UT2: IDENTIFICACIÓN DE LOS ELEMENTOS DE UN PROGRAMA INFORMÁTICO EN JAVA

Contenido

1 Estructura y bloques fundamentales de un programa Java

El código fuente de un programa en Java debe estar escrito en un **fichero de texto con extensión ".java"**. Para crear o editar el fichero debe utilizarse un editor de texto sin formato como el bloc de notas de Windows, el editor de texto de Linux, o alguno de los programas y entornos de desarrollo específicos para la creación de aplicaciones.

```
public class HolaMundo {
        public static void main(String args[]) {
            System.out.println("Hola Mundo!");
        }
}
```

La primera línea de código indica el nombre de la clase Java que estamos desarrollando. Todos los programas Java están formados por una o más clases. Es importante tener en cuenta que el nombre de la clase pública dentro del fichero debe corresponder exactamente con el nombre del fichero de texto que contiene el código fuente. En el caso de este ejemplo, el código debe almacenarse en un fichero de texto denominado "HolaMundo.java". Solo puede haber una clase pública en el fichero fuente.

```
public class HolaMundo
```

Las llaves { } indican el inicio y fin de cada bloque. Siempre deben ir en pareja, es decir, que por cada llave de apertura debe existir siempre una llave de cierre.

La siguiente línea realiza la declaración del método llamado "main". Cada clase Java que forma una aplicación contendrá uno o varios métodos, que son bloques de código que permiten tenerlo más organizado. Si una determinada clase Java contiene un método con el nombre "main", la ejecución de dicha clase comenzará por el código contenido en ese método (se trata de una aplicación, porque es "ejecutable").

```
public static void main(String args[])
```

Las líneas de código dentro del método **main** del ejemplo contienen las sentencias que se van a ejecutar. En este caso tan solo hay una sentencia, pero podría haber todas las que se quisiéramos. Aquí en concreto se utiliza la llamada a *System.out.println()* que permite mostrar en pantalla una serie de caracteres. El texto mostrado es el indicado entre comillas: "Hola Mundo!".

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

2 Variables

Las ***variables*** identifican datos mediante un nombre simbólico, haciendo referencia a un espacio de memoria principal en el que se sitúan los datos. Esto se hace para que puedan ser utilizados por el procesador, y así poder hacer cualquier tipo de operación con ellos.

Los nombres utilizados para identificar a las variables deben cumplir una serie de ***condiciones***:

- No pueden empezar por un dígito numérico.
- No pueden utilizarse espacios, y los únicos caracteres especiales válidos son el guion bajo (_) y el símbolo del dólar ($).
- Son sensibles a las mayúsculas y minúsculas, es decir, las variables "suma" y "Suma" se consideran variables distintas.

No pueden utilizarse como nombres de variables las palabras reservadas de Java, que son las siguientes:

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

Es una convención entre programadores de Java empezar los nombres de las variables por una letra minúscula.

Cuando el nombre de una variable está formado por más de una palabra, se suele utilizar una letra mayúscula para distinguir el comienzo de las palabras a partir de la segunda. Por ejemplo: sumaTotal.

Es muy recomendable utilizar nombres que hagan referencia al contenido que van a almacenar para facilitar la comprensión del código. Es mucho más claro utilizar el nombre "suma" que "s".

Ejemplos de nombres de variables válidos: indice, ventas, compras, saldoGeneral, importetotal, contador_lineas, $valor, num2.

Ejemplos de nombres de variables no válidos: 3valores, suma&total, super, edad media.

Antes de poder utilizar una variable, esta debe ser declarada en el programa. La declaración de variables se debe realizar siguiendo el siguiente formato de sentencia:

```
tipoDato nombreVariable;
```

Donde **tipoDato** es uno de los tipos de datos básicos o el nombre de una clase (byte, short, int, long, float, double, boolean, char, String, Coche, etc.), y **nombreVariable** es el nombre que se desea asignar a la variable siguiendo las normas anteriores.

Es posible declarar más de una variable de un mismo tipo en la misma línea separando los nombres con comas:

```
tipoDato nombreVariable1, nombreVariable2, nombreVariable3;
```

Ejemplos de declaraciones de variables:

```
int num1, num2, suma;
char letraNIF;
String saludoInicial;
boolean mayorEdad;
```

Es posible **asignar un valor inicial** a las variables en el momento de declararlas. Para **inicializar variables** se debe seguir el siguiente formato de sentencia:

```
tipoDato nombreVariable = valorInicial;
```

Donde valorInicial puede ser un valor literal, otra variable declarada anteriormente o una expresión combinando valores literales y variables con operadores. Lo que asignamos debe ser del mismo tipo de dato que la variable que se está declarando.

Ejemplos de declaraciones de variables con inicialización:

```
int num1 = 34;
int doble = num1 * 2;
String saludo = "Hola";
char letraA = 'A',  letraB = 'B';
```

En el código del programa es posible asignar valores a las variables que previamente han sido declaradas. Al hacerlo, el valor que guardará la variable anteriormente se perderá. Se debe utilizar el siguiente formato:

```
nombreVariable = valor;
```

Donde valor puede ser de nuevo un valor literal, una variable declarada anteriormente (puede ser la misma variable) o una expresión combinando valores literales y variables con operadores. El valor debe ser del mismo tipo de dato que la variable a la que se está asignando el nuevo valor.

```
package ejemplos;

public class AsignacionVariables {
	public static void main(String[] args) {
		int a=5, b=0, c;

		b = a * 3; // Se cambia el valor de b a 15
		c = a; // Se guarda en c el valor de a que es 5
		a = a + 6; // Se suma 6 al valor que tenía a.
		// Ahora a vale 11
		b = a - c; // b guarda 11 - 5 , luego vale 6
		System.out.println("La variable a contiene: " + a);
		System.out.println("La variable b contiene: " + b);
		System.out.println("La variable c contiene: " + c);
	}
}
```

```
Se muestra lo siguiente:
     La variable a contiene: 11
     La variable b contiene: 6
     La variable c contiene: 5
```

Las variables pueden ser utilizadas dentro del bloque de código en el que han sido declaradas, es decir, dentro de las llaves "{" y "}" que marcan el inicio y el fin de un bloque de código. Se denomina **ámbito** de la variable al bloque de código en el que se declara la variable, que es el único sitio donde se puede utilizar.

```
package ejemplos;
public class AmbitoVariables {
static int variableGlobal;

public static void main(String[] args) {
        int variableDelMain = 10;
        /*Aquí se pueden usar variableGlobal y
        variableDelMain. No se puede usar
        variableDeOtroMetodo */
        System.out.println("variableGlobal " + variableGlobal);
        System.out.println("variableDelMain " + variableDelMain);
        otroMetodo();
}

static void otroMetodo() {
        int variableDeOtroMetodo=90;
        /* Aquí se pueden usar variableGlobal y
        variableDeOtroMetodo. No se puede usar
        variableDelMain */
        System.out.println("variableGlobal " + variableGlobal);
        //System.out.println("variableDelMain " + variableDelMain);
        System.out.println("variableDeOtroMetodo " +
        variableDeOtroMetodo);
}
}
```

3 Tipos de datos básicos

El lenguaje de programación Java permite la utilización de los siguientes tipos de datos básicos:

Números enteros: Representan a los números enteros (sin parte decimal) con signo (pueden ser positivos o negativos). Se dispone de varios tipos de datos, ocupando cada uno de ellos un espacio distinto en memoria. Cuanta más capacidad de almacenamiento, más grande es el rango de valores permitidos, aunque ocupará más espacio de memoria principal. Se dispone de los siguientes tipos:

- **byte**: Ocupan 8 bits (1 byte), permitiendo almacenar valores entre -128 y 127.
- **short**: Ocupan 16 bits (2 bytes), permitiendo almacenar valores entre -32.768 y 32.767.

- **int**: Ocupan 32 bits (4 bytes), permitiendo almacenar valores entre -2.147.483.648 y 2.147.483.647. Es el tipo de datos por defecto para los valores numéricos enteros. Este tipo de datos es lo suficientemente grande para almacenar los valores numéricos que vayan a usar tus programas. Se suelen usar los tipos anteriores si se pueden producir problemas con el espacio de memoria.
- **long**: Ocupan 64 bits (8 bytes), permitiendo almacenar valores entre

  -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.

Números reales: Representan a los números reales con parte decimal y signo positivo o negativo. Hay dos tipos de datos numéricos reales que permiten obtener mayor o menor precisión. Utilizan un método para almacenar los datos que puede ocasionar que el valor original varíe levemente del valor almacenado realmente. Cuanta más precisión se utilice, habrá menor variación.

- **float:** Ocupan 32 bits (4 bytes). Se le denomina de simple precisión. Almacenan valores desde -3.40282347E+38 a + 3.40282347E+38
- **double**: Ocupan 64 bits (8 bytes). Se le denomina de doble precisión. Es el tipo de datos por defecto para los valores numéricos reales. Almacenan valores desde:

  - 1.79769313486231570E+308 a +1.79769313486231570E+308

Tipo	Representación / Valor	Tamaño (en bits)	Valor mínimo	Valor máximo	Valor por defecto
`boolean`	`true` o `false`	1	N.A.	N.A.	`false`
`char`	Carácter Unicode	16	\u0000	\uFFFF	\u0000
`byte`	Entero con signo	8	-128	128	0
`short`	Entero con signo	16	-32768	32767	0
`int`	Entero con signo	32	-2147483648	2147483647	0
`long`	Entero con signo	64	-9223372036854775808	9223372036854775807	0
`float`	Coma flotante de precisión simple Norma IEEE 754	32	±3.40282347E+38	±1.40239846E-45	0.0
`double`	Coma flotante de precisión doble Norma IEEE 754	64	±1.79769313486231570E+308	±4.94065645841246544E-324	0.0

Valores lógicos: Representan dos únicos posibles valores: verdadero y falso.

- **boolean:** Ocupan 1 bit, pudiendo almacenar los valores true (verdadero) y false (falso).

Caracteres: Representan las letras, dígitos numéricos y símbolos contenidos en la tabla de caracteres Unicode.

- **char:** Ocupan 16 bits (2 bytes). Permite representar un único carácter, encerrado entre comillas simples, por ejemplo, la letra 'A'.

- **String:** Realmente **no es un tipo de dato básico de Java**, pero por su interés se incluye aquí. Permite representar un conjunto de caracteres, encerrado entre comillas dobles, por ejemplo: "Saludos para todos".

4 Valores literales

Un **valor literal** es la representación de un valor fijo en el código fuente de un programa.

Los valores correspondientes a los tipos de datos **numéricos enteros** (byte, short, int y long) se pueden expresar usando el sistema numérico decimal, octal o hexadecimal. Los números en sistema decimal se expresan de la manera habitual, simplemente escribiendo su valor con los dígitos numéricos, por ejemplo, 284. No se pueden emplear separadores de millares, por lo que para indicar el valor 1.000.000 (un millón) se debe escribir como 1000000. Para representar **valores negativos** se añade el carácter "-" (guion) delante del número, como es habitual es la escritura normal, por ej. -376.

Para representar un valor numérico entero en **sistema octal**, debe ir precedido del carácter 0 (cero), por ejemplo, el valor 284 se representa en octal como 0434. Asimismo, para representar un valor en el **sistema hexadecimal**, se debe emplear el prefijo 0x, por lo que el valor 284 se representa en hexadecimal como 0x11C.

Por defecto, los **valores literales numéricos enteros** se almacenan en memoria con **el formato del tipo de dato "int"**. Si se desea almacenar como "long", con el fin de poder obtener resultados con valores muy altos en los cálculos, se debe emplear el sufijo L en mayúscula o minúscula (sería recomendable utilizar la L mayúscula por el parecido de la letra minúscula con el valor 1). Por ejemplo, el valor 284L correspondería al valor entero 284 utilizando 64 bits (tipo long) para almacenarlo en memoria.

Para representar **valores literales** de los tipos de datos **numéricos reales** (float y double) se puede emplear el sistema decimal o la notación científica. En el sistema decimal se expresan los números con parte decimal de la forma usual, utilizando el punto como separador de la parte entera y decimal. En este caso tampoco se puede emplear los separadores de millares. Así, por ejemplo, el valor 21.843,83 se debe expresar como 21843.83 en el código fuente.

Los **números reales** expresados en **notación científica** deben emplear la letra "E" o "e" para separar la parte correspondiente al exponente. El valor 7,433·10^6 se debe expresar como 7.433e6 en el código fuente. Si el exponente es negativo se escribe el guion detrás de la letra E, por ejemplo, 7,433·10^-6 se expresa como 7.433e-6, y si el valor es negativo se indica el guion al principio, por ejemplo, -7,433·10^6 se expresa como -7.433e6.

Por defecto, los **valores literales numéricos reales** se almacenan en memoria con el **formato del tipo de dato "double"**. Si se desea almacenarlos como "float", con el fin de emplear menos espacio de memoria y necesitar menos precisión en los resultados de los cálculos, se debe emplear el sufijo F en mayúscula o minúscula. Por ejemplo, el valor 21843.83F correspondería al valor entero 21843.83 utilizando 32 bits (tipo float) para almacenarlo en memoria, en vez de 64 bits.

Los valores literales de los tipos **char** (carácter) y **String** (cadena de caracteres), pueden contener cualquier carácter Unicode. Los valores de tipo char se deben expresar encerrados entre **comillas simples** (en los teclados españoles habituales se encuentra junto a la tecla del cero), por ejemplo, la letra A se debe expresar como 'A'. Por otro lado, las cadenas de caracteres (tipo String) se expresan entre **comillas dobles** (en la tecla del 2), por ejemplo, el texto Saludos a todos, se debe escribir como "Saludos a todos".

En caso de que se necesite escribir un carácter de la tabla de caracteres Unicode que no se encuentre en el teclado, se puede hacer indicando el código hexadecimal correspondiente a dicho carácter precedido del modificador \u (utilizando la barra invertida situada en la tecla junto al 1). En todo caso se debe encerrar entre comillas simples o dobles según se vaya a tratar como carácter o dentro de una cadena de caracteres. Por ejemplo, para escribir la letra griega beta (ß) se puede emplear '\u00DF', o para escribir la palabra España es posible utilizar "Espa\u00F1a".

El lenguaje de programación Java también soporta un pequeño **conjunto de caracteres especiales** que se pueden utilizar en los valores literales char y String: \b (retroceso), \t (tabulador), \n (nueva línea), \f (salto de página), \r (retorno de carro), \" (comilla doble), \' (comilla simple), \\ (barra invertida).

Por **ejemplo**, la cadena de caracteres "La palabra "hola" es un saludo" se tiene que escribir en el código fuente como: "La palabra \"hola\" es un saludo", para que no confunda las comillas dobles de inicio y fin de la cadena de caracteres.

El carácter especial '\n' permite introducir un salto de línea dentro de una cadena de caracteres. Por ejemplo, "Primera línea\nSegunda línea" mostraría ese texto separado en dos líneas.

Los valores literales para el tipo de dato **boolean** solo pueden ser **true** o **false**, que corresponden a los valores Verdadero y Falso. Hay que observar que se deben indicar sin comillas de ningún tipo, ya que no son cadenas de caracteres.

En un programa, es posible mostrar cualquiera de estos valores literales a través de la salida estándar (terminal, ventana de salida, etc) utilizando la siguiente sentencia:

```
System.out.println(valorLiteral);
```

Donde valorLiteral debe ser el valor que se desea mostrar manteniendo las normas comentadas.

Ejemplos:

```
//Mostrar un valor numérico entero
System.out.print("Número entero: ");
System.out.println(284);

//Mostrar un valor numérico real
System.out.print("Número real: ");
System.out.println(21843.83);
//Mostrar un carácter
System.out.print("Carácter: ");
System.out.println('A');

//Mostrar una cadena de caracteres
System.out.print("Cadena de caracteres: ");
System.out.println("Saludos a todos");

//Mostrar un valor lógico
System.out.print("Valor lógico: ");
System.out.println(true);
```

5 Constantes

Son similares a las variables en cuanto que son datos a los que se hace referencia mediante un nombre y a los que se les asigna un valor. Pero a diferencia de las variables, a las constantes no se les puede modificar el valor asignado.

El formato de declaración de las constantes es prácticamente igual que el utilizado para las variables. La única diferencia es que se debe indicar el modificador final delante del tipo de dato que almacenará:

```
final tipoDato nombreConstante = valor;
```

Al igual que en la declaración de variables, el valor que se asigna en la declaración puede ser un valor literal, una variable o una expresión.

Ejemplo:

```
final double PI = 3.1415926536;
```

Por convenio, el nombre de las constantes se escribe en **mayúsculas**.

El beneficio de usar constantes es evitar la repetición de escribir un mismo valor en el programa y facilitar su modificación.

El empleo de las constantes a lo largo del programa es igual que el utilizado con las variables. Se indica su nombre dentro de cualquier expresión o como parámetro para métodos como println().

```
System.out.println("Perímetro = " + 2*PI*radio);
```

6 Operadores

El lenguaje de programación Java incorpora una serie de operadores que permiten realizar cálculos y escribir expresiones que realicen una serie de operaciones sobre los datos.

Operadores Aritméticos:

+ (suma)

- (resta)

* (multiplicación)

/ (división entera o con decimales según operandos)

% (resto de la división)

Todos estos operadores aritméticos deben utilizarse con dos operandos, situados delante y detrás de los operadores, pudiéndose encadenar las operaciones. Se pueden incluir espacios para aclarar más el código. Los datos usados como operandos deben ser de alguno de los tipos de datos numéricos (byte, short, int, long, float o double).

Ejemplos:

4 + 3

8 - 5 + 2

6 * 2 / 3

8.5 - 3 + 4.3

El resultado de la división tendrá decimales o no, según el tipo de operandos que se utilice. **Si los dos son enteros, el resultado no tendrá decimales, pero si al menos uno de los operandos es de tipo numérico real (float o double) el resultado será de ese tipo**.

Ejemplos:

8 / 2 resulta 4

7 / 2 resulta 3

7.0 / 2 resulta 3.5

7 / 2.0 resulta 3.5

7.4 / 2 resulta 3.7

8 / 2.5 resulta 3.2

8.5 / 2.5 resulta 3.4

Es posible modificar el tipo de dato de cualquier operando indicando delante el nuevo tipo de dato entre paréntesis. Así se hace una **conversión de tipo (casting)**:

(double)7 / 2 resulta 3.5

7 / (float)2 resulta 3.5

El operador resto se debe utilizar con tipos de datos numéricos enteros. El resultado será el resto de la división entre los dos operandos:

7 % 2 resulta 1

8 % 3 resulta 2

El resultado de cualquier operación aritmética será del tipo de dato más grande que se utilice en los operandos. Por ejemplo, si se hace una operación entre dos números enteros (int) el resultado será del mismo tipo, pero si se hace entre un int y un long es resultado es de tipo long.

Ejemplos:

2147483647 * 2 resulta un dato incorrecto, en concreto -2, porque se están multiplicando dos int y el resultado sobrepasa el límite de los enteros.

2147483647L * 2 resulta 4294967294 porque el primer operando es de tipo long (se ha indicado L al final).

Los caracteres pueden ser utilizados para realizar cálculos aritméticos. En caso de que aparezca algún carácter en una expresión aritmética, se toma el valor numérico que le corresponde a cada carácter en la tabla de codificación Unicode.

Ejemplos:

'A' + 1 resulta 66, ya que el carácter 'A' tiene el código 65.

(char) ('A' + 1) resulta 'B' ya que se ha hecho una conversión de tipos del resultado.

Operadores Relacionales

Los operadores relacionales permiten comparar dos valores numéricos.

> (mayor que)

>= (mayor o igual que)

< (menor que)

<= (menor o igual que)

== (igual que)

!= (distinto de)

Cada uno de estos operadores relacionales debe emplearse con dos valores numéricos a ambos lados, pudiendo ser dos valores literales o resultados de expresiones aritméticas.

Ejemplos:

4 > 3 resulta true.
7 <= 2 resulta false.
5 + 2 == 4 + 3 resulta true.
4 * 3 != 12 resulta false.

Operadores Lógicos

Los operadores lógicos permiten unir valores o expresiones lógicas, obteniendo como resultado si es verdadera o falsa la expresión combinada. Son los siguientes:

&& (Y lógico - conjunción)

|| (O lógico - disyunción)

! (NO lógico)

Los operadores && y || deben utilizarse con dos valores o expresiones lógicas a ambos lados, mientras que el operador de negación ! solo se aplica al valor o expresión lógica que tenga a su derecha. El resultado que se obtiene utilizando estos operadores se obtiene de la siguiente tabla de verdad:

El **operador Y** resulta true solo si ambos operandos son true:

- true && true resulta true
- true && false resulta false
- false && true resulta false
- false && false resulta false

El **operador O** resulta true si al menos uno de los operandos son true:

- true || true resulta true
- true || false resulta true
- false || true resulta true
- false || false resulta false

El **operador NO** resulta true si el operando es false:

- !true resulta false.
- !false resulta true.

Ejemplos:

4 > 3 && 5 <= 5 resulta true, porque las dos expresiones son true.
4 > 3 && 5 != 5 resulta false, porque al menos una expresión es false.
4 > 3 || 5 != 5 resulta true, porque al menos una expresión es true.
4 > 3 && !(5 != 5) resulta true, porque las dos expresiones son true.

Operadores de Asignación

Permiten asignar valores a variables. El operador de asignación elemental es el igual (=) que asigna un valor o el resultado de una expresión a una variable, siguiendo el siguiente formato:

```
nombreVariable = valorAsignado;
```

El tipo de dato del valor asignado debe ser del mismo tipo que el de la variable a la que se pretende asignar. En caso de que no sean del mismo tipo, hay que

utilizar algún tipo de conversión como el casting, utilizando entre paréntesis el nombre del tipo de dato al que se quiere convertir.

```
int numEntero;
numEntero = 364; //Correcto
numEntero = 264.38; //Incorrecto
numEntero = (int)264.83; //Correcto, se asigna el valor 264
```

También se puede utilizar el operador de asignación para guardar en una variable el resultado de una expresión que dé como resultado un valor del mismo tipo que la variable.

```
nombreVariable = Expresión;
```

```
int numEntero;
numEntero = 364 * 2 + 266;
```

Además del operador igual (=), se pueden emplear otros operadores de asignación, que a la vez que asignan un valor realizan un cálculo:

+= (le suma a la variable un valor y guarda el resultado en la misma variable).

-= (le resta a la variable un valor y guarda el resultado en la misma variable).

***= (multiplica la variable por un valor y guarda el resultado en la misma variable).**

/= (divide la variable por un valor y guarda el resultado en la misma variable).

%= (obtiene el resto de dividir la variable por un valor y guarda el resultado en la misma variable).

```
calculo = 5;
calculo += 6; //Incrementa en 6 el valor de la variable calculo
System.out.println(calculo); //Muestra 11
calculo *= 2; //Duplica el valor de cálculo
System.out.println(calculo); //Muestra 22
```

Por tanto, una sentencia como **a+=3 es lo mismo que a=a+3**.

Operadores Incrementales

Son los operadores que nos permiten incrementar las variables en una unidad:

++ (incrementa en 1 el valor de una variable).
-- (decrementa en 1 el valor de una variable).

Ejemplo:

```
num1 = 3;
num2 = 7;
num1++; //Suma 1 a num1
System.out.println(num1); //Muestra 4
num2--; //resta 1 a num2
System.out.println(num2); //Muestra 6
```

Por tanto, una sentencia como **x++ es lo mismo que x=x+1**, y **x-- es lo mismo que x=x-1**.

Hay dos formas de utilizar el incremento y el decremento, x++ ó ++x. La diferencia estriba en el modo en el que se comporta la asignación.

Ejemplo:

```
int x=5, y=5, z;
z=x++; /* z vale 5, x vale 6 porque primero se asigna
el valor de x a z después se incrementa x. Postincremento */
z=++y; /* z vale 6, y vale 6 porque primero incrementa
la variable y, después se asigna el valor a z. Preincremento */
```

Operador Condicional.

Existe un operador condicional, que permite asignar a una variable un valor u otro dependiendo de una expresión condicional. El formato es el siguiente:

```
variable = condición ? valor1 : valor2;
```

La condición que se indica debe ser una variable booleana o una expresión condicional que resulte al evaluarla un valor de tipo booleano (true o false).

Ejemplo:

```
mensaje = (num1 >= 0) ? "Positivo" : "Negativo";
```

Si se muestra en pantalla posteriormente el valor de la variable mensaje, aparecerá "Positivo" si el valor de la variable num1 es mayor o igual que cero, y "Negativo" en caso contrario.

Precedencia de operadores en Java

(Los operadores en una misma fila tienen la misma precedencia.La precedencia disminuye según se baja en la tabla.)

( )	**[ ]**	**.**			
-(unario)	**--**	**++**	**!**		
new (tipo)					
*****	**/**	**%**			
+	-				
>	> =	<	< =		
= =	!=				
&&					
| |					
? :					
=	+=	-=	*=	/=	%=

Descripción	Operadores
operadores posfijos	op++ op--
operadores unarios	++op --op +op -op ~ !
multiplicación y división	* / %
suma y resta	+ -
desplazamiento	<< >> >>>
operadores relacionales	< > <= =>
equivalencia	== !=
operador AND	&
operador XOR	^
operador OR	|
AND booleano	&&
OR booleano	||
condicional	?:
operadores de asignación	= += -= *= /= %= &= ^= |= <<= >>= >>>=

7 Conversiones de tipo.

Java es un lenguaje fuertemente tipificado, lo que significa que es bastante estricto al momento de asignar valores a una variable. El compilador solo permite asignar un valor del tipo declarado en la variable; no obstante, en ciertas circunstancias es posible realizar conversiones que permiten almacenar en una variable un tipo diferente al declarado. En Java es posible realizar conversiones en todos los tipos básicos, con excepción de boolean, que es incompatible con el resto de los tipos.

Las conversiones de tipo pueden realizarse de dos maneras: implícitamente y explícitamente.

Conversión implícita.

Las conversiones implícitas se realizan de manera automática, es decir, el valor o expresión que se va a asignar a una variable es convertido automáticamente por el compilador, antes de almacenarlo en la variable.

Para que una conversión pueda realizarse de manera automática (implícitamente), el tipo de la variable destino debe ser de tamaño igual o superior al tipo de origen, si bien esta regla tiene dos excepciones:

a) Cuando la variable destino es entera y el origen es decimal (float o double), la conversión no podrá ser automática.

b) Cuando la variable destino es "char" y el origen es numérico; independientemente del tipo específico, la conversión no podrá ser automática.

Ejemplos de conversiones implícitas:

```
// Declaraciones.
int k = 5, p;
short s = 10;
char c = 'ñ';  // Código decimal 241
float h;

// Conversiones implícitas.
p = c; // Conversión implícita de char a int.
h = k; // Conversión implícita de int a float.
k = s; // Conversión implícita de short a int.
```

Los siguientes ejemplos de conversión implícita provocarían un error.

```
// Declaraciones.
int n;
long c = 20;
float ft = 2.4f;
char k;
byte s = 4;

// Conversiones implícitas.

n = c; // Error, el tipo destino es menor al tipo origen.
k = s; // Cuando la variable destino es "char" y el origen es numérico;
independientemente del tipo específico, la conversión no podrá ser
automática.
n = ft; // Cuando la variable destino es entera y el origen es decimal (float o
double), la conversión no podrá ser automática.
```

Conversión explícita.

Cuando no se cumplan las condiciones para una conversión implícita, esta podrá realizarse de manera explícita utilizando la expresión:

```
variable_destino = (tipo_destino) dato_origen;
```

Con esta expresión se obliga al compilador que convierta "dato_origen" a "tipo_destino" para que pueda ser almacenado en "variable_destino". A esta operación se le conoce como "casting" o "conversión".

Ejemplos de conversiones explícitas:

```
// Declaraciones.
int n;
byte k;
int p = 400;
double d = 34.6;
// Conversiones explícitas.
n = (int)d; // Se elimina la parte decimal (trunca), no se redondea.
k = (byte)p; // Se provoca una pérdida de datos, pero la conversión es
posible.

public class Conversiones {
public static void main (String [] args) {
int a = 2;
double b = 3.0;
float c = (float) (20000*a/b + 5);
```

```
System.out.println("Valor en formato float: " + c);
System.out.println("Valor en formato double: " +
(double) c);
System.out.println("Valor en formato byte: " +
(byte) c);
System.out.println("Valor en formato short: " +
(short) c);
System.out.println("Valor en formato int: " + (int) c);
System.out.println("Valor en formato long: " + (long) c);
}
}
```

8 Comentarios.

Cuando se escribe código en general es útil realizar comentarios explicativos. Los comentarios no tienen efecto como instrucciones para el ordenador, simplemente sirven para que cuando una persona lea el código pueda comprender mejor lo que lee. En Java existen dos formas de poner comentarios.

La primera es cuando la línea de comentario solo ocupa una línea de código. En este caso se pone dos barras inclinadas (//) antes del texto.

```
// Comentario de una línea
```

En el caso de comentarios de más de una línea se pone /* para empezar y */ para finalizar.
El código nos quedará de la siguiente forma:

```
/* Comentario
de varias
líneas */
```