad con java C/C++ Ruby PHP javascript, tiene algunas mejoras con UML aunque no es el más optimo tiene algo interesante a al hora de crear programas para android.

Microsoft Visio (http://office.microsoft.com/en-us/visio/)

Herramienta de diagramación avanzada con gran variedad de plantillas que permiten simplificar las tareas complejas con elementos visuales dinámicos basados en datos, UML bases de datos Arquitectura ect con Share point con más facilidad sin generar codigo pero bastante atractiva para hacer distintos diagramas

Eclipse (http://www.omondo.com/)

Eclipse dispone de un editor de texto y la compilación es en tiempo real tiene pruebas unitarias con JUNIT control de versiones con CVS y es código abierto lo cual se puede montar herramientas de desarrollo para cualquier lenguaje mediante la implementación de plugins adecuados para la realización de diagramas en otros códigos.

Inforgrafia: http://javax-peru.blogspot.com.es/2008/11/herramientas-case.html y otras búsquedas por google.