



UD2

LENGUAJE DE PROGRAMACIÓN JAVA

MP_0485
Programación

1.1 Introducción al
lenguaje Java

Introducción

Java es un lenguaje de programación de propósito general, concurrente y orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su objetivo es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código puede escribirse una sola vez y ser ejecutado en cualquier tipo de dispositivos (PC, móvil, etc.).

Las características de Java son:

- ⊕ Sencillo: Es un lenguaje sencillo de aprender.
- ⊕ Orientado a Objetos: Posiblemente sea el lenguaje más orientado a objetos de todos los existentes; en Java, a excepción de los tipos fundamentales de variables (int, char, long...), todo es un objeto.
- ⊕ Distribuido: Java está muy orientado al trabajo en red, soportando protocolos como TCP/IP, UDP, HTTP y FTP. Por otro lado, el uso de estos protocolos es bastante sencillo comparándolo con otros lenguajes que los soportan.
- ⊕ Robusto: El compilador Java detecta muchos errores que otros compiladores solo detectarían en tiempo de ejecución o incluso nunca.
- ⊕ Seguro: Sobre todo un tipo de desarrollo: los Applet. Estos son programas diseñados para ser ejecutados en una página web.
- ⊕ Portable: En Java no hay aspectos dependientes de la implementación, todas las implementaciones de Java siguen los mismos estándares en cuanto a tamaño y almacenamiento de los datos.
- ⊕ Arquitectura Neutral: El código generado por el compilador Java es independiente de la arquitectura: podría ejecutarse en un entorno UNIX, Mac, Windows, Móvil, etc.
- ⊕ Rendimiento medio: Actualmente la velocidad de procesamiento del código Java es semejante a las de otros lenguajes orientados a objetos.
- ⊕ Multithread: Soporta de modo nativo los threads (hilos de ejecución), sin necesidad del uso de librerías específicas.

Primer ejemplo

La aplicación más pequeña posible es la que simplemente imprime un mensaje en la pantalla. Tradicionalmente, el mensaje suele ser "Hola Mundo!". Esto es justamente lo que hace el siguiente fragmento de código:

```

12  public class HolaMundo {
13
14      public static void main(String[] args) {
15          // TODO code application logic here
16          System.out.println("Hola Mundo!");
17      }
18
19  }

```

Hay que ver en detalle la aplicación anterior, línea a línea. Esas líneas de código contienen los componentes mínimos para imprimir *Hola Mundo!* en la pantalla. Es un ejemplo muy simple, que no instancia objetos de ninguna otra clase; sin embargo, accede a otra clase incluida en el JDK.

public class HolaMundo

Esta línea declara la clase **HolaMundo**. El nombre de la clase especificado en el fichero fuente se utiliza para crear un fichero *nombredeclase.class* en el directorio en el que se compila la aplicación.

En este caso, el compilador creara un fichero llamado *HolaMundo.class*.

public static void main(String args[])

Esta línea especifica un método que el intérprete Java busca para ejecutar en primer lugar. Igual que en otros lenguajes, Java utiliza una palabra clave *main* para especificar la primera función a ejecutar. En este ejemplo tan simple no se pasan argumentos.

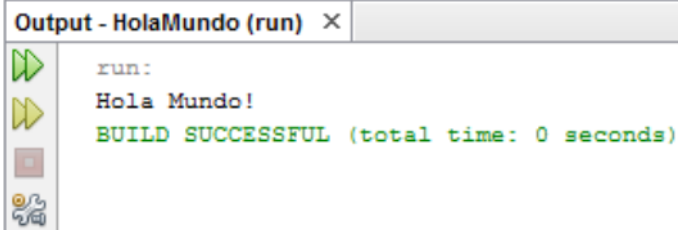
- ⊕ **public** significa que el método `main()` puede ser llamado por cualquiera, incluyendo el intérprete Java.
- ⊕ **static** es una palabra clave que le dice al compilador que `main` se refiere a la propia clase `HolaMundo` y no a ninguna instancia de la clase. De esta forma, si alguien intenta hacer otra instancia de la clase, el método `main()` no se instanciaría.
- ⊕ **void** indica que `main()` no devuelve nada. Esto es importante ya que Java realiza una estricta comprobación de tipos, incluyendo los tipos que se ha declarado que devuelven los métodos.
- ⊕ **args[]** es la declaración de un array de *Strings*. Estos son los argumentos escritos tras el nombre de la clase en la línea de comandos: `java HolaMundo arg1 arg2 ...`

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

Esta es la funcionalidad de la aplicación. Esta línea muestra el uso de un nombre de clase y método. Se usa el **método `println()`** de la **clase `out`** que está en el **paquete `System`**.

El método `println()` toma una cadena como argumento y la escribe en el *stream* de salida estándar; en este caso, la ventana donde se lanza la aplicación. La clase `PrintStream` tiene un método instanciable llamado `println()`, que lo que hace es presentar en la salida estándar del Sistema el argumento que se le pase. En este caso, se utiliza la variable o instancia de `out` para acceder al método.

El resultado sería el siguiente:



Todas las instrucciones (creación de variables, llamadas a métodos, asignaciones) se deben de finalizar con un punto y coma.

Elementos básicos

Comentarios

En java hay tres tipos de comentarios:

```
//Comentarios para una sola línea
```

```
/*  
Comentarios de una o más líneas  
*/
```

```
/**  
Comentarios de documentación de una o más líneas  
*/
```

Los dos primeros tipos de comentarios son los que todo programador conoce y se utilizan del mismo modo.

Los comentarios de documentación, colocados inmediatamente antes de una declaración (de variable o función), indican que ese comentario ha de ser colocado en la documentación que se genera automáticamente cuando se utiliza la herramienta de Java, **javadoc**, no disponible en otros lenguajes de programación. Este tipo de comentario lo veremos más adelante.

Identificadores

Los identificadores nombran variables, funciones, clases y objetos; cualquier cosa que el programador necesite identificar o usar.

Reglas para la creación de identificadores:

- ⊕ Java hace distinción entre mayúsculas y minúsculas, por lo tanto, nombres o identificadores como var1, Var1 y VAR1 son distintos.
- ⊕ Pueden estar formados por cualquiera de los caracteres del código Unicode, por lo tanto, se pueden declarar variables con el nombre: añoDeCreación, raïm, etc., aunque eso sí, el primer carácter no puede ser un dígito numérico y no pueden utilizarse espacios en blanco ni símbolos coincidentes con operadores.
- ⊕ La longitud máxima de los identificadores es prácticamente ilimitada.
- ⊕ No puede ser una palabra reservada del lenguaje ni los valores lógicos true o false.
- ⊕ No pueden ser iguales a otro identificador declarado en el mismo ámbito.
- ⊕ IMPORTANTE: Por convenio:
 - Los nombres de las variables y los métodos deberían empezar por una letra minúscula y los de las clases por mayúscula.
 - Si el identificador está formado por varias palabras, la primera se escribe en minúsculas (excepto para las clases) y el resto de las palabras se hace empezar por mayúscula (por ejemplo: añoDeCreación).
 - Estas reglas no son obligatorias, pero son convenientes ya que ayudan al proceso de codificación de un programa, así a su legibilidad. Es más sencillo distinguir entre clases y métodos o variables.

Serían identificadores válidos, por ejemplo:

- ⊕ contador
- ⊕ suma
- ⊕ edad
- ⊕ sueldoBruto
- ⊕ sueldoNeto
- ⊕ nombre_usuario
- ⊕ nombre_Completo
- ⊕ letraDni

y su uso sería, por ejemplo:

- ⊕ `int contador; // crea variable de tipo int llamada contador`
- ⊕ `float sueldoNeto; // crea variable de tipo float llamada sueldoNeto`
- ⊕ `char letraDni; // crea variable de tipo char llamada letraDni`

Tipos de datos

En Java existen dos tipos principales de datos:

- ⊕ Tipos de datos simples: Nos permiten crear variables que almacenan un solo valor. Por ejemplo, para un contador, edad, precio, etc. Son los que más vamos a utilizar por ahora.
- ⊕ Tipos de datos compuestos: Estructuras de datos más complejas que permiten almacenar muchos datos (vectores, objetos, etc.). Las veremos en futuras unidades.

Tipos de datos simples soportados por Java:

- ⊕ Para números enteros: *byte, short, int, long*
- ⊕ Para números reales: *float, double*
- ⊕ Para caracteres: *char*
- ⊕ Para valores lógicos: *boolean*

Tipo	Descripción	Memoria ocupada	Rango de valores permitidos
byte	Número entero de 1 byte	1 byte	-128 ... 127
short	Número entero corto	2 bytes	-32768 ... 32767
int	Número entero	4 bytes	-2147483648 ... 2147483647
long	Número entero largo	8 bytes	-9223372036854775808 ... 9223372036854775807
float	Número real en coma flotante de precisión simple	32 bits	$\pm 3,4 \cdot 10^{-38} \dots \pm 3,4 \cdot 10^{38}$
double	Número real en coma flotante de precisión doble	64 bits	$\pm 1,7 \cdot 10^{-308} \dots \pm 1,7 \cdot 10^{308}$
char	Un solo carácter	2 bytes	
boolean	Valor lógico	1 bit	true o false

⚡ Java no realiza una comprobación de los rangos.

Por ejemplo: si a una variable de tipo short con el valor 32.767 se le suma 1, sorprendentemente el resultado será -32.768 (no produce un error de tipo desbordamiento como en otros lenguajes de programación, sino que se comporta de forma cíclica).

Existe un tipo de dato compuesto llamado **String** que conviene conocer ya que permite representar texto. Mas adelante veremos cómo se utiliza.

Declaración de variables

La forma básica de declarar (crear) una variable es la siguiente:

tipo identificador;

Por ejemplo, creamos una variable de tipo int llamada edad:

int edad;

Las variables pueden ser inicializadas en el momento de su declaración, es decir, se les puede dar un valor inicial al crearlas. Por ejemplo, creamos una variable de tipo int llamada edad y le asignamos 25 como valor inicial:

int edad = 25;

Esto es equivalente a primero declararla y luego asignarle el valor:

```
int edad;  
edad = 25;
```

También es posible declarar varias variables en una sola línea. Por ejemplo, creamos tres variables de tipo *float* llamadas precio1, precio2 y precio3:

```
float precio1, precio2, precio3;
```

Esto es equivalente a:

```
float precio1;  
float precio2;  
float precio3;
```

A su vez, también pueden inicializarse. Por ejemplo:

```
float precio1 = 7.0, precio2 = 7.25, precio3 = 0.5;
```

Esto es equivalente a:

```
float precio1 = 7.0;  
float precio2 = 7.25;  
float precio3 = 0.5;
```

En resumen, la declaración de variables sigue el siguiente patrón:

```
Tipo identificador [=valor][,identificador [=valor]...];
```

Es decir, es **obligatorio indicar el tipo y el identificador** (además de terminar en punto y coma como todas las instrucciones). Opcionalmente (indicado entre corchetes) se puede inicializar y/o se pueden declarar más variables.

Si una variable no ha sido inicializada, Java le asigna un valor por defecto.

Este valor es:

- ⊕ Para las variables de tipo **numérico**, el valor por defecto es cero (**0**).
- ⊕ Las variables de tipo **char**, el valor **'\u0000'**.
- ⊕ Las variables de tipo **boolean**, el valor **false**.
- ⊕ Para las variables de tipo referencial (**objetos**), el valor **null**.

Es una **buena práctica** inicializar siempre todas las variables.

Palabras clave

Las siguientes son palabras clave que no se pueden utilizar como identificadores ya que Java las utiliza para otras cosas:

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

Palabras reservadas

Además, el lenguaje se reserva unas cuantas palabras más, pero que hasta ahora no tienen un cometido específico. Son:

cast	future	generic	inner
operator	outer	rest	var

Ámbito de una variable

El **ámbito** de una variable es la porción del programa donde dicha variable puede utilizarse.

El ámbito de una variable depende del lugar del programa donde es declarada, pudiendo pertenecer a cuatro categorías distintas.

- ⊕ Variable local.
- ⊕ Atributo.
- ⊕ Parámetro de un método.
- ⊕ Parámetro de un tratador de excepciones.

Por ahora utilizaremos solo variables locales, las demás categorías las veremos en posteriores unidades.

Una variable local se declara dentro del cuerpo de un método de una clase y es visible únicamente dentro de dicho método.

Se puede declarar en cualquier lugar del cuerpo, incluso después de instrucciones ejecutables, aunque es una buena costumbre declararlas justo al principio. También pueden declararse variables dentro de un bloque con llaves {...}. En ese caso, solo serán “visibles” dentro de dicho bloque.

Por ejemplo (No es necesario entender lo que hace el programa):

```
14  [ ] public static void main(String[] args) {  
15  
16      int i;  
17  
18      for (i=0;i<10;i++)  
19          System.out.println(i);  
20  }
```

En este ejemplo existe una variable local: **int i**; únicamente puede utilizarse dentro del bloque **main** donde fue creada.

Ámbito de una variable

Al declarar una variable puede utilizarse la palabra reservada **final** para indicar que el valor de la variable no podrá modificarse (es una constante).

Por ejemplo, creamos variable constante tipo int llamada x con valor 18:

final int x = 18;

Por ejemplo, creamos variable constante tipo float llamada pi con valor 3.14:

```
final float pi = 3.14;
```

Si posteriormente intentamos modificar sus valores se producirá un error y Java nos avisará de que no es posible.

```
x = 20; // no permitido, produce error
```

```
pi = 7; // no permitido, produce error
```

Por lo tanto una variable precedida de la palabra **final** se convierte en una **constante**. O lo que es lo mismo, para definir una constante en Java deberemos preceder su declaración de la palabra reservada **final**.

Operadores

Los **operadores** son una parte indispensable de la programación ya que nos permiten realizar cálculos matemáticos y lógicos, entre otras cosas. Los operadores pueden ser:

- ⊕ Aritméticos: sumas, restas, etc.
- ⊕ Relacionales: menor, menor o igual, mayor, mayor o igual, etc.
- ⊕ Lógicos: and, or, not, etc.
- ⊕ Bits: prácticamente no los utilizaremos en este curso.
- ⊕ Asignación: =

Aritméticos

Operador	Formato	Descripción
+	op1 + op2	Suma aritmética de dos operandos.
-	op1 - op2 -op1	Resta aritmética de dos operandos. Cambio de signo.
*	op1 * op2	Multiplicación de dos operandos
/	op1 / op2	División entera de dos operandos
%	op1 % op2	Resto de la división entera (o módulo)
++	++op1 op1++	Incremento unitario
--	--op1 op1--	Decremento unitario

El operador - puede utilizarse en su versión unaria (- op1) y la operación que realiza es la de invertir el signo del operando. Los operadores ++ y -- realizan un incremento y un decremento unitario respectivamente. Es decir:

x++ equivale a **x = x + 1**

x-- equivale a **x = x - 1**

Los operadores ++ y -- admiten notación postfija y prefija:

- ⊕ **op1++**: Primero se ejecuta la instrucción en la cual está inmerso y después se incrementa op1.
- ⊕ **op1--**: Primero se ejecuta la instrucción en la cual está inmerso y después se decrementa op1.
- ⊕ **++op1**: Primero se incrementa op1 y después ejecuta la instrucción en la cual está inmerso.
- ⊕ **--op1**: Primero se decrementa op1 y después ejecuta la instrucción en la cual está inmerso.

Los operadores incrementales suelen utilizarse a menudo en los bucles (estructuras repetitivas). Lo veremos más adelante.

Relacionales

Operador	Formato	Descripción
>	op1 > op2	Devuelve true (cierto) si op1 es mayor que op2
<	op1 < op2	Devuelve true (cierto) si op1 es menor que op2
>=	op1 >= op2	Devuelve true (cierto) si op1 es mayor o igual que op2
<=	op1 <= op2	Devuelve true (cierto) si op1 es menor o igual que op2
==	op1 == op2	Devuelve true (cierto) si op1 es igual a op2
!=	op1 != op2	Devuelve true (cierto) si op1 es distinto de op2

Los operadores relacionales actúan sobre valores enteros, reales y caracteres (char); y devuelven un valor del tipo boolean (true o false).

Ejemplo:

```

15 public static void main(String[] args){
16
17     double op1,op2;
18     char op3,op4;
19
20     op1=1.34;
21     op2=1.35;
22     op3='a';
23     op4='b';
24
25     System.out.println("op1=" + op1 + " op2=" + op2);
26     System.out.println("op1>op2 = " + (op1 > op2));
27     System.out.println("op1<op2 = " + (op1 < op2));
28     System.out.println("op1==op2 = " + (op1 == op2));
29     System.out.println("op1!=op2 = " + (op1 != op2));
30     System.out.println("'a'>'b' = " + (op3 > op4));
31
32 }

```

Salida:

```

run:
op1=1.34 op2=1.35
op1>op2 = false
op1<op2 = true
op1==op2 = false
op1!=op2 = true
'a'>'b' = false
BUILD SUCCESSFUL (total time: 0 seconds)

```

Lógicos

Operador	Formato	Descripción
&&	op1 && op2	Y lógico. Devuelve true (cierto) si son ciertos op1 y op2
	op1 op2	O lógico. Devuelve true (cierto) si son ciertos op1 o op2
!	! op1	Negación lógica. Devuelve true (cierto) si es false op1.

Estos operadores actúan sobre operadores o expresiones lógicas, es decir, aquellos que se evalúan a cierto o falso (true / false).

Ejemplo:

```

15 public static void main(String[] args){
16
17     boolean a, b, c, d;
18
19     a=true;
20     b=true;
21     c=false;
22     d=false;
23
24     System.out.println("true Y true = " + (a && b) );
25     System.out.println("true Y false = " + (a && c) );
26     System.out.println("false Y false = " + (c && d) );
27     System.out.println("true O true = " + (a || b) );
28     System.out.println("true O false = " + (a || c) );
29     System.out.println("false O false = " + (c || d) );
30     System.out.println("NO true = " + !a);
31     System.out.println("NO false = " + !c);
32     System.out.println("(3 > 4) Y true = " + ((3 >4) && a) );
33
34 }
35

```

Salida:

```

run:
true Y true = true
true Y false = false
false Y false = false
true O true = true
true O false = true
false O false = false
NO true = false
NO false = true
(3 > 4) Y true = false
BUILD SUCCESSFUL (total time: 0 seconds)

```

De asignación

El operador de asignación es el símbolo =

variable = expresión

Asigna a la variable el resultado de evaluar la expresión de la derecha. Es posible combinar el operador de asignación con otros operadores para, de forma abreviada, realizar un cálculo y asignarlo a una variable:

Operador	Formato	Equivalencia
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2
&=	op1 &= op2	op1 = op1 & op2
=	op1 = op2	op1 = op1 op2
^=	op1 ^= op2	op1 = op1 ^ op2
>>=	op1 >>= op2	op1 = op1 >> op2
<<=	op1 <<= op2	op1 = op1 << op2
>>>=	op1 >>>= op2	op1 = op1 >>> op2

Expresiones

Una expresión es la combinación de varios operadores y operandos. Por ejemplo, tenemos las siguientes expresiones:

$7 + 5 * 4 - 2$
 $10 + (1 \% 5)$
 $(7 * x) \leq N$
 etc.

El lenguaje Java evalúa las expresiones aplicando los operadores uno a uno siguiendo un orden específico. Este orden se detalla en el siguiente punto.

Precedencia de operadores

Indica el **orden en el que se evalúan los operadores** en una expresión. No es necesario saberse toda la lista de memoria, pero **es importante conocer al menos los más utilizados**: matemáticos, relacionales, lógicos y de asignación.

Algunos de estos operadores los veremos en unidades posteriores, ahora mismo no es necesario que sepas que hacen.

1. Operadores postfijos: `[]`. (paréntesis)
2. Operadores unarios: `++expr`, `--expr`, `-expr`, `~ !`
3. Creación o conversión de tipo: `new (tipo)expr`
4. **Multiplicación y división:** `*`, `/`, `%`

5. **Suma y resta:** +, -
6. Desplazamiento de bits: <<, >>, >>>
7. **Relacionales:** <, >, <=, >=
8. **Igualdad y desigualdad:** ==, !=
9. AND a nivel de bits: &
10. **AND lógico:** &&
11. XOR a nivel de bits: ^
12. OR a nivel de bits: |
13. **OR lógico:** ||
14. Operador condicional: ? :
15. **Asignación:** =, +=, -=, *=, /=, %=, ^=, &=, |=, >>=, <<=

Precedencia de operadores

Clase especial llamada **Math** dentro del paquete java.lang. Esta clase posee muchos métodos muy interesantes para realizar cálculos matemáticos complejos como cálculo de potencias, raíces cuadradas, valores absolutos, seno, coseno, etc.


















Por ejemplo:

```
double x = Math.pow(3,3); // Potencia 3 ^ 3
double y = Math.sqrt(9); // Raiz cuadrada de 9
```

También posee constantes como:

```
double PI -> El numero  $\Pi$  (3,14159265...)
double E -> El numero e (2, 7182818245...)
```

Algunos ejemplos de otros métodos:

 E	double	^
 PI	double	
 IEEEremainder (double f1, double f2)	double	
 abs (double a)	double	
 abs (float a)	float	
 abs (int a)	int	
 abs (long a)	long	
 acos (double a)	double	
 addExact (int x, int y)	int	
 addExact (long x, long y)	long	
 asin (double a)	double	
 atan (double a)	double	
 atan2 (double y, double x)	double	
 cbrt (double a)	double	
 ceil (double a)	double	
 copySign (double magnitude, double sign)	double	
 copySign (float magnitude, float sign)	float	v

Literales

A la hora de tratar con valores de los tipos de datos simples (y Strings) se utiliza lo que se denomina “literales”. Los literales son elementos que sirven para representar un valor en el código fuente del programa.

En Java existen literales para los siguientes tipos de datos:

- Lógicos (boolean).
- Carácter (char).
- Enteros (byte, short, int y long).
- Reales (double y float).
- Cadenas de caracteres (String).

Literales lógicos

Son únicamente dos, las palabras reservadas *true* y *false*.

Ejemplo: boolean activado = false;

Literales enteros

Los literales de tipo entero: *byte*, *short*, *int* y *long* pueden expresarse en decimal (base 10), octal (base 8) o hexadecimal (base 16). Además, puede añadirse al final del mismo la letra *L* para indicar que el entero es considerado como long (64bits).

En Java, el compilador identifica un entero decimal (base 10) al encontrar un numero cuyo primer dígito es cualquier símbolo decimal excepto el cero (del 1 al 9). A continuación, pueden aparecer dígitos del 0 al 9.

La letra *L* al final de un literal de tipo entero puede aplicarse a cualquier sistema de numeración e indica que el numero decimal sea tratado como un entero largo (de 64 bits). Esta letra *L* puede ser mayúscula o minúscula, aunque es aconsejable utilizar la mayúscula ya que de lo contrario puede confundirse con el dígito uno (1) en los listados.

Ejemplo:

```
long max1 = 9223372036854775807L; //valor máximo para un entero largo
```

Literales enteros

Los literales de tipo real sirven para indicar valores *float* o *double*. A diferencia de los literales de tipo entero, no pueden expresarse en octal o hexadecimal.

Existen dos formatos de representación: mediante su parte entera, el punto decimal (.) y la parte fraccionaria; o mediante notación exponencial o científica:

Ejemplos equivalentes:

```
3.1415
0.31415e1
.31415e1
0.031415E+2
.031415e2
314.15e-2
31415E-4
```

Al igual que los literales que representan enteros, se puede poner una letra como sufijo. Esta letra puede ser una *F* o una *D* (mayúscula o minúscula indistintamente).

F --> Trata el literal como de tipo *float*.

D --> Trata el literal como de tipo *double*.

Ejemplo:

```
3.1415F
.031415d
```

Literales carácter

Los literales de tipo carácter se representan siempre entre comillas simples. Entre las comillas simples puede aparecer:

- ⊕ Un **símbolo** (letra) siempre que el carácter este asociado a un código Unicode. Ejemplos: *'a'*, *'B'*, *'{'*, *'ñ'*, *'á'*.
- ⊕ Una **"secuencia de escape"**. Las secuencias de escape son combinaciones del símbolo contrabarra \ seguido de una letra, y sirven para representar caracteres que no tienen una equivalencia en forma de símbolo.

Las posibles secuencias de escape son:

`\n` ----> Nueva Línea.
`\t` ----> Tabulador.
`\r` ----> Retroceso de Carro.
`\f` ----> Comienzo de Pagina.
`\b` ----> Borrado a la Izquierda.
`\\` ----> El carácter barra inversa (`\`).
`\'` ----> El carácter prima simple (`'`).
`\"` ----> El carácter prima doble o bi-prima (`"`).

Por ejemplo:

Para imprimir una diagonal inversa se utiliza: `\\`

Para imprimir comillas dobles en un String se utiliza: `\"`

Literales cadenas

Los Strings o cadenas de caracteres no forman parte de los tipos de datos elementales en Java, sino que son instanciados a partir de la clase *java.lang.String*, pero aceptan su inicialización a partir de literales de este tipo, por lo que se tratan en este punto.

Un literal de tipo String va encerrado entre comillas dobles (`"`) y debe estar incluido completamente en una sola línea del programa fuente (no puede dividirse en varias líneas).

Entre las comillas dobles puede incluirse cualquier carácter del código Unicode (o su código precedido del carácter `\`) además de las secuencias de escape vistas anteriormente en los literales de tipo carácter. Así, por ejemplo, para incluir un cambio de línea dentro de un literal de tipo String deberá hacerse mediante la secuencia de escape `\n` :

Ejemplo:

```
System.out.println("Primera línea\nSegunda línea del string\n");  
System.out.println("Hola");
```

La visualización del String anterior mediante *println()* produciría la siguiente salida por pantalla:

Primera línea
Segunda línea del string
Hola

La forma de incluir los caracteres: comillas dobles (") y contrabarra (\) es mediante las secuencias de escape \" y \\ respectivamente (o mediante su código Unicode precedido de \).

Si el String es demasiado largo y debe dividirse en varias líneas en el fichero fuente, puede utilizarse el operador de concatenación de Strings (+) de la siguiente forma:

"Este String es demasiado largo para estar en una línea de " +
"fichero fuente y se ha dividido en dos."

Salida y entrada estándar

Salida estándar

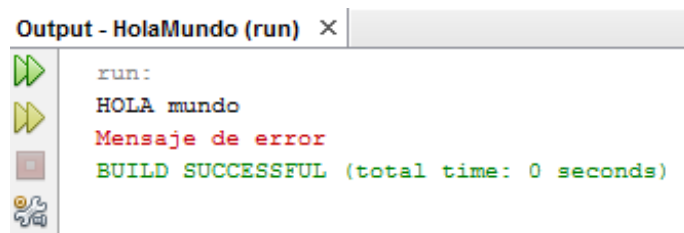
Ya hemos visto el uso de **System.out** para mostrar información por pantalla:

- ⊕ **print(...)** imprime texto por pantalla
- ⊕ **println(...)** imprime texto por pantalla e introduce un salto de línea.

La utilización de *System.err* sería totalmente análoga para enviar los mensajes producidos por errores en la ejecución (es el canal que usa también el compilador para notificar los errores encontrados). Por ejemplo, para presentar el mensaje de saludo habitual por pantalla, y después un mensaje de error, tendríamos la siguiente clase (aunque en realidad toda la información va a la consola de comandos donde estamos ejecutando el programa):

```
14 public static void main(String[] args) {  
15  
16     System.out.print("HOLA ");  
17     System.out.println("mundo");  
18     System.err.println("Mensaje de error");  
19 }
```

Y la salida sería la siguiente:



```
Output - HolaMundo (run) ×  
run:  
HOLA mundo  
Mensaje de error  
BUILD SUCCESSFUL (total time: 0 seconds)
```

También pueden imprimirse variables de cualquier tipo, así como combinaciones de texto y variables concatenadas con el operador +

```
14  □ public static void main(String[] args) {  
15      String nombre = "Pepito";  
16      int edad = 25;  
17      System.out.println(nombre);  
18      System.out.println(edad);  
19      System.out.println(nombre + " tiene " + edad + " años");  
20  }
```

Y la salida seria la siguiente:

```
Pepito  
25  
Pepito tiene 25 años.
```

Entrada estándar

La entrada estándar (leer información del teclado, escrita por el usuario) es un poco más compleja. Hay varias formas de hacerlo, pero la más sencilla es utilizar la clase Scanner. Siempre que queramos leer información del teclado primero tendremos que declarar un objeto Scanner que lea de la entrada estándar *System.in* así:

```
Scanner reader = new Scanner(System.in);
```

NOTA: En este ejemplo hemos creado un objeto Scanner llamado reader pero podríamos ponerle cualquier nombre.

Ahora podremos utilizar *reader* tantas veces como queramos para leer información del teclado. Por ejemplo:

```
String texto = reader.nextLine();
```

El método **reader.nextLine()** recogerá el texto que el usuario escriba por teclado (hasta presionar la tecla Intro) y lo guardará en **texto** (de tipo String).

Existen mucho otros métodos según el tipo de dato que se quiera leer:

- ⊕ **nextByte()**: obtiene un numero entero tipo byte.
- ⊕ **nextShort()**: obtiene un numero entero tipo short.
- ⊕ **nextInt()**: obtiene un numero entero tipo int.
- ⊕ **nextLong()**: obtiene un numero entero tipo long.
- ⊕ **nextFloat()**: obtiene un número real float.
- ⊕ **nextDouble()**: obtiene un número real double.
- ⊕ **next()**: obtiene el siguiente token (texto hasta un espacio).

No existen métodos de la clase Scanner para obtener directamente booleanos ni para obtener un solo carácter.

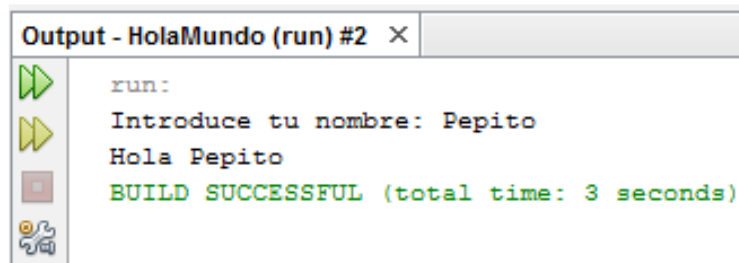
IMPORTANTE: Para poder utilizar la clase Scanner es necesario importarla desde el paquete *java.util* de Java. Es decir, arriba del todo (antes del `public class...`) hay que escribir la siguiente sentencia:

```
import java.util.Scanner;
```

Ejemplo en el que leemos una cadena de texto y la mostramos por pantalla:

```
12  import java.util.Scanner;
13
14  public class EjemploScanner {
15
16      public static void main(String[] args){
17
18          String nombre;
19
20          Scanner entrada = new Scanner(System.in);
21
22          System.out.print("Introduce tu nombre: ");
23
24          nombre = entrada.nextLine();
25
26          System.out.println("Hola " + nombre);
27
28      }
29  }
```

Salida:



```
run:
Introduce tu nombre: Pepito
Hola Pepito
BUILD SUCCESSFUL (total time: 3 seconds)
```

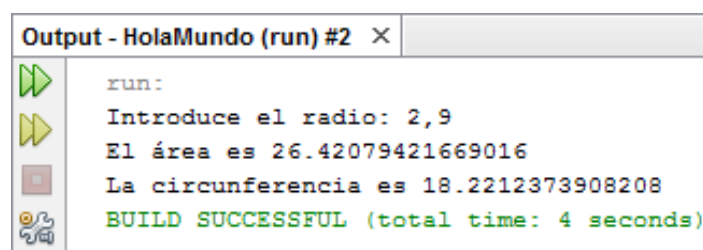
Ejemplo en el que leemos un valor tipo *double*. El programa pide al usuario que introduzca el radio de un círculo, luego calcula su área y circunferencia, por último lo muestra por pantalla.

```

12  import java.util.Scanner;
13
14  public class EjemploScanner {
15
16      public static void main(String[] args){
17
18          double radio, area, circunferencia;
19
20          Scanner entrada = new Scanner(System.in);
21
22          System.out.print("Introduce el radio: ");
23
24          radio = entrada.nextDouble();
25
26          // Se hace uso de la librería Math para usar PI y la potencia(pow)
27          area = Math.PI * Math.pow(radio, 2);
28
29          circunferencia = 2 * Math.PI * radio;
30
31          System.out.println("El área es " + area);
32
33          System.out.println("La circunferencia es " + circunferencia);
34      }
35  }

```

Salida:



```
run:
Introduce el radio: 2,9
El área es 26.42079421669016
La circunferencia es 18.2212373908208
BUILD SUCCESSFUL (total time: 4 seconds)
```

Referencias

Apuntes elaborados a partir de la siguiente documentación:

- [1] Apuntes Fernando Barber y Ricardo Ferris. Universidad de Valencia.
- [2] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.
- [3] Apuntes de Programación de Carlos Cacho y Raquel Torres. Ceedcv.
- [4] Apuntes de Programación Edix Digital Workers.

Licencia



[CC BY-NC-SA 3.0 ES](https://creativecommons.org/licenses/by-nc-sa/3.0/es/) Reconocimiento - No Comercial - Compartir Igual (by-nc-sa)

No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

NOTA: Esta es una obra derivada de la original realizada por Carlos Cacho y Raquel Torres.