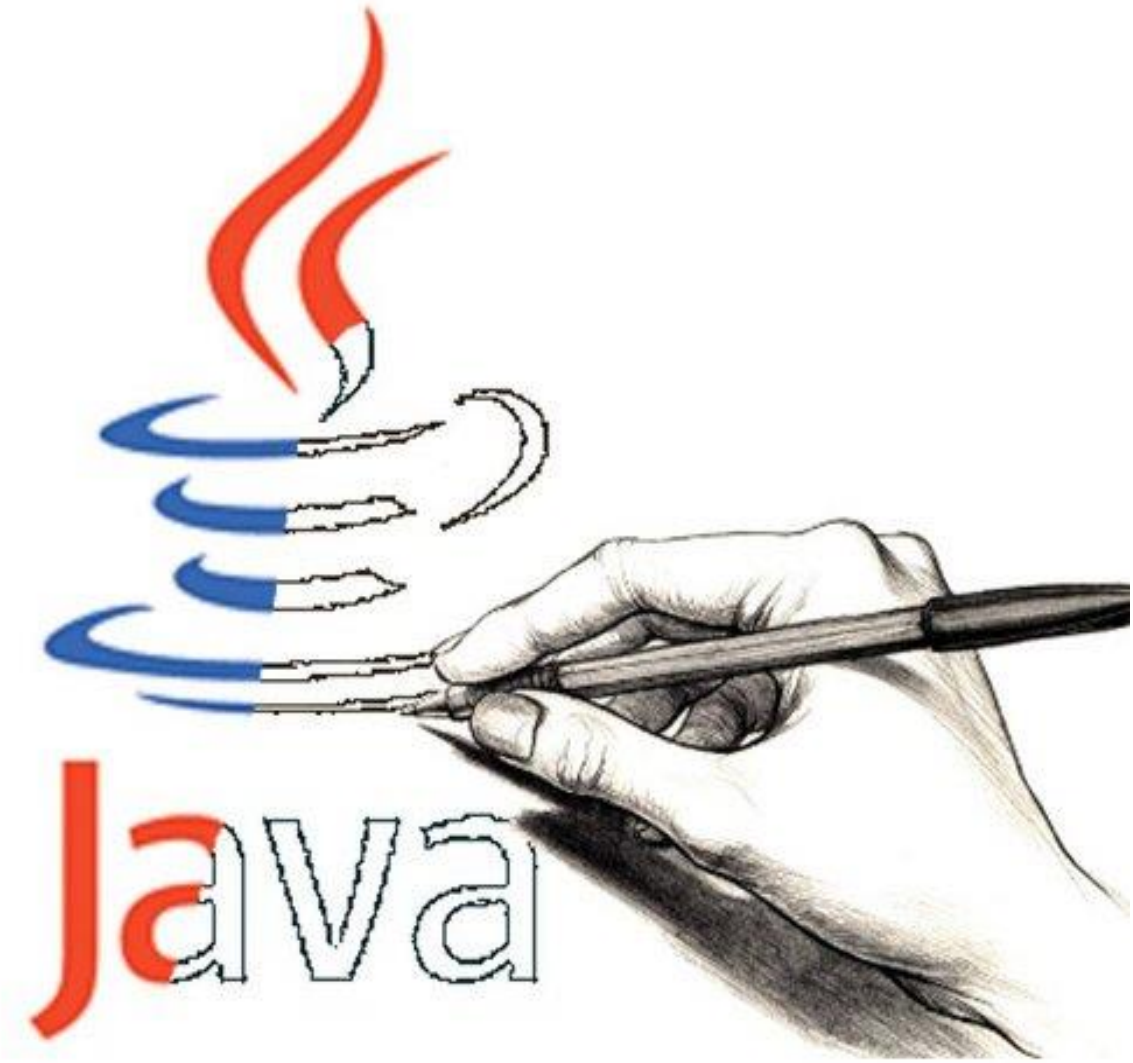
Francisco José Rodríguez López y Alejandro Serrano Zamarreño (2º ASIR)

TUTORIAL DE JAVA



INDICE

1. Introducción

A modo de introducción, decir que en este tutorial sobre Java se van a tratar todos los puntos enunciados en el índice que hemos podido observar, donde se va a tratar el funcionamiento de Java, sus componentes, etc.

Se puede decir a modo introductor, que la principal característica de Java es la de ser un lenguaje compilado e interpretado, en el cual, todo programa en Java ha de compilarse y el código que se genera bytecodes es interpretado por una máquina virtual. De este modo se consigue la independencia de la máquina, ya que el código compilado se ejecuta en máquinas virtuales que si son dependientes de la plataforma.

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

La programación orientada a objetos es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas de computadoras. Un paradigma de programación representa un enfoque particular para la construcción del software. No es mejor uno que otro sino que cada uno tiene ventajas y desventajas.

En la programación orientada a objetos las entidades centrales son los objetos, que son tipos de datos que encapsulan con el mismo nombre estructuras de datos, operaciones o algoritmos que manipulan esos datos.

Podemos distinguir varias propiedades del paradigma orientado a objetos como **abstracción**, **encapsulamiento**, **modularidad** y **jerarquía.**

Por tanto, dicho esto se puede decir que 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. Por lo que, Java se ha convertido, a partir de 2012, uno de los lenguajes de programación más populares en uso.

El lenguaje de programación Java fue desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

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

Los elementos típicos de cualquier lenguaje son los siguientes:

- Identificadores
- Tipos de datos
- Palabras reservadas
- Sentencias
- Bloques de código
- Comentarios
- Expresiones
- Operadores

Tipos de datos

Tipo	Descripción
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 4.94065645841246544e–324d a 1.7976931348623157e+308d.)

Los tipos básicos que se utilizarán en la mayor parte de los programas serán **boolean**, **int** y **double**.

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

Las variables de instancia o miembros dato como veremos más adelante, se usan para guardar los atributos de un objeto particular.

Las variables de clase o miembros dato estáticos son similares a las variables de instancia, con la excepción de que los valores que guardan son los mismos para todos los objetos de una determinada clase. 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 área es una variable local a la función calcular área en la que se guarda el valor del área de un objeto de la clase círculo. 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 o un array.

El lenguaje Java utiliza el conjunto de caracteres Unicode, que incluye no solamente el conjunto ASCII sino también carateres específicos de la mayoría de los alfabetos. Así, podemos declarar una variable que contenga la letra ñ

```
int año=1999;
```

Se ha de poner nombres significativos a las variables, generalmente formados por varias palabras combinadas, la primera empieza por minúscula, pero las que le siguen llevan la letra inicial en mayúsculas. Se debe evitar en todos los casos nombres de variables cortos como *xx*, *i*, etc.

```
double radioCirculo=3.2;
```

Las variables son uno de los elementos básicos de un programa, y se deben

- Declarar
- Inicializar
- Usar

Los operadores son muy similares a los de C++, ya lo advertimos en su momento.

- **Operadores Aritméticos.** Los habituales son:
  - Suma +.
  - Resta - .
  - Multiplicación *.
  - División /.
  - Resto de la División %.
- **Operadores de Asignación.** El principal es "=" pero hay más operadores de asignación con distintas funciones que explicamos brevemente ahora.
  - '+=' : op1 += op2 à op1 = op1 + op2
  - '-=' : op1 -= op2 à op1 = op1 - op2
  - '*=' : op1 *= op2 à op1 = op1 * op2
  - '/=' : op1 /= op2 à op1 = op1 / op2
  - '%=' : op1 %= op2 à op1 = op1 % op2
- **Operadores Unarios.** El mas (+) y el menos (-). Para cambiar el signo del operando.
- **Operador Instanceof.** Nos permite saber si un objeto pertenece a una clase o no.
  - NombreObjeto instanceof NombreClase
- **Operadores Incrementales.** Son los operadores que nos permiten incrementar las variables en una unidad. Se pueden usar delante y detrás de la variable dependiendo de lo que queramos, es decir, si queremos que incremente o viceversa antes de utilizar o lo contrario.
  - '++'
  - '--'
- **Operadores Relacionales.** Permiten comparar variables según relación de igualdad/desigualdad o relacción mayor/menor. Devuelven siempre un valor boolean.
  - '>': Mayor que

- '<': Menor que
- '==': Iguales
- '¡=': Distintos
- '>=': Mayor o igual que
- '<=': Menor o igual que

- **Operadores Lógicos.** Nos permiten construir expresiones lógicas.
  - '&&': devuelve true si ambos operandos son true.
  - '||': devuelve true si alguno de los operandos son true.
  - '!' : Niega el operando que se le pasa.
  - '&' : devuelve true si ambos operandos son true, evaluándolos ambos.
  - '|' : devuelve true uno de los operandos es true, evaluándolos ambos.
- **Operador de concatenación con cadena de caracteres '+':**
  - Por ejemplo: System.out.println("El total es"+ result +"unidades");
- **Operadores que actúan a nivel de bits.** Son mucho menos utilizados por eso los explicamos más por encima.
  - '>>': desplazamiento a la derecha de los bits del operando
  - '<<': desplazamiento a la izquierda de los bits de operando
  - '&': operador and a nivel de bit.
  - '|': operador or a nivel de bit**.**

4. 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.

- Sentencias de selección.

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

La sentencia if es una sentencia de bifurcación condicional, para dirigir el flujo de ejecución hacia dos caminos diferentes.
Además, es importante destacar los if anidados. Por lo que, un if anidado es una sentencia if que está dentro de otro if o else.
Ejemplo:

```
i f ( i == 10) {
i f ( j < 20) a = b ;
i f ( k > 100) c = d ;
```

```
else a = c ;
} else a = d ;
```

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:

}
```

- Sentencias de iteración.

Las sentencias **while**, **do-while** y **for** permiten crear lo que se denominan ciclos.

- Sentencias de salto.

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

Break se utiliza para finalizar un ciclo.

La sentencia continue forza en un ciclo la ejecución de una nueva iteración descartando el procesamiento de la iteración actual.

Return se utiliza para salir de un método.

5. Clase y Objetos en java. Estructura básica.

Una clase en Java se puede entender como un prototipo que define las variables y los métodos comunes a un cierto tipo de instancias, una clase define todo lo que caracteriza y pueden hacer una o varias instancias.

En Java, un objeto es básicamente una instancia de una clase.

Una clase en Java nos proporcionará los atributos y métodos necesarios que un objeto necesitará para interactuar con otros, por lo que es importante saber estructurar correctamente las clases.

Con la siguiente sentencia, estamos declarando una clase:

```
public class NombreDeLaClase{}
```

La palabra "public" indica que el alcance de la clase será público, no significa "publicar clase". Después de 'public class' escribimos el identificador de la clase. Se recomienda elegir un identificador abstracto a la clase, para facilitar la comprensión del código.

Por ejemplo, si queremos modelar objetos de tipo "Persona", es recomendable que el identificador sea "Persona" y no "Juan", el identificador de la clase debe proporcionar una **idea general** y no una idea específica.

En Java se usan llaves { } para agrupar trozos de código determinados, es por eso que todo el contenido de la clase que estamos haciendo debe estar entre sus respectivas llaves.

6. Sistema de Excepciones en Java. Exception.

El manejo de excepciones es una técnica de programación que permite al programador controlar los errores ocasionados durante la ejecución de un programa informático. Cuando ocurre cierto tipo de error, el sistema reacciona ejecutando un fragmento de código que resuelve la situación, por ejemplo retornando un mensaje de error o devolviendo un valor por defecto.

Ejemplo:

```
import java.io.IOException;

// ...

public static void main(String[] args) {
   try {
      // Se ejecuta algo que puede producir una excepción
   } catch (IOException e) {
      // manejo de una excepción de entrada/salida
   } catch (Exception e) {
      // manejo de una excepción cualquiera
   } finally {
      // código a ejecutar haya o no excepción
   }
}
```

7. Estructuras estáticas: Vectores y Matrices.

Un vector, también llamado array unidimensional, es una estructura de datos que permite agrupar elementos del mismo tipo y almacenarlos en un solo bloque de memoria juntos, uno después de otro. A este grupo de elementos se les identifica por un mismo nombre y la posición en la que se encuentran. La primera posición del array es la posición 0.

Podríamos agrupar en un array una serie de elementos de tipo enteros, flotantes, caracteres, objetos, etc.

```
#include <iostream>
using namespace std;

int main()
{
  int dim;
  cout << "Ingresa la dimension del vector" << endl;
  cin >> dim; // Supongamos que ingrese 10
  int vector[dim]; // mi vector es de tamanyo 10

  for(int i = 0; i < dim; i++){
    vector[i] = i * 10;
    cout << vector[i] << endl;
  }

  return 0;
}
```

Una matriz es un vector de vectores o también recibe el nombre de array bidimensional.

```
int matrix[rows][cols];
```

int es el tipo de dato, matrix es el nombre del todo el conjunto de datos y debo de especificar el numero de filas y columnas.

Las matrices también pueden ser de distintos tipos de datos como char, float, double, etc. Las matrices en C++ se almacenan al igual que los vectores en posiciones consecutivas de memoria.

Usualmente uno se hace la idea que una matriz es como un tablero, pero internamente el manejo es como su definición lo indica, un vector de vectores, es decir, los vectores están uno detrás del otro juntos.

La forma de acceder a los elementos de la matriz es utilizando su nombre e indicando los 2 subíndices que van en los corchetes.

Si coloco int matriz[2][3] = 10; estoy asignando al cuarto elemento de la tercera fila el valor 10.

No olvidar que tanto filas como columnas se enumeran a partir de 0. Bueno y para recorrer una matriz podemos usar igualmente un bucle. En este caso usando 2 for:

```
for(int i = 0; i < rows; i++) {
```

```
  for(int j = 0; j < cols; j++) {
    matrix[i][j] = i % j;
  }
}
```

8. Clase Object.

La clase Object está situada en la parte más alta del árbol de la herencia en el entorno de desarrollo de Java. Todas las clases del sistema Java son descendentes (directos o indirectos) de la clase Object. Esta clase define los estados y comportamientos básicos que todos los objetos deben tener, como la posibilidad de compararse unos con otros, de convertirse a cadenas, de esperar una condición variable, de notificar a otros objetos que la condición variable ha cambiado y devolver la clase del objeto.

Los métodos de la clase Object en Java

Los métodos públicos y protegidos de la clase Object son:

- **Public boolean equals(Object obj):** compara si dos objetos son iguales, por defecto un objeto es igual solamente a si mismo.
- **Public int hashCode():** Devuelve un valor de código hash para el objeto. Este método se apoya en beneficio de tablas hash tales como los proporcionados por **java.util.Hashtable.**
- **Protected Object clone() throws CloneNotSuportedException:** devuelve una copia binaria del objeto, al parecer al hacer la copia hace referencia a una nueva posición de memoria.
- **Public final Class getClass():** devuelve el objeto del tipo Class que representa dicha clase durante la ejecución, es decir devuelve el tipo de clase al que pertenece.
- **Protected void finalize() throws Throwable:** se usa para finalizar el objeto, es decir, se avisa al administrador de la memoria que ya no se usa dicho objeto, y se puede ejecutar código especial antes de que se libere la memoria.
- **Public String toString()**:devuelvo una cadena describiendo el objeto.
- **Void nativa public final notify():** Se despierta un solo hilo que está esperando en el monitor de este objeto. Un subproceso espera en el monitor de un objeto llamando a uno de los de espera métodos.
- **Final public native void notifyAll ():** Se despierta todos los temas que están en espera en el monitor de este objeto. Un subproceso espera en el monitor de un objeto llamando a uno de los de espera métodos.

- **Final public void wait ():** Espera que se le notifique por otro subproceso de un cambio en este objeto.

9. Clase System.

Hay ocasiones en que se necesita acceder a recursos del sistema, como son los dispositivos de entrada/salida, el reloj del sistema, etc. Java dispone de la clase System, que proporciona acceso a estos recursos, independientemente de la plataforma. Es decir, que si se ejecuta un programa en una plataforma diferente a la que se ha desarrollado, no es necesaria ninguna modificación para tener en cuenta las peculiaridades de la nueva plataforma.

La clase System es miembro del paquete java.lang y en ella se definen los dispositivos estándar de entrada/salida:

```
static PrintStream err;
static InputStream in;
static PrintStream out;
```

Además, dispone de varios métodos, algunos de los cuales ya se han utilizado en secciones anteriores:

```
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 )
```

No se puede instanciar ningún objeto de la clase System, porque es una clase final y todos sus contenidos son privados; por ellos es por lo que no hay una lista de constructores en la enumeración de métodos. Es decir, la clase System siempre está ahí disponible para que se pueda invocar cualquiera de sus métodos utilizando la sintaxis de punto (.) ya conocida

```
System.out.println( "Hola Java" );
```

10. Herencia en Java.

La herencia es una propiedad de algunos lenguajes de programación como Java que permite que un objeto sea creado a partir de otro existente, obteniendo características como atributos y métodos, lo que nos permite crear objetos derivados a partir de objetos bases.

Ejemplo:

```
public class Main {

    // ArrayList de objetos SeleccionFutbol. Idenpendientemente de la clase hija a la que pertenezca el objeto
        public static ArrayList<SeleccionFutbol> integrantes = new ArrayList<SeleccionFutbol>();

        public static void main(String[] args) {

                Entrenador delBosque = new Entrenador(1, "Vicente", "Del Bosque", 60, "284EZ89");
                Futbolista iniesta = new Futbolista(2, "Andres", "Iniesta", 29, 6, "Interior Derecho");
                Masajista raulMartinez = new Masajista(3, "Raúl", "Martinez", 41, "Licenciado en Fisioterapia", 18);

                integrantes.add(delBosque);
                integrantes.add(iniesta);
                integrantes.add(raulMartinez);

                // CONCENTRACION
                System.out.println("Todos los integrantes comienzan una concentracion. (Todos ejecutan el mismo método)");
                for (SeleccionFutbol integrante : integrantes) {
        System.out.print(integrante.getNombre()+" "+integrante.getApellidos()+" -> ");
                 integrante.Concentrarse();
            }

            // VIAJE
                System.out.println("nTodos los integrantes viajan para jugar un partido. (Todos ejecutan el mismo método)");
                for (SeleccionFutbol integrante : integrantes) {
             System.out.print(integrante.getNombre()+" "+integrante.getApellidos()+" -> ");
                integrante.Viajar();
            }
      .....
```

11. 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.

En los siguientes dos puntos (11 y 12) se desarrolla estas dos clases que acabamos de mencionar: la clase String y la clase StringBuffer.

12. Clase String.

Un String en Java representa una cadena de caracteres no modificable.

Todos los literales de la forma "*cualquier texto*", es decir, literales entre comillas dobles, que aparecen en un programa java se implementan como objetos de la clase String.

Se puede crear un String de varias formas, entre ellas:

- Utilizando una cadena de caracteres entre comillas:

String s1 = "abcdef";

- Utilizando operador de concatenación +con dos o más objetos String:

String s2 = s1 + "ghij"; //s2 contiene "abcdefghij"

String s3 = s1 + s2 + "klm"; //s3 contiene " abcdefabcdefghijklm"

Además la clase String proporciona varios constructores, entre ellos:

Constructor	**Descripción**

String()	Constructor por defecto. El nuevo String toma el valor ""
	String s = new String();//crea el string s vacío.
	Equivale a: String s = "";

String(String s)	Crea un nuevo String, copiando el que recibe como parámetro. String s = "hola"; String s1 = new String(s);

	//crea el String s1 y le copia el contenido de s
String(char[] v)	Crea un String y le asigna como valor los caracteres contenidos en el array recibido como parámetro. char [] a = {'a', 'b', 'c', 'd', 'e'}; String s = new String(a); //crea String s con valor "abcde"
String(char[] v, int pos, int n)	Crea un String y le asigna como valor los n caracteres contenidos en el array recibido como parámetro, a partir de la posición pos. char [] a = {'a', 'b', 'c', 'd', 'e'}; String s = new String(a, 1, 3); //crea String s con valor "bcd";

La clase String proporciona métodos para el tratamiento de las cadenas de caracteres: acceso a caracteres individuales, buscar y extraer una subcadena, copiar cadenas, convertir cadenas a mayúsculas o minúsculas, etc.

Método	**Descripción**
length()	Devuelve la longitud de la cadena
indexOf(‘caracter’)	Devuelve la posición de la primera aparición de carácter
lastIndexOf(‘caracter’)	Devuelve la posición de la última aparición de carácter
charAt(n)	Devuelve el carácter que está en la posición n
substring(n1,n2)	Devuelve la subcadena comprendida entre las posiciones n1 y n2-1
toUpperCase()	Devuelve la cadena convertida a mayúsculas
toLowerCase()	Devuelve la cadena convertida a minúsculas
equals("cad")	Compara dos cadenas y devuelve true si son iguales
equalsIgnoreCase("cad")	Igual que equals pero sin considerar mayúsculas y minúsculas
compareTo(OtroString)	Devuelve 0 si las dos cadenas son iguales. <0 si la primera es alfabéticamente menor que la segunda ó >0 si la primera es alfabéticamente mayor que la segunda.

compareToIgnoreCase(OtroString)	Igual que compareTo pero sin considerar mayúsculas y minúsculas.
valueOf(N)	Método estático. Convierte el valor N a String. N puede ser de cualquier tipo.

Se puede decir que los objetos String no son modificables, por lo que, los métodos que actúan sobre un String con la intención de modificarlo lo que hacen es crear un nuevo String a partir del original y devolverlo modificado.

Por ejemplo: una operación como convertir a mayúsculas o minúsculas un String no lo modificará sino que creará y devolverá un nuevo String con el resultado de la operación.

13. Clase Stringbuffer.

Java proporciona la clase StringBuffer y a partir de Java 5 la clase StringBuilder para trabajar con cadenas de caracteres sobre las que vamos a realizar modificaciones frecuentes de su contenido.

La diferencia entre StringBuffer y StringBuilder es que los métodos de StringBuffer están sincronizados y los de StringBuilder no lo están. Por este motivo StringBuilder ofrece mejor rendimiento que StringBuffer y la utilizaremos cuando la aplicación tenga un solo hilo de ejecución.

En general decidiremos cuando usar String, StringBuilder o StringBuffer según lo siguiente:

- Usaremos String si la cadena de caracteres no va a cambiar.

- Usaremos StringBuilder si la cadena de caracteres puede cambiar y solamente tenemos un hilo de ejecución.

- Usaremos StringBuffer si la cadena de caracteres puede cambiar y tenemos varios hilos de ejecución.

En esta entrada utilizaremos StringBuilder teniendo en cuenta que todo lo que se explica aquí es aplicable a StringBuffer.

Constructores de la Clase StringBuilder

Un objeto de tipo StringBuilder gestiona automáticamente su capacidad:

- Se crea con una capacidad inicial.
- La capacidad se incrementa cuando es necesario.

La clase StringBuilder proporcionan varios **constructores**, algunos de ellos son:

Constructor	Descripción
StringBuilder ()	Crea un StringBuilder vacío. StringBuilder sb = new StringBuilder ();
StringBuilder(int n)	Crea un StringBuilder vacío con capacidad para n caracteres.
StringBuilder(String s);	Crea un StringBuilder y le asigna el contenido del String s. String s = "ejemplo"; StringBuilder sb = new StringBuilder (s);

14. Clase Math.

Esta clase ya viene incluida en nuevas versiones de Java, por lo que no habrá que importar ningún paquete para ello.

Para utilizar esta clase, debemos escribir Math.método(parámetros); donde método sera uno de los siguientes y parámetros aquellos que tengamos que usar. Un método puede estar sobrescrito para distintos tipos de datos.

Recuerda que si almacenas el resultado de la función, debe coincidir con el tipo de la variable.

Método	Descripción	Expresión de ejemplo	Resultado del ejemplo
abs	Valor absoluto	Math.abs(-2)	2
sin, cos, tan	Funciones trigonométricas, reciben el argumento en radianes	Math.cos(Math.PI)	-1
asin, acos, atan	Funciones trigonométricas inversas	Math.asin(1)	1.57
exp, log	Exponenciación y logaritmo, base E	Math.log(Math.E)	1
ceil	Devuelve el entero más pequeño mayor o igual al argumento	Math.ceil(-2.7)	-2
floor	Devuelve el entero más grande menor o igual al argumento	Math.floor(-2.7)	-3
round	Devuelve el entero más cercano o igual al argumento	Math.round(-2.7)	-3
min, max	Devuelve el menor (o mayor) de sus dos argumentos	Math.min(2,4)	2

pow	Exponenciación, siendo el primer argumento la base y el segundo el exponente	Math.pow(2,3)	8
sqrt	Raíz cuadrada	Math.sqrt(25)	5
random	Genera un valor aleatorio comprendido entre 0 y 1.	Math.random()	Ej. 0.7345

15. 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.

Estos son los pasos que veremos para construir un array:

- Declarar el array:

Para declarar un array se escribe

tipo_de_dato[] nombre_del_array;

- Crear el array

Para crear un array de 4 número enteros escribimos

numeros=new int[4];

(La declaración y la creación del array se puede hacer en una misma línea

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

- Inicializar los elementos del array

Para inicializar el array de 4 enteros escribimos

```
numeros[0]=2;
numeros[1]=-4;
numeros[2]=15;
numeros[3]=-25;
```

Se pueden inicializar en un bucle for como resultado de alguna operación

```
for(int i=0; i<4; i++){
    numeros[i]=i*i+4;
}
```

(Los arrays se pueden declarar, crear e inicializar en una misma línea, del siguiente modo

int[] numeros={2, -4, 15, -25};

String[] nombres={"Juan", "José", "Miguel", "Antonio"};)

- Usar el array

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

Los tipos nativos de Java, son los tipos de datos "fáciles de usar" es decir, no es necesario crear un objeto de manera explícita para hacer uso de ellos. Los tipos primitivos son los tipos de datos más básicos y simples del sistema de tipos de Java y es bastante fácil usarlos.

Tipos de datos primitivos y sus características:

- CHAR: Permite representar caracteres aislados, por medio de un único char no podemos representar palabras completas sino más bien letras individuales. Por ejemplo entonces la palabra "carácter" estaría conformada por un total de ocho chars ‘c’, ‘a’, ‘r’, ‘a’, ‘c’, ‘t’, ‘e’ y ‘r’.

- BOOLEAN: Nos permite representar valores lógicos o booleanos. Una variable boolean, solo puede tomar dos posibles valores false o true, este tipo de variables son especialmente usadas para evaluar condiciones en las cuales es necesario conocer el valor de verdad de cierta operación lógica.

- DATO BYTE: Este tipo de datos representa pequeños números enteros, puede contener números entre -128 y 127.

- SHORT: Este es usado para representar números enteros más grandes que byte, varía entre -32768 y 32767

- INT: Este tipo de dato es uno de los más populares, pues generalmente cuando se piensa en un número entero inmediatamente se coloca la variable tipo int, sin embargo generalmente int suele ser más grande de lo que llegamos a necesitar, aproximadamente desde -2.147’483.648 hasta 2.147’483.647.

- LONG: El tipo primitivo long es usado para representar números enteros realmente grandes, aproximadamente desde -9.223’’372.036.854’775.808 hasta 9.223’’372.036.854’775.807.

- FLOAT: El tipo float es útil cuando queremos representar números decimales, como por ejemplo resultados de divisiones, raíces o similares.

- DOUBLE: Este tipo es el tipo de dato numérico más grande, con este podemos representar casi cualquier número que necesitemos.

17. Clase ArrayList y Clase Vector.

La clase ArrayList permite crear una "colección" de elementos al interior de sí misma, es similar a un Vector o arreglo, y posee una gran variedad de métodos y atributos que nos permiten por ejemplo, buscar un elemento cualquiera, insertar un nuevo elemento, eliminarlo, entre otras. Es básicamente un arreglo pero con una enorme cantidad de facilidades para gestionar los datos que este contiene.

Para crear un ArrayList:

ArrayList nombreArrayList = new ArrayList(); *//Array List simple*

Para realizar las operaciones comunes tales como ingresar, eliminar, buscar, modificar datos en un ArrayList encontraremos algunos métodos útiles para tal objetivo:

- add(X)**:** Añade un valor u objeto al final del ArrayList.
- size()**:** Retorna el tamaño del ArrayList.
- indexOf(X): Retorna el indice ó la posicion del elemento X.
- contains(X): Retorna true si existe el elemento X en el ArrayList.
- set(i, X)**:** Modifica el elemento que está en la posición i, por el nuevo elemento X.
- remove(X): Elimina el elemento X o en su defecto el elemento en la posición X.
- get(i): Obtiene el elemento en la posición i del Array List.

La clase Vector también implementa a List, pero de un modo especial. Este modo especial es sincronizado, lo que permite que se pueda usar en entornos concurrentes (en varios procesos que se ejecutan al mismo tiempo). Es la principal característica que la diferencia de otras clases estudiadas anteriormente como ArrayList.

Solo utilizaremos la clase Vector si tenemos previstas circunstancias especiales como procesos concurrentes.

Un objeto de tipo Vector contiene elementos que pueden ser accedidos por un índice y puede aumentar o disminuir su tamaño dinámicamente en tiempo de ejecución.

18. Tratamiento de ficheros en Java. E/S java.io.*

Un paquete Java es un conjunto de clases e interfaces relacionadas entre sí. Las clases e interfaces que forman parte de la plataforma Java se estructuran en varios paquetes organizados por funciones o tareas. Las clases fundamentales están en java.lang, las clases para operaciones de entrada y salida de datos están en java.io, etc.

Tipos de paquetes:

- **Java.lang** contiene las clases e interfaces más empleadas en la mayoría de los programas de Java. Es importado automáticamente por todos los programa Java: no se necesita sentencia import, para utilizar lo declarado en este paquete.

- **Java.io** contiene clases que permiten las operaciones de entrada y salida de datos de un programa.

- **Java.util** contiene clases e interfaces de utilidades: operaciones con la fecha y la hora, generación de números aleatorios...
- **Java.awt** es el paquete Abstract Windowing Toolkit. Contiene muchas clases e interfaces necesarias para trabajar con la interfaz de usuario gráfica clásica.
- **Java.beans** contiene clases para facilitar a los programadores la generación de componentes de software reutilizables.

Cualquier programador puede introducir sus clases e interfaces en paquetes para facilitar tanto su uso en el desarrollo de un programa como su reutilización en varios de ellos. Para incluir una clase en un paquete debe incluirse la siguiente sentencia al principio del archivo fuente que contiene la clase: *packageidentificadordelPaquete;*

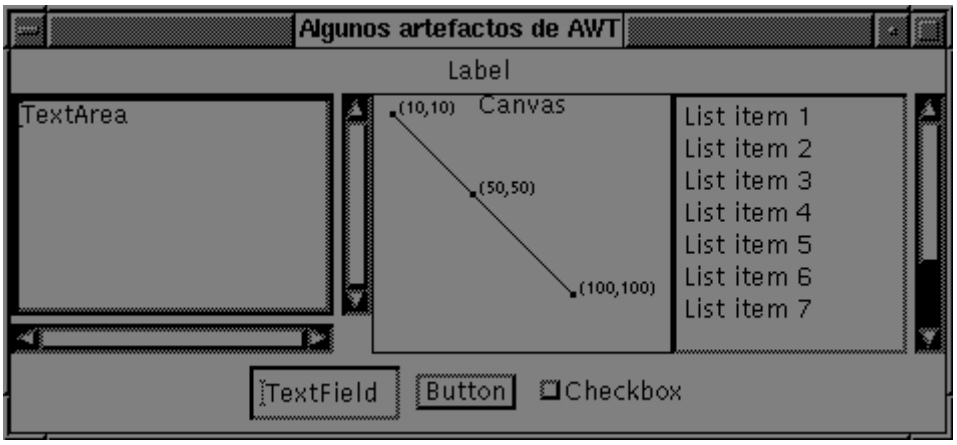
Existen cuatro tipos de modificadores de acceso en Java: **public**, **private**, **protected** y **package**. Los tres primeros se han de definir explícitamente mediante la palabra reservada en la declaración del atributo o del método correspondiente.

- **Public**. Acceso desde cualquier método de cualquier clase.
- **Package.** Acceso desde cualquier método de una clase perteneciente al mismo paquete (mismo directorio).
- **Protected.** Acceso desde cualquier método de una clase perteneciente al mismo paquete (mismo directorio) y desde las clases descendientes (subclases).
- **Private.** Acceso exclusivo desde los métodos de la clase correspondiente.

19. Programación de escritorio. AWT/SWING.

La Abstract Window Toolkit (AWT) es un kit de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java. AWT es ahora parte de las Java Foundation Classes (JFC) - la API estándar para suministrar una interfaz gráfica de usuario (GUI) para un programa Java.

AWT permite hacer interfaces gráficas mediante artefactos de interacción con el usuario, como botones, menús, texto, botones para selección, barras de deslizamiento, ventanas de diálogo, selectores de archivos, etc. Y por supuesto despliegue gráfico general. La siguiente figura muestra algunos de estos artefactos de interacción:



Estos artefactos de interacción se denominan *widgets*. En la ventana los artefactos se organizan en una jerarquía de componentes gráficas:

- En la parte superior hay una etiqueta que dice label. La aplicación usa las etiquetas para mostrar texto.

A la izquierda hay un área para texto que contiene:

  - Una superficie para ingresar texto.
  - Una barra de deslizamiento vertical.
  - Una barra de deslizamiento horizontal.

- A la derecha se observa una lista de ítemes que contiene:
  - Una superficie para mostrar texto.
  - Una barra de deslizamiento (sólo aparece si es necesario).
- Al centro hay un canvas, donde la aplicación dibuja figuras geométricas y/o texto.
- Abajo hay un panel de componentes con:
  - Un campo para ingreso de texto (una sola línea).
  - Un botón.
  - Un botón de encendido/apagado que dice checkbox.

Es el programador de la interfaz gráfica el que diseña esta jerarquía de componentes.

20. Herramientas CASE para el desarrollo de artefactos SWen Java.

Son diversas aplicaciones informáticas o programas informáticos destinados a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de 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.

Aunque es difícil y existen muchas formas de clasificarlas, las herramientas CASE se pueden clasificar teniendo en cuenta los siguientes parámetros:

- Las plataformas que soportan.
- Las fases del ciclo de vida del desarrollo de sistemas que cubren.
- La arquitectura de las aplicaciones que producen.
- Su funcionalidad.

Ejemplos:

- **Microsoft Project** es un software de gestión de proyectos, desarrollado y comercializado por Microsoft, que está diseñado para ayudar a un administrador de proyectos en el desarrollo de planes, la asignación de recursos a tareas, el seguimiento de los progresos, la gestión del presupuesto, y el análisis de las cargas de trabajo.
- **Visual Paradigm for UML.** La herramienta está diseñada para un amplio rango de usuarios, incluyendo ingenieros de software, analistas de sistema, analistas de negocio, arquitectos de sistemas y cualquiera que esté interesado en construir un sistema confiable a gran escala a través del usuario del enfoque orientado a objetos.
- **ArgoUML.** Herramienta que contiene funciones avanzadas en las etapas de diseño y modelación de software.
- **Poseidón.** Es una herramienta para modelar cualquier clase de sistema, relacionado o no con programación por computadoras. Se presenta en dos ediciones: Community Edition y Professional Edition.
- **Modelio** tiene una versión de paga y otra de código abierto mantenido por la comunidad que no tiene ningún costo. Soporta diagramas UML, entre otras funcionalidades.

21. Conclusiones

A modo de conclusión, decir que para nosotros ha sido elemental investigar en sí que es java, ya que se caracteriza por ser un lenguaje muy utilizado actualmente.

Además, hay que destacar que Java como ya bien sabemos, es un lenguaje que fue creado a similitud del lenguaje c, pero con una diferencia que está en que este lenguaje nos ayuda a tener una mayor seguridad en el programa que se está ejecutando.

22. Bibliografía

A continuación, podemos observar la bibliografía que hemos utilizado para poder llevar a cabo este tutorial sobre Java:

- http://es.slideshare.net/NesMey/paradigma-orientado-a-objetos-4954115
- http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)
- http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/introduccion/primero.htm
- http://lsc.fie.umich.mx/~pedro/metodoprogra/sentenciasdecontrol.pdf
- http://puntocomnoesunlenguaje.blogspot.com.es/2013/03/java-stringbuilder-stringbuffer.html
- https://ronnyml.wordpress.com/2009/07/04/vectores-matrices-y-punteros-en-c/
- http://aprendepooconjava.blogspot.com.es/2012/09/estructura-de-una-clase-en-java.html
- http://puntocomnoesunlenguaje.blogspot.com.es/2013/02/clase-string.html
- http://www.javascriptya.com.ar/temarios/descripcion.php?cod=21
- http://www.discoduroderoer.es/metodos-de-la-clase-math-de-java/
- http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/arays.htm
- https://www.programarya.com/Cursos/Java/Sistema-de-Tipos
- http://jagonzalez.org/arraylist-en-java-con-ejemplo-practico/
- http://aprenderaprogramar.es/index.php?option=com_content&view=article&id=606:la-clase-vector-del-api-java-metodos-trimtosize-y-ensurecapacity-ejemplo-y-ejercicios-resueltos-cu00919c&catid=58:curso-lenguaje-programacion-java-nivel-avanzado-i&Itemid=180
- http://yaqui.mxl.uabc.mx/~eherrera/clasevector.htm
- http://ocw.upm.es/lenguajes-y-sistemas-informaticos/programacion-en-java-i/Contenidos/LecturaObligatoria/19-packages-o-paquetes.pdf
- http://users.dcc.uchile.cl/~lmateu/Java/Apuntes/awt.htm
- https://es.wikiversity.org/wiki/Herramientas_CASE