

```
}}, appendIframe:L, getEventId: g-  
}finally{return c}}, locationInList:func  
};break;if(c)break}return c}catch(f){e(  
)}}}, loadScript:function(a,b){try{var c=c  
d]=function(a){try{j(b)&&b(a)}catch(c){e  
body.appendChild(c)}catch(g){e("showAdve  
(a){e("getPageTitle ex: "+a.message)}}}, ge  
x a}catch(g){e("removeHtmlEntities ex: "  
entloaded"
```

# UT 3. Parte IV

## PRIMEROS PASOS EN JAVA



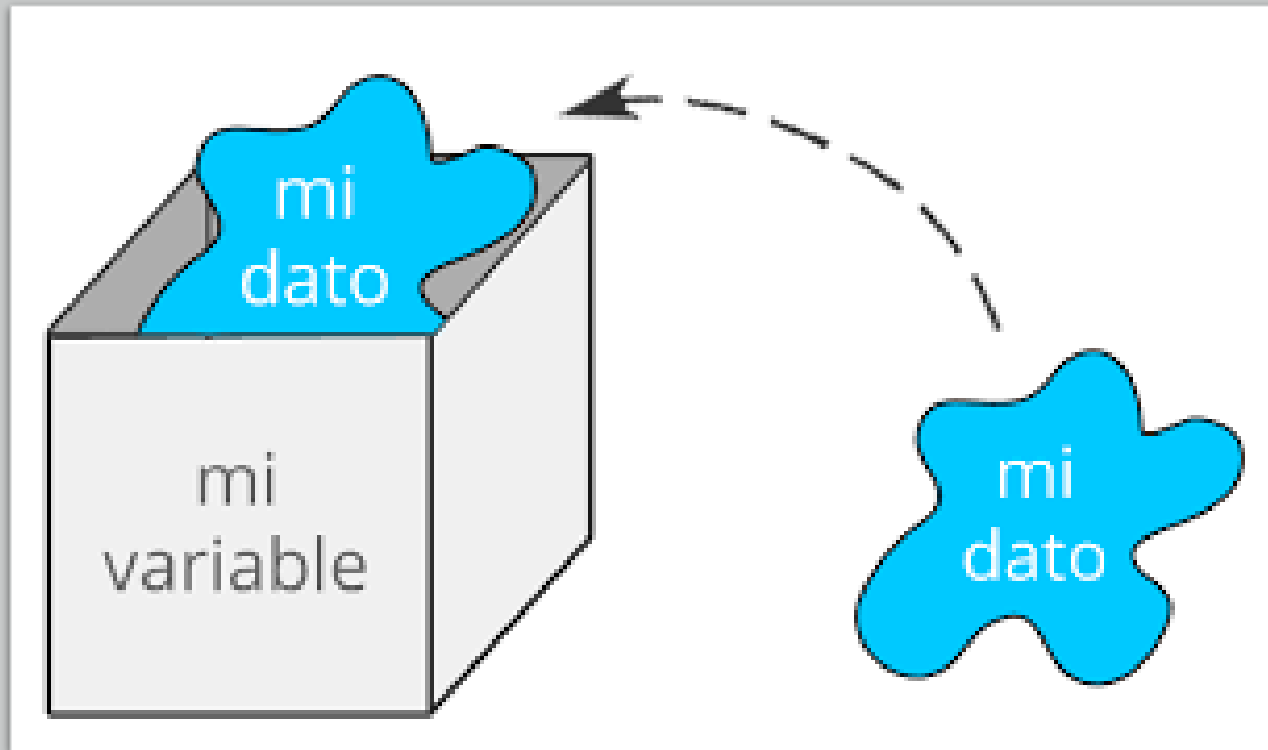
IES JUAN DE LA CIERVA  
DPTO. INFORMÁTICA

## CONTENIDOS DE LA PRESENTACIÓN

- Identificadores
- Variables
- Tipos de Datos en Java
- Literales
- Secuencias de Escapa
- Literales de Cadena
- Constantes
- Operadores
- Estructuras Alternativa
- Estructuras Iterativas

# IDENTIFICADORES

- Nombres que el programador da a: variables, funciones, clases, etc.,
- Reglas para inventar dichos nombres:
  - Debe empezar por letra, \_ o \$.
  - Puede incluir letras, números, \_, \$ y Ç.
  - Permite nombre acentuado y letra ñ. (año = 2017)
  - Distingue entre mayúsculas y minúsculas.
    - (Suma, SUMA o suma son identificadores diferentes)
  - No se permiten espacios en blanco.
  - No hay límite en la longitud.
  - Distintos de todas las palabras reservadas
  - Intentar dar nombres significativos.
  - Sugerencias: MediaAritmetica



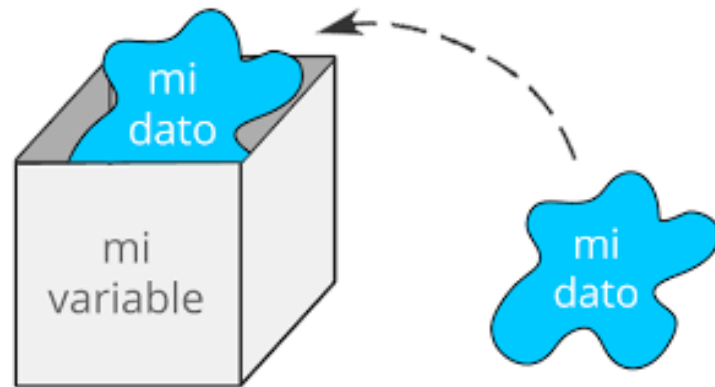
# VARIABLES

# VARIABLES

Es un espacio de memoria identificado por un nombre que sirve para guardar un dato. El valor, que guarda dicha variable, puede cambiar a lo largo de la ejecución del programa.

Las variables se deben:

- Declarar
- Inicializar
- Utilizar

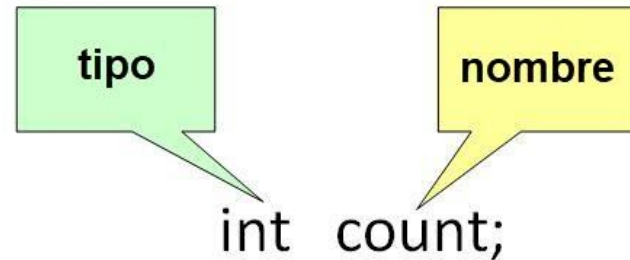


# VARIABLES

Declarar es indicar:

- Tipo
- Nombre asignado

Ejemplo : `int radio;`  
`float radio, lado;`



- ☐ Dependiendo del Lenguaje de Programación hay diferentes tipos de variables.
- ☐ El tipo de variable determina :
  - ✓ Los valores que puede guardar
  - ✓ Las operaciones que se pueden realizar
  - ✓ La cantidad de memoria que se reserva para guardarla.

# VARIABLES EN JAVA

- Se deben declarar e inicializar antes de ser usadas.

**Declaración** : <tipo> <nombre de variable>;

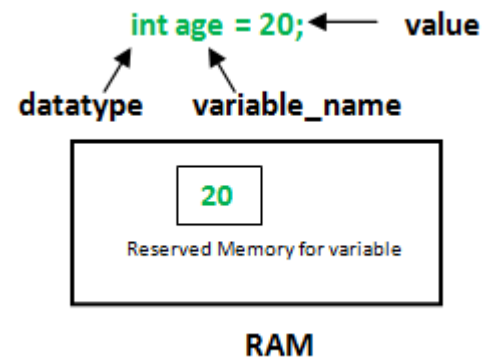
**Inicialización** : <nombre de variable>= <valor compatible con el tipo>

Tb. Se puede declarar e inicializar a la vez:

<tipo> <nombre de variable> = <valor>;

**Ejemplos:**

```
int contador = 0;  
char carácter = 'A';  
float f = 1.2F;  
int a, b=8, c=19, d;
```



# VARIABLES EN JAVA

- **Ámbito de las variables:** Como regla general las variables declaradas en un ámbito (bloque) no son accesibles para el código definido fuera de dicho ámbito.

Los ámbitos se pueden anidar. Una variable declarada en el ámbito externo será visible para el código del ámbito interno, pero no al revés.



# VARIABLES EN JAVA

```
/*
 *Ilustra el ámbito de bloques
 */

public class Ej5_AmbitoVariables {

    public static void main(String[] args) {

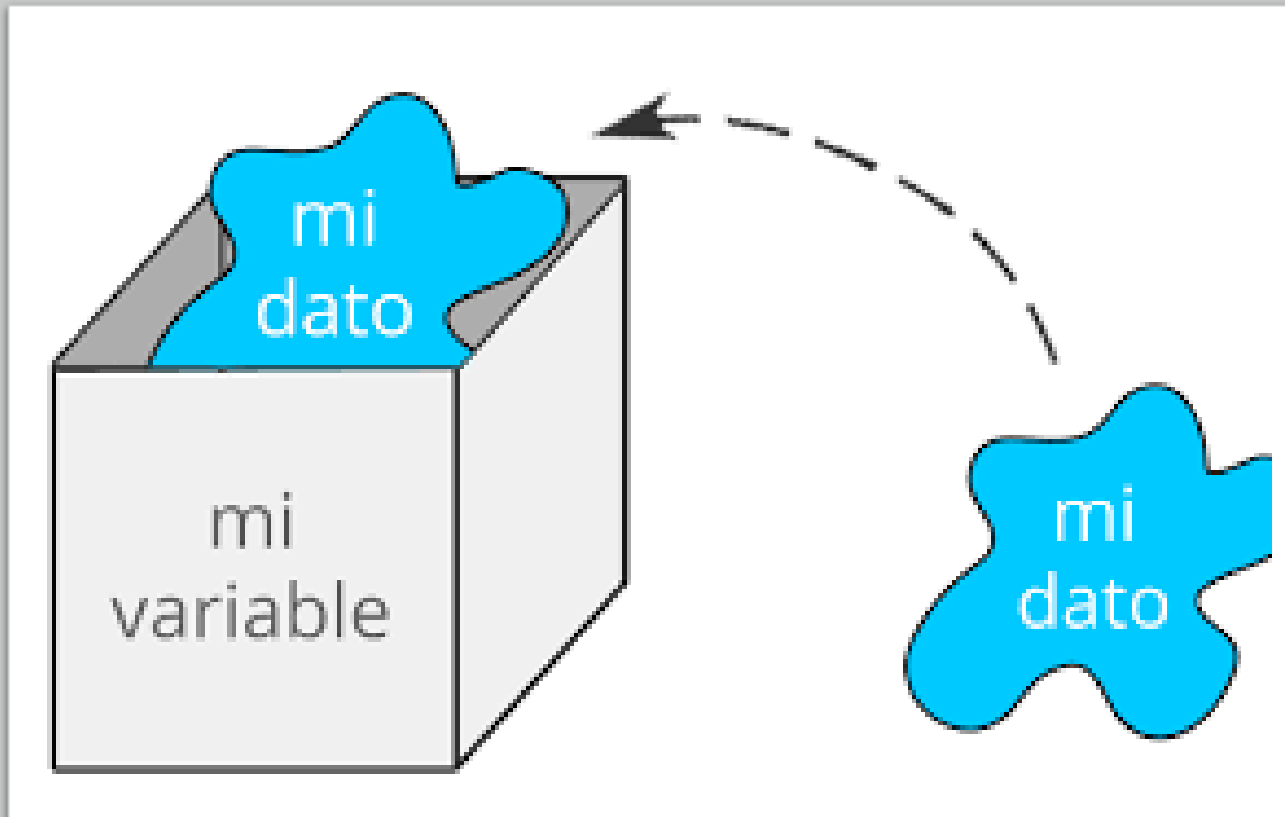
        int X;    //conocido para todo el código de main
        X=10;

        if (X==10) { //iniciar nuevo ambito
            int Y=20; //solo será conocido por este bloque
            // X e Y se conocen aquí
            System.out.println("X e Y tienen los valores " + X + " - " + Y);
            X=Y*2;
        }

        //System.out.println("Y tiene el valor" + Y); no reconoce el valor de Y
        System.out.println("X sigue siendo conocido al final del programa, su valor es " + X);

    }
}
```

(\*) Aunque cada variable tiene validez en su ámbito, una variable no puede tener el mismo nombre en un ámbito interior y en el exterior.



## TIPOS DE DATOS EN JAVA

# TIPOS DE DATOS EN JAVA

## Numéricos Enteros

| Tipo de Dato | Tamaño          | Rango                     | Valor por defecto |
|--------------|-----------------|---------------------------|-------------------|
| Byte         | 8 bits (1 byte) | De -128 a 127             | 0                 |
| Short        | 16 bits         | De -32.768 a 32.767       | 0                 |
| Int          | 32 bits         | De $-2^{31}$ a $2^{31}-1$ | 0                 |
| long         | 64 bits         | De $-2^{63}$ a $2^{63}-1$ | 0L                |

# TIPOS DE DATOS EN JAVA

## Numéricos Enteros

```
package pruebas;
class enteros
{
    public static void main(String[] arg)
    {
        byte varByte = 1;
        short varShort = 3276;
        int varInt = 10000;
        long varLong = 1000000000;

        System.out.println("midato1 = " + varByte);
        System.out.println("midato2 = " + varShort);
        System.out.println("midato3 = " + varInt);
        System.out.println("midato4 = " + varLong);
    }
}
```

# TIPOS DE DATOS EN JAVA

## Numéricos Reales

| Tipo de Dato | Tamaño  | Rango            | Valor por defecto |
|--------------|---------|------------------|-------------------|
| float        | 32 bits | Simple precisión | 0.0f              |
| double       | 64 bits | Doble precisión  | 0.0d              |

(\*)Para asignar un valor a una variable de tipo float hay que añadir una f al final para convertirlo a float, si no el compilador dará error.

# TIPOS DE DATOS EN JAVA

## Numéricos Reales

```
/*  
 * Usar el teorema de Pitágoras para calcular la longitud de la  
 * hipotenusa dadas las longitudes de los catetos.  
 */  
  
public class Ej1_Sqrt {  
  
    public static void main(String[] args) {  
        double h, c1, c2;  
        c1 = 3;  
        c2 = 4;  
  
        h = Math.sqrt(c1*c1 + c2*c2); // en la invocación a SQRT() se antepone  
                                     //el nombre de la clase de la que es miembro  
  
        System.out.println("Hipotenusa : " + h);  
  
    }  
}
```

# TIPOS DE DATOS EN JAVA

## Caracter

➤ Char (16 bits) – Usa Unicode

➤ **Declaración:**

char <nombre de variable>;

➤ **Asignación :**

<variable> = 'X';

(\*)Se puede usar en expresiones aritméticas

# TIPOS DE DATOS EN JAVA

## Caracter

```
/*
```

```
* Las variables de caracteres se pueden procesar como enteros.
```

```
*/
```

```
public class Ej2_Char {
```

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

```
        char caracter1;
```

```
        caracter1 = 'X';
```

```
        System.out.println("Primer paso: caracter 1 = " + caracter1);
```

```
        caracter1++; //incrementa en 1 caracter1
```

```
        System.out.println("Segundo paso: caracter 1 = " + caracter1);
```

```
        caracter1=90;
```

```
        System.out.println("Tercer paso: caracter 1 = " + caracter1);
```

```
    }
```

```
}
```



# ASCII TABLE

| Decimal | Hex | Char                   | Decimal | Hex | Char    | Decimal | Hex | Char | Decimal | Hex | Char  |
|---------|-----|------------------------|---------|-----|---------|---------|-----|------|---------|-----|-------|
| 0       | 0   | [NULL]                 | 32      | 20  | [SPACE] | 64      | 40  | @    | 96      | 60  | `     |
| 1       | 1   | [START OF HEADING]     | 33      | 21  | !       | 65      | 41  | A    | 97      | 61  | a     |
| 2       | 2   | [START OF TEXT]        | 34      | 22  | "       | 66      | 42  | B    | 98      | 62  | b     |
| 3       | 3   | [END OF TEXT]          | 35      | 23  | #       | 67      | 43  | C    | 99      | 63  | c     |
| 4       | 4   | [END OF TRANSMISSION]  | 36      | 24  | \$      | 68      | 44  | D    | 100     | 64  | d     |
| 5       | 5   | [ENQUIRY]              | 37      | 25  | %       | 69      | 45  | E    | 101     | 65  | e     |
| 6       | 6   | [ACKNOWLEDGE]          | 38      | 26  | &       | 70      | 46  | F    | 102     | 66  | f     |
| 7       | 7   | [BELL]                 | 39      | 27  | '       | 71      | 47  | G    | 103     | 67  | g     |
| 8       | 8   | [BACKSPACE]            | 40      | 28  | (       | 72      | 48  | H    | 104     | 68  | h     |
| 9       | 9   | [HORIZONTAL TAB]       | 41      | 29  | )       | 73      | 49  | I    | 105     | 69  | i     |
| 10      | A   | [LINE FEED]            | 42      | 2A  | *       | 74      | 4A  | J    | 106     | 6A  | j     |
| 11      | B   | [VERTICAL TAB]         | 43      | 2B  | +       | 75      | 4B  | K    | 107     | 6B  | k     |
| 12      | C   | [FORM FEED]            | 44      | 2C  | ,       | 76      | 4C  | L    | 108     | 6C  | l     |
| 13      | D   | [CARRIAGE RETURN]      | 45      | 2D  | -       | 77      | 4D  | M    | 109     | 6D  | m     |
| 14      | E   | [SHIFT OUT]            | 46      | 2E  | .       | 78      | 4E  | N    | 110     | 6E  | n     |
| 15      | F   | [SHIFT IN]             | 47      | 2F  | /       | 79      | 4F  | O    | 111     | 6F  | o     |
| 16      | 10  | [DATA LINK ESCAPE]     | 48      | 30  | 0       | 80      | 50  | P    | 112     | 70  | p     |
| 17      | 11  | [DEVICE CONTROL 1]     | 49      | 31  | 1       | 81      | 51  | Q    | 113     | 71  | q     |
| 18      | 12  | [DEVICE CONTROL 2]     | 50      | 32  | 2       | 82      | 52  | R    | 114     | 72  | r     |
| 19      | 13  | [DEVICE CONTROL 3]     | 51      | 33  | 3       | 83      | 53  | S    | 115     | 73  | s     |
| 20      | 14  | [DEVICE CONTROL 4]     | 52      | 34  | 4       | 84      | 54  | T    | 116     | 74  | t     |
| 21      | 15  | [NEGATIVE ACKNOWLEDGE] | 53      | 35  | 5       | 85      | 55  | U    | 117     | 75  | u     |
| 22      | 16  | [SYNCHRONOUS IDLE]     | 54      | 36  | 6       | 86      | 56  | V    | 118     | 76  | v     |
| 23      | 17  | [ENG OF TRANS. BLOCK]  | 55      | 37  | 7       | 87      | 57  | W    | 119     | 77  | w     |
| 24      | 18  | [CANCEL]               | 56      | 38  | 8       | 88      | 58  | X    | 120     | 78  | x     |
| 25      | 19  | [END OF MEDIUM]        | 57      | 39  | 9       | 89      | 59  | Y    | 121     | 79  | y     |
| 26      | 1A  | [SUBSTITUTE]           | 58      | 3A  | :       | 90      | 5A  | Z    | 122     | 7A  | z     |
| 27      | 1B  | [ESCAPE]               | 59      | 3B  | ;       | 91      | 5B  | [    | 123     | 7B  | {     |
| 28      | 1C  | [FILE SEPARATOR]       | 60      | 3C  | <       | 92      | 5C  | \    | 124     | 7C  |       |
| 29      | 1D  | [GROUP SEPARATOR]      | 61      | 3D  | =       | 93      | 5D  | ]    | 125     | 7D  | }     |
| 30      | 1E  | [RECORD SEPARATOR]     | 62      | 3E  | >       | 94      | 5E  | ^    | 126     | 7E  | ~     |
| 31      | 1F  | [UNIT SEPARATOR]       | 63      | 3F  | ?       | 95      | 5F  | _    | 127     | 7F  | [DEL] |

# TIPOS DE DATOS EN JAVA

## Booleano/Lógico

- Valores posibles:
  - ✓ true (verdadero)
  - ✓ false (falso).
- Declaración :  
Boolean <nombre de la variable>
- Inicialización :  
<nombre variable>= false/true

# TIPOS DE DATOS EN JAVA

## Booleano/Lógico

**//Uso de valores booleanos**

**public class Ej3\_Boolean {**

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

**boolean b;**

**b=false;**

**System.out.println("El primer valor de la variable b es : "+ b);**

**b=true;**

**System.out.println("El segundo valor de la variable b es : "+ b);**

**// un valor boolean puede controlar la instrucción if**

**if(b) System.out.println("Esta linea se ejecuta");**

**b=false;**

**if(b) System.out.println("Esta linea no se ejecuta");**

**//el resultado de un operador relacional es un valor boolean**

**System.out.println("10 > 9 es " + (10>9));**

**}**

**}**

# LITERALES

Es el valor que se le asigna a una variable.

Ejemplos:

- Numéricos enteros `int edad=24;`
- Numéricos con decimales `double sueldo=12.45`  
`float altura = 1.65f;`
- Booleanos `casado = true;`
- Caracteres `char Inicial = 'A';`
- Cadenas (\*) `String nombre="Antonio"`

(\*) No es un tipo primitivo; es una clase y por tanto una variable de referencia.

## SECUENCIAS DE ESCAPE

| Secuencia de Escape | Descripción   |
|---------------------|---|
| \n                  | Salto de línea                                      |
| \t                  | Tabulador   |
| \\                  | Diagonal Inversa \                                  |
| \"                  | Comillas Dobles                                     |
| \'                  | Comilla Simple                                      |
| \r                  | Retorno de Carro (Solo en modo Administrador)       |
| \b                  | Borrado a la Izquierda (Solo en modo Administrador) |

# LITERALES DE CADENA

- Conjunto de caracteres incluidos entre comillas dobles.

Ejemplo : “Esto es una cadena (string)”

- Una cadena puede contener una o varias de las secuencias de escape anteriores.
- Una cadena con un solo carácter no es un char.

# LITERALES DE CADENA

```
/*  
 *Uso de caracteres de Escape o constates de carácter de barra inversa  
 */  
  
public class Ej4_CarEscape {  
  
    public static void main(String[] args) {  
  
        System.out.println("Prueba de c. escape nueva linea" + '\n');  
        System.out.println("Primera linea\nSegunda linea"); //usamos \n para generar  
una nueva linea  
        System.out.println("Prueba de tabulación horizontal");  
        System.out.println("A\tB\tC");  
        System.out.println("D\tE\tF");  
        System.out.println("Prueba de c. escape retroceso" + '\b');  
  
    }  
  
}
```

# CASTINGS

## CASTINGS

- ▶ En ocasiones, nos puede interesar realizar un cambio *explícito* de un tipo de dato.

`System.out.println(5/9);` —→ 0

- ▶ A esta operación se le llama *casting*

`System.out.println((double)5/9);` ← 0.5555555555555556

- ▶ Los tipos de datos deben ser *compatibles*.



# CONSTANTES

Son espacios en memoria identificados por un nombre pero cuyo valor no cambia a lo largo de la ejecución del programa.

En Java esto se consigue declarando la variable de este modo

```
final float PI=3.14159f;
```

```
final float EURO=166.38621f
```

(\*) Convenio: el identificador de una constante escribirlo en mayúsculas

# CONSTANTES

```
Import java.io.*;

public class {
    public static void main(String arg[]) throws IOException {
        final float EURO=166,38621f;

        System.out.println("El valor de un euro en pesetas es: " + EURO );
        EURO=100;
        System.out.println("\n\nFinal del programa");

    } //cierra main
}
```

ARITMÉTICAS

RELACIONALES

LÓGICAS

+

-

\*

/

>

<

>=

<=

==

!=

AND

OR

NOT

Operadores

**OPERADORES EN JAVA**

# OPERADORES

Símbolo que indica al compilador que realice una operación matemática o lógica concreta.

## OPERADORES NUMÉRICOS

| Tipo            | Operador     | Precedencia                    | Operación realizada                  |
|-----------------|--------------|--------------------------------|--------------------------------------|
| Prefix, postfix | -- ++        | expr++ expr-- ++expr<br>--expr | Incremento/Decremento en una unidad. |
| Unarios         | + -          | + -                            | Cambio de signo                      |
| Multiplicativos | * / %        | * / %                          | Multiplicación, división y resto     |
| Aditivos        | + -          | + -                            | Suma, resta                          |
| De movimiento   | << >><br>>>> | << >> >>>                      | Desplazamiento a nivel de bits.      |

# OPERADORES PREFIX Y POSTFIX

## Operadores Incremento y Decremento

- El operador incremento suma 1 a su operando
  - $X=X+1$  es igual a  $X++$  y a  $++X$
- El operador decremento resta 1 a su operando
  - $X=X-1$  es igual a  $X--$  y a  $--X$

Aclaración sobre la utilización como prefijo o sufijo:

Ejemplo:

```
X=10;
```

```
Y=++X; //Al finalizar X tendrá valor 11 e Y valor 11.
```

```
X=10;
```

```
Y=X++; //Al finalizar X tendrá valor 11 e Y valor 10.
```

# OPERADORES RELACIONALES Y LÓGICOS

| OPERADOR RELACIONAL | Significado       |
|---------------------|-------------------|
| ==                  | Igual a           |
| !=                  | Distinto a        |
| >                   | Mayor que         |
| <                   | Menor que         |
| >=                  | Mayor o igual que |
| <=                  | Menor o igual que |

| OPERADOR LÓGICO | Significado            |
|-----------------|------------------------|
| &               | AND                    |
|                 | OR                     |
| ^               | XOR (OR exclusivo)     |
|                 | OR (de cortocircuito)  |
| &&              | AND (de cortocircuito) |
| !               | NOT                    |

# COMBINACIÓN DE OPERADORES PARA HACER ASIGNACIONES

| Operador        | Uso                     | Equivalente a                |
|-----------------|-------------------------|------------------------------|
| <code>+=</code> | <code>op1 += op2</code> | <code>op1 = op1 + op2</code> |
| <code>-=</code> | <code>op1 -= op2</code> | <code>op1 = op1 - op2</code> |
| <code>*=</code> | <code>op1 *= op2</code> | <code>op1 = op1 * op2</code> |
| <code>/=</code> | <code>op1 /= op2</code> | <code>op1 = op1 / op2</code> |
| <code>%=</code> | <code>op1 %= op2</code> | <code>op1 = op1 % op2</code> |

# COMBINACIÓN DE OPERADORES PARA HACER ASIGNACIONES

| Objetivo                   | Operador        | Ejemplos<br><code>int a = 6, b = 2;</code> | Resultados          |
|----------------------------|-----------------|--|---------------------|
| Sumar a y asignar          | <code>+=</code> | <code>a += b</code>                        | <code>a = 8</code>  |
| Restar de y asignar        | <code>-=</code> | <code>a -= b</code>                        | <code>a = 4</code>  |
| Multiplicar por y asignar  | <code>*=</code> | <code>a *= b</code>                        | <code>a = 12</code> |
| Dividir entre y asignar    | <code>/=</code> | <code>a /= b</code>                        | <code>a = 3</code>  |
| Obtener el resto y asignar | <code>%=</code> | <code>a %= b</code>                        | <code>a = 0</code>  |





```
private $host;  
private $username;  
private $password;  
private $database;  
private $charset;  
  
static private $link = null;  
  
public function Connect()  
{  
    self::$link = mysql_connect(self::$host, self::$username, self::$password);  
    if (!$link) {  
        throw new MySqlConnectionException("Cannot connect to MySQL database: " . mysql_error());  
    }  
    mysql_select_db(self::$database, $link);  
    mysql_set_charset(self::$charset, $link);  
}
```

# CÓDIGO EN JAVA. PRIMEROS PASOS

# ANATOMÍA DE UN PROGRAMA EN JAVA

```
public class <Nombre_Clase> {  
    public static void main(String[] args) {  
        .....  
        .....  
    }  
}
```

# COMPILAR Y EJECUTAR

- **Compilar un programa en Java:**

```
javac [ruta\]nombre_fichero.java
```

- **Ejecutar un fichero objeto:**

```
java nombre_fichero -> No escribimos la extensión
```

# PRIMEROS PASOS EN JAVA

- DECLARACIÓN DE VARIABLES

**<tipo de variable> <lista nombres variables> ;**

- INICIALIZAR VARIABLES:

**<variable>= <valor inicial>;**

- INCLUIR COMENTARIOS:

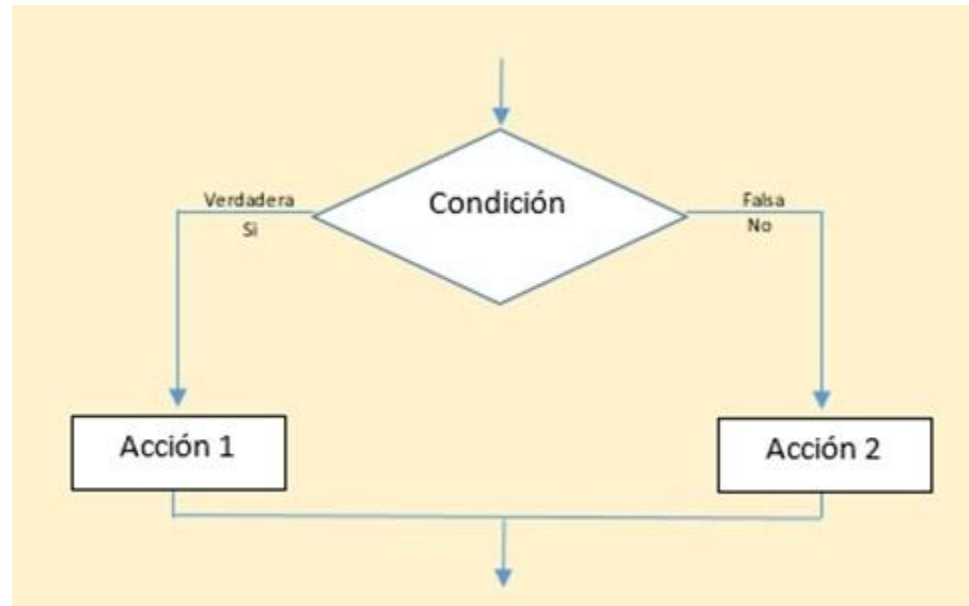
**// comentario de una sola línea**

**/\* Comentario de varias líneas**

**\* Línea 1**

**\* Línea 2**

**\*/**



# ESTRUCTURAS ALTERNATIVAS

# ALTERNATIVA SIMPLE

## CONDICIONAL SIMPLE

```
if ( condición )  
    sentencia;
```

```
if (condición ) {  
    sentencia 1;  
    sentencia 2;  
}
```

(\*)Si tan solo tenemos una sentencia podemos omitir las llaves.

# ALTERNATIVA DOBLE

```
if ( condición )  
    sentencia;  
else  
    sentencia;
```

(\*) Si tenemos más de una sentencia en cada una de las partes podremos las llaves.

(\*) Si tan solo tenemos una sentencia podemos omitir las llaves.

# Ejemplo: Alternativas anidadas

```
if (i == 100) {  
    if (j < 20) a=b;  
    if (k > 100)  
        c=d;  
    else  
        a = c : // esta cláusula else hace referencia a if (k > 100)  
}  
else  
    a = d; // esta cláusula else hace referencia a if (i == 10)
```



# Ejemplo: Alternativas anidadas

**P\_3\_1\_11 ORDENAR 3 NÚMERO INTRODUCIDOS POR TECLADO**

# ALTERNATIVA MÚLTIPLE

```
switch(variable o expresión ) {  
    case constante 1:  
        sentencias;  
        break;  
    case constante 2:  
        sentencias;  
        break;  
    .....  
    default:  
        sentencias;  
}
```

En función del valor de la condición del switch, se ejecutara un case en concreto.

**El valor debe ser entero o de tipo carácter o String desde v7**

# ALTERNATIVA MÚLTIPLE

```
switch(i) {  
    case 1:  
    case 2:  
    case 3:  
        sentencias;  
        break;  
    case 4:  
        sentencias;  
        break  
}
```

(\*) Los switch se pueden anidar y en ese caso las constantes case pueden contener valores comunes

# EJEMPLO ALTERNATIVA MÚLTIPLE

```
class UsoSwitch {  
public static void main(String args[]) {  
    int i;  
    for ( i=0; i<10; i++)  
        switch(i) {  
            case 0:  
                System.out.println("El valor es cero");  
                break;  
            case 1:  
                System.out.println("El valor es uno");  
                break;  
            case 2:  
                System.out.println("El valor es dos");  
                break;  
            case 3:  
                System.out.println("El valor es tres");  
                break;  
            default :  
                System.out.println("El valor es cuatro o más");  
                break;  
        }  
    }  
}
```

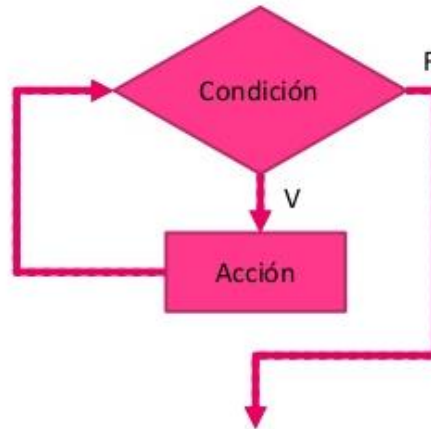
# EJEMPLO ALTERNATIVA MÚLTIPLE

*// Programa Java para demostrar switch case*

*// con tipo de datos primitivos (int)*

```
public class Test {  
    public static void main(String[] args){  
        int day = 5;  
        String dayString;  
        // instrucción switch con tipo de datos int  
        switch (day) {  
            case 1: dayString = "Lunes";  
                break;  
            case 2: dayString = "Martes";  
                break;  
            case 3: dayString = "Miercoles";  
                break;  
            case 4: dayString = "Jueves";  
                break;  
            case 5: dayString = "Viernes";  
                break;  
            case 6: dayString = "Sabado";  
                break;  
            case 7: dayString = "Domingo";  
                break;  
            default: dayString = "Dia inválido";  
                break;  
        }  
        System.out.println(dayString);  
    }  
}
```

## ESTRUCTURA REPETITIVA MIENTRAS (WHILE)



# ESTRUCTURAS ITERATIVAS

# BUCLE PARA (FOR)

➤ Para una sola instrucción:  
**for (inicialización; condición; iteración) instrucción;**

➤ Para repetir un bloque:  
**for(inicialización; condición; iteración) {**  
    **instrucción1;**  
    **instrucción2;**  
    .....  
**}**

**Inicialización:** instrucción de asignación que establece el valor inicial de la variable de control que sirve como contador.

**Condición:** instrucción booleana que determina que se repite el bucle si su resultado es TRUE

**Iteración:** cantidad que cambia la variable de control del bucle en cada repetición.

## EJEMPLOS BUCLE PARA (FOR)

```
for (int i=0; i<10; i++)
```

```
for(int i=0; i<10; i=i+2) ) {
```

```
.....
```

```
}
```

```
for(int i=0; i<10; i-=1) ) {
```

```
.....
```

```
}
```



# BUCLE MIENTRAS (WHILE)

Se utiliza si sabemos de antemano cuantas veces queremos repetir la secuencia de instrucciones

Para una sola instrucción:

```
while(condición) instrucción;
```

Para repetir un bloque:

```
while (condición) {
```

```
    instrucción1;
```

```
    instrucción2;
```

```
    .....
```

```
}
```

# BUCLE MIENTRAS (WHILE)

```
public class WhileAlfabeto {  
    public static void main(String[] args) {  
        char ch;  
        ch='a';  
        while (ch<= 'z') {  
            System.out.print(ch+"\t");  
            ch++;  
        }  
    }  
}
```

# BUCLE MIENTRAS (DO WHILE)

```
do {  
    instrucción1;  
    instrucción2;  
    .....  
} while (condición);
```

# EJEMPLO BUCLE MIENTRAS ( DO WHILE)

Ejemplo: Utilizar do para repetir la petición de un carácter hasta que el usuario escriba 'S'


```
public class HastaEscribirQ {  
  
    public static void main(String[] args) throws java.io.IOException {  
        char letra;  
        do {  
            System.out.println("Escribe una letra y pulsa ENTER (PARA SALIR PULSA S");  
            letra=(char) System.in.read();  
        }while (letra!='S');  
    }  
}
```

# USO DEL BREAK

Instrucción que fuerza la salida inmediata de un bucle.

Tener en cuenta:

- Sólo afecta al bloque del que forma parte, no a otro posible bucle externo contenedor.
  - Debe evitarse su uso indebido.
  - Su uso en un switch solo provoca la salida del caso no del un posible bucle que contenga al switch.
- 
- Uso de break como GOTO Y ETIQUETAS



```
private $host;  
private $username;  
private $password;  
private $database;  
private $charset;  
  
static private $link = null;  
  
public function Connect()  
{  
    self::$link = mysql_connect(self::$host, self::$username, self::$password);  
    if (!$link) {  
        throw new MySqlConnectionException("Cannot connect to MySQL database: " . mysql_error());  
    }  
    mysql_select_db(self::$database, $link);  
    mysql_set_charset(self::$charset, $link);  
}
```

CÓDIGO EN JAVA. ENTRADA Y SALIDA DE DATOS

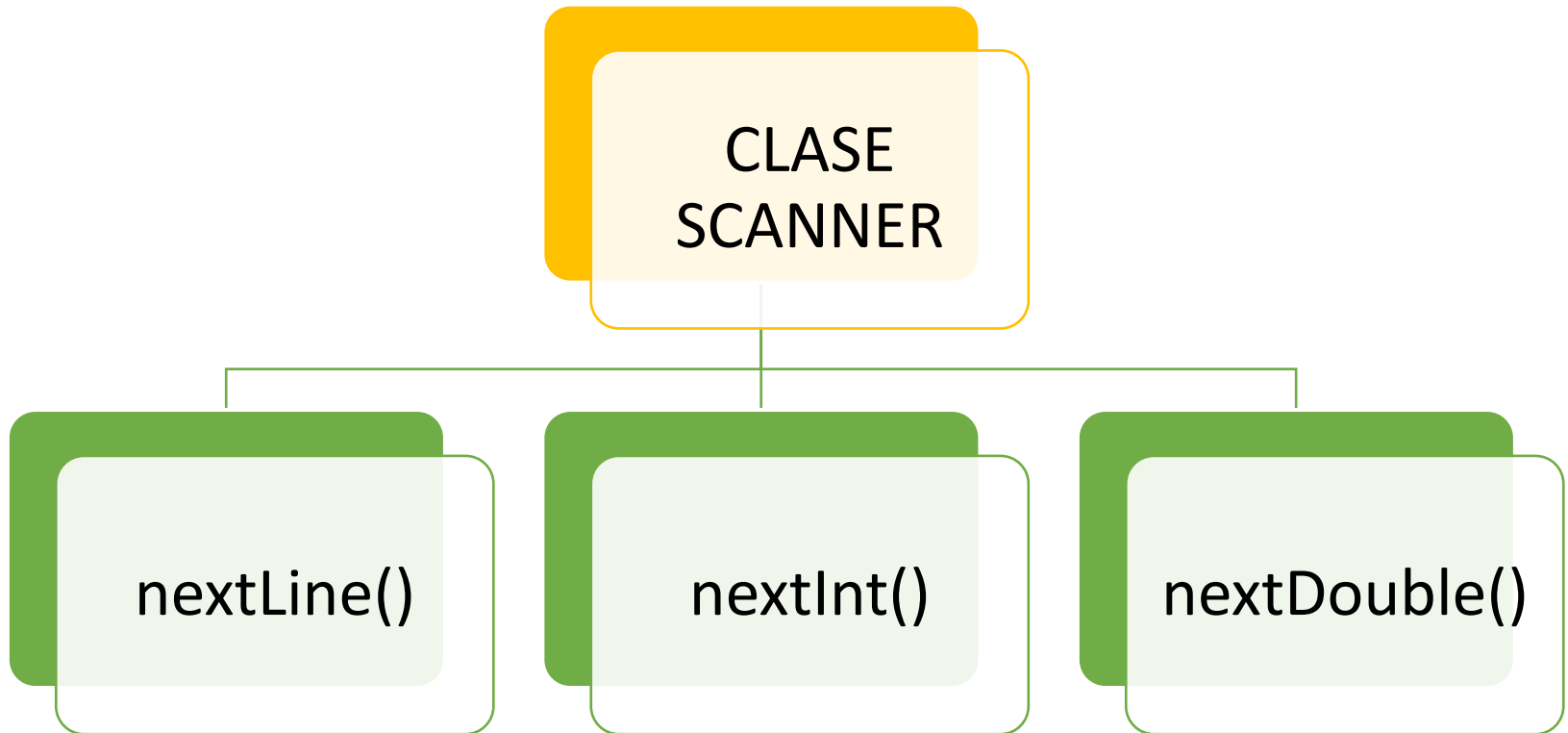
# MOSTRAR DATOS Y MENSAJES

**System.out.print(.....);**

**System.out.println(.....);**

**//con salto de línea**

# CAPTURAR DATOS Y MENSAJES





# CAPTURAR DATOS Y MENSAJES

## CLASE SCANNER

❑ Pertenece al paquete `java.util.Scanner`

❑ Crear objeto de la clase Scanner:

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

❑ Utilizar métodos de la clase Scanner para pedir el tipo de dato que nos interese:

- `next`: pedir un String (1 token)
- `nextLine` : pedir un String(hasta \n)
- `nextDouble()` : Dato numérico de doble precisión
- `nextFloat()` : Dato numérico de simple precisión
- `nextInt()` : Entrada de dato numérico de tipo entero
- `nextBoolean()` : Entrada de dato lógico. (true/false)

# CAPTURAR DATOS Y MENSAJES

## System.in.read()

❑ Introducción de un carácter por teclado

```
char <variable>;
```

```
<variable> = (char) System.in.read();
```

(\*) Debes añadir **throws IOException** a continuación de la línea del método main e importar el paquete **import java.io.IOException;**