# 1 Elementos de un programa informático

## OBJETIVOS DEL CAPÍTULO

- ✓ Conocer qué es un programa, un lenguaje de programación y las diferencias entre lenguajes de programación como Java y C o C++.
- ✓ Reconocer el aspecto de un programa básico en Java y sus características principales.
- ✓ Instalar y utilizar un IDE.
- ✓ Compilar y ejecutar programas sencillos en Java dentro y fuera de un Entorno de desarrollo.
- ✓ Conocer y utilizar fundamentos básicos del lenguaje Java como los tipos de datos, constantes, literales, variables, comentarios, operadores y expresiones.
- ✓ Identificar las ventajas y limitaciones de Java frente a otros lenguajes de programación.

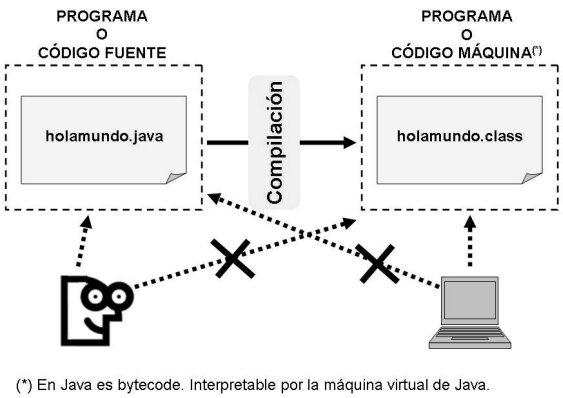
La información de este capítulo muchas veces es un resumen y en ocasiones no trata en profundidad ciertos aspectos. No obstante, el alumno en la sección de bibliografía puede encontrar libros y páginas aconsejadas en los que puede ampliar o contrastar la información en este libro proporcionada.

## 1.1 PROGRAMA Y LENGUAJES DE PROGRAMACIÓN

**Definición de programa**
Un programa es una serie de órdenes o instrucciones ordenadas con una finalidad concreta que realizan una función determinada.

Todo el mundo estamos familiarizados con la ejecución de programas (editores de textos, navegadores, juegos, reproductores de música o películas, etc.). Por regla general, cuando queremos ejecutar un programa se lo indicamos al sistema haciendo doble click sobre él e incluso algunos usuarios más avanzados ejecutan comandos desde un intérprete de comandos o consola. Si una vez has tenido la curiosidad de abrir un programa con un bloc de notas o editor de texto te habrás dado cuenta que aparece algo horrible en el editor, una serie de símbolos ininteligibles (por los humanos). Eso es porque los programas están en binario, que es el lenguaje que entienden las máquinas. Entonces te preguntarás: si al final de este libro seré capaz de escribir programas, ¿podré entender esos códigos? La respuesta es No. En este libro vamos a aprender un lenguaje de programación para escribir programas de manera entendible por los humanos que luego traduciremos al lenguaje máquina entendible por los ordenadores mediante otros programas llamados intérpretes o compiladores.

En la siguiente figura se verá todo esto de modo más gráfico:




(*) En Java es bytecode. Interpretable por la máquina virtual de Java.

***Figura 1.1.*** *Programas en código fuente y máquina*

Como se puede observar, el código fuente es el que escribe el programador que luego lo compila a código máquina. Compilar equivale a transformar el programa inteligible por el programador al programa inteligible por la máquina. El código fuente o programa fuente está escrito en un lenguaje de programación y el compilador es un programa que se encarga de transformar el código fuente en código máquina.

Los compiladores son programas específicos para un lenguaje de programación, los cuales transforman el programa fuente en un programa directa o indirectamente ejecutable por la máquina destino. No es posible compilar un programa escrito en lenguaje Java con un compilador de C porque éste no lo entendería.

El lenguaje máquina que genera Java es un lenguaje intermedio interpretable por una máquina virtual instalada en el ordenador donde se va a ejecutar. Una máquina virtual es una máquina ficticia que traduce las instrucciones máquina ficticias en instrucciones para la máquina real. La ventaja de la misma es que los programas se pueden ejecutar en cualquier tipo de hardware siempre y cuando tenga instalada la máquina virtual correspondiente. Los programas no van a cambiar, lo que cambiará es la máquina virtual dependiendo del hardware (no será igual la máquina virtual de un smartphone que la de un PC).

**Compiladores e Intérpretes**

A diferencia de los compiladores, los intérpretes leen línea a línea el código fuente y lo ejecutan. Este proceso es muy lento y requiere tener cargado en memoria el intérprete. La ventaja de los intérpretes es que la depuración y corrección de errores del programa es mucho más sencilla que con los compiladores.

### 1.1.1 EL LENGUAJE JAVA

Java es uno de los lenguajes más utilizados en la actualidad. Es un lenguaje de propósito general y su éxito radica en que es el lenguaje de Internet. *Applets*, *Servlets*, páginas JSP o JavaScript utilizan Java como lenguaje de programación.

El éxito de Java radica en que es un lenguaje multiplataforma. Java utiliza una máquina virtual en el sistema destino y por lo tanto no hace falta recompilar de nuevo las aplicaciones para cada sistema operativo. Java, por lo tanto, es un lenguaje interpretado que para mayor eficiencia utiliza un código intermedio (*bytecode*). Este código intermedio o *bytecode* es independiente de la arquitectura y por lo tanto puede ser ejecutado en cualquier sistema.

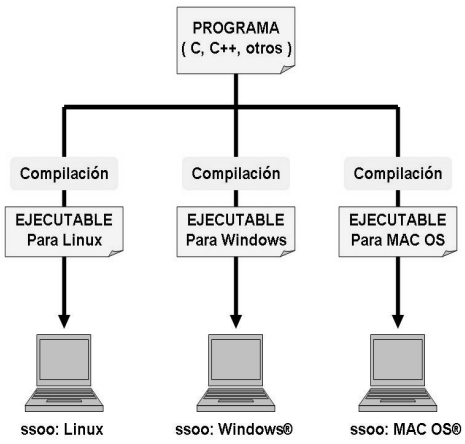



***Figura 1.2.*** *Recompilación del programa para cada sistema operativo*

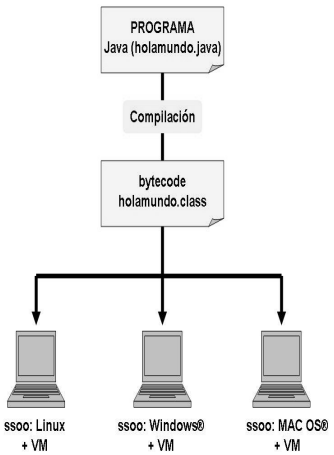



***Figura 1.3.*** *En Java una única compilación*

Como puede apreciarse en las figuras anteriores, en Java, una vez compilado el programa, se puede ejecutar en cualquier plataforma solamente con tener instalada la máquina virtual (*Virtual Machine* – VM) de Java. Sin embargo en C, C++ u otro lenguaje, deberemos recompilar el programa para el sistema destino con la consiguiente pérdida de flexibilidad.

Por lo tanto, Java es un compilador y a la vez un intérprete. El compilador compila a bytecode y el intérprete se encargará de ejecutar ese código intermedio en la máquina real.

**Recuerda**

Java es multiplataforma y programas en Java pueden ser ejecutados en Windows®, GNU/Linux y Mac OS X entre otros sistemas.

James Gosling trabajaba para Sun Microsystems® y fue el diseñador de Java en 1990. El primer nombre que tuvo Java fue OAK y tuvo como referentes C y C++ (de hecho se parece mucho a ellos en el aspecto, pero la filosofía de funcionamiento es totalmente distinta). SUN® desarrollo este lenguaje en principio con otra orientación, la idea es que fuese utilizado en microelectrónica y sistemas embebidos. Lo que nunca se pensó SUN® es la repercusión y evolución que tendría más tarde este lenguaje.

**Cuatro razones para aprender Java**

1. Por el futuro y presente que tiene.
2. Es un lenguaje sencillo.
3. Es un lenguaje orientado a objetos.
4. Es independiente de la plataforma.

### 1.1.2 EL JDK

El JDK (*Java Development Kit*), aunque no contiene ninguna herramienta gráfica para el desarrollo de programas, sí que contiene aplicaciones de consola y herramientas de compilación, documentación y depuración. El JDK incluye el JRE (*Java Runtime Environment*) que consta de los mínimos componentes necesarios para ejecutar una aplicación Java, como son la máquina virtual y las librerías de clases.

El JDK contiene, entre otras, las siguientes herramientas de consola:

- **java**. Es la máquina virtual de Java.
- **javac**. Es el compilador de Java. Con él es posible compilar las clases que desarrollemos.
- **javap**. Es un desensamblador de clases.
- **jdb**. El depurador de consola de Java
- **javadoc**. Es el generador de documentación.
- **appletviewer**. Visor de *Applets*.

**Importante**
Una vez descargado e instalado el JDK hay que modificar los valores de dos variables de entorno:
- Variable PATH. Apunta donde está situado el directorio bin del JDK.
- Variable CLASSPATH. Apunta donde están situadas las clases del JDK.

Podemos descargar y utilizar varios JDK simplemente modificando los valores de ambas variables.

## A FONDO

### CONOCIENDO LA VERSIÓN DE JAVA

Para conocer la versión de java con la que estamos trabajando basta con ejecutar lo siguiente en una shell o intérprete de comandos:

```
java -version
```

Y aparecerá en la ventana algo parecido a esto:

```
C:\Documents and Settings\JUAN CARLOS>java -version
java version "1.6.0_20"
Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
Java HotSpot(TM) Client VM (build 16.3-b01, mixed mode, sharing)
```

### 1.1.3 LOS PROGRAMAS EN JAVA

Los programas o aplicaciones en Java se componen de una serie de ficheros .class que son ficheros en bytecode que contienen las clases del programa. Estos ficheros no tienen por qué estar situados en un directorio concreto, sino que pueden estar distribuidos en varios discos o incluso en varias máquinas.

La aplicación se ejecuta desde el método principal o *main()* situada en una clase. A partir de aquí se van creando objetos a partir de las clases y se va ejecutando la aplicación. El *main()* es un método estático (ya se explicará esto más adelante) el cual puede empezar a crear los objetos, incluidos los de su propia clase.

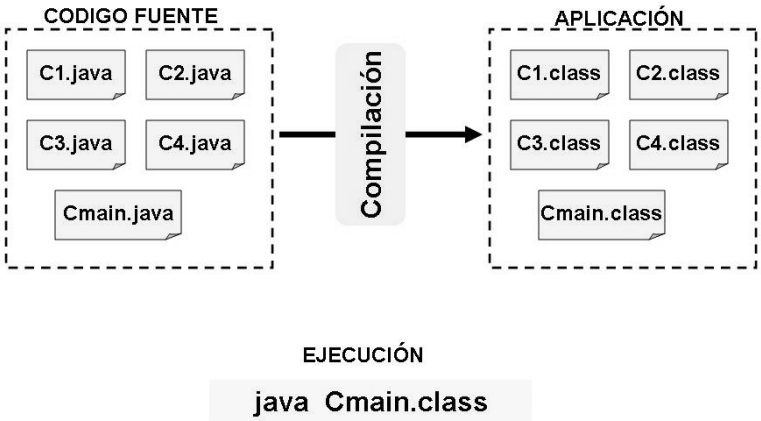



***Figura 1.4.*** *Proceso de compilación de un programa*

## 1.2 ESTRUCTURA Y BLOQUES FUNDAMENTALES DE UN PROGRAMA

En este apartado se va a ver el programa de inicio por excelencia en cualquier lenguaje de programación y se comentará cada una de sus líneas. El proceso de compilación y ejecución se explica en el siguiente apartado.

```
public class holamundo {
/* programa holamundo*/
    public static void main(String[] args) {
           /* lo único que hace este programa es mostrar
             la cadena "Hola Mundo" por pantalla*/
           System.out.println("Hola Mundo");
    }
}
```

**Los comentarios**

Existen comentarios de una línea solamente (//) y comentarios multilínea (/* */).

- //. Estos comentarios comienzan en la doble barra y terminan hasta el final de la línea.
- /* */ . Estos comentarios comienzan con los caracteres /* y terminan con los caracteres */ y se pueden extender múltiples líneas.

- La clase *holamundo*

En java generalmente cada clase es un fichero distinto. Si existieran varias clases en el fichero, la clase cuyo nombre coincide con el nombre del fichero debería de llevar el modificador public (public class holamundo) y es la que se puede utilizar desde fuera del fichero. Las clases tienen el mismo nombre que su fichero .java y es importante que mayúsculas y minúsculas coincidan. La clase abarca desde la primera llave que abre hasta la última que cierra.

```
public class holamundo {
......
}
```

- La función o método *main*

```
public static void main (String [ ] args)
{
...
}
```

El código Java en las clases se agrupa en métodos o funciones. Cuando Java va a ejecutar el código de una clase, lo primero que hace es buscar el método *main* de dicha clase para ejecutarlo.

El método **main** tiene las siguientes particularidades:

- Es público (**public**). Esto es así para poder llamarlo desde cualquier lado.
- Es estático (**static**). Al ser *static* se le puede llamar sin tener que instanciar la clase.
- No devuelve ningún valor (modificador **void**).
- Admite una serie de parámetros (**String [ ] args**) que en este ejemplo concreto no son utilizados.

Como puede verse en el ejemplo, el método *main* abarca todo el código contenido entre las llaves.

**Mostrar texto por pantalla.**

Parece intuitivo saber que el texto se mostrará por pantalla ejecutando la siguiente línea:

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

Para sacar información por pantalla en Java se utiliza la clase **System** que puede ser llamada desde cualquier punto de un programa, la cual tiene un atributo **out** que a su vez tiene dos métodos muy utilizados: **print**() y **println**(). La diferencia entre estos dos últimos métodos es que en el segundo se añade un retorno de línea al texto introducido. Como se puede ver la orden termina en **;** (todas las ordenes en Java terminan en **;** salvo los cierres de llaves a los cuales no hace falta ponérselo pues se sobreentiende que se finaliza la orden).

## 1.3 ENTORNOS INTEGRADOS DE DESARROLLO

Un IDE o Entorno Integrado de Desarrollo es una herramienta con el cual poder desarrollar y probar proyectos en un lenguaje determinado.

**Recuerda**
JDK o Java Development Kit es el software necesario para poder desarrollar y ejecutar programas java. También se denomina SDK (*Standard Development Kit*) o incluso J2SE (*Java 2 platform Standard Edition*).

Lo primero que hay que hacer cuando se instala un IDE es configurar como mínimo la ruta del JDK (*Java Development Kit*). Si no se tiene el JDK no se podrá trabajar con Java, luego habrá que instalarlo primero. En Ubuntu Linux basta con ejecutar desde consola el siguiente comando:

```
$ sudo apt-get install sun-java6-jdk
```

**Importante**
Cuando se instale un entorno integrado de desarrollo hay que asegurarse que las opciones que indican las rutas de las bibliotecas, el JDK y demás recursos son correctas. Si no se hace esto el programa nunca podrá ejecutar ni compilar programas.

Una buena opción para empezar a programar en Java es instalar Geany. Geany es un IDE muy liviano y muy intuitivo y su instalación es sumamente sencilla. En Ubuntu Linux se instala ejecutando desde consola el siguiente comando:

```
$ sudo apt-get install geany
```

Una vez instalado el programa, hay que configurar la variable PATH en Windows® (Panel de control -> sistema -> Opciones Avanzadas -> variables de entorno) o las variables JAVA_HOME y JAVA en Linux.

**¿Necesitas ayuda para instalar Geany en tu equipo?**
En el material adicional del libro tienes un manual paso a paso para instalar Geany y el JDK en Windows® y en Linux.

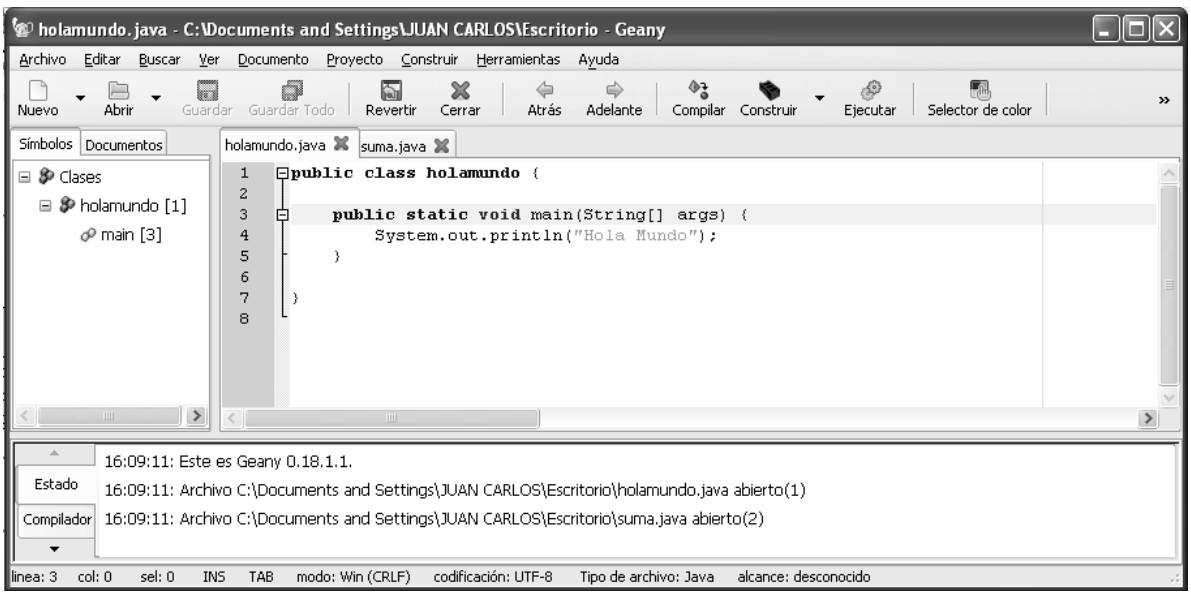



***Figura 1.5.*** *Geany. Un entorno de desarrollo ligero y versátil*

La secuencia de creación y ejecución de un programa en Java es un proceso que sigue los siguientes pasos: EDITAR →GUARDAR →COMPILAR →EJECUTAR.

Existen muchos IDE para trabajar con Java. En todos los ejemplos se ha utilizado Geany pero si se quiere algo mas potente, una buena opción es Eclipse. Eclipse fue desarrollado primeramente por IBM, aunque actualmente es un IDE de código abierto desarrollado y mantenido por la Fundación Eclipse (http://www.eclipse.org/). Eclipse puede utilizarse para Java y añadiendo plugins pueden utilizarse otros lenguajes de programación. Eclipse ha desarrollado numerosas versiones, todas con nombres estelares (Callisto, Europa, Ganymede, Galileo, Helios…). Otra opción no

menos interesante; es NetBeans de la extinta SUN® ahora Oracle®. NetBeans es una aplicación de código abierto y muchos desarrolladores Java la utilizan. Como consejo, se recomienda Geany para pequeños proyectos y programas como NetBeans o Eclipse para proyectos más serios.

**Recuerda**
NetBeans y Eclipse son entornos de desarrollo libres.

### ¿Es necesario un IDE para compilar y ejecutar Java?

La respuesta es No. No es necesario compilar desde un IDE nuestro programa. En principio, si la variable PATH está correctamente configurada bastaría con ejecutar desde línea de comando y desde el mismo directorio donde se encuentra el fichero holamundo.java el siguiente comando:

```
$javac holamundo.java
```

Si el programa está correctamente escrito y el compilador no muestra ninguna salida de error aparecerá un fichero holamundo.class que será el bytecode o código que podrá ser ejecutado en cualquier máquina virtual Java.

## ACTIVIDADES

- Investiga qué es el bytecode y qué es y cómo funciona una máquina virtual de Java.

Una vez tenemos el fichero .class hay que ejecutar el programa. Esto se realiza con el siguiente comando:

```
$java holamundo
```

O si se está en un entorno Windows®:

```
C:\> java holamundo
```

Y saldrá en la pantalla la cadena que queremos “Hola Mundo”.

## ACTIVIDADES

- Instala Geany y el JDK en tu máquina. Una vez instaladas estas dos cosas configura las variables PATH y CLASSPATH en la máquina.
- Prueba a compilar dentro y fuera del IDE el programa holamundo y comprueba que funciona ejecutándolo.

# 1.4 TIPOS DE DATOS SIMPLES

Los tipos de datos se utilizan generalmente al declarar variables y son necesarios para que el intérprete o compilador conozca de antemano el tipo de información que va a contener una variable. Los tipos de datos primitivos en Java son los siguientes:

**Tabla 1.1.** Tipos de datos simples

| Tipo de datos | Información representada | Rango | Descripción |
|---|---|---|---|
| byte | Datos enteros | -128 ⟷ +127 | Se utilizan 8 bits (1 byte) para almacenar el dato. |
| short | Datos enteros | -32768 ⟷ +32767 | Dato de 16 bits de longitud (independientemente de la plataforma). |
| int | Datos enteros | -2147483648 ⟷ +2147483647 | Dato de 32 bits de longitud (independientemente de la plataforma). |
| long | Datos enteros | -9223372036854775808 ⟷ +9223372036854775807 | Dato de 64 bits de longitud (independientemente de la plataforma). |
| char | Datos enteros y caracteres | 0 ⟷ 65535 | Este rango es para representar números en unicode, los ASCII se representan con los valores del 0 al 127. ASCII es un subconjunto del juego de caracteres Unicode. |
| float | Datos en coma flotante de 32 bits | Precisión aproximada de 7 dígitos | Dato en coma flotante de 32 bits en formato IEEE 754 (1 bit de signo, 8 para el exponente y 24 para la mantisa). |
| double | Datos en coma flotante de 64 bits | Precisión aproximada de 16 dígitos | Dato en coma flotante de 64 bits en formato IEEE 754 (1 bit de signo, 11 para el exponente y 52 para la mantisa). |
| boolean | Valores booleanos | true/false | Utilizado para evaluar si el resultado de una expresión booleanas es verdadero (true) o falso(false). |

## ACTIVIDADES

- Se propone al alumno que investigue y recopile información sobre el juego de caracteres Unicode y ASCII con especial detenimiento en este último.

### 1.4.1 ¿CÓMO SE UTILIZAN LOS TIPOS DE DATOS?

A continuación, se muestran ejemplos de utilización de tipos de datos en la declaración de variables.

**Tabla 1.2.** Utilización de tipos de datos

| Tipo de dato | Código |
|---|---|
| byte | byte a; |
| short | short b, c=3; |
| int | int d = -30;<br>int e = 0xC125; |
| long | long b=434123 ;<br>long b=5L ; /* la L en este caso indica Long*/ |
| char | char car1='c';<br>char car2=99; /*car1 y car2 son lo mismo porque el 99 en decimal es la 'c' */ |
| float | float pi=3.1416;<br>float pi=3.1416F; /* la F en este caso indica Float*/<br>float medio=1/2F; /*0.5*/ |
| double | double millón=1e6; /* 1x106 */<br>double medio1/2D; /*0.5 la D en este caso indica Double*/ |

# 1.5 CONSTANTES Y LITERALES

### 1.5.1 LAS CONSTANTES

**Cuestión de estilo**
Las constantes se declaran en mayúscula mientras que las variables se hacen en minúscula (esto se realiza como norma de estilo).

Las constantes se declaran siguiendo el siguiente formato:

```
final [static] <tipo de datos> <nombre de la constante> = <valor>;
```

Donde el calificador *final* identificará que es una constante, la palabra *static* si se declara implicará que solo existirá una copia de dicha constante en el programa aunque se declare varias veces, el tipo de datos de la constante seguido del nombre y por último el valor que toma.

```
final static double PI=3.141592;
```

**Importante**
Las constantes se utilizan en datos que nunca varían (IVA, PI, etc.). Utilizando constantes y no variables nos aseguramos que su valor no va a poder ser modificado nunca. También utilizar constantes permite centralizar el valor de un dato en una sola línea de código (si se quiere cambiar el valor del IVA se hará solamente en una línea en vez de si se utilizase el literal 18 en muchas partes del programa).

### 1.5.2 LOS LITERALES

Un literal puede ser una expresión:

- De tipo de dato simple.
- El valor *null*.
- Un *string* o cadena de caracteres (por ejemplo “Hola Mundo”).

Ejemplos de literales en Java pueden ser ‘a’, 322, 3.1416, “pi” o “programación estructurada”.

## 1.6 VARIABLES

Una variable no es más ni menos que una zona de memoria donde se puede almacenar información del tipo que desee el programador.

**Las palabras clave**
Las palabras clave son las órdenes del lenguaje de programación. El compilador espera esos identificadores para comprender el programa, compilarlo y poder ejecutarlo. Por lo tanto queda PROHIBIDO utilizar palabras clave como (*boolean*, *double*, *long*, *if*, *private*, *etc*.) utilizadas por el propio Java para nombrar variables dentro de un programa. Tampoco se pueden utilizar caracteres especiales para nombrar variables como (+, -, /, etc.).

```
class suma
{
     static int n1=50;  // variable miembro de la clase
     public static void main(String [] args)
     {
     int  n2=30, suma=0; // variables locales
     suma=n1+n2;
     System.out.println("LA SUMA ES: " + suma);
     }
}
```

Como puede verse en el ejemplo anterior, las variables se declaran dentro de un bloque (por bloque se entiende el contenido entre las llaves { }) y son accesibles solo dentro de ese bloque.

Las variables declaradas en el bloque de la clase como *n1* se consideran miembros de la clase, mientras que las variables *n2* y suma pertenecen al método main y solo pueden ser utilizados en el mismo. Las variables declaradas en el bloque de código de un método son variables que se crean cuando el bloque se declara, y se destruyen cuando finaliza la ejecución de dicho bloque.

**Inicialización de variables**
Las variables miembros de una clase se inicializan por defecto (las numéricas con 0 los caracteres con '\0' y las referencias a objetos y cadenas con *null*) mientras que las variables locales no se inicializan por defecto.

**Importante**
Una variable local no puede ser declarada como *static*.

### 1.6.1 VISIBILIDAD Y VIDA DE LAS VARIABLES

Visibilidad, *scope* o ámbito de una variable son sinónimos. Visibilidad es la parte del código de una aplicación donde la variable es accesible y puede ser utilizada.

**Recuerda**
En Java las variables no pueden declararse fuera de una clase.

Por regla general, en Java, todas las variables que están dentro de un bloque (entre { y }) son visibles y existen dentro de dicho bloque. Las funciones miembro de una clase, podrán acceder a todas las variables miembro de dicha clase pero no a las variables locales de otra función miembro.

# 1.7 OPERADORES Y EXPRESIONES

## 1.7.1 OPERADORES ARITMÉTICOS

Los operadores aritméticos son utilizados para realizar operaciones matemáticas.

**Tabla 1.3.** Operadores aritméticos

| Operador | Uso | Operación |
|---|---|---|
| + | A + B | Suma |
| - | A - B | Resta |
| * | A * B | Multiplicación |
| / | A / B | División |
| % | A % B | Módulo o resto de una división entera |

En el siguiente ejemplo se puede observar la utilización de operadores aritméticos:

```
int  n1=2, n2;
n2=n1 * n1;       // n2=4
n2=n2-n1;         // n2=2
n2=n2+n1+15; // n2=19
n2=n2/n1;         // n2=9
n2=n2%n1;       // n2=1
```

### 1.7.2 OPERADORES RELACIONALES

Con los operadores relacionales se puede evaluar la igualdad y la magnitud. En la siguiente tabla A y B no son los operadores, sino que son los operandos como se puede ver:

**Tabla 1.4.** Operadores relacionales

| Operador | Uso | Operación |
|---|---|---|
| < | A< B | A menor que B |
| > | A > B | A mayor que B |
| <= | A<= B | A menor o igual que B |
| >= | A>= B | A mayor o igual que B |
| != | A!= B | A distinto que B |
| = = | A= = B | A igual que B |

En el siguiente ejemplo se puede observar la utilización de operadores relacionales:

```
int  m=2, n=5;
boolean res;
res =m > n;//res=false
res =m < n;//res=true
res =m >= n;//res=false
res =m <= n;//res=true
res =m == n;//res=false
res =m != n;//res=true
```

### 1.7.3 OPERADORES LÓGICOS

Con los operadores lógicos se pueden realizar operaciones lógicas. En la siguiente tabla A y B no son los operadores, sino que son los operandos como se puede ver:

**Tabla 1.5.** Operadores lógicos

| Operador | Uso | Operación |
|---|---|---|
| && o & | A&& B o A&B | A AND B. El resultado será true si ambos operandos son true y false en caso contrario. |
| \|\| o \| | A \|\| B o A \| B | A OR B. El resultado será false si ambos operandos son false y true en caso contrario. |
| ! | !A | Not A. Si el operando es true el resultado es false y si el operando es false el resultado es true. |
| ^ | A ^ B | A XOR B. El resultado será true si un operando es true y el otro false, y false en caso contrario. |

En el siguiente ejemplo se puede observar la utilización de operadores lógicos:

```
int  m=2, n=5;
boolean res;
res =m > n && m >= n;//res=false
res =!(m < n || m != n);//res=false
```

### 1.7.4 OPERADORES UNITARIOS O UNARIOS

**Tabla 1.6.** Operadores unitarios

| Operador | Uso | Operación |
|---|---|---|
| ~ | ~A | Complemento a 1 de A |
| - | -A | Cambio de signo del operando |
| -- | A-- | Decremento de A |
| ++ | A++ | Incremento de A |
| ! | ! A | Not A (ya visto) |

En el siguiente ejemplo se puede observar la utilización de operadores unitarios:

```
int  m=2, n=5;
m++; //  m=3
n--;  //    n=4
```

### 1.7.5 OPERADORES DE BITS

**Tabla 1.7.** Operadores de bits

| Operador | Uso | Operación |
|---|---|---|
| & | A & B | AND lógico. A AND B. |
| \| | A \| B | OR lógico. A OR B. |
| ^ | A ^ B | XOR lógico. A XOR B. |
| << | A << B | Desplazamiento a la izquierda de A B bits rellenando con ceros por la derecha. |
| >> | A >> B | Desplazamiento a la derecha de A B bits rellenando con el BIT de signo por la izquierda. |
| >>> | A >>> B | Desplazamiento a la derecha de A B bits rellenando con ceros por la izquierda. |

En el siguiente ejemplo se puede observar la utilización de operadores de bits:

```
int  num=5;
num = num << 1; //  num = 10, equivale a num = num * 2
num = num >> 1; //  num = 5, equivale a num = num / 2
```

### 1.7.6 OPERADORES DE ASIGNACIÓN

**Tabla 1.8.** Operadores de asignación

| Operador | Uso | Operación |
|---|---|---|
| = | A = B | Asignación. Operador ya visto. |
| *= | A *= B | Multiplicación y asignación. La operación A*=B equivale a A=A*B. |
| /= | A /= B | División y asignación. La operación A/=B equivale a A=A/B. |
| %= | A %= B | Módulo y asignación. La operación A%=B equivale a A=A%B. |
| += | A += B | Suma y asignación. La operación A+=B equivale a A=A+B. |
| -= | A -= B | Resta y asignación. La operación A-=B equivale a A=A-B. |

En el siguiente ejemplo se puede observar la utilización de operadores de asignación:

```
int  num=5;
num += 5; //  num = 10, equivale a num = num + 5
```

### 1.7.7 PRECEDENCIA DE OPERADORES

**Consejo**
Utiliza paréntesis y de esa forma puedes dejar los programas más legibles y controlar las operaciones sin tener que depender de la precedencia.

La precedencia de operadores se resume en la siguiente tabla:

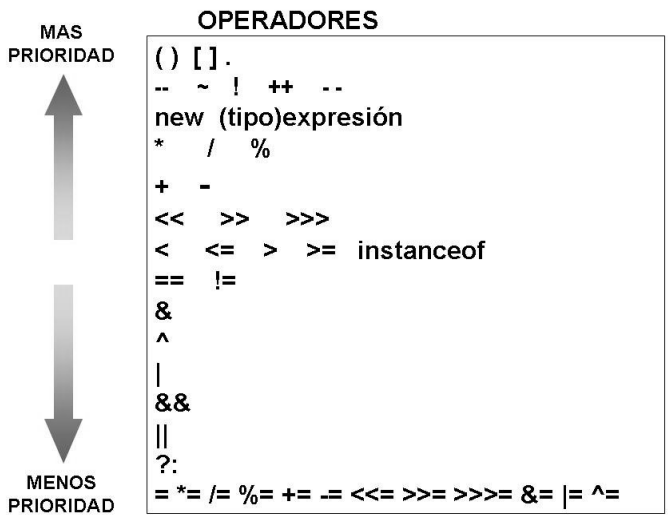



***Figura 1.6.*** *Prioridad de los operadores*

Imaginemos que se tiene un código como el siguiente:

```
int a = 4;
a = 5 * a + 3;
```

Se desea conocer el valor que tomará a. Para ello se mira en la tabla y se puede observar que el operador * tiene más precedencia que el operador +, con lo cual primero se ejecutará 5 * a, y al resultado de esta operación se le sumará 3. El resultado de la expresión será 23 y por lo tanto el valor de a será 23 al ejecutar este código.

## 1.8 CONVERSIONES DE TIPOS (CAST)

Existen dos tipos de conversiones, las conversiones explícitas e implícitas.

- **Conversiones implícitas**. Se realiza de forma automática entre dos tipos de datos diferentes. Requiere que la variable destino (la colocada a la izquierda) tenga más precisión que la variable origen (situada a la derecha).

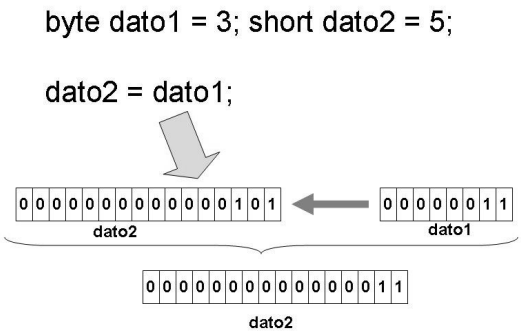



***Figura 1.7.*** *Ejemplo de conversión implícita*

- **Conversiones explícitas**. En este caso es el programador el que fuerza la conversión mediante una operación llamada *cast* con el formato:

  (tipo) expresión

**Recuerda**
Como puede ser comprensible no se pueden realizar conversiones entre enteros y *booleanos* o reales y *booleanos*.

Un ejemplo de conversión explícita sería el siguiente:

```
int idato=5;
byte bdato;
bdato = (byte)idato;
System.out.println(bdato); // sacará 5 por pantalla
```

**Consejo**
Intenta evitar las conversiones de tipos en la medida de lo posible. En algunas conversiones explícitas como ya supondrás pueden perder información en algunos casos.

# RESUMEN DEL CAPÍTULO

En este tema se introduce al alumno en los lenguajes de programación, y más concretamente, al lenguaje Java. Este tema es una primera toma de contacto del alumno con la programación y se hace desde una posición global viendo cómo funciona un programa muy sencillo (el famoso Hola Mundo) y desde los aspectos básicos como son las variables, los tipos de datos, los comentarios, los operadores, etc.

Es posible que el alumno en estos primeros temas vea conceptos que luego se estudien con más profundidad en apartados siguientes. El alumno en este tema deberá de entender la estructura, cómo funciona Java y poner énfasis en el estudio de los aspectos básicos del lenguaje, los distintos tipos de datos y en la utilidad y uso de los operadores y expresiones.

Tambien es importante para el capítulo que el alumno sepa instalar y utilizar un IDE. La instalación, compilación y ejecución de programas es una cuestión básica que debe de manejar el alumno para el resto del libro.

# EJERCICIOS RESUELTOS

**1.** Realiza un método para la clase *Test* que genere letras de forma aleatoria. Como ejercicio complementario investiga el funcionamiento y uso de la función Math.random().

**Solución**:

```
class Test {
    public static char getLetras(){
    return (char)(Math.random()*26 + 'a`);
    }
    public static void main(String[] args) {
    System.out.println(getLetras());
    System.out.println(getLetras());
    System.out.println(getLetras());
    System.out.println(getLetras());
    }
}
```

**2.** El objetivo de este ejercicio es cumplimentar la segunda columna de la siguiente tabla. Como puedes observar ya está cumplimentada pero en los ejercicios propuestos tendrás que comprobar y cumplimentarla tú.

**Tabla 1.9.** Tabla ejercicio 2

| ¿Compilará y funcionará el siguiente código? | | En caso afirmativo explica que mostrará por pantalla. En caso negativo explica por qué no funciona. |
|---|---|---|
| int a = 'a';<br>System.out.println(a); | ☒ Funciona<br>☐ No funciona | El código introduce 97 en la variable a que es el valor del código ASCII 'a' y lo muestra por pantalla. |
| int pi = 3.14;<br>System.out.println(pi); | ☐ Funciona<br>☒ No funciona | No funciona. No es posible introducir un número real en una variable de precisión entero |
| double pi = 3,14;<br>System.out.println(pi); | ☐ Funciona<br>☒ No funciona | Para que funcione basta con cambiar la coma por un punto. |
| boolean adivina = (1 == 4);<br>System.out.println(adivina); | ☒ Funciona<br>☐ No funciona | Correcto. Muestra false por pantalla porque los dos valores no son iguales. |
| boolean adivina = (97 == 'a' == 97 );<br>System.out.println(adivina); | ☐ Funciona<br>☒ No funciona | No funcionará porque la primera parte de la comparación 97 == 'a' genera un booleano, y al comparar un booleano con un entero (97) el compilador dará un error. |
| boolean adivina = (97 == 'a' == true );<br>System.out.println(adivina); | ☒ Funciona<br>☐ No funciona | Muestra true por pantalla porque 97 es el código ASCII de 'a' y por lo tanto dará true. Al comparar true con otro valor booleano como true el resultado será true. |

**3.** Averigua si las siguientes afirmaciones son verdaderas o falsas:

- En Java generalmente un programa consta de varias clases las cuales se compilan en un único fichero.
- El método *main* puede ser *static* o no. En caso de no ser *static* puede haber varios en un mismo programa.
- Los métodos y funciones difieren en Java en que en los primeros no devuelven ningún valor.
- Es posible hacer **byte a = 200;**. El único problema es que como una variable byte solamente almacena hasta el valor 127 la variable a valdrá solo 127.

La solución a este ejercicio está al final de los ejercicios propuestos.

**4.** Realiza un programa en Java que dada dos variables a y b, intercambie los valores de a y b.

**Solución**:

```
class intercambio {
    public static void main(String[] args) {
        int a= 5, b= 8;
        int tmp;

        tmp=a;
        a=b;
        b=tmp;
        System.out.println("El valor de a ahora es: "+a);
        System.out.println("El valor de b ahora es: "+b);
   }
}
```

**5.** Dentro de una clase joven tenemos las variables enteras *edad*, *nivel_de_estudios* e *ingresos*.

Necesitamos almacenar en la variable *booleana jasp* el valor:

- Verdadero. Si la edad es menor o igual a 28, el nivel_de_estudios es mayor que tres y los ingresos superan los 28.000 (euros).
- Falso. En caso contrario.

■ Escribe el código necesario (2 líneas).

**Solución**:

```
jasp = false;
jasp = ((edad <= 28) && (nivel_de_estudios > 3) && (ingresos > 28000));
```

**6.** ¿Que mostrará este programa por pantalla?

```
public class Test {
   public static void main(String[] args) {
        int i=0x100;
        i >>>= 1;
        System.out.println(i);
   }
}
```

**Solución**:

128

# EJERCICIOS PROPUESTOS

**1.** Modifica el siguiente programa para hacer que compile y funcione:

```
class suma
{
     static int n1=50;
     public static void main(String [] args)
     {
     int  n2=30, suma=0, n3;
     suma=n1+n2;
     System.out.println("LA SUMA ES: " + suma);
     suma=suma+n3;
     System.out.println(suma);
     }
}
```

**2.** ¿Por qué no compila el siguiente programa? Modifícalo para hacer que funcione.

```
class suma
{
     public static void main(String [] args)
     {
     int  n1=50,n2=30,
     boolean suma=0;
     suma=n1+n2;
     System.out.println("LA SUMA ES: " + suma);
     }
}
```

**3.** El siguiente programa tiene 3 fallos, averigua cuáles son y modifica el programa para que funcione.

```
class cuadrado
{
     public static void main(String [] args)
     {
     int  numero=2,
     cuad=numero * número;
     System.out.println("EL CUADRADO DE "+NUMERO+" ES: " + cuad);
```

```
    }
}
```

- **4.** ¿Qué mostrará el siguiente código por pantalla?

```
int  num=5;
num += num - 1 * 4 + 1;
System.out.println(num);
num=4;
num %= 7 * num % 3 * 7 >> 1;
System.out.println(num);
```

- **5.** Realiza un programa que calcule la longitud de una circunferencia de radio 3 metros.
- **6.** Realiza un programa que calcule el área de una circunferencia de radio 5,2 centímetros.
- **7.** Realiza un programa que muestre en pantalla, respetando los retornos de línea, el siguiente texto:

  Me gusta la programación

  cada día más.
- **8.** (Ejercicio de dificultad alta) Realiza un programa que genere letras aleatoriamente y determine si son vocales o consonantes.
- **9.** Cumplimenta la siguiente tabla:

**Tabla 1.10.** Tabla ejercicio 9

| ¿Compilará y funcionará el siguiente código? | | En caso afirmativo explica qué mostrará por pantalla. En caso negativo explica por qué no funciona. |
|---|---|---|
| boolean adivina = ((97 == 'a') && true );<br>System.out.println(adivina); | ☐ Funciona<br>☐ No funciona | |
| int a=1;<br>int b = a>>>2;<br>System.out.println(b); | ☐ Funciona<br>☐ No funciona | |
| int a = 7 \| 4;<br>System.out.println(a);<br>int b = 3 \| 4;<br>System.out.println(b); | ☐ Funciona<br>☐ No funciona | |

| | | |
|---|---|---|
| int a = 7 & 4;<br>System.out.println(a);<br>int b = 3 & 4;<br>System.out.println(b); | ☐ Funciona<br>☐ No funciona | |
| int a = ~4;<br>System.out.println(a); | ☐ Funciona<br>☐ No funciona | |
| int a = (~4 * 5)&1;<br>System.out.println(a); | ☐ Funciona<br>☐ No funciona | |

**10.** Dentro de una clase *joven* tenemos las variables enteras *edad*, *nivel_de_estudios* e *ingresos*.

Necesitamos almacenar en la variable *booleana* jasp el valor:

- Verdadero. Si la *edad* es menor o igual a 28 y el *nivel_de_estudios* es mayor que tres, o bien, la *edad* es menor de 30 y los *ingresos* superan los 28.000 (euros).
- Falso. En caso contrario.

- Escribe el código necesario (2 líneas).

**11.** Realiza un programa con una variable entera **t** la cual contiene un tiempo en segundos y queremos conocer este tiempo pero expresado en horas, minutos y segundos.

**12.** (Ejercicio de dificultad alta) Realiza un programa que dado un importe en euros nos indique el mínimo número de billetes y la cantidad sobrante que se pueden utilizar para obtener dicha cantidad.

Por ejemplo:

232 euros:

1 billete de 200.

1 billete de 20.

1 billete de 10

Sobran 2 euros.

**Solución al ejercicio resuelto número 3**:

Todas las afirmaciones son falsas.