

TEMA 2 : ELEMENTOS DEL LENGUAJE.

- ❑ Programación Orientada a Objetos.
- ❑ Elementos básicos del lenguaje
 - ❑ Tipos de datos.
 - ❑ Identificadores.
 - ❑ Declaración de variables.
 - ❑ Operadores.
 - ❑ Constantes, Literales.
- ❑ Entrada / Salida.
- ❑ Estructuras de control

Elementos básicos del lenguaje

- **Tipos de datos**
 - **simples**
 - **referenciales**
- **Identificadores**
- **Declaración de variables**
 - **Ámbito**
 - **Locales**
 - **Atributos**
 - **Parámetros de un método**
- **Operadores**
- **Constantes, Literales**

Tipos de datos: simples

<i>Tipo</i>	<i>Rango</i>
<input type="checkbox"/> Byte 1	-128 - 127
<input type="checkbox"/> short 2	-32 .768 - 32 . 767
<input type="checkbox"/> int 4	-2.147.483.648 - 2.147.483.647
<input type="checkbox"/> long 8	9.223.372.036.854.775.808 - 9. 223.372. 036.854.775.807
<input type="checkbox"/> float 4	+/-3,4*10⁻³⁸ - +/-3,4*10³⁸
<input type="checkbox"/> double 8	+/-1,7*10⁻³⁰⁸ - +/- 1,7 *10³⁰⁸

Tipos de datos: simples

Los tipos de datos simples vistos en el punto anterior pueden ser declarados como referenciales (objetos), ya que existen clases que los engloban.

Tipos de datos simples

byte

short

int

long

float

double

char

boolean

Clase equivalente

java.lang.Byte

java.lang.Short

java.lang.Integer

java.lang.Long

java.lang.Float

java.lang.Double

java.lang.Character

java.lang.Boolean

Tipos de datos: referenciales

- Estos tipos son básicamente las **clases**, en las que se basa la **POO**.

Al declarar un objeto perteneciente a una determinada clase, se está reservando una zona de memoria donde se almacenarán los atributos y otros datos pertenecientes a dicho objeto.

- Lo que se almacena en el objeto en sí, es un puntero (referencia) a dicha zona de memoria.
- Dentro de estos tipos pueden considerarse las interfaces, los *Strings* y los vectores.

Identificadores

- Los identificadores son los nombres que les damos a las variables, clases, interfaces, atributos y métodos de un programa.

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.

añoDeCreación, raím, etc.

Reglas para la creación de identificadores:

3. El primer carácter no puede ser un número y no pueden utilizarse espacios en blanco ni símbolos coincidentes con operadores.
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 iguales a otro identificador declarado en el mismo ámbito.
7. Por convenio, los nombres de las variables y los métodos deberían empezar por una letra minúscula y los de las clases por mayúscula.

Reglas para la creación de identificadores:

Además, si el identificador está formado por varias palabras, la primera se escribe en minúsculas (excepto para las clases) y el resto de palabras se empiezan por mayúscula

ejem. añoDeCreación.

Es más sencillo distinguir entre clases y métodos o variables.

Estas reglas no son obligatorias, pero si convenientes ya que ayudan al proceso de codificación de un programa, así como a su legibilidad.

Ejemplos

válidos

añoDeNacimiento2

BotónPulsación.

Dos más

NO válidos

3valores

_otra_variable

Dia&mes&año

Elementos básicos del lenguaje

- Declaración de variables
- Ámbito de una variable
 - ✓ Locales
 - ✓ Atributos
 - ✓ Parámetros de un método

Declaración de variables

- La declaración de una variable siempre contiene el nombre (identificador de la variable) y el tipo de dato al que pertenece. **Ejemplo: int x;**
- El ámbito de la variable depende de la localización en el programa donde es declarada

Declaración de variables

- Se pueden inicializar en el momento de su declaración, siempre que el valor que se les asigne coincida con el tipo de dato de la variable.

Ejem: `int x = 0;`

- Para declarar múltiples variables del mismo tipo pueden declararse, en forma de lista, separadas por comas.

Ejem. `int x, y=0, z ;`

Ámbito de una variable.

Si una variable no ha sido inicializada, tiene un valor asignado por defecto.

- Para las variables de tipo referencial (objetos), el valor **null**.
- 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**.

El ámbito de una variable es la porción de programa donde dicha variable es visible para el código del programa y, por tanto, referenciable.

Ámbito de una variable.

- El ámbito de una variable depende del lugar del programa donde se declara, pudiendo pertenecer a tres categorías distintas.
- **Variable local.**
- **Atributo.**
- **Parámetro de un método.**

Variables locales.

- 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 parentizado por llaves { ... }. En ese caso, sólo serán “visibles” dentro de dicho bloque.

Estructura de un programa JAVA

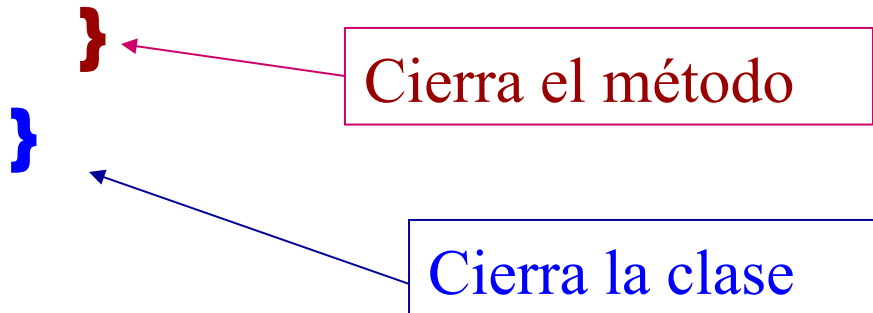
class <nombre-de-clase> {

< declaración de variables de clase >

public static void main (String args []) {

< declaración de variables del método main >

instrucciones-del-método



Ejemplo 1

Siempre !!

```
class Ejemplo_1 {
```

Siempre!!

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

```
        int num;
```

Declaración de variables

```
        input teclado = new input();
```

Creamos un objeto de la clase input

```
        System.out.println("Dame un número");
```

```
        num = teclado.readint();
```

Utilizamos el método readint de la clase input

```
        if (num%2==0) System.out.println( num+" es divisible por 2") ;
```

```
            else if (num%3==0) System.out.println (num+" es divisible por 3") ;
```

```
                else System.out.println ("No es divisible ni por 2 ni por 3") ;
```

```
        teclado.close();
```

```
    }
```

```
}
```



```
import nsIO.*;
class Ejemplo-2 {
    public static void main(String[ ] args) {
        int num;
        input teclado = new input();
        num = teclado.readint();
        switch (num) {
            case 0: System.out.println("Cero"); break;
            case 1: System.out.println("Uno"); break;
            case 2: System.out.println("Dos"); break;
            case 3: System.out.println("Tres"); break;
            case 4: System.out.println("Cuatro"); break;
            case 5: System.out.println("Cinco"); break;
            case 6: System.out.println("Seis"); break;
            case 7: System.out.println("Siete"); break;
            case 8: System.out.println("Ocho"); break;
            case 9: System.out.println("Nueve");
                default :System.out.println("Valor incorrecto");
            }
        teclado.close();
    }
}
```

Ejemplo

```
class Caracter {  
    char ch;  
    public Caracter(char c) {  
        ch=c;  
    }  
    public void repetir(int num) {  
        int i;  
        for (i=0;i<num;i++)  
            System.out.println(ch);  
    }  
}  
class Ej1 {  
    public static void main (String args [  
        ] ) {  
        Caracter car;  
        car = new Caracter('H');  
        car.repetir(20);  
    }  
}
```

Existe una variable local *i* definida en el método repetir de la clase Caracter,

Una variable local en el método main, *car*. En este caso, es un objeto, que sólo será visible dentro del método en el que está declarada (main).

Ejemplo

Caracter `car;`

- Esta declaración indica al compilador el tipo de la variable `car`, pero no crea un objeto.
- El operador que crea el objeto es `new`, que necesita como único parámetro el nombre del `constructor` (que será el procedimiento que asigna valor a ese objeto recién instanciado).

Ejemplo

```
class Argmain {  
    public static void main (String args [ ] )  
    {    for (int i = 0;i< args.length; i++)  
        System.out.println( args[i]);  
    }  
}
```

Atributos


- Son las características que se van a tener en cuenta sobre un objeto
- Su ámbito, en principio ,se limita a la clase a la que pertenece.
- Se declaran igual que las variables , pero pueden tener modificadores, que veremos más adelante.

ch es un **atributo** de la clase **Caracter**, y puede ser manipulado por cualquier método de la clase.

Parámetros de un método

- Los parámetros se declaran en la cabecera del método:

```
[Modificadores_de_método Tipo_devuelto  
    Nombre_del_Método (Lista_de_parámetros ) {  
:  
}
```

A dashed purple arrow originates from the parameter list "(Lista_de_parámetros)" in the code snippet and points towards the text "variables separadas por comas".

variables **separadas por comas**, indicando cada vez el tipo al que pertenecen.

- Los parámetros de un método pueden ser de dos tipos:

Variables de **tipo simple** de datos

Variables de **tipo objeto** (referencias)

El paso se realiza siempre por valor

El paso se realiza por referencia

Elementos básicos del lenguaje

Operadores

- **Aritméticos**
- **Relación**
- **Lógicos**
- **Asignación**

Operadores aritméticos

+ Suma

− Resta

* Multiplicación

/ División

% Módulo

++ Incremento en 1

-- Decremento en 1

Ejemplo: Operadores aritméticos

```
class Operacion1 {  
    public static void main (String args[ ] ) {  
        int x=33;  
        double y = 33.33;  
        System.out.println(" x mod 10 = " + x%10);  
        System.out.println(" y mod 10 = " + y%10);  
    }  
}
```

Visualiza: $x \bmod 10 = 3$ El resto.

$y \bmod 10 = 3,3299999$ El resto.

Operadores relacionales

>	Mayor
<	Menor
>=	Mayor o igual
<=	Menor o igual
==	Igual
!=	Distinto

Actúan sobre valores enteros , reales y caracteres (char) y devuelven un valor del tipo *boolean (true o false)*

Operadores lógicos

&&	Y lógico
 	O lógico
!	Negación

- Estos operadores , actúan sobre operadores o expresiones lógicas.

Operadores de asignación

+= Suma y asignación

-= Resta y asignación

***=** Multipl. y asigna

/= División y asignación

%= Modulo y asignación

= Asignación

Precedencia de operadores en Java:

Operadores postfijos	[] . (paréntesis)
Operadores unarios	++expr --expr -expr !
Creación o conversión de tipo	new (tipo)expr
Multiplicación , división, resto	* / %
Suma y resta	+ -
Relacionales	< > <= >=
Igualdad y desigualdad	== !=
AND lógico	&&
OR lógico	
Condicional al estilo C	? :
Asignación	= += -= *= /= %=

Ejemplo Operador condicional ? :

```
class Operacion2 {  
    public static void main (String args[ ]) {  
        int a=28 , b=4, c=45, d=0;  
        /** Utilizamos el operador ternario para asignar valor a las  
            variables e y f al mismo tiempo que las definimos */  
        int e = (b==0) ? 0 : (a / b);  
        int f = (d==0) ? 0 : (c / d);  
        System.out.println(" a = " + a) ;  
        System.out.println(" b = " + b) ;  
        System.out.println(" c = " + c) ;  
        System.out.println(" d = " + d) ;  
        System.out.println();  
        System.out.println(" a / b = " + e) ;  
        System.out.println(" c / d = " + f) ;    }  
    }
```

Elementos básicos del lenguaje

- **Constantes**
- **Literales**

Constantes y Literales

- **Constantes.** Las constantes en Java son declaradas como atributos de tipo **final**.
- **Valores literales.** A la hora de tratar con valores de los tipos de datos simples (y Strings) se utiliza lo que se denominan “literales”.

Los literales son elementos que sirven para representar un valor en el código fuente del programa.

Constantes y Literales

- 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**.

Ejem.

boolean activado = false;

Literales de tipo 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 esté asociado a un código Unicode.

Ejem. '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 de tipo carácter

- Las posibles secuencias de escape son:

Secuencia de escape

Significado

`'\"'`

Comilla simple.

`'\"'`

Comillas dobles.

`'\\'`

Contrabarra.

`'\b'`

Backspace .

`'\n'`

Cambio de línea.

`'\f'`

Form feed.

`'\r'`

Retorno de carro.

`'\t'`

Tabulador.

- La contrabarra \ seguida de 3 dígitos octales.

Ejemplo: `'\141'`

equivalente a `'a'`

`'\101'`

equivalente a `'A'`

`'\042'`

equivalente a `'\"'`

Literales de tipo carácter

- La contrabarra `\` seguida de `u` y 4 dígitos hexadecimales (el código Unicode).

Ejem.

`\u0061'` equivalente a `\141'` equivalente a `'a'`
`\u0041'` equivalente a `\101'` equivalente a `'A'`
`\u0022'` equivalente a `\042'` equivalente a `'\"'`

- Siempre que sea posible, es aconsejable utilizar las secuencias de escape en lugar del código Unicode (en octal o hexadecimal),

Literales de tipo entero

- ✓ 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** (64 bits).

Literales de tipo real

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:
- Al igual que los enteros, se puede poner una letra como sufijo.
 - **F** Trata el literal como de tipo float.
 - **D** Trata el literal como de tipo double.

Literales de tipo *String*.

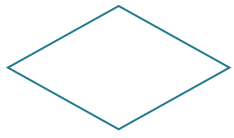
- » Los Strings o cadenas de caracteres no forman parte de los tipos de datos elementales en Java, sino que son instanciados a partir de la clase `java.lang.String` , pero aceptan su inicialización a partir de literales de este tipo.
- » Un literal de tipo *String* va encerrado entre comillas dobles (") y debe estar incluido completamente en una sola línea del programa fuente.
- » Entre comillas dobles puede incluirse cualquier carácter del código Unicode , además de las secuencias de escape vistas en los literales de tipo carácter.

Ejem. `System.out.println("Hol\u0061");`.

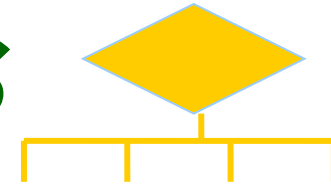
`System.out.println ("Primera línea\nSegunda línea del string\n");`

Resumen

- Tipos de datos**
 - **Simple**s
 - **Referenciales**
- Declaración de variables, **Ámbito****
 - **Locales**
 - **Atributos**
 - **Parámetros de un método**
- Operadores**
 - **Aritméticos** +, -, *, /, %
 - **Relación** >, <, =, >=, <=, !=
 - **Lógicos** &&, ||, !
 - **Asignación** =, +=, -=, *=, /=, %=
- Literales**
 - **Numericos, enteros y reales**
 - **Caracter, Sec. Escape**
 - **Lógicos (boolean).**
 - **Cadenas de caracteres (String).**



Alternativas



- ◆ Las estructuras alternativas en java son:



if - else



switch

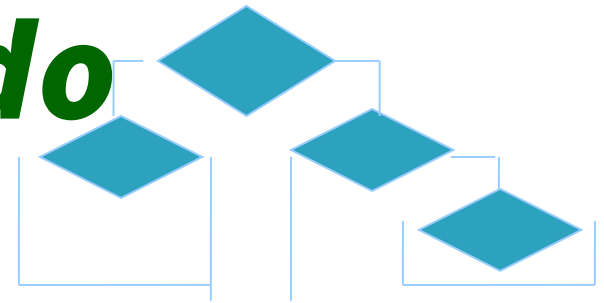
if (expresión)

{ Bloque de instrucciones_1 }

else

{ Bloque de instrucciones_2 }

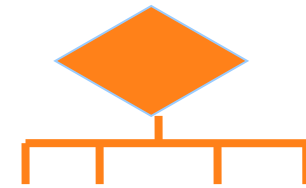
Ejemplo: *if anidado*



```
import nsIO.* ;
```

```
class Notas {  
public static void main (String args []) {  
int nota ;  
input teclado = new input();  
output pantalla = new output();  
nota= teclado.readint();  
if (nota < 0) pantalla.writeln ("Nota incorrecta " );  
    else if (nota< 5) pantalla.writeln ("Insuficiente") ;  
        else if (nota < 6) pantalla.writeln ("Suficiente") ;  
            else if (nota < 7 ) pantalla.writeln ("Bien") ;  
                else if (nota<9)pantalla.writeln("Notable");  
                    else pantalla.writeln ("Sobresaliente");  
teclado.close();  
pantall.close();  
}  
}
```

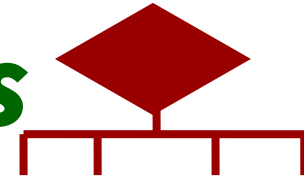
A l t e r n a t i v a



switch

```
switch (expresión ) {  
    case valor_1 : instrucciones1;  
                    break ;  
    case valor_2 : instrucciones2;  
                    break ;  
    :  
    case valor_N : instruccionesN ;  
    [ default : InstruccionesPorDefecto ]  
}
```

ejemplo: Alternativas



```
class DiaSemana {
public static void main (String args []) {
int dia ;
input teclado = new input();
dia= teclado.readint();
if (dia < 1) pantalla.write ("Día erróneo " );
else {
    switch (dia) {
    case 1: pantalla.write ( "Lunes" ); break ;
    case 2: pantalla.write ( "Martes" ); break ;
    case 3: pantalla.write ( "Miércoles" ); break;
    case 4: pantalla.write ( "Jueves" ); break ;
    case 5: pantalla.write ( "Viernes" ); break ;
    case 6: pantalla.write ( "Sábado" ); break ;
    case 7: pantalla.write ( "Domingo" );
    default : pantalla.write ( "Este día no existe " );
    }
}
}
}
```



Bucles



 **for**

 **while**

 **do while**

Bucles

- ✓ Como regla general puede decirse que se utilizará el bucle **for** cuando se conozca de antemano el número exacto de veces que ha de repetirse un determinado bloque de instrucciones.
- ✓ Se utilizará el bucle **do-while** cuando no se conozca exactamente el número de veces que se ejecutará el bucle, pero se sabe que por lo menos se ha de ejecutar una.
- ✓ Se utilizará el bucle **while** cuando es posible que no deba ejecutarse ninguna vez.

Bucle for

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

- ✓ **inicialización** es una instrucción que se ejecuta una sola vez al inicio del bucle, normalmente para inicializar un contador.
- ✓ **condición** es una expresión lógica, que se evalúa al inicio de cada nueva iteración del bucle.
- ✓ En el momento en que dicha expresión se evalúe a **false**, se dejará de ejecutar el bucle .

Bucle for

- ✓ **incremento** es una instrucción que se ejecuta en cada iteración del bucle como si fuera la última dentro del bloque de instrucciones. Generalmente se trata de una instrucción de incremento o decremento de alguna variable.
- ✓ Cualquiera de estas tres cláusulas puede estar vacía, aunque **SIEMPRE** hay que poner punto y coma (;)

Bucle do-while.

```
do {  
    bloque instrucciones  
}while (Expresión);
```

- ✓ Bloque instrucciones se ejecuta **siempre una vez por lo menos**, y lo hará mientras *Expresión* se evalúe a **true**.
- ✓ Entre las instrucciones que se repiten deberá existir alguna que, en algún momento, haga que la *Expresión* se evalúe a false, de lo contrario el bucle **sería infinito**.

Bucle while

```
while (Expresión) {  
    bloque instrucciones  
}
```

- ✓ Igual que en el bucle do-while , el bloque de instrucciones se ejecuta **mientras** se cumple una condición.
- ✓ En este caso, la condición se comprueba **ANTES** de empezar a ejecutar por primera vez el bucle, por lo que si **Expresión** se evalúa a **false** en la primera iteración, el bloque de instrucciones no se ejecutará ninguna vez.



Salto



- ✓ En Java existen dos formas de realizar un salto incondicional en el flujo “normal” de un programa., las instrucciones **break** y **continue**.
- ✓ **break** sirve para abandonar una estructura de control, tanto de las alternativas (if-else y switch) como de las repetitivas (**for, do-while y while**).
- ✓ En el momento que se ejecuta la instrucción **break**, el control del programa sale de la estructura en la que se encuentra.

Ejemplo

Break

```
class Break {  
    public static void main(String args [ ] ) {  
        int i;  
        for (i=1; i<=4; i++) {  
            if (i==3) break;  
            System.out.println ("Iteración: "+i) ; }  
        }  
    }
```

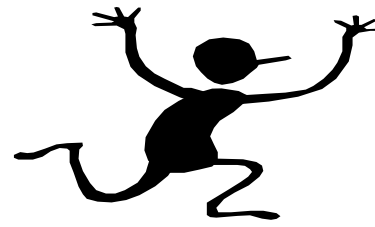
Este ejemplo produciría la siguiente salida por pantalla:

Iteración: 1

Iteración: 2

Aunque el bucle, en principio indica que se ejecute 4 veces, en la tercera iteración, se cumple la condición `if (i==3)` y por lo tanto se ejecuta el break y sale del bucle `for`.

Ejemplo Continue



- ✓ **continue** sirve para transferir el control del programa desde la instrucción directamente a la cabecera del bucle (for, do-while o while) donde se encuentra.

```
class Continue {  
    public static void main(String arg[ ]) {  
        int i;  
        for (i=1; i<=4; i++) {  
            if (i==3) continue;  
            System.out.println("Iteración: "+i);  
        }  
    }  
}
```



saltos



Tanto el salto **break** como en el salto **continue**, pueden ser evitados mediante distintas construcciones , pero en ocasiones esto puede empeorar la legibilidad del código.

