Tema 3. Estructuras de Control

1. INTRODUCCIÓN

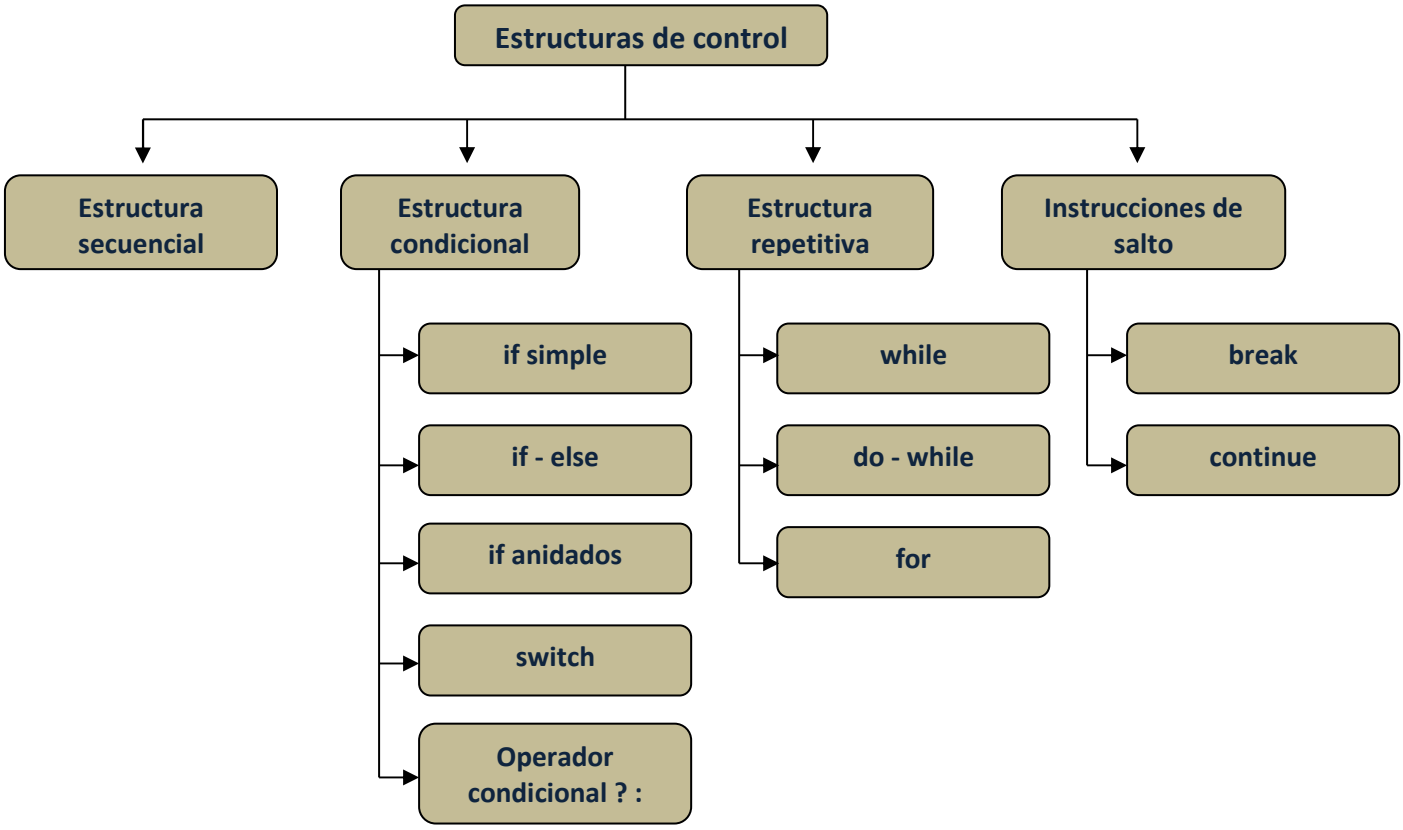
Los programas contienen instrucciones que se ejecutan generalmente una a continuación de la otra según la secuencia en la que el programador ha escrito el código. Sin embargo, hay ocasiones en las que es necesario romper esta secuencia de ejecución para hacer que una serie de instrucciones se ejecuten o no dependiendo de una determinada condición o hacer que una serie de instrucciones se repitan un número determinado de veces.

Las estructuras de control permiten modificar el orden natural de ejecución de un programa. Mediante ellas podemos conseguir que el flujo de ejecución de las instrucciones sea el natural o varíe según se cumpla o no una condición o que un bloque de instrucciones se repitan dependiendo de que una condición se cumpla o no.

Las estructuras de control tienen las siguientes características:

- Tienen un único punto de entrada y un único punto de salida.
- Están compuestas por instrucciones o por otras estructuras de control.

Las estructuras de control en Java son las siguientes:



2. ESTRUCTURA SECUENCIAL

Las instrucciones de un programa se ejecutan por defecto en orden secuencial. Esto significa que las instrucciones se ejecutan en secuencia, una después de otra, en el orden en que aparecen escritas dentro del programa.

La estructura secuencial es el orden natural de ejecución. Las instrucciones que componen esta estructura se ejecutan en orden una a continuación de la otra.

La mayoría de las instrucciones están separadas por el carácter punto y coma (;).

Las instrucciones se suelen agrupar en bloques.

El bloque de instrucciones se define por el carácter llave de apertura **{** para marcar el inicio del mismo, y el carácter llave de cierre **}** para marcar el final.

Ejemplo:

```
{
   int n = 1;
   int m = 0;
   m = n++ ;
   System.out.println(m);
   System.out.println(n);
}
```

Si el bloque de instrucciones está constituido por una única instrucción no es obligatorio el uso de las llaves de apertura y cierre { }, aunque es recomendable utilizarlas.

Ejemplo: programa que lee dos números por teclado y los muestra por pantalla. En este caso hay un solo bloque de instrucciones contenidas entre las llaves de apertura y cierre del método main. Las instrucciones se ejecutan en el mismo orden que aparecen escritas.

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

    public static void main(String[] args){ //--> inicio del método main
        //declaración de variables
        int n1, n2;
        Scanner sc = new Scanner(System.in);
        //leer el primer número
        System.out.println("Introduce un número entero: ");
        n1 = sc.nextInt();      //lee un entero por teclado
        //leer el segundo número
        System.out.println("Introduce otro número entero: ");
        n2 = sc.nextInt();      //lee un entero por teclado

        //mostrar resultado
        System.out.println("Ha introducido los números: " + n1 + " y " + n2);
    } //--> final del método main
}
```

Ejemplo: programa que lee por teclado dos valores de tipo double y a continuación muestra su suma, resta y multiplicación. En este caso el programa también está formado por un único bloque de instrucciones que se ejecutan secuencialmente dentro del método main.

```
/*
 * Mostrar la suma, resta y multiplicación de dos valores
 * de tipo double que se introducen por teclado.
 */
import java.util.Scanner;
public class Secuencial2 {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        double numero1, numero2;
        System.out.println("Introduce el primer número:");
        numero1 = sc.nextDouble();
        System.out.println("Introduce el segundo número:");
        numero2 = sc.nextDouble();
        System.out.println("Números introducido: " + numero1 + "  " + numero2);
        System.out.println(numero1 + " + " + numero2 + " = " + (numero1+numero2));
        System.out.println(numero1 + " - " + numero2 + " = " + (numero1-numero2));
        System.out.println(numero1 + " * " + numero2 + " = " + numero1*numero2);
    }
}
```

3. ESTRUCTURA CONDICIONAL O SELECTIVA

Es una de las estructuras que permiten modificar el orden de ejecución de las instrucciones del programa.

Una estructura condicional determina si se ejecutan unas acciones u otras según se cumpla o no una determinada condición.

La condición que se comprueba para decidir si unas instrucciones se ejecutan o no debe ser una **expresión booleana** es decir debe dar como resultado un valor booleano **true ó false**.

En java se implementan mediante:

- Instrucciones if.
- Instrucción switch.
- Operador condicional ? :

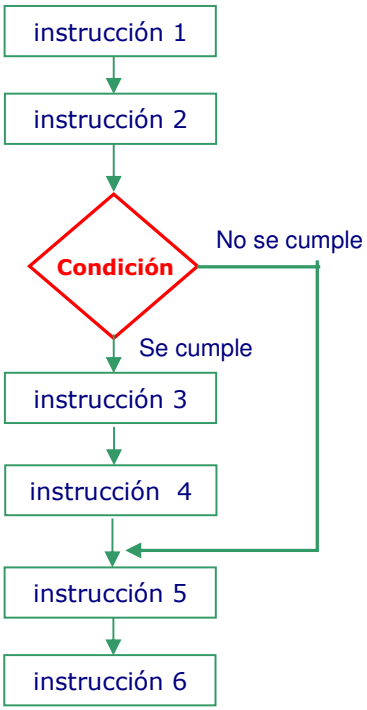
3.1 INSTRUCCION if

La instrucción if puede ser del tipo:

- Condicional simple: if
- Condicional doble: if ... else ...
- Condicional multiple o if anidados: if .. else if ..

ESTRUCTURA CONDICIONAL SIMPLE

Como su nombre indica, es la estructura condicional más sencilla en Java. La usaremos para decidir si una instrucción o bloque de instrucciones se ejecuta o no dependiendo de una condición. Se evalúa la condición y si se cumple se ejecuta. Si no se cumple la condición, se salta dicho grupo de instrucciones.



Sintaxis:

```
instrucción 1;
instrucción 2;
if(condición){ //inicio de la condición
     instrucción 3;
     instrucción 4;
}                    //fin de la condición
instrucción 5;
instrucción 6;
```

En este caso se ejecutan las instrucciones 1 y 2 y a continuación se evalúa la condición. Si se cumple, se ejecutan las instrucciones 3 y 4 y a continuación la 5 y 6. Si la condición no se cumple, las instrucciones 3 y 4 no se ejecutan, siguiendo la ejecución del programa por las instrucciones 5 y 6.

Ejemplo: Programa que pide que se introduzca una nota por teclado y muestra dos mensajes si la nota es mayor o igual a 5.

```
/*
 * Programa que pide una nota por teclado y muestra dos mensajes si la nota es
 * mayor o igual que 5
 */
import java.util.Scanner;
public class CondicionalSimple1 {
    public static void main( String[] args ){
        Scanner sc = new Scanner( System.in );
        System.out.print("Nota: ");
        int nota = sc.nextInt();
        if (nota >= 5){//----------------------inicio de la condición
            System.out.println("Enhorabuena!!");
            System.out.println("Has aprobado");
        }//-------------------------------------fin de la condición
        System.out.println("Hasta Pronto!");
    }
}
```

En este ejemplo si se introduce un número >= 5 el programa mostrará:

```
Enhorabuena!!
Has aprobado
Hasta Pronto!
```

Si introducimos un valor < 5 mostrará:

```
Hasta Pronto!
```

Si un bloque de instrucciones contiene **una sola instrucción** no es necesario escribir las llaves { } aunque para evitar confusiones se recomienda escribir las llaves siempre.

Supongamos que en el ejemplo anterior la instrucción if la escribimos así:

```
        if (nota >= 5){//----------------------inicio de la condición
            System.out.println("Enhorabuena!!");
        }//-------------------------------------fin de la condición
```

Como el bloque de instrucciones dentro del if ahora solo contiene una instrucción, las llaves se pueden omitir. En ese caso el programa quedaría así:

```
/*
 * Programa que pide una nota por teclado y muestra un mensaje si la nota es
 * mayor o igual que 5
 */
import java.util.Scanner;
public class CondicionalSimple1 {
    public static void main( String[] args ){
        Scanner sc = new Scanner( System.in );
        System.out.print("Nota: ");
        int nota = sc.nextInt();
        if (nota >= 5)
            System.out.println("Enhorabuena!!");
        System.out.println("Hasta Pronto!");
    }
}
```

Ahora si se introduce un número >= 5 el programa mostrará:

```
Enhorabuena!!
Hasta Pronto!
```

Si introducimos un valor < 5 mostrará:

```
Hasta Pronto!
```

No escribir las llaves en estos casos puede conducir a un código confuso de entender por lo que es siempre recomendable escribir las llaves incluso en los casos en los que no sea necesario.

Ten en cuenta que el código no es obligatorio escribirlo con ese formato en el que las instrucciones están tabuladas. Aunque no es un buen estilo de programación, se podría hacer escrito así:

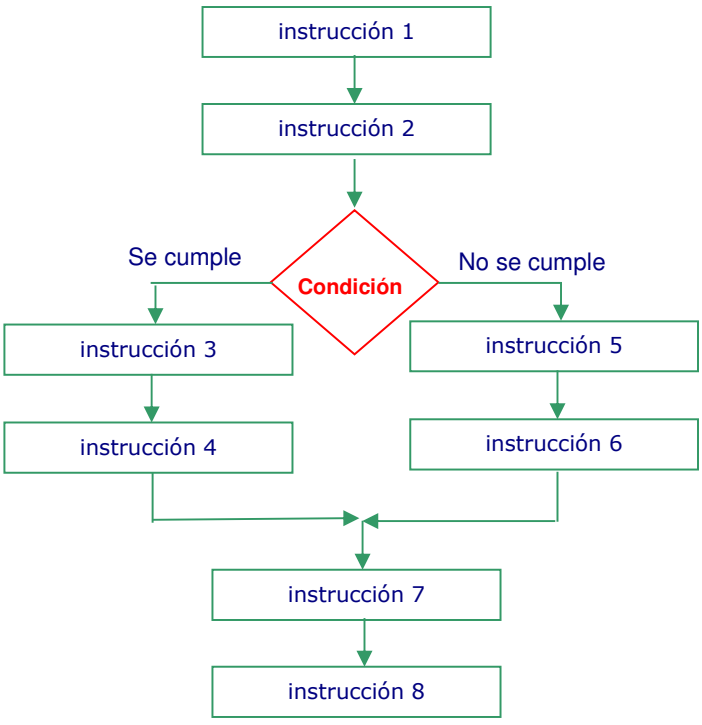
```
if (nota >= 5)
System.out.println("Enhorabuena!!");
System.out.println("Hasta Pronto!");
```

Escribiéndolo así puede llevar a confusion al no estar claro a simple vista donde comienza el bloque if y donde acaba. Por eso es recomendable escribir las llaves siempre.

ESTRUCTURA CONDICIONAL DOBLE

Mediante esta estructura de control el flujo de ejecución del programa puede seguir dos caminos distintos dependiendo del valor de una condición.

Se evalúa la condición y si se cumple se ejecuta una determinada instrucción o grupo de instrucciones. Si no se cumple se ejecuta otra instrucción o grupo de instrucciones.



Sintaxis:

```
instrucción 1;
instrucción 2;
if(condición){
    instrucción 3;
    instrucción 4;
}else{
    instrucción 5;
    instrucción 6;
}
instrucción 7;
instrucción 8;
```

Instrucciones que se ejecutan **si se cumple** la condición

Instrucciones que se ejecutan **si no se cumple** la condición

En este caso se ejecutan las instrucciones 1 y 2 y a continuación se evalúa la condición. Si se cumple, se ejecutan las instrucciones 3 y 4. Si la condición no se cumple se ejecutan las instrucciones 5 y 6. En ambos casos el programa continuará su ejecución por las instrucciones 7 y 8.

Ejemplo: Programa que lee una nota por teclado y muestra un mensaje indicando si se ha aprobado o no. Si la nota es >= 5 mostrará los mensajes:

```
Enhorabuena!!
Has aprobado
```

Si la nota es < 5 mostrará el mensaje:

```
Lo Siento, has suspendido
```

```
/*
 * Programa que pide una nota por teclado y muestra si se ha aprobado o no
 */
import java.util.Scanner;
public class EjemploIfElse {
    public static void main( String[] args ){
        Scanner sc = new Scanner( System.in );
        System.out.print("Nota: ");
        int nota = sc.nextInt();
        if (nota >= 5){
            System.out.println("Enhorabuena!!");
            System.out.println("Has aprobado");
        } else {
            System.out.println("Lo Siento, has suspendido");
        }
        System.out.println("Hasta Pronto!");
    }
}
```

Ejemplo: programa que lee un número entero por teclado y muestra un mensaje indicando si el número es par o impar.

```
/*
 * Programa que pide un número por teclado y muestra si es par o impar
 */
import java.util.Scanner;
public class EjemploIfElse2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num;
        System.out.print("Introduzca un número entero: ");
        num = sc.nextInt();
        if (num % 2 == 0) {
            System.out.println("PAR");
        } else {
            System.out.println("IMPAR");
        }
    }
}
```

Ejemplos de ejecución:

Ejemplo 1

```
Introduzca un número entero: 3
IMPAR
```

Ejemplo 2

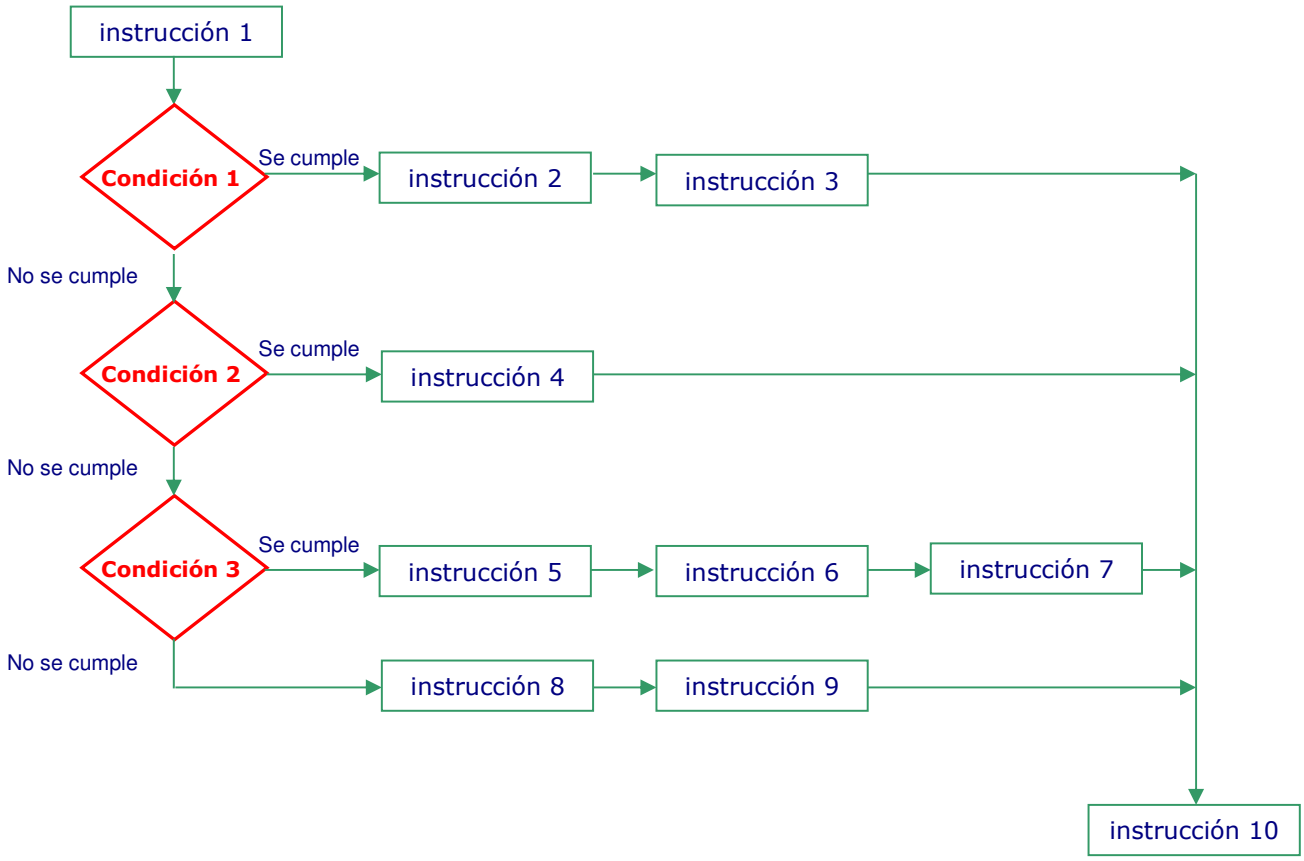
```
Introduzca un número entero: 214
PAR
```

ESTRUCTURA CONDICIONAL MÚLTIPLE. IF ANIDADOS

La estructura condicional múltiple se obtiene encadenando sentencias *if ... else*.

Mediante esta estructura podemos construir estructuras de selección más complejas.

Un ejemplo de *if* múltiple expresado en diagrama de flujo puede ser este:



Sintaxis en Java:

```
instrucción1
if(condición1){   //inicio de la estructura condicional múltiple
   instruccion2;
   instrucción3;
```
Instrucciones que se ejecutan **si se cumple** la condición 1

```
}else if(condición2){
        instruccion4;
```
Instrucción que se ejecuta **si se cumple** la condición 2

```
}else if(condición3){
        instruccion5;
        instrucción6;
        instrucción7;
```
Instrucciones que se ejecutan **si se cumple** la condición 3

```
}else{
      instruccion8;
      instrucción9;
```
Instrucciones que se ejecutan **si no se cumple ninguna condición**

```
} //fin de la estructura condicional múltiple
instrucción10;
```

Cada *else* se corresponde con el if más próximo que no haya sido emparejado.

Una vez que se ejecuta uno de los bloques de instrucciones, la ejecución del programa continúa en la siguiente instrucción que aparezca después de la estructura condicional múltiple.

Ejemplo: programa que pide que se introduzca una hora y muestra un mensaje según la hora introducida corresponda a la mañana, la tarde o la noche.

```
//Programa que muestra un saludo distinto según la hora introducida
import java.util.Scanner;
public class EjemploIfAnidados {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int hora;
        System.out.print("Introduzca una hora (>= 0 y <= 23): ");
        hora = sc.nextInt();
        if (hora >= 0 && hora < 12) {
            System.out.println("Buenos días");
        } else if (hora >= 12 && hora < 21) {
            System.out.println("Buenas tardes");
        } else if (hora >= 21 && hora < 24) {
            System.out.println("Buenas noches");
        } else {
            System.out.println("Hora no válida");
        }
        System.out.println("Hasta pronto!!!");
    }
}
```

Cuatro ejemplos de ejecución del programa:

```
Introduzca una hora (>= 0 y <= 23): 6
Buenos días
Hasta pronto!!!
```

```
Introduzca una hora (>= 0 y <= 23): 21
Buenas noches
Hasta pronto!!!
```

```
Introduzca una hora (>= 0 y <= 23): 25
Hora no válida
Hasta pronto!!!
```

```
Introduzca una hora (>= 0 y <= 23): 13
Buenas tardes
Hasta pronto!!!
```

En estos ejemplos podemos comprobar que una vez que se ejecuta una de las instrucciones dentro del bloque de if anidados, la ejecución del programa continúa en la siguiente instrucción a continuación del bloque de if anidados, en este caso la ejecución sigue por la instrucción

```
System.out.println("Hasta pronto!!!");
```

Ejemplo: Programa que pide que se introduzca un valor numérico de tipo double correspondiente a la nota de un alumno en una determinada asignatura y muestra un texto con la calificación equivalente de la siguiente forma:

Si la nota es menor que 5 muestra “Suspenso”
Si la nota es mayor o igual que 5 y menor que 6 muestra “Suficiente”
Si la nota es mayor o igual que 6 y menor que 7 muestra “Bien”
Si la nota es mayor o igual que 7 y menor que 9 muestra “Notable”
Si la nota es mayor o igual que 9 y menor que 10 muestra “Sobresaliente”
Si la nota es igual a 10 muestra “Matrícula de honor”
Si la nota es menor que 0 o mayor que 10 mostrará el mensaje “Nota no válida”

```
// programa que lee una nota y escribe la calificación correspondiente
import java.util.Scanner;
public class EjemploIfAnidados2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double nota;
        System.out.println("Introduzca una nota entre 0 y 10: ");
        nota = sc.nextDouble();
        System.out.println("La calificación del alumno es ");
        if (nota < 0 || nota > 10) {
            System.out.println("Nota no válida");
        } else if (nota == 10) {
            System.out.println("Matrícula de Honor");
        } else if (nota >= 9) {
            System.out.println("Sobresaliente");
        } else if (nota >= 7) {
            System.out.println("Notable");
        } else if (nota >= 6) {
            System.out.println("Bien");
        } else if (nota >= 5) {
            System.out.println("Suficiente");
        } else {
            System.out.println("Suspenso");
        }
    }
}
```

Algunos errores comunes que se suelen cometer en las instrucciones if

Escribir un punto y coma al final de la condición:

```
if (nota >= 6);
    System.out.println("Notable");
```

En este caso el mensaje *"Notable"* saldrá siempre. El *if* termina en el ;

Escribir `elseif` en lugar de `else if`

```
elseif (nota >= 5)
        System.out.println("Suficiente");
```

Usar = en lugar de == en las comparaciones

```
if (nota=10)
        System.out.println("Matrícula de Honor");
```

Comparar dos String.

Para comparar dos String **NO se deben utilizar los operadores relaciones** ==, >, >= , etc.

Si, por ejemplo, cadena1 y cadenas2 son dos variables de tipo String, para comprobar si son iguales (si ambas contienen exactamente la misma cadena de caracteres) no se puede usar el operador ==

```
if(cadena1 == cadena2){ //Comparación no válida de 2 String
```

Para comprobar si las dos variables son iguales se debe usar el método **equals**

```
if (cadena1.equals(cadena2))
```

Para comprobar si las dos variables son distintas se escribirá la instrucción de esta forma:

```
if (!cadena1.equals(cadena2))
```

Si lo que queremos es comprobar si dos cadenas son iguales sin tener en cuenta mayúscula o minúscula se usa el método **equalsIgnoreCase**

```
if (cadena1.equalsIgnoreCase(cadena2))
```

Para comparar dos String en el orden alfabético se usa el método **compareTo**

El método *compareTo* compara los dos String alfabéticamente y devuelve un valor entero con el resultado de la comparación que puede ser negativo, positivo o cero, de la siguiente forma:

- Devuelve 0 si los dos String son iguales.
- Devuelve un valor < 0 si el primer String es alfabéticamente menor que el segundo.
- Devuelve un valor > 0 si el primer String es alfabéticamente mayor que el segundo.

Si por ejemplo, tenemos dos String texto1 y texto2 con este contenido:

```
texto1 = "veloces"
texto2 = "verde"
```

Si queremos mostrarlos por pantalla en orden alfabético, hay que realizar la comparación entre ellas:

```
if(texto1.compareTo(texto2) < 0){
    System.out.printl(texto1);
    System.out.printl(texto2);
}else{
    System.out.printl(texto2);
    System.out.printl(texto1);
}
```

En la instrucción if

```
if(texto1.compareTo(texto2) < 0)
```

Si esta condición es cierta significa que compareTo ha devuelto un valor negativo como resultado de la comparación de los dos String. Esto indica que texto1 alfabéticamente es menor que texto2.

Si la condición es cierta se muestra primero texto1 y después texto2.

Si el false se muestra texto2 y después texto1.

También podemos mostrarlos ordenados alfabéticamente comprobando si el resultado de la comparación es > 0

```
if(texto1.compareTo(texto2) > 0){
    System.out.printl(texto2);
    System.out.printl(texto1);
}else{
    System.out.printl(texto1);
    System.out.printl(texto2);
}
```

En este caso la instrucción if es

```
if(texto1.compareTo(texto2) > 0)
```

Si esta condición es cierta significa que compareTo ha devuelto un valor positivo como resultado de la comparación de los dos String. Esto indica que texto1 alfabéticamente es mayor que texto2.

Si la condición es cierta se muestra primero texto2 y después texto1.

Si el false se muestra texto1 y después texto2.

La forma interna en la que realiza compareTo la comparación de los String es la siguiente:

Para comparar los dos String se toma el valor ASCII de cada carácter y se van restando los valores de los caracteres de ambos que se encuentran en la misma posición comenzando por el carácter de la izquierda.

La comparación finaliza cuando al restar dos caracteres se obtiene un resultado distinto de cero o cuando se hayan restado todos los caracteres de ambas cadenas, en este caso el resultado será cero.

El valor obtenido es lo que devuelve el método compareTo.

Ejemplo:

```
texto1 = "veloces"
texto2 = "verde"

if(texto1.compareTo(texto2) < 0){
```

Para hacer la comparación se resta un carácter de texto1 y un carácter de texto2

```
               v     e     l     o     c     e     s
COD.ASCII:   118   101   108   111    99   101   115
               |     |     |
               v     e     r     d     e
COD.ASCII:   118   101   114   100   101
               |     |     |
Diferencia:    0     0    -6
```

En este caso el valor que ha provocado que finalice la comparación es -6. Se ha obtenido al restar los caracteres `l - r (108 - 114)`. El método compareTo devuelve en este caso el valor -6.

Se ha obtenido un valor < 0 eso indica que el carácter `l` va antes que el carácter `r` en el orden alfabético y por lo tanto "veloces" alfabéticamente va antes que "verde"

El valor -6 devuelto por compareTo hace que la condición

```
if(texto1.compareTo(texto2) < 0){
```

sea cierta.

Ejemplos de uso:

Suponiendo que cadena1 y cadena2 son de tipo String y ambas contienen una cadena de caracteres, podemos compararlas de varias formas según necesitemos:

```
if(cadena1.compareTo(cadena2) < 0) //si la condición se cumple significa que
                                   //cadena1 es alfabéticamente menor que cadena2

if(cadena1.compareTo(cadena2) > 0) //si la condición se cumple significa que
                                   //cadena1 es alfabéticamente mayor que cadena2

if(cadena1.compareTo(cadena2) == 0)//si la condición se cumple significa que
                                   //cadena1 es igual que cadena2

if(cadena2.compareTo(cadena1) > 0) //si la condición se cumple significa que
                                   //cadena2 es alfabéticamente mayor que cadena1

if(cadena2.compareTo(cadena1) < 0) //si la condición se cumple significa que
                                   //cadena2 es alfabéticamente menor que cadena1
```

Para comparar Strings en el orden alfabético sin tener en cuenta mayúsculas o minúsculas se usa el método **compareToIgnoreCase**

```
if(cadena1.compareToIgnoreCase(cadena2) > 0)
```

En este caso, si la condición se cumple significa que cadena 1 es alfabéticamente mayor que cadena2 sin importar que los caracteres en ambas cadenas estén en mayúsculas o en minúsculas.

3.2 INSTRUCCION switch

Esta estructura de control se utiliza para seleccionar una opción de entre múltiples opciones posibles.

Es una alternativa a los if .. else anidados.

Si el número de anidamientos if .. else es elevado, la estructura switch produce un código más sencillo de leer y modificar.

El flujo de ejecución del programa lo determina el valor de una variable o expresión.

El tipo de esta variable o expresión puede ser uno de los siguientes:

- Tipo primitivo: byte, short, char, int.
- La clase envolvente de los tipos primitivos anteriores: Byte, Short, Character, Integer.
- Tipo String.
- Tipos enumerados (enum).

La instrucción switch no permite que la variable o expresión sea del tipo float o double.

La sintaxis general de una estructura switch es la siguiente:

```
switch (expresión){
      case VALOR1:
           instrucciones1;
           break;
      case VALOR2:
           instrucciones2;
           break;
      // tantos case como sean necesarios
      default:
           instrucciones3;
}
```

Funcionamiento de la instrucción *switch*:

- Primero se evalúa la expresión y salta al case cuyo valor coincida con el valor de la expresión.
- Se ejecutan las instrucciones que contiene el case seleccionado hasta que se encuentra un *break* o hasta el final del *switch*. El *break* produce un salto y la ejecución continúa por la siguiente instrucción que se encuentre a continuación del *switch*.
- Si el valor de la expresión no coincide con ningún *case* se ejecuta el bloque *default*.

El **bloque default es opcional** por lo tanto no es obligatorio que exista siempre un bloque *default* en un *switch*.

El bloque *default* suele escribirse de forma habitual al final del *switch* a continuación de todos los *case* pero no es obligatorio escribirlo al final, puede aparecer en cualquier lugar.

Los valores que aparecen en los *case* deben ser únicos. No puede haber dos *case* con el mismo valor. En ese caso se produciría un error de compilación.

El valor de un *case* no puede ser una variable. Debe ser un literal o una constante (una variable declarada como *final*).

Ejemplo: Programa que lee por teclado un número de mes y muestra el nombre de mes correspondiente.

```
//Programa que pide un número de mes y muestra el nombre correspondiente
import java.util.Scanner;
public class EjemploSwitch {
    public static void main(String[] args) {
        int mes;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un numero de mes: ");
        mes = sc.nextInt(); //se introduce un número de mes y se guarda en la variable
        switch (mes) {  //inicio del switch
                case 1: System.out.println("ENERO");
                        break;
                case 2: System.out.println("FEBRERO");
                        break;
                case 3: System.out.println("MARZO");
                        break;
                case 4: System.out.println("ABRIL");
                        break;
                case 5: System.out.println("MAYO");
                        break;
                case 6: System.out.println("JUNIO");
                        break;
                case 7: System.out.println("JULIO");
                        break;
                case 8: System.out.println("AGOSTO");
                        break;
                case 9: System.out.println("SEPTIEMBRE");
                         break;
                case 10: System.out.println("OCTUBRE");
                         break;
                case 11: System.out.println("NOVIEMBRE");
                         break;
                case 12: System.out.println("DICIEMBRE");
                         break;
                default : System.out.println("Mes no válido");
        }//fin del switch
    }
}
```

En este ejemplo el valor introducido por teclado se guarda en la variable *mes* de tipo int.

Esta variable es la que se escribe en la instrucción switch para comprobar su valor.

Según el valor que contenga *mes* se ejecutará el case que coincida con ese valor.

Por ejemplo, si se ha introducido por teclado el valor 3, se ejecutará el código contenido en el case 3, en este caso, se ejecutará la instrucción

```
System.out.println("MARZO");
```

A continuación de esta instrucción se ha escrito la instrucción `break;` La instrucción break hace que la ejecución del programa continúe en la siguiente instrucción que haya después del switch.

Ejemplo: programa que lee por teclado un String correspondiente al nombre de un mes y muestra el número de días que tiene.

```
// Programa que pide un nombre de mes y muestra el número de días correspondiente
import java.util.Scanner;
public class Ejemplo2Switch {
    public static void main(String[] args) {
        String mes;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un nombre de mes: ");
        mes = sc.nextLine();
        switch (mes.toUpperCase()) {
            case "ENERO": System.out.println("31 DÍAS");
                          break;
            case "FEBRERO": System.out.println("28 ó 29 DÍAS");
                            break;
            case "MARZO": System.out.println("31 DÍAS");
                          break;
            case "ABRIL": System.out.println("30 DÍAS");
                          break;
            case "MAYO":  System.out.println("31 DÍAS");
                          break;
            case "JUNIO": System.out.println("30 DÍAS");
                          break;
            case "JULIO": System.out.println("31 DÍAS");
                          break;
            case "AGOSTO": System.out.println("31 DÍAS");
                           break;
            case "SEPTIEMBRE": System.out.println("30 DÍAS");
                               break;
            case "OCTUBRE": System.out.println("31 DÍAS");
                            break;
            case "NOVIEMBRE": System.out.println("30 DÍAS");
                              break;
            case "DICIEMBRE": System.out.println("31 DÍAS");
                              break;
            default: System.out.println("Mes no válido");
        }
    }
}
```

En este ejemplo el valor introducido por teclado se guarda en la variable *mes* de tipo String.

Esta variable es la que se escribe en la instrucción switch para comprobar su valor.

En lugar de escribir la instrucción switch de esta forma:

```
switch(mes){
```

Se ha escrito

```
switch (mes.toUpperCase()) {
```

El método toUpperCase() toma el contenido del String *mes* y lo pasa a mayúsculas para que switch realice la comprobación con el contenido de la variable en mayúsculas. El contenido original de la variable *mes* no cambia. Esto sirve para, por ejemplo, si se ha introducido por teclado “Enero” al pasarlo a mayúsculas coincidirá con el primer case. Si no lo pasamos a mayúscula, “Enero” no coincide con ningún case y se ejecutará la instrucción default.

Un *case* puede contener además de instrucciones que se ejecuten en secuencia, otras instrucciones como condiciones o bucles o incluso otro *switch*.

Ejemplo: programa que pide por teclado dos números y un operador +, -, *, / y calcula y muestra el resultado de la operación según el operador introducido.

Si se introduce el operador división, se debe comprobar si el divisor es cero, en ese caso la división no se puede realizar y se muestra un mensaje indicándolo. En este caso el resultado de la operación no se mostrará por pantalla.

Si se introduce un operador no válido se muestra un mensaje indicándolo.

```
//Programa que pide dos números y un operador y muestra el resultado de la operación
import java.util.Scanner;
public class Ejemplo3Switch {
    public static void main(String[] args){
        double A,B, Resultado = 0 ;
        char operador;
        boolean calculado = true;//variable para comprobar si se ha realizado la operación
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un número:");
        A = sc.nextInt();
        System.out.print("Introduzca otro número:");
        B = sc.nextInt();
        sc.nextLine(); //limpiar buffer
        System.out.print("Introduzca un operador (+,-,*,/):");
        operador = sc.nextLine().charAt(0);
        switch (operador) {
                case '-' : Resultado = A - B;
                           break;
                case '+' : Resultado = A + B;
                           break;
                case '*' : Resultado = A * B;
                           break;
                case '/' : if(B!=0){
                              Resultado = A / B;
                           }else{
                              System.out.println("\nNo se puede dividir por cero");
                              calculado = false;
                           }
                           break;
                default : System.out.println("\nOperador no valido");
                          calculado = false;
        }
        //si la variable boolean sigue valiendo true se muestra el resultado
        if(calculado){
            System.out.println("\nEl resultado es: " + Resultado);
        }
    }
}
```

Si una misma instrucción o bloque de instrucciones se tienen que ejecutar para distintos valores, en ese caso podemos escribir *case* multiples. Esto consiste en escribir varias instrucciones case seguidas.

Ejemplo: programa que pide introducir por teclado un número entero entre 1 y 7 correspondiente a un día de la semana y muestra un mensaje indicando si ese día hay clase de programación o no.

```
import java.util.Scanner;
public class Ejemplo4Switch {
    public static void main(String[] args) {
        int numeroDia;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca el numero del día de la semana (1 a 7): ");
        numeroDia = sc.nextInt();
        switch (numeroDia){
                case 1:
                case 3:
                case 5: System.out.println("Tenemos clase de programación");
                        break;
                case 2:
                case 4: System.out.println("Vaya, que pena");
                        System.out.println("Hoy no tenemos clase de programación");
                        break;
                case 6:
                case 7: System.out.println("Estoy deseando que llegue el lunes!!");
                        break;
                default : System.out.println("Número de día no válido");
        }
    }
}
```

En este programa si se introduce por ejemplo un 1 como número de día, la ejecución pasará al *case 1* y comienza a ejecutar las instrucciones que encuentre desde ahí hacia abajo hasta encontrar un *break*.

Es **importante** entender que una vez que la ejecución del programa entra en uno de los *case*, el programa sigue ejecutándose desde ese punto hasta que encuentre el primer *break* (estén dentro del mismo *case* o no) o hasta que se llegue al final del *switch* en cuyo caso continuará por la instrucción que se encuentre a continuación del *switch*.

Ejemplo: programa que pide un valor entero por teclado entre 1 y 6 y realiza una serie de operaciones con él dependiendo de su valor inicial.

```
import java.util.*;
public class Ejemplo5Switch {
    public static void main(String[] args) {
        int x, y = 10;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un número >= 1 y <= 6: ");
        x = sc.nextInt();
        switch (x){
                case 1: x--;
                case 2: x+=5;
                case 3: x = x + y;
                        break;
                case 4:
                case 5:
                case 6: x++;
                        break;
                default : System.out.println("Número no válido");
        }
        System.out.println("x = " + x);
    }
}
```

Si en este programa se introduce un 1 como valor para la x, en el *switch* se entrará en el *case 1* y se ejecutarán todas las instrucciones hacia abajo hasta encontrar un *break*, es decir, si x es un 1 se ejecutan las instrucciones:

```
x--;
x+=5;
x = x + y;
```

Siete ejemplos de ejecución de este programa:

```
Introduzca un número >= 1 y <= 6: 1
x = 15
```

```
Introduzca un número >= 1 y <= 6: 2
x = 17
```

```
Introduzca un número >= 1 y <= 6: 3
x = 13
```

```
Introduzca un número >= 1 y <= 6: 4
x = 5
```

```
Introduzca un número >= 1 y <= 6: 5
x = 6
```

```
Introduzca un número >= 1 y <= 6: 6
x = 7
```

```
Introduzca un número >= 1 y <= 6: 0
Número no válido
x = 0
```

switch a partir de Java 12 -> expresión switch

Desde la versión 12 de Java en adelante la instrucción switch ha ido evolucionando, incorporando una serie de cambios, la mayoría de ellos en modo *preview (modo provisional),* hasta que en la versión Java 17 estos cambios se han consolidado.

Los cambios realizados en switch hacen que se hayan incorporado al lenguaje las **expresiones switch**

A partir de Java 17 podremos utilizar el switch de dos formas:

- podemos utilizar **instrucciones switch,** la instrucción switch que se ha explicado anteriormente y que existe desde la primera versión de Java.
- podemos utilizar las nuevas **expresiones switch**.

La principal diferencia entre una instrucción y una expresión switch es que **las expresiones switch pueden devolver un valor** mientras que las instrucciones switch solo evalúan el dato pero no pueden devolver un valor.

Ejemplo: Vamos a escribir un programa que pida que se introduzca por teclado un número del 1 al 7 correspondiente a un número de día y guarde en una variable de tipo String el nombre del día. Por ejemplo, si se introduce el número 1, se guardará en la variable “Lunes”.

Vamos a resolverlo utilizando primero una instrucción switch tradicional y después lo resolveremos utilizando una expresión switch.

Solución utilizando una **instrucción** switch:

```
import java.util.Scanner;
public class Ejemplo6Switch {
    public static void main(String[] args) {
        int dia;
        String nombreDia;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduce un número del 1 al 7:");
        dia = sc.nextInt();
        switch(dia){
            case 1 : nombreDia = "Lunes";
                     break;
            case 2 : nombreDia = "Martes";
                     break;
            case 3 : nombreDia = "Miércoles";
                     break;
            case 4 : nombreDia = "Jueves";
                     break;
            case 5 : nombreDia = "Viernes";
                     break;
            case 6 : nombreDia = "Sábado";
                     break;
            case 7 : nombreDia = "Domingo";
                     break;
            default : nombreDia = "Valor no válido";
                      break;
        }
        System.out.println(nombreDia);
    }
}
```

Solución utilizando una **expresión** switch:

```
import java.util.Scanner;
public class Ejemplo7Switch {
    public static void main(String[] args) {
        int dia;
        String nombreDia;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduce un número del 1 al 7:");
        dia = sc.nextInt();
        nombreDia = switch (dia) {
            case 1 -> "Lunes";
            case 2 -> "Martes";
            case 3 -> "Miércoles";
            case 4 -> "Jueves";
            case 5 -> "Viernes";
            case 6 -> "Sábado";
            case 7 -> "Domingo";
            default -> "Valor no válido";
        };
        System.out.println(nombreDia);
    }
}
```

En la expresión switch se han sustituido los **:** después del case por una flecha ->

Lo que aparece después de la flecha es lo que devuelve el switch. En este caso es lo que se le asigna a la variable *nombreDia*.

Este ejemplo nos sirve para explicar el funcionamiento de las expresiones switch:

- Primero se evalúa el valor de la variable, en este caso el valor que contiene la variable *día*, y salta al case cuyo valor coincida con el valor de la variable.
- Lo que aparece a la derecha de la flecha correspondiente a ese *case* es lo que devuelve el switch **y el switch finaliza**. En este ejemplo, la cadena de texto que aparece después de la flecha del case es lo que se le asigna a la variable *nombreDia*.
- La instrucción **break desaparece** en las expresiones switch. Solo se ejecuta el case que corresponde al valor de la variable o a default. Después la ejecución del programa continúa por la siguiente instrucción que se encuentre a continuación del switch.
- **Si en los case no se cubren todos los posibles valores** que puede tomar la variable que se le envía al switch, **será obligatorio incluir un *default***. Si no se incluye obtendremos el error *the switch expression does not cover all possible input values.* En este ejemplo debemos incluirlo ya que los valores que puede tomar una variable entera son muchos más que los valores del 1 al 7 que aparecen en los case. Como la expresión switch debe devolver algo para asignárselo a la variable *nombreDia*, si introducimos un valor que no coincide con ningún case podrá devolver lo que contenga default.
- Como la expresión switch devuelve un valor que se le asigna a una variable, estamos realizando una instrucción de asignación y debe finalizar con un ; después de la llave que cierra el switch.

  Si por ejemplo día = 3, el resultado de este switch es esta asignación:

  ```
  nombreDia = "Miércoles";
  ```

  y como cualquier instrucción de este tipo debe finalizar con ;

En este ejemplo se ha guardado en una variable lo que devuelve switch y después se muestra la variable por pantalla. Si en lugar de guardarlo en una variable lo mostramos directamente por pantalla, el programa ahora lo podemos escribir así:

```
import java.util.Scanner;
public class Ejemplo8Switch {
    public static void main(String[] args) {
        int dia;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduce un número del 1 al 7:");
        dia = sc.nextInt();
        switch (dia) {
            case 1 -> System.out.println("Lunes");
            case 2 -> System.out.println("Martes");
            case 3 -> System.out.println("Miércoles");
            case 4 -> System.out.println("Jueves");
            case 5 -> System.out.println("Viernes");
            case 6 -> System.out.println("Sábado");
            case 7 -> System.out.println("Domingo");
            default -> System.out.println("Valor no válido");
        }
    }
}
```

En este caso la expresión switch realmente no devuelve nada sino que muestra por pantalla el mensaje correspondiente al case que coincida con el valor de la variable o el mensaje indicado en default.

En este ejemplo no sería obligatorio incluir default ya que switch se limita a mostrar un mensaje por pantalla pero no devuelve nada por lo que si no se cumple ningún case simplemente no mostraría nada.

Otra de las mejoras que incorporan las expresiones switch es que permiten **incluir varios valores en un mismo case**.

Como ejemplo de esto podemos volver a escribir el programa `Ejemplo4Switch` de esta forma:

```
public class Ejemplo4SwitchV2 {
    public static void main(String[] args) {
        int numeroDia;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca el número del día de la semana (1 a 7): ");
        numeroDia = sc.nextInt();
        switch (numeroDia){
             case 1, 3, 5 -> System.out.println("Tenemos clase de programación");
             case 2, 4 -> {
                 System.out.println("Vaya, que pena");
                 System.out.println("Hoy no tenemos clase de programación");
             }
             case 6, 7 -> System.out.println("Estoy deseando que llegue el lunes!!");
             default -> System.out.println("Número de día no válido");
        }
    }
}
```

En las expresiones switch un case puede contener un bloque de código, en este caso se utiliza la instrucción **yield** para devolver el valor dentro del bloque de código.

Ejemplo: programa que introduce dos números enteros por teclado y los guarda en las variables A y B. El programa asigna a otra variable C lo siguiente:

Si A es 1 asigna a C el valor A * 2

SI A es 2 asigna a C el valor A * 100

Si A es 3 y B es par asigna a C el valor A + B.

Si A es 3 y B es impar asigna a C el valor A – B.

Si el valor de A es otro asigna a C el valor 0.

```
import java.util.Scanner;
public class JavaApplication9 {
    public static void main(String[] args) {
        int A, B, C;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un número: ");
        A = sc.nextInt();
        System.out.print("Introduzca otro número: ");
        B = sc.nextInt();
        C = switch (A) {
                case 1 -> A * 2;
                case 2 -> A * 100;
                case 3 -> {
                    if(B % 2 == 0){
                        yield A + B;
                    }else{
                        yield A - B;
                    }
                }
                default -> 0;
        };
        System.out.println(C);
    }
}
```

Estas son algunas de las características que incorporan las expresiones switch. Hay otras más avanzadas como el pattern matching, las expresiones lambda dentro de switch, guarded patterns … de las que puedes encontrar abundante documentación en la web.

3.3 OPERADOR CONDICIONAL ? :

El operador condicional se puede utilizar en sustitución de la instrucción condicional *if-else.*

Lo forman los caracteres **?** y **:**

Sintaxis:

`expresión1 ? expresión2 : expresión3`

Si expresión1 es cierta entonces se evalúa expresión2 y éste será el valor que devuelve el operador.

Si expresión1 es falsa, se evalúa expresión3 y éste será el valor que devuelve el operador.

Ejemplo:

Programa que pide por teclado un número entero y muestra si es positivo o negativo.

Consideramos el cero como positivo.

```
import java.util.Scanner;
public class OperadorCondicional {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num;
        System.out.println("Introduzca número: ");
        num = sc.nextInt();
        System.out.println(num >= 0 ? "POSITIVO" : "NEGATIVO");
    }
}
```

En el ejemplo, la instrucción:

```
System.out.println(num >= 0 ? "POSITIVO" : "NEGATIVO");
```

Es equivalente a escribir esto:

```
if(num >= 0){
   System.out.println("POSITIVO");
}else{
   System.out.println("NEGATIVO");
}
```

Ejemplo: programa que pide por teclado un número entero y muestra si es par o impar.

```
import java.util.*;
public class Ejemplo1OperadorCondicional2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num;
        System.out.println("Introduzca número: ");
        num = sc.nextInt();
        System.out.println(num%2 == 0 ? "PAR" : "IMPAR");
    }
}
```

En el ejemplo, la instrucción:

```
System.out.println(num%2 == 0 ? "PAR" : "IMPAR");
```

Es equivalente a escribir esto:

```
if(num%2 == 0){
   System.out.println("PAR");
}else{
   System.out.println("IMPAR");
}
```

4. ESTRUCTURA REPETITIVA O ITERATIVA

Esta estructura de control permite ejecutar de forma repetida una instrucción o un bloque de instrucciones.

Las instrucciones se repiten mientras que se cumpla una determinada condición.

Esta condición se conoce como **condición de salida** del bucle.

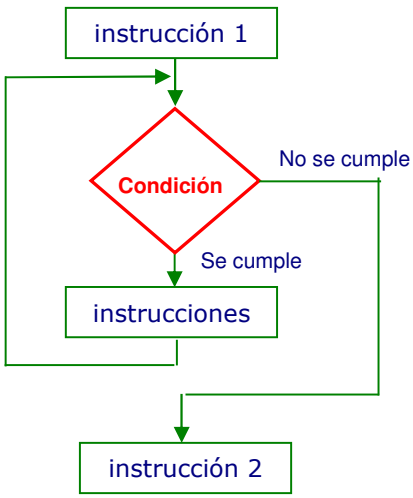
En Java las estructuras repetitivas se implementan mediante:

- ciclo while
- ciclo do - while
- ciclo for

4.1 CICLO WHILE

Las instrucciones se repiten mientras la condición sea cierta.

En un ciclo *while* la condición **se comprueba al principio** del bucle por lo que las instrucciones que lo forman se **ejecutan 0 ó más veces**.



Sintaxis:

```
instrucción 1;
while (condición){ //inicio while
      instrucciones;
}   //fin while
instrucción 2;
```

La ejecución de un bucle *while* sigue los siguientes pasos:

1. Se evalúa la condición.
2. Si el resultado es *false* las instrucciones no se ejecutan y el programa sigue ejecutándose por la siguiente instrucción a continuación del *while*.
3. Si el resultado es true se ejecutan las instrucciones y se vuelve al paso 1

Ejemplo: programa que pide introducir números enteros por teclado. La lectura de números termina cuando se introduce un número negativo. El programa muestra la suma de todos los números introducidos excepto el negativo.

```
/*
 * Programa que lee números hasta que se lee un negativo y muestra la
 * suma de los números leídos
 */
import java.util.*;
public class Ejemplo1While {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num;        //variable que contiene el número introducido
        int suma = 0;  //variable donde acumularemos la suma de los números

        System.out.print("Introduzca un número ( < 0 para finalizar): ");
        num = sc.nextInt(); //lectura del primer número

        while (num >= 0){ //inicio del bucle while
               suma = suma + num; //se suma el número introducido
               System.out.print("Introduzca un número ( < 0 para finalizar): ");
               num = sc.nextInt(); //lectura del siguiente número
        } //fin del bucle while
        System.out.println("La suma es: " + suma ); //se muestra la suma
    }
}
```

En este ejemplo, según el enunciado, se trata de leer números enteros hasta que se introduzca un número negativo, por lo tanto, **las instrucciones contenidas dentro del *while* se repiten mientras el número que introduzcamos sea >= 0.**

Este es un ejemplo de estructura repetitiva en el que no sabemos a priori cuántas veces se repetirán las instrucciones. El número de iteraciones del bucle depende del valor de la variable *num* que se introduce por teclado. En el ejemplo podemos ver también que la lectura del primer número se realiza antes de que comience la estructura *while*. Hay que hacerlo así porque es posible que el primer número que se introduzca sea el negativo, en ese caso no tendríamos que sumarlo, por lo tanto el while no se debe ejecutar y se muestra directamente el mensaje con la suma del número.

Esta técnica se conoce como **lectura adelantada o anticipada** y es la forma correcta de utilizar la estructura *while* cuando no sabemos el número de iteraciones que se van a realizar.

Dos ejemplos de ejecución del programa:

```
Introduzca un número ( < 0 para finalizar): 6
Introduzca un número ( < 0 para finalizar): 2
Introduzca un número ( < 0 para finalizar): 5
Introduzca un número ( < 0 para finalizar): -1
La suma es: 13
```

En este caso se introducen los números 6, 2 y 5. Cuando se introduce el valor -1 el bucle finaliza y se muestra la suma.

```
Introduzca un número ( < 0 para finalizar): -2
La suma es: 0
```

En este ejemplo de ejecución el primer número introducido es negativo por lo que el bucle *while* no se ejecuta. Se muestra el resultado de la suma que en este caso será 0.

Ejemplo: programa que muestra N asteriscos por pantalla. El valor de N se introduce por teclado.

```
//programa que lee un número n y muestra n asteriscos
import java.util.Scanner;
public class Ejemplo2While {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N; //variable que contiene el número de asteriscos a mostrar
        int contador = 0; //variable para contar los asteriscos que se han mostrado
        System.out.print("Introduce número de asteriscos a mostrar: ");
        N = sc.nextInt(); //leemos el número total de asteriscos a mostrar

        while (contador < N){ //inicio del bucle while
               System.out.print("*");
               contador++;
        }  //fin del bucle while

        System.out.println("Fin programa");
    }
}
```

En este programa se trata de mostrar tantos asteriscos como indique el número N que se ha introducido por teclado.

Este es un ejemplo de estructura repetitiva en la que sabemos a priori cuántas veces se va a repetir el *while*. En este caso el *while* se va a repetir N veces. La variable *contador* será la **variable de control** que hará que el *while* termine. Cada vez que se muestra un asterisco por pantalla se suma 1 a esta variable. El *while* se repite mientras el valor del contador sea menor que el número total de asteriscos a mostrar.

Dos ejemplos de ejecución:

```
Introduce número de asteriscos a mostrar: 10
**********
Fin programa
```

```
Introduce número de asteriscos a mostrar: 0

Fin programa
```

Ejemplo: programa que muestra una tabla con temperaturas expresadas en grados Fahrenheit y su equivalente en grados Celsius. Los valores de las temperatura en grados Fahrenheit estarán comprendidas entre 10 ºF y 100 ºF y se mostrarán con un incremento de 10º entre ellas (10, 20, 30 ...).

Fórmula para pasar de ºF a ºC: `°C = (°F − 32) * 5 / 9`

```
public class Ejemplo3While {
    public static void main(String[] args) {
        int fahrenheit = 10; //contiene las temperaturas fahrenheit
        double celsius;      //contiene las temperaturas celsius
        System.out.printf(" °F \t  °C \n");
        System.out.println("---------------");

        while (fahrenheit <= 100 ){ //inicio del bucle while
               celsius = (fahrenheit - 32)*5/9.0;
               System.out.printf("%3d\t%6.2f \n", fahrenheit, celsius);
               fahrenheit += 10;
        }  //fin del bucle while
    }

}
```

Salida por pantalla:

```
 ºF       ºC
---------------
 10     -12,22
 20      -6,67
 30      -1,11
 40       4,44
 50      10,00
 60      15,56
 70      21,11
 80      26,67
 90      32,22
100      37,78
```

En este ejemplo la variable *fahrenheit* es la variable de control y será la encargada de determinar cuando finaliza el bucle *while*.

Su valor inicial es 10 y se deben mostrar las temperaturas desde 10ºF hasta 100ºF de 10 en 10, por lo tanto en cada iteración se incrementará su valor en 10.

El *while* se repite mientras el valor de la variable sea <= 100.

4.2 CICLO DO – WHILE

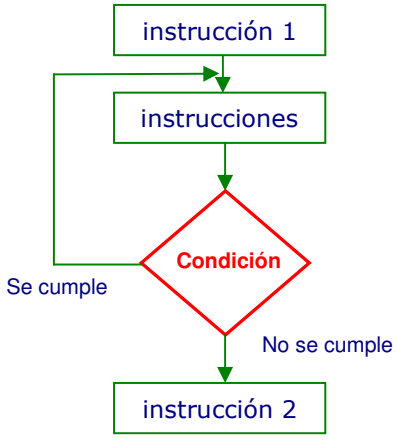
Las instrucciones se ejecutan mientras la condición sea cierta.

La condición **se comprueba al final** del bucle por lo que el bloque de instrucciones se ejecutarán **al menos una vez**.

Esta es la diferencia fundamental entre el bucle *while* y el bucle *do .. while.* Las instrucciones de un bucle *while* es posible que no se ejecuten si la condición inicialmente es falsa.

Bucle ***while*** se ejecuta **0 o más veces**.

Bucle ***do .. while*** se ejecuta **1 o más veces**.



Sintaxis:

```
instrucción1;
do{   //inicio do .. while
     instrucciones;
}while(condición);   //fin do .. while
instrucción2;
```

La ejecución de un bucle *do - while* sigue los siguientes pasos:

1. Se ejecutan las instrucciones a partir de la instrucción `do{`
2. Se evalúa la condición.
3. Si la condición no se cumple el programa sigue ejecutándose por la siguiente instrucción a continuación del *while*.
4. Si la condición se cumple volvemos al paso 1

Ejemplo: Programa que lee un número entero menor o igual que 100. Si el número es mayor que 100 se muestra un mensaje indicándolo y se vuelve a pedir. Finalmente se muestra por pantalla el número introducido.

```
//Programa que pide un número menor o igual que 100
import java.util.Scanner;
public class Ejemplo1DoWhile {
    public static void main(String[] args) {
        int valor;
        Scanner in = new Scanner( System.in );
        do {  //inicio del do .. while
            System.out.print("Introduce un número entero <= 100: ");
            valor = in.nextInt();
            if(valor > 100){
                System.out.println("Número no válido");
            }
        }while (valor > 100);  //fin del do .. while
        System.out.println("Ha introducido: " + valor);
    }
}
```

Dos ejemplos de ejecución:

```
Introduce un número entero <= 100: 27
Ha introducido: 27
```

```
Introduce un número entero <= 100: 690
Número no válido
Introduce un número entero <= 100: 584
Número no válido
Introduce un número entero <= 100: 58
Ha introducido: 58
```

En los ejemplos podemos comprobar que las instrucciones dentro del bucle *do .. while* se ejecutan al menos una vez.

Ejemplo: Programa que lee un número entero entre 1 y 10 ambos valores incluidos. Si el número es válido se muestra un mensaje indicándolo y se vuelve a pedir. Finalmente se muestra por pantalla el número introducido.

```
// Programa que lee un número entre 1 y 10
import java.util.Scanner;
public class Ejemplo2DoWhile {
    public static void main(String[] args) {
        int n;
        Scanner sc = new Scanner(System.in);
        do {  //inicio del do .. while
            System.out.print("Escribe un número entero entre 1 y 10: ");
            n = sc.nextInt();
            if (n < 1 || n > 10) {
                System.out.println("Valor no válido");
            }
        } while (n < 1 || n > 10);   //fin del do .. while

        System.out.println("Ha introducido: " + n);
    }
}
```

Dos ejemplos de ejecución del programa:

```
Escribe un número entero entre 1 y 10: 3
Ha introducido: 3
```

```
Escribe un número entero entre 1 y 10: -2
Valor no válido
Escribe un número entero entre 1 y 10: 6
Ha introducido: 6
```

Ejemplo: Programa que muestra los números del 1 al 10 todos en la misma fila y separados por un espacio en blanco.

```
import java.util.Scanner;
public class Ejemplo3DoWhile {
    public static void main(String[] args) {
        int i = 1;
        do {   //inicio del do .. while
            System.out.print(i + " ");
            i++;
        } while (i<=10);   //fin del do .. while
        System.out.println("\nFin programa");
    }
}
```

El programa muestra por pantalla:

```
1 2 3 4 5 6 7 8 9 10
Fin programa
```

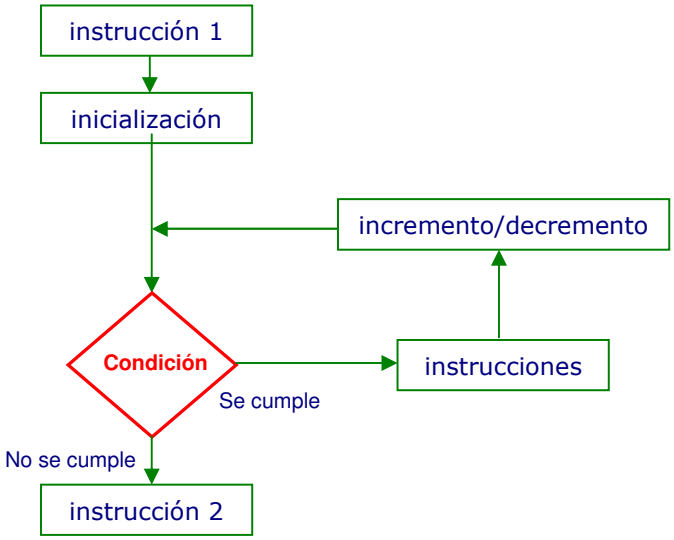
En este caso la variable *i* es la que contienen los valores a mostrar por pantalla. También es la variable de control que determina cuando finaliza el bucle. Inicialmente vale 1 y en cada iteración, después de mostrar su valor por pantalla se incrementa. El *do .. while* se repite mientras el valor de *i* sea menor o igual a 10.

4.3 CICLO FOR

Hace que una instrucción o bloque de instrucciones se repitan un **número determinado de veces mientras se cumpla la condición**.

Los bucles *for* son los más adecuados cuando se conoce el número de veces que se van a repetir las instrucciones.

La representación en diagrama de flujo de una instrucción *for* es la siguiente:



Sintaxis:

```
instrucción1;
for(inicialización; condición; incremento/decremento){ //inicio for
     instrucciones;
}   //fin for
instrucción2;
```

A continuación de la palabra for y entre paréntesis debe haber siempre **tres zonas separadas por punto y coma**:

- zona de inicialización.
- zona de condición
- zona de incremento ó decremento.
-

zona de condición

```
for(inicialización; condición; incremento/decremento){
```

zona de inicialización — zona de incremento ó decremento

Las tres zonas son opcionales. Si en alguna ocasión no fuese necesario escribir alguna de estas zonas se pueden dejar en blanco, pero los punto y coma deben aparecer.

Inicialización es la parte en la que la variable o variables de control del bucle toman su valor inicial. Puede haber una o más instrucciones en la zona de inicialización. Si hay varias instrucciones deben estar separadas por comas. La inicialización **se realiza solo una vez**.

Condición es una expresión booleana que determina si la sentencia o bloque de sentencias se ejecutan o no. Las instrucciones contenidas dentro del bucle *for* se ejecutan **mientras que la condición sea cierta**.

Incremento/decremento es una expresión que modifica la variable o variables de control del bucle. En esta zona puede haber más de una expresión para modificar las variables. Si hay varias expresiones deben estar separadas por comas.

La ejecución de un bucle *for* sigue los siguientes pasos:

1. Se inicializa la variable o variables de control (zona de inicialización)

2. Se evalúa la condición (zona de condición).

3. Si la condición es falsa, finaliza la ejecución del *for* y el programa continúa su ejecución en la siguiente instrucción después del *for*.

4. Si la condición es cierta se ejecutan las instrucciones contenidas dentro del *for*.

5. Se actualiza la variable o variables de control (zona incremento/decremento)

6. Se vuelve al punto 2.

Igual que el bucle *while*, **un bucle *for* se puede ejecutar 0 ó más veces**.

Ejemplo: Programa que muestra los números del 1 al 10 ambos incluidos todos en la misma línea y separados por un espacio en blanco.

```
/*
 * programa que muestra los números del 1 al 10
 */
public class Ejemplo1For {
    public static void main(String[] args) {
        int i;
        for (i = 1; i <= 10; i++) {    //inicio del for
            System.out.print(i + " ");
        }  //fin del for

        System.out.println("\nFin programa");
    }
}
```

Salida por pantalla:

```
1 2 3 4 5 6 7 8 9 10
Fin programa
```

La instrucción *for* del ejemplo la podemos interpretar de esta forma:

Primero asigna a i el valor inicial 1 y a continuación se comprueba la condición. Mientras que i sea menor o igual a 10 muestra `i + " "`, a continuación incrementa el valor de i y se comprueba de nuevo la condición. El *for* se repite mientras i sea <= 10.

En los programas de este tipo en los que la variable de control, en este caso la i, solo se utiliza dentro del bucle *for* es habitual declararla en la zona de incialización. Si lo hacemos de esa forma el programa quedaría así:

```
// Programa que muestra los números del 1 al 10
public class Ejemplo1For {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {    //inicio del for
            System.out.print(i + " ");
        }  //fin del for
        System.out.println("\nFin programa");
    }
}
```

Ejemplo: Programa que muestra los números del 10 al 1 ambos incluidos todos en la misma línea y separados por un espacio en blanco.

```
// Programa que muestra los números del 10 al 1
public class Ejemplo2For {
    public static void main(String[] args) {
        for (int i = 10; i > 0; i--) {    //inicio del for
            System.out.print(i + " ");
        }  //fin del for
        System.out.println("\nFin programa");
    }
}
```

Salida por pantalla:

```
10 9 8 7 6 5 4 3 2 1
Fin programa
```

La instrucción *for* del ejemplo la podemos interpretar así:

Primero asigna a i el valor inicial 10 y a continuación se comprueba la condición. Mientras que i sea mayor que 0 muestra `i + " "`, a continuación se decrementa el valor de i y se comprueba de nuevo la condición. El *for* se repite mientras i sea > 0.

Ejemplo: Vamos a hacer de nuevo el programa que muestra una tabla con temperaturas expresadas en grados Fahrenheit y su equivalente en grados Celsius pero esta vez utilizaremos un bucle *for* para resolverlo. El programa debe realizar los mismo que el anterior, los valores de las temperatura en grados Fahrenheit estarán comprendidas entre 10 ºF y 100 ºF y se mostrarán con un incremento de 10º entre ellas (10, 20, 30 ...).

```
//Programa que muestra una tabla de equivalencias entre grados Fahrenheit y celsius
public class Ejemplo3For {
    public static void main(String[] args) {
        double celsius;
        System.out.printf(" °F \t  °C \n");
        System.out.println("---------------");

        for (int fahrenheit = 10; fahrenheit <= 100; fahrenheit+= 10) {
             celsius = (fahrenheit - 32)*5/9.0;
             System.out.printf("%3d\t%6.2f \n", fahrenheit, celsius);
        }
    }
}
```

Salida por pantalla:

```
 °F       °C
---------------
 10     -12,22
 20      -6,67
 30      -1,11
 40       4,44
 50      10,00
 60      15,56
 70      21,11
 80      26,67
 90      32,22
100      37,78
```

La instrucción *for* del ejemplo la podemos interpretar así:

Primero asigna a *fahrenheit* el valor inicial 10 y a continuación se comprueba la condición. Mientras que *fahrenheit* sea menor o igual que 100 se ejecutan las instrucciones del *for*, a continuación se incrementa el valor de *fahrenheit* en 10 y se comprueba de nuevo la condición. El *for* se repite mientras *fahrenheit* sea <= 100.

Ejemplo: Dadas dos variables enteras a y b con valor inicial 1, escribimos un programa que muestre una tabla con el valor de ambas variables y su suma. En cada iteración el valor de *a* se incrementa en 1 unidad y el valor de *b* se incrementa en 2 unidades. La tabla finaliza cuando la suma de a y b sea >= 10.

Este es un ejemplo de programa que utiliza varias variables en las zonas de inicialización e incremento/decremento.

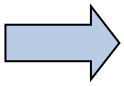
```
// Programa que muestra el valor de a, b y su suma mientras que la suma sea menor de 10.
// En cada iteración el valor de a se incrementa en 1 unidad y el de b en 2
public class Ejemplo4For {
    public static void main(String[] args) {
        for(int a = 1, b = 1; a + b < 10; a++, b+=2){
            System.out.println("a = " + a + "  b = " + b + "  a + b = " + (a+b));
        }
    }
}
```

La salida de este programa es:

```
a = 1  b = 1  a + b = 2
a = 2  b = 3  a + b = 5
a = 3  b = 5  a + b = 8
```

La instrucción *for* del ejemplo la podemos interpretar así:

Primero asigna a las variables *a* y *b* el valor 1 y a continuación se comprueba la condición. Mientras que la suma de a + b sea menor que 10 se ejecuta las instrucción del *for*, a continuación se suma 1 a la variable *a* y se suma 2 a la variable *b* y se comprueba de nuevo la condición. El *for* se repite mientras que la suma *a +b* sea menor que 10.



Un **error** que se puede cometer cuando escribimos un *for* es escribir el punto y coma después del paréntesis final. Un bucle *for* generalmente nunca lleva punto y coma final.

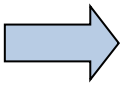
Por ejemplo el bucle:

```
for (int i = 1; i <= 10; i++);
{
     System.out.println("Elementos de Programación");
}
```

no visualiza la frase *"Elementos de Programación"* 10 veces como cabría esperar, ni produce un mensaje de error por parte del compilador.

En realidad lo que sucede es que se visualiza **una vez** la frase *"Elementos de Programación"*, ya que aquí la sentencia *for* es una sentencia vacía, sin instrucciones a realizar, al terminar con un punto y coma (;).

El *for* en este caso se limita a realizar la asignación inicial de la variable, y mientras se cumpla la condición incrementar el valor de i. La variable i toma como valor incial 1 y cuando el *for termina* valdrá 11. A continuación se mostrará el mensaje por pantalla.



¿Qué estructura repetitiva usar?

Aunque un *for* se puede utilizar también cuando no se sabe a priori el número de iteraciones a realizar, esta instrucción es especialmente indicada para bucles donde se conozca el número de pasadas.

Como regla práctica podemos decir que:

- La instrucción *for* se utiliza generalmente cuando se conoce el número de iteraciones a realizar.
- Las instrucciones *while* y *do-while* se utilizan generalmente cuando no se conoce a priori el número de iteraciones a realizar.
- La instrucción *do .. while* será más adecuada que la instrucción *while* cuando el bloque de instrucciones se deban repetir al menos una vez

4.4 BUCLES ANIDADOS

Hablamos de bucles anidados cuando se incluyen instrucciones *for*, *while* o *do-while* unas dentro de otras. Los anidamientos de estas estructuras tienen que ser correctos, es decir, una estructura anidada dentro de otra lo debe estar totalmente, sin solaparse una con la otra.

Sintaxis para anidar dos bucles *for*:

```
for(inicialización; condición; incremento/decremento){ //inicio del for1
     instrucciones1;
     for(inicialización; condición; incremento/decremento){ //inicio for2
         instrucciones2;   //instrucciones que se ejecutan en for2
     }  //fin del for2
     instrucciones3;
}   //fin del for1
```

Sintaxis para anidar dos bucles *while*:

```
while(condición){    //inicio del while 1
      instrucciones1;
      while(condición){    //inicio del while 2
            instrucciones2;
      }    //fin del while 2
      instrucciones3;
}    //fin del while 1
```

Sintaxis para anidar dos bucles *do .. while*:

```
do{  //inicio del do .. while 1
   instrucciones1;
   do{   //inicio del do .. while 2
      instrucciones2;
   }while(condición);  //fin del do .. while 2
   instrucciones3;
}while(condición);    // fin del do .. while 1
```

Los bucles se pueden anidar aunque sean de tipos distintos, por ejemplo podríamos anidar un bucle *for* dentro de un *do .. while* y éste a su vez dentro de un *while* de esta forma:

```
while(condición){    //inicio del while
      instrucciones1;
      do{  //inicio del do .. while
         instrucciones2;
         for(inicialización; condición; incremento/decremento){ //inicio del for
             instrucciones3;
         }  //fin del for
         instrucciones4;
      }while(condición);  //fin del do .. while
      instrucciones5;
}  //fin del while
```

Ejemplo: programa que dibuja un rectángulo formado por asteriscos. El número de filas y columnas del rectángulo se pide por teclado. El número de filas y de columnas debe ser > 1.

```
// Rectángulo sólido de asteriscos.
import java.util.Scanner;
public class Ejemplo1BuclesAnidados {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int filas, columnas;

        //leer número de filas hasta que sea un número > 1
        do{
           System.out.print("Introduce número de filas: ");
           filas = sc.nextInt();
        }while(filas<2);

        //leer número de columnas hasta que sea un número > 1
        do{
           System.out.print("Introduce número de columnas: ");
           columnas = sc.nextInt();
        }while(columnas<2);

        for(int i = 1; i<=filas; i++){    //for para las filas

            for(int j = 1; j<=columnas; j++){   //for para las columnas
                System.out.print(" * ");
            }  //fin del for para las columnas

            System.out.println();
        }  //fin del for para las filas

    }
}
```

Ejemplo de ejecución:

```
Introduce número de filas: 6
Introduce número de columnas: 10
 *  *  *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *  *  *
 *  *  *  *  *  *  *  *  *  *
```

En este programa se han anidado dos bucles *for* para mostrar el rectángulo de asteriscos.

El bucle *for* externo es el que corresponde a las filas. El *for* interno corresponde a las columnas.

Para cada una de las filas se ejecuta completamente el *for* de las columnas con lo que se consigue que para cada fila se muestren tantos asteriscos como columnas haya.

Al final de cada fila se escribe un salto de línea para que la siguiente fila comience a mostrarse en la línea siguiente.

Ejemplo: programa que dibuja un cuadrado formado por dígitos del 0 al 9. El número de filas del cuadrado se pide por teclado. El número de filas debe ser > 1.

```
import java.util.Scanner;
public class Ejemplo2BuclesAnidados {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int filas, columnas, contador = 0;

        //leer número de filas del cuadrado
        do{
           System.out.print("Introduce número de filas del cuadrado: ");
           filas = sc.nextInt();
        }while(filas<2);

        for(int i = 1; i<=filas; i++){    //filas

            for(int j = 1; j<=filas; j++){   //columnas = filas
                 System.out.print(contador % 10 + " ");
                 contador++;
            } //fin del for para las columnas

            System.out.println();
        } //fin del for para las filas

    }
}
```

Ejemplo de ejecución:

```
Introduce número de filas del cuadrado: 6
0 1 2 3 4 5
6 7 8 9 0 1
2 3 4 5 6 7
8 9 0 1 2 3
4 5 6 7 8 9
0 1 2 3 4 5
```

Ejemplo: A partir de una variable x que tomará los valores desde 1 hasta 10, mostrar una tabla de potencias de x. Para cada valor de x se mostrará x^2 x^3 x^4

```
// Programa que muestra una tabla con las potencias de x (x x² x³ x⁴)
// para valores de x desde 1 hasta 10
import java.util.Scanner;
public class Ejemplo1BuclesAnidados3{
    public static void main(String[] args) {
        int x, n;
        //mostrar la cabecera de la tabla
        System.out.printf("%10s%10s%10s%10s%n", "x", "x^2", "x^3", "x^4");

        for (x = 1; x <= 10; x++) {   //filas

            for (n = 1; n <= 4; n++) {   //columnas
                System.out.printf("%10.0f", Math.pow(x, n));
            }  //fin del for para las columnas

            System.out.println();
        } //fin del for para las filas
    }
}
```

La salida de este programa es:

```
         x       x^2       x^3       x^4
         1         1         1         1
         2         4         8        16
         3         9        27        81
         4        16        64       256
         5        25       125       625
         6        36       216      1296
         7        49       343      2401
         8        64       512      4096
         9        81       729      6561
        10       100      1000     10000
```

5. INSTRUCCIONES DE SALTO

Una instrucción de salto provoca la modificación del flujo de ejecución de un programa.

Java ofrece dos instrucciones de salto:

- break
- continue

5.1 BREAK

Esta instrucción provoca la finalización de una instrucción *switch*, *while*, *do-while* o *for*. En aquellos casos en los que existan estructuras de control repetitivas anidadas, un *break* produce la salida inmediata de aquel bucle en el que se encuentre incluida pero no de los demás.

La única situación en la que un buen programador debería utilizar la instrucción *break*, es para separar los diferentes casos de una instrucción *switch*.

Ejemplo: Programa que muestra números desde 1 hasta 50. Cuando encuentre el primer número que sea múltiplo de 3 y de 5 dejará de mostrar números.

```
public class Ejemplo1Break {
    public static void main(String[] args) {
            for (int i = 1; i <= 50; i++) {  //inicio del for
                 if (i % 3 == 0 && i % 5 == 0) {
                     break;
                 }
                 System.out.println(i);
            } //fin del for

            System.out.println("Hasta Pronto!!!");
    }
}
```

En este ejemplo, cuando se ejecuta la instrucción *break* el *for* finaliza y la ejecución del programa continúa en la instrucción que aparece a continuación del *for*.

5.2 CONTINUE

Esta instrucción provoca la ejecución de la siguiente iteración en el bucle, es decir, se salta las instrucciones que quedan hasta el final del bucle, y vuelve al inicio del mismo. Si se trata de un bucle *for* vuelve a la zona de incremento/decremento.

Un buen programador deberá evitar el uso de esta instrucción.

Ejemplo:

Programa que muestra los números desde el 1 hasta el 50 excepto los que sean múltiplos de 3 y de 5.

```
public class Ejemplo1Continue {
    public static void main(String[] args) {
        for (int i = 1; i <= 50; i++) {   //inicio del for
            if (i % 3 == 0 && i % 5 == 0) {
               continue;
            }
            System.out.println(i);
        }  //fin del for
    }
}
```