MÓDULO 0485 - PROGRAMACIÓN

UT 2-

ELEMENTOS Y SINTAXIS DE PROGRAMACIÓN

Técnico Superior En Desarrollo De Aplicaciones Web

1º DAW (Grupo 1WV) (Vespertino)

Curso 2025-26

Profesor: Javier Rojo

ÍNDICE

PARTE TEÓRICA/CONCEPTUAL

- Elementos y sintaxis de programación

PARTE PRÁCTICA

- Introducción al entorno
- Primeros programas en Java

ELEMENTOS Y SINTAXIS DE PROGRAMACIÓN

ELEMENTOS Y SINTAXIS DE PROGRAMACIÓN

ÍNDICE

INTRODUCCIÓN

JAVA

1. **Java** es un lenguaje de programacion de proposito general, concurrente y orientado a objetos que fue disenado especificamente para tener tan pocas dependencias de implementacion como fuera posible.

2. Su objetivo es permitir que los **desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo** (conocido en ingles 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, movil, etc.).

CARACTERÍSTICAS DE JAVA

- **Sencillo** : Es un lenguaje sencillo de aprender.
- **Orientado a Objetos** : Posiblemente sea el lenguaje mas 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 esta 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 disenados para ser ejecutados en una pagina web.

CARACTERÍSTICAS DE JAVA

- **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, Movil, etc.

- **Rendimiento medio**: Actualmente la velocidad de procesado 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 especificas.

PRIMER EJEMPLO

PRIMER EJEMPLO

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

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

```
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    }
```

PRIMER EJEMPLO

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

```
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  }
```

PRIMER EJEMPLO

public class HolaMundo

1. 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 creará un fichero llamado *HolaMundo.class*.

```
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    }
```

PRIMER EJEMPLO

public static void main(String args[])

1. Esta línea especifica un método que el intérprete Java busca para ejecutar en primer lugar.

2. Igual que en otros lenguajes, Java utiliza una palabra clave **main** para especificar la primera función a ejecutar.

3. Es similar a la palabra reservada *Proceso* en PSeINT, es decir, será donde comenzará a ejecutarse nuestra aplicación.

```
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  }
```

PRIMER EJEMPLO

```
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    }
```

public static void main(String args[])

1. Significado de cada parte de su nombre:

   - **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 …

PRIMER EJEMPLO

```
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    }
```

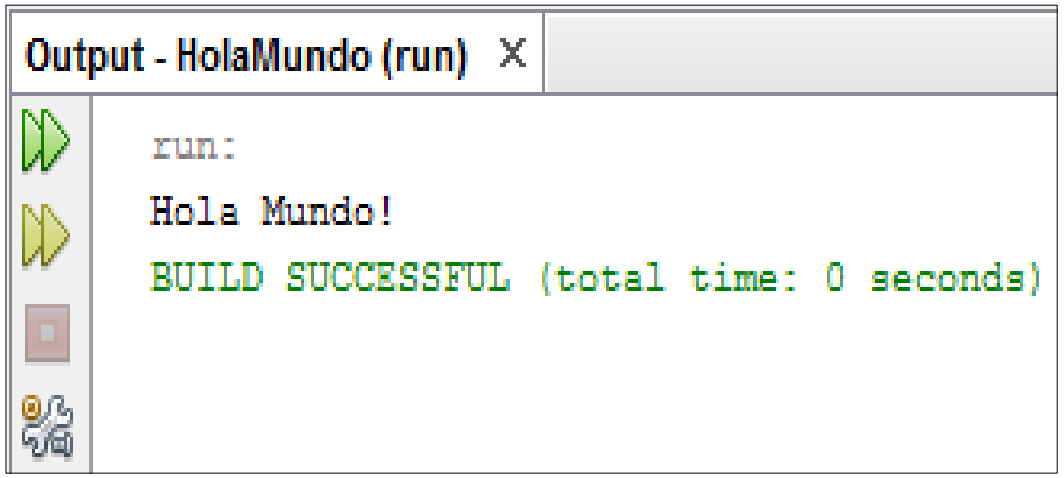
System.out.println("Hola Mundo!");

1. Esta es la funcionalidad de la aplicación.
2. 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**.
3. 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.
4. Es similar a la palabra reservada *Escribir* en PSeINT.
5. 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.
6. En este caso, se utiliza la variable o instancia de out para acceder al método.

PRIMER EJEMPLO

```
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    }
```

El resultado del programa sería:



EXTRA: ESTRUCTURA DE UN PROYECTO JAVA

ESTRUCTURA DE UN PROYECTO JAVA

Proyecto simple (solo código sin uso de gestores de dependencias)

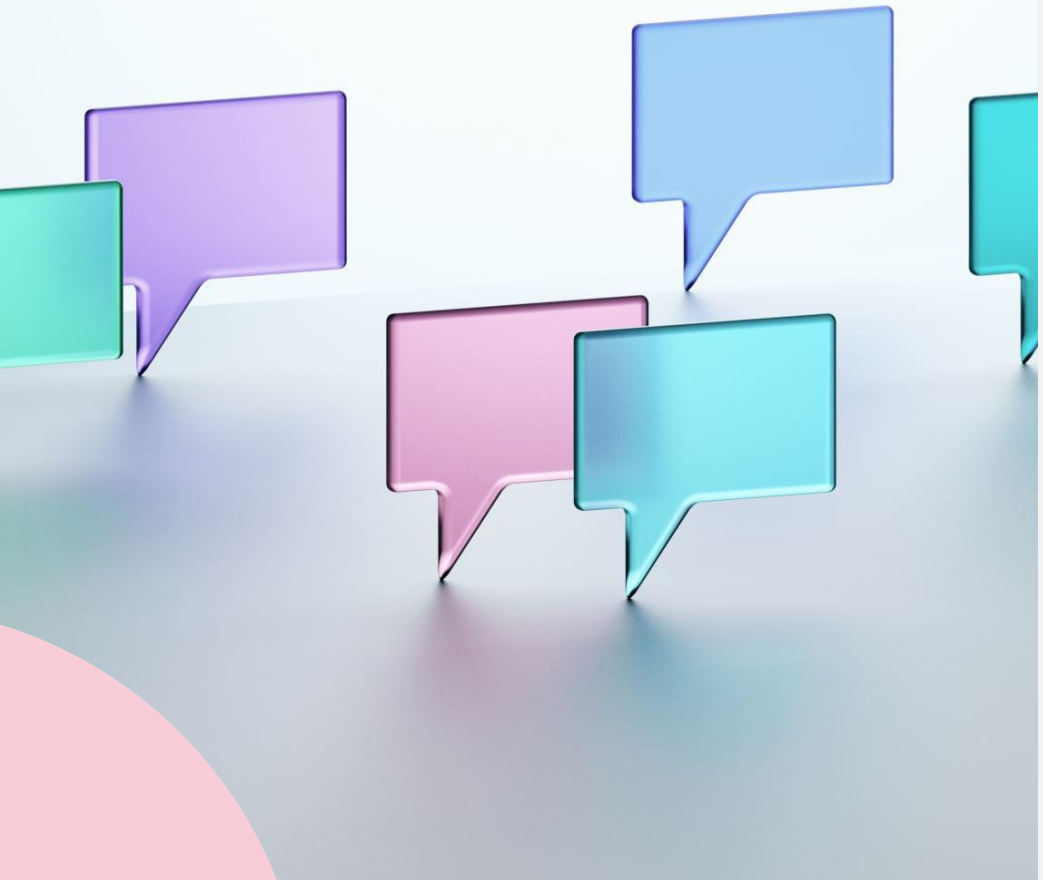
```
MiProyecto/
├── src/              # Código fuente
│   └── main/
│       └── java/
│           └── paquete/
│               └── MiClase.java
├── bin/              # Archivos .class compilados (a veces se crea al compilar)
├── lib/              # Librerías externas (.jar)
├── docs/             # Documentación (JavaDoc, manuales, etc.)
└── README.md         # Información del proyecto
```

ESTRUCTURA DE UN PROYECTO JAVA

Proyecto en entornos empresariales (con pruebas y uso de gestores de dependencias como Maven)

```
MiProyecto/
├── pom.xml                 # Archivo de configuración de Maven
└── src/
    ├── main/
    │   ├── java/           # Código fuente (.java)
    │   ├── resources/      # Archivos de configuración (XML, properties, etc.)
    │   └── webapp/         # (Opcional) Archivos web (HTML, JSP, etc.)
    └── test/
        ├── java/           # Código de pruebas unitarias (JUnit, etc.)
        └── resources/      # Recursos de prueba
```

ELEMENTOS BÁSICOS



1. COMENTARIOS

COMENTARIOS

En Java hay tres tipos de comentarios:

```
// comentarios para una sola línea
```

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

```
/** comentario de documentación, de una o más
líneas (JavaDoc)
*/
```

COMENTARIOS

```
// comentarios para una sola línea
```

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

```java
public class ComentariosEjemplo {

    public static void main(String[] args) {
        // Este es un comentario de UNA línea.
        // Sirve para anotar o explicar una parte concreta del código.

        /*
         * Este es un comentario de VARIAS líneas.
         * Se suele usar cuando necesitas explicar algo más extenso,
         * o incluso para "desactivar" un bloque de código temporalmente.
         */

        System.out.println("Ejemplo de comentarios en Java");
    }
}
```

1. Los dos primeros tipos de comentarios son los que todo programador conoce y se utilizan del mismo modo. **Sirven para explicar lo que hace el código**

COMENTARIOS

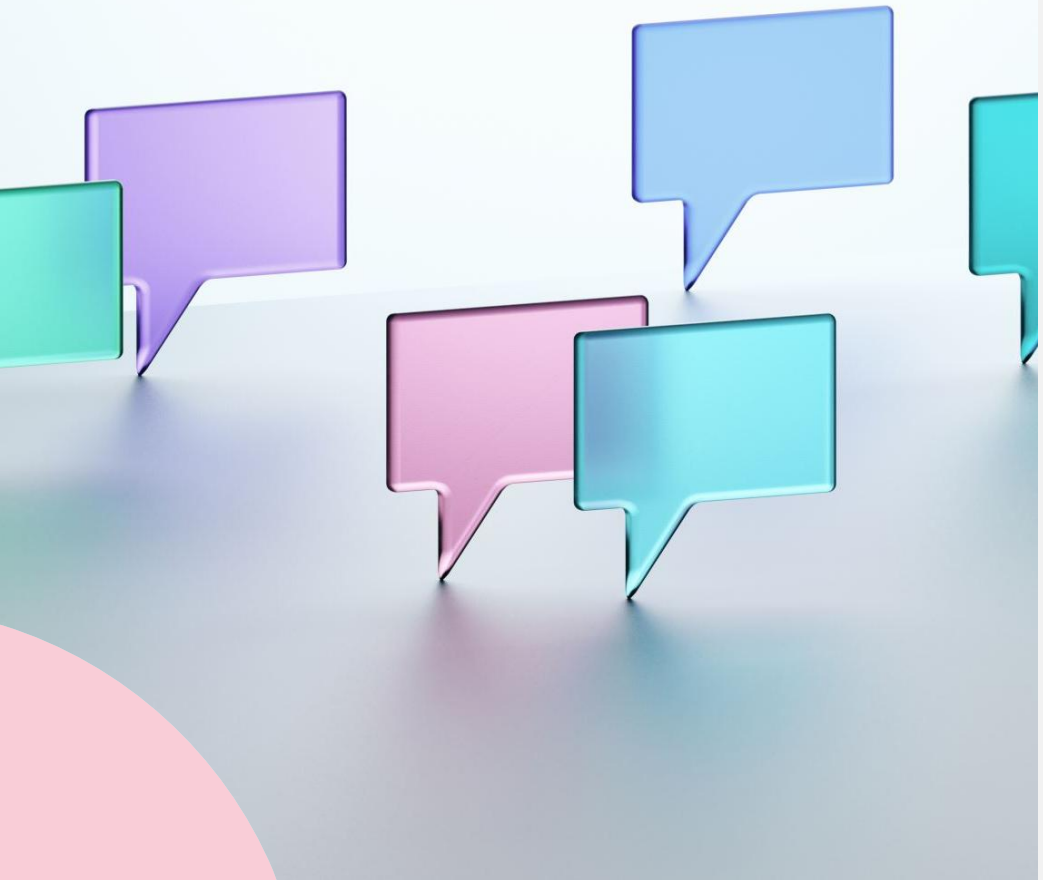
/** comentario de documentación, de una o más líneas (JavaDoc)
*/

2. 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.**

```java
/**
 * Clase Calculadora que realiza operaciones matemáticas básicas.
 *
 * <p>Este comentario es de tipo Javadoc.
 * Se utiliza para documentar clases, métodos o atributos.
 * Puede incluir etiquetas como:
 * - @author
 * - @param
 * - @return
 */
public class Calculadora {

    /**
     * Suma dos números enteros.
     * @param a primer número
     * @param b segundo número
     * @return resultado de la suma
     */
    public int sumar(int a, int b) {
        return a + b;
    }
}
```



2. IDENTIFICADORES

IDENTIFICADORES

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

Reglas para la creación de identificadores:

1. **Java hace distinción entre mayúsculas y minúsculas**, por lo tanto, nombres o identificadores como var1, Var1 y VAR1 son distintos.

2. 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**
   - **no** pueden utilizarse **espacios en blanco** ni símbolos coincidentes con operadores.

3. No obstante, lo más recomendable es:

   - no utilizar caracteres especiales, como la ñ o tildes,
   - declarar las variables y funciones en inglés para que el código sea entendido por el mayor número de programadores.

IDENTIFICADORES

Reglas para la creación de identificadores:

4. La **longitud** máxima de los identificadores es prácticamente **ilimitada**.

5. **No** puede ser una **palabra reservada del lenguaje** ni los valores lógicos true o false.

6. **No** pueden ser i**guales a otro identificador** declarado en el mismo ámbito.

IDENTIFICADORES

Convenio de Java (**no es obligatorio pero sí recomendable seguirlo**):

1. Los **nombres** de las **variables y** los **métodos** deberían **empezar** por una **letra minúscula** y los de las **clases por mayúscula**.

2. Si el identificador está formado por **varias palabras**, la **primera** se escribe en **minúsculas** (excepto para las clases) y el **resto** de palabras se hace **empezar por mayúscula** (por ejemplo: anioDeCreacion).

3. Estas **reglas** no son obligatorias, pero son **convenientes** ya que ayudan al proceso de codificación de un programa, así como a su legibilidad. Es más sencillo distinguir entre clases y métodos o variables.

IDENTIFICADORES

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

TIPOS DE DATOS

En Java, al igual que en PSeINT, existen dos tipos principales de datos:

1. **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.

2. **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

Tipos de datos simples soportados por Java:

- Para números enteros: **byte**, **short**, **int, long**
  - **En PSeINT**: Definir var como Entero
- Para números reales: **float**, **double**
  - **En PSeINT**: Definir var como Real
- Para caracteres: **char**
  - **En PSeINT**: Definir var como Cadena (aunque tiene una longitud mayor)
- Para valores lógicos: **boolean**.
  - **En PSeINT**: Definir var como Logico

TIPOS DE DATOS

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	$\pm 3{,}4*10^{-38}$ … $\pm 3{,}4*10^{38}$
double	Número real en coma flotante de precisión doble	64 bits	$\pm 1{,}7*10^{-308}$ … $\pm 1{,}7*10^{308}$
char	Un solo carácter	2 bytes	
boolean	Valor lógico	1 bit	true o false

¡OJO! **Java no realiza** una **comprobación** de los **rangos**. Si os salís de rango, el valor de "estropeará", pero no os avisará

TIPOS DE DATOS

¡OJO! **Java no realiza** una **comprobación** de los **rangos**. Si os salís de rango, el valor de "estropeará", pero no os avisará

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 → se comporta de forma cíclica.

TIPOS DE DATOS

Tipos de datos compuestos soportados por Java:

- Vectores, objetos, etc. → Los iremos viendo en las siguientes unidades.
- Existe un tipo de dato compuesto llamado **String** que conviene conocer ya que permite representar texto. Más adelante veremos cómo se utiliza.

DECLARACIÓN DE VARIABLES

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;

DECLARACIÓN DE VARIABLES

- 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;

DECLARACIÓN DE VARIABLES

Inicialización de la variable

- 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;
```

DECLARACIÓN DE VARIABLES

Inicialización de la variable

- A su vez, si declaramos varias variables en una misma línea, 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;

DECLARACIÓN DE VARIABLES

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.

DECLARACIÓN DE VARIABLES

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

- Este valor es:

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

Cuidado con esto, que no en todos los lenguajes es así. En otros, como C, os encontraréis basura de la información anterior que hubiese en esa dirección de memoria

Es una buena práctica inicializar siempre todas las variables.
¡Siempre deberéis hacerlo!

DECLARACIÓN DE VARIABLES

Nombre de las variables

1. Existe una lista de palabras clave que no se pueden utilizar como identificadores ya que Java las utiliza para otras cosas:

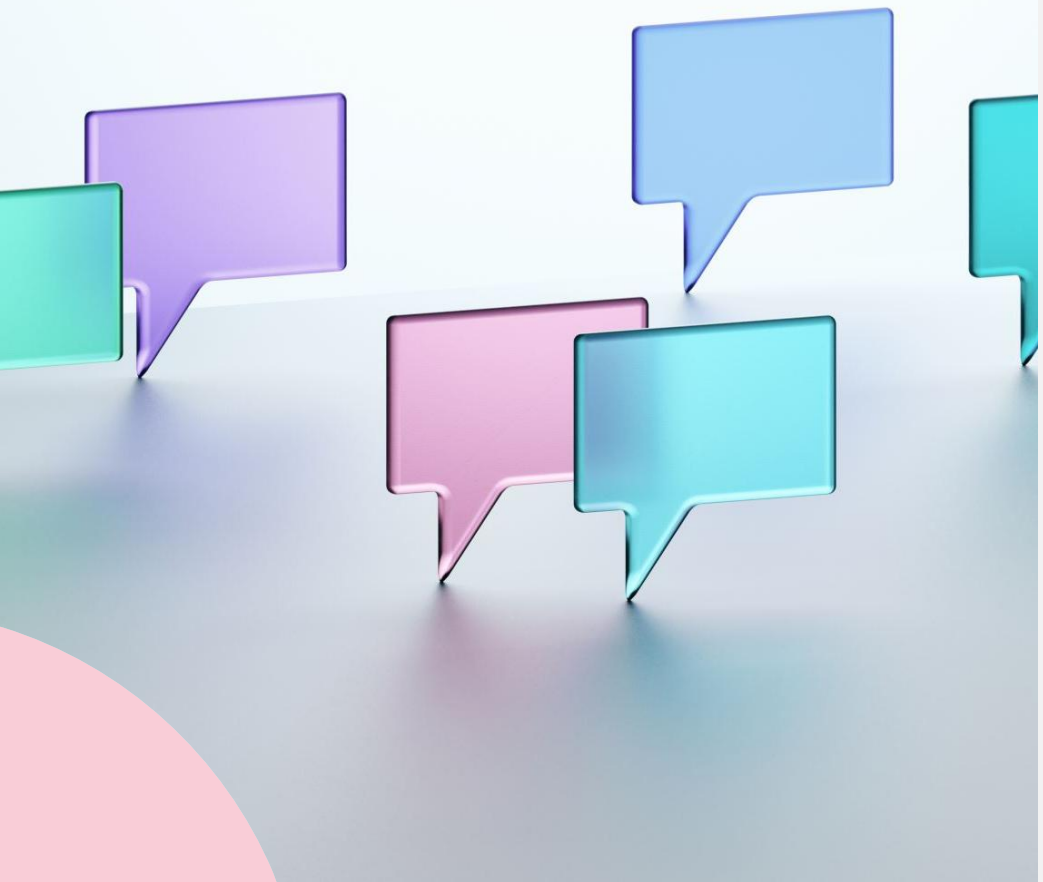
abstract	continue	for	new	switch
boolean	default	goto	null	synchronized
break	do	if	package	this
byte	double	implements	private	threadsafe
byvalue	else	import	protected	throw
case	extends	instanceof	public	transient
catch	false	int	return	true
char	final	interface	short	try
class	finally	long	static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

DECLARACIÓN DE VARIABLES

Nombre de las variables

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

cast	uture	generic	inner
operator	outer	rest	var

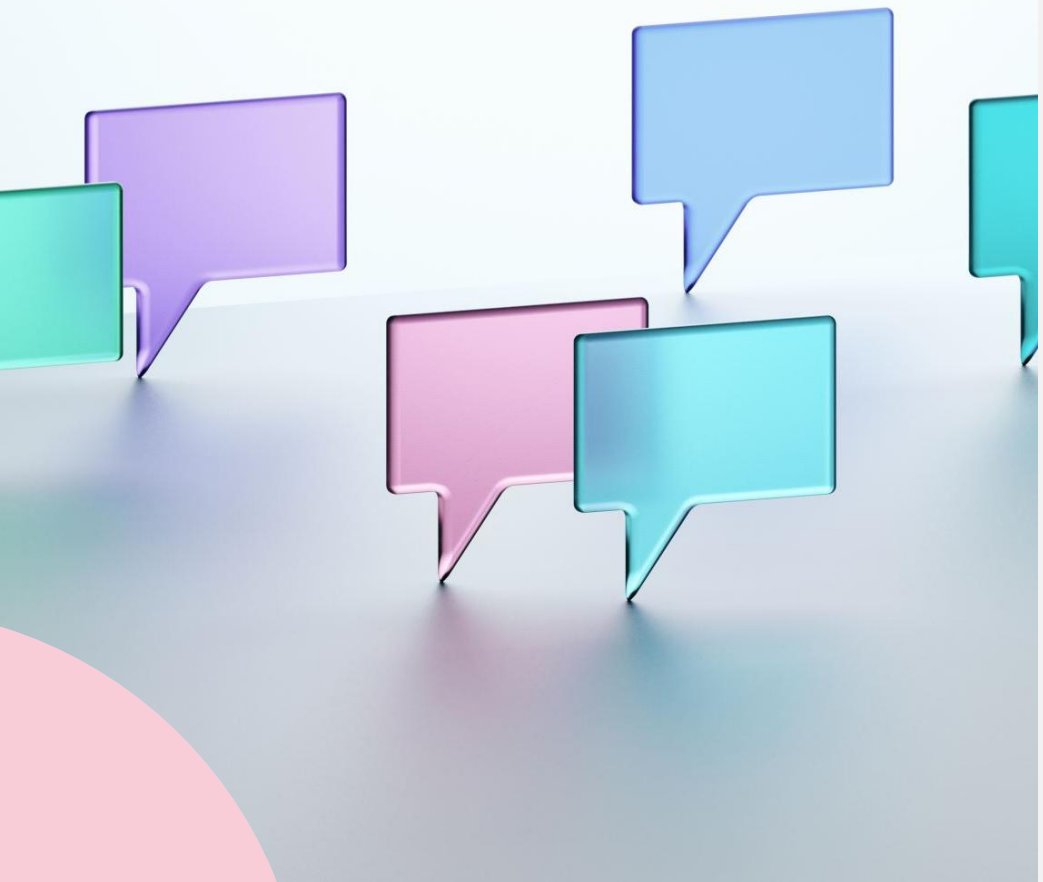


1. ÁMBITO DE UNA VARIABLE

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.
    1. Variable local.
    2. Atributo.
    3. Parámetro de un método.
    4. Parámetro de un tratador de excepciones.
- **Por ahora utilizaremos solo variables locales**, las demas categorias las veremos en posteriores unidades.



2. VARIABLES LOCALES

VARIABLES LOCALES

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

1. Se puede declarar en cualquier lugar del cuerpo, incluso después de instrucciones ejecutables, aunque es una buena costumbre declararlas justo al principio.

2. También pueden declararse variables dentro de un bloque con llaves {...}. En ese caso, sólo serán "visibles" dentro de dicho bloque.

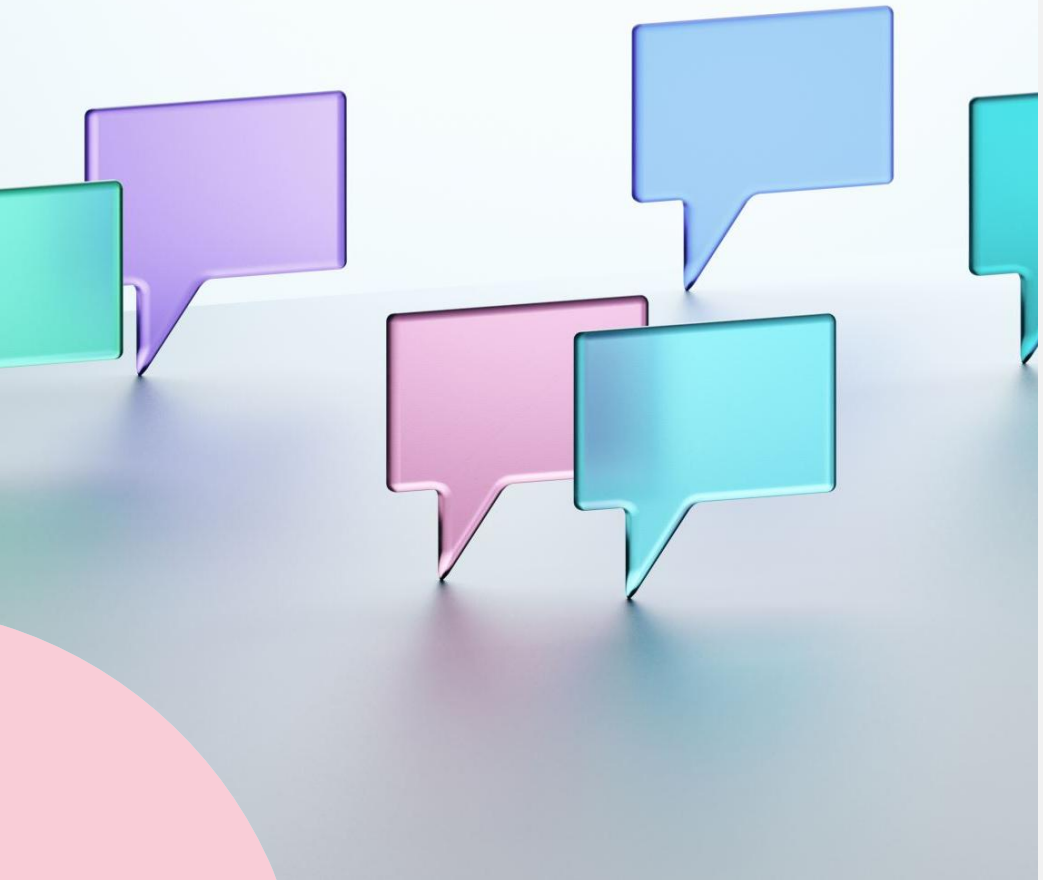
3. Por ejemplo: existe una variable local: **int i**; únicamente puede utilizarse dentro del bloque **main** donde fue creada.

```
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      }
```

VARIABLES LOCALES

4. Segundo ejemplo: existen dos variables locales: int i y j; únicamente pueden utilizarse dentro del bloque con llaves donde fue creada (en este caso dentro de un bucle for).

```
16      public static void main(String[] args) {
17
18          for (int i = 0; i < 10; i++) {
19              int j = i;
20              System.out.println( x: j);
21          }
22
23      }
```



3. CONSTANTES (FINAL)

CONSTANTES (FINAL)

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;

  - Y también 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.***

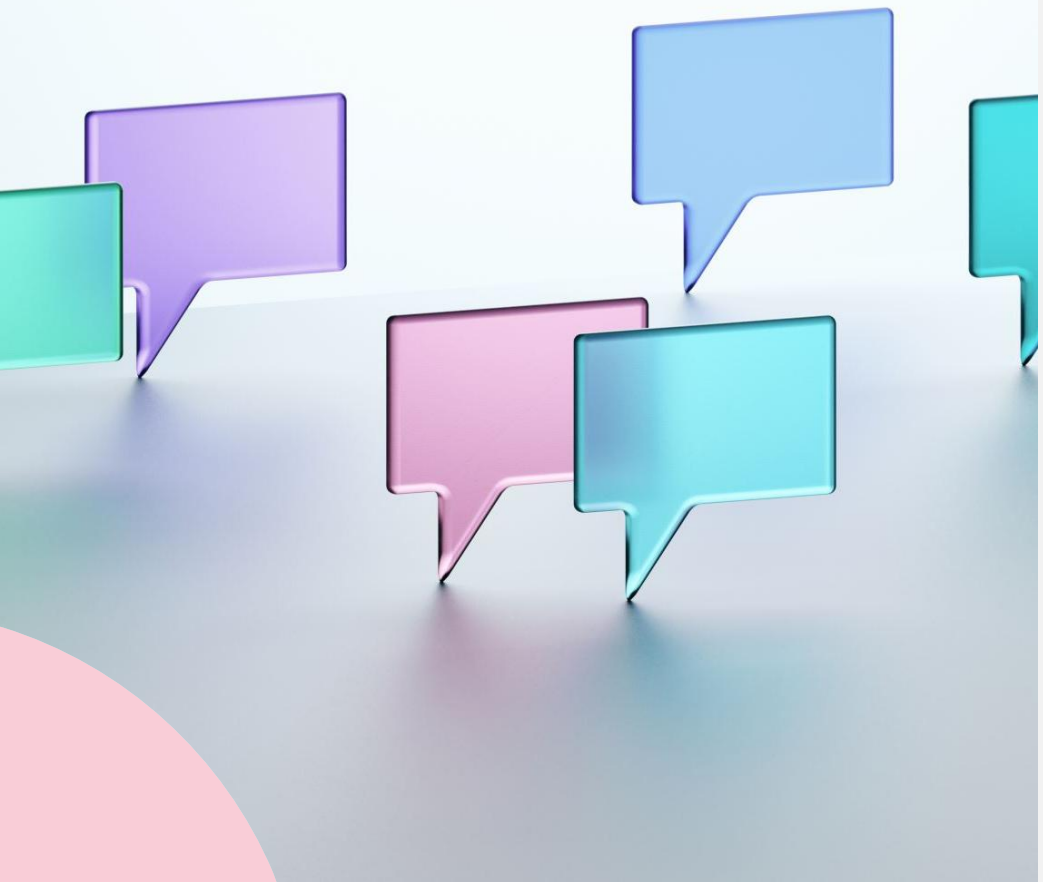
OPERADORES

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:

1. **Aritméticos**: sumas, restas, etc.
2. **Relacionales**: menor, menor o igual, mayor, mayor o igual, etc.
3. **Lógicos**: and, or, not, etc.
4. **Bits**: prácticamente no los utilizaremos en este curso.
5. **Asignación**: =

OJO: no confundir el "=" de la asignación con el "==" de la comparación



1. OPERADORES ARITMÉTICOS

OPERADORES 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

OPERADORES ARITMÉTICOS

- El **operador -** puede utilizarse en su versión unaria (- op1) y la operación que realiza es la de invertir el signo del operando.

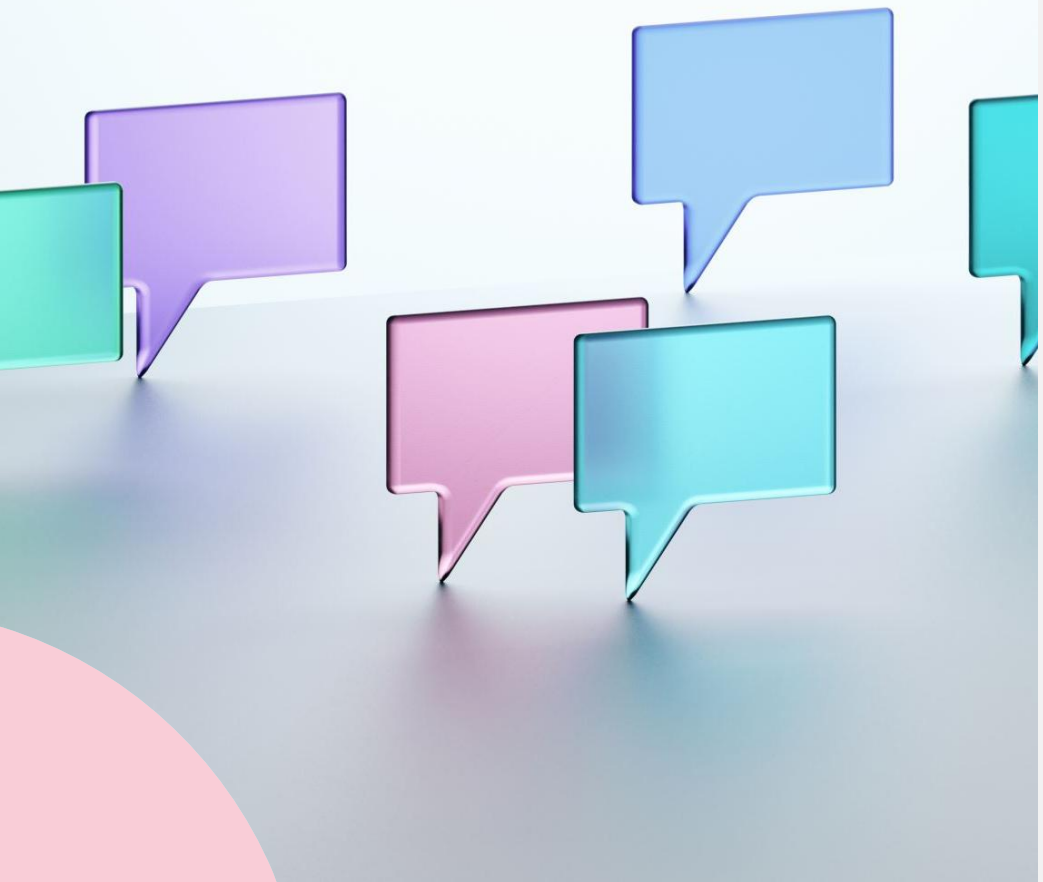
- Los **operadores ++ y --** realizan un incremento y un decremento unitario respectivamente.
  - Es decir:

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

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

OPERADORES ARITMÉTICOS

- Los operadores **++** y **--** admiten notación postfija y prefija:
  - **op1++**: Primero se ejecuta la instrucción en la cual esta inmerso y después se incrementa op1.
  - **op1--**: Primero se ejecuta la instrucción en la cual esta inmerso y después se decrementa op1.
  - **++op1**: Primero se incrementa op1 y después ejecuta la instrucción en la cual esta inmerso.
  - **--op1**: Primero se decrementa op1 y después ejecuta la instrucción en la cual esta inmerso.
- Los operadores incrementales suelen utilizarse a menudo en los bucles (estructuras repetitivas). Lo veremos mas adelante.



2. OPERADORES RELACIONALES

OPERADORES RELACIONALES

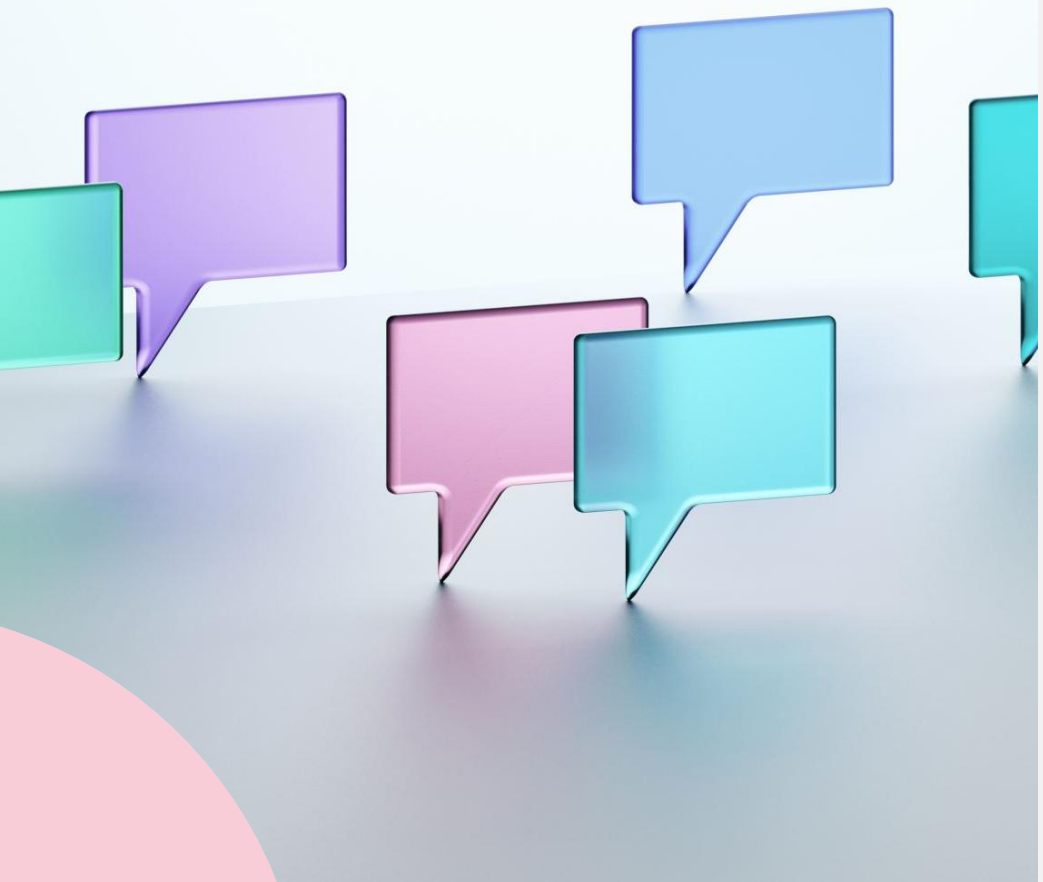
Operador	Formato	Descripción
>	op1 > op2	Devuelve true (cierto) si op1 es mayor que op2
<	op1 < op2	Devuelve true (cierto) si op1 es menor que op2
>=	op1 >= op2	Devuelve true (cierto) si op1 es mayor o igual que op2
<=	op1<= op2	Devuelve true (cierto) si op1 es menor o igual que op2
==	op1 == op2	Devuelve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

OPERADORES RELACIONALES

- Los operadores relacionales actuan 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      }
```

```
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)
```



3. OPERADORES LÓGICOS

OPERADORES 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.

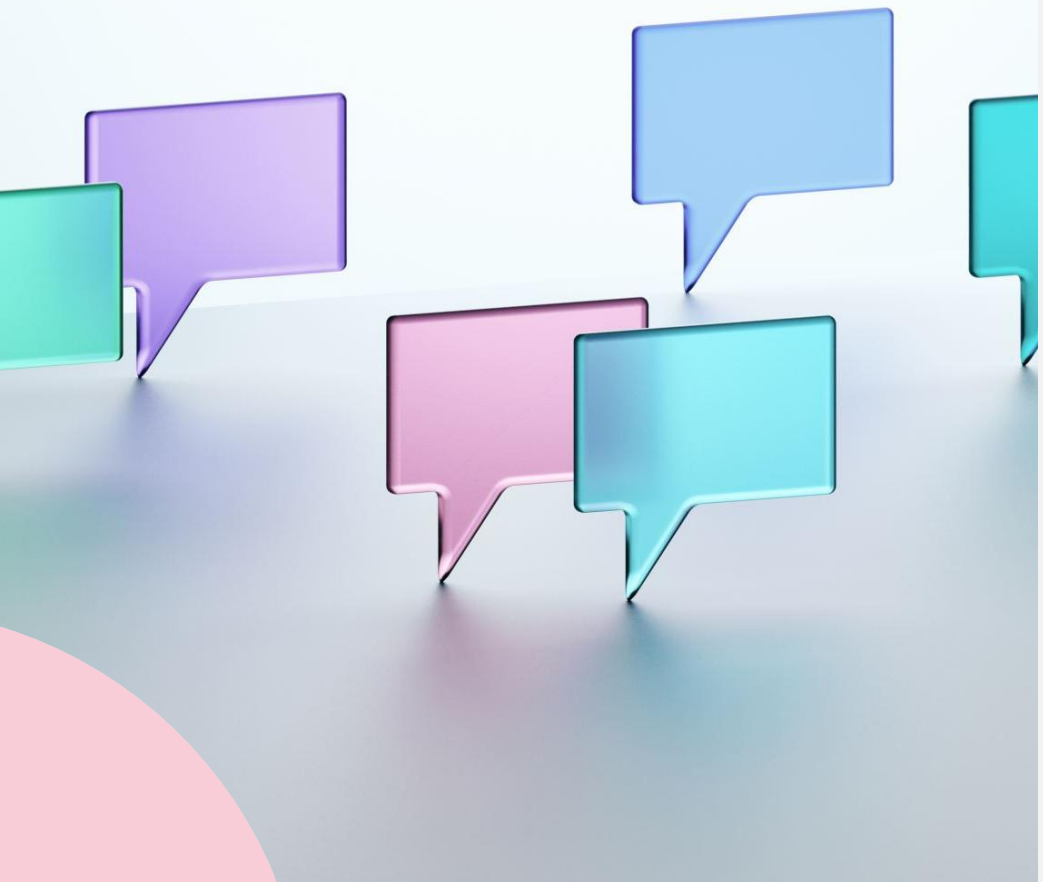
OPERADORES LÓGICOS

1. Estos operadores actuan sobre operadores o expresiones logicas, es decir, aquellos que se evaluan a cierto o falso (true / false).

2. 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  }
```

```
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)
```



4. OPERADORES DE ASIGNACIÓN

OPERADORES DE ASIGNACIÓN

- El operador de asignación es el símbolo igual: **=**

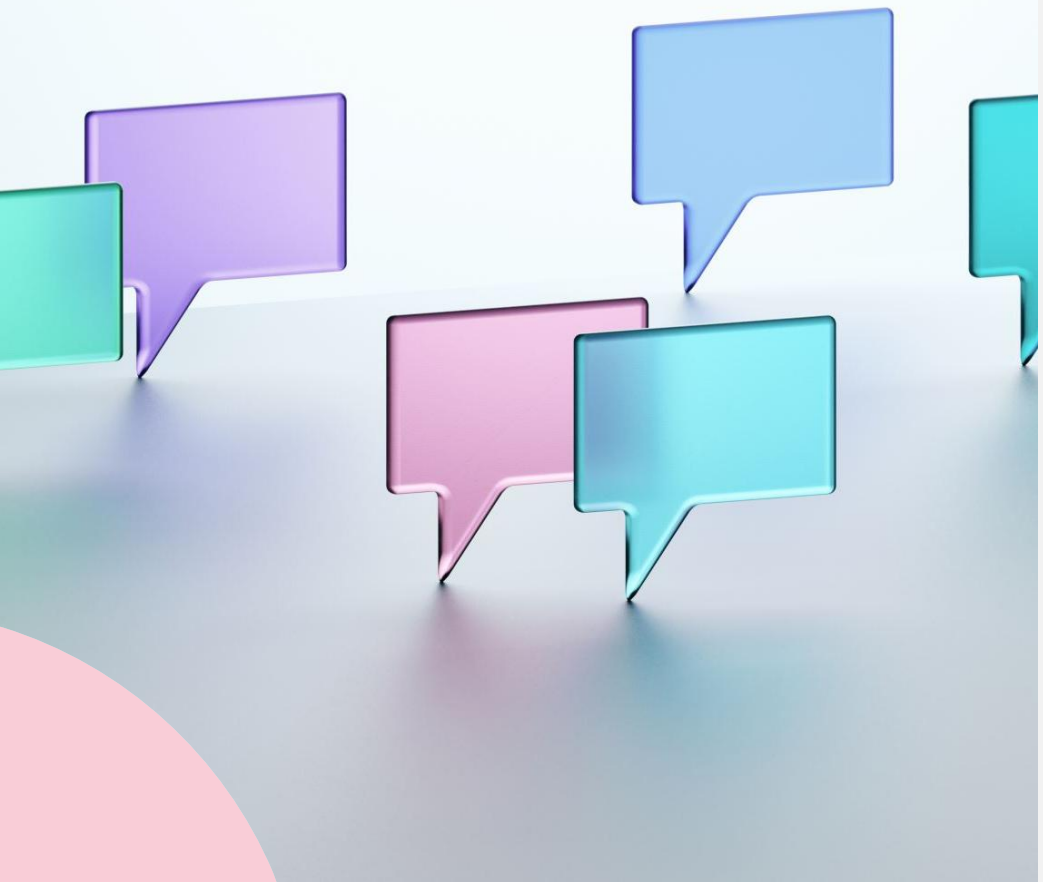
variable = expresión

- Asigna a la variable el resultado de evaluar la expresión de la derecha.

OPERADORES DE ASIGNACIÓN

- Es posible combinar el operador de asignación con otros operadores para, de forma abreviada, realizar un calculo 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



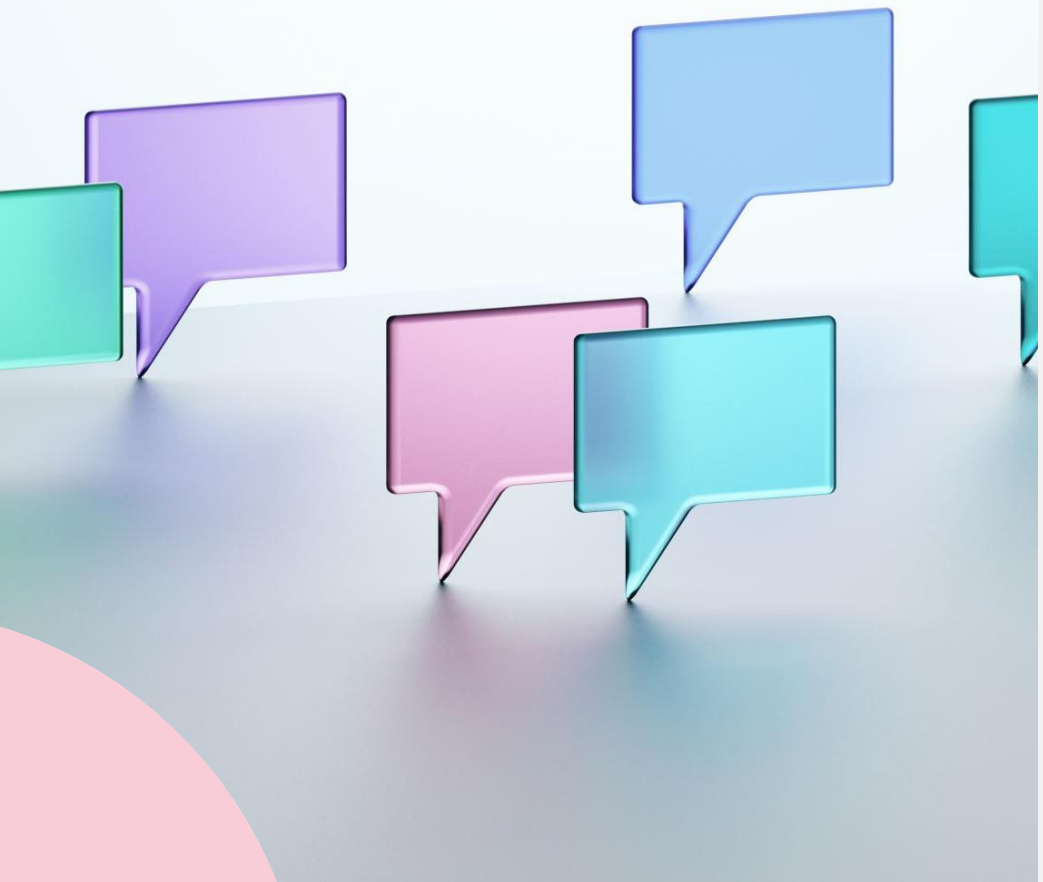
5. EXPRESIONES

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) <= 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.



6. PRECEDENCIA DE OPERADORES

PRECEDENCIA DE OPERACIONES

- Indica el **orden en el que se evalúan los operadores** en una expresion. No es necesario saberse toda la lista de memoria, pero **es importante conocer al menos los más utilizados**: matematicos, relacionales, logicos y de asignacion.
- 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. Creacion o conversion 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: &

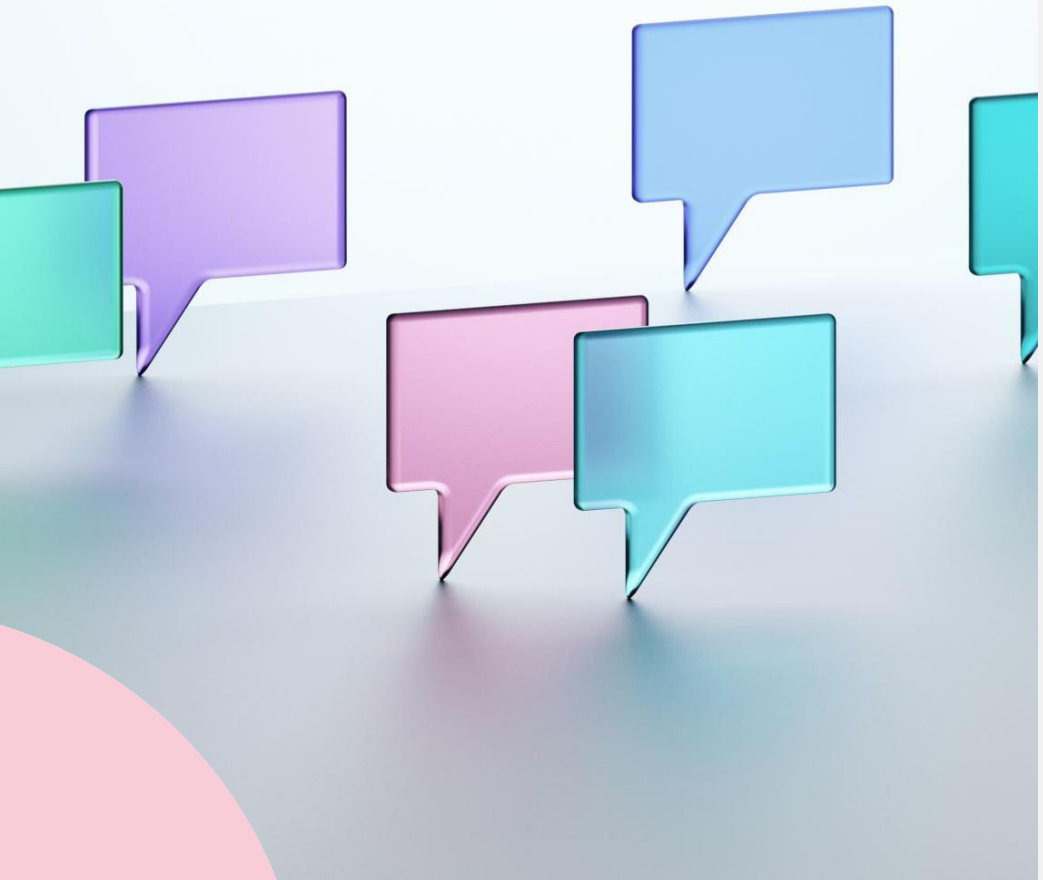
10. AND lógico: &&

11. XOR a nivel de bits: ^

12. OR a nivel de bits: |

13. OR lógico: || 14. Operador condicional: *? :*

15. **Asignación: =, +=, -=, *=, /=, %=, ^=, &=, |=, >>=, <<=**



7. LA CLASE *MATH*

LA CLASE MATH

- Se echan de menos operadores matemáticos mas potentes en Java. Por ello se ha incluido una clase especial llamada **Math** dentro del paquete java.lang.
- Esta clase posee muchos métodos muy interesantes para realizar cálculos matemáticos complejos como calculo 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); // Raíz cuadrada de 9
```

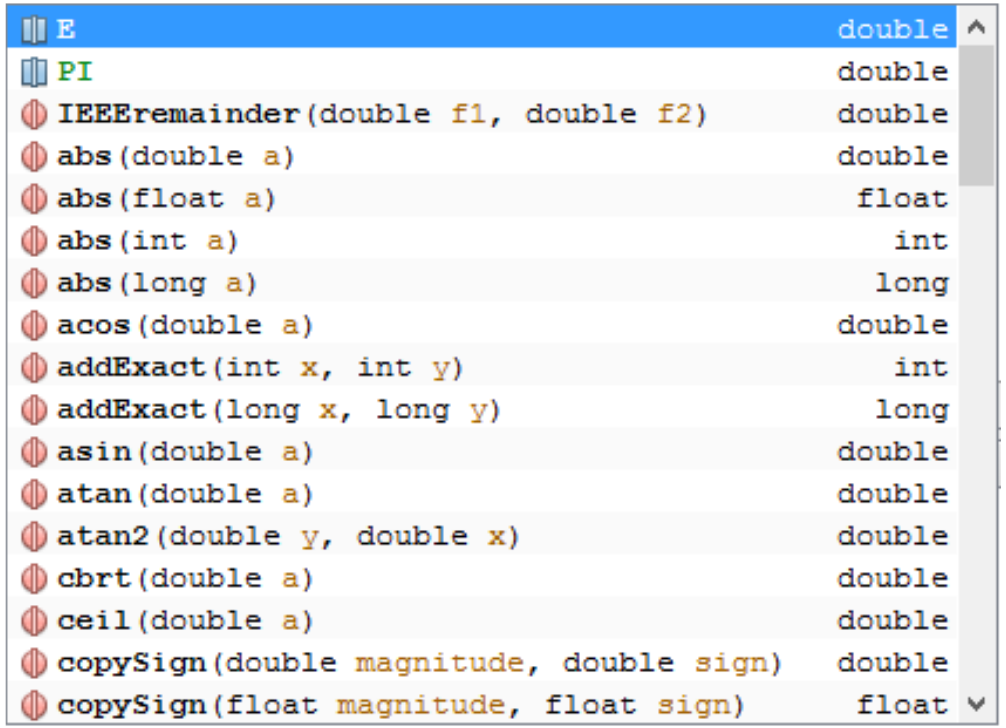
  - También posee constantes como:

double PI -> El numero Π (3,14159265…)

double E -> El numero e (2, 7182818245...)

LA CLASE MATH

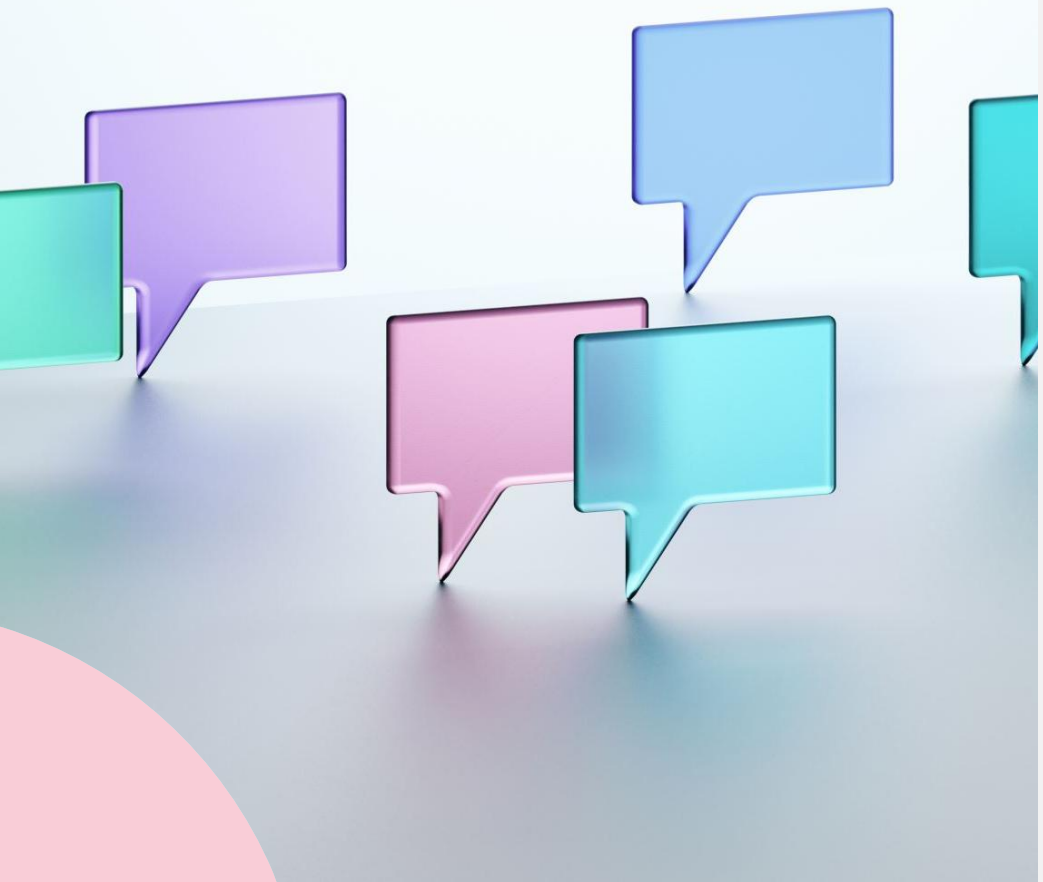
- Algunos ejemplos de otros métodos:



LITERALES

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).

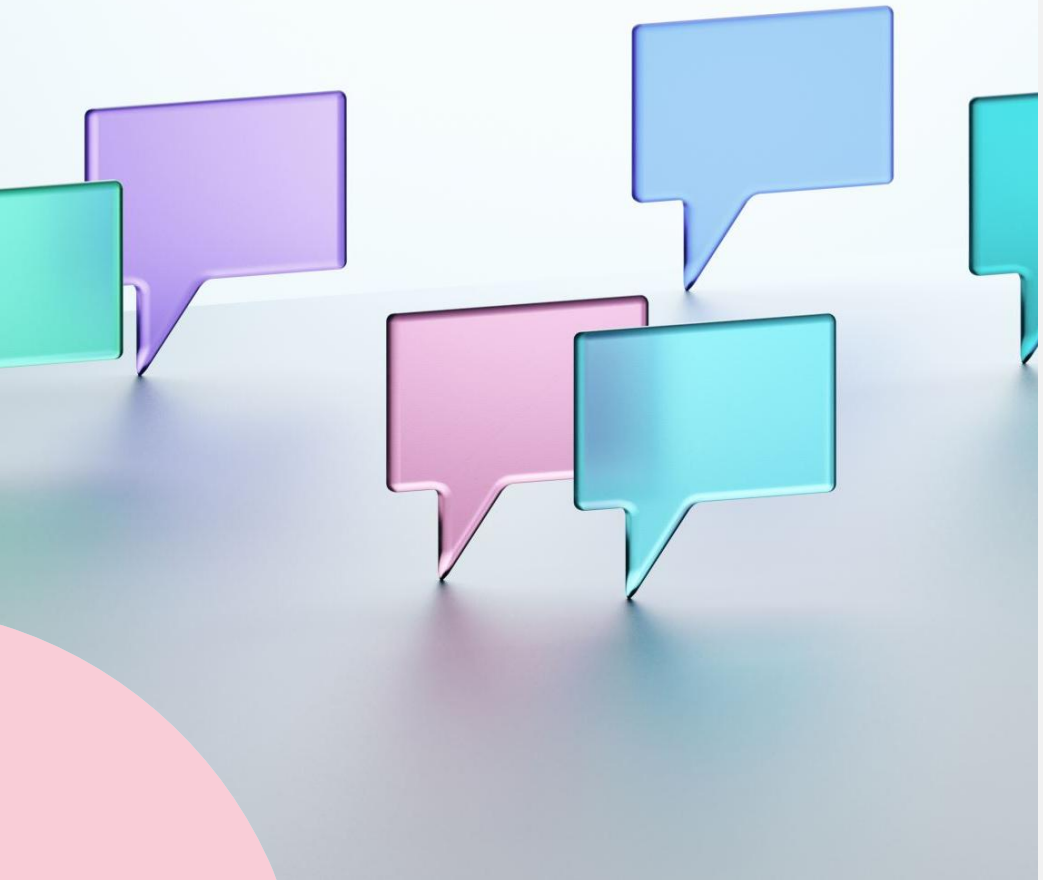


1. LITERALES LÓGICOS

LITERALES LÓGICOS

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

boolean activado = false;



2. LITERALES ENTEROS

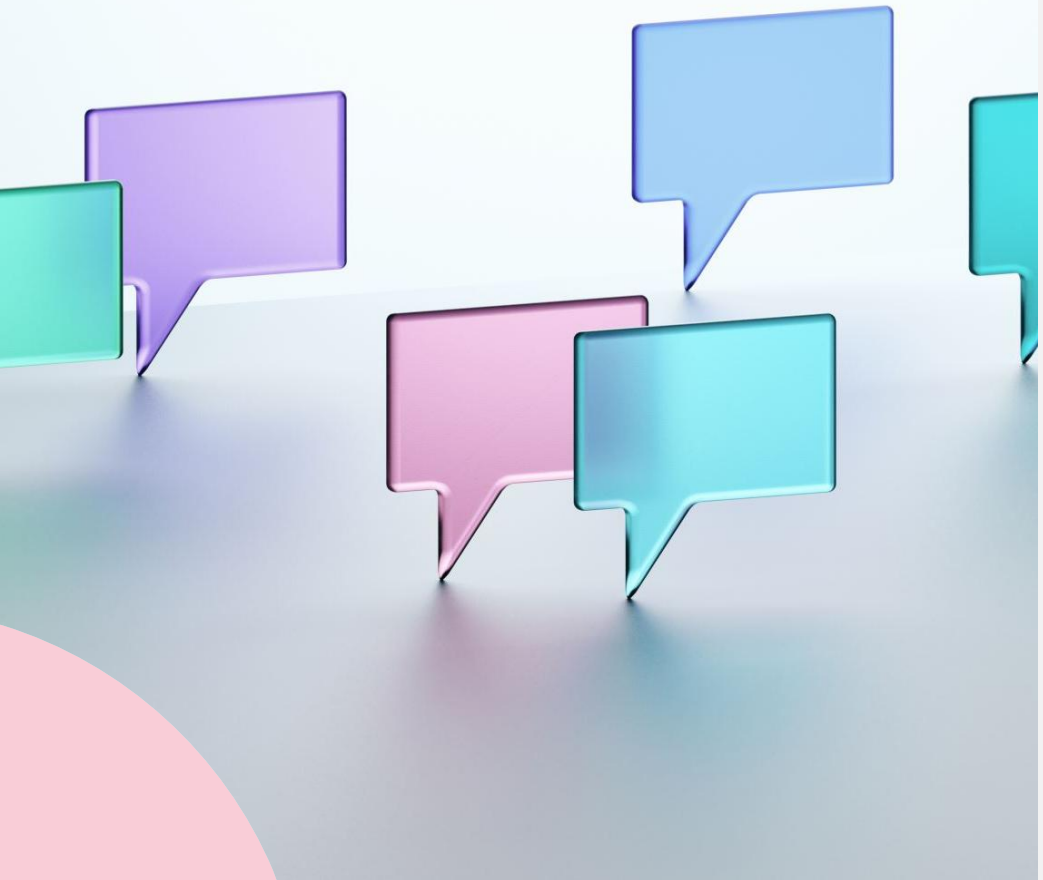
LITERALES ENTEROS

- Los literales de tipo entero: *byte, short, int* y *long* pueden expresarse en decimal (base 10).
- Además, puede añadirse al final de este 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 digito es cualquier símbolo decimal excepto el cero (del 1 al 9). A continuación pueden aparecer dígitos del 0 al 9.

LITERALES ENTEROS

- 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 digito uno (1) en los listados.
- Ejemplo:

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



3. LITERALES REALES

LITERALES REALES

- Los literales de tipo real sirven para indicar valores *float* o *double*.
- Existen dos formatos de representación:
  - mediante su parte entera, el punto decimal (.) y la parte fraccionaria;
  - o mediante notación exponencial o científica.
- Ejemplos equivalentes:

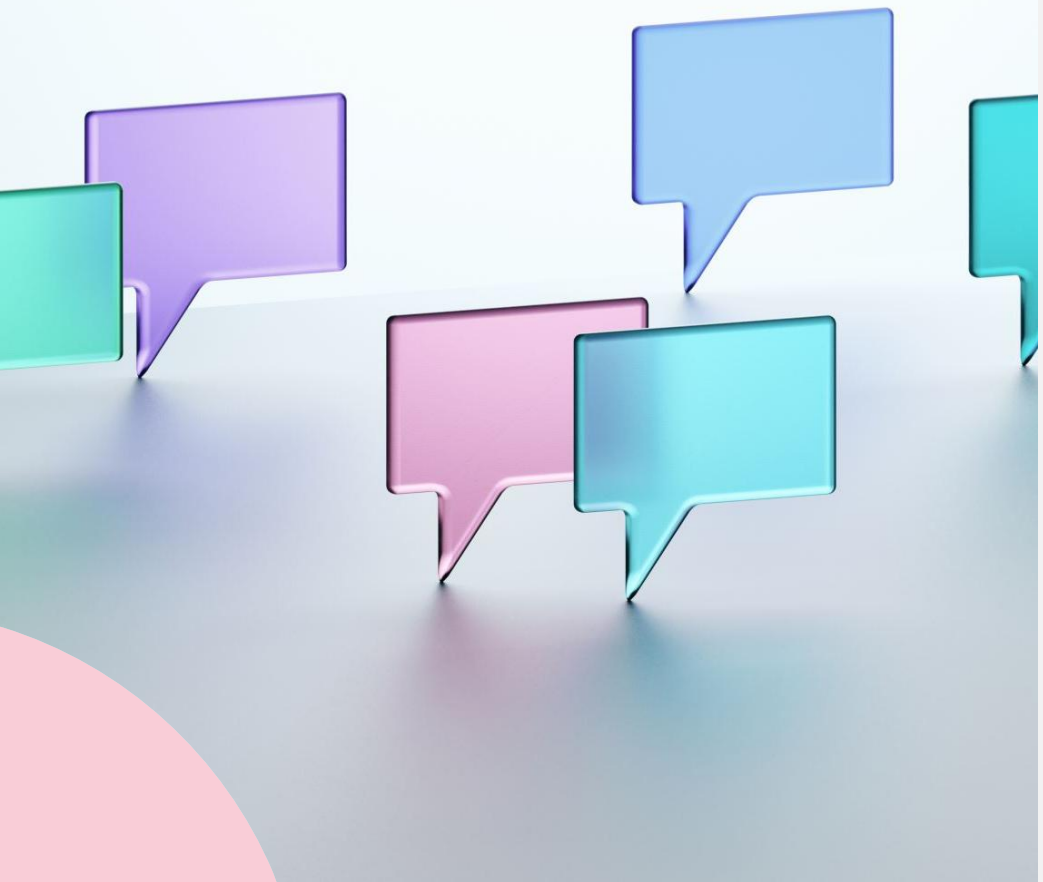
```
3.1415

0.31415e1

.31415e1

0.031415E+2

.031415e2

314.15e-2

31415E-4
```

LITERALES REALES

- Al igual que los literales que representan enteros, se puede poner una letra como sufijo.
- Esta letra puede ser una *F* o una *D* (mayuscula o minuscula indistintamente).
  - *F* --> Trata el literal como de tipo *float*.
  - *D* --> Trata el literal como de tipo *double*.
- Ejemplo:

3.1415F

.031415d



4. LITERALES CARÁCTER

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.

LITERALES CARÁCTER

- Las posibles secuencias de escape son:

  \n -----> Nueva Linea.

  \t -----> Tabulador.

  \r -----> Retroceso de Carro.

  \f -----> Comienzo de Pagina.

  \b -----> Borrado a la Izquierda.

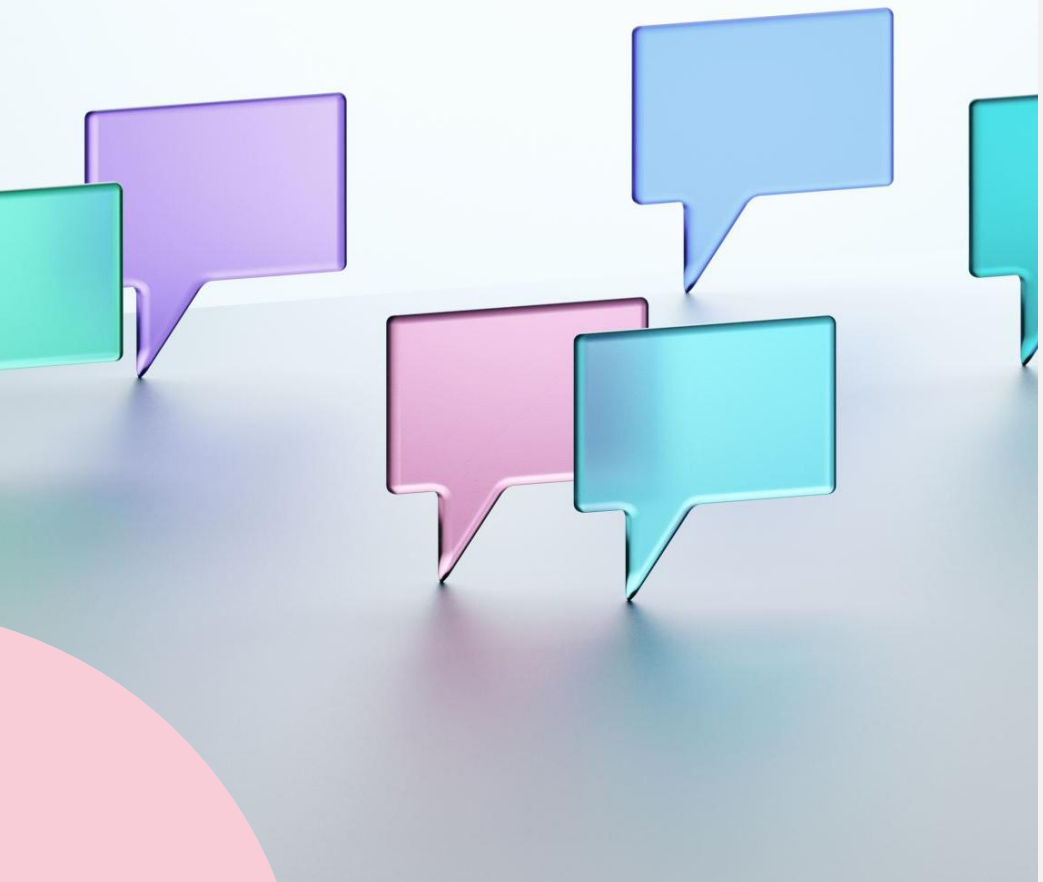
  \\ -----> El caracter barra inversa (\ ).

  \' -----> El caracter comilla simple (').

  \" -----> El caracter comilla doble o bi-prima (").

- Por ejemplo:

  1. Para imprimir una diagonal inversa se utiliza: \\
  2. Para imprimir comillas dobles en un String se utiliza: \"



5. LITERALES CADENAS

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).

LITERALES CADENAS

- 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");

LITERALES CADENAS

- La visualizacion del *string* anterior mediante *println()* produciria la siguiente salida por pantalla:

  Primera linea

  Segunda linea del string

  Hola

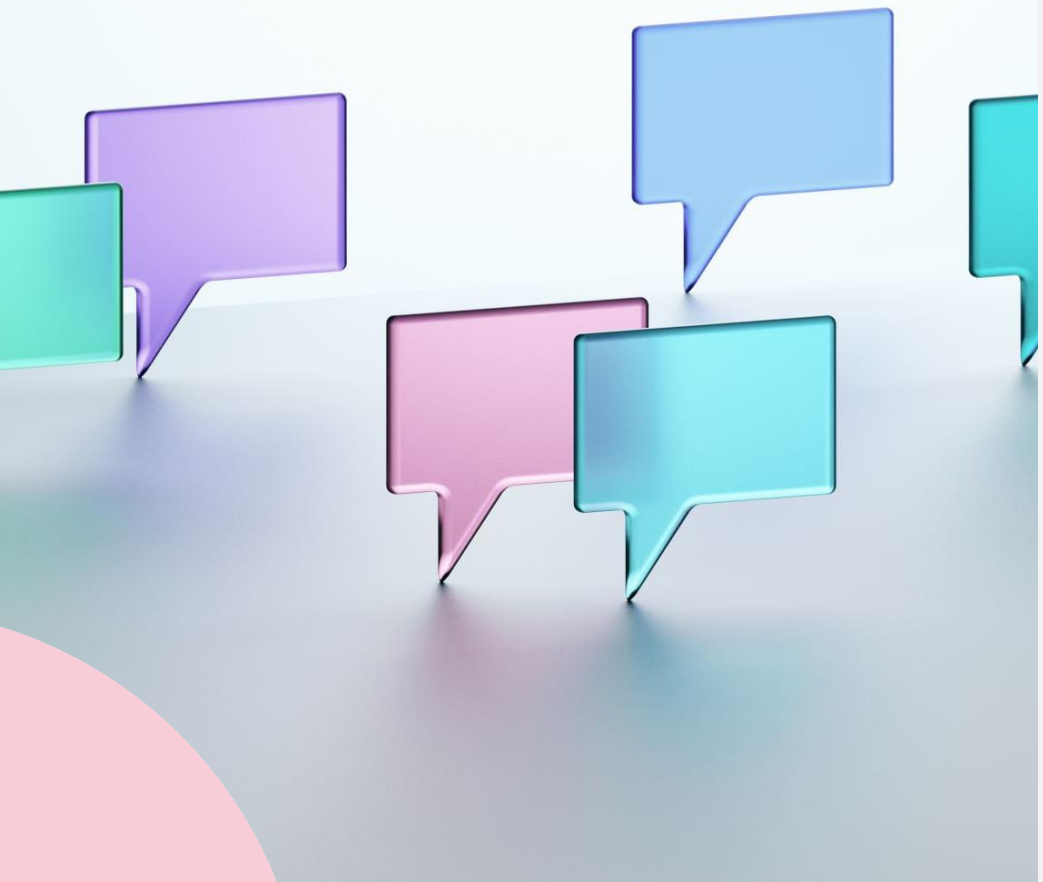
- La forma de incluir los caracteres: comillas dobles (“) y contrabarra (\ ) es mediante las secuencias de escape \” y \\ respectivamente (o mediante su codigo Unicode precedido de \ ).

LITERALES CADENAS

1. Si el string es demasiado largo y debe dividirse en varias líneas en el fichero fuente
2. Para ello, puede utilizarse el operador de concatenación de strings (+) de la siguiente forma:

```
"Este String es demasiado largo para estar en una linea del " +
"fichero fuente y se ha dividido en dos."
```

SALIDA Y ENTRADA ESTÁNDAR



1. SALIDA ESTÁNDAR

SALIDA ESTÁNDAR

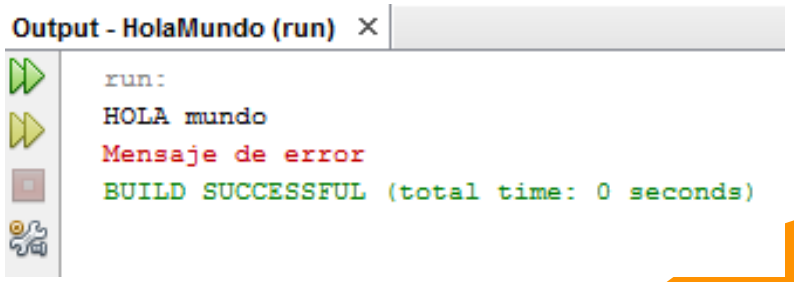
- Ya hemos visto el uso de ***System.out*** para mostrar información por pantalla:
  - ***print("...")*** imprime texto por pantalla (Similar a Escribir Sin Saltar de PSeINT)
  - ***println("...")*** imprime texto por pantalla e introduce un salto de linea.
- La utilización de ***System.err*** seria 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).

SALIDA ESTÁNDAR

- 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 seria la siguiente:



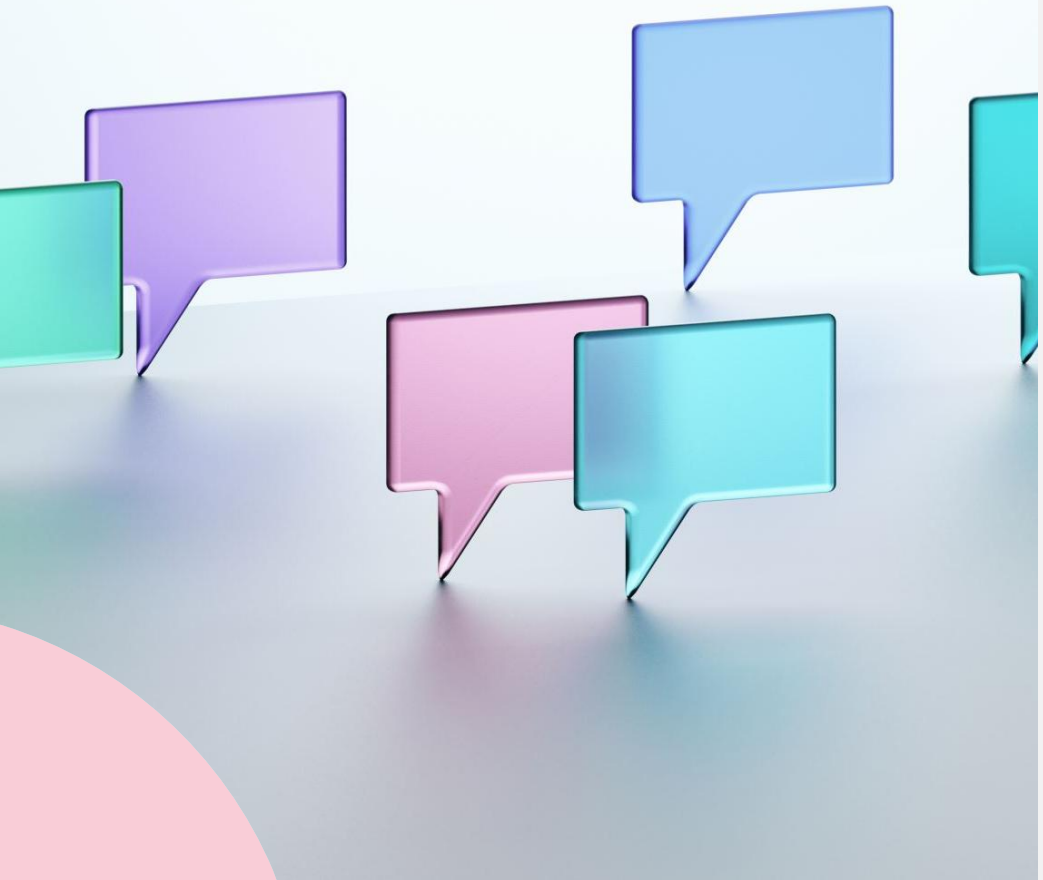
SALIDA ESTÁNDAR

1. Tambien pueden imprimirse variables de cualquier tipo, asi 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.
```



2. ENTRADA ESTÁNDAR

ENTRADA ESTÁNDAR

- La entrada estándar (leer información del teclado, escrita por el usuario) es un poco mas compleja.
- Hay varias formas de hacerlo, pero la mas 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.

ENTRADA ESTÁNDAR

- 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 guardara en **texto** (de tipo String).
    - Similar a ***Leer*** en PSeINT.

ENTRADA ESTÁNDAR

- Existen mucho otros métodos según el tipo de dato que se quiera leer:
  1. **nextByte()**: obtiene un numero entero tipo byte.
  2. **nextShort()**: obtiene un numero entero tipo short.
  3. **nextInt()**: obtiene un numero entero tipo int.
  4. **nextLong()**: obtiene un numero entero tipo long.
  5. **nextFloat()**: obtiene un numero real float.
  6. **nextDouble()**: obtiene un numero real double.
  7. **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 caracter.

ENTRADA ESTÁNDAR

- **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  }
```

```
Output - HolaMundo (run) #2
run:
Introduce tu nombre: Pepito
Hola Pepito
BUILD SUCCESSFUL (total time: 3 seconds)
```

ENTRADA ESTÁNDAR

- Ejemplo en el que leemos un valor tipo double.
- El programa pide al usuario que introduzca el radio de un circulo, luego calcula su área y circunferencia y, 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 portencia(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  }
```

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

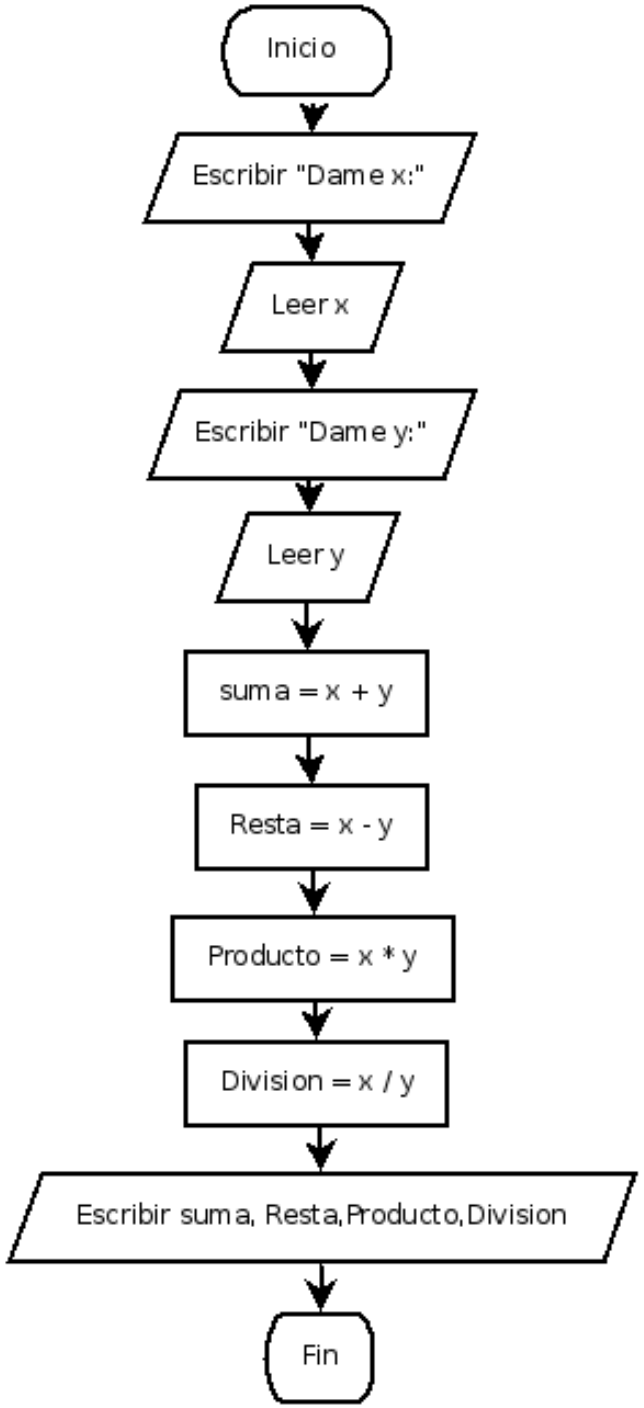
EJEMPLO COMPLETO

EJEMPLO COMPLETO

1. **Enunciado**: Programa que lea dos números, calcule y muestre el valor de su suma, resta, producto y división.

EJEMPLO COMPLETO

- **Diagrama de flujo**:



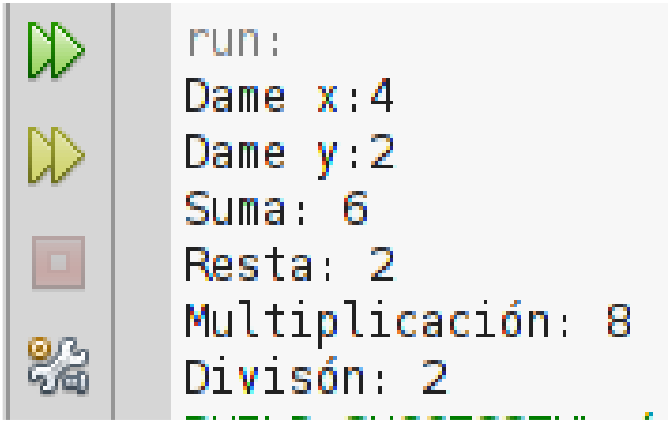
EJEMPLO COMPLETO

- **Código**:

```java
package ejemplo1;

import java.util.Scanner; // Importames la clase Scanner

public class Ejemplo1 {

    public static void main(String[] args) {

        // Declaramos las variables que vamos a necesitar
        int x, y, suma, resta, mult, div;

        // Creamos el objeto Scanner para leer por teclado
        Scanner reader = new Scanner(System.in);

        // Pedimos y leemos x
        System.out.print("Dame x:");
        x = reader.nextInt();

        // Pedimos y leemos y
        System.out.print("Dame y:");
        y = reader.nextInt();

        // Realizamos los cálculos necesarios
        suma  = x + y;
        resta = x - y;
        mult  = x * y;
        div   = x / y;

        // Mostramos los cálculos por pantalla
        System.out.println("Suma: " + suma);
        System.out.println("Resta: " + resta);
        System.out.println("Multiplicación: " + mult);
        System.out.println("Divisón: " + div);
    }
}
```

EJEMPLO COMPLETO

- **Salida**:



BIBLIOGRAFÍA

BIBLIOGRAFÍA

Apuntes actualizados y adaptados a partir de la siguiente documentación:

1. [1] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.

2. [2] Apuntes Programación de Javier Valero Lionel Tarazón. Ceedcv.

¿DUDAS?

FIN

Javier Rojo
fjrojom001@educarex.es