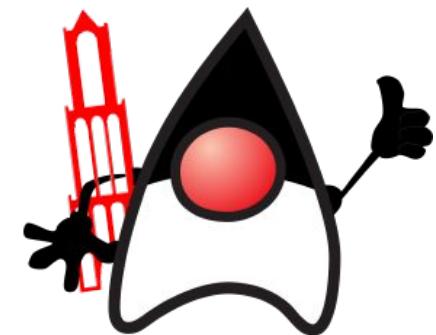
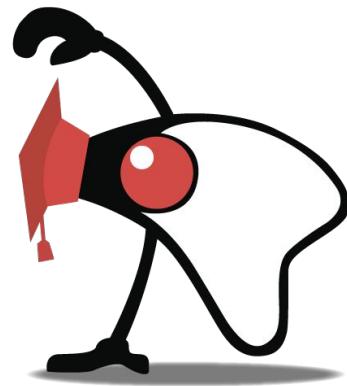
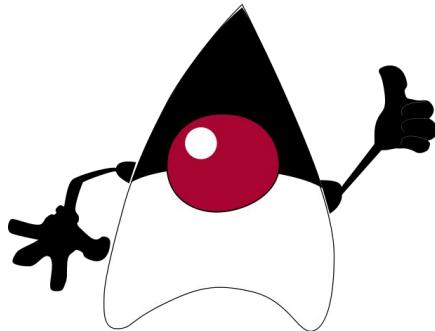


Tecnicatura en Programación

# Programación I



# Condicional if – else

## (Toma de decisiones o sentencia)

### Tres formas de la sentencia if:

Existen tres (3) formas básicas de una sentencia if :

“**if**”, se utiliza cuando se quiere hacer una cosa o no hacer nada.

“**if, else**”, se utiliza cuando se quiere hacer una u otra cosa.

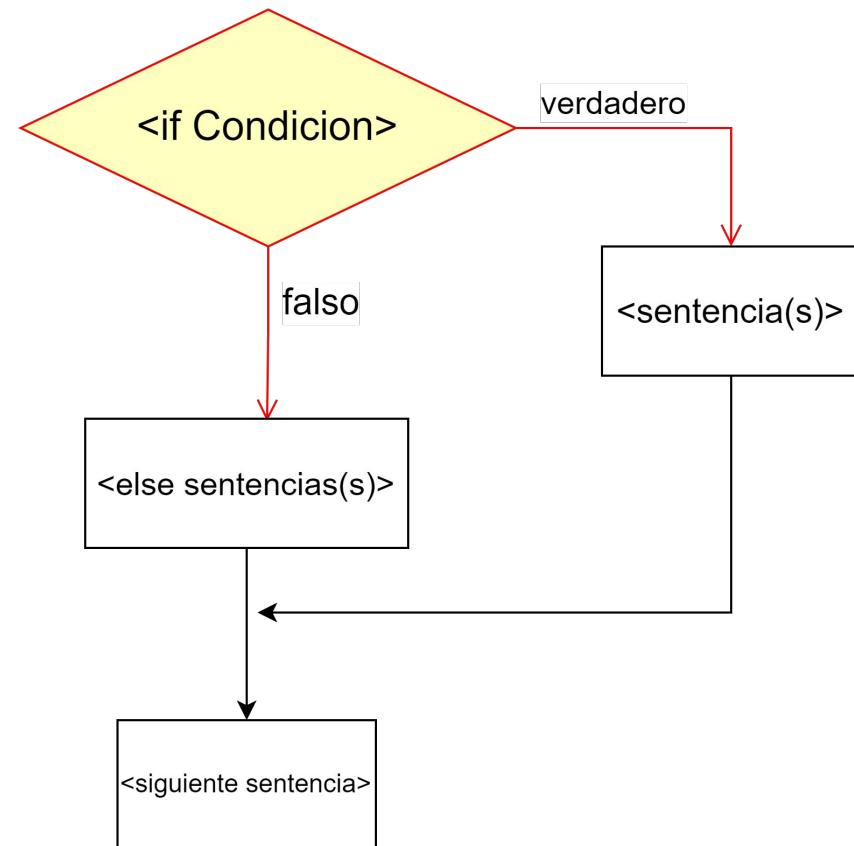
“**if, else if**”, se utiliza cuando hay tres o más posibilidades.

La toma de decisión se resuelve con la sentencia (if-else) su formato es:

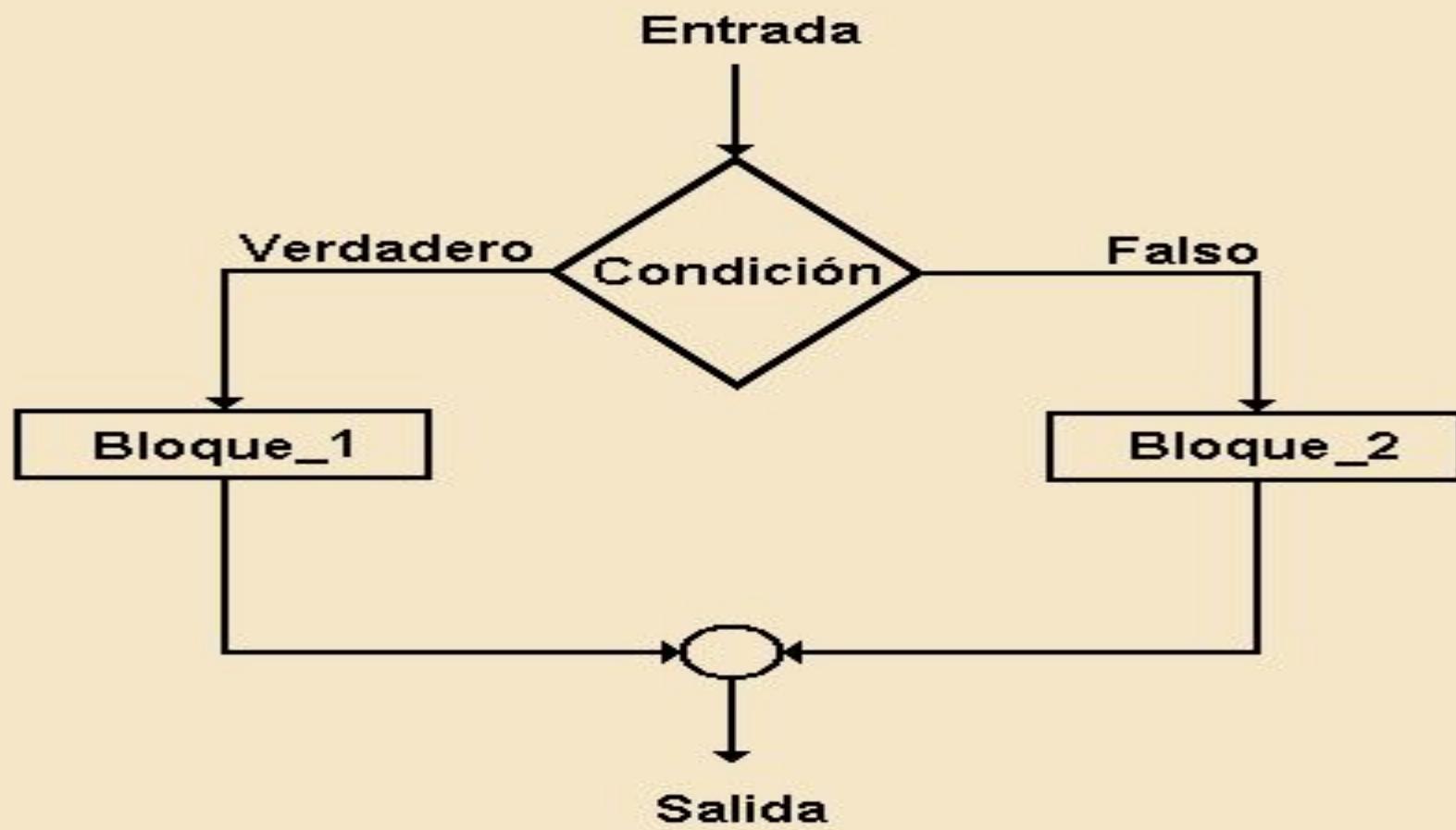
```
//Si condición es verdadera, se cumple A de lo contrario B
if(condición) {
    sentencia A;
} else{
    Sentencia B;
}
}
```

# Sentencia if-else

```
if (condition) {  
    // bloque de código a ejecutar  
    // si la condición es verdadera  
} else {  
    // bloque de código a ejecutar  
    // si la condición es falsa  
}
```



# Diagrama if-else tallarin



# Diagrama if-else Nassi Chapin

Representación de la estructura de control condicional

Algoritmo

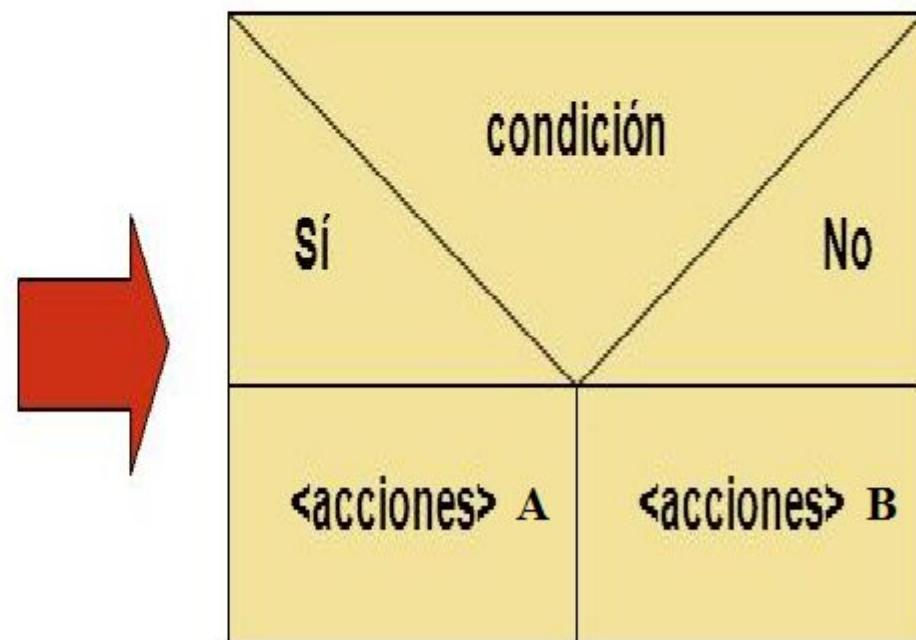
**si** <condición> **entonces**

<acción 1> A

**si\_no** (de lo contrario)

<acción 2> B

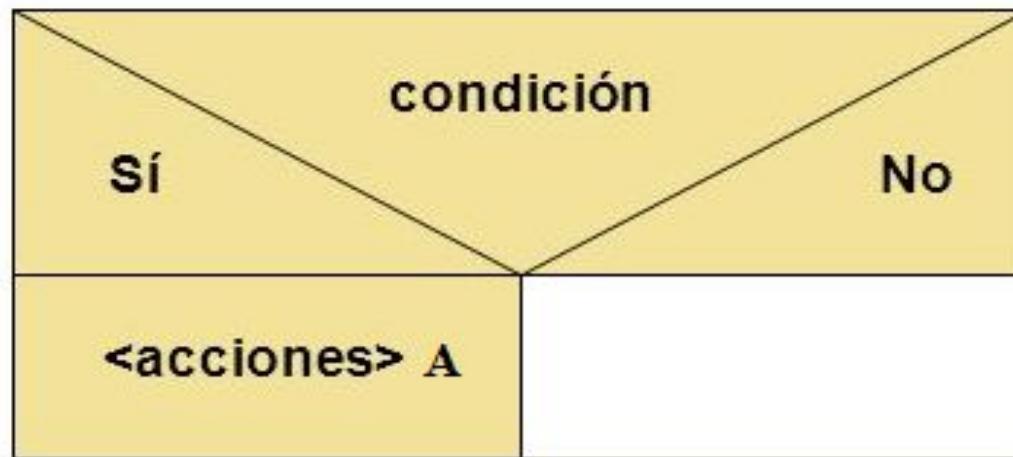
Diagrama N-S



# Ejemplo de sentencia if-else no anidada

(el punto y coma en el else cierra el bloque indicando sentencia nula)

```
if (condicion)
    sentencia A;
```

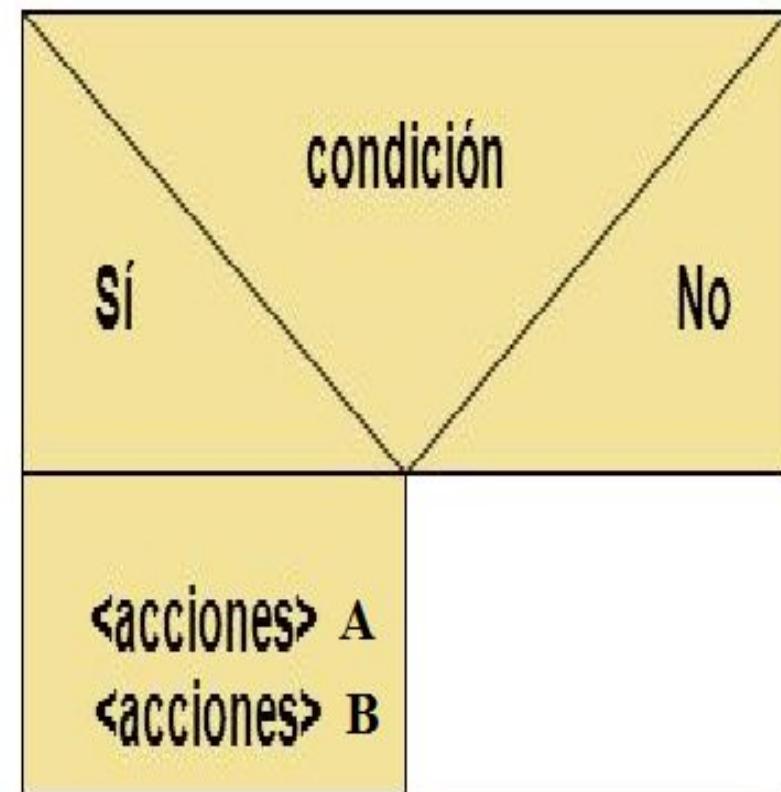


Otra variante del caso anterior seria:

```
if (condicion)
    sentencia A;
else;
```

# Bloque de sentencias múltiples en el “si”. (La estructura necesita de la presencia de llaves)

```
if (condicion){  
    sentencia A;  
    sentencia B;  
}
```



# Condición

La condición evaluada tanto en la forma de las decisiones como en las iteraciones, son expresiones que deben arrojar como resultado, verdadero o falso. Debido a esto, reciben la denominación de condiciones Booleanas(George Boole, matemático inglés, desarrolló un álgebra proposicional cuya expresión más simple maneja solo estados verdadero y falso)

Es posible que la expresión utilizada en la condición, ya sea por error o intencionalmente, no tenga “la forma” de una condición Booleana. De todas maneras el procesador evaluará dicha expresión en términos lógicos obteniendo como resultado, verdadero o falso.

Condición: si **a<b** se cumple la condición en la sentencia A .

Ej:

```
if (condicion){ //la condicion puede ser True o False “condicion simple”.
    Sentencia A;
}
```

# Operadores Relacionales

Los operadores relacionales son aquellos que permiten determinar la relación entre dos variables o valores. estos son:

(> ; < ; >= ; <= ; == ; !=) Todos ellos son operadores binarios. Por lo tanto, no es posible realizar una comparación entre más de 2 valores utilizando solamente operadores relacionales.

El resultado de una operación relacional es un valor booleano, es decir, verdadero o falso.

Si se asigna a una variables entera el resultado de una operación relacional, los valores asignados seran “0” o “1”, siendo equivalentes a (falso) y (verdadero) respectivamente.

Ejemplo:

Int F;

F = 4>2; /\*F toma el valor de 1 \*/

F = 4 != 2; /\*F toma el valor de 0 \*/

F= !(4>2); /\*inversor lógico (4>2) verdadero al colocar (!) lo invierto a falso o F=(0).

F= !A ; /\*si “A” es falso el negarlo “! A” será verdadero y si “A” es verdadero “! A” será falso.

En algunos lenguajes se considera que todo valor numérico distinto de cero es **VERDADERO**, mientras que cero representa **FALSO**.

# Condición compuesta, null, simple.

**condicionales**: en una decisión se ingresa a evaluar la condición.

Si se cumple se ejecuta la sentencia(que puede ser **simple**, **nula** o **compuesta**),y luego vuelve a evaluarse la condición si es requerida.

Por lo tanto, si la condición se cumple se ejecutará la sentencia.

**Condición compuesta**: cuando una única expresión booleana está formada por más de una condición, se dice que es una condición compuesta.

Para construirla se recurre a la utilización de los operadores lógicos los cuales son:

Operador tipo monario: (!) su función es de inversor lógico.

Operador tipo binario: (&&) su función es un AND.

Operador tipo binario: (||) su función es un OR.

**Condición null o vacía**: es cuando una condición no tiene ningún valor asignado, sirve en caso que usted no halla entrado un valor puede colocar una función que le advierta “try catch”.

**Condición simple**: es aquella que cumple una condición .

# EJ Condición Null o vacia

```
public class OpNull{  
  
    public static void main(String args[]){  
  
        promptUser();  
  
    }  
  
    private static void promptUser(){//metodo que ejecuta si el valor entrado es null o vacio  
  
        String name = "pablo"; //si dejo las “ ” sin argumento dira no entro su nombre (no colocar null)  
  
        if(name == null || name.trim().isEmpty()){ //funcion trim().isEmpty()  
  
            System.out.println("No entro su nombre");  
        }else{  
  
            System.out.println("Entro su nombre: " + name);  
        }  
    }  
}
```

# EJ sentencia (If – else) edad

```
public class Edad { //true o verdadera y false o falso que no se cumplio
    public static void main (String [] args){
        int edad = 15; //aqui ingresamos la edad
        if (edad <= 18)//si condicion es (true) pasa a sentencia "A" si es (false) pasara a "B".
            System.out.println ("Eres menor de 18 años");//sentencia A
        }else{
            System.out.println ("Eres mayor de 18 años");//sentencia B
        }
    }
}
```

# Uso de Condiciones en toma de Decisiones

## USO DE CONDICIONES EN TOMA DE DECISIONES

Decision verdadera o falsa

Toma de Decisiones:

```
if( condicion ){
    si es "true" sentencia A;
}else{
    si es "false" sentencia B;
}
```

(num%2) el operador resto (%)  
dara el resto de la operacion si  
es uno (1) o si es cero (0).

**Nota:** una condicion es true o verdadera cuando lo que ingresas "num" y su operacion "num%2" es igual a lo que se espera, que en este caso es igual a cero "0".

```
num = 8; //puedo agregar otros valores a comparar
if((num % 2) == 0){ // (0 == 0) es "true" se cumple la condicion
```

si es "true" sentencia A;

}else{

si es "false" sentencia B;

}

$$\begin{array}{r} 8 \longdiv{2} \\ 8 \quad 4 \\ \hline 0 \end{array}$$

resto cero

**Nota:** una condicion es true o verdadera cuando lo que ingresas "num" y su operacion "num%2" es igual a lo que se espera, lo que se espera que sea distinto de cero y efectivamente es (1) es distinto de cero (0) cumpliendo la condicion.

```
num = 9; //puedo agregar otros valores a comparar
if((num % 2) != 0){ // (1 != 0) es "true" se cumple la condicion
```

si es "true" sentencia A;

}else{

si es "false" sentencia B;

}

$$\begin{array}{r} 9 \longdiv{2} \\ 8 \quad 4 \\ \hline 1 \end{array}$$

resto uno

# Sentencia (if-else) número par / impar

//El programa que calcula si un número entero es par o impar es el siguiente:  
//los numeros pares son los multiplos del numero 2.

```
import java.util.*;  
  
public class ParImpar{  
  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        int Numero;  
        System.out.print("Introduzca Número entero: ");  
        Numero = sc.nextInt(); //aqui se introduce el valor tipo entero  
  
        if(Numero%2==0){ //numeros pares son divisibles por 2 y resto cero  
            System.out.println("Par");  
        }else{  
            System.out.println("Impar");  
        }  
    }  
}
```

# EJ. Sentencia if-else Operador Not (!) o inversor

```
import java.util.Scanner;

public class RestoInvert{

    public static void main(String args[]){

        int num, div;

        Scanner teclado = new Scanner(System.in);

        num = teclado.nextInt();
        div = teclado.nextInt();

        If( ! ((num%div) != 0) ){ //operador not (!) o inversor y operador distinto (!=)

            System.out.printf("\n\n%d es divisible por %d ",num, div);

        }else{

            System.out.printf("\n\n%d no es divisible por %d ",num, div);

        }

    }
}
```

**En Agilent se hace un 30% de descuento a los clientes cuya compra supere los \$300  
lo que dice es que Agilent te dará un descuento de 30% si tu compra supera \$300  
¿Cuál será la cantidad que pagará una persona por su compra?**

```
public class Compra {  
  
    public static void main(String[] args) {  
  
        double compra, descuento, total = 0;  
        compra = 410;  
  
        descuento = compra*0.2;  
        total = compra;  
  
        if (compra>300) {  
  
            total = compra - descuento;  
  
            System.out.println("El descuento es de: "+descuento);  
            System.out.println("El total a pagar es: "+total);  
        } else{  
            System.out.println("Sin descuentos, el total es:"+total);  
        }  
    }  
}
```

# Conversion decimal a porcentaje

para convertir decimales en porcentajes, sólo multiplica el decimal por 100, pero recuerda poner el signo "%" para que sepamos que es por 100.

Ej: el decimal 0.3 a porcentaje es  $0.3 \times 100 = 30\%$  lo contrario seria dividir  $30/100 = 0.3$

La manera más fácil de multiplicar por 100 es mover el punto decimal 2 posiciones a la derecha:

De decimal	a porcentaje	
0,125	0,125 2 posiciones	12,5% muestra el punto decimal <b>2 posiciones a la derecha</b> , y añade el signo de "%".

Ejemplo: Convertir 0,65 en porcentaje

Mueve el punto decimal dos posiciones: 0,65 -> 6,5 -> 65.

La respuesta es **0,65 = 65%**

# Expresar Decimales, fracciones y porcentaje

## Decimales, fracciones y porcentajes

**Decimales, fracciones y porcentajes son diferentes maneras de escribir el mismo valor:**



**La mitad** se puede escribir...

Como fracción:

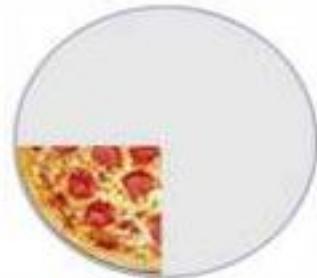
$\frac{1}{2}$

Como decimal:

0,5

Como porcentaje:

50%



**Un cuarto** se puede escribir...

Como fracción:

$\frac{1}{4}$

Como decimal:

0,25

Como porcentaje:

25%

# Promedio

El promedio es un valor "central" calculado entre un conjunto de números.

Es fácil de calcular: suma todos los números y divide por la cantidad de números que hay, y se obtiene el promedio.

Podemos sacar promedio de notas, de población, etc.

$$\text{Promedio} = a_1 + a_2 + a_3 / n$$

Donde ( $a_1$ ,  $a_2$  y  $a_3$ ) son valores y ( $n$ ) es el número de valores a sumar.

**Ejemplo:** el promedio de 4, 6 y 11 es  $(4+6+11)/3 = 21/3 = 7$

# Condición simple con “if – else” datos int

```
public class Promedio{//promedia un entero (6)

    public static void main(String args[]){

        int num1 =5;
        int num2 = 6;
        int num3 = 7;
        int promedio = 0; //el promedio sera un entero por su declaracion

        promedio = (num1+num2+num3)/3;//no esta casteado el promedio

        if(promedio >= 4){

            System.out.println("El alumno aprobo: "+promedio);

        } else {

            System.out.println("El alumno reprobo: "+promedio);

        }
    }
}
```

# Sin castear int a double

```
Public class Promedio{//promedia un valor con punto (6.0)

    Public static void main(String args[]){

        Int num1 =5;
        Int num2 = 6;
        Int num3 = 7;

        double promedio; //el promedio sera un numero real con parte
fraccionaria cero

        promedio =  (num1+num2+num3)/3;//casteo de entero a double

        If(promedio >= 4){

            System.out.println("El alumno aprobó: "+promedio);

        } else {

            System.out.println("El alumno reprobó: "+promedio);

        }
    }
}
```

# Número Real con parte fraccionaria

**Ejemplos:** número **real** menos la parte **entera** de la parte **fraccionaria**, puedo tener un número real con parte **fraccionaria cero** ejemplo “6.0” donde la parte **entera** es “6” y la **fraccionaria** es “0”.

Si  $x = 3.14$ , entonces la parte fraccionaria de  $x$  es:  $\{3.14\} = 3.14 - [3.14] = 3.14 - 3 = 0.14$ .

Si  $x = 5$ , entonces la parte fraccionaria de  $x$  es:  $\{5\} = 5 - [5] = 5 - 5 = 0$ .

Número **entero**: 6

Número **real** con parte **fraccionaria** cero: 6.0

Número **real** con parte **fraccionaria** distinta de cero: 6.5

# Convertir decimal a fraccion

Para convertir un Decimal a una Fracción sigue estos pasos:

Paso 1: Escribe el decimal dividido por 1.

Paso 2: Multiplica los números de arriba y abajo por 10 una vez por cada número luego de la coma.  
(Por ejemplo, si hay dos números luego del decimal, multiplicarlos por 100, si hay tres usa el 1000, etc.)

Paso 3: Simplifica (reduce) la fracción

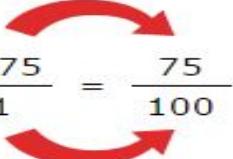
# Ejemplo de Conversión decimal a fraccion

## Ejemplo 1: Expresar 0,75 como fracción

Paso 1: Escribe:

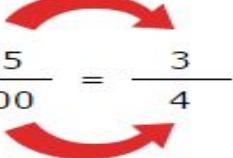
$$\frac{0,75}{1}$$

Paso 2: Multiplica el numero de abajo y el de arriba por 100 (porque hay 2 dígitos luego de la coma):

$$\frac{0,75}{1} \times 100 = \frac{75}{100}$$


(¿Ves como el número de arriba se convierte en un entero?)

Paso 3: Simplifica la fracción:

$$\frac{75}{100} \div 25 = \frac{3}{4}$$


Respuesta = 3/4

Nota: 75/100 se llama una **fracción decimal** y 3/4 es llamada una **fracción común** !

# Castear int a double

```
Public class Promedio{  
  
    Public static void main(String args[]){  
  
        Int num1 =5;  
        Int num2 = 6;  
        Int num3 = 7;  
        double promedio;  
  
        promedio = (double) (num1+num2+num3)/3;//casteo de entero a double  
  
        If(promedio >= 4){  
  
            System.out.println("El alumno aprobo: "+promedio);  
  
        } else {  
  
            System.out.println("El alumno reprobo: "+promedio);  
  
        }  
    }  
}
```

# Castear float a int

```
public class FloatInteger{  
  
    public static void main(String args[]){  
  
        float num = 22.20f;  
  
        int cast = (int) num;  
  
        System.out.println(cast);  
    }  
}
```

# Castear double a float con formato (%)

```
public class DoubleFloatC{//estilo lenguaje c++\n\n    public static void main(String args[]){\n\n        double num = 2.2;//double num = 2.2f; es float, ok\n                    //float num = 2.2; es double, ilegal\n                    //float tiene 32bits y double 64 bits no hay problema\n                    //de double(64) a float(32) si hay diferencia en tamaño\n\n        System.out.printf("Numero: %f \n",num);// seria: double = 2.2f;\n    }\n}
```

Un obrero necesita calcular su salario semanal, el cual se obtiene de la siguiente manera:

Si trabaja 40 horas o menos se le paga \$16 por hora.

Si trabaja más de 40 horas se le paga \$16 por cada una de las primeras 40 horas y \$20 por cada hora extra.

```
public class Salario {  
  
    public static void main(String[] args) {  
  
        int horasTrabajadas, horasExtras, salarioSemanal;  
  
        horasTrabajadas = 77;  
  
        if ( horasTrabajadas > 40 ) { //solo debe trabajar 40 horas mas de eso es extra  
  
            horasExtras = horasTrabajadas - 40;  
            salarioSemanal = horasExtras * 20 + 40 * 16; // 7*20 + 40*16 = 140 + 640 = 780  
  
            System.out.println("El salario con horas extras es de: "+salarioSemanal);  
  
        }else{  
  
            salarioSemanal = horasTrabajadas * 16;  
            System.out.println("Su sueldo es de " + salarioSemanal);  
        }  
    }  
}
```

# Orden de evaluación (operadores matemáticos)

En matemáticas y programación (**informática**), el orden de operaciones aclara de forma inequívoca los procedimientos a realizar en el cálculo de una determinada expresión matemática.

Por **ejemplo**, en matemáticas y en la mayoría de los lenguajes de programación, la operación de **multiplicación** tiene preferencia a la de **adición**; por ejemplo en la expresión  $2 + 3 \times 4$  la respuesta algebraica es:  $2 + 12 = 14$ . Los paréntesis o corchetes, se pueden utilizar para evitar confusiones, por lo que la expresión anterior también puede ser escrita como  $2 + (3 \times 4)$ .

Desde la introducción de la notación algebraica moderna la multiplicación tiene precedencia sobre la suma, cualquiera que sea el lado del número donde aparezca. Por lo tanto  $3 + 4 \times 5 = 4 \times 5 + 3 = 23$ . Los exponentes tienen precedencia sobre las sumas y multiplicaciones, y tendrían que ser colocados únicamente como superíndice a la derecha de su base. Para cambiar el orden de las operaciones, se utiliza paréntesis. Por lo tanto, para forzar la adición sobre la multiplicación, se escribe  $(2 + 3) \times 4 = 20$ , y para forzar la adición sobre la exponenciación, se escribe  $(3 + 5)^2 = 64$ .

# Reglas para Orden de Operaciones

## **Reglas para Orden de Operaciones**

1. Resolver paréntesis, u otros símbolos. ( ) [ ] { }
2. Resolver exponentes y raíces.
3. Multiplicación y división de izquierda a derecha.
4. Suma y resta de izquierda a derecha.

Ejemplo:

$$\begin{aligned} 2 + 7 \cdot 8 / 2 & \\ 2 + 56 / 2 & \quad [\text{Se multiplicó } 7 \cdot 8] \\ 2 + 28 & \quad [\text{Se dividió } 56 / 2] \\ 30 & \quad [\text{Se sumó } 28 + 2] \end{aligned}$$

Ejemplo 1:

$$2 [ 6 - (9 / 3) + 8 ]$$

Como el paréntesis está dentro del corchete, hay que resolver éste para luego resolver el corchete.

$$\begin{aligned} 2 [ 6 - (9 / 3) + 8 ] & \\ 2 [ 6 - 3 + 8 ] & \\ 2 [ 3 + 8 ] & \\ 2 [ 11 ] = 22 & \end{aligned}$$

# Orden operaciones

Cuando hay un paréntesis ( ), llave { } y corchete [ ], hay que resolver lo que está dentro de estos símbolos, antes de efectuar alguna otra operación.

Ejemplo:

$$\begin{array}{ll} 5 \cdot (9 - 6) + 8 & \text{<Se resuelve el paréntesis>} \\ 5 \cdot 3 + 8 & \text{< Se restó } 9 - 6 = 3 \text{>} \\ 15 + 8 & \text{< Se multiplicó } 5 \cdot 3 \text{>} \\ 23 & \text{< Se sumó } 15 + 8 \text{>} \end{array}$$

Otro ejemplo:

$$\begin{array}{ll} 2 [ 6 \cdot (-1)] + 8 / 2 & \text{<Primero, se resuelve el [ ] >} \\ 2 [ -6] + 8 / 2 & \text{< Se multiplicó } 6 \cdot -1 \text{>} \\ -12 + 8 / 2 & \text{< Se multiplicó } 2 \cdot -6 \text{>} \\ -12 + 4 & \text{< Se dividió } 8 / 2 \text{>} \\ -8 & \text{< Se sumó } -12 + 4 \text{>} \end{array}$$

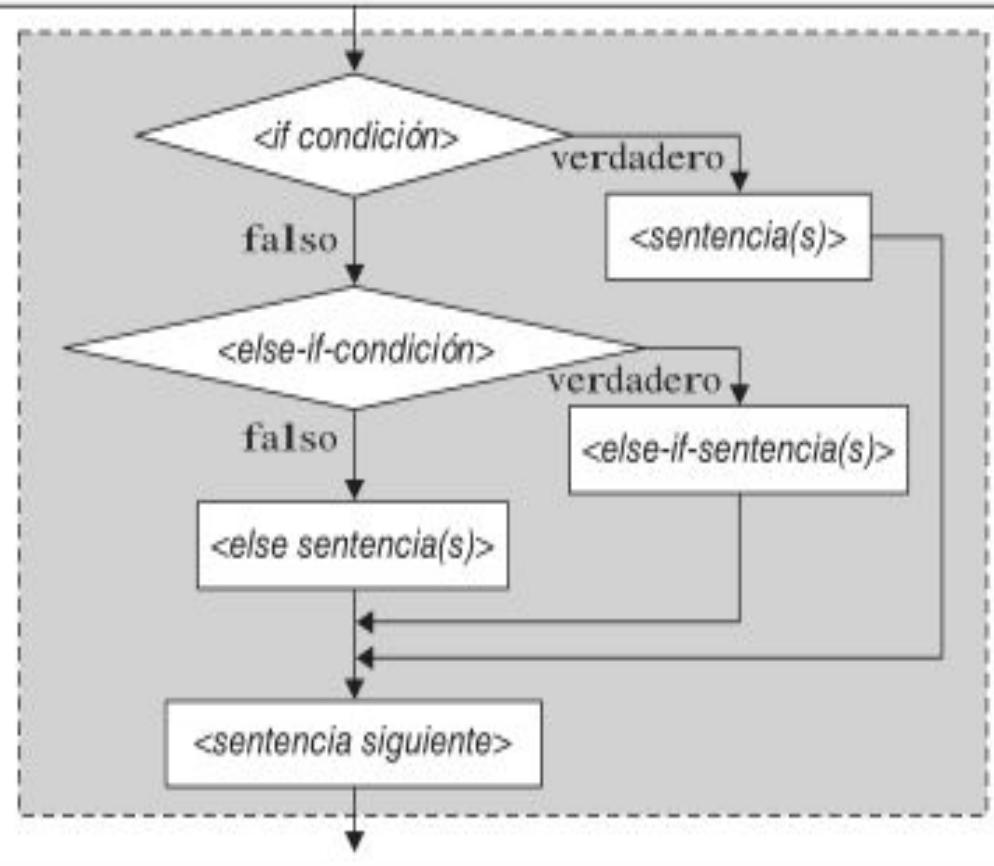
Cuando hay una combinación de paréntesis, corchetes y llaves, hay que resolver éstos de adentro hacia fuera.

# Decisiones anidadas if-else

```
if (<if-condición>)
{
  <if-sentencia(s)>
}
else if (<else-if-condición>)
{
  <else-if-sentencia(s)>
}
.
.
.
else
{
  <else-sentencia(s)>
}
```

else if adicionales

opcional



Sintaxis y semántica para la forma "if, else if" de la sentencia **if**.

# Decisiones anidadas

```
Public class Anidado{

    Int scanner = 1; //introducir manualmente valores de (1 a 4)una sola entrada
    Int num1 = 8;
    Int num2 = 4;
    Int resultado = 0;

    //operaciones matematicas a eleccion

    If( scanner== 1){

        resultado = num1 + num2;
        System.out.println("El resultado de la suma es: "+resultado);

    }else if(scanner == 2){

        resultado = num1 - num2;
        System.out.println("El resultado de la resta es: "+resultado);

    }else if(scanner == 3){

        resultado = num1 * num2;
        System.out.println("El resultado de la multiplicacion es: "+resultado);

    }else if(scanner == 4){

        resultado = num1 / num2;
        System.out.println("El resultado de la division es: "+resultado);

    }

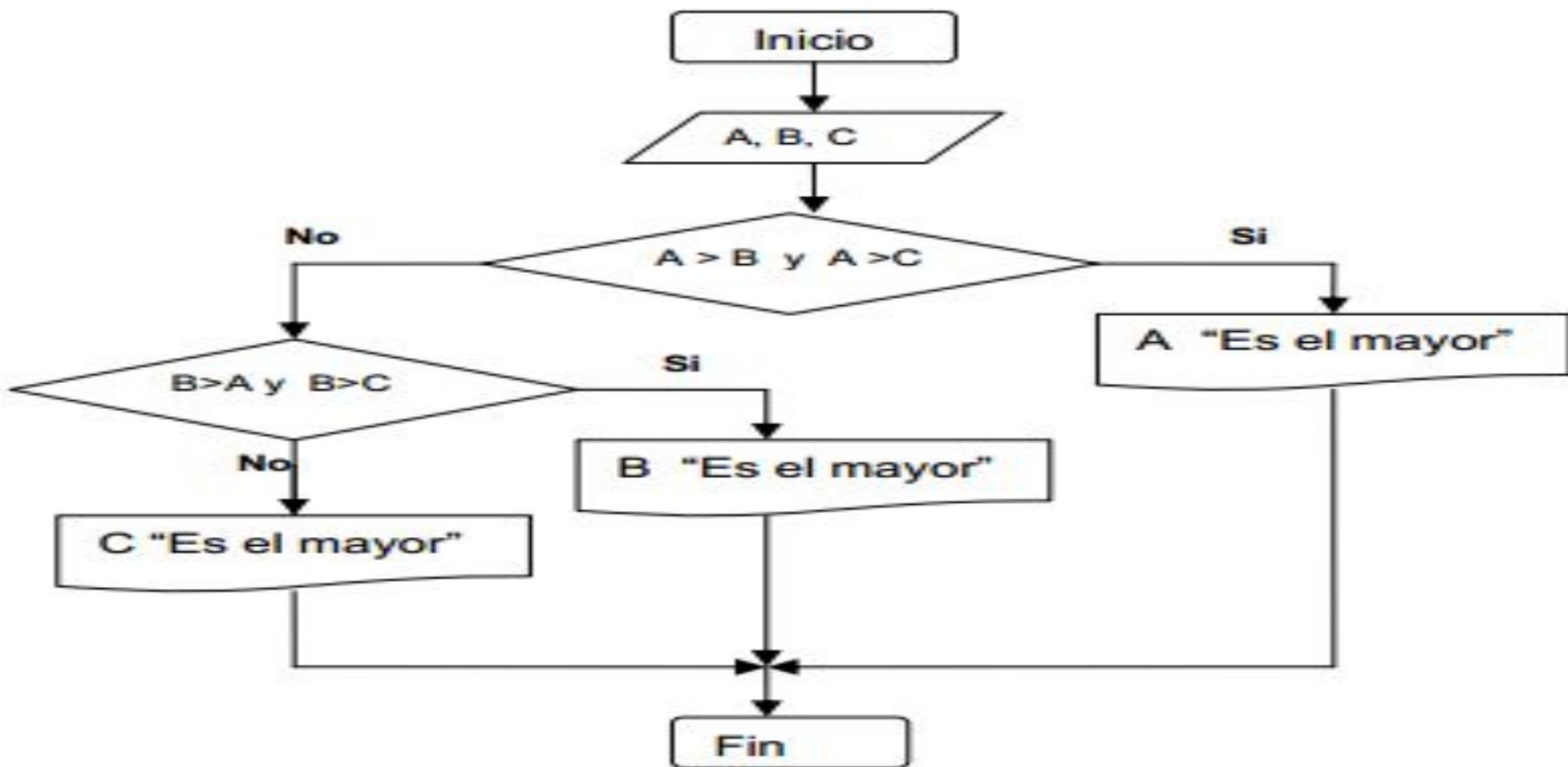
}
```

# Escalonador

(Decisiones anidadas y sucesivas en la que cada nivel de decisión se encuentra en la salida por “no” del nivel anterior)

```
//Se desea clasificar a los socios de un club por edades de la siguiente manera:  
//infantil (menor a 14), cadete(14 y 20), activo(21 y 59),  
//senior(60 o mas)  
  
public class Escalonador {  
  
    public static void main(String args[]) {  
  
        int edad = 34; //una sola entrada a comparar  
  
        if (edad > 59) {  
  
            System.out.println("socio senior: " + edad);  
  
        } else if (edad > 20) {  
  
            System.out.println("socio activo: " + edad);  
  
        } else if (edad > 13) {  
  
            System.out.println("socio cadete: " + edad);  
  
        } else {  
  
            System.out.println("socio infantil: " + edad);  
  
        }  
    }  
}
```

# Diagrama mayor de tres números



# Mayor de tres números

```
public class MayorDeTres1 {

    public static void main(String args[]) {
        int n1 = 55;
        int n2 = 27;
        int n3 = 115;

        if ((n1 > n2) && (n1 > n3)) {

            System.out.println("El mayor es: " + n1);

        } else if (n2 > n3) {

            System.out.println("El mayor es: " + n2);

        } else {
            System.out.println("El mayor es: " + n3);

        }

    }

}
```

# Mayor de tres números

```
public class MayorDeTres2 {  
  
    public static void main(String args[]) {  
        int a = 55;  
        int b = 27;  
        int c = 115;  
  
        if ((a > b) && (a > c)) {  
  
            System.out.println("El mayor es: " + a);  
  
        } else if ((b > a) && (b > c)) {  
  
            System.out.println("El mayor es: " + b);  
  
        } else {  
            System.out.println("El mayor es: " + c);  
  
        }  
    }  
}
```

# Condicional cual es el mayor de tres

```
/* * Programa que lee tres números distintos y nos dice cuál de ellos es el mayor*/
import java.util.*;
public class MayorDeTres {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int a, b, c;
        System.out.print("Introduzca primer número: ");
        a = sc.nextInt();
        System.out.print("Introduzca segundo número: ");
        b = sc.nextInt();
        System.out.print("Introduzca tercer número: ");
        c = sc.nextInt();

        if (a > b) {

            if (a > c) {

                System.out.println("El mayor es a: " + a);

            } else {

                System.out.println("el mayor es c: " + c);

            }
        } else {

            if (b > c) {

                System.out.println("el mayor es: " + b);

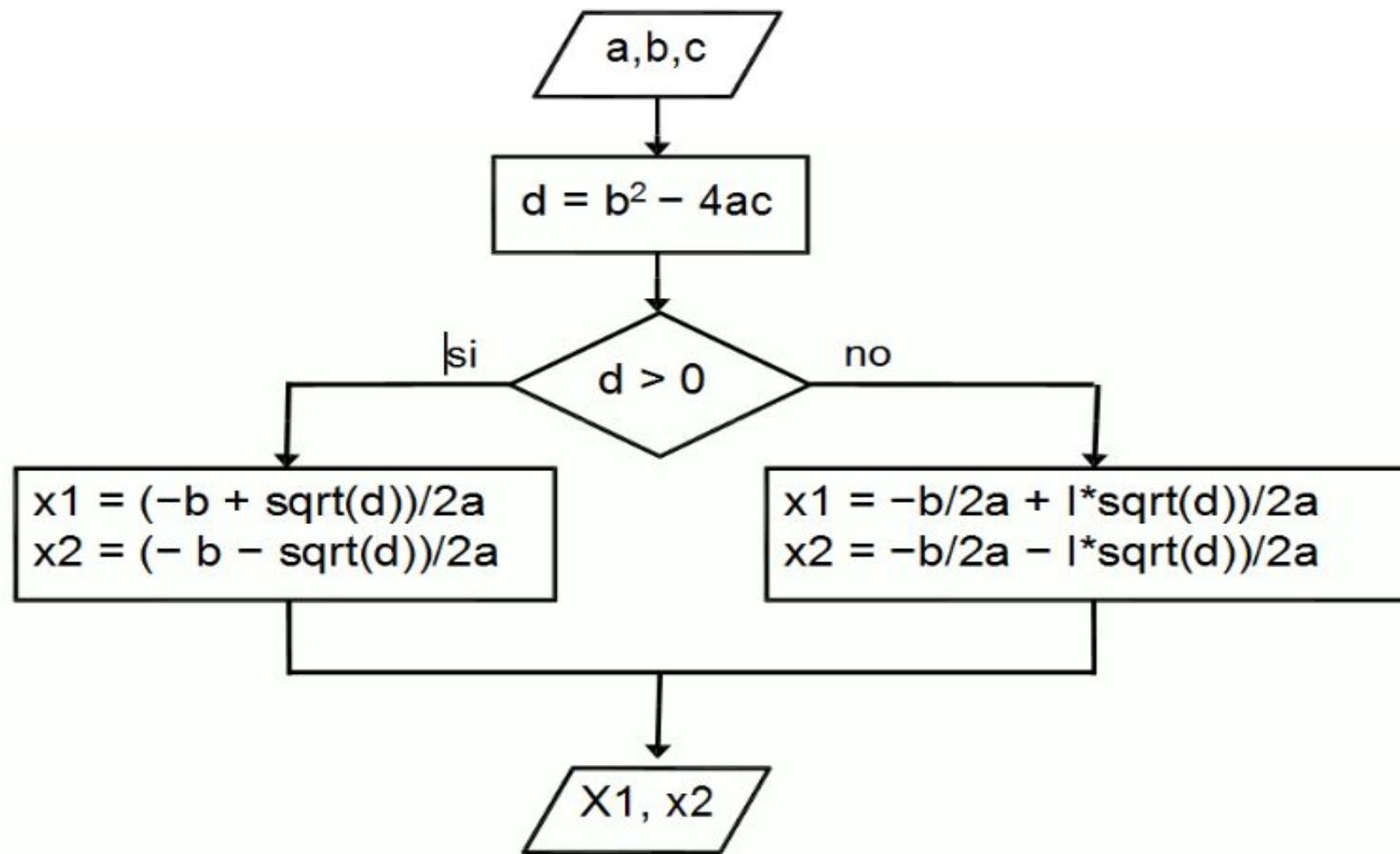
            } else {

                System.out.println("el mayor es: " + c);

            }
        }
    }
}
```

Hacer una maquina de café como Gosling  
Ejercicio para alumnos de selección de tipo de  
cafe

# Diagrama de flujo Cuadrática



# Ecuacion Cuadratica y raíces

$$ax^2 + bx + c = 0$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

# Código ecuación cuadrática:

"NaN" significa "no es un número". "Nan" se produce si una operación de punto flotante tiene algunos parámetros de entrada que hacen que la operación para producir algún resultado indefinido.

Por ejemplo, 0,0 dividido por 0,0 es aritméticamente indefinido. Tomando la raíz cuadrada de un número negativo también está definido.

```
public class Cuadratica { //ax^2+bx+c=0

    public static void main(String args[]) {

        double a = 1; //1(r), 1(ns)
        double b = 1; //−3(r), 1(ns)
        double c = 2; //2(r), 2(ns)

        double x1;
        double x2;

        if (Math.pow(b, 2) - (4 * a * c) > 0) { //discriminante

            x1 = (-b + Math.sqrt(Math.pow(b, 2) - (4 * a * c))) / (2 * a);
            x2 = (-b - Math.sqrt(Math.pow(b, 2) - (4 * a * c))) / (2 * a);

            System.out.println("Tiene dos soluciones x1: " + x1 + " y x2: " + x2);

        } else if (Math.pow(b, 2) - (4 * a * c) == 0) {

            x1 = (-b + Math.sqrt(Math.pow(b, 2) - (4 * a * c))) / (2 * a);
            x2 = (-b - Math.sqrt(Math.pow(b, 2) - (4 * a * c))) / (2 * a);

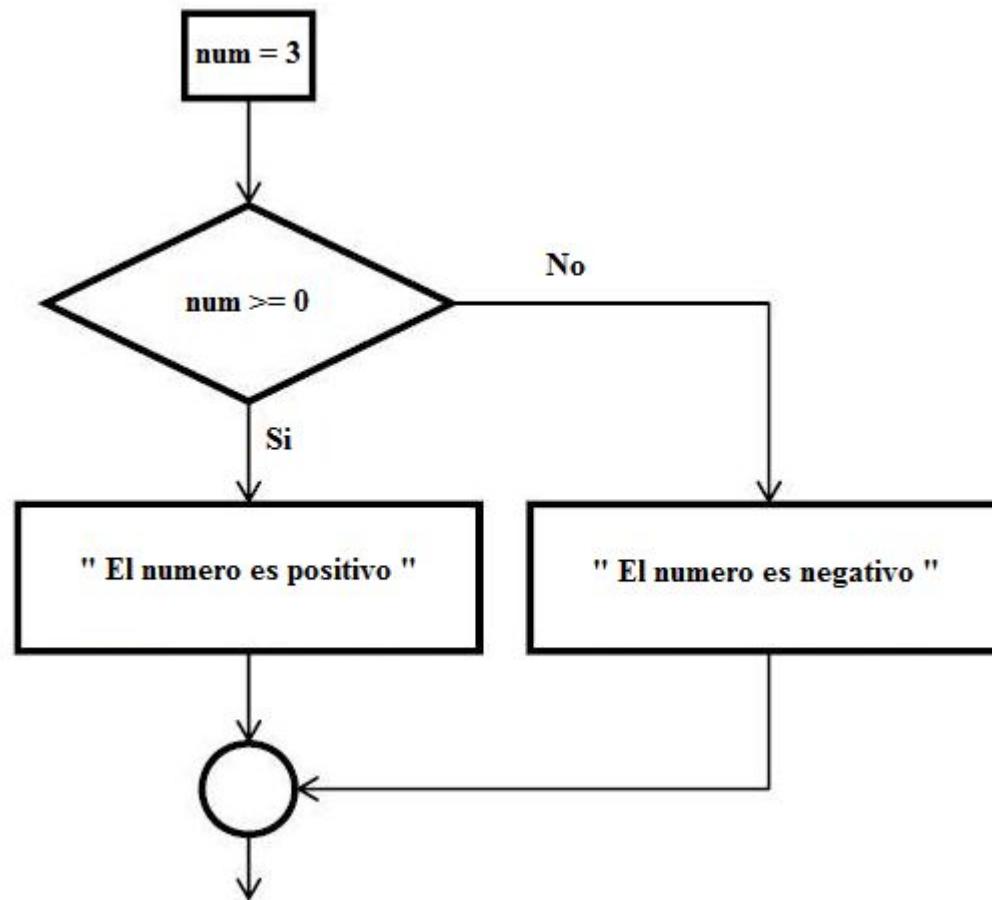
            System.out.println("La solución es real x1: " + x1 + " y x2: " + x2);

        } else if (Math.pow(b, 2) - (4 * a * c) < 0) {

            x1 = (-b + Math.sqrt(Math.pow(b, 2) - (4 * a * c))) / (2 * a);
            x2 = (-b - Math.sqrt(Math.pow(b, 2) - (4 * a * c))) / (2 * a);

            System.out.println("no tiene solución real x1: " + x1 + " y x2: " + x2);
        }
    }
}
```

# Número positivo y negativo



# EJ. Número positivo y negativo

```
public class PositivoNegativo {  
  
    public static void main(String args[]) {  
  
        int num = -1;  
  
        if (num >= 0) {  
  
            System.out.println("El numero es positivo");  
  
        } else {  
  
            System.out.println("El numero es negativo");  
  
        }  
    }  
}
```

# EJ. Comparación 1 carácter

```
public class CarIgual {  
  
    public static void main(String args[]) {  
  
        char letra = 'a'; //almacena un solo valor  
  
        if (letra == 'a') {  
  
            System.out.println("El valor es a");  
  
        } else {  
  
            System.out.println("El valor No es a");  
        }  
    }  
}
```

# Carácter de formato (%)

Tienen la función de indicar, dentro de la cadena de control, que formato tendran los elementos de la lista de valores.

Los caracteres de formato comienzan con el carácter de escape “%”.

Usamos el printf para salida la (f) es de formato.

Se utiliza para dar formato a la salida en la consola.

System.out.printf("formato",variable); // formato (%s, %d, %f) y la variable

Existen muchos especificadores de conversión para el tipo de parámetro recibido, a continuación mostramos una tabla de los soportados por el método **'String.format()'** que es otro metodo de conversion de formatos

# Carácter formatos java

Conversor	Valor
%b	Booleano
%h	Hashcode
%s	Cadena
%c	Caracter unicode
%d	Entero decimal
%o	Entero octal
%x	Entero hexadecimal
%f	Real decimal
%e	Real notación científica
%g	Real notación científica o decimal
%a	Real hexadecimal con mantisa y exponente
%t	Fecha u hora

# Conversión con carácter de formato a (Hexa, String, Octal)

```
//System.out.println(Integer.toBinaryString(numero)); metodo decimal binario

public class FormaToRest1 {

    public static void main(String args[]) {

        int n = 2813;

        System.out.printf("En Hexadecimal es: %x \nEn string es: %s \nEn Octal es: %o\n", n, n, n);
    }
}
```

# Tipos primitivos y sus valores

Tipo	Representación / Valor	Tamaño (en bits)	Valor mínimo	Valor máximo	Valor por defecto
<b>boolean</b>	true o false	1	N.A.	N.A.	false
<b>char</b>	Carácter Unicode	16	\u0000	\uFFFF	\u0000
<b>byte</b>	Entero con signo	8	-128	128	0
<b>short</b>	Entero con signo	16	-32768	32767	0
<b>int</b>	Entero con signo	32	-2147483648	2147483647	0
<b>long</b>	Entero con signo	64	-9223372036854775808	9223372036854775807	0
<b>float</b>	Coma flotante de precisión simple Norma IEEE 754	32	$\pm 3.40282347E+38$	$\pm 1.40239846E-45$	0.0
<b>double</b>	Coma flotante de precisión doble Norma IEEE 754	64	$\pm 1.79769313486231570E+308$	$\pm 4.94065645841246544E-324$	0.0

# Ejemplo formato Float (%f)

Si queremos mostrar el número 12.3698 de tipo double con dos (2) decimales:

```
System.out.printf("%.2f %n", 12.3698);
```

El primer % indica que en esa posición se va a escribir un valor. El valor a escribir se encuentra a continuación de las comillas.

El **(.2)** indica el número de decimales en este caso dos (2).

La (f) indica que el número es de tipo float o double.

% indica un salto de línea. Equivale a \n. con printf podemos usar ambos para hacer un salto de línea.

La **salida por pantalla** será: 12.**37** redondeando el número decimal y sale con dos decimales.

Valor de n con tres decimales **(.3)**:

Double n = 1.25036; //el formato double es un real de coma flotante de precisión simple

System.out.printf("%.3f %n",n); //mientras que el float es un double de precisión doble "mas largo"

La **salida por pantalla** es: 1.**250** salió con tres decimales

# EJ. formato Float (%+.3f)

Para mostrar el signo (+) en un numero positivo lo hacemos de la siguiente manera:  
Si queremos salga: +1.250

Double n = 1.25036;

System.out.printf("%+.3f %n",n);

La salida sera: +1.250

# Ejemplo formato Entero (%d)

Si el número a mostrar es un entero se utiliza el carácter (d). Recordemos que un entero no tiene parte decimal (.25036) como el ejemplo anterior en float que era (1.25036) por tanto no se colocara (%.**2**d) porque sería error si no tenemos decimales. El número 10 es un entero. El **(.2)** indica el número de decimales a quitar un entero no tiene decimales. Pero si puede decir (%4d) como separador de espacios incluyendo los dos del 10 sería 4.

Int x = 10;

```
System.out.printf("%4d %n",x);
```

Saldrá en pantalla:bb10 //quedan dos (2) espacios tomando el 10 donde (b) es espacio.

Con signo agregó el (+) y sale: +10 de la siguiente manera %+d

# Ejemplo formato String (%s)

Tambien se puede cuando vamos a mostrar una cadena de caracteres.

```
String variable = "Hola Mundo";
```

```
System.out.printf("%s",variable); //el formato es (%s) en este caso y la variable  
System.out.printf("formato",variable);
```

Salida por pantalla es. Hola Mundo

## Ejemplo para varias variables de formato (%s, %d, %f)

Para mostrar varias variables tendremos tantos % como valores a mostrar

```
double n = 1.25036;  
Int x = 10;
```

```
System.out.printf("n = %.2f x = %d %n",n , x);
```

La **salida por pantalla** es: n = 1.25 x = 10

**Otro tipo de formato** que presenta: (1\$, 2\$, 3\$)

```
System.out.printf("n = %1$.2f x = %2$d %n", n, x);
```

La **salida por pantalla** será: 1\$, 2\$, 3\$

**Otro tipo de formato** será: si queremos mostrar el 123.4567 y su cuadrado ambos con dos decimales se escribirá así:

```
Double n = 123.4567;
```

```
System.out.printf("El cuadrado de %.2f es %.2f\n", n, n*n);
```

La **salida por pantalla** será: El cuadrado de 123.4567 es 15241.56

# Formatos varios espaciado (%d, %f, %s)

Printf, permite mostrar valores con un ancho de campo determinado. Por ejemplo, si queremos mostrar el contenido de (n) en un ancho de campo de 10 caracteres:

```
double n = 1.25036;
```

```
System.out.printf("%+10.2f %n",n); //el 10 es el espaciado derecha y .2 son los decimales a mostrar
```

```
System.out.printf("%-10.2f %n",n); //el 10 es el espaciado izquierda y .2 son los decimales a mostrar
```

La **salida por pantalla** con (+) derecha será: **bbbbbb+1.25** //tomando las (b) son 5 y (+1.25) son 5 con el punto dará los diez espacios. El (+) es a la derecha 10 espacios y (-) es a la izquierda 10 espacios

Donde cada (b) indica un espacio en blanco en este caso automáticamente toma 5 teniendo en cuenta el (+1.25) que son 5 incluyendo el punto dando en total 10 espacios incluyendo los caracteres (+1.25).

La **salida por pantalla** con (-) izquierda será: **1.25**

# Formatos varios número completado por ceros

Si queremos que en lugar de espacios en blancos nos muestre el numero completando el ancho con ceros:

```
double n = 1,25036
```

```
System.out.printf("%+010.2f %n", n); //escribimos un cero antes del 10
```

La salida por pantalla es: +000001.25 // el (+) a la derecha (-) a la izquierda

# Formato espaciado String

Mostrar la cadena “total:” con un ancho de 10 caracteres y alineada a la derecha.

```
String total = "hola mundo";
```

```
System.out.printf("La cadena de caracteres es: %10s", total); // 10 espacios a la derecha  
System.out.printf("La cadena de caracteres es: %-10s", total); // 10 espacios a la izquierda
```

La salida por pantalla es: bbbbbbtotal // las (b) son espaciados

La salida por pantalla es: total

// para mostrar títulos espaciados.

```
System.out.printf("%20s %20s %20s %n", "número de control", "alumno", "calificación"); // justificado a la derecha
```

```
System.out.printf("%-20s %-20s %-20s %n", "número de control", "alumno", "calificación"); // justificado a la izquierda
```

## Formato variado (%d, %f, %s)

```
Public static void main(String args[]) {  
  
    double q = 1.0 / 3.0;  
    System.out.printf("1.0 / 3.0 = % 5.3 f % n", q);  
    System.out.printf("1.0 / 3.0 = % 7.5 f % n", q);  
    q = 1.0 / 2.0;  
    System.out.printf("1.0 / 2.0 = % 09.3 f % n", q);  
    q = 1000.0 / 3.0;  
}
```

La salida por pantalla es:

```
1.0 / 3.0 = 0.333  
1.0 / 3.0 = 0.33333  
1.0 / 2.0 = 00000.500
```

# EJ. Casteo y ascii decimal

```
public class CarCasteoInt{//su decimal es 65 lo cual (Alt + 65) y hexadecimal (0041)

    public static void main(String args[]){

        char ch = 'A';

        int chNumero = (int)ch;//castea a entero

        System.out.printf("ch:%c chNumero:%d chHex:%04x%n",ch, chNumero,
chNumero);    //muestra el ascii y hexadecimal
    }

}
```

# EJ. Carácter de Formato (%) usado en java

```
public class FormatoNum{  
  
    public static void main(String args[]){  
  
        int a, b;  
  
        a = 130;  
        b = 7;  
  
        System.out.printf("\nDivision entera: \t %d / %d = %d", a, b, a/b);  
        System.out.printf("\nResto de Division entera: \t %d %% %d = %d", a, b, a%b);  
        System.out.printf("\nDivision real: \t\t %d / %d = %.2f", a, b, (float)a/b);  
  
    }  
  
}
```

# EJ. Comparación carácter entrada

```
import java.io.*;  
  
public class CondicionalChart {  
  
    public static void main(String[] args) throws IOException {  
  
        char car1, car2;  
        System.out.print("Introduzca primer carácter: ");  
        car1 = (char) System.in.read(); //lee un carácter  
  
        System.in.read(); //saltar el intro que ha quedado en el buffer  
  
        car2 = 'A';  
  
        if (car1 == car2)  
            System.out.println("Son iguales");  
        else  
            System.out.println("No son iguales");  
    }  
}
```

# Carácter formato char a Decimal



```
public class CaraterFormat{  
  
    public static void main(String args[]){  
  
        char ch = 'A';  
  
        int chNumero = (int) ch;  
  
        System.out.printf("Caracter:%c    ASCII decimal: %d\n", ch, chNumero);  
    }  
}
```

# Entrada carácter in.read()

```
public class CharacterAscii {  
  
    public static void main(String args[]) throws java.io.IOException {  
  
        System.out.println("Introducir un caracter: ");  
  
        char entrada;  
  
        entrada = (char) System.in.read(); //returna codigo ascii de 1 caracter  
  
        System.out.println("El caracter tipeado es: " + entrada); //podria imprimir los caracteres  
    }  
}
```

# EJ. Ingreso de un carácter por teclado.

```
● ● ●

import java.util.Scanner;

public class CarArt {

    public static void main(String args[]) {

        Scanner teclado = new Scanner(System.in);

        System.out.println("Ingrese un caracter: ");

        char caracter = teclado.next().charAt(0); //asegura que la posicion sea (0)
        //o sea el primer caracter ingresado

        System.out.println("Su caracter ingresado es: " + caracter);

    }
}
```

# EJ. Largo de una palabra y selección de un carácter.



```
public class CadenaChart {  
  
    public static void main(String args[]) {  
  
        String nombre = "Pepe";  
  
        System.out.println("Mi nombre es: " + nombre);  
  
        //metodo que nos da el largo de la palabra  
        System.out.println("Mi nombre tiene " + nombre.length() + " letras");  
        //posision de caracter  
        //System.out.println("La segunda letra del nombre es"+nombre.charAt(indice));  
        System.out.println("La segunda letra del nombre es " + nombre.charAt(1));  
  
    }  
}
```

# EJ. Posición para una frase y un carácter.

```
public class PosicionChart {  
  
    public static void main(String args[]) {  
  
        String cadena = "la cadena es larga";  
  
        int posicion0 = cadena.indexOf("es"); //donde comienza la frase  
        int posicion1 = cadena.lastIndexOf("a"); //da la posicion del ultimo caracter "a" escrito  
        int posicion2 = cadena.indexOf("a"); //da la posicion del primer caracter "a" escrito  
        Int posicion3 = cadena.indexOf(" "); //espacio vacio  
  
        String primeraFrase = cadena.substring(0, posicion0); //posicion cero hasta primera frase  
  
        System.out.println("La posicion de la frase inicia en: " + posicion0); //para una frase  
        System.out.println("El ultimo caracter esta en la posicion: " + posicion1); //para un caracter  
        System.out.println("El primer caracter esta en la posicion: " + posicion2); //para un carácter  
        System.out.println("El primer espacio esta en la posicion: " + posicion3); //para un carácter  
        System.out.println("Mi primera frase es: " + primeraFrase);  
    }  
}
```

# EJ. Comparar String e Instancias.



```
public class CompareStrings {  
  
    public static void main(String args[]) {  
  
        //String cadena1 = "hola";//cadenas iguales y desiguales  
        //String cadena2 = "Hola";  
  
        String cadena1 = new String("hola"); //cadenas desiguales por  instanciacion  
        String cadena2 = new String("hola"); //segunda instancia  
  
        if (cadena1 == cadena2) {  
  
            System.out.println("cadenas iguales");  
  
        } else {  
  
            System.out.println("cadenas distintas");  
        }  
    }  
}
```

# Uso de clase String

El resultado es que las cadenas son desiguales.

Explicación:

Al hacer `String cadena1 = "Prueba"`, el compilador ya conserva en memoria "Prueba" y al setearle a `cadena2 = "Prueba"`, el compilador utiliza la misma instancia.

Pero en el caso de abajo que hacemos:

**String cadena1 = new String("Prueba")** y

**String cadena2 = new String("Prueba")** , el operador **NEW** hace una nueva instancia del string "Prueba", por lo cual son instancias diferentes. Por lo tanto, al ser instancias diferentes sus valores son distintos.

# Comparaciones String equals() y equalsIgnoreCase()

## **1º Comparar con .equals()**

El método equals() lo poseen todos las clases ya que se heredan de la clase base Object, sirve especialmente para comparar una cadena de un objeto específico, devolviendo un dato booleano.

## **2º Comparar con el .equalsIgnoreCase()**

Funciona igual que el equals(), pero ignora las mayúsculas.

## **3º Comparar con el .compareTo()**

Este método sirve para comparar dos cadenas, devolviendo 0, positivo o negativo.

Devuelve positivo si la cadena pasada por parámetro se ordena antes.

Devuelve negativo si la cadena pasada por parámetro se ordena después.

Hay que tener en cuenta que el compareTo() compra según estén ordenados los caracteres en la tabla de caracteres que usemos ya que una A mayúscula va antes que una minúscula y esta va antes que una á (a con acento).

La cadena2 puede llegar a ser nula lo que provocaría un nullpointer excepción.

## **4º Comparar con compareIgnoreCase().**

Este método sirve para comparar dos cadenas, devolviendo 0, positivo o negativo.

Funciona casi igual que el compareTo() pero se diferencia que ignora las mayúsculas y minúsculas.

# Operador lógico y Equals

## Operador lógico ||

Ahora se echará un vistazo al complemento del operador “and”: el operador “or” (o). Asuma que tiene una variable llamada respuesta que contiene 1) una “s” mayúscula o minúscula si el usuario desea salir o 2) algún otro carácter si el usuario desea continuar.

Escriba un fragmento de código que imprima “Adiós” si el usuario introduce una “s” ya sea mayúscula o minúscula. Al utilizar pseudocódigo, podría escribir algo como lo siguiente para la parte crítica del algoritmo: si respuesta es igual a “s” o “S” imprimir “Adiós”.

Observe la “o” en la condición de la sentencia if. Esto funciona bien para el pseudocódigo, donde las reglas de sintaxis son poco exigentes, pero en Java se tendría que utilizar el operador || en la computadora, Buscar este símbolo en el teclado y presionarlo dos veces. He aquí una implementación tentativa del fragmento de código en Java:



# Programa (==) igual e igual

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

String respuesta;

System.out.print("Introduzca s o S: ");

respuesta = stdIn.nextLine();

if (respuesta = "s" || respuesta = "S"){ //esto compila pero no funciona

    System.out.println("Adiós");

}

}
```

# Programa (==) igual e igual

Observe que la variable respuesta aparece dos veces en la sentencia de condición if. Esto es necesario porque si ambos lados de una condición || involucran la misma variable, ésta debe repetirse.

La llamada indica que algo está mal, ¿qué es esto? Si se inserta este fragmento de código en un programa, éste compila y ejecuta pero cuando el usuario responda a la petición introduciendo ya sea una “s” o una “S” nada sucederá. El programa no imprime “Adiós”. ¿Por qué no? ¿Se debió haber utilizado un paréntesis interior en la condición “if”? La figura 4.6 muestra que el operador == tiene una mayor precedencia que el operador ||, entonces lo que se hizo es correcto. El problema radica en algo más.

# Comparar Cadena de Caracteres

## **No utilizar == para comparar cadenas de caracteres**

El problema es con las expresiones respuesta == "s" y respuesta == "S". Enfoquemos la expresión con la respuesta == "s". La variable de cadena de caracteres respuesta y la cadena literal de caracteres "s" apuntan ambas a la dirección de memoria que apunta a objetos de tipo cadena de caracteres;

no almacenan en sí objetos de tipo String. Así, cuando se utiliza el doble signo de igual (==) se están

comparando las direcciones de memoria almacenadas en la variable tipo String respuesta y la cadena

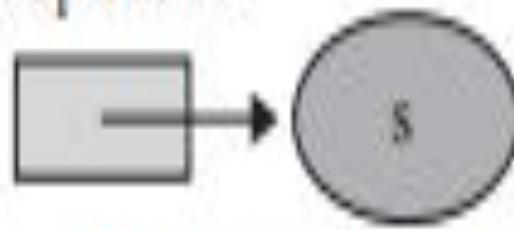
literal "s". Si la variable de respuesta tipo cadena de caracteres y la cadena literal de caracteres "s" contienen direcciones de memoria diferentes (apuntan a objetos tipo String diferentes), entonces la comparación

da como resultado el valor falso, aunque ambos objetos de cadena de caracteres contengan la misma secuencia de caracteres. La siguiente imagen muestra esto. Las flechas representan direcciones de

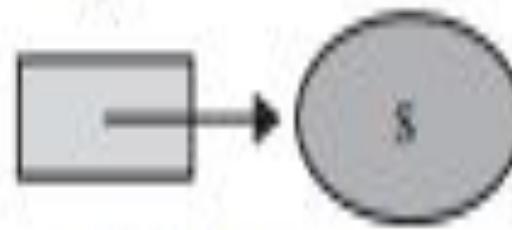
memoria. Puesto que apuntan a diferentes objetos, respuesta == "s" se evalúa como falso.

# Esquema de programación

respuesta



"s"



respuesta == "s"  $\Rightarrow$  falso

respuesta.equal s("s")  $\Rightarrow$  verdadero

# Comparar Cadena de Caracteres

Entonces, ¿qué se puede hacer para resolver este problema? En el capítulo 3 se aprendió a utilizar el método `respuesta` para probar la igualdad de dos cadenas de caracteres. El método `respuesta` compara los objetos de tipo `String` referenciados por una dirección de memoria. En la figura de arriba, los objetos de tipo `String` tienen la misma secuencia de caracteres, `s` y `s`, por lo que la llamada al método `respuesta.respuesta("s")` devuelve un valor de verdadero, que es lo que se desea. He aquí el fragmento de código corregido:

```
if (respuesta.equals("s") || respuesta.respuesta("S"))
{
    System.out.println("Adiós");
}
```

O, como una alternativa más compacta, se puede utilizar el método `respuesta.ignoreCase` como a continuación:

# IgnoreCase Igualacion

```
if (respuesta.respuestaignoreCase("s"))
{
    System.out.println("Adiós");
}
```

Una tercera alternativa es utilizar el método charAt de la clase String para convertir la cadena de entrada en un carácter y después utilizar el operado == para comparar ese carácter con los caracteres literales 's' y 'S'.

```
char resp = respuesta.charAt(0);
if (resp == 'q' || resp == 'Q')
{
    System.out.println("Adiós");
}
```

# Errores lógicos Equals

## Errores

Se hizo un gran esfuerzo por no utilizar los signos == para comparar cadenas de caracteres porque es muy fácil cometer el error y muy difícil encontrarlo. Es fácil cometer el error porque se utilizan los == todo el tiempo para comparar valores primitivos. Es difícil encontrar el error porque los programas que utilizan el == para comparación de cadenas de caracteres compilan y ejecutan sin errores reportados. ¿Sin errores reportados? Entonces, ¿por qué preocuparse? Porque aunque no se reporten errores, éstos existen: se denominan errores lógicos.

Un error lógico ocurre cuando el programa ejecuta por completo sin mensajes de error, y la salida es incorrecta. Los errores lógicos son los más difíciles de encontrar y de solucionar porque no hay mensaje que los saque a la luz, indicando lo que se hizo mal. Para empeorar las cosas, la utilización de == para comparación de cadenas de caracteres genera un error lógico sólo en ocasiones, no siempre. Debido a que el error lógico ocurre sólo en ocasiones, los programadores pueden confiarse y pensar que su código está correcto cuando en realidad no es así.

Hay tres categorías principales de errores: de compilación, de ejecución y lógicos. Un error de compilación es aquel que es identificado por el proceso de compilación. Un error de ejecución es el que ocurre mientras se ejecuta el programa y provoca que el programa termine anormalmente. El compilador genera un mensaje por un error de compilación, y la máquina virtual de Java (JVM) genera un mensaje por un error de ejecución. Desafortunadamente, no hay mensajes de error para los lógicos. Depende del programador solucionarlos mediante el análisis de las salidas y planeando cuidadosamente el código.

# Que es una Instancia en una clase

Cuando se crea un objeto (se **instanci**a una clase) es posible definir un proceso de inicialización que prepare el objeto para ser usado (inicializar sus miembros). Esta inicialización se lleva a cabo invocando un **método** especial denominado **constructor**.

Esta invocación es implícita y se realiza automáticamente cuando se utiliza el operador (new).

El operador (**new**) se llama operador de **instanciación**. Este código nos dice: que crea una variable de referencia que se llama (cadena1 o cadena2), que será del tipo String y el nuevo objeto creado (con new) le dará una posición de memoria determinada con el cual podrá ubicar a este objeto String en la memoria.

Los constructores tienen algunas características especiales:

- . El nombre del constructor tiene que ser igual al de la clase.
- . Puede recibir cualquier número de argumentos de cualquier tipo, como cualquier otro método.
- . No devuelve ningún valor (en su declaración no se declara ni siquiera void).
- . Puedo tener más de un constructor por clase. Cuando se declaran varios constructores para una misma clase estos deben **distinguirse en la lista de argumentos, bien en el número, o bien en el tipo**.

**Resumen:** cuando defino una clase en nuestro caso (String) y con ella creo objetos de tipo String y sus respectivas variables de referencia. A esto en la jerga se llama instanciar la clase. (instanciar es crear el objeto y colocarle su variable de referencia).

Es decir, un objeto es una instancia de una clase. Que queda identificado por su variable de referencia.

# Instanciación de una clase y ejemplo real

Como hemos comentado, una clase es el tipo que tendrán nuestros objetos, también llamados instancias de esa clase, es decir, la clase es la forma que tendrán los objetos empleados por nuestra aplicación.

Un ejemplo puede ser el plano de una casa, el plano sería nuestra clase, pero con ese mismo plano podremos construir diversas casas que serían los objetos. El plano en sí no es nada, es sólamente la especificación de lo que será una casa una vez construida.

**Alcance de los miembros:** de instancia o de clase.

Cuando creamos una instancia de la clase en ejecución tendremos las variables de instancia, una por cada atributo, y todas integradas dentro de la misma instancia. Todas las instancias de la misma clase tendrán los mismos atributos (nombre y tipo), pero cada una de ellas irá tomando distintos valores para ellos. Es decir, tendremos una copia de los atributos por cada objeto de una clase que creamos. Por ejemplo, tu Casa sería una instancia de la clase Casa y daremos a la variable de instancia altura Máxima el valor 3.0 mts. Mientras que la instancia casaAmericana tiene su propia variable alturaMaxima con el valor 3.5.

# Ejemplo de instancias en clases

Hay otro tipo de atributos, las llamadas **variables de clase**, que son los **atributos** que sólo existen una vez para cada clase. Es decir este tipo de **atributos** será compartido por todos los objetos que creemos de una clase, de modo que si cambio el valor del atributo en un objeto se verá reflejado en todos los demás.

Por ejemplo, el atributo numDeCasas de nuestra clase Casa, de tipo entero, pensado para contar las casas que vamos teniendo en nuestro sistema, sólo debería existir una vez: no tiene sentido que esta variable sea distinta en tuCasa y en casaAmericana. Es la misma (única) variable para toda la clase Casa.

El comportamiento, la funcionalidad de la clase, es la concreción de qué puede hacer cada una de las instancias de la clase, cómo cambian su estado, o cómo se relaciona con otras.

Por ejemplo, en la clase Casa podríamos determinar la siguiente funcionalidad:  
Altos biombos, librerías, sofás o butacas ayudan a delimitar las distintas zonas de la casa etc.

Podemos tener casas de madera, concreto, metal (aluminio, chapa de hierro containers), plastico etc.  
El comportamiento va relacionado con su función.

Para definir el **comportamiento de los objetos** definiremos métodos, que no serán más que funciones o procedimientos definidas dentro de la clase que se ejecutan y operan sobre instancias de esa clase. Notar que estos métodos se encapsulan dentro de la clase, es decir, encapsulamos conjuntamente datos y funcionalidad relativa a un mismo objeto, cada objeto debe proveer el mecanismo de almacenamiento y gestión de sus datos, y la funcionalidad relativa a dicho objeto como una entidad cerrada.

Igual que los atributos, también hay métodos de instancia (se aplican sobre una instancia: son los más comunes, sobre un objeto concreto) y métodos de clase (se aplican sobre la propia clase).

# EJ. Comparación equals con instancia

```
public class ComparIquals {

    public static void main(String args[]) {

        //String cadena1 = "hola";//cadenas iguales y desiguales
        //String cadena2 = "Hola";

        //Decimos que un objeto es una instancia de una clase.
        // puedo crear otro objeto de tipo String con variable de referencia (cadena2)

        String cadena1 = new String("hola"); //cadena1 es la variable de referencia que creara otro objeto de tipo String
        String cadena2 = new String("Hola"); //la segunda instancia es cadena2

        if (cadena2.equals(cadena1)) {

            System.out.println("cadenas iguales");

        } else {

            System.out.println("cadenas distintas");
        }
    }
}
```

# EJ. Método Equals to String

```
public class EqualString {  
  
    public static void main(String args[]) {  
  
        String a = "hola";  
        String b = "Hola";  
  
        //boolean igual = a.equals(b); //tienen que ser iguales  
        //boolean igual = a.equalsIgnoreCase(b); //no importa si es mayuscula o minuscula  
  
        //colocar igual como condicion si es booleano  
  
        if (a.equals(b)) { //ignora si son mayusculas o minusculas  
  
            System.out.println("Son iguales");  
  
        } else {  
  
            System.out.println("No son iguales");  
  
        }  
    }  
}
```

# Comparando Caracteres con ChartAt

Una tercera alternativa es utilizar el método `charAt` de la clase `String` para convertir la cadena de entrada en un carácter y después utilizar el operador `(==)` para comparar ese carácter con los caracteres literales `'s'` y `'S'`.



```
public class CarAt1 {

    public static void main(String args[]) {

        char resp = respuesta.charAt(0);

        if(resp == 's' || resp == 'S') {

            System.out.println("Adiós");
        }
    }
}
```

# Método Equals con OR para ignorar Mayuscula de minuscula

```
public class ComparToIgnoreCase2 {  
  
    public static void main(String args[]) {  
  
        Scanner scanner = new Scanner(Sys t em.i n);  
        String respuesta;  
        System.out.print("Introduzca s o S: ");  
        respuesta = scanner.nextLine();  
  
        if (respuesta.equals("s") || respuesta.respuesta("S")) {  
  
            System.out.println("caracteres iguales");  
  
        } else {  
  
            System.out.println("caracteres distintos");  
        }  
    }  
}
```

# Funcion o Metodo CompareTo()

Esta función o método recibe como argumentos dos strings y retorna un valor entero que indica el resultado de la comparación entre ambos.

Compara este objeto con el objeto especificado para el pedido. Devuelve un entero **negativo**, **cero** o un entero **positivo** ya que este objeto es menor que, igual o mayor que el objeto especificado.

**Su prototipo o método es:** x.compareTo(y);

El implementador debe asegurar **sgn** (x.compareTo (y)) == - **sgn** (y.compareTo (x)) para todo x e y.

(Esto implica que x.compareTo (y) debe lanzar una excepción, si y.compareTo (x) lanza una excepción.)

El implementador también debe asegurar que la relación sea transitiva:  
(x.compareTo (y)> 0 && y.compareTo (z)> 0) , implica x.compareTo (z)> 0.

Finalmente, el implementador debe asegurar que x.compareTo (y) == 0 implica que **sgn** (x.compareTo (z)) == **sgn** (y.compareTo (z)), para todo z.

# Método CompareTo() valores retornados

Se recomienda encarecidamente, pero no se requiere estrictamente que `(x.compareTo(y) == 0) == (x.equals(y))`. En general, cualquier clase que implementa la interfaz Comparable y viola esta condición debe indicar claramente este hecho. El lenguaje recomendado es "Nota: esta clase tiene un orden natural que es inconsistente con iguales."

En la descripción anterior, la notación `sgn (expresión)` designa la función signum matemática, que se define para devolver uno de -1, 0 o 1 según que el valor de la expresión sea negativo, cero o positivo.

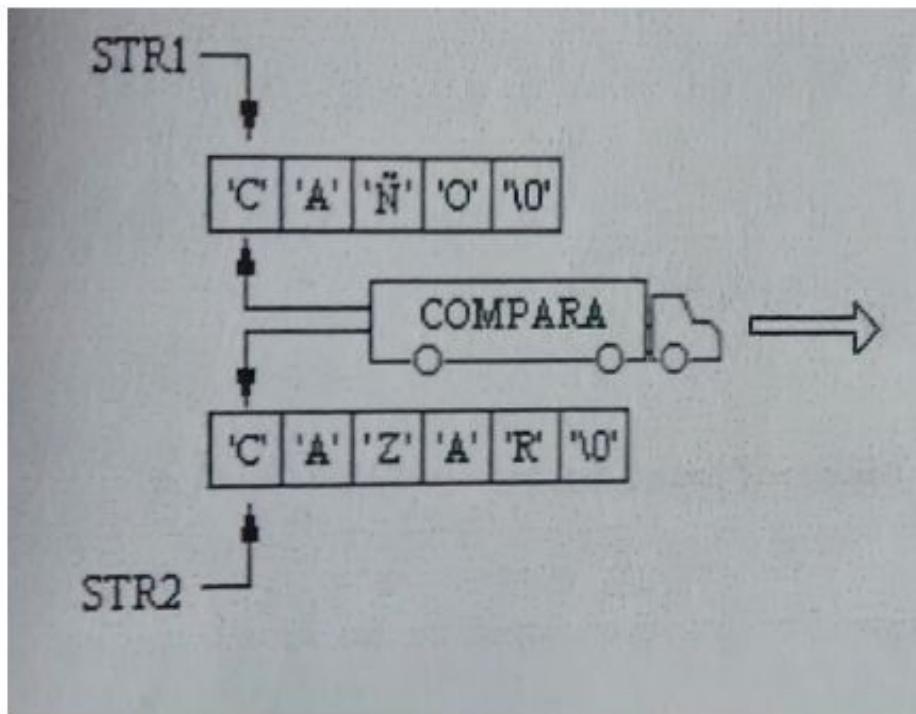
La función retorna un valor entero que responde al siguiente patrón de comparación:

Valor retornado > 0      equivale a (1)      `string1 > string2`

Valor retornado = 0      equivale a (0)      `string1 = string2`

Valor retornado < 0      equivale a (-1)      `string1 < string2`

# Ejemplo de Comparación



El mecanismo segun el que trabaja compareTo( ) es el de ir comparando los caracteres de cada string desde sus posiciones inferiores ( en el dibujo, desde la izquierda ) hasta encontrar algun par desequilibrante.

# Comparacion de Strings

En este ejemplo de la figura se aprecia que la palabra “**CAÑO**” antecede a “**CAZAR**” en un ordenamiento alfabético. Es decir, “**CAÑO**” es menor que “**CAZAR**”.

Sin embargo la comparación de caracteres que realiza **compareTo()** es a nivel de codificación **ASCII**.

Según esta, los códigos correspondientes a “**Ñ**” y “**Z**” son **165** y **90** respectivamente.

De lo anterior surge que la palabra “**CAÑO**” es mayor que “**CAZAR**” (erróneamente).

# EJ. Convertir código decimal Ascii a carácter.

```
● ● ●

import java.util.Scanner;

public class ConvertirAscii {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Introduzca un código decimal Ascii: ");

        String data = sc.nextLine();

        int d = Integer.parseInt(data);
        char c = (char)(d);

        System.out.println(data + " = " + c);
    }
}
```

# Convertir cadena Carácteres a decimal Ascii

```
● ● ●

import java.util.Scanner;

public class ConvertCharToAscii {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Entre un caracter o simbolo:");

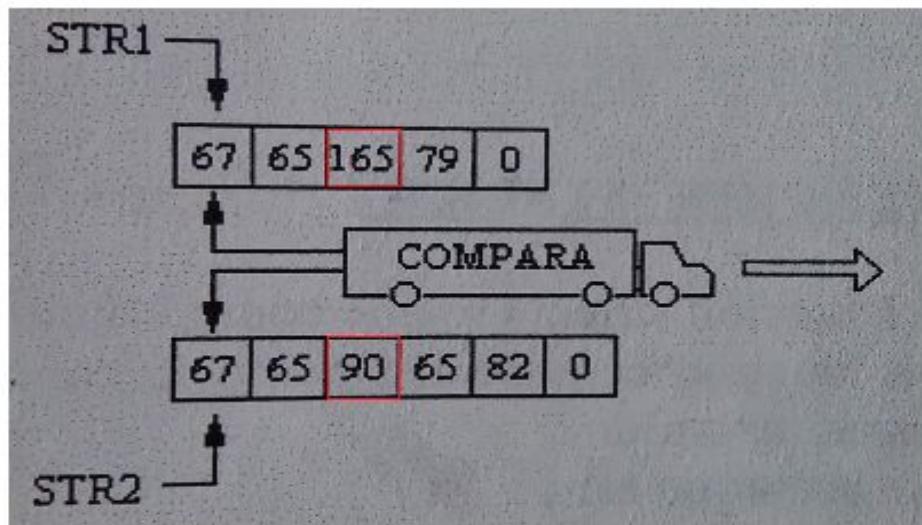
        String data = sc.nextLine();

        System.out.println(data);

        for (int cont = 0; cont < data.length(); cont++) {

            int ascii = data.charAt(cont);
            System.out.println(data.charAt(cont) + "=" + ascii);
        }
    }
}
```

# Comparación String compareTo() verifica las tres opciones



STRING1 > STRING2 esto es: 165 > 90 es positivo (1)

Del mismo modo debe considerarse que los códigos ASCII de las letras minúsculas son mayores numéricamente que los de sus correspondientes mayúsculas

## REGLA NEMOTECNICA:

A fin de recordar las relaciones anteriores suponga que la función compareTo() simplemente resta STR1 - STR2 como si fueran números.

Ej:  $75 > 0$  por lo que  $165 - 90 = 75$  retorna un número positivo (1) "el primero es mayor que el segundo"

$0 = 0$  por lo que  $165 - 165 = 0$  retorna un número igual (0) "ambos son iguales"

$-75 < 0$  por lo que  $90 - 165 = -75$  retorna un número negativo (-1) "el primero es menor que el segundo"

$(a - b) > 0$  entonces  $a > b$ ;  $(a - b) < 0$  entonces  $a < b$ ;  $(a - b) = 0$  entonces  $a = b$

## Método compareTo() cadenas iguales “str1 == str2” es cero (0)

```
public class ComparToIgnor {  
  
    public static void main(String args[]) {  
  
        //String cadena1 = "hola";//cadenas iguales y desiguales  
        //String cadena2 = "Hola";  
  
        String cadena1 = new String("hola");  
        String cadena2 = new String("Hola");  
  
        if (cadena2.compareTo(cadena1) = 0) {  
  
            System.out.println("cadenas iguales");  
  
        } else {  
  
            System.out.println("cadenas distintas");  
        }  
    }  
}
```

# Comparación compareTo() “str1<str2” o negativo (-1)

“Esto no se hace porque cuando son iguales difieren”

```
public class ComparToCadenas {  
  
    public static void main(String args[]) {  
  
        //String cadena1 = "hola";//cadenas iguales y desiguales  
        //String cadena2 = "Hola";// H = 72; h = 104  
  
        String cadena1 = new String("Hola");  
        String cadena2 = new String("hola");  
  
        if (cadena2.compareTo(cadena1) < 0) {  
  
            System.out.println("La cadena1 es menor");  
  
        } else {  
  
            System.out.println("La cadena2 es menor");  
        }  
    }  
}
```

# Comparación compareTo() “string1 > string2” es positivo (1)

## “Esto no se hace porque cuando son iguales difieren”

```
public class ComparToCadenas2 {  
  
    public static void main(String args[]) {  
  
        //String cadena1 = "hola";//cadenas iguales y desiguales  
        //String cadena2 = "Hola";// H = 72; h = 104  
  
        String cadena1 = new String("Hola");  
        String cadena2 = new String("hola");  
  
        if (cadena2.compareTo(cadena1) > 0) { //error cuando compara con (>) no identifica mayor o menor  
            //deberian ser iguales y dice que cadena2 es menor que cadena1  
  
            System.out.println("La cadena1 es menor a cadena2");  
  
        } else {  
  
            System.out.println("La cadena2 es menor a cadena 1");  
        }  
    }  
}
```

# EJ. Método compareToIgnoreCase()

```
public class ComparToIgnoreCase {  
    public static void main(String args[]) {  
        //String cadena1 = "hola";//cadenas iguales y desiguales  
        //String cadena2 = "Hola";  
  
        String cadena1 = new String("hola"); //cadenas desiguales por  instanciacion  
        String cadena2 = new String("Hola"); //segunda instancia  
  
        if (cadena2.compareToIgnoreCase(cadena1) == 0) {  
            System.out.println("cadenas iguales");  
        } else {  
            System.out.println("cadenas distintas");  
        }  
    }  
}
```

# Ejercicio resumido de compareTo()

```
import java.util.Scanner;
public class EqualCompare {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Introducir primer dato s1: ");
        String s1 = sc.nextLine();

        System.out.println("Introducir segundo dato s2: ");
        String s2 = sc.nextLine();

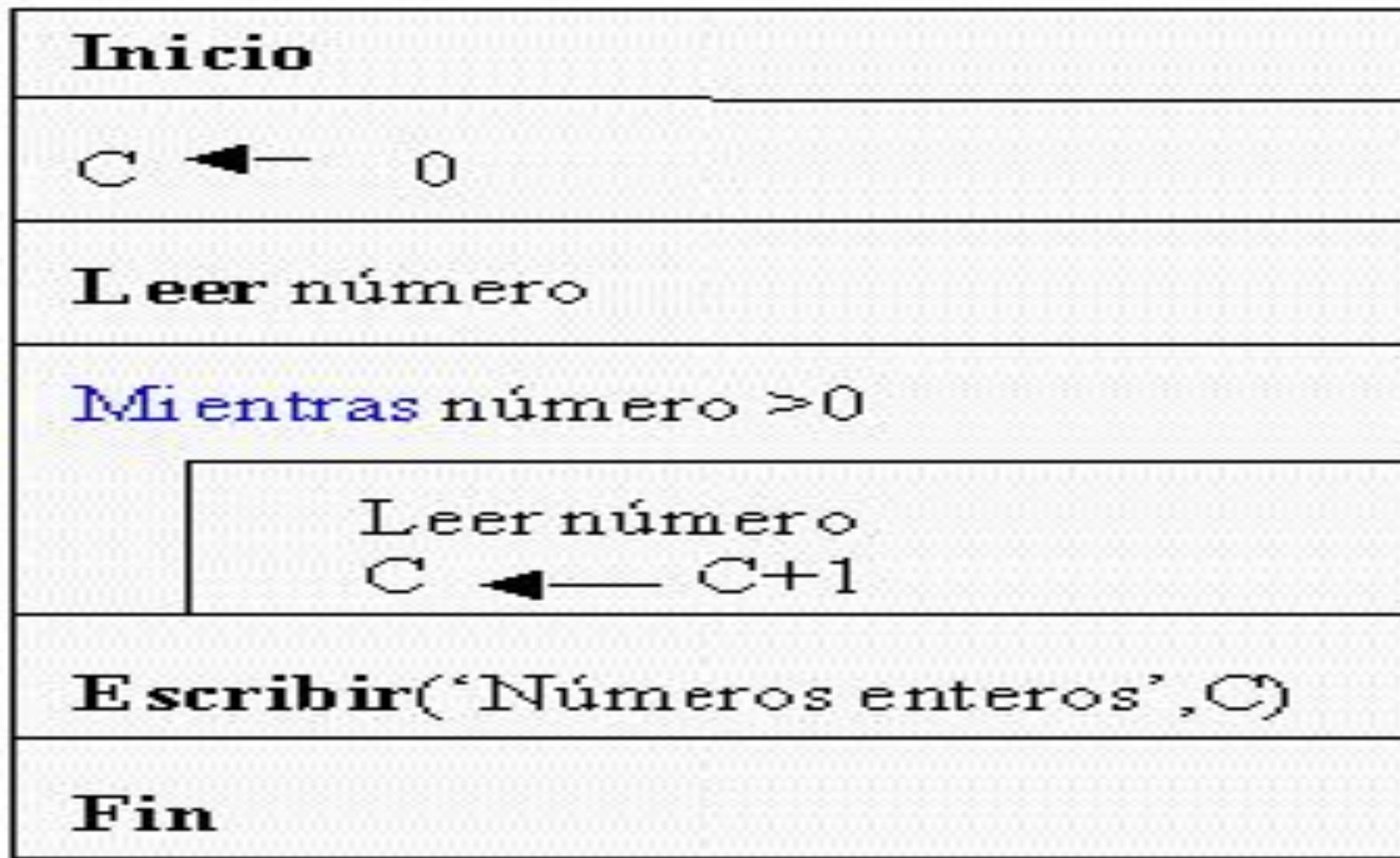
        if (s1.compareTo(s2) < 0) {

            System.out.println("es negativo s1 es menor a s2.");
        } else if (s1.compareTo(s2) > 0) {

            System.out.println("es positivo s1 es mayor a s2.");
        } else if (s1.compareTo(s2) == 0) {

            System.out.println("s1 y s2 son iguales.");
        }
    }
}
```

# Diagrama bucle while o mientras



# Bucles en java

## Bucles

```
[ inicialización; ]  
do {  
    sentencias;  
    [ iteración; ]  
} while(expresión-booleana);
```

```
for( inicialización; exp-booleana; iteración ) {  
    sentencias;  
}
```

```
[ inicialización; ]  
while( expresión-booleana ) {  
    sentencias;  
    [ iteración; ]  
}
```

# Tipos de Bucle “bucle While”

**Bucle while:** en este lazo se ingresa a evaluar la condición.

Si se cumple se ejecuta la sentencia(que puede ser **simple, nula o compuesta**),y luego vuelve a evaluarse la condición.

Por lo tanto mientras (While) la condición se cumpla se ejecutara la sentencia, como lo mostró la figura anterior.

Nótese que la sentencia puede no ejecutarse nunca si la condición no se cumple en primera instancia.

**Condición compuesta:** cuando una única expresión booleana está formada por más de una condición, se dice que es una condición compuesta.

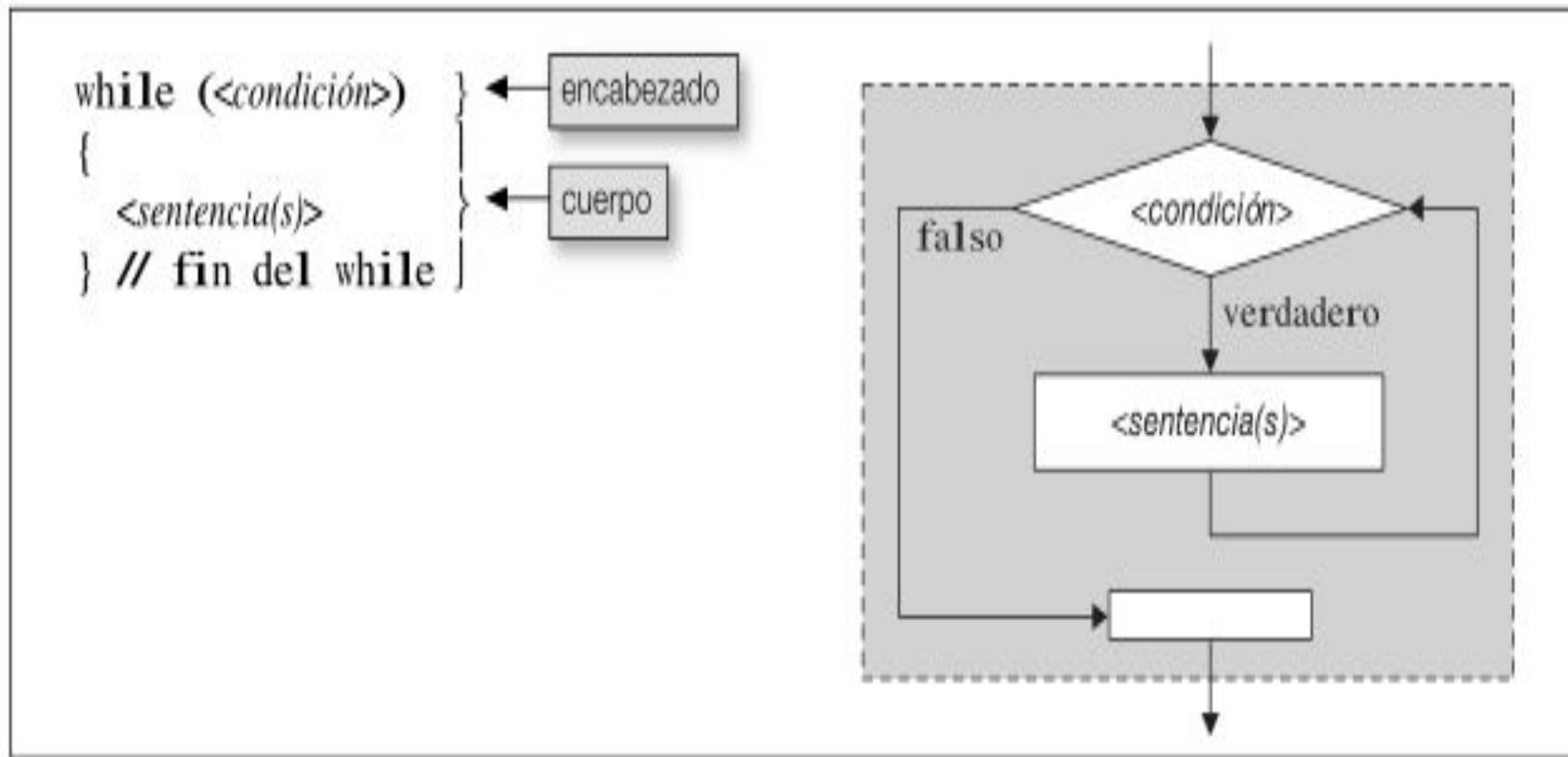
Para construirla se recurre a la utilización de los operadores lógicos los cuales son:

Operador tipo monario: (!) su función es de inversor lógico.

Operador tipo binario: (&&) su función es un AND.

Operador tipo binario: (||) su función es un OR.

# Diagrama tipo while



Sintaxis y semántica del ciclo **while**.

## Iteraciones o contador (pre-incremento y pos-incremento)

```
contadorPost = 0;  
salidaPost = contadorPost;  
contadorPost = contadorPost + 1; //equivale a (contador++) pos-incremento
```

$$\begin{array}{rcl} 0 & = & 0 \\ 1 & = & 0 + 1 \end{array}$$

```
contadorPre = 0;  
contadorPre = contadorPre + 1; //equivale a (++contador) pre-incremento  
salidaPre = contadorPre;
```

$$\begin{array}{rcl} 1 & = & 0 + 1 \\ 1 & = & 1 \end{array}$$

# EJ. Post \_incremento

```
● ● ●

public class WhileControlPost { //comienza en cero

    public static void main(String arags[]) {

        int contador = 0;
        int salida = 0;

        while (contador <= 0) {

            System.out.println("La iteracion es: [" + contador + "]");

            //salida = ++contador; //forma abreviada de contador = contador + 1
            salida = contador;
            contador = contador + 1;

        }

        System.out.println("La iteracion externa: [" + salida + "]");

    }

}
```

# EJ: Pre\_incremente

```
● ● ●

public class WhileControlPre { //comienza en uno

    public static void main(String arags[]) {

        int contador = 0;
        int salida = 0;

        while (contador <= 8) {

            System.out.println("La iteracion es: [" + contador + "]");

            //salida = ++contador;//forma abreviada de contador = contador + 1
            contador = contador + 1;
            salida = contador;

        }

        System.out.println("La iteracion externa: [" + salida + "]");

    }

}
```

# EJ. Preincremento y Postincremento

```
public class PreincrePost {

    public static void main(String args[]) {

        //PRE_INCREMENTO O INCREMENTA ANTES

        int contadorPre = 0; //incrementa antes dando (1)
        /*contadorPre = contadorPre + 1;
        int salidaPre = contadorPre;*/

        int contadorPre1 = ++contadorPre; //tipo (++contadorPre)

        //POST_INCREMENTO O INCREMENTA DESPUES

        int contadorPost = 0; //incrementa despues dando (0)
        /*int salidaPost = contadorPost;
        contadorPost = contadorPost + 1;*/

        int contadorPost1 = contadorPost++; //tipo (contadorPost++)

        System.out.println("[" + contadorPre1 + "]"); //pre_incrementa en uno pasa de 0 a 1
        //System.out.println("Soy preincremento: ["+salidaPre+"]");
        System.out.println("[" + contadorPost1 + "] "); //post_incremento comienza en cero 0
        //System.out.println("Soy postincremento: ["+salidaPost+"]");

    }
}
```

# EJ. Operadores de incremento y decremento



```
public class IncreDecre {  
  
    public static void main(String args[]) {  
        int numero;  
        numero = 0;  
  
        System.out.println(++numero); //incrementa en uno pasa de 0 a 1  
        System.out.println(numero++); //post_incremeto comienza en cero 0  
        //System.out.println(--numero); //decrementa comienza en -1  
        //System.out.println(numero--); //post decrementa comienza en cero 0  
  
    }  
}
```

# Acumulador y contador

**CONTADOR = 0;**

**CONTADOR = CONTADOR + 1;**

**EJ:**

1	-	0	+	1
2	-	1	+	1
3	=	2	+	1
4	-	3	+	1
5	=	4	+	1
6	-	5	+	1

**NUMERO = 5 ;**

**CONTADOR = 0 ;**

**CONTADOR = CONTADOR + 1;**

**CONTADOR < NUMERO**

**ACUMULADOR = 1;**

**ACUMULADOR = ACUMULADOR \* CONTADOR;**

1	=	1	*	1
2	-	1	*	2
6	=	2	*	3
24	-	6	*	4
120	=	24	*	5

**Nota:** el contador cuenta el numero de veces a realizar la operacion y por cada operacion la va acumulando.

$n! = 5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$  multiplicacion acumulada.

**ACUMULADOR = 0;**

**ACUMULADOR = ACUMULADOR + CONTADOR;**

1	-	0	+	1
3	=	1	+	2
6	=	3	+	3
10	=	6	+	4
15	-	10	+	5
21	=	15	+	6

**Nota:** el contador cuenta de 1 a 6

el acumulador acumula valores de la suma de 1 a 6

suma acumulada:  $1+2+3+4+5+6 = 21$

**MULTIPLICADOR O CONTADOR = 1;**

**MULTIPLICANDO = 3;**

**PRODUCTO;**

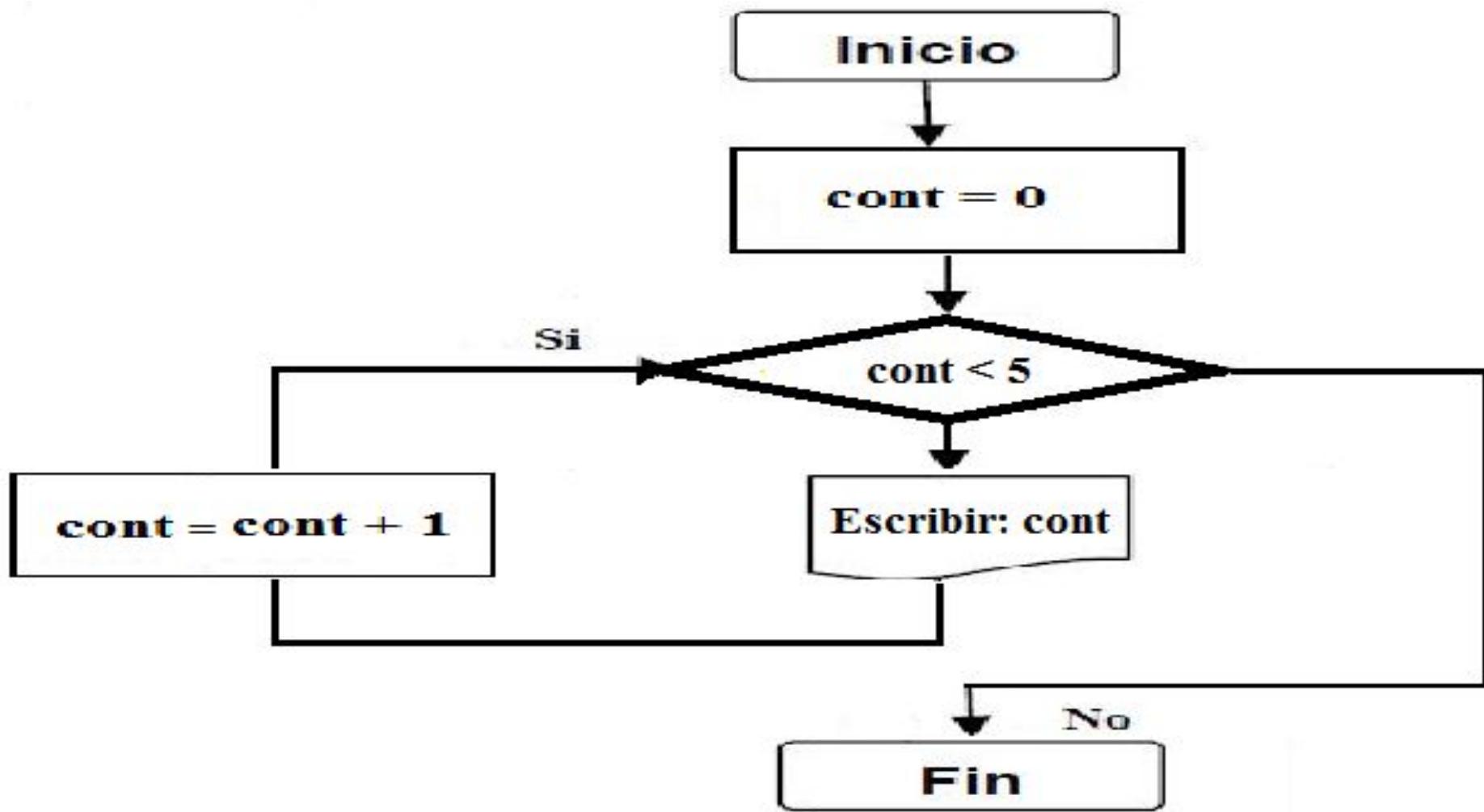
**PRODUCTO = CONTADOR \* MULTIPLICANDO ;**

//	3	=	1	*	3
//	6	=	2	*	3
//	9	=	3	*	3
//	12	=	4	*	3

$$\begin{array}{r}
 3 \\
 \times 1 \\
 \hline
 3
 \end{array}$$

**Multiplicando**  
**Multiplicador (contador)**  
**Producto**

# Contador con While



# EJ-.Bucle While con su iteración (controlado por contador)

```
● ● ●

public class WhileControl {

    public static void main(String args[]) {

        int contador = 0; //inicializamos la iteracion o cuenta en cero

        while (contador < 5) { //mientras iterador sea menor a 10 hacer lo
            //que esta dentro de las llaves osea (true)

            //contador++; //iterador o contador uno a uno (contador = contador + 1)

            System.out.println("La iteracion es: [" + contador + "]");

            contador++; //iterador o contador uno a uno (contador = contador + 1)

        }

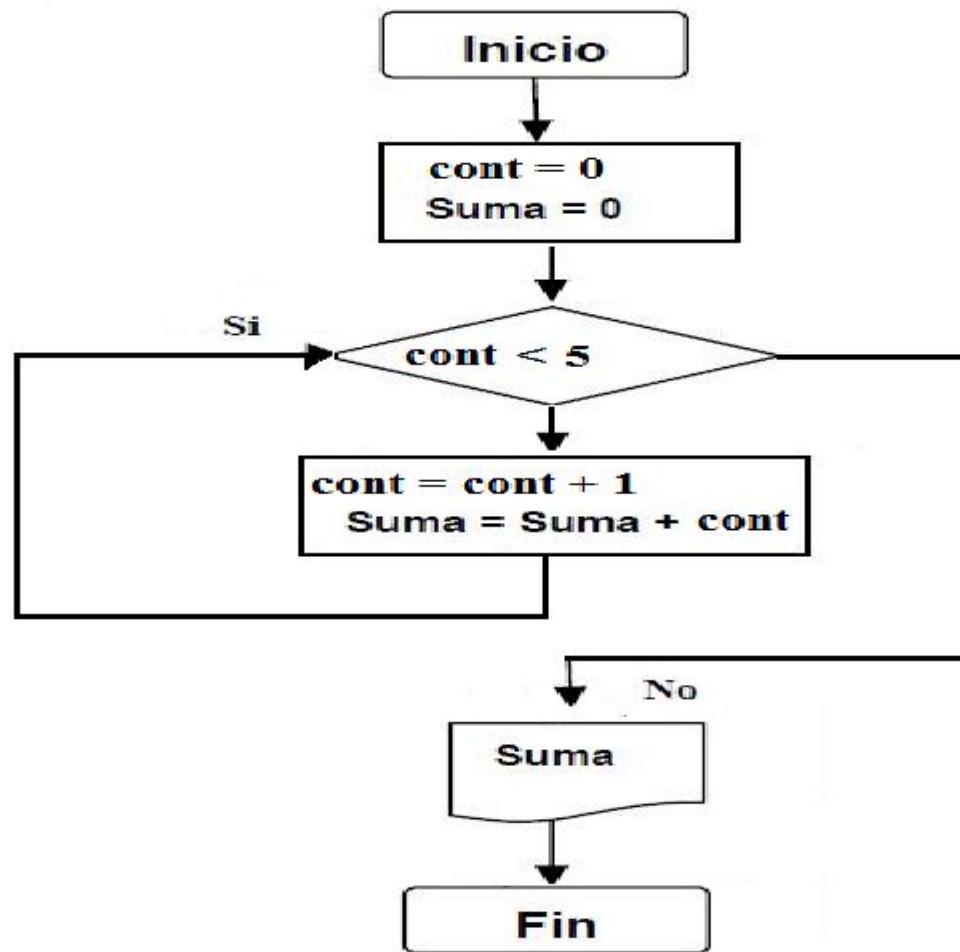
    }

}
```

# Bucle While Infinito

```
public class WhileInfinito {  
  
    public static void main(String arags[]) {  
  
        int contador = 0; //inicializamos la iteracion en cero  
        boolean comparador = contador < 10; //es true por tanto si quiero comparar tengo  
                                         //que hacerlo en la condicion  
  
        while (comparador) { //mientras comparador sea true hacer esto infinitamente  
            //en realidad no estoy comparando nada solo coloco (1)  
            //aqui no hay condicion que finalize el programa  
  
            System.out.println("La iteracion es infinita: [" + contador + "]");  
  
            contador++; //sigue contando y no tiene fin y llenara la memoria  
                         //hasta desbordar  
        }  
    }  
}
```

# Suma Bucle While con acumulador



# Programa while de una suma



```
public class WhileAcum2 { //programa para sumar

    public static void main(String args[]) {

        int contador = 0; //comienza en cero por ser una suma
        int suma = 0;

        while (contador < 5) {

            contador = contador + 1;
            // 0      =      0      +      1
            // 1      =      0      +      1
            // 2      =      1      +      1
            // 3      =      2      +      1
            // 4      =      3      +      1
            // 5      =      4      +      1

            suma = suma + contador;
            // 0      =      0      +      0
            // 1      =      0      +      1
            // 3      =      1      +      2
            // 6      =      3      +      3
            // 10     =      6      +      4
            // 15     =      10     +      5

        }

        System.out.println("La suma de los " + contador + " numeros es: " + suma);
    }
}
```

# EJ. Mostrando contador y acumulador

```
● ● ●

public class WhileAcumP { //programa para sumar

    public static void main(String args[]) {

        int contador = 0; //comienza en cero por ser una suma
        int acumulador = 0;

        while (contador < 5) {

            contador = contador + 1;
            // 0      =      0      +      1
            // 1      =      0      +      1
            // 2      =      1      +      1
            // 3      =      2      +      1
            // 4      =      3      +      1
            // 5      =      4      +      1

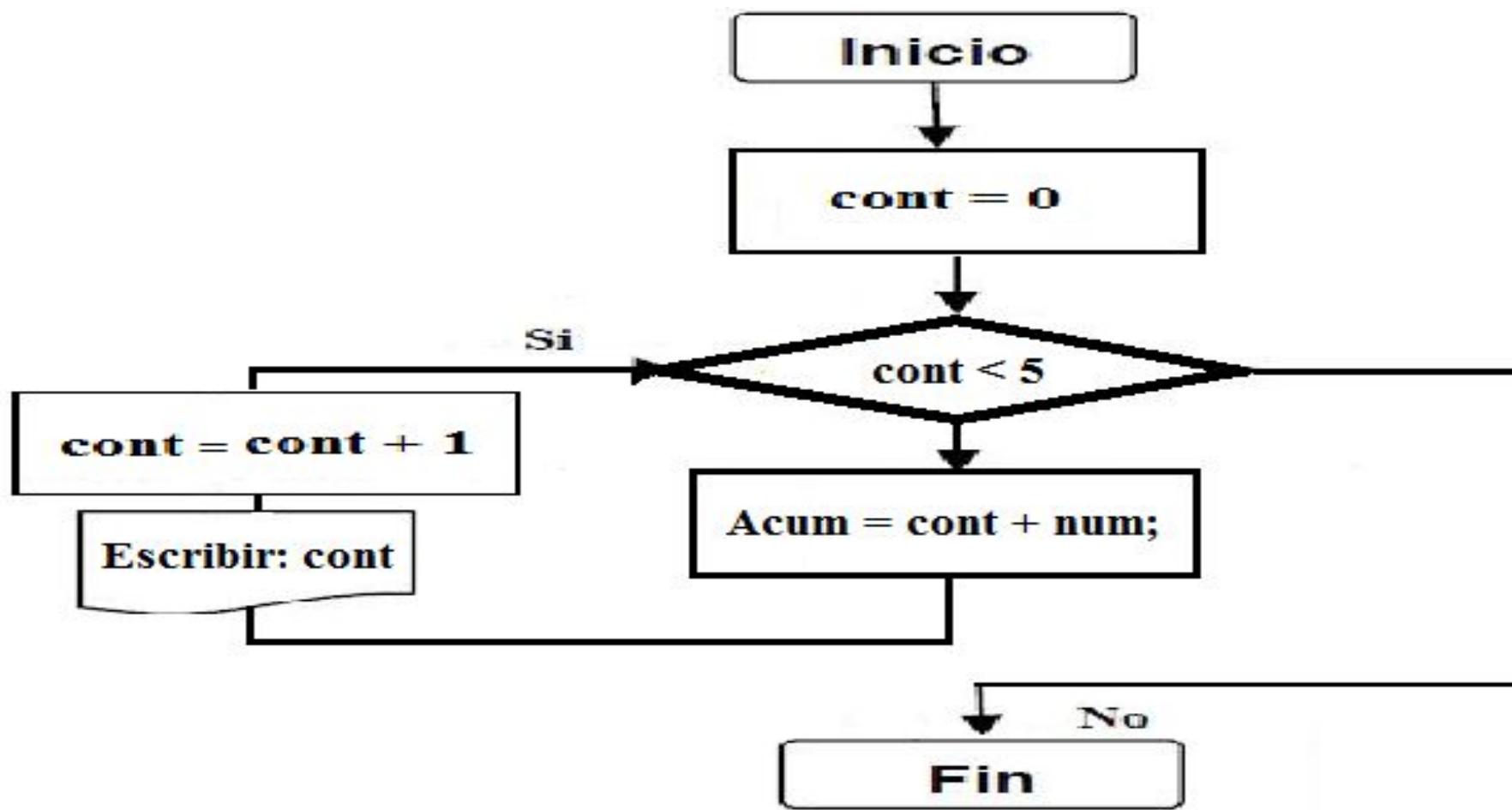
            acumulador = acumulador + contador;

            // 0      =      0      +      0
            // 1      =      0      +      1
            // 3      =      1      +      2
            // 6      =      3      +      3
            // 10     =      6      +      4
            // 15     =      10     +      5

            System.out.println("contador:" + contador + "  acumulador:" + acumulador);

        }
        System.out.println(" ");
        System.out.println("La suma de los " + contador + " numeros es: " + acumulador);
    }
}
```

# Multiplos de tres (3)



# EJ.Tabla de Multiplicar del (3) tres con while

```
● ● ●

public class TablaMult { //ejercicios-clase

    public static void main(String args[]) {

        int contador = 1; //contador comienza en 1 distinto de la suma que comienza en cero 0
        //todo numero multiplicado por cero 0 da cero 0 por eso se comienza de uno 1
        int acumulador;
        int numero = 3; // este valor es fijo

        System.out.println("Tabla de multiplicar del 3: ");

        while (contador <= 4) { //contador termina cuando es menor o igual a 10

            acumulador = contador * numero; //el acumulador guarda el numero de la tabla y contador va de 1 a 10, numero es fijo (3)
            // 3      =      1      *      3
            // 6      =      2      *      3
            // 9      =      3      *      3
            // 12     =      4      *      3

            //contador = contador + 1;//si coloco el contador aqui comenzara a contar desde el dos

            System.out.println("3 x [" + contador + "] = " + acumulador); //aqui sera 3 x [2]= 6 lo cual es falso

            contador = contador + 1; //contador de uno en uno y comienza en uno
        }
    }
}
```

# Múltiplos de 3 con While

Los múltiplos del número tres son 0, 3, 6, 9, ... y así sucesivamente. Como puedes ver, conseguir esta lista de números es bastante sencillo:  $b = n * 3$  con  $n = 0, 1, 2, 3, \dots$  y aunque no es muy común usarlo, 0, por definición, es múltiplo de 3 también. Abajo encontrarás recopilados los primeros 101 números múltiplos de 3:

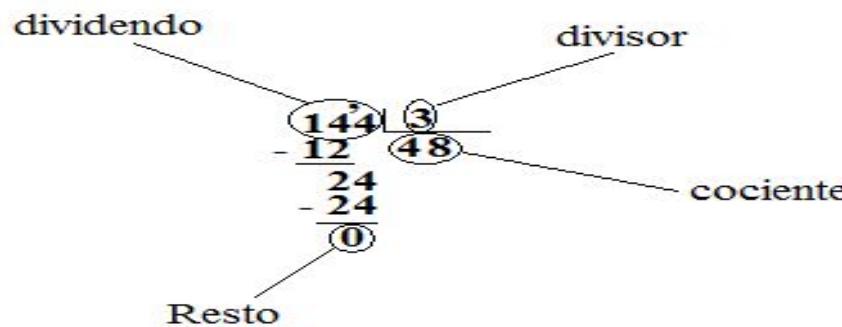
acum = 0;

acum = acum + multiplo; esto seria:  $3 = 0 + 3$ ;  
 $6 = 3 + 3$ ;  
 $9 = 6 + 3$

...

0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99, 102, 105, 108, 111, 114, 117, 120, 123, 126, 129, 132, 135, 138, 141, 144.

Tambien se puede dividir:  $126/3 = 42$ ; otro seria  $144/3 = 48$  dando resto cero (0)



# EJ. Múltiplos (3) tres “suma”



```
public class MultiplosTres {

    public static void main(String args[]) {

        int multiplo = 3;
        int numero = 10; //numero de veces que quiero los multiplos de 3
        int acum = 0;
        int contador = 0;

        System.out.println("he ingresado un multiplo: " + multiplo);

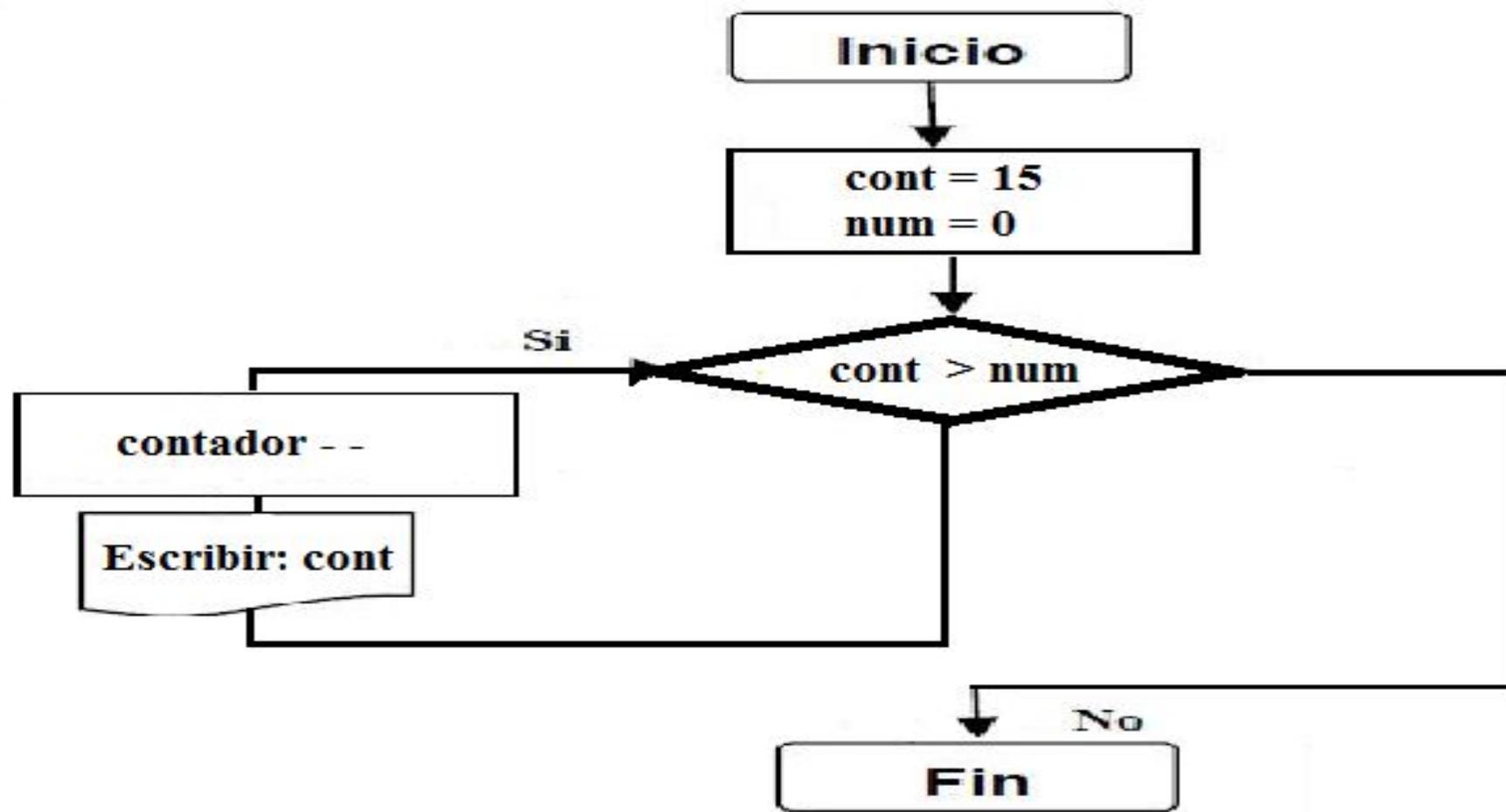
        while (contador < numero) {

            acum = acum + multiplo; //comienzo de cero porque es suma
            ++contador;

            System.out.println("Multiplo de 3: [" + contador + "] = " + acum);

        }
    }
}
```

# Contador Regresivo While



# EJ. Contador Regresivo While

```
● ● ●

public class ContRegresivo {

    public static void main(String args[]) {

        int numero = 0;
        int contador = 15; //contador comienza de 15 y termina en 0

        while (contador > numero) {

            System.out.println("el numero es:[" + contador + "]");
            Contador--;
        }
    }
}
```

# Ejemplo Múltiplos de (3) con Resto

pedir dos números y decir si es múltiplo del otro con resto cero

Ejemplo:  $4\%2 = 0$  múltiplo, pero  $4\%3 = 1$  no es múltiplo resto (1)

```
● ● ●

public class MultiploRestoIf {

    public static void main(String args[]) {

        int dividendo = 144;
        int divisor = 3;
        Int resto;

        if ((resto = dividendo % divisor) = 0) {

            System.out.println("son multiplos resto: " + resto);

        } else {

            System.out.println("no son multiplos resto: " + resto);

        }

    }

}
```

# EJ. Múltiplos de (3) solo con resto cero

```
● ● ●

public class MultiploRestoWhile1 {

    public static void main(String args[]) {

        int contador = 1;
        int n = 144; //numero de veces a iterar

        int resto;

        while (contador < n) {

            if ((resto = contador % 3) = 0) {

                System.out.println("si es multiplo de 3 de 1 a " + n
                    + ": [ " + contador + " ] = resto: " + resto);

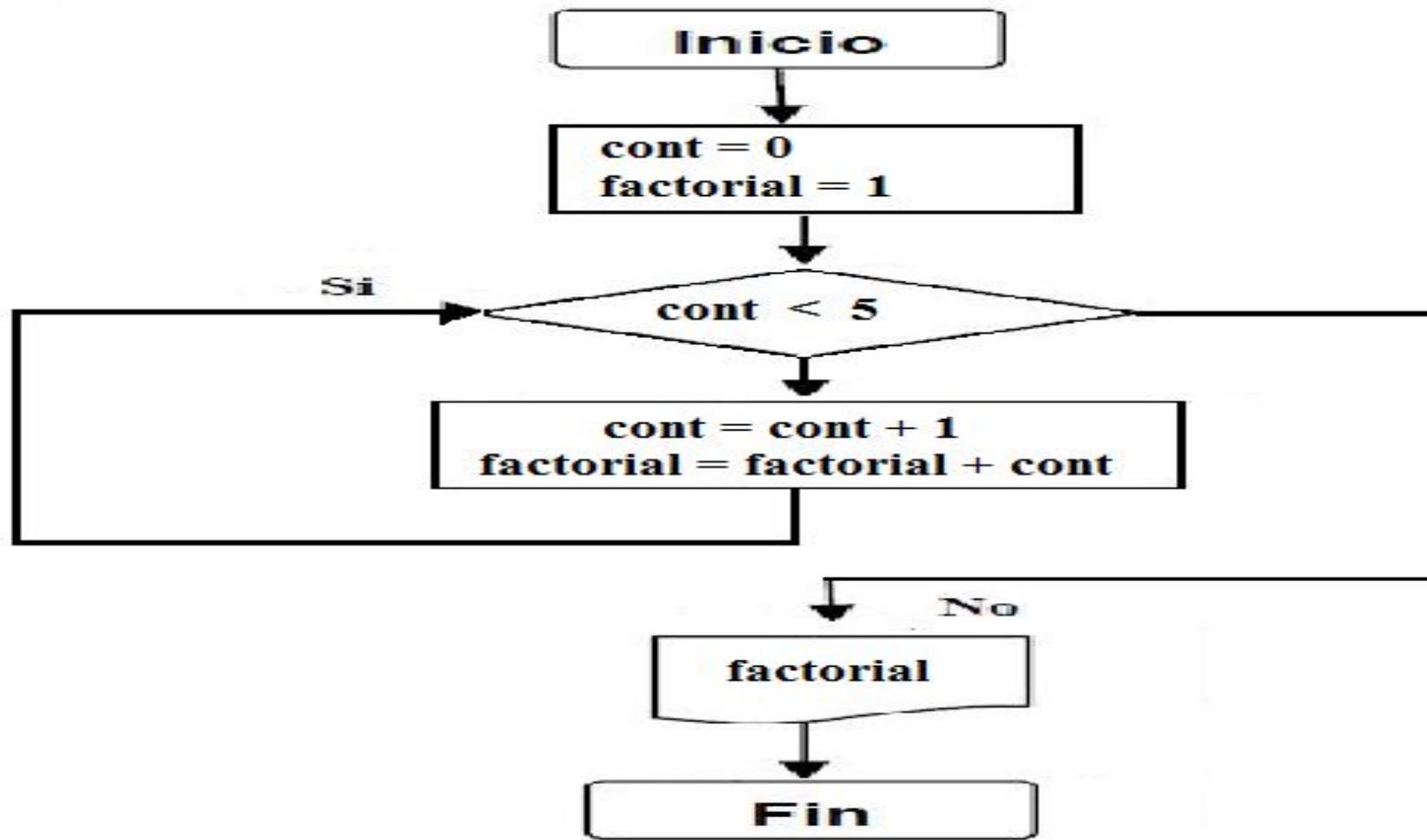
            }

            contador++;
        }
    }
}
```

## EJ. While para múltiplos de 3 con resto cero y distinto de cero

```
public class MultiploRestoWhile {  
  
    public static void main(String args[]) {  
  
        int contador = 1;  
        int n = 144;  
  
        int resto;  
  
        while (contador <= n) {  
  
            if ((resto = contador % 3) == 0) {  
  
                System.out.println("si es multiplo de 3 de 1 a " + n + ": [" + contador + "] = resto: " + resto);  
            } else if ((resto = contador % 3) != 0) {  
  
                System.out.println("no es multiplo de 3 de 1 a " + n + ": [" + contador + "] = resto: " + resto);  
            }  
            contador++;  
        }  
    }  
}
```

# Factorial con While



# Factorial con While

```
public class Factorial {  
  
    public static void main(String args[]) { // 5! = 1×2×3×4×5 = 120  
  
        int factorial = 1; //se coloca 1 por la multiplicacion  
        int contador = 0; //valor inicial del contador  
  
        int numfact = 5;  
        int numero = numfact; //intercambio  
  
        while (contador < numero) {  
  
            contador = contador + 1; //incremento  
            //1 = 0 + 1  
            //2 = 1 + 1  
            //3 = 2 + 1  
  
            factorial = factorial * contador;  
            //1 = 1 * 1  
            //2 = 1 * 2  
            //6 = 2 * 3  
            //24 = 6 * 4  
            //120 = 24 * 5  
  
            //contador = contador + 1;//si lo coloco aqui dara cero  
        }  
  
        System.out.println("El factorial del numero " + numfact + " es: " + factorial);  
    }  
}
```

# Librería Scanner java.util

La clase Scanner está disponible a partir de Java 5 y facilita la lectura de datos en los programas Java.

Primero veremos varios ejemplos de lectura de datos en Java con Scanner y después explicaremos en detalle cómo funciona.

Para utilizar Scanner en el programa tendremos que hacer lo siguiente:

1. Escribir el **import**

La clase Scanner se encuentra en el **paquete java.util** por lo tanto se debe incluir al inicio del programa la instrucción:

**import java.util.Scanner;**

2. Crear un objeto de tipo Scanner

Tenemos que crear un objeto de la clase Scanner asociado al dispositivo de entrada.

Si el dispositivo de entrada es el teclado escribiremos:

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

Se ha creado el objeto **sc** asociado al teclado representado por **System.in**

Una vez hecho esto podemos leer datos por teclado.

# Ejemplo de entrada Scanner

## Ejemplos de lectura:

Para leer podemos usar el método nextXxx() donde Xxx indica en tipo, por ejemplo nextInt() para leer un entero, nextDouble() para leer un double, etc.

### Ejemplo de lectura por teclado de un número **tipo entero**:

```
int n;  
System.out.print("Introduzca un número entero: ");  
n = sc.nextInt();
```

### Ejemplo de lectura de un número de **tipo double**:

```
double x;  
System.out.print("Introduzca número con coma de tipo double: ");  
x = sc.nextDouble();
```

### Ejemplo de lectura de una cadena de **tipo caracteres**:

```
String s;  
System.out.print("Introduzca texto: ");  
s = sc.nextLine();
```

# Metodos para ingresar datos por Scanner

METODO	DESCRIPCIÓN
<b>nextXxx()</b>	Devuelve el siguiente token como un tipo básico. Xxx es el tipo. Por ejemplo, <code>nextInt()</code> para leer un entero, <code>nextDouble</code> para leer un double, etc.
<b>next()</b>	Devuelve el siguiente token como un String.
<b>nextLine()</b>	Devuelve la línea entera como un String. Elimina el final <code>\n</code> del buffer
<b>hasNext()</b>	Devuelve un boolean. Indica si existe o no un siguiente token para leer.
<b>hasNextXxx()</b>	Devuelve un boolean. Indica si existe o no un siguiente token del tipo especificado en Xxx, por ejemplo <code>hasNextDouble()</code>
<b>useDelimiter(String)</b>	Establece un nuevo delimitador de token.

# Limpiar el Buffer de Entrada

## Cómo limpiar el buffer de entrada en Java

Cuando en un programa se leen por teclado datos numéricos y datos de tipo carácter o String debemos tener en cuenta que al introducir los datos y pulsar intro estamos también introduciendo en el buffer de entrada el intro (tecla intro).

Buffer de entrada si se introduce un 5: **5\n**

En esta situación, la instrucción:

```
n = sc.nextInt();
```

Asignan el valor **5** pero el intro permanece en el buffer

Buffer de entrada después de leer el entero: **\n**

Si ahora se pide que se introduzca por teclado una cadena de caracteres:

```
System.out.print("Introduzca su nombre: ");
nombre = sc.nextLine(); //leer un String
```

Es decir, cuando en un programa introducimos un dato y pulsamos el intro como final de entrada, el carácter intro también pasa al buffer de entrada.

El método `nextLine()` extrae del buffer de entrada todos los caracteres hasta llegar a un intro y elimina el intro del buffer. En este caso asigna una cadena vacía a la variable `nombre` y limpia el intro. Esto provoca que el programa no funcione correctamente, ya que no se detiene para que se introduzca el nombre.

Solución: Se debe limpiar el buffer de entrada si se van a leer datos de tipo carácter a continuación de la lectura de datos numéricos.

La forma más sencilla de limpiar el buffer de entrada en Java es ejecutar la instrucción: `sc.nextLine();`

# Ejemplo Scanner por teclado

```
● ● ●

import java.util.Scanner;

public class ScaneoTeclado { //con un double o float es coma (,) pero en consola pero en netbean es con punto (.)

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in); //crear un objeto Scanner

        String nombre;
        double radio;
        float radio1;
        int n;

        System.out.print("Introduzca su nombre: ");
        nombre = sc.nextLine(); //leer un String
        System.out.println("Hola " + nombre + "!!!");

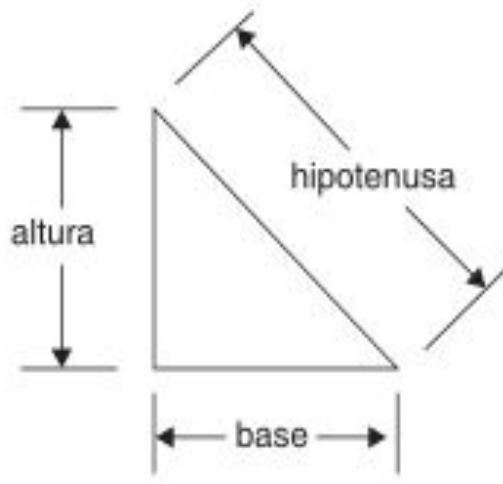
        System.out.print("Introduzca el radio de la circunferencia: ");
        radio = sc.nextDouble(); //leer un double colocando coma
        System.out.println("Longitud de la circunferencia en Double: " + 2 * Math.PI * radio);

        System.out.print("Introduzca un número entero para hallar el cuadrado: ");
        n = sc.nextInt(); //leer un entero
        System.out.println("El cuadrado es: " + Math.pow(n, 2));

        System.out.print("Introduzca el radio de la circunferencia: ");
        radio1 = sc.nextFloat(); //leer un flotante colocando coma
        System.out.println("Longitud de la circunferencia en float: " + 2 * Math.PI * radio1);

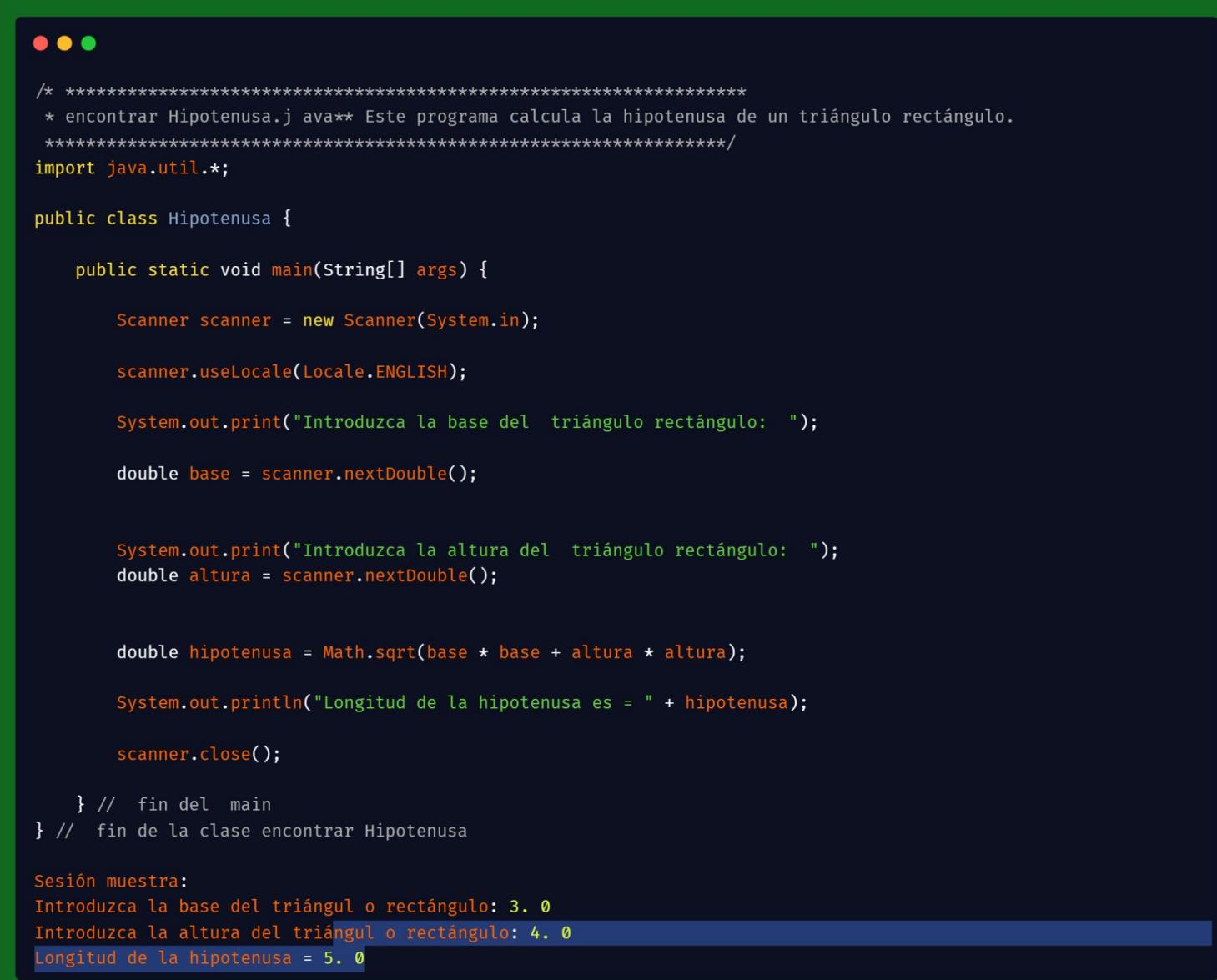
    }
}
```

# Hallando la Hipotenusa



$$\text{hipotenusa} = \sqrt{\text{altura}^2 + \text{base}^2}$$

# Ingreso por teclado (Hipotenusa)



```
/* ****
 * encontrar Hipotenusa.j ava** Este programa calcula la hipotenusa de un triángulo rectángulo.
 ****/
import java.util.*;

public class Hipotenusa {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        scanner.useLocale(Locale.ENGLISH);

        System.out.print("Introduzca la base del triángulo rectángulo: ");

        double base = scanner.nextDouble();

        System.out.print("Introduzca la altura del triángulo rectángulo: ");
        double altura = scanner.nextDouble();

        double hipotenusa = Math.sqrt(base * base + altura * altura);

        System.out.println("Longitud de la hipotenusa es = " + hipotenusa);

        scanner.close();

    } // fin del main
} // fin de la clase encontrar Hipotenusa

Sesión muestra:
Introduzca la base del triángulo rectángulo: 3. 0
Introduzca la altura del triángulo rectángulo: 4. 0
Longitud de la hipotenusa = 5. 0
```

# Cambio a punto y no coma

nextDouble() usa los puntos decimales y separadores de miles propios del idioma. En español el punto es separador de miles y la coma es los decimales.

Si quieras cambiarlo, a la clase Scanner llama al método useLocale() y pásale un Locale inglés, creo que sería algo como esto

Código java:

```
recibir.useLocale(Locale.ENGLISH)
```

# Scanneo sin Coma

```
import java.util.*;  
  
public class ScannerDemo {  
  
    public static void main(String[] args) {  
  
        Scanner scanner = new Scanner(System.in);  
  
        scanner.useLocale(Locale.ENGLISH);  
  
        double Num = scanner.nextDouble();  
  
        System.out.println(" numero " +Num);  
  
        scanner.close();  
    }  
}
```

# Clase Scanner de Java.util (ingreso por teclado)

La clase **java.util.Scanner** proporciona una serie de métodos para realizar la lectura de datos desde un dispositivo de entrada o archivo, tanto en forma de cadena de caracteres como en cualquier tipo básico.

```
package paquete;
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        String nombre;
        int edad;
        Scanner teclado = new Scanner(System.in);
        System.out.print("Escriba su nombre:");
        nombre = teclado.nextLine();
        System.out.print("Escriba su edad:");
        edad = teclado.nextInt();
        System.out.println("Hola "+nombre+" tienes "+edad+" años");
    } //fin del void main
} //fin de la clase Principal
```

1 Importamos la clase **Scanner** del paquete **java.util**

2 Creamos una variable llamada **teclado** de tipo **Scanner**

3 Asignamos el valor de las lecturas a las variables para que lo almacenen

# While controlado por Centinela

(cuando desconocemos el número de repeticiones para detener el ciclo)

```
import java.util.Scanner;

public class Centinela { //ciclos controlados por valor centinela que sirve
para terminar el bucle y que no se haga infinito

    public static void main(String args[]) {

        /*entrada de datos numericos, centinela -1*/

        Scanner entrada = new Scanner(System.in);

        final int centinela = -1; //cuando introduzca "-1" saldra del bucle
        int suma = 0, cuenta = 0;

        System.out.print("Introduzca primera nota: ");

        int nota = entrada.nextInt();

        while (nota != centinela) {

            cuenta++;

            suma = suma + nota;

            System.out.print("Introduzca siguiente nota: ");

            nota = entrada.nextInt();
        }

        System.out.println("Final");
    }
}
```

# Centinela con While – if y un break

```
import java.util.Scanner;
public class CentinelaIf {

    public static void main(String args[]) {

        int cont = 0;
        final int centinela = -1;
        int acum = 0;
        int nota = 0;
        Scanner sc = new Scanner(System.in);

        System.out.println("Introduzca un nota: ");
        while (cont < 5) {
            System.out.print("Introduzca la nota [" + (cont + 1) + "] :");
            nota = sc.nextInt();
            acum = acum + nota;

            if (nota == centinela) {

                break;
            }
            ++cont;
        }
        System.out.println("la nota final es: " + acum);
    }
}
```

# Entrar 10 valores y sacar Promedio con While



```
import java.util.Scanner;

public class WhileProm {

    public static void main(String arags[]) {

        Scanner input = new Scanner(System.in);

        int total = 0;
        int numero;
        int promedio;
        int contador = 0;

        while (contador < 10) { //ciclo iterador

            System.out.println("Ingresar un numero: ");

            numero = input.nextInt(); //ingresar 10 numeros

            total = total + numero;

            contador++;

        }

        promedio = total / 10;

        System.out.println("su promedio es: " + promedio);
    }
}
```

# Clase misterio N°1

```
public class Misterio {  
  
    public static void main(String args[]) {  
  
        int y;  
        int x = 1;  
        int total = 0;  
  
        while (x <= 10) {  
  
            y = x * x;  
  
            System.out.println(y);  
  
            total = total + y;  
  
            ++x;  
        }  
  
        System.out.printf("El total es: %d", total);  
    }  
}
```

# Operador Ternario sin While

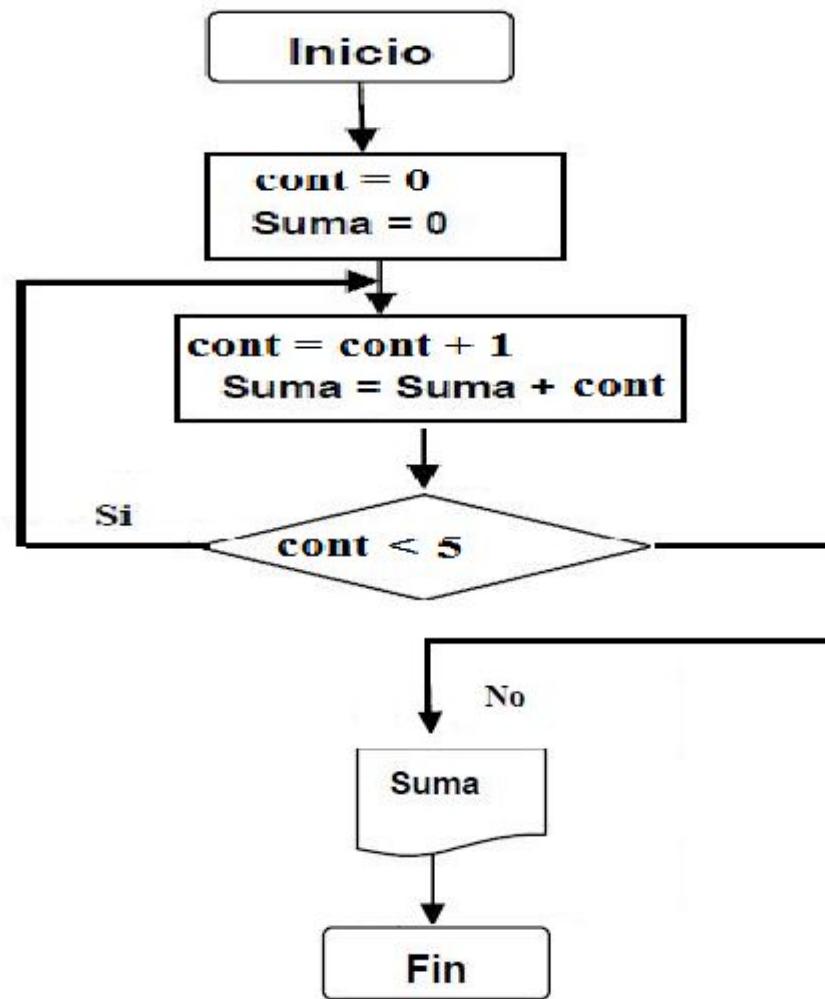
```
public class Ternario1 {  
  
    public static void main(String args[]) { //se aloja en una variable (total)  
  
        int cantidad = 10;  
        int buscamos = 11;  
        String total = "";  
  
        total = (buscamos > cantidad) ? "cantidad alta: " + buscamos : "cantidad baja: " + cantidad;  
  
        System.out.println(total);  
  
    }  
}
```

# Clase misterio N°2 While y operador ternario



```
public class Misterio2 { //saldra intercalado + y *  
  
public static void main(String args[]) {  
  
    int cuenta = 1;  
  
    while (cuenta <= 10) {  
  
        System.out.println(cuenta % 2 == 1 ? " *** * " : "++ ++ +");  
  
        ++cuenta;  
    }  
}
```

# Suma con acumulador y contador (mostrada fuera del ciclo do-While)



# Bucle do-While contador



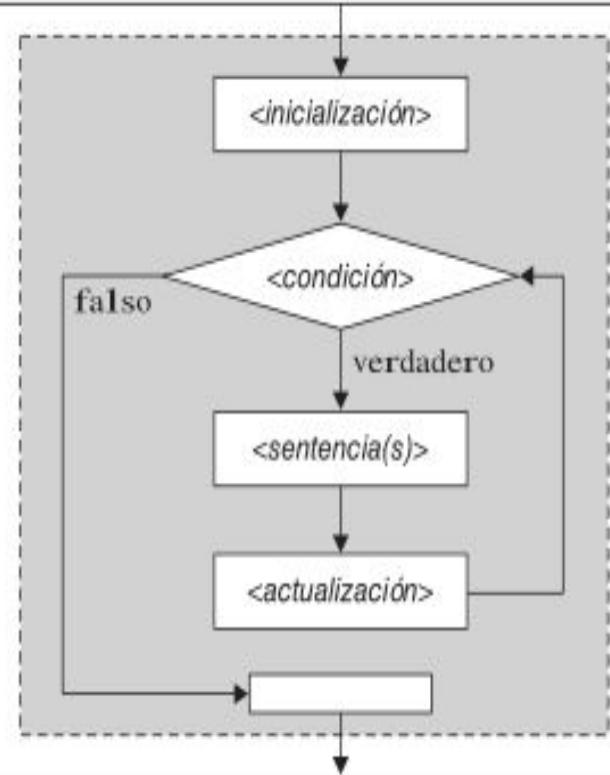
```
public class DoWhile {  
  
    public static void main(String args[]) {  
  
        int contador = 0;  
  
        do {  
  
            System.out.println("Numero: [" + contador + "]");  
  
            contador++;  
  
        } while (contador <= 10);  
    }  
}
```

# Bucle do-While contador-acumulador

```
public class DoWhileSuma {  
  
    public static void main(String args[]) {  
  
        int contador = 0;  
        int suma = 0;  
  
        do {  
  
            System.out.println("Numero: [" + contador + "] la suma es:" + suma);  
  
            contador = contador + 1;  
  
            suma = suma + contador;  
  
        } while (contador < 10);  
    }  
}
```

# Bucle for y sentencias

```
for (<inicialización>; <condición>;  
<actualización>)  
{  
    <sentencia(s)>  
}
```



Sintaxis y semántica del ciclo for.

# Bucle “for” como contador

```
public class ForIterator1 {  
  
    public static void main(String args[]) {  
  
        int contador;  
  
        for (contador = 0; contador <= 5; contador++) {  
  
            System.out.println("contador:[" + contador + "]"); //muestra el contador de 0 a 5  
        }  
    }  
}
```

# Bucle for con dato incorporado



```
public class ForInt {  
  
    public static void main(String args[]) {  
  
        for (int contador = 0; contador <= 5; ++contador) {  
  
            System.out.println("bucle for Numero:[" + contador + "]");  
        }  
    }  
}
```

# For impresión interna y externa

```
public class ForIntExt {  
  
    public static void main(String args[]) {  
  
        int contador; //declaracion variable  
  
        for (contador = 0; contador <= 1; contador++) { //1º condiciona (0<=1)si  
            System.out.println("repeticion numero :" + contador);  
            //2º imprime el cero (0) luego incrementa pasando a (1)  
            //pregunta (1<=1) si es igual, e imprime (1) incrementa en (2)  
            //sale como valor final (2) condiona y se corta por ser mayor  
    }  
  
    System.out.println("valor final :" + contador); //da el valor final  
}  
}
```

# Bucle for contador y acumulador

```
● ● ●

public class ForIterator {

    public static void main(String args[]) {

        int contador;
        int suma = 0;

        for (contador = 0; contador <= 5; contador++) {

            System.out.println("contador:[" + contador + "]"); //muestra el contador
            suma = suma + contador;
        }

        System.out.println("La suma del acumulador es:" + suma);
    }
}
```

# Tablas de Multiplicar del uno al diez “for” anidado

contador1 = 1; FILA = 10;

Iteracion FILA va de 1 a 10 for(contador1 <= FILA)

contador2 = 1; COLUMNA = 10;  
Iteracion COLUMNA va de 1 a 10  
for(contador2 <= COLUMNA)

	FILA	FILA	FILA	FILA	FILA			FILA
	1	2	3	4	5			10
COLUMNA 1	1 x 1	2 x 1	3 x 1	4 x 1	5 x 1			10 x 1
COLUMNA 2	1 x 2	2 x 2						...
COLUMNA 3	1 x 3	2 x 3						...
4	1 x 4	2 x 4						...
5	1 x 5	2 x 5						...
6	1 x 6	2 x 6						...
7	1 x 7	2 x 7						...
8	1 x 8	2 x 8						...
9	1 x 9	2 x 9						...
COLUMNA 10	1 x 10	2 x 10	3 x 10	4 x 10	5 x 10			10 x 10

# For Anidado (tablas del 1 al 10)

```
public class ForDoble {  
  
    public static void main(String args[]) {  
  
        int respuesta;  
        Int FILA = 10; //si coloco uno 1 solo saldra la fila con uno  
        Int COLUMNA = 10; // si coloco 10 recorrerá de 1 a 10 en columna  
  
        for (int contador1 = 1; icontador1 <= FILA; contador1++) {  
            for (int contador2 = 1; contador2 <= COLUMNA; contador2++) {  
  
                respuesta = contador1 * contador2;  
                System.out.println(contador1 + " x " + contador2 + " = " + respuesta);  
  
            }  
  
            System.out.println("FIN!!!");  
        }  
  
    }  
}
```

# For con Break



```
public class ForBreak {  
  
    public static void main(String args[]) {  
  
        for (int contador = 1; contador <= 10; contador++) {  
  
            System.out.println("El valor de contador es :" + contador);  
  
            if (contador == 5) {  
  
                break; //hace un pare u stop en 5  
  
            }  
  
            //System.out.println("El valor de contador es :" + contador); //lo hace en 4  
  
        }  
    }  
}
```

# For con Continue

```
public class ForBreak {  
  
    public static void main(String args[]) {  
  
        for (int contador = 1; contador <= 10; contador++) {  
  
            System.out.println("El valor de contador es :" + contador); //incluye el 5  
  
            if (contador == 5) {  
  
                continue; //salta el valor en condicion  
            }  
  
            System.out.println("El valor de contador es :" + contador); //cuenta hasta 4 salta a 6  
            //no incluye el 5  
        }  
    }  
}
```

# Switch Case

La sentencia **switch** funciona de manera similar a como lo hace la forma “if, else if” de la sentencia **i f**, en el sentido que permite seguir una de varias rutas; pero una **diferencia** clave entre la sentencia **switch** y la sentencia **i f** es que la determinación sobre la ruta a seguir en la sentencia **switch** **se basa** en un **simple valor**. (En una sentencia **i f**, la determinación sobre la ruta a seguir se basa en múltiples condiciones, una para cada ruta.) Tener la determinación con base en un simple valor puede resultar en una implementación más compacta y más entendible.

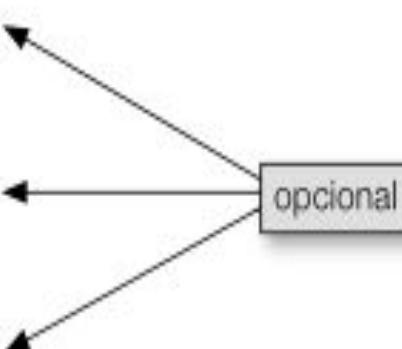
Cuando se ejecuta una sentencia **switch**, el control salta a la constante **case** que concuerda con el valor de expresión de control, y la computadora ejecuta todas las sentencias subsiguientes hasta toparse con la sentencia **break**.

La sentencia **break** provoca que el control **salga** de la sentencia **switch** (debajo del paréntesis de llave de cierre).

Si **no hay** constantes **case** que concuerden con el valor de control de la expresión, entonces el control salta a la etiqueta **default** (si es que la hay) o fuera de la sentencia **switch** si no hay sentencia **default**.

# Switch Case o selección

```
switch (<expresión de control>)
{
    case <constante>:
        <sentencia(s)>;
        break;
    case <constante>:
        <sentencia(s)>;
        break;
    .
    .
    .
    default:
        <sentencia(s)>;
    } // fin del switch
```



The diagram illustrates the optional nature of the curly braces in the switch statement. Three arrows point from the closing brace of each case block to a rectangular box labeled "opcional".

Sintaxis de la sentencia **switch**.

# Switch Case Sentencia

```
public class SwitchCase {  
  
    public static void main(String args[]) {  
  
        char x = 'a'; //entrada de seleccion (a y b)  
  
        switch (x) { //switch inicio  
  
            case 'a':  
            {  
  
                System.out.println("Esta es la opcion 0");  
  
                break;  
  
            }  
            case 'b':  
            {  
  
                System.out.println("Esta es la opcion 1");  
  
                break;  
  
            }  
            default:  
            {  
                System.out.println("Esta opcion es la opcion por defecto");  
            }  
  
        } //switch final  
  
    }  
  
}
```

# Do while con opcion switch

```
import java.util.Scanner;

public class DoWhileSwitch {

    public static void main(String args[]) {

        int opcion; //opcion elejida del menu

        Scanner teclado = new Scanner(System.in);

        do { //menu dentro del bucle

            System.out.println("1. Numeros del 1 al 5"); //solo como mostrario
            System.out.print("Introduce una opcion: ");

            opcion = teclado.nextInt(); //introducimos la opcion

        } while (opcion > 2); //se repite si es mayor a 2

        switch (opcion) {

            Case 1:
                int contador = 0;
                int num = 5;

                while (contador < num) {

                    System.out.println("bucle while Numero:[" + contador + "]");

                    contador = contador + 1;

                }
                break;

            default:
                System.out.println("Elige una opcion correcta.");
        }
    }
}
```