

# DESARROLLO DE APLICACIONES MULTIPLATAFORMA PROGRAMACIÓN

## TEMA 2

### ELEMENTOS BÁSICOS DE UN PROGRAMA

(Lenguaje de Programación Java)

# CONTENIDOS

2.1. Programa y lenguajes de programación.

2.1.1. El lenguaje Java.

2.1.2. El JDK.

2.1.3. Los programas en Java.

2.2. Estructura y bloques de un programa.

2.3. Entornos integrados de desarrollo.

2.4. Tipos de datos simples.

2.5. Constantes y literales.

2.6. Variables.

2.7. Operadores y expresiones.

2.8. Conversiones de tipos.

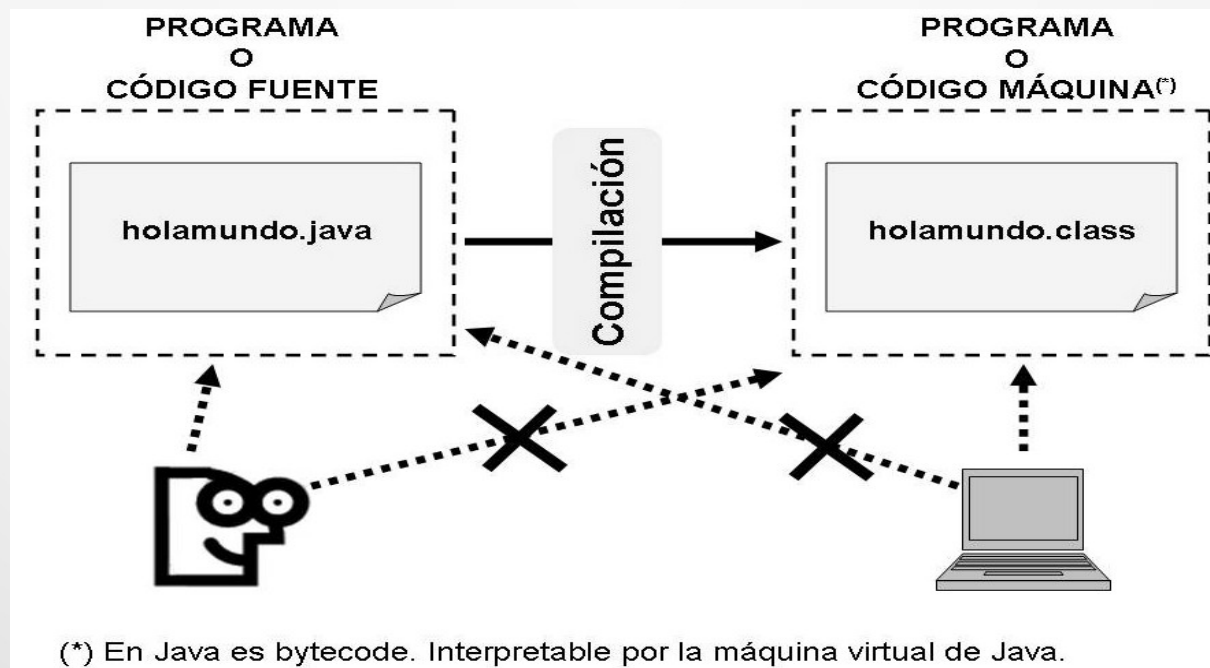
2.9. Programación de consola.

2.10. Salida con formato.

## 2.1. PROGRAMA Y LENGUAJES DE PROGRAMACIÓN.

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

Los programas están codificados en lenguaje binario, nuestro objetivo es aprender un lenguaje de programación para escribir programas de manera entendible por las personas que luego traduciremos al lenguaje máquina entendible por los ordenadores mediante otros programas llamados intérpretes o compiladores.



## 2.1. PROGRAMA Y LENGUAJES DE PROGRAMACIÓN.

Los compiladores son programas específicos para un lenguaje de programación, los cuales transforman el programa fuente en un programa directa o indirectamente ejecutable por la máquina destino.

El lenguaje máquina que genera Java es un lenguaje intermedio (*bytecode*) interpretable por una máquina virtual instalada en el ordenador donde se va a ejecutar.

Una máquina virtual es una máquina ficticia que traduce las instrucciones máquina ficticias en instrucciones para la máquina real. La ventaja de usar una máquina virtual es que los programas se pueden ejecutar en cualquier tipo de hardware siempre y cuando tenga instalada la máquina virtual correspondiente.

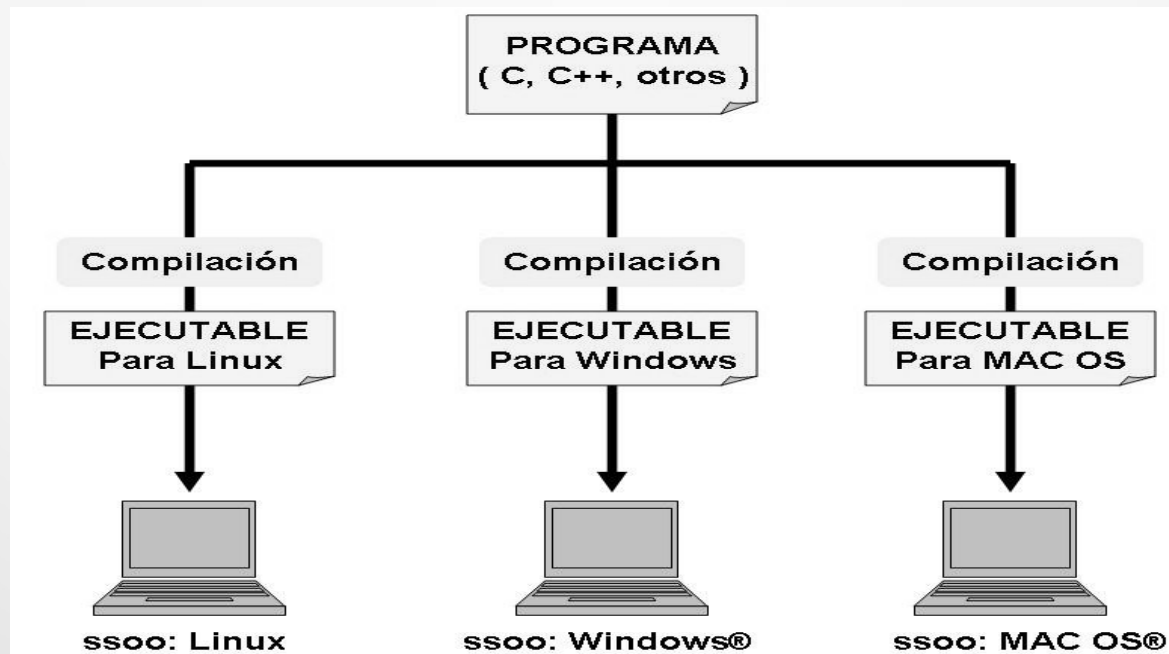
*A diferencia de los compiladores, los intérpretes leen línea a línea el código fuente y lo ejecutan. Este proceso es muy lento y requiere tener cargado en memoria el intérprete. La ventaja es que la depuración y corrección de errores del programa es mucho más sencilla que con los compiladores.*

## 2.1. PROGRAMA Y LENGUAJES DE PROGRAMACIÓN.

### 2.1.1. EL LENGUAJE JAVA.

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

Java es un lenguaje multiplataforma, utiliza una máquina virtual en el sistema destino, por lo que no hace falta recompilar de nuevo las aplicaciones para cada sistema operativo.

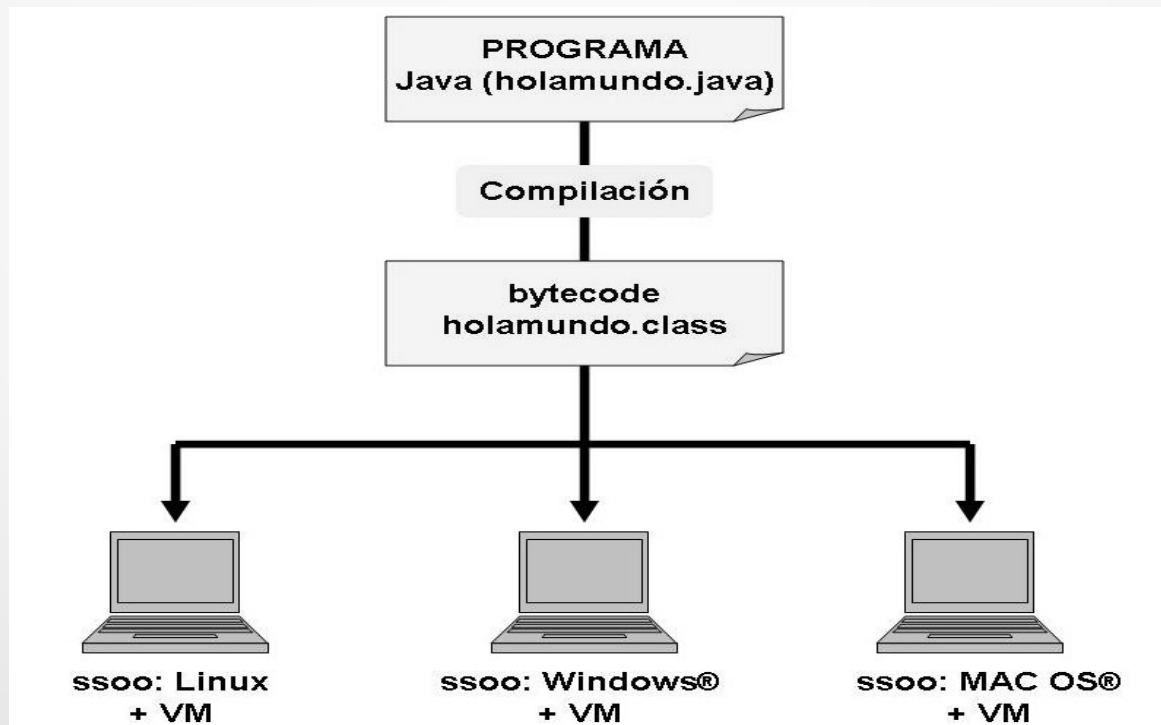


## 2.1. PROGRAMA Y LENGUAJES DE PROGRAMACIÓN.

### 2.1.1. EL LENGUAJE JAVA.

Java es un lenguaje interpretado, utiliza un código intermedio (*bytecode*) independiente de la arquitectura que puede ser ejecutado en cualquier sistema.

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



## 2.1. PROGRAMA Y LENGUAJES DE PROGRAMACIÓN.

### 2.1.2. EL JDK.

El JDK (*Java Development Kit*), aunque no contiene ninguna herramienta gráfica para el desarrollo de programas, sí que contiene aplicaciones de consola y herramientas de compilación, documentación y depuración.

El JDK incluye el JRE (*Java Runtime Environment*) que consta de los mínimos componentes necesarios para ejecutar una aplicación Java, como son la máquina virtual y las librerías de clases.

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

*java*. Es la máquina virtual de Java.

*javac*. Es el compilador de Java. Permite compilar las clases desarrolladas.

*javap*. Es un desensamblador de clases.

*jdb*. Es el depurador de consola de Java.

*javadoc*. Es un generador de documentación.

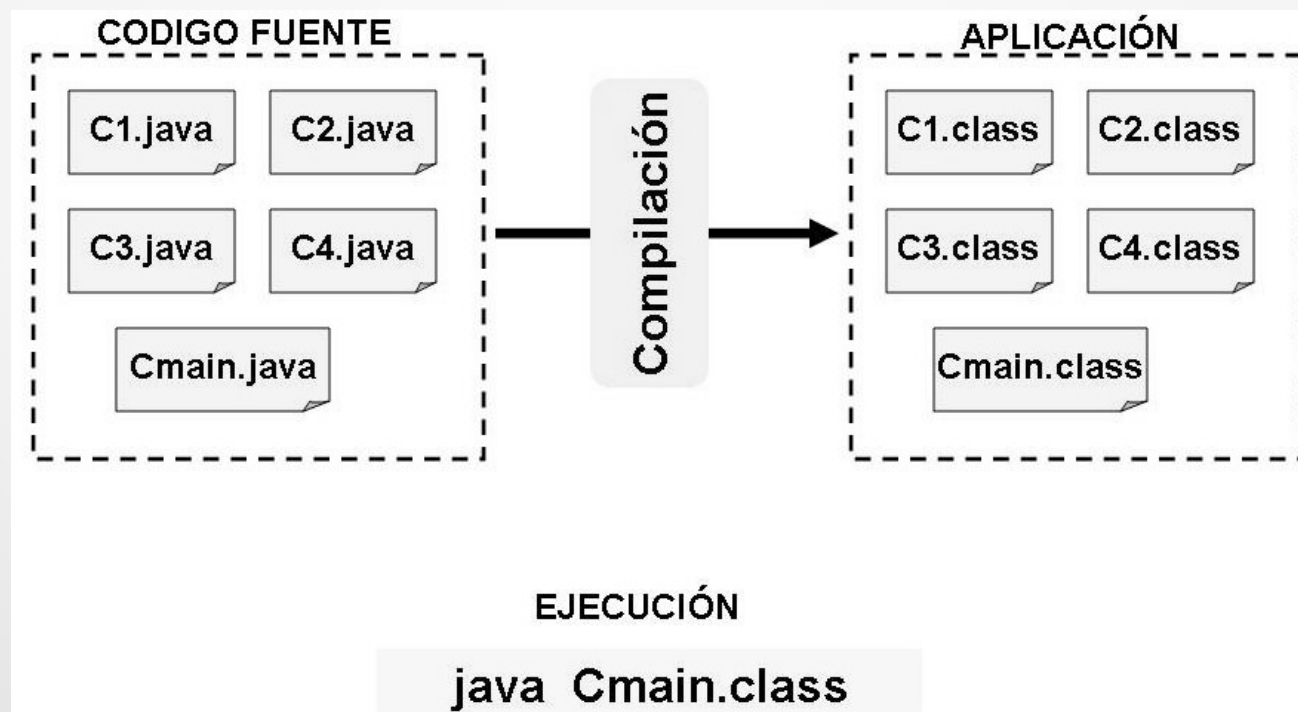
*appletviewer*. Visor de *Applets*.

## 2.1. PROGRAMA Y LENGUAJES DE PROGRAMACIÓN.

### 2.1.3. LOS PROGRAMAS EN JAVA.

Los programas en Java se componen de una serie de ficheros *.class* que son ficheros en *bytecode* que contienen las clases del programa. Estos ficheros no tienen por qué estar situados en un único directorio concreto.

La aplicación se ejecuta desde el método principal o *main()* situado en una clase. A partir de aquí se van creando objetos de las clases y se va ejecutando la aplicación.





## 2.2. ESTRUCTURA Y BLOQUES DE UN PROGRAMA.

En Java normalmente cada clase es un fichero distinto. Si existieran varias clases en un fichero, la clase cuyo nombre coincide con el nombre del fichero debería llevar el modificador *public* y es la que se puede utilizar desde fuera del fichero.

Las clases tienen el mismo nombre que su fichero *.java* y es importante que mayúsculas y minúsculas coincidan. La clase abarca desde la primera llave que abre hasta la última que cierra (bloque de código).

```
public class Holamundo {  
  
    public static void main(String[] args) {  
  
        /*Muestra un mensaje por pantalla*/  
        System.out.println("Hola mundo!");  
    }  
  
}
```

## 2.2. ESTRUCTURA Y BLOQUES DE UN PROGRAMA.

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

El método *main* tiene las siguientes particularidades:

- Es público (*public*). Esto es así para poder llamarlo desde cualquier lado.

- Es estático (*static*). Se le puede llamar sin tener que instanciar la clase.

- No devuelve ningún valor (modificador *void*).

- Admite una serie de parámetros (*String[] args*).

En el ejemplo anterior, el método *main* abarca todo el código contenido entre llaves ( `{...}` ).

## 2.2. ESTRUCTURA Y BLOQUES DE UN PROGRAMA.

Para sacar información por pantalla en Java se utiliza la clase *System* que puede ser llamada desde cualquier punto del programa, la cual tiene un atributo *out* que a su vez tiene dos métodos muy utilizados: *print()* y *println()*. La diferencia es que el segundo añade un retorno de línea al texto introducido.

Todas las órdenes en Java terminan en punto y coma ( ; ) salvo los cierres de llaves.

Existen dos tipos de comentarios en Java:

// . Estos comentarios comienzan en la doble barra y terminan en el final de la línea.

/\* \*/ . Estos comentarios comienzan con los caracteres /\* y terminan con los caracteres \*/ y se pueden extender por múltiples líneas.

## 2.3. ENTORNOS INTEGRADOS DE DESARROLLO.

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

Lo primero que hay que hacer cuando se instala un IDE es configurar como mínimo la ruta del JDK.

Cuando se instala un IDE hay que asegurarse que las opciones que indican las rutas de las bibliotecas, el JDK y demás recursos son correctas. Si no se hace esto el IDE nunca podrá ejecutar ni compilar programas.

Existen muchos IDEs para trabajar con Java:

Geany.

Eclipse. Desarrollado inicialmente por IBM pero actualmente un IDE de código abierto mantenido por la Fundación Eclipse.

NetBeans. Desarrollado inicialmente por SUN pero actualmente de Oracle. También de código abierto y muy utilizado.

## 2.3. ENTORNOS INTEGRADOS DE DESARROLLO.

¿Es necesario un IDE para compilar y ejecutar Java?

No es necesario compilar desde un IDE nuestro programa.

Si la variable PATH del sistema está correctamente configurada, basta con ejecutar desde línea de comandos y desde el mismo directorio donde se encuentra el fichero *.java* el siguiente comando:

```
javac Holamundo.java
```

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

Una vez tengamos el fichero *.class* hay que ejecutar el programa con el siguiente comando:

```
java Holamundo
```

## 2.4. TIPOS DE DATOS SIMPLES.

Los tipos de datos son necesarios para que el intérprete o compilador conozca de antemano el tipo de información que va a contener una constante, una variable, un parámetro o devolver una función.

Los tipos de datos primitivos en Java son los siguientes:

Tipo de datos	Información representada	Rango	Descripción
byte	Datos enteros	-128 $\longleftrightarrow$ +127	Se utilizan 8 bits (1 byte) para almacenar el dato.
short	Datos enteros	-32768 $\longleftrightarrow$ +32767	Dato de 16 bits de longitud (independientemente de la plataforma).
int	Datos enteros	-2147483648 $\longleftrightarrow$ +2147483647	Dato de 32 bits de longitud (independientemente de la plataforma).
long	Datos enteros	-9223372036854775808 $\longleftrightarrow$ +9223372036854775807	Dato de 64 bits de longitud (independientemente de la plataforma).

## 2.4. TIPOS DE DATOS SIMPLES.

char	Datos enteros y caracteres	0 $\longleftrightarrow$ 65535	Este rango es para representar números en unicode, los ASCII se representan con los valores del 0 al 127. ASCII es un subconjunto del juego de caracteres Unicode.
float	Datos en coma flotante de 32 bits	Precisión aproximada de 7 dígitos	Dato en coma flotante de 32 bits en formato IEEE 754 (1 bit de signo, 8 para el exponente y 24 para la mantisa).
double	Datos en coma flotante de 64 bits	Precisión aproximada de 16 dígitos	Dato en coma flotante de 64 bits en formato IEEE 754 (1 bit de signo, 11 para el exponente y 52 para la mantisa).
boolean	Valores booleanos	true/false	Utilizado para evaluar si el resultado de una expresión booleanas es verdadero (true) o falso(false).

## 2.4. TIPOS DE DATOS SIMPLES.

Ejemplos de utilización de tipos de datos en la declaración de variables:

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



## 2.5. CONSTANTES Y LITERALES.

Las constantes se utilizan en datos que nunca varían y hay que asignarles un valor inicial. Utilizando constantes y no variables nos aseguramos que su valor no va a poder ser modificado nunca.

También, utilizar constantes permite centralizar el valor de un dato en una sola línea de código (para cambiar el valor basta con modificar una línea en vez del literal correspondiente en muchas partes del programa).

Las constantes se declaran siguiendo el siguiente formato:

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

“final” identifica que es una constante.

“static”, que es opcional, indica que solo existirá una copia de dicha constante en el programa aunque se declare varias veces.

“tipo de datos” de la constante.

“nombre” de la constante.

“valor” que toma la constante.

## 2.5. CONSTANTES Y LITERALES.

Un literal puede ser una expresión:

De tipo de dato simple:

Número, con o sin decimales.

Carácter, encerrado entre comillas simples ( ' ' ).

*Booleano*, valores de “true” o “false”.

El valor “null”.

Un *string* o cadena de caracteres, encerrado entre comillas dobles, no simples como los caracteres, ( “ ” ).

Ejemplos: 12, 3.1415, ‘a’, “hola”, true.

*El nombre de las constantes se declaran en mayúsculas mientras que las variables se hacen en minúscula, esto se realiza como norma de estilo.*

## 2.6. VARIABLES.

Una variable es una zona de memoria donde se puede almacenar información del tipo de dato que desee el programador.

Las variables se declaran dentro de un bloque, que es el contenido encerrado entre llaves { ... }, y son accesibles solo dentro de ese bloque. Se crean cuando el bloque se declara y se destruyen cuando finaliza la ejecución de dicho bloque.

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

## 2.6. VARIABLES.

La visibilidad, scope o ámbito de una variable es la parte del código de una aplicación donde la variable es accesible y puede ser utilizada.

Por regla general, en Java, todas las variables que están dentro de un bloque son visibles y existen dentro de dicho bloque.

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

*Las palabras clave o reservadas son las órdenes del lenguaje de programación. El compilador espera esos identificadores para comprender el programa, compilarlo y poder ejecutarlo. Por lo tanto, está prohibido utilizar palabras clave para nombrar variables dentro del programa. Tampoco se pueden utilizar caracteres especiales (+, -, /,...).*

## 2.7. OPERADORES Y EXPRESIONES.

Los operadores aritméticos se utilizan para realizar operaciones matemáticas:

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

Ejemplos:

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

## 2.7. OPERADORES Y EXPRESIONES.

Los operadores relacionales se utilizan para evaluar la igualdad y la magnitud:

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

Ejemplos:

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

## 2.7. OPERADORES Y EXPRESIONES.

Los operadores lógicos se utilizan para realizar operaciones lógicas:

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

Ejemplos:

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

## 2.7. OPERADORES Y EXPRESIONES.

Los operadores unitarios solo afectan a un operando:

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

Ejemplos:

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



## 2.7. OPERADORES Y EXPRESIONES.

Los operadores de bits realizan operaciones lógicas o de desplazamiento a nivel de bits:

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

Ejemplos:

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

## 2.7. OPERADORES Y EXPRESIONES.

Los operadores de asignación permiten combinar la operación de asignación con operaciones matemáticas:

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

Operador condicional:



( ) ? :      Ejemplo: num2 = (num1==1) ? 2 : 3;

Ejemplos:

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

## 2.7. OPERADORES Y EXPRESIONES.

La precedencia de operadores permite saber qué operadores se evalúan antes que otros en caso de ambigüedad:

MAS PRIORIDAD		OPERADORES
		( ) [ ] . -- ~ ! ++ -- new (tipo)expresión * / % + - << >> >>> < <= > >= instanceof == != & &   &&    ?: = *= /= %= += -= <<= >>= >>>= &=  = ^=
		
MENOS PRIORIDAD		

Ejemplos:

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

## 2.8. CONVERSIONES DE TIPOS.

Existen dos tipos de conversiones de tipos de datos:

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

```
byte a=1; short b=2;  
b=a;
```

Conversiones explícitas. En este caso el programador fuerza la conversión mediante una operación llamada “cast”:

```
byte a=1; int b=2;  
a=(byte)b; //posible pérdida de información!!
```

*No se pueden realizar conversiones entre tipos incompatibles, por ejemplo, entre enteros y booleanos, o entre reales y booleanos.*

## 2.9. PROGRAMACIÓN DE CONSOLA.

Java permite la entrada por teclado y la salida de información a través de la pantalla mediante la clase *System* del paquete *java.lang*. La clase *System* contiene, entre otros, tres objetos los cuales están asociados a tres flujos estándar que se abren cuando se ejecuta el programa y se cierran cuando éste finaliza.

Estos objetos (*static*) son los siguientes:

*System.out*. Referencia a la salida estándar (pantalla).

*System.in*. Referencia a la entrada estándar (teclado).

*System.err*. Referencia a la salida de error estándar (pantalla). Utilizado para mostrar mensajes de error.

Tanto la salida de error como la salida estándar disponen de los métodos *print()* y *println()*.

## 2.9. PROGRAMACIÓN DE CONSOLA.

La entrada estándar por teclado *System.in* no es una manera cómoda de pedir datos. Existe una clase en Java llamada *Scanner* del paquete *java.util* que facilita esta tarea.

Los métodos disponibles en esta clase para la lectura de distintos tipos de datos son:

Tipo	Método a invocar
byte	<code>teclado.nextByte();</code>
short	<code>teclado.nextShort();</code>
int	<code>teclado.nextInt();</code>
long	<code>teclado.nextLong();</code>
float	<code>teclado.nextFloat();</code>
double	<code>teclado.nextDouble();</code>
boolean	<code>teclado.nextBoolean();</code>
String	<code>teclado.next();</code>
String	<code>teclado.nextLine();</code>

## 2.9. PROGRAMACIÓN DE CONSOLA.

Ejemplo de uso de la clase *Scanner*:

```
import java.util.Scanner;
public class PedirDatos {
    public static void main(String[] args) {
        // El valor del numero pi
        final double PI = 3.1415926536;
        double radio;
        double altura;

        Scanner teclado = new Scanner(System.in);

        System.out.println("Introduzca los datos del cilindro:");
        System.out.print("Radio: ");
        radio = teclado.nextDouble();
        System.out.print("Altura: ");
        altura = teclado.nextDouble();
        System.out.print("El área del cilindro es: ");
        System.out.println(PI * radio * radio * altura);
    }
}
```

## 2.10. SALIDA CON FORMATO.

Java utiliza el método *System.out.printf* para mostrar datos con formato (similar a C). El primer argumento del método es una "cadena de formato" que puede consistir de texto fijo y especificadores de formato.

*Printf* imprime el texto fijo al igual que *print* o *println*. Cada especificador de formato recibe un valor y especifica el tipo a imprimir. Los especificadores de formato empiezan por el signo % y van seguidos de un carácter que representa el tipo de datos.

*Printf* permite:

- Redondear valores de punto flotante a un número indicado de decimales.

- Alinear una columna de números con puntos decimales.

- Justificar a la derecha o izquierda.

- Inserción de caracteres literales en posiciones precisas.

- Visualizar de tipos de datos con anchuras de campo de tamaño fijo.

- Visualizar fechas y horas en diversos formatos.

- ...



## 2.10. SALIDA CON FORMATO.

### Tipos de especificadores de formato:

'd'	Muestra un entero decimal (base10)
'o'	Muestra un entero octal (base8)
'x' o 'X'	Muestra un entero hexadecimal (base16)
'e'	Muestra un valor de punto flotante en notación expon.
'f'	Muestra un valor de punto flotante en formato decimal
'c' o 'C'	Muestra un carácter individual
's' o 'S'	Muestra una cadena de caracteres
'tc','tF','tD','tr',...	Muestra distintos formatos de fecha
'b'	Muestra el valor 'true' o 'false' de un booleano
'\n', '\t'	Retorno de línea y tabulador

## 2.10. SALIDA CON FORMATO.

### Ejemplos:

```
System.out.printf("%d\n",-26);  
System.out.printf("%f\n", 12345678.9);  
System.out.printf("%s\n", "Esta es una cadena");  
System.out.printf("%tD\n", fechaHora);  
System.out.printf("%B\n", prueba);
```

El tamaño exacto de un campo se especifica mediante una "anchura de campo". Si la anchura es mayor que los datos a mostrar, se justifican a la derecha. Se especifica un entero entre el signo % y el carácter de conversión. También se puede especificar la precisión de los datos, en este caso, el valor de la anchura cambia su significado dependiendo el tipo de dato.

### Ejemplos:

```
System.out.printf("%4d\n", 123);  
System.out.printf("%9.3f\n", 123.456789);
```

## 2.10. SALIDA CON FORMATO.

*Printf* permite el uso de "banderas" para mejorar la visualización de los datos:

- '-' Justifica a la izquierda.
- '+' Muestra un signo + para valores positivos y- para valores negativos.
- ' ' Muestra un espacio.
- '0' Muestra 0's a la izquierda.
- ...

### Ejemplos:

```
System.out.printf( "%+d\t%+d\n", 786,-786 );  
System.out.printf( "% d\n% d\n", 547,-547 );  
System.out.printf( "%+09d\n", 452 );  
System.out.printf( "%09d\n", 452 );  
System.out.printf( "% 9d\n", 452 );
```

## TEMA 2

# ELEMENOS BÁSICOS DE UN PROGRAMA

FIN