Tutorial2ASIRJava.WEB

Índice de Contenidos:

Manuel Martínez Molina y Jose Gabriel Esparza Sánchez

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

Paradigma Orientado a Objetos.

Un paradigma de programación es una propuesta tecnológica adoptada por una comunidad de programadores cuyo núcleo central es incuestionable en cuanto a que unívocamente trata de resolver uno o varios problemas claramente delimitados Un paradigma de programación está delimitado en el tiempo en cuanto a aceptación y uso ya que nuevos paradigmas aportan nuevas o mejores soluciones que la sustituyen parcial o totalmente. Probablemente el paradigma de programación que actualmente es el más usado a todos los niveles es la orientación a objeto. El núcleo central de este paradigma es la unión de datos y procesamiento en una entidad llamada "objeto", relacionable a su vez con otras entidades "objeto". Con la orientación a objetos y características como el encapsulado, polimorfismo o la herencia se permitió un avance significativo en el desarrollo de software a cualquier escala de producción.

Algunos de los paradigmas de programación más comunes son:

• Imperativo o por procedimientos: es considerado el más común y está representado por C o BASIC.

• Orientado a objetos:fue introducido por Smalltalk, un lenguaje completamente orientado a objetos.

• Funcional: está representado por Scheme o Haskell. Este es un caso del paradigma declarativo.

• Declarativo: la programación funcional, la programación lógica, o la combinación lógico-funcional.

• Lógico: está representado por Prolog. Este es otro caso del paradigma declarativo.

origen de JAVA.

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases 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 ejecutado 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.

http://dis.um.es/~lopezquesada/documentos/IES_1314/IAW/curso/UT3/java/java7/html/actividad1.html

http://dis.um.es/~lopezquesada/documentos/IES_1415/IAW/curso/UT3/ActividadesAlumnos/java7/paginas/pag1.html

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

Elemento de lenguajes

Identificadores

Comentarios

Sentencias

Bloques de código

Expresiones

Variables

Los tipos básicos de datos

Las cadenas de caractares o strings

Palabras reservadas

Variables

Una variable es un nombre que se asocia con una porción de la memoria del ordenador, en la que se guarda el valor asignado a dicha variable. Hay varios tipos de variables que requieren distintas cantidades de memoria para guardar datos.

Todas las variables han de declararse antes de usarlas, la declaración consiste en una sentencia en la que figura el tipo de dato y el nombre que asignamos a la variable. Una vez declarada se le podrá asignar valores.

Java tiene tres tipos de variables:

- de instancia
- de clase
- locales

En el siguiente ejemplo, *PI* es una variable de clase y *radio* es una variable de instancia. *PI* guarda el mismo valor para todos los objetos de la clase *Circulo*, pero el radio de cada círculo puede ser diferente

```
class Circulo{
      static final double PI=3.1416;
      double radio;
//...
}
```

Las variables locales se utilizan dentro de las funciones miembro o métodos. En el siguiente ejemplo *area* es una variable local a la función *calcularArea* en la que se guarda el valor del área de un objeto de la clase *Circulo*. Una variable local existe desde el momento de su definición hasta el final del bloque en el que se encuentra.

```
class Circulo{
```

```
//...
	double calcularArea(){
		double area=PI*radio*radio;
		return area;
	}
}
```

En el lenguaje Java, las variables locales se declaran en el momento en el que son necesarias. Es una buena costumbre inicializar las variables en el momento en el que son declaradas. Veamos algunos ejemplos de declaración de algunas variables

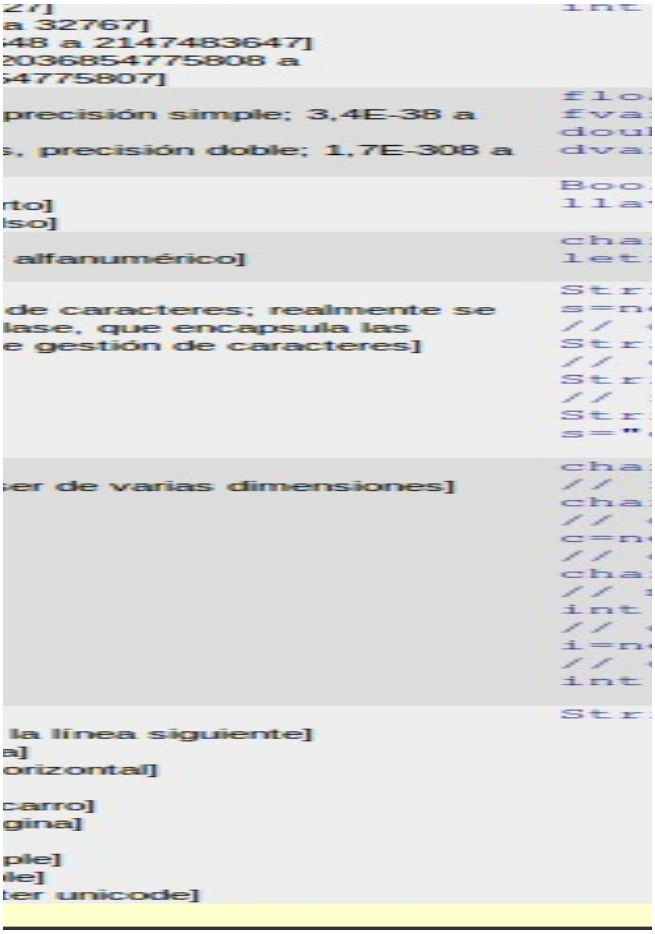
```
int x=0;
String nombre="Angel";
double a=3.5, b=0.0, c=-2.4;
boolean bNuevo=true;
int[] datos;
```

Delante del nombre de cada variable se ha de especificar el tipo de variable que hemos destacado en letra negrita. Las variables pueden ser

- Un tipo de dato primitivo
- El nombre de una clase
- Un array

Tipos de Datos

Tipo	**Descripcion**
boolean	Tiene dos valores **true** o **false**.
char	Caracteres Unicode de 16 bits. Los caracteres alfa-numéricos son los mismos que los ASCII con el bit alto puesto a 0. El intervalo de valores va desde 0 hasta 65535 (valores de 16-bits sin signo).
byte	Tamaño 8 bits. El intervalo de valores va desde -2^7 hasta 2^7 -1 (-128 a 127)
short	Tamaño 16 bits. El intervalo de valores va desde -2^{15} hasta 2^{15}-1 (-32768 a 32767)
int	Tamaño 32 bits. El intervalo de valores va desde -2^{31} hasta 2^{31}-1 (-2147483648 a 2147483647)
long	Tamaño 64 bits. El intervalo de valores va desde -2^{63} hasta 2^{63}-1 (-9223372036854775808 a 9223372036854775807)
float	Tamaño 32 bits. Números en coma flotante de simple precisión. Estándar IEEE 754-1985 (de 1.40239846e–45f a 3.40282347e+38f)
double	Tamaño 64 bits. Números en coma flotante de doble precisión. Estándar IEEE 754-1985. (de 4.9406564584124654e–324d a 1.7976931348623157e+308d.)



http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/introduccion/primero.htm#Tipos%20de%20datos%20primitivos

Operdores.

Los operadores aritméticos en java son:

+ Suma. Los operandos pueden ser enteros o reales

- Resta. Los operandos pueden ser enteros o reales

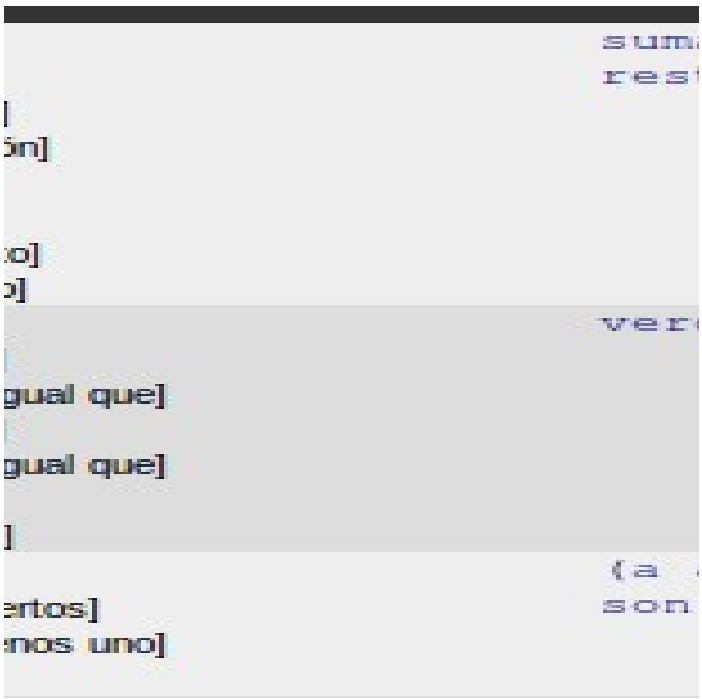
* Multiplicación. Los operandos pueden ser enteros o reales

/ 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.

Ejemplo de operaciones aritméticas:

```
int a = 10, b = 3;
double v1 = 12.5, v2 = 2.0;
char c1='P', c2='T';
```


sum
res
ón]
o]
o]
ver
gual que]
gual que]
]
(a
son
ertos]
nos uno]

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.
-Salto

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

IF

La sentencia

es una sentencia de bifurcación condicional, para dirigir el flujo de ejecución hacia dos caminos diferentes. Sintaxis:

if (condición)
sentencia1;

else sentencia2;
La condición es cualquier expresión que devuelva un valor booleano y la clausula else es opcional.

Sentencias de selección
if anidados
Un if anidado es una sentencia if que esta dentro de otro if o else..

```
if( i == 10){
if( j<20) a = b ;
if( k>100) c = d ;
else a = c ;}
elsea = d ;
```

SWITCH

La sentencia **switch** es una sentencia de bifurcación múltiple, con el siguiente formato general:

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

case valor2:
sentencias
break;

case valorN:
sentencias
break;
default:
}
```

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

4. Clase y Objetos en java. Estructura básica.

Estructura básica.

Toda aplicación de consola en Java contiene una clase principal con un método (o función) main, al igual que en C++, la función main es lo primero que se ejecuta cuando se inicia la aplicación desde la línea de comandos. De esta forma podemos deducir que cualquier llamado, declaración o método que no sea llamado de manera directa o indirecta desde el método main nunca se ejecutará. En Java el método main recibe como argumento un arreglo de String. Este arreglo contiene los argumentos enviados por la línea de comandos para la invocación del programa.

La estructura básica de un programa desarrollado usando Java es la siguiente:

```
1  public class nombre_clase
2  {
3      public static void main (String args[])
4      {
5         bloque de sentencias;
6      }
7  }
```

CLASE

El elemento básico de la programación orientada a objetos en Java es la clase. Una clase define la forma y comportamiento de un objeto.

Para crear una clase sólo se necesita un archivo fuente que contenga la palabra clave reservada class seguida de un identificador legal y un bloque delimitado por dos llaves para el cuerpo de la clase.

class MiPunto {}

Un archivo de Java debe tener el mismo nombre que la clase que contiene, y se les suele asignar la extensión ".java". Por ejemplo la clase MiPunto se guardaría en un fichero que se llamase MiPunto.java. Hay que tener presente que en Java se diferencia entre mayúsculas y minúsculas; el nombre de la clase y el de archivo fuente han de ser exactamente iguales.

Aunque la clase MiPunto es sintácticamente correcta, es lo que se viene a llamar una clase vacía, es decir, una clase que no hace nada. Las clases típicas de Java incluirán variables y métodos de instancia. Los programas en Java completos constarán por lo general de varias clases de Java en distintos archivos fuente.

Una clase es una plantilla para un objeto. Por lo tanto define la estructura de un objeto y su interfaz funcional, en forma de métodos. Cuando se ejecuta un programa en Java, el sistema utiliza definiciones de clase para crear instancias de las clases, que son los objetos reales. Los términos instancia y objeto se utilizan de manera indistinta. La forma general de una definición de clase es:

```
class Nombre_De_Clase {

  tipo_de_variable nombre_de_atributo1;

  tipo_de_variable nombre_de_atributo2;

  // . . .

  tipo_devuelto nombre_de_método1( lista_de_parámetros ) {

    cuerpo_del_método1;

  }

  tipo_devuelto nombre_de_método2( lista_de_parámetros ) {

    cuerpo_del_método2;

  }

  // . . .
```

Objetos

Los tipos simples de Java describían el tamaño y los valores de las variables. Cada vez que se crea una clase se añade otro tipo de dato que se puede utilizar igual que uno de los tipos simples. Por ello al declarar una nueva variable, se puede utilizar un nombre de clase como tipo. A estas variables se las conoce como referencias a objeto.

Todas las referencias a objeto son compatibles también con las instancias de subclases de su tipo. Del mismo modo que es correcto asignar un byte a una variable declarada como int, se puede declarar que una variable es del tipo MiClase y guardar una referencia a una instancia de este tipo de clase:

MiPunto p;

esta es una declaración de una variable p que es una referencia a un objeto de la clase MiPunto, de momento con un valor por defecto de null. La referencia null es una referencia a un objeto de la clase Object, y se podrá convertir a una referencia a cualquier otro objeto porque todos los objetos son hijos de la clase Object.

```
MiPunto( ) {

  inicia( -1, -1 );

}
```

http://dis.um.es/~lopezquesada/documentos/IES_1213/IAW/curso/UT3/ActividadesAlumnos/1/articles.html

5. Sistema de Excepciones en Java. Exception.

Los programas escritos en Java también pueden lanzar excepciones explícitamente mediante la instrucción throw, lo que facilita la devolución de un “código de error” al método que invocó el método que causó el error. Veamos un ejemplo.

```
public class Excepciones1 {
        public static void main(String[] arg) {
                metodo();
}
static void metodo() {
int divisor = 0;
int resultado = 100/divisor;
System.out.println(“Resultado: ” + resultado); System.out.println(“Una Suma:” + (3+4));
        }
}
```

Si ejecutamos ese programa, se generara una excepción automáticamente y se nos indicara por pantalla.

Generación de excepciones en Java

Es preferible tener declaradas todas las posibles excepciones que se puedan generar en dicho método, para lo que se utilizara la sentencia throws de la declaración de métodos.

Para poder lanzar una excepción es necesario crear un objeto de tipo Exception o alguna de sus subclases como ArithmeticException y lanzarlo mediante la instrucción throw cmo se muestra en este ejemplo:

```
class LanzaExcepcion {
public static void main(String argumentos[]) throws ArithmeticException {
int i=1, j=2;
if (i/j< 1)
throw new ArithmeticException();
else
System.out.println(i/j);
}
}
```

http://dis.um.es/~lopezquesada/documentos/IES_1415/IAW/curso/UT3/ActividadesAlumnos/java7/paginas/pag5.html

6. Estructuras estáticas: Vectores y Matrices.

Crear un vector

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.

Añadir elementos al vector

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

v.addElement("uno");

En la siguiente porción de código, se crea un vector con una capacidad inicial de 10 elementos, valor por defecto, y se le añaden o insertan objetos de la clase *String*.

Podemos eliminar todos los elementos de un vector, llamando a la función miembro *removeAllElements*. O bien, podemos eliminar un elemento concreto

v.removeElement("”nombredelelemento”");

Crear un Matriz

Hay que recordar que los elementos empiezan a numerarse por 0. Así la esquina superior izquierda de la matriz será el elemento [0][0] y la esquina inferior derecha será el [2][2].

Si queremos representar una matriz en Java hay que crear un array bidimensional. Por ejemplo para declarar una matriz de 2x2 haríamos lo siguiente:

int matriz[ ] [ ]= new int[2][2];

http://dis.um.es/~lopezquesada/documentos/IES_1314/IAW/curso/UT3/java/java3/tutorial5.html

7. Clase Object

La clase **Object**, es la clase raíz de todo el árbol de la jerarquía de clases Java, y proporciona un cierto número de métodos de utilidad general que pueden utilizar todos los objetos. La lista completa se puede ver en la documentación del API de Java, aquí solamente se tratarán algunos de ellos; por ejemplo, **Object**proporciona:

- Un método por el que un objeto se puede comparar con otro objeto
- Un método para convertir un objeto a una cadena
- Un método para esperar a que ocurra una determinada condición
- Un método para notificar a otros objetos que una condición ha cambiado
- Un método para devolver la clase de un objeto

El método *equals()*

- public boolean **equals**(Object obj);

El método *getClass()*

- public final native Class **getClass**();

El método *toString()*

- public String **toString**();

Otros Métodos

protected void **finalize**();
que se tratará en el apartado de *finalizadores*. O también, los métodos que se utilizan en la programación de threads para hacer que varios threads se sincronicen, como son

```
public final void wait();
public final native void wait( long timeout );
public final native void notify();
public final native void notifyAll();
```

http://dis.um.es/~bmoros/Tutorial/parte5/cap5-10.html

8. Clase System.

Al contrario que la mayoría de las clases, no se debe ejemplarizar la clase System para utilizarla. Para ser más precisos, no se puede ejemplarizar-- es una clase final y todos sus constructores son privados.
Todas las variables y métodos de la clase System son métodos y variables de clase -- están declaradas como **static**. Para una completa explicación sobre las variables y métodos de clase y en qué se diferencian de las variables y métodos de ejemplar, puede referirse a Miembros del Ejemplar y de la Clase.

Para utilizar una variable de clase, se usa directamente desde el nombre de la clase utilizando la notación de punto ('.') de Java. Por ejemplo, para referirse a la variables **out** de la clase System, se añade el nombre de la variable al nombre de la clase separados por un punto. Así:

System.out
Se puede llamar a los métodos de clase de una forma similar. Por ejemplo, para llamar al método **getProperty()** de la clase System se añade el nombre del método al nombre de la clase separados por un punto:
System.getProperty(*argument*);
El siguente programa Java utiliza dos veces la clase System, primero para obtener el nombre del usuario actual y luego para mostrarlo.

```
class UserNameTest {
   public static void main(String[] args) {
      String name;
      name = System.getProperty("user.name");
      System.out.println(name);
   }
}
```

http://www.binarykode.com/bdescargas/Manuales%20y%20Documentos/JAVA/Interfaces%20de%20Usuario/Tutorial%20JAVA%20avanzado%20(I)/recursos/using.html

9. Herencia en Java.

Pero además de esta técnica de composición es posible pensar en casos en los que una clase es una extensión de otra. Es decir una clase es como otra y además tiene algún tipo de característica propia que la distingue. Por ejemplo podríamos pensar en la clase Empleado y definirla como:

```
class Empleado {
   String nombre;
   int numEmpleado , sueldo;

    static private int contador = 0;

   Empleado(String nombre, int sueldo) {
            this.nombre = nombre;
            this.sueldo = sueldo;
     numEmpleado = ++contador;
   }

    public void aumentarSueldo(int porcentaje) {
          sueldo += (int)(sueldo * aumento / 100);
   }

   public String toString() {
         return "Num. empleado " + numEmpleado + " Nombre: " + nombre +
     " Sueldo: " + sueldo;
   }
}
```

En el ejemplo el Empleado se caracteriza por un nombre (String) y por un número de empleado y sueldo (enteros). La clase define un constructor que asigna los valores de nombre y sueldo y calcula el número de empleado a partir de un contador (variable estática que siempre irá aumentando), y dos métodos, uno para calcular el nuevo sueldo cuando se produce un aumento de sueldo (método aumentarSueldo) y un segundo que devuelve una representación de los datos del empleado en un String.(método toString).

Con esta representación podemos pensar en otra clase que reuna todas las características de Empleado y añada alguna propia. Por ejemplo, la clase Ejecutivo. A los objetos de esta clase se les podría aplicar todos los datos y métodos de la clase Empleado y añadir algunos, como por ejemplo el hecho de que un Ejecutivo tiene un presupuesto.

Así diriamos que la clase Ejecutivo extiende o hereda la clase Empleado. Esto en Java se hace con la clausula **extends** que se incorpora en la definición de la clase, de la siguiente forma:

```
class Ejecutivo extends Empleado {
   int presupuesto;
   void asignarPresupuesto(int p) {
         presupuesto = p;
   }
}
```

Con esta definición un Ejecutivo es un Empleado que además tiene algún rasgo distintivo propio. El cuerpo de la clase Ejecutivo incorpora sólo los miembros que son específicos de esta clase, pero implícitamente tiene todo lo que tiene la clase Empleado.

A Empleado se le llama clase base o superclase y a Ejecutivo clase derivada o subclase.

Los objetos de las clases derivadas se crean igual que los de la clase base y pueden acceder tanto sus datos y métodos como a los de la clase base. Por ejemplo:

```
Ejecutivo jefe = new Ejecutivo( "Armando Mucho", 1000);
jefe.asignarPresupuesto(1500);
jefe.aumentarSueldo(5);
```

Nota: La discusión acerca de los constructores la veremos un poco más adelante.

Atención!: Un Ejecutivo ES un Empleado, pero lo contrario no es cierto. Si escribimos:

```
Empleado curri = new Empleado ( "Esteban Comex Plota" , 100) ;
curri.asignarPresupuesto(5000);  // error
```

se producirá un error de compilación pues en la clase Empleado no existe ningún método llamado asignarPresupuesto.

http://www.arrakis.es/~abelp/ApuntesJava/Herencia.htm

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. Esta clase, llamada String es la representación como objeto de una matriz de caracteres que no se puede cambiar. Hay una clase que la acompaña, llamada StringBuffer, que se utiliza para crear cadenas que pueden ser manipuladas después de ser creadas.

String y StringBuffer

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.

Como son constantes, los Strings son más económicos (utilizan menos memoria) que los StringBuffers y pueder ser compartidos. Por eso es importante utilizar String siempre que sea apropiado.

Creación de cadenas

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.

Se pueden utilizar cadenas literales en cualquier lugar donde se pueda utilizar un objeto String. Por ejemplo, System.out.println() acepta un argumenteo String, por eso se puede utilizar una cadena literal en su lugar:

```
System.out.println("Hola Mundo!");
```

http://dis.um.es/~lopezquesada/documentos/IES_1213/IAW/curso/UT3/ActividadesAlumnos/15/index.html

11. Clase String.

¿Qué es un String?
Los Strings son una secuencia de caracteres. En el lenguaje de programación Java, las cadenas de caracteres son objetos y la plataforma proporciona la clase `String` para crear y manipular dichas cadenas.
¿Cómo creamos un String?
La forma más fácil para crear un String es la siguiente:

```
String nombre = “¡Codehero!";
```

Cada vez que se encuentra con una cadena de caracteres en el código, el compilador crea un objeto String con su valor, en este caso, “¡Codehero!'.

Al igual que con cualquier otro objeto en Java, podemos crear objetos `String` mediante la palabra clave `new` y un constructor de nuestra preferencia. La clase `String` tiene once constructores que nos pueden ayudar a proporcionar el valor inicial de la cadena de caracteres usando diferentes fuentes. Veamos un ejemplo del uso apropiado de uno de esos constructores. En el siguiente caso vamos a construir un **String** a partir de una array con caracteres.

```
public class Ejemplo{

  public static void main(String args[]){
    char[] arrayCaracteres = { 'c', 'a', 'r', 'l', 'o', 's'};
    String nombre = new String(arrayCaracteres);
    System.out.println(nombre);
  }
}
```

La clase String es inmutable, por lo que una vez que se crea un objeto String no se puede cambiar. Si hay una necesidad de hacer una gran cantidad de modificaciones a las cadenas de caracteres, entonces debes usar las siguientes clases **String Buffer** y **String Builder**.

Los métodos mas usados de la clase String en Java son los siguientes:

- length()
- concat()
- printf()

http://codehero.co/java-desde-cero-string/

12. Clase StringBuffer.

Un objeto StringBuffer representa una cadena cuyo tamaño puede variar. La clase StringBuffer dispone de muchos métodos para modificar el contenido de los objetos StringBuffer. Si el contenido de una cadena va a ser modificado en un programa, habrá que sacrificar el uso de objetos String en beneficio de StringBuffer, que aunque consumen más recursos del sistema, permiten ese tipo de manipulaciones.

```
StringBuffer( int len );
```

Métodos principales de StringBuffer.

- setCharAt(int, char)

- StringBuffer insert (int offset, valor)

- StringBuffer append (valor) donde los valores pueden ser:

  - int, char, boolean, float, double, long
  - Object
  - String
  - char [] str

A continuación vamos a ver un ejemplo de StringBuffer append:

```
public static String ugly(String foo, String bar) {
        StringBuffer buffer = new StringBuffer( );
        buffer.append (foo);
        buffer.append (“  ”);
        buffer.append (bar);
}
public static String neat (String foo, String bar) {
      return foo + “   ” + bar;
}
```

http://puntocomnoesunlenguaje.blogspot.com.es/2013/03/java-stringbuilder-stringbuffer.html

13. Clase Math.

La clase **Math** ya viene incluida en nuevas versiones de Java, por lo que no habrá que importar ningún paquete para ello.
Esta clase representa la librería matemática de Java. Su constructor es privado, lo cual nos permite crear instancias de la clase. Puede ser utilizada de la forma **public** o **static**, utilizaremos public para poder llamarla desde cualquier lugar y static para que necesite inicializarla. Esta clase tiene muchos métodos, algunos de los metodos son los siguientes:

Método	Descripción	Parámetros	Tipo de dato devuelto
abs	Devuelve el valor absoluto de un numero.	Un parametro que puede ser un int, double, float o long	El mismo que introduces.
arcos	Devuelve el arco coseno de un angulo en radianes.	Double	Double
asin	Devuelve el arco seno de un ángulo en radianes.	Double	Double
atan	Devuelve el arco tangente entre -PI/2 y PI/2.	Double	Double
atan2	Devuelve el arco tangente entre -PI y PI.	Double	Double
ceil	Devuelve el entero más cercano por arriba.	Double	Double
floor	Devuelve el entero más cercano por debajo.	Double	Double
round	Devuelve el entero más cercano.	Double o float	long (si introduces un double) o int (si introduces un float)

También podemos encontrar constantes definidas:

Constante	Descripción
PI	Devuelve el valor de PI. Es un double.
E	Devuelve el valor de E. Es un double.

Ejemplo:

```
public class PruebaApp {
     public static void main(String[] args) {

       double operador1=25.5;
       double operador2=15.21;

       System.out.println(Math.ceil(operador1)); // Devuelve 26.0
       System.out.println(Math.floor(operador2)); //Devuelve 15.0
       System.out.println(Math.pow(operador1, operador2)); // Devuelve
2.474435537975361E21
       System.out.println(Math.max(operador1, operador2)); //Devuelve 25.5
       System.out.println(Math.sqrt(operador1));   //Devuelve 5.049752469181039
     }
}
```

http://www.discoduroderoer.es/metodos-de-la-clase-math-de-java/

14. Clase Arrays.

Un array es un medio con el cual se guardan 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 declarar un array se escribe:

```
tipo_de_dato[] nombre_del_array;
```

Para declarar y crear un array de 4 número enteros escribimos:

```
int[] numeros =new int[4];
```

Para inicializar el array de 3 enteros escribimos:

```
numeros[0]=2;
numeros[1]=-4;
numeros[2]=15;
```

Los arrays se pueden declarar, crear e inicializar en una misma línea.

```
int[] numeros={2, -4, 15, -25};
String[] nombres={"Juan", "José",   "Miguel",
"Antonio"};
```

Arrays multidimensionales.

Una matriz bidimensional puede tener varias filas, y en cada fila no tiene porque haber el mismo número de elementos o columnas.

Por ejemplo, podemos declarar e inicializar la siguiente matriz bidimensional

```
double[ ][ ] matriz={{1,2,3,4},{5,6},
{7,8,9,10,11,12},{13}};
```

Si usamos matriz.length nos proporciona el número de filas, y matriz[i].length, nos proporciona el número de elementos en cada fila.

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

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

Un tipo primitivo está predefinido por el lenguaje y se nombra con una palabra clave reservada. Los valores primitivos no comparten estado con otros valores primitivos. Algunos tipos de datos primitivos incluidos en el lenguaje de programación Java son los siguientes:

Boolean.
La clase Boolean es una clase que permite manejar los datos equivalentes de tipo primitivo. En este caso la clase Boolean es un wrap del tipo primitivo boolean. Los métodos de esta clase permiten el manejo de los valores primitivos true o false, su modificación o su comparación ya que implementa la interfaz Comparable.

Double.
Es la clase wrap correspondiente al tipo primitivo double, por lo que lo métodos son muy parecidos a los de la clase Boolean, pero manejando los tipos primitivos para double. Permitiendo obtener, modificar, comparar, etc valores de tipo double.

Float.
El tipo de dato float es un dato en coma flotante IEEE 754 de 32 bits y precisión simple.

Integer.
Maneja tipos primitivos de tipo int. Tiene una gran cantidad de métodos sobre todo para poder convertir el entero a otros tipos como long, float, double, etc.

Math.
Tiene una gran cantidad de métodos para poder hacer operaciones matemáticas, como las funciones sin (double a) que calcula el seno del valor a, tan (double a) que calcula la tangente de a, etc.

String.
Permite la definición y manejo de cadenas de caracteres. Pero un inconveniente posible es que se define como constante y tras su creación no puede ser cambiada.

Paquete java.lang.*

Dentro de este paquete están gran parte de las clases más utilizadas dentro de las aplicaciones o programas creados con tecnología Java.

http://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=583:el-paquete-javalang-interfaces-clases-string-integer-stringbuffer-concepto-de-inmutabilidad-cu00909c&catid=58:curso-lenguaje-programacion-java-nivel-avanzado-i&Itemid=180

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

Clase ArrayList.

La clase ArrayList permite almacenar datos en memoria de forma similar a los Arrays, con la ventaja almacena los elementos de forma dinámica, es decir, que no es necesario declarar su tamaño como pasa con los Arrays. Los ArrayList nos permiten añadir, eliminar y modificar elementos (que pueden ser objetos o elementos atómicos) de forma trasparente para el programador.

Para la creacion de un ArrayList declaramos su contructor que seria el siguiente:

```
ArrayList nombre_del_objeto=new ArrayList( );
```

Clase Vector.

Los métodos más habituales en su manipulación son addElement() para insertar elementos en el Vector, elementAt() para recuperarlos y elements() para obtener una Enumeration con el número de elementos del Vector.

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);

Hay dos formas de añadir elementos a un vector. Podemos añadir un elemento a continuación del último elemento del vector, mediante la función miembro addElement.

```
v.addElement("uno");
```

Podemos también insertar un elemento en una determinada posición, mediante insertElementAt.

Métodos de clase ArrayList.

```
boolean add(Objeto)
```

Agrega el elemento especificado al final de esta lista.

```
void add(int indice, Objeto)
```

Inserta el elemento especificado en la posición especificada en esta lista.

```
void clear()
```

Elimina todos los elementos de la lista.

```
boolean contains(Objecto)
```

Devuelve true si la lista contiene el elemento especificado.

```
get (int indice)
```

Devuelve el elemento en la posición especificada en la lista.

```
int indexOf ( Objecto)
```

Nos devuelve la posicion o indice del Objeto declarado en el parametro del metodo.

```
boolean isEmpty()
```

Nos regresa true si la lista no contiene ningun elemento.

```
E remove(int index)
```

Quita el elemento en la posición especificada en esta lista.

```
boolean remove(Object o)
```

Quita la primera aparición del elemento especificado de la lista, si está presente.

```
E set(int index, E element)
```

Reemplaza el elemento en la posición especificada en esta lista con el elemento especificado.

```
int size()
```

Devuelve el numero de elementos almacenados en la colleccion.

http://www.taringa.net/post/ciencia-educacion/14792395/Java-Clase-ArrayList.html

http://jarroba.com/arraylist-en-java-ejemplos/

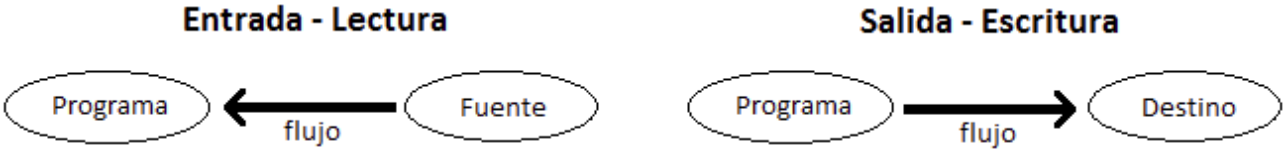
17. Introducción al tratamiento de ficheros en Java. E/S java.io.*

Paquete java.io

Todo dato tiene un origen de entrada o un destino de salida (convenientemente en inglés: I/O). Un programa trabaja con datos que provienen de una entrada la cual

puede ser un teclado, un archivo en disco o un socket a través de la red. Así mismo un programa genera resultados enviándolos a una salida la cual puede ser la pantalla, un archivo en disco o un socket a través de la red. Como podemos comprender hay distintos orígenes de entrada y destinos de salida para los datos, en Java todos estos se consideran flujos (en inglés: streams) de datos para los cuales disponemos, en la API estándar, del paquete java.io especializado en la tarea de poder controlar estos flujos con cierta abstracción sobre el origen de los mismos, gracias a esto siempre podemos recurrir al mismo grupo de clases independientemente del caso a tratar.

Flujo de entrada y salida.
En Java podemos tener un flujo de entrada por el cual recibimos datos desde el exterior de nuestro programa, o un flujo de salida el cual envía nuestros datos hacia un cierto destino.



Hay dos tipos de flujos según el tipo de dato:

• Flujos binarios – byte (8-bit): Es el tipo de flujo mas primitivo y portable, de echo cualquier otro tipo de flujo esta construido sobre este porque hablando a bajo nivel todas las operaciones de I/O son flujos de bytes. Nos permitirá trabajar adecuadamente con datos binarios tales como archivos de imagen, sonido, etc. Las clases principales para manejar estos flujos son las clases abstractas InputStream yOutputStream de las cuales heredan otras sub-clases que implementan en formas mas concretas la misma tarea.

• Flujos de caracteres – char (16-bit): Es un tipo de flujo de caracteres en codificación Unicode, listo para la internacionalización, ideal para trabajar con texto plano. Las clases principales para manejar estos flujos son las clases abstractas Reader y Writerde las cuales heredan otras sub-clases que implementan en formas mas concretas la misma tarea. Cualquiera de estas clases realiza la conversión correspondiente de byte a char para leer o de char a byte para escribir.

Lectura.

Clase InputStream:

```
int read()
```

- Lee el próximo byte del flujo representado en un entero. Devuelve -1 si no quedan mas datos que

leer.

```
int read(byte[] b)
```

- Lee un arreglo de bytes del flujo.

```
int read(byte[] b, int off, int len)
```

- Lee un arreglo de bytes del flujo, desde y hasta la posición indicada.

La clase Reader:

```
int read()
```

- Lee el próximo carácter del flujo representado en un entero. Devuelve -1 si no quedan mas datos que leer.

```
int read(char[] cbuf)
```

- Lee un arreglo de caracteres del flujo.

```
int read(char[] cbuf, int off, int len)
```

- Lee un arreglo de caracteres del flujo, desde y hasta la posición indicada.

Escritura.

La clase OutputStream:

```
void write(int b)
```

- Escribe un solo byte en el flujo.

```
void write(byte[] b)
```

- Escribe un arreglo de bytes en el flujo.

```
void write(byte[] b, int off, int len)
```

- Escribe una porción de un arreglo de bytes en el flujo.

La clase Writer:

```
void write(int c)
```

- Escribe un solo carácter en el flujo.

```
void write(char[] cbuf)
```

- Escribe un arreglo de caracteres en el flujo.

```
void write(char[] cbuf, int off, int len)
```

- Escribe una porción de un arreglo de caracteres en el flujo.

Java.io.IOException

Es un tipo de excepción generalizada que es lanzada cuando algún tipo de operación I/O ah fallado. Esta excepción debe de ser tratada ya sea manejándola con un bloque de código *try-catch* en el mismo momento que se realice la operación o dejando que se propague hacia arriba en la pila de llamadas utilizando throws IOException.

https://darkbyteblog.wordpress.com/2011/01/19/java-paquete-java-io-introduccion-a-los-flujos-de-datosio/

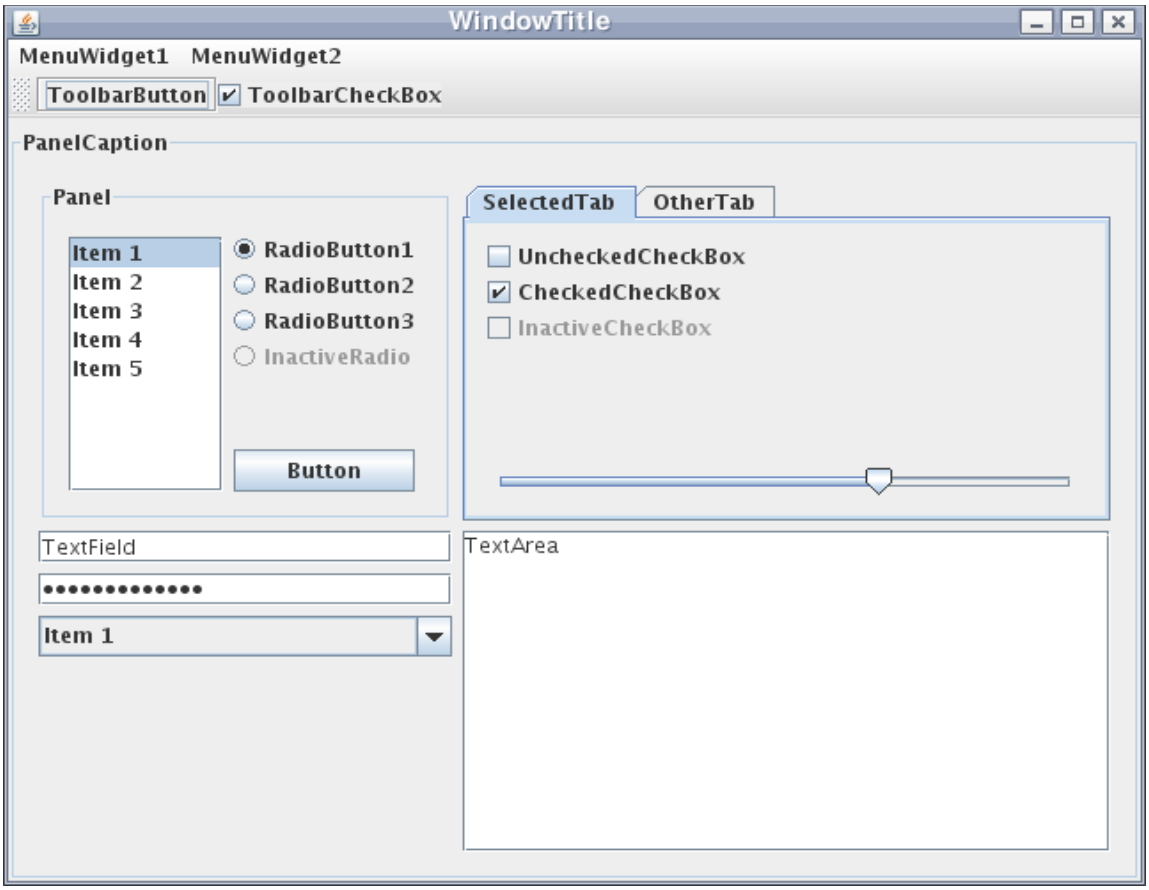
18. Introducción a la programaciónde escritorio. AWT/SWING.

AWT.

Abstract Window Toolkit es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java.

Swing.

Swing es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas. Viene a complementar y ampliar al modelo de componentes y eventos de AWT, basándose en este. Es una de las API's del JFC (Java Foundation Classes).



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

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

Herramientas CASE.

Las herramientas CASE son un conjunto de herramientas y métodos asociados que proporcionan asistencia automatizada en el proceso de desarrollo del software a lo largo de su ciclo de vida. Fueron desarrolladas para automatizar esos procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el

ciclo de desarrollo de software. Pueden realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Clasificación.

1.- Herramientas integradas, I-CASE.
2.- Herramientas de alto nivel, U-CASE.
3.- Herramientas de bajo nivel, L-CASE.
4.- Juegos de herramientas o Tools-Case, son el tipo más simple de herramientas CASE.

Ventajas:

Estas herramientas pueden proveer muchos beneficios en todas las etapas del proceso de desarrollo de software, algunas de ellas son:

- Mejora en la productividad.
- Mejora en la eficacia.
- Mejora en la calidad del sistema de información.
- Disminución de tiempo.
- Automatización de tareas tediosas.
- Garantizar la consistencia de los procedimientos.
- Verificar el uso de todos los elementos en el sistema diseñado.
- Automatizar el dibujo de diagramas.
- Ayudar en la documentación del sistema.
- Ayudar en la creación de relaciones en la Base de Datos.
- Generar estructuras de código.

Desventajas:

- Confiabilidad en los métodos estructurados.
- Falta de niveles estándar para el soporte de la metodología.
- Conflictos en el uso de los diagramas.
- Diagramas no utilizados.
- Función limitada.

- Costo de adquisicion.

Ejemplo CASE – NetBeans:

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE2 es un producto libre y gratuito sin restricciones de uso.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun MicroSystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos.

Ejemplo CASE – Eclipse:

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

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

http://dis.um.es/~lopezquesada/documentos/IES_1415/IAW/curso/material.html