Introducción a la Programación en Java

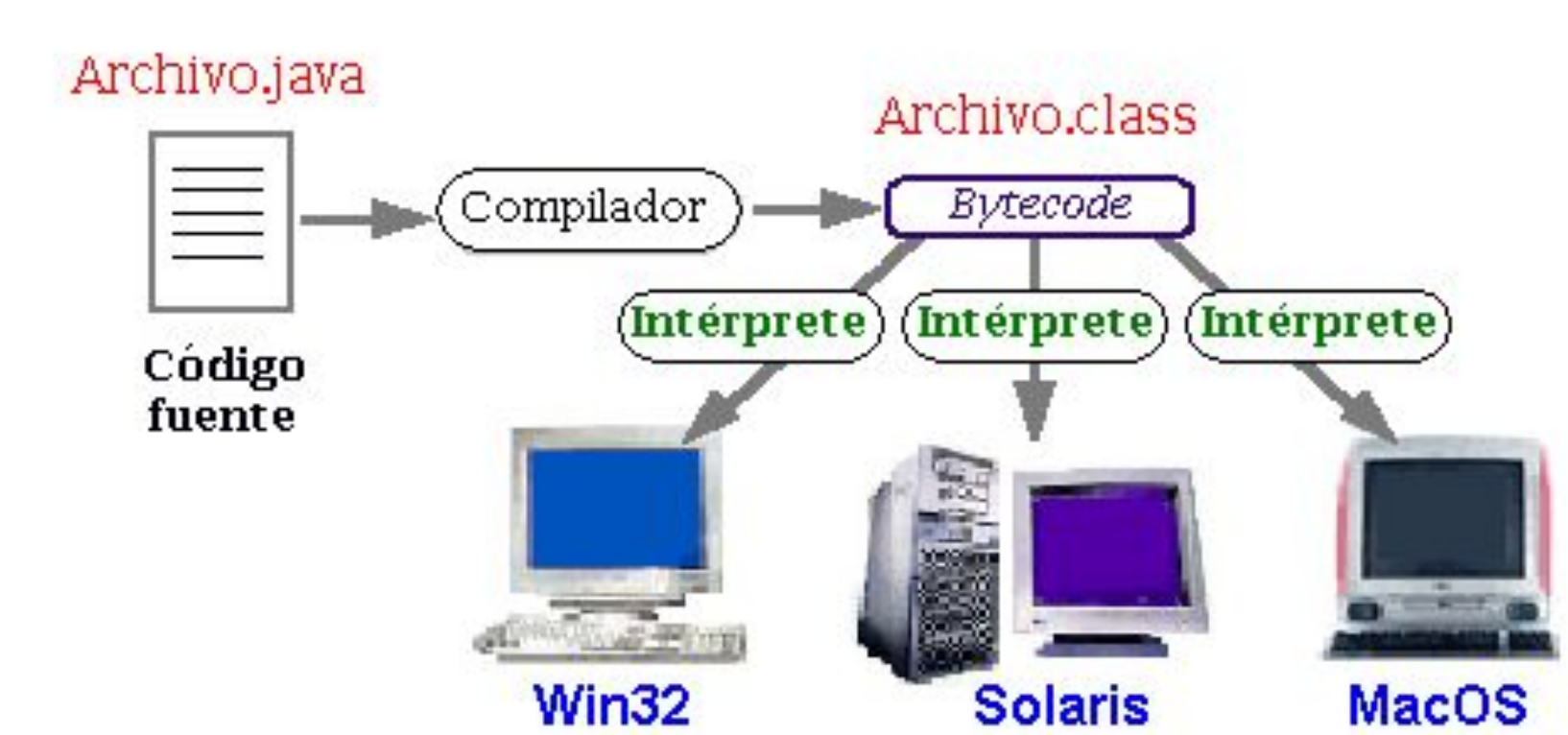
PROGRAMACIÓN
PRIMERO

Características Java

Java tiene una serie de características que lo distinguen como lenguaje de programación:

- James Gosling 1995
- Sun Microsystem Ahora Oracle
- Lenguaje de Alto Nivel
- Programación Orientada a Objetos
- Basado en Clases
- Para el desarrollo de todo tipo de programas
- Es un lenguaje interpretado → portable (JVM)
- Sintaxis similar a C y C++
- Distribuido y dinámico
- Robusto y seguro
- Multitarea

Características Java



Conceptos:

- Código con extensión .java
- Compilación comando javac → fichero .class
- Ejecución comando java → Intérprete → JVM

Estructura de un programa en Java

```
package paquete1.subpaquete1;

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

Conceptos:

- Paquete
- Clase
- Método main

- Crear un proyecto Java en Eclipse
- Crear un paquete
- Crear una clase + main
- Compilar / Ejecutar un programa

Variables

- **Declaración.**

  Al declarar una variable estamos reservando espacio en memoria para almacenar su futuro valor.

  El espacio dependerá del tipo que haya indicado en la declaración.

  Si la variable es de tipo primitivo, además, la variable quedará inicializada a un valor por defecto automáticamente.

- **Definición**

  Al definir una variable se le asigna un valor específico. La variable debe haber sido declarada.

Variables: Declaración Tipos Primitivos

Tipo	Uso	Tamaño	Rango
`byte`	entero corto	8 bits	de –128 a 127
`short`	entero	16 bits	de –32 768 a 32 767
`int`	entero	32 bits	de –2 147 483 648 a 2 147 483 647
`long`	entero largo	64 bits	± 9 223 372 036 854 775 808
`float`	real precisión sencilla	32 bits	de -10^{32} a 10^{32}
`double`	real precisión doble	64 bits	de -10^{300} a 10^{300}
`boolean`	lógico	1 bit	*true* o *false*
`char`	texto	16 bits	cualquier carácter

Variables: Declaración Tipos Primitivos

Existen otros tipos no primitivos. Por ahora conoceremos uno, el tipo **String**.

Las cadenas de caracteres se utilizan para almacenar palabras y frases. Todas las cadenas de caracteres deben ir entrecomilladas.

```
String miPalabra = "cerveza";
String miFrase = "¿dónde está mi cerveza?";

System.out.println("Una palabra que uso con frecuencia: " + miPalabra);

System.out.println("Una frase que uso a veces: " + miFrase);
```

Variables: Declaración Tipos Primitivos

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Variables: Declaración Sintaxis

- **Identificador.**

  Java es CaseSensitive

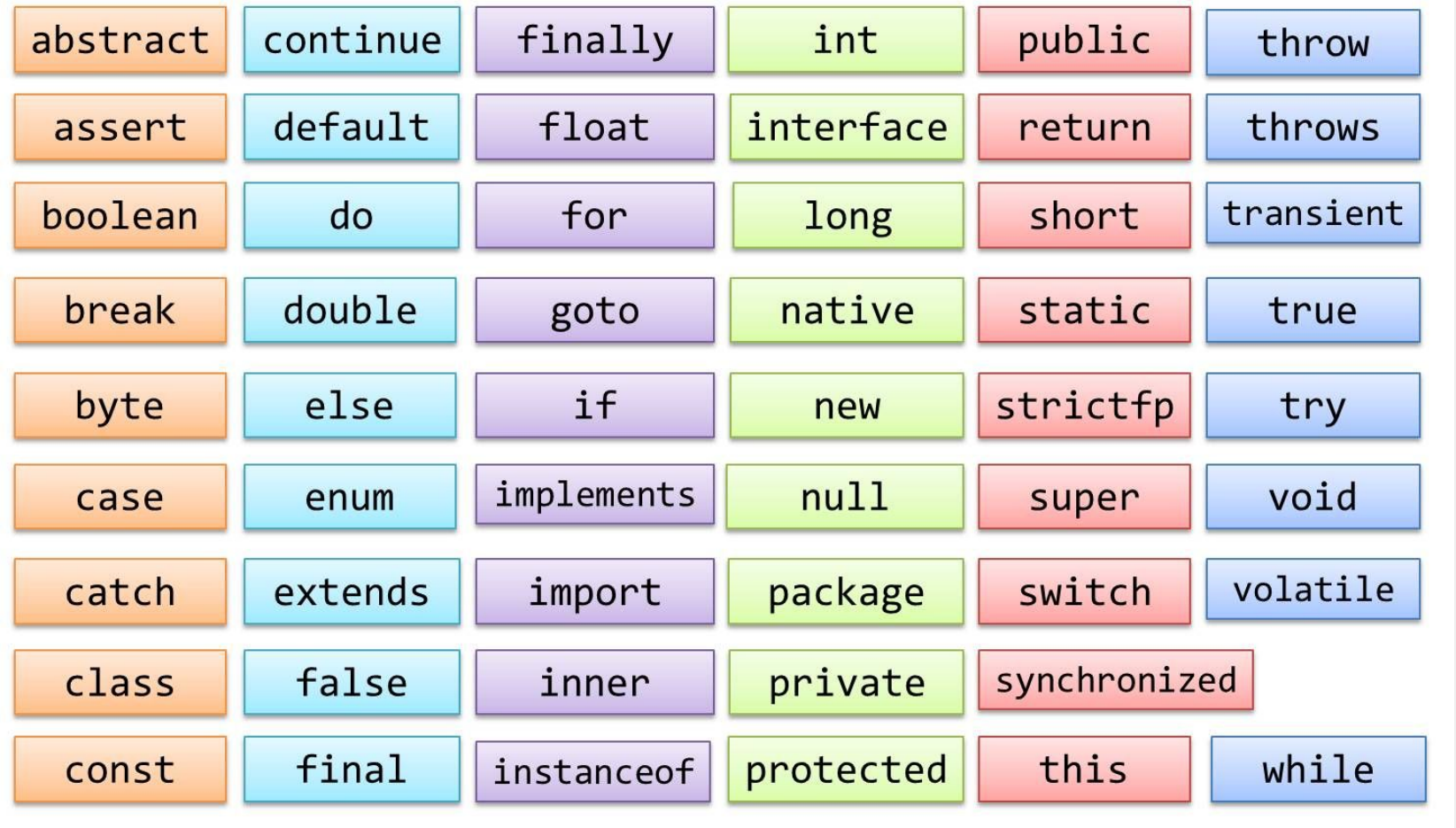
  No existe longitud máxima

  Puede empezar por letras, _o $ Es decir: [A-Z][a-z][0-9] _ $

  No puede usarse como nombre para una variable ninguna palabra reservada.

```
int variable1;

variable1 = 15;

char variable2;

String variable3 = "Hola";

float variable4;

boolean variable5;
```

Palabras reservadas



Es tu turno: Variables

Determinar cuáles de los siguientes son identificadores válidos, y cuáles son inválidos o no adecuados. ¿por qué?

a) registro1	b) 1registro	c) archivo_3
d) main	e) $impuesto	f) nombre y direccion
g) dirección	h) diseño	g) dirección

Es tu turno: Variables

Indica si es válido o no cada uno de estos ejemplos y porqué:

1. `int 1variable1 = 15;`
2. `double precio, descuento;`

   `precio = 50.0;`

   `descuento=0.15`

1. `apellido =  "Hola";`
2. `String $precio = "Hola";`
3. `int @nombreVariable = 15;`
4. `boolean encontrado1 = false;`
5. `int entero= 3000000000;`

Es tu turno: Variables

¿Qué identificador, y qué tipo de datos usarías para representar los valores siguientes?

a) La altura de un edificio.

b) Los años de un niño.

c) El sueldo de un empleado.

d) Indicar si una persona tiene hijos o no.

e) La letra de un apartamento.

f) Edad de una persona

g) Producto interior bruto de un país

h) Sexo de una persona

i) Estado civil

Constantes

Una constante es una variable para la que su valor no se modifica durante la ejecución de un programa.

Se definen con el modificador final delante del tipo de la variable.

```
final float PI = 3.141592f;
final String inicialEspana = "ES";
final int mayoriaEdad = 18;
```

Comentarios

Los comentarios pueden situarse en cualquier lugar del código

- **Comentario multilíneas**

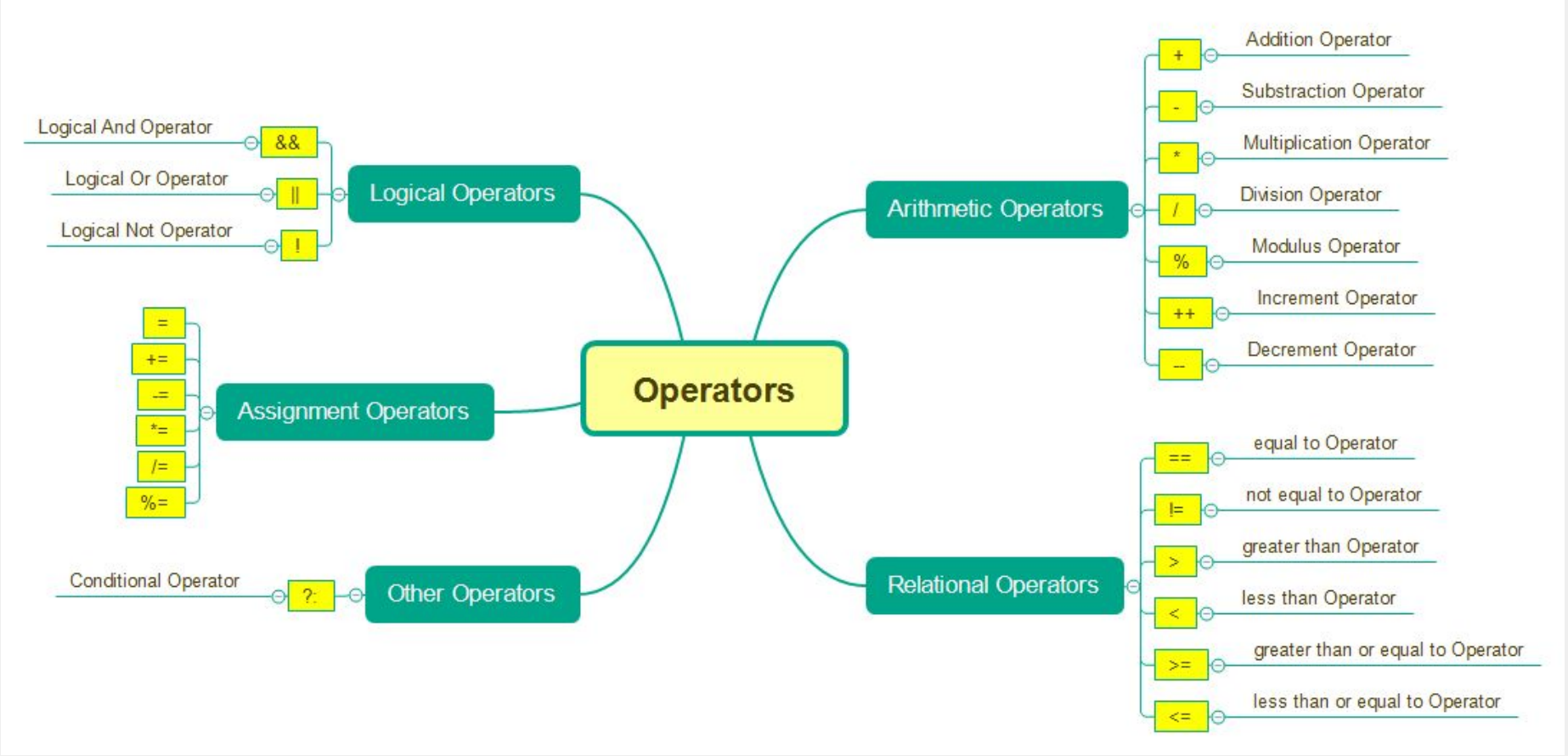
```
/*
 Comentario de varias líneas
*/
```

- **Comentario hasta final de línea**

```
// Comentarios de una línea
```

- **Comentario de documentación.** Los comentarios pueden seguir una estructura concreta interpretable por Javadoc. Esta herramienta genera una documentación automática a partir de los comentarios del código fuente.

Operadores


Operators
Logical Operators
Logical And Operator
&&
Logical Or Operator
||
Logical Not Operator
!
Assignment Operators
=
+=
-=
*=
/=
%=
Other Operators
Conditional Operator
?:
Arithmetic Operators
+
Addition Operator
-
Substraction Operator
*
Multiplication Operator
/
Division Operator
%
Modulus Operator
++
Increment Operator
--
Decrement Operator
Relational Operators
==
equal to Operator
!=
not equal to Operator
>
greater than Operator
<
less than Operator
>=
greater than or equal to Operator
<=
less than or equal to Operator

Operadores aritméticos

Símbolo	Descripción
+	Suma
+	Más unario: positivo
−	Resta
−	Menos unario: negativo
*	Multiplicación
/	División
%	Resto módulo
++	Incremento: +1
−−	Decremento: –1

```
int a, b, c; //declaramos la variables de tipo entero
a = 1;
b = -a; // b vale -1
```

es equivalente a

```
a ++;      a = a + 1;
b --;      b = b - 1;
```

```
a = 1; //a la variable a le asignamos 1
b = a ++; //primero asignamos el valor de a a b, y después incrementamos a
c = ++ a; //primero incrementamos a, y después asignamos su valor a c
```

Operadores relacionales

Son aquellos que producen un resultado lógico o booleano

En Java estos valores se representan mediante los literales `true` y `false`.

Símbolo	Descripción
==	Igual que
!=	Distinto que
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

3 <= 3 ¿es 3 menor o igual que 3? Es cierto.

3 != 4 ¿es 3 distinto de 4? Cierto; ya que 3 es distinto a 4.

3 < 5 ¿es 3 menor que 5? Es cierto; 3 es un número más pequeño que 5. Por tanto, la expresión devuelve `true`.

3 == 5 ¿es 3 igual que 5? Falso; ambos números son distintos, es decir, no son iguales. La expresión devuelve `false`.

3 <= 5 ¿es 3 menor o igual que 5? Cierto. La expresión devuelve `true`.

Operadores lógicos

Permiten operar a partir de expresiones lógicas, formando expresiones más complejas, que devuelven, a su vez, un valor lógico (Tabla 1.6). Existen los operadores *and* (conjunción *Y*), *or* (disyunción *O*) y *not* (negación).

Tabla 1.6. Operadores lógicos

Símbolo	Descripción
&&	Operador and: Y
||	Operador or: O
!	Operador not: Negación

Operadores lógicos

3 <= 5 && 2 == 2 ¿es 3 menor o igual que 5 y a la vez, es 2 igual a 2? Cierto, ya que tanto la expresión 3 <= 5 como 2 == 2 son ciertas.

3 <= 5 && 2 >10 ¿es 3 menor o igual que 5 y a la vez, es 2 mayor que 10? Falso, ya que al menos una expresión utilizada es falsa (2 > 10).

1 != 2 || 5 < 3 ¿es cierto que 1 sea distinto de 2 o que 5 sea menor que 3? La primera expresión es cierta: 1 es distinto de 2, mientras que la segunda expresión es falsa. Por tanto, la expresión completa se evalua como verdadera.

!(1 < 2) la expresión se evalua como la negación —lo contrario— de 1 menor que 2, que es verdadera. Por tanto, la expresión completa se evalua falsa.

Operadores asignación o de opera y asigna

Símbolo	Descripción
$+=$	Suma y asigna
$-=$	Resta y asigna
$*=$	Multiplica y asigna
$/=$	Divide y asigna
$\%=$	Módulo y asigna

```
var += 3;
```

Lo que es equivalente a:

```
var = var + 3;
```

```
x *= 2;
```

es equivalente a

```
x = x * 2;
```

Precedencia de operadores

operadores sufijo	[] . (params) expr++ expr--
operadores unarios	++expr --expr +expr -expr ~ !
creación o tipo	new (type)expr
multiplicadores	* / %
suma/resta	+ -
desplazamiento	<< >> >>>
relacionales	< > <= >= instanceof
igualdad	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
AND lógico	&&
OR lógico	||
condicional	? :
asignación	= += -= *= /= %= ^= &= |= <<= >>= >>>=

Es tu turno

Encontrar el valor de la variable **res** después de la ejecución de las siguientes secuencias de sentencias.

Todas las variables que aparecen se suponen reales:

a)
```
y = 4.0;
z = 3.0;
res = y / z - y *z;
```

b)
```
x = 3.0;
res = x - 3.0;
res = res - x;
```

Crea un programa con el que compruebes que lo que has pensado es correcto.

Conversiones entre tipos

```
int a = 2;
double x = 2.3;
```

```
int a = 2.6;
```

```
int a = 3;
double x = a;
```

conversiones de ensanchamiento.

Nos permitirá, por ejemplo, guardar valores `byte`, `short`, `int`, `long` o `float` en una variable `double`[4].

```
int a = (int) 2.6; //(int) indica el tipo al que se convertirá el valor
```

El *cast* es lo que va entre paréntesis. Lo que hace es eliminar (truncar) la parte decimal de 2.6 y convertirlo en el entero 2, que podrá ser asignado a la variable `a` sin problemas. Este tipo de conversión se llama de *estrechamiento*, ya que fuerza la asignación de un tipo de dato en una variable de menor tamaño, eso sí, con pérdida de información.

Salidas por consola

Una de las operaciones más básica que proporciona la API es aquella que permite mostrar mensajes en el monitor, con idea de aportar información al usuario que utiliza el programa. Cuando los mensajes se muestran de forma simple: en modo texto y sin interfaz gráfica, se habla de *salida por consola*. Java dispone para ello de la clase **System** con los métodos:

System.out.print("Mensaje"); que muestra literalmente el mensaje en el monitor.

System.out.println("Mensaje"); igual que el anterior pero, tras el mensaje, inserta un retorno de carro (nueva línea).

```
int edad = 8;
System.out.print("Su edad es de " + edad + " años.");

System.out.println("Hola.");
System.out.print("Encantando \nde \nconocerte."); //dos retornos de carro
System.out.print("Adiós.");
```

Entrada de datos

`Scanner` es una clase de la API que se utiliza de forma no estática, es decir, necesita del operador `new`. Y la forma de trabajar con ella es siempre la misma: en primer lugar tendremos que crear un nuevo escáner,

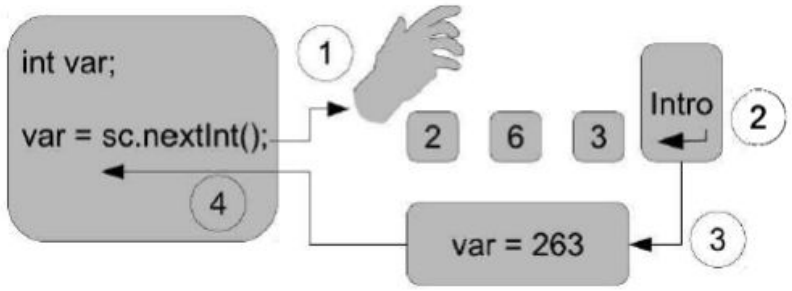
```
Scanner sc = new Scanner(System.in);
```

`System.in` indica que vamos a leer del teclado. Una vez que hemos creado nuestro escáner, que hemos llamado `sc`, ya solo queda utilizarlo. Para ello disponemos de los métodos:

- `sc.nextInt()`: que lee un número entero (`int`) por teclado.
- `sc.nextDouble`: lee un número real (`double`).
- `sc.nextLine()`: lee una cadena de caracteres hasta que se pulsa Intro.

Entrada de datos

```
Scanner sc = new Scanner(System.in); //creamos el nuevo escáner

double numero; //declaramos la variable numero
numero = sc.nextDouble(); //leemos por consola un valor double
//ahora disponemos del valor introducido, a través de la variable numero
System.out.println("Ha escrito: " + numero);
```



Entrada de datos

`boolean`	`nextBoolean()` Scans the next token of the input into a boolean value and returns that value.
`byte`	`nextByte()` Scans the next token of the input as a `byte`.
`String`	`next()` Finds and returns the next complete token from this scanner.
`double`	`nextDouble()` Scans the next token of the input as a `double`.
`float`	`nextFloat()` Scans the next token of the input as a `float`.
`int`	`nextInt()` Scans the next token of the input as an `int`.
`short`	`nextShort()` Scans the next token of the input as a `short`.
`String`	`nextLine()` Advances this scanner past the current line and returns the input that was skipped.
`long`	`nextLong()` Scans the next token of the input as a `long`.

Operadores avanzados: operador pre y post incremento

`++x → Operador pre-incremento`

1. Primero incrementa
2. Devuelve el valor

`x++ → Operador post- incremento`

1. Primero devuelve el valor
2. Segundo incrementa

Diferencias entre operador pre y post incremento

```
int k= 3;
int l = 3;
int j,h,z, w;
j = ++k; //j= 4, k = 4
h = l++; // h= 3, l = 4
z= ++k+j;
w= (l++)+h;
```

Es tu turno: ¿cuál es el valor de cada uno de las variables tras la ejecución de las dos últimas líneas?

Operadores avanzados: +=,=+, -=, =-, *=, =*, /=, =/

+=: Operador de Incremento

Realiza primero la suma y luego la asignación.

x +=a; // Es equivalente a x = x+a;

```
int x = 5;
x += 3; // Esto es equivalente a x = x + 3;
// Ahora, el valor de x es 8
```

=+: No es un Operador Estándar en Java

Se interpretará como una asignación simple, =

```
int y = 10;
y =+ 5; // Esto es equivalente a y = 5;
// Ahora, el valor de y es 5
```

Para el resto de operadores, el comportamiento es equivalente.

Operadores avanzados: evaluación perezosa

Java define:

- Dos operadores para la operación OR: el operador | y el operador ||
- Y otros dos operadores para la operación AND: el operador **&** y el operador **&&**

¿Por qué hace esto Java?La respuesta es por rendimiento gracias a la evaluación perezosa.

Operadores avanzados: evaluación perezosa OR

Recordemos que, para que una expresión sea cierta con el operador **OR**, basta con que **una de las dos partes sea cierta**.

Tanto el operador | como el operador || equivalen a la operación OR

Sin embargo, el operador || tiene evaluación perezosa.

```
boolean exprLogica = expresion1 || expresion2;
exprLogica = expresion1 | expresion2;
```

|| → Si expresion1 es cierto, no se ejecuta expresion2

|→ Aunque expresion1 sea cierta, se ejecutará expresion2

Operadores avanzados: evaluación perezosa AND

El operador AND implica que para que una expresión sea cierta, **AMBAS partes deben ser ciertas**. O lo que es lo mismo, en el momento en el que una es falsa, la expresión final es falsa.

La operación AND se representa con: el **operador & y el operador &&.**

¿Cuál es la diferencia? El operador && tiene evaluación perezosa.

```
boolean exprLogica = expresion1 && expresion2;
exprLogica = expresion1 & expresion2;
```

&& → Si expresion1 es falsa, no se ejecuta expresion2

& → Aunque expresion1 sea falsa, se ejecuta expresion2

Operadores avanzados: evaluación perezosa

Ejemplo1:

```
int i = 10;
int j = 12;
if( (i<j) | (i=3) > 5 )
    System.out.println("i:"+i);
```

Ejemplo2:

```
int i = 10;
int j = 12;
if( (i<j) || (i=3) > 5 )
    System.out.println("i:"+i);
```

Operadores avanzados: evaluación perezosa

Ejemplo3:

```
int i = 12;
int j = 10;
if( (i<j) & (i=3) > 5 )
    System.out.println("i:"+i);
```

Ejemplo4:

```
int i = 12;
int j = 10;
if( (i<j) && (i=3) > 5 )
    System.out.println("i:"+i);
```

Estructuras de control: Condicionales

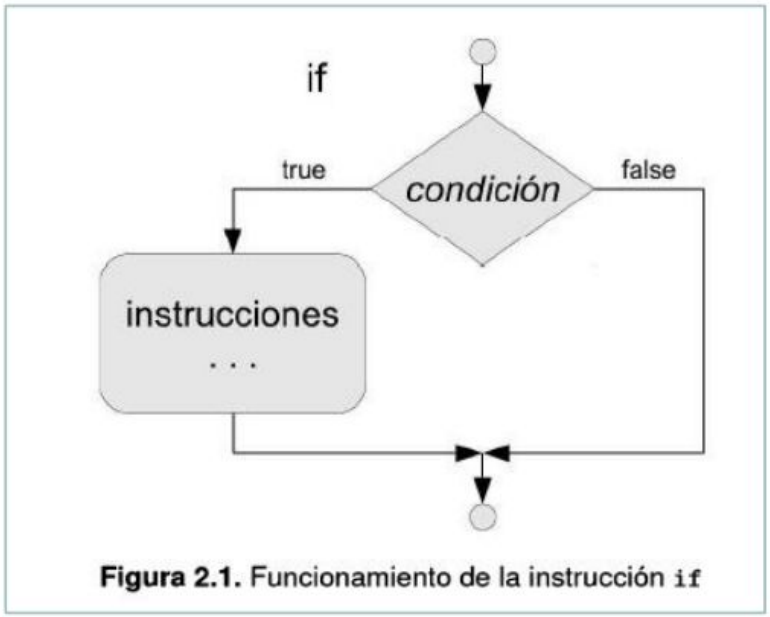


Figura 2.1. Funcionamiento de la instrucción `if`

```
if(condicion)
{
    /** instrucciones a
     * ejecutar
     * si se cumple
     */
}
```

Estructuras de control: Condicionales

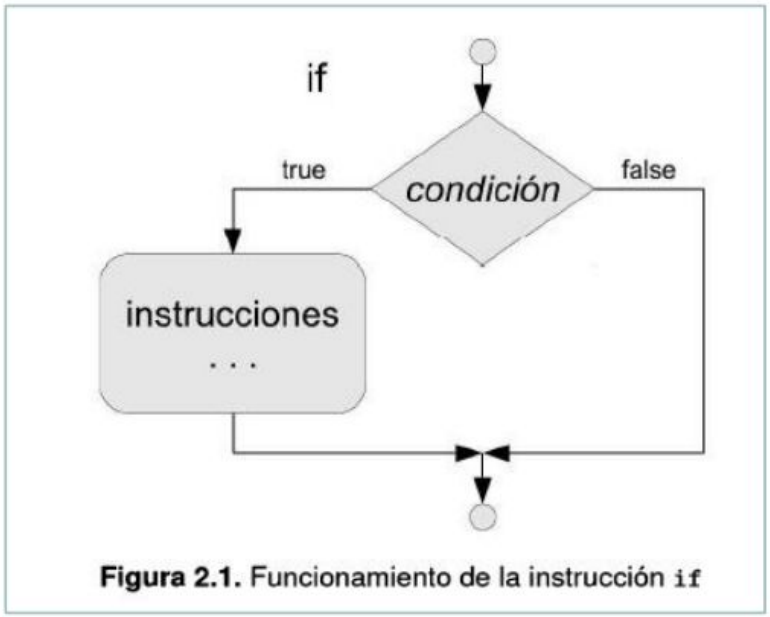


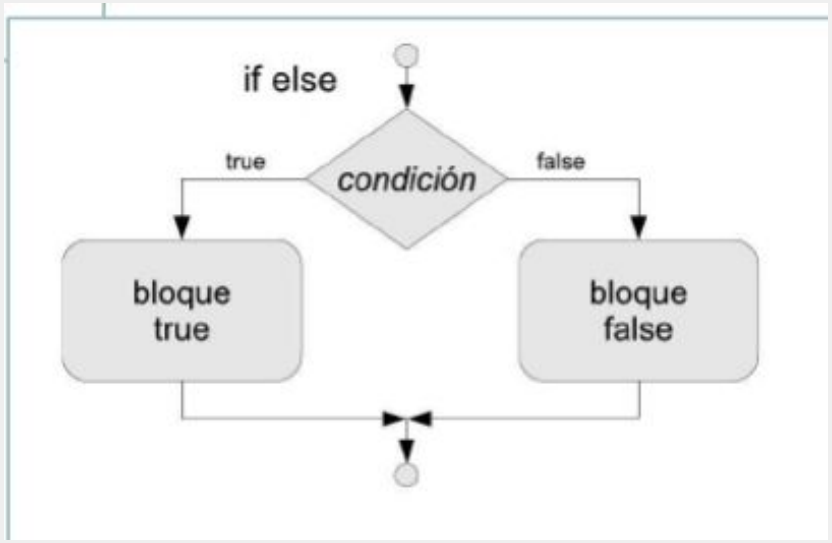
Figura 2.1. Funcionamiento de la instrucción `if`

```
if(condicion)
{
    /** instrucciones a
     * ejecutar
     * si se cumple
     */
}
```

```
a = 3;

if (a + 1 < 10) {
   a = 0;
   System.out.println("Hola");
}

System.out.println("El valor de a es: " + a);
```

Estructuras de control: Condicionales

```
if(condicion)
{
//instruccion si cumple
}
else
{
//instrucciones si no cumple
}
```




```
if (a < 0) {
   System.out.println("Valor negativo");
} else {
   System.out.println("Valor positivo");
}
```

Estructuras de control: Condicionales

```
if (condicion) {
    // cumple condicion
} else if (condicion2) {
    //cumple 2ª cond
} else {
    //en otro caso
}
```

```
if (a - 2 == 1) {
    System.out.println("Hola ");
} else if (a - 2 == 5) {
    System.out.println("Me ");
} else if (a - 2 == 8) {
    System.out.println("Alegro ");
} else if (a - 2 == 9) {
    System.out.println("De ");
} else if (a - 2 == 11) {
    System.out.println("Conocerte.");
} else {
    System.out.println("Sin coincidencia");
}
```

Estructuras de control: Condicionales

La estructuras condicionales pueden anidarse.

```
if (a - 2 == 1) {
   System.out.println("Hola ");
} else {
   if (a - 2 == 5) {
      System.out.println("Me ");
   } else {
      if (a - 2 == 8) {
         System.out.println("Alegro ");
      } else {
         if (a - 2 == 9) {
            System.out.println("De ");
         } else {
            if (a - 2 == 11) {
               System.out.println("Conocerte.");
            } else {
               System.out.println("Sin coincidencia");
            }
         }
      }
   }
}
```

Operadores avanzados: operador condicional

Operador ? :

exprLogica? expr1 : expr2

Este operador funciona de manera similar a una estructura condicional. Si se cumple la expresión lógica, entonces se ejecuta la expresión 1 y si no, se ejecuta la expresión 2

Ejemplo1:

Ejemplo de expresión	Resultado del ejemplo

a = 4;
b = a == 4 ? a+5 : 6-a;
b = a > 4 ? a*7 : a+8;
	b vale 9
	b vale 12

Ejemplo2:

```
int x= 1;
int y = (x == 1) ? 61: 90;
```

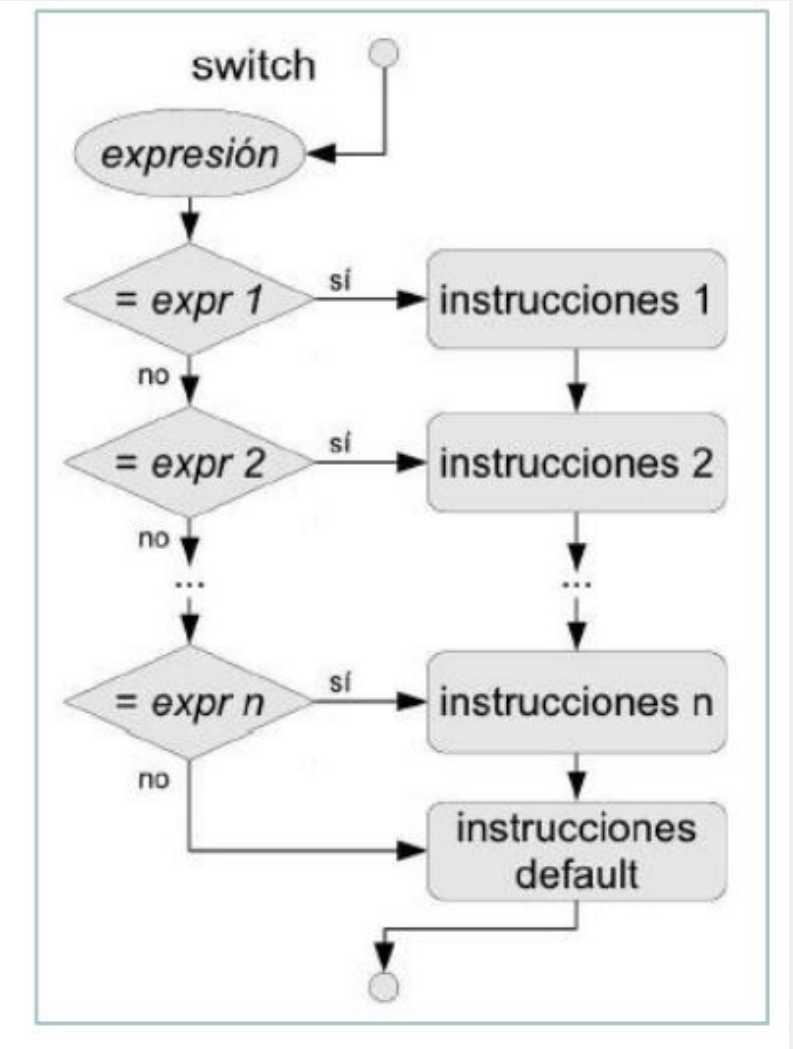
Operadores avanzados: operador condicional

Es tu turno, ¿qué imprime resultado?

```
int x=23;
int y=33;
int z=43;
int resultado = (x > y) ?
                (x > z ? x : z)
                : (y > z ? y : z);
System.out.println(resultado);
```

Estructuras de control: Condicionales

```
switch (expression)
{
    case 0:
       // para valor 0
       break;
    case 1:
       // para valor 1.
       break;
    …
    case valueN :
      // valor n
       break;
    default:
      //para el resto
}
```

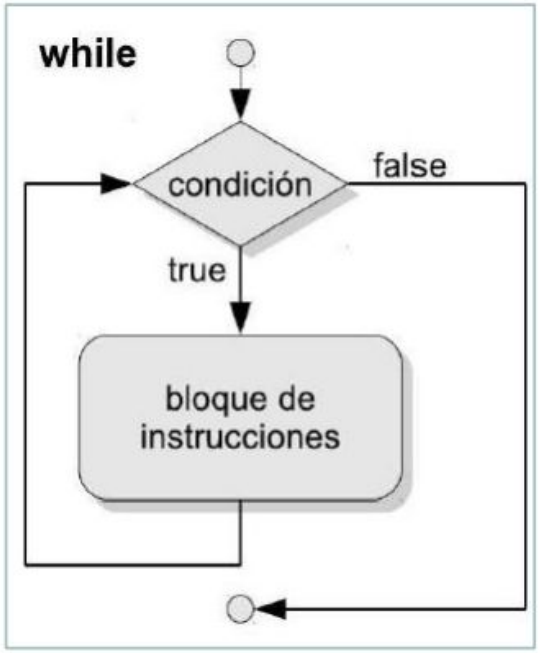


Es tu turno

- Averigua qué pasa si en un `switch` no incluyes la instrucción **break**;
- Averigua qué pasa si en un if no incluyes las **{}**
- Averigua para qué sirve en Eclipse

  Ctrl + BARRA_ESPACIADORA
- Prueba **syso Ctrl + BARRA_ESPACIADORA**
- *A diferencia de Python, en Java las tabulaciones sólo son por claridad estética, sin que tengan ningún impacto en el código a ejecutar. No obstante, la combinación* **Shift+Ctrl+F,** *te formatea el código*

Estructuras de Control: Iteraciones While

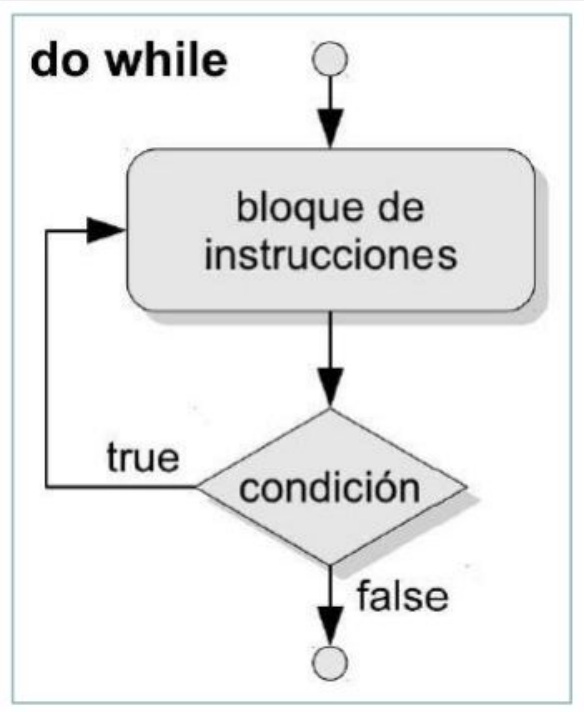



```
while (condición) {
    bloque de instrucciones
    ...
}
```

```
boolean encontrado = false;
int numeroAlAzar = (int) Math.random();
Scanner input = new Scanner(System.in);
int apuesta;
while (!encontrado)
{
        System.out.println("Adivina mi n°");
        apuesta = input.nextInt();
        if(apuesta == numeroAlAzar)
            encontrado = true;
}
```

¿Cuántas veces se ejecuta, como mínimo y como máximo?

Iteraciones Do While

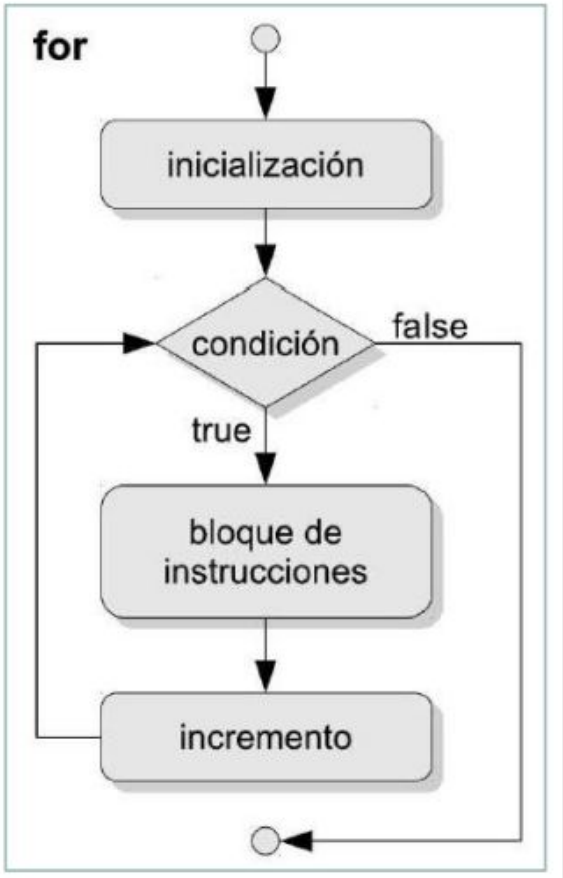



```
do {
    bloque de instrucciones
    ...
} while (condición);
```

```
int x = 8;
do
{
    /*Este bloque se
    imprime siempre */
    x++;
    System.out.println("x: "+x);
}
while (x < 10);
```

¿Cuántas veces se ejecuta, como mínimo y como máximo?

Iteraciones For




```
for (inicialización; condición; incremento) {
    bloque de instrucciones
    ...
}
```

```
for (int i = 1; i < 3; i++)
{
    System.out.println("i: "+i);
}
```

Estructuras de control anidadas

Las estructuras de control pueden contener otras estructuras de control.

Recuerda que en Java NO se anida tabulando.

Ejemplo de bucles anidados:

```
for (int i = 3; i < 6; i++)
{
    for (int j = 10 ; j > 3; j--)
        System.out.println("i+j vale:"+(i+j));
   System.out.println("==================");
}
```

Estructuras de control anidadas

```
for (i = 1; i <= 2; i++) {
   System.out.println("Bucle externo, i=" + i);
   for (j = 1; j <= 3; j++) {
      System.out.println("...Bucle medio, j=" + j);
      for (k = 1; k <= 4; k++) {
         System.out.println("......Bucle interno, k=" + k);
      }
   }
   System.out.println("");
}
```

Estructuras de control anidadas

```
for (i = 1; i <= 3; i++) {
    System.out.println("Bucle externo, i=" +i);

    j = 1;
    while (j <= i) {
        System.out.println("...Bucle interno, j=" + j);
        j++;
    }
}
```

Estructuras de control anidadas

Es tu turno:

- Realiza la traza para el primer bloque anidado. ¿Cuál es el último valor que se imprime?
- Realiza la traza para el segundo bloque anidado, ¿Cuál es el valor final para i, j y k?
- Para el tercer bloque de código, realiza la traza mostrando TODO lo que se imprime


¿Dudas?