

# **U.T.1. Identificación de un Programa Informático**

## **Módulo de Programación**

Departamento de Informática

---

I.E.S. Monte Naranco

## 1.- INTRODUCCIÓN

### ¿QUÉ ES UN PROGRAMA?

Es un conjunto de instrucciones que nos permiten dar una solución a un problema.

### FASES O PASOS PARA REALIZAR UN PROGRAMA

#### 1. Análisis del problema

Consiste en el estudio del problema que hay que resolver.

#### 2. Fase de diseño

Consiste en diseñar una solución al problema mediante un algoritmo.

Un algoritmo es una secuencia ordenada de pasos que conducen a la solución de un problema dado.

Los algoritmos son independientes de los lenguajes de programación y de las computadoras donde se ejecutan. Un mismo algoritmo puede ser expresado en diferentes lenguajes de programación y podría ser ejecutado en diferentes dispositivos.

Para representar gráficamente los algoritmos que vamos a diseñar, tenemos a nuestra disposición diferentes herramientas que ayudarán a describir su comportamiento de una forma precisa y genérica, para luego poder codificarlos con el lenguaje que nos interese. Entre otras tenemos:

**Diagramas de flujo:** Esta técnica utiliza símbolos gráficos para la representación del algoritmo.

**Pseudocódigo:** Esta técnica se basa en el uso de palabras clave en lenguaje natural, constantes, variables, otros objetos, instrucciones y estructuras de programación que expresan de forma escrita la solución del problema. Es la técnica más utilizada actualmente.

#### 3. Implementación

Consta de varias fases:

##### 3.1. Fase de codificación

Consiste en escribir el problema en un lenguaje de programación adecuado (código fuente), es decir, consiste en escribir un programa.

##### 3.2. Fase de edición

Consiste en introducir el programa dentro del ordenador.

### 3.3. Fase de compilación o interpretación

Consiste en traducir el programa fuente, al lenguaje que realmente entiende el ordenador que es el lenguaje máquina (0's y 1's), dando lugar al programa objeto.

### 3.4. Fase de ejecución

Consiste en probar el programa con datos, y comprobar si los resultados son correctos.

### 4. Documentación

Puede ser interna (comentarios que se incluyen en el código fuente), o externa (manuales para una mejor utilización del programa).

## CARACTERÍSTICAS DE LOS PROGRAMAS

Para un determinado problema se pueden construir diferentes algoritmos o programas que los resuelvan. Elegir una solución u otra se debe basar en los siguientes puntos:

- Qué el programa tenga legibilidad, es decir, que no sólo lo entienda la persona que lo hizo sino todo el mundo.
- Qué el programa sirva para todos los casos y los datos de entrada que el programador le proporcione (eficacia).
- Qué el programa sea eficiente, es decir, que resuelva el problema en un tiempo mínimo y con un uso óptimo de los recursos del sistema.

## INTRODUCCIÓN AL LENGUAJE JAVA

### HISTORIA

Java surgió en 1991, cuando un grupo de ingenieros de Sun Microsystems (empresa absorbida por Oracle), trataron de diseñar un nuevo lenguaje de programación destinado a programar pequeños dispositivos electrónicos (mandos a distancia, teléfonos inalámbricos), con la característica común de tener un software instalado en su interior que les dice como funcionar. La dificultad de estos dispositivos es que cambian continuamente, y para que un programa funcione en el siguiente dispositivo aparecido hay que rescribir el código. Por eso la empresa Sun quería crear un lenguaje cuyo código fuera pequeño, compacto y neutro respecto a la arquitectura, esto es, **independiente del dispositivo** (su principal característica).

Este lenguaje recibe primeramente el nombre de OAK (roble), debido a un roble que se veía desde la ventana del despacho de uno de los ingenieros, pero este nombre ya estaba registrado para otro lenguaje de programación. Finalmente lo registraron con el nombre de JAVA, debido a la cantidad de café que los ingenieros de Sun tomaban mientras trabajaban (en EEUU las tazas de loza donde se toma café se les denomina vulgarmente java).

Primeramente intentaron vender la tecnología a diferentes empresas de ámbito tecnológico pero fracasaron y el lenguaje quedo en stand-by.

En 1994 nació internet. Para conectarse a la red los usuarios utilizaban distintas plataformas (windows, linux...), lo que hacía que el lenguaje java se adaptara perfectamente a esa circunstancia por ser multiplataforma, por eso en 1995 se dio a conocer al público como lenguaje de programación para computadores. Java pasa a ser un lenguaje totalmente independiente de la plataforma y a la vez potente y orientado a objetos. Esa filosofía y su facilidad para crear aplicaciones para redes TCP/IP, ha hecho que sea uno de los lenguajes más utilizados en la actualidad.

El factor determinante para su expansión es que fue presentado conjuntamente con un navegador Web denominado HotJava, lo que supuso una gran revuelo en Internet.

### CARACTERÍSTICAS PRINCIPALES DE LENGUAJE JAVA

- ✓ Es neutro respecto a la arquitectura (multiplataforma). El código generado por el compilador Java es independiente de la arquitectura.
- ✓ Sencillo. Para facilitar su aprendizaje, su lectura y el mantenimiento de programas.
- ✓ Está totalmente orientado a objetos.
- ✓ Es distribuido, preparado para crear aplicaciones que trabajan en red.
- ✓ Dispone de un amplio conjunto de bibliotecas.
- ✓ Es robusto. No se interrumpe fácilmente a consecuencia de fallos, para ello, realiza comprobaciones del código en tiempo de compilación y de ejecución, así como manejar excepciones.
- ✓ La seguridad está garantizada, ya que las aplicaciones Java no acceden a zonas delicadas de memoria o del sistema.
- ✓ Interpretado. Ejecutado sobre cualquier plataforma, generando código máquina.
- ✓ Alto rendimiento. Interpreta el código en el momento de la ejecución, generando código máquina una sola vez, y en las sucesivas ejecuciones reutiliza dicho código máquina en lugar de volver a generarlo cada vez que se ejecuta el programa.

### TIPOS DE APLICACIONES EN JAVA

Java permite al programador crear los distintos tipos de aplicaciones:

- ✓ **Aplicaciones de consola:**

Son aquellas que leen y escriben hacia y desde la consola, sin ninguna interfaz gráfica de usuario.

- ✓ **Aplicaciones gráficas:**

Aquellas que utilizan las clases con capacidades gráficas como Swing, que es la biblioteca para la interfaz gráfica de usuario avanzada de la plataforma Java SE.

### ✓ **Applets:**

Son programas incrustados en otras aplicaciones, normalmente una página web que se muestra en un navegador. Cuando el navegador carga una web que contiene un applet, éste se descarga en el navegador web y comienza a ejecutarse. Esto nos permite crear programas que cualquier usuario puede ejecutar con tan solo cargar la página web en su navegador.

### ✓ **Servlets:**

Los servlets al contrario de los applets, son programas que están pensados para trabajar en un servidor. Nos permiten desarrollar aplicaciones Web que interactúen con los clientes, generando respuestas a las peticiones recibidas de los clientes.

### ✓ **Midlets:**

Son aplicaciones creadas en Java para su ejecución en dispositivos móviles.

## **PLATAFORMA DE JAVA**

Una plataforma es tanto el entorno hardware y/o software donde se ejecuta un programa.

### **Ejemplos de plataformas:**

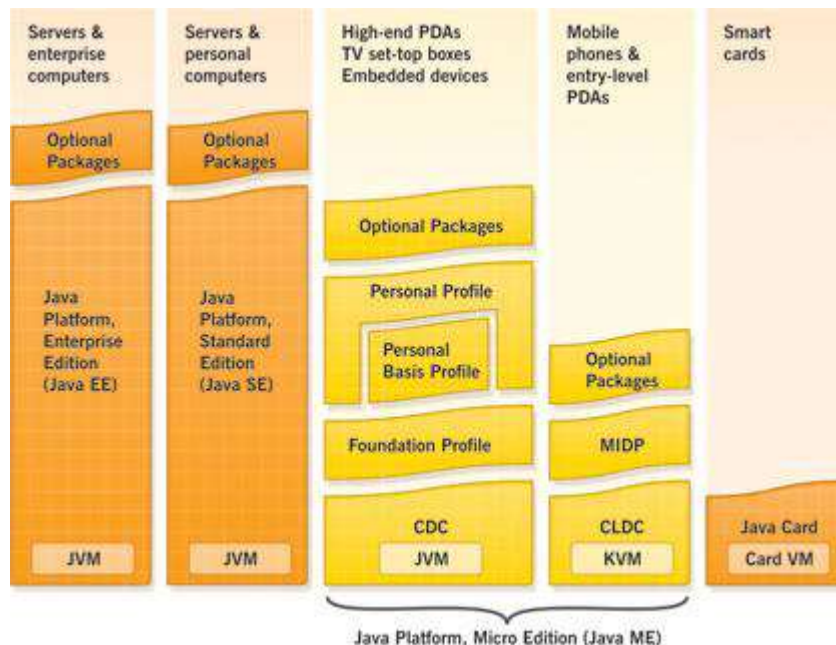
1. Plataformas Intel, RISC, SPARC...
2. Plataformas Windows, Linux, AIX, Solaris, HP-UX, z/OS...
3. Plataformas IBM, Sun, HP, Microsoft...
4. Windows+Intel=wintel

La plataforma Java es una plataforma software que se ejecuta sobre otra plataforma hardware/software.

Podemos hablar de ediciones de la plataforma java, dirigidas cada una de ellas a ayudar a los programadores en áreas de una especialización específica:

- **Java SE** (Java Standard Edition): enfocada a crear tanto aplicaciones web como aplicaciones de escritorio, es decir, está orientada a entornos de gama media y estaciones de trabajo, como por ejemplo PC de escritorio.
- **Java EE** (Java Enterprise Edition): enfocada a crear aplicaciones para servidores, por tanto está orientada a entornos empresariales distribuidos o internet. Es la edición más grande de java, contiene toda la edición estándar y mucho más.
- **Java ME** (Java Micro Edition): es una versión reducida de la edición estándar. Enfocada a la creación de aplicaciones tanto en dispositivos móviles como en dispositivos integrados (PDA, consolas de vídeo juegos, televisores inteligentes, etc), por tanto está orientada a entornos con recursos limitados.

- **Java Card:** enfocada para crear aplicaciones para tarjetas inteligentes y para dispositivos móviles a través de las tarjetas SIM.
- **Java FX:** más que una edición es una familia de tecnologías para la creación de aplicaciones de Internet Enriquecida (RIAS)



Una edición proporciona paquetes y clases que se van agregando a la que se conoce como edición estándar (edición original), para desarrollar las aplicaciones correspondientes.

En este caso la edición estándar es Java SE (su nombre ha ido cambiando: JDK, J2SE y Java SE). Es necesario tenerla instalada si se quieren utilizar las demás.

En nuestro caso vamos a desarrollar programas utilizando la edición estándar (Java SE)

### JAVA SE

Podemos distinguir dos partes:

#### 1. JRE (Java Runtime Environment o entorno de ejecución de java)

Se puede definir como el requerimiento mínimo de software que debe tener un equipo para poder ejecutar un programa en java.

Tiene los siguientes componentes:

- **La máquina virtual Java (JVM) o intérprete de Java**

Es un programa de reducido tamaño y gratuito para cada sistema operativo.

Para que un programa pueda ejecutarse hay que convertirlo a código máquina (binario). A este proceso se le denomina "compilar" y generalmente se lleva a cabo mediante un programa llamado compilador. Sin embargo el compilador de java funciona diferente. Revisa nuestro programa escrito en lenguaje java (código fuente con extensión .java) para que no tenga errores de sintaxis, cuando sintácticamente esté bien escrito genera un código binario específico denominado ByteCode (archivo con extensión .class). Este código binario no lo entiende el sistema operativo y por tanto no puede ser ejecutado directamente en nuestro ordenador, es necesario una Máquina Virtual de java capaz de entenderlo y traducirlo (interpretarlo) a código máquina nativo para poder ejecutarlo. De esta forma un programa se escribirá una sola vez y se podrá ejecutar en cualquier plataforma (Windows, Linux, Mac, etc), donde tengamos instalada la Máquina Virtual correspondiente (es por lo que se dice que java es un lenguaje compilado y multiplataforma).

### • La Interfaz de Programación de Aplicaciones (API)

Es lo que se conoce como Bibliotecas Java (código preescrito) . Está formada por:

- **Bibliotecas de integración:** para crear aplicaciones para BD, seguridad y comunicaciones.
  - **Bibliotecas base:** este conjunto de bibliotecas proporciona al programador paquetes de clases útiles, para la realización de múltiples tareas dentro de un programa. Está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente.
  - **Herramientas** para la interfaz con el usuario
- **Herramientas de despliegue**, que permitan correr applets en los navegadores así como ejecutar aplicaciones a través de una red.
  - **Otras herramientas** para la interfaz de usuario.

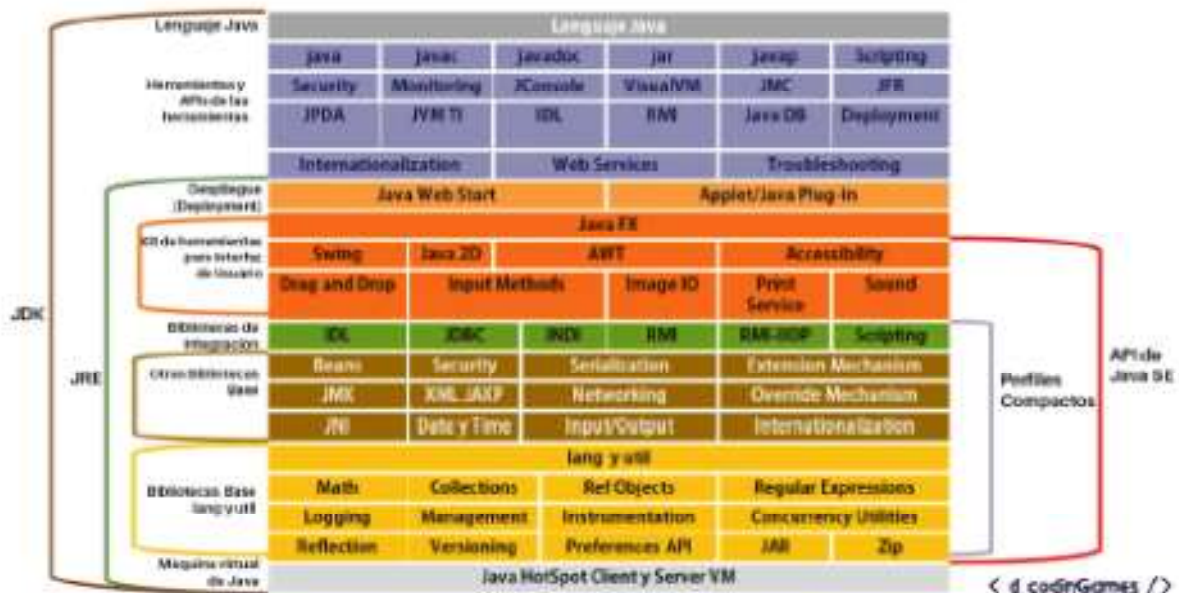
## 2. JDK (Java Development Kit o Kit de Desarrollo Java)

Es el paquete de herramientas que nos permite desarrollar aplicaciones en java. Es el entorno de desarrollo más básico y no visual. Entre otros tiene los siguientes componentes para la línea de comandos:

- **javac.exe:** compilador de java.
- **java.exe:** intérprete de java.
- **jdb.exe:** depurador.
- **javadoc.exe:** generador de documentación.



Al instalar un JDK, este siempre incluye una implementación del entorno de ejecución java (JRE) para ser utilizado por él. Por tanto, para poder desarrollar un programa en java sólo se necesitará un editor de textos plano (notepad, block de notas...) y los elementos propios del entorno de desarrollo.



## ENTORNO DE DESARROLLO INTEGRADO (IDEs)

Son aplicaciones que ofrecen la posibilidad de llevar a cabo el proceso completo de desarrollo de software a través de un único programa. Podemos realizar las labores de edición, compilación, depuración y ejecución de programas escritos en Java o en otros lenguajes de programación bajo un entorno gráfico (no mediante línea de comandos). Junto a las capacidades descritas cada entorno añade otras que ayudan a realizar el proceso de programación, como por ejemplo: código fuente coloreado, plantillas para diferentes tipos de aplicaciones, creación de proyectos, etc.

Entre los IDEs actuales, los más utilizados por los programadores de java son: Eclipse, NetBeans y JCreator.



## 2.- ESTRUCTURA Y BLOQUES FUNDAMENTALES

Un programa en Java tiene la siguiente estructura básica.

**package** ejemplo;

```
public class PrimerPrograma {  
    public static void main(String[] args) {  
        System.out.println("hola mundo");  
    }  
}
```

La salida del programa anterior será, **hola mundo**

Todos los programas han de incluir una clase general en la que se incluyen todos los demás elementos del programa. Entre otras cosas contiene el método o función `main()`, que representa al programa principal desde el que se llevará a cabo la ejecución del programa.

Esta clase puede contener a su vez otras clases del usuario, pero sólo una puede ser `public`. El nombre del fichero `.Java` que contiene el código fuente de nuestro programa, coincidirá con el nombre de esta clase general.

**Java distingue entre mayúsculas y minúsculas.** Si le damos a la clase general el nombre `PrimerPrograma`, el archivo `.Java` tendrá como identificador **`PrimerPrograma.Java`**, que es totalmente diferente a `primerprograma.Java`.

**`public static void main (String[ ] args):`** Es el método que representa al programa principal, en él se podrán incluir las instrucciones que estimemos oportunas para la ejecución del programa. Desde él se podrá hacer uso del resto de clases creadas. Todos los programas Java tienen un método `main`.

## 3. IDENTIFICADORES

Un identificador es un nombre que damos en un programa, a una clase, a un método, a una variable o constante, sea del tipo que sea. El lenguaje java utiliza el conjunto de caracteres Unicode que incluye no solamente el conjunto de caracteres ASCII, sino también caracteres específicos de la mayoría de los alfabetos, como la letra ñ.

Todos los lenguajes tienen ciertas **reglas para componer los identificadores**:

- Todos los identificadores han de comenzar con una letra, el carácter subrayado ( `_` ) o el carácter dólar ( `$` ).
- Puede incluir, pero no comenzar por un número.
- No puede incluir el carácter espacio en blanco.

- Distingue entre letras mayúsculas y minúsculas.
- No se pueden utilizar las palabras reservadas como identificadores.
- Pueden ser de cualquier longitud.

Además de estas restricciones hay ciertas convenciones que hacen que el programa sea más legible, pero que no afectan a la ejecución del mismo. La primera y fundamental es la de encontrar un nombre que sea significativo. El tiempo que se pretende ahorrar eligiendo nombres cortos y poco significativos se pierde con creces cuando se revisa el programa después de cierto tiempo.

Tipo de identificador	Convención	Ejemplo
nombre de una clase	Comienza por letra mayúscula	String, Rectangulo, CinematicaApplet
nombre de función	Comienza con letra minúscula	calcularArea, getValue, setColor
nombre de variable	Comienza por letra minúscula, suelen ser sustantivos.	area, color, appletSize
nombre de constante	En letras mayúsculas, suelen ser sustantivos.	PI, MAX_ANCHO

**Los siguientes identificadores son válidos:** variable, \$variable2, CONSTANTE, nombre\_usuario, nombreUsuario, \_variable\_sistema

**Los siguientes identificadores no son válidos:** 1variable, int, #variable, variable%Final

## 4.-TIPOS DE DATOS

Son el conjunto de valores a los cuales puede pertenecer una constante o donde puede tomar valor una variable. Los proporciona el propio lenguaje.

Son necesarios para que el compilador conozca de antemano el tipo de información que va a contener una variable de memoria.

En Java existen dos tipos de datos genéricos:

### 1. Tipos Primitivos

Existen ocho tipos de datos primitivos clasificados en cuatro grupos diferentes:

- **Lógico:** boolean.
- **Carácter:** char.
- **Números enteros:** byte, short, int y long.
- **Números reales:** double y float.

## 2. Tipos Complejos o clases

Existe un caso especial que es el de enumeración: enum.

### TIPOS PRIMITIVOS

Tipos Primitivos	Descripción
<b>boolean</b>	Tiene dos valores <b>true</b> o <b>false</b> . <b>Ejm:</b> boolean encontrado=false;
<b>char</b>	Caracteres Unicode de 16 bits (2 bytes). Los caracteres alfanuméricos 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). Un carácter está siempre rodeado de comillas simples. <b>Ejm:</b> char letra='a'; <b>Ejm:</b> char caracter; Su valor por defecto es null, no espacio en blanco.
<b>byte</b>	Tamaño 8 bits (1 byte). El intervalo de valores va desde $-2^7$ hasta $2^7 - 1$ (-128 a 127). <b>Ejm:</b> byte edad=20;
<b>short</b>	Tamaño 16 bits (2 bytes). El intervalo de valores va desde $-2^{15}$ hasta $2^{15} - 1$ (-32768 a 32767). <b>Ejm:</b> short paga=28000;
<b>int</b>	Tamaño 32 bits (4 bytes). El intervalo de valores va desde $-2^{31}$ hasta $2^{31} - 1$ (-2147483648 a 2147483647). Java toma por defecto los números enteros como int. <b>Ejm:</b> int numero=24; <b>Ejm:</b> int num; //su valor por defecto es 0.
<b>long</b>	Tamaño 64 bits (8 bytes). El intervalo de valores va desde $-2^{63}$ hasta $2^{63} - 1$ (-9223372036854775808 a 9223372036854775807). <b>Ejm:</b> long m=30L; Cuando asignamos un valor de tipo long a una variable debemos añadir al final de este valor el modificador L, puesto que sino el valor numérico se toma por defecto como int y el compilador da un error.

<b>float</b>	<p>Tamaño 32 bits (4 bytes).</p> <p>Números reales de simple precisión. Va de 1.40239846e-45f a 3.40282347e+38f.</p> <p>La parte entera se separa de la decimal por punto en lugar de coma.</p> <p>Por defecto el compilador toma las variables decimales como tipo double, por ello cuando una variable decimal es de tipo float, al asignarle su valor debemos especificar el modificador f al final del valor.</p> <p><b>Ejm:</b> float precio=12.5f;</p> <p><b>Ejm:</b> float dinero; //su valor por defecto es 0.0.</p>
<b>double</b>	<p>Tamaño 64 bits (8 bytes).</p> <p>Números decimales de doble precisión.</p> <p>Va de 4.94065645841246544e-324d a 1.7976931348623157e+308d.</p> <p>Java toma por defecto los números decimales como double.</p> <p><b>Ejm:</b> double c= 5.6;</p> <p><b>Ejm:</b> double prima=66.20d //no haría falta poner el modificador d, es redundante.</p>

### SECUENCIA DE ESCAPE (tipo char)

Un tipo especial de carácter es la secuencia de escape. Se utiliza para representar caracteres de control o caracteres que no se imprimen. Una secuencia de escape está formada por la barra invertida (\) y un carácter. En la siguiente tabla se muestran las secuencias de escape más utilizadas.

Carácter	Secuencia de escape
Retorno de carro	\r
Tabulador horizontal	\t
Espacio hacia atrás	\b
Nueva línea	\n
Barra invertida	\\
Comilla simple	\'
Comilla doble	\"

**Ejm:** System.out.println ("Un mensaje con \t un carácter tabulador y \n un salto de línea");

La secuencia de escape nos da también la posibilidad de imprimir caracteres que no podemos conseguir con facilidad en nuestro teclado. Podemos definir cualquier carácter de codificación Unicode de la siguiente forma: ("uxxx"), donde xxx es el valor hexadecimal del carácter Unicode.

### TIPOS COMPLEJOS O CLASES

#### 1. Cadenas de caracteres

Las cadenas de caracteres o String en Java son objetos de la clase *String*.

Ejm:

```
public class PrimeroApp{
    public static void main(String[] args) {
//imprime un mensaje
        String mensaje="El primer programa";
        System.out.println(mensaje);
    }
}
```

#### 2. Tipo enumerado

Se trata de un tipo de dato complejo algo especial que surge con la versión 5.0 de Java. Implementa una clase que tiene un atributo que puede tomar varios valores y solo esos.

**Ejemplo:** enum Semaforo { VERDE, AMBAR, ROJO } /\* Por convenio los valores se escriben todos en mayúscula\*/.

Se suelen utilizar para tener una lista de posibles valores asociados a una variable y solamente dichos valores.

Ejm: Semaforo estadoSemaforo= Semaforo.ROJO

### 5.-CONVERSIONES DE TIPO

Es lo que se conoce como **casting**, refiriéndose a "*colocar un molde*".

Un tipo de datos numérico puede llegar a convertirse en otro tipo, por ejemplo cuando se mezclan variables de un tipo con variables de otro tipo en expresiones aritméticas, en sentencias de asignación, en llamadas a métodos con paso de parámetros, etc.

Podemos hablar de dos tipos de conversiones: implícita y explícita.

#### Conversión implícita

Se resuelve en tiempo de compilación. En este caso los tipos más pequeños se convierten en los de tipo más grandes de forma automática.

- | char  
| byte (se añaden ceros por la izquierda)  
| short  
| int  
| long  
| float  
+▼ double

**Ejm:** float valor=3; //El compilador convierte automáticamente el int a float añadiendo un cero decimal.

**Ejm:** long l = 42; // un int se convierte en long utilizando 8 bytes para su representación.

### Conversión explícita

Se resuelve en tiempo de ejecución.

Se utiliza cuando queremos convertir un tipo de datos más grande a otro más pequeño. En estos casos se puede perder información.

Para realizar una conversión explícita basta con indicar el tipo de datos al que se desea convertir entre paréntesis, antes del valor.

**Ejm:** int i = (int) 42L //Un long se convierte en int.

**Ejm:** char c= (char) 67; //Convierte el 67 al carácter correspondiente en ASCII

**Ejm:**

```
short sueldoBase=1980;
```

```
short complementos=400;
```

```
short sueldoTotal= (short) (sueldoBase +complementos);
```

En este caso hay que hacer la conversión pues la suma trabaja como int, por tanto java hace una conversión implícita de short a int en sueldoBase y complementos. El casting lo que hace es convertir el resultado de la suma (int) a short, que es el tipo de sueldoTotal.

Java permite el moldeo de cualquier tipo primitivo en otro tipo primitivo excepto en el caso de los booleanos, en que no se permite moldeo alguno. Tampoco se permite el moldeo de clases. La clase **String** es un caso especial que se verá más adelante.

## 6.-VARIABLES

Una variable es un nombre que se asocia con una porción de memoria del ordenador en la que se guarda el valor asignado a dicha variable. Los valores de las variables pueden ir variando a lo largo del programa.

Todas las variables han de declararse antes de usarlas. La declaración consiste en una sentencia en la que figura el tipo de dato (primitivo, nombre de una clase, array) y el nombre que asignamos a la variable.

Los nombres asignados deben ser significativos. Si consta de varias palabras la primera empieza por minúscula, pero las que le siguen llevan la letra inicial en mayúscula. Se deben evitar en todos los casos nombres de variables cortos como i, x, etc.

**Ejm:** int contador;

Una vez declarada se le podrá asignar un valor mediante una sentencia de asignación (inicialización). Dependiendo del tipo de dato de la variable estas requieren más o menos cantidad de memoria para guardar sus valores.

**Ejm:** int contador = 10;

Por tanto, **una variable tiene:**

- Un tipo
- Un nombre
- Un dato o valor

### TIPOS DE VARIABLES DE JAVA

Java tiene tres tipos de variables:

- **De instancia**

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

- **De clase**

Las variables de clase o miembros de 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;  
    //...  
}
```

### • Locales

Las variables locales se utilizan dentro de los 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*.

```
class Circulo{  
    double calcularArea() {  
        double area=PI*radio*radio;  
        return area;  
    }  
}
```

Las variables locales se deben inicializar en el momento en que son declaradas. Si se usan las variables sin que tengan un valor previo daría un error de compilación.

Las variables de instancia y clase si no se inicializan se les asigna sus valores por defecto de forma automática.

## ÁMBITO DE LAS VARIABLES EN JAVA

El ámbito de una variable es la zona de código donde se puede utilizar dicha variable.

El lugar donde se declara una variable establece su ámbito.

Los distintos ámbitos existentes son:

### • Las clases

Las variables de instancia y clase (atributos) se pueden utilizar solamente asociadas a la clase a la que pertenecen.

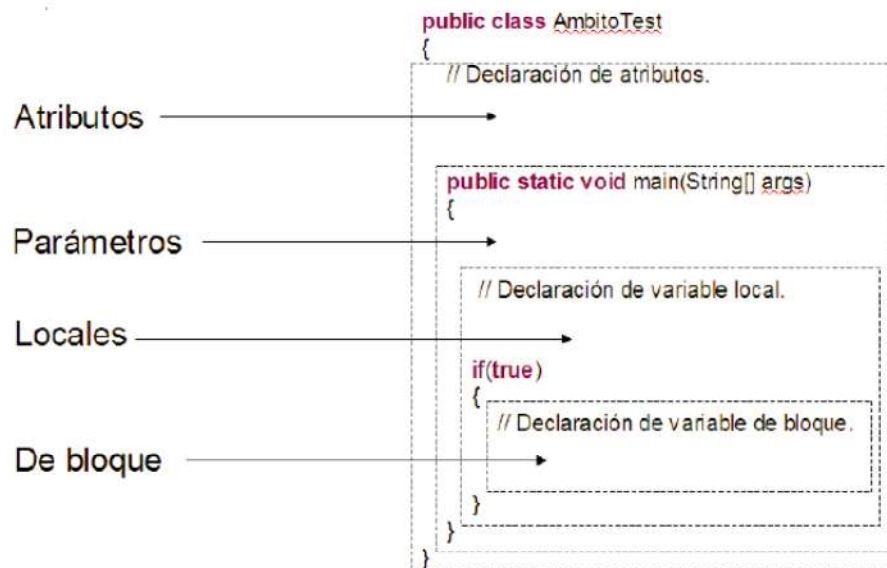
### • Los métodos

Los parámetros del método se pueden utilizar solamente en el método del cual son parámetros.

Las variables locales de un método, solamente se puede acceder a ellas dentro del método donde han sido declaradas. Estas variables deben inicializarse siempre.

### • Los bloques

Las variables de bloque, solamente puede accederse a ellas dentro del bloque donde han sido declaradas. Deben inicializarse siempre.



En el caso de que se declaren variables con el mismo identificador en ámbitos distintos (en el mismo no se puede), tienen preferencia las del ámbito más interno.

El siguiente ejemplo muestra el error de compilación al intentar referenciar una variable fuera de su ámbito.

```
public class AmbitoTest2
{
    public static void main(String[] args)
    {
        if(true)
        {
            int i = 12;
        }
        System.out.println("El valor de i es: " + i);
    }
}
```

Tasks (1 item)					
✓	!	Description	Resource	In Folder	Location
✗		i cannot be resolved	AmbitoTest2.java		line 9
Tasks Console Debug Search Synchronize					

El siguiente ejemplo muestra cuál es la variable utilizada según el ámbito en el que está declarada.

```
public class AmbitoI est3
{
    static int i = 5;

    public static void main(String[] args)
    {
        int i = 10;
        System.out.println("El valor de i es: " + i);
    }
}
```



En resumen, las variables son uno de los elementos básicos de un programa y se deben:

- Declarar.
- Inicializar.
- Usar.

## 7.- CONSTANTES

Son los identificadores cuyos valores no varían a lo largo del programa.

Las constantes deben declararse y darles un valor antes de usarlas.

La declaración consiste en una sentencia en la que figura la palabra reservada `final`, el tipo de dato y el nombre que asignamos a la constante que debe ser significativo.

Normalmente las constantes de un programa se suelen poner en letras mayúsculas, para distinguirlas de las que no son constantes.

**Ejm:**

```
final double PI=3.141592653589793;
final int MAX_DATOS=150;
```

### 8.-SENTENCIAS

Una sentencia es una instrucción que se le da al programa para realizar una tarea específica como: mostrar un mensaje en la pantalla, declarar una variable (para reservar espacio en memoria), inicializarla, llamar a una función, etc.

Las sentencias acaban con ;. Este carácter separa una sentencia de la siguiente.

Normalmente las sentencias se ponen unas debajo de otras empleando tabuladores para favorecer la legibilidad de un programa, aunque sentencias cortas pueden colocarse en una misma línea.

De la misma forma se favorece la legibilidad de un programa mediante el uso de espacios en blanco entre elementos del código fuente.

Ejm:

```
int i=1;
import java.awt.*;
System.out.println("El primer programa");
```

### 9.-BLOQUES DE CÓDIGO

Un bloque de código es un conjunto de sentencias agrupadas entre llaves { }.

Como ejemplos de bloques de código tenemos la definición de una clase, de un método, una sentencia iterativa for, los bloques try ... catch para el tratamiento de las excepciones, etc.

Los bloques de código pueden estar anidados.

Ejm:

```
public class Principal {

    public static void main(String[] args) {
        NuevaClase clase=new NuevaClase();
        clase.crearObjetos();
        clase.imprimir();
    }

}
```



El operador más (+) se puede utilizar para concatenar cadenas como se observa en el ejemplo siguiente:

```
System.out.println ("miVariable tiene el valor " + miVariable + " en este programa");
```

Esta operación hace que se tome el valor que tiene miVariable para su uso en la expresión. De ninguna forma se altera el valor que contiene la variable.

### Los operadores unarios que soporta Java son:

- + Indica un valor positivo o convierte al operando en int en caso de que fuese byte, char o short.
- Indica un valor negativo o cambia el signo algebraico.
- ++ Suma 1 al operando (incremento en 1).
- Resta 1 al operando (decremento en 1).

Los operadores de incremento y decremento pueden estar situados como prefijos o como sufijos. La diferencia entre estas versiones es el momento en el tiempo en que se realiza el incremento o decremento.

#### Ejm:

```
int sumar=2;  
int resultado = ++sumar;
```

En este caso primero se incrementa el valor de sumar, y luego ese valor se asigna a la variable resultado. Es decir, primero incrementa y luego se asigna.

La variable resultado sería igual a 3.

#### Ejm:

```
int sumar=2;  
int resultado =sumar++;
```

En este caso primero se asigna el valor de sumar a resultado, y luego se incrementa en uno sumar. Es decir, primero se asigna y luego se incrementa.

La variable resultado sería igual a 2.

#### Ejm:

```
int contador=1;  
contador++; // Es lo mismo que poner contador=contador+1;
```

En este caso se suma uno a la propia variable contador que se queda con el valor incrementado.

La variable contador sería igual a dos.

### Operadores Relacionales y Condicionales

Los operadores relacionales en Java devuelven un tipo booleano, *true* o *false*. Solo se pueden utilizar para datos primitivos no para objetos.

>	El operando izquierdo es mayor que el derecho.
>=	El operando izquierdo es mayor o igual que el derecho.
<	El operando izquierdo es menor que el derecho.
<=	El operando izquierdo es menor o igual que el derecho.
==	El operando izquierdo es igual que el derecho.
!=	El operando izquierdo es distinto del derecho.

Ejm:

```
if (5<7) {-----}
```

**Los operadores condicionales que soporta Java son:**

&&	El resultado es verdad si todas las condiciones son verdad.
	El resultado es verdad si alguna de las condiciones es verdad.
!	Niega la condición.

Ejm: `if (5<7 && 4<8)`

Los operadores relacionales combinados con los operadores condicionales se utilizan para obtener expresiones más complejas.

Una característica importante del funcionamiento de los operadores && y ||, es que las expresiones se evalúan de izquierda a derecha y que la evaluación de la expresión finaliza tan pronto como se pueda determinar el valor de la expresión.

En la expresión siguiente:

```
if ( ( a < b ) || ( c < d ) )
```

Si la variable *a* es menor que la variable *b*, no hay necesidad de evaluar el operando derecho del operador || para determinar el valor de la expresión entera.

### Operadores a Nivel de Bits

Java tiene un conjunto de operadores que realizan operaciones sobre un solo bit cada vez.

**Los operadores de bits que soporta Java son los que aparecen en la siguiente tabla:**



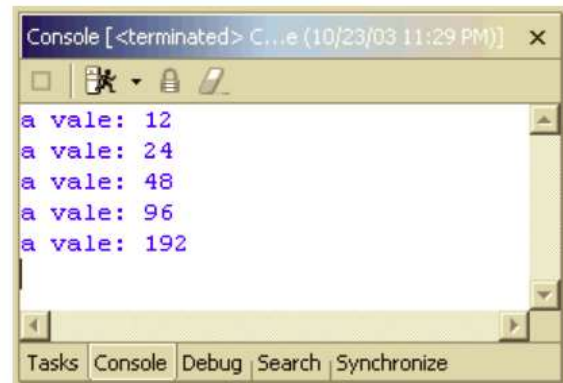
Operador	Uso	Operación
>>	Operando >> Desplazamiento	Desplaza los bits del operando hacia la derecha tantas veces como indique el segundo operando. Tiene en cuenta el signo. Realiza la misma función que dividir entre 2 tantas veces se desplace.
<<	Operando << Desplazamiento	Desplaza los bits del operando hacia la izquierda tantas veces como indique el segundo operando. Realiza la misma función que multiplicar por 2 tantas veces se desplace.
>>>	Operando >>> Desplazamiento	Desplaza los bits del operando hacia la derecha tantas veces como indique el segundo operando. Realiza la misma función que dividir entre 2 tantas veces se desplace, pero sin tener en cuenta la posición del signo.
&	Operando & Operando	Realiza una operación AND lógico a nivel de bit entre los dos operandos. El resultado es 1 si los bits de ambos operandos son 1.
	Operando   Operando	Realiza una operación OR lógico a nivel de bit entre los dos operandos. El resultado es 1 si alguno de los bits de los operandos es 1.
^	Operando ^ Operando	Realiza una operación lógica OR Exclusiva entre los dos operandos (XOR). Si los bit que se comparan son iguales el resultado es 0. Si uno de los bit que se compara es 0 y el otro 1 el resultado es 1.
~	~ Operando (NOT)	Complementario del operando. Cambia los ceros por unos y viceversa.

En Java el operador de desplazamiento hacia la derecha sin signo, rellena los bits que pueden quedar vacíos con ceros. Los bits que son desplazados fuera del entorno se pierden.

En el desplazamiento a la izquierda hay que ser precavidos cuando se trata de desplazar enteros positivos pequeños, porque el desplazamiento a la izquierda tiene el efecto de multiplicar por 2 para cada posición de bit que se desplace. Esto es peligroso porque si se desplaza un bit 1 a la posición más alta (bit 31 o 63), el valor se convertirá en negativo.

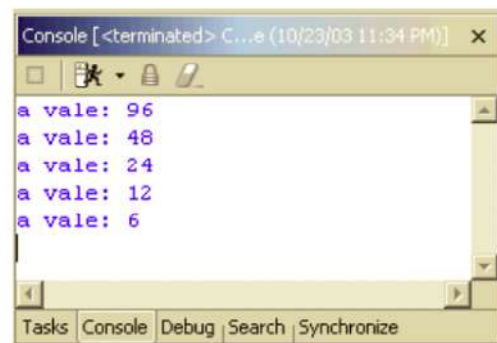
Ejm:

```
public class Multiplicador
{
    public static void main(String[] args)
    {
        int a = 6;
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
        a = a << 1;
        System.out.println("a vale: " + a);
    }
}
```



Ejm:

```
public class Dividor
{
    public static void main(String[] args)
    {
        int a = 192;
        a = a >> 1;
        System.out.println("a vale: " + a);
        a = a >> 1;
        System.out.println("a vale: " + a);
        a = a >> 1;
        System.out.println("a vale: " + a);
        a = a >> 1;
        System.out.println("a vale: " + a);
        a = a >> 1;
        System.out.println("a vale: " + a);
        a = a >> 1;
        System.out.println("a vale: " + a);
    }
}
```



Ejm:

```
int op1=3; // 00000011
```

```
int op2=2; // 00000010
```

```
int resultado= op1&op2 // El resultado es el número 2 (0000010)
```

### Operadores de Asignación

= es un operador binario de asignación de valores. El valor de la derecha es copiado a la variable situada a la izquierda.

`+= -= *= /= %= &= |= ^=`

**Ejm:**

```
int x=2;
```

```
int y=3;
```

`x += y;` // Guarda la suma de los dos operandos en el primero. El resultado sería: `x=5, y=3`. Es lo mismo que poner: `x = x + y;`

**Ejm:**

```
int x=3;
```

```
int y=2;
```

`x -= y;` /\* Guarda la resta de los dos operandos en el primero. El resultado sería: `x=1, y=2`. Es lo mismo que poner: `x = x - y;` \*/

### Operador ternario *if-else*

La forma general del operador es:

`expresión ? sentencia1 : sentencia2`

La variable expresión puede ser cualquier expresión de la que se obtenga como resultado un valor booleano. Si es *true* se ejecuta la sentencia1, en caso contrario se ejecuta la sentencia2. La limitación que impone el operador es que sentencia1 y sentencia2 deben devolver el mismo tipo de datos y no puede ser void.

**Ejm:**

```
cociente = numerador == 0 ? 0 : numerador / denominador
```

Cuando Java evalúa la asignación, primero mira la expresión que está a la izquierda del interrogante. Si numerador es cero entonces evalúa la expresión que está entre el interrogante y los dos puntos y se asigna el resultado a cociente. Si numerador no es cero entonces evalúa la expresión que está después de los dos puntos y se asigna el resultado a cociente.

Ejm:

```
public class java402 {  
    public static void main( String args[] ) {  
        int a = 28;  
        int b = 4;  
        int e = (a == 0) ? 0 : (a / b);  
        System.out.println( "e = " + e );  
    }  
}
```

El programa se ejecuta sin errores y la salida que genera por pantalla es: **e=7**

## 11.-COMENTARIOS

Un comentario es un texto adicional que se añade al código para explicar su funcionalidad a otras personas que lean el programa o al propio autor como recordatorio.

Los comentarios son una parte importante de la documentación de un programa. Son ignorados por el compilador por lo que no incrementan el tamaño del archivo ejecutable. Se pueden por tanto añadir libremente al código para que pueda entenderse mejor aunque tampoco se puede abusar de ellos.

**En Java existen dos tipos de comentarios:**

### 1) Comentarios de implementación

Se suelen poner dentro de los métodos y clases.

Existen dos formas distintas de escribir estos comentarios:

#### a) Comentario de una sola línea (//)

Abarca desde el comienzo del comentario // hasta el final de línea.

#### b) Comentario de una o más líneas (/\* \*/)

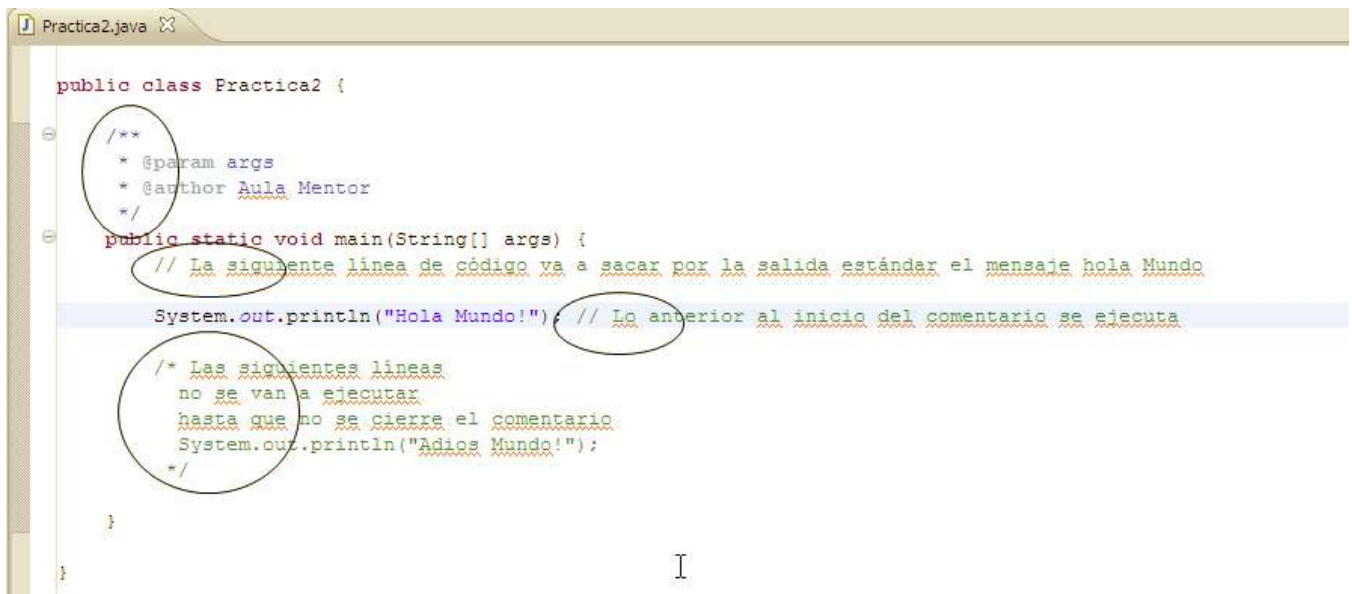
Abarca desde el comienzo del comentario /\* hasta el final del mismo \*/. Cuando abarca más de una línea se denominan comentarios de bloque.

### 2) Comentarios de documentación (/\*\* \*/)

Utilizado por la herramienta javadoc.exe para generar la documentación del código en formato html. Abarca desde el comienzo del comentario /\*\* hasta el final del mismo \*/.

Se suelen poner antes de la declaración de las clases y de la declaración de los métodos.

Ejm:



```
public class Practica2 {  
    /**  
     * @param args  
     * @author Aula Mentor  
     */  
    public static void main(String[] args) {  
        // La siguiente línea de código va a sacar por la salida estándar el mensaje hola Mundo  
        System.out.println("Hola Mundo!"); // Lo anterior al inicio del comentario se ejecuta  
        /* Las siguientes líneas  
         no se van a ejecutar  
         hasta que no se cierre el comentario  
        System.out.println("Adios Mundo!");  
        */  
    }  
}
```

## 12.-PALABRAS RESERVADAS

En el siguiente cuadro se listan las palabras reservadas, aquellas que emplea el lenguaje Java y que el programador no puede utilizar como **identificadores**.

**\*abstract \*double \*int \*strictfp \*\* \*boolean \*else \*interface \*super \*break \*extends \*long \*switch \*byte \*final \*native \*synchronized \*case \*finally \*new \*this \*catch \*float \*package \*throw \*char \*for \*private \*throws \*class \*goto \*protected \*transient \*const \* if \*public\* try \*continue \*implements \*return \*void \*default \*import \*short \*volatile \*do \*instanceof \*static \*while**

Las palabras reservadas se pueden clasificar en las siguientes categorías:

- Tipos de datos: **boolean, float, double, int, char.**
- Sentencias condicionales: **if, else, switch.**
- Sentencias iterativas: **for, do, while, continue.**
- Tratamiento de las excepciones: **try, catch, finally, throw.**
- Estructura de datos: **class, interface, implements, extends.**
- Modificadores y control de acceso: **public, private, protected, transient.**
- Otras: **super, null, this.**

## 13.-PREGUNTAS MÁS FRECUENTES

- **¿Cómo se escriben las sentencias en un programa sencillo?** Si queremos que un programa sencillo realice instrucciones o sentencias para obtener un determinado resultado, es necesario colocar éstas una detrás de otra exactamente en el orden en que deben ejecutarse.
- **¿Podrían colocarse todas las sentencias una detrás de otra separadas por puntos y comas en una misma línea?** Claro que sí, pero no es muy recomendable. Cada sentencia debe estar escrita en una línea, de esta manera el código será mucho más legible y la localización de errores de los programas será más rápida. De hecho cuando se utilizan herramientas de programación los errores suelen asociarse a un número o números de línea.
- **¿Puede una misma sentencia ocupar varias líneas en el programa?** Sí. Existen sentencias que por su tamaño pueden generar varias líneas, pero siempre finalizarán con un punto y coma.
- **¿En Java todas las sentencias se terminan con punto y coma?** Efectivamente. Si detrás de una sentencia ha de venir otra pondremos un punto y coma escribiendo la siguiente sentencia en una nueva línea. Pero en algunas ocasiones sobre todo cuando utilizamos estructuras de control de flujo, detrás de la cabecera de una estructura de este tipo no debe colocarse punto y coma.
- **¿Qué es la sentencia nula en Java?** La sentencia nula es una línea que no contiene ninguna instrucción y en la que sólo existe un punto y coma. Como su nombre indica esta sentencia no hace nada.
- **¿Qué es un bloque de sentencias?** Es un conjunto de sentencias que se encierran entre llaves y que se ejecutan como si fueran una única orden. Sirve para agrupar sentencias y para clarificar el código. Los bloques de sentencias son utilizados en Java en la práctica totalidad de estructuras de control de flujo, clases, métodos, etc. La siguiente tabla muestra dos formas de construir un bloque de sentencias.

### Bloque de sentencias 1

```
{sentencia1; sentencia2;...; sentencia N;}
```

### Bloque de sentencias 2

```
{  
sentencia1;  
sentencia2;  
...;  
sentenciaN;  
}
```

- **¿En un bloque de sentencias éstas deben estar colocadas con un orden exacto?**  
En ciertos casos sí, aunque si al final de su ejecución se obtiene el mismo resultado podrían ocupar diferentes posiciones en nuestro programa.