

# UNIDAD 3. ESTRUCTURAS DE CONTROL.

---

## ÍNDICE

- [INTRODUCCIÓN](#)
- [SENTENCIAS DE CONTROL DE FLUJO](#)
  - [IF - Sentencia condicional simple](#)
  - [IF - Sentencia condicional compuesta](#)
  - [IF - Anidación](#)
- [EJERCICIOS](#)
  - [SWITCH](#)
- [EJERCICIOS](#)
  - [WHILE - Sentencia repetitiva mientras](#)
    - [Bucles con contador](#)
    - [Bucles con centinela](#)
  - [DO WHILE - Sentencia repetitiva hacer mientras](#)
- [EJERCICIOS](#)
  - [FOR - Sentencia repetitiva para](#)
- [EJERCICIOS](#)
- [ANEXO DEL TEMA. BUCLES ANIDADOS](#)
  - [¿EN QUE CONSISTEN?](#)
- [EJERCICIOS](#)

## INTRODUCCIÓN

Hasta ahora las instrucciones que hemos visto, son instrucciones que se ejecutan secuencialmente. Las instrucciones de control de flujo permiten alterar esta forma de ejecución. A partir de ahora habrá líneas en el código que se ejecutarán o no dependiendo de una condición.

Esa condición se construye utilizando lo que se conoce como expresión lógica. Una expresión lógica es cualquier tipo de expresión Java que dé un resultado booleano (verdadero o falso).

Las expresiones lógicas se construyen por medio de variables booleanas o bien a través de los operadores relacionales (`=`, `>`, `<`, ...) y lógicos (`&&`, `||`, `!`).

## SENTENCIAS DE CONTROL DE FLUJO

### IF - Sentencia condicional simple

Se trata de una sentencia que, tras evaluar una expresión lógica, ejecuta una serie de instrucciones en caso de que la expresión lógica sea verdadera. Si la expresión tiene un resultado falso, no se ejecutará ninguna expresión. Su sintaxis es:

```
if(expresión lógica)
{
    instrucciones
    ...
}
```

```

}

// Ejemplo:

if(nota>=5)
{
    System.out.println("Aprobado");
    aprobados++;
}

```

Las llaves se requieren sólo si va a haber varias instrucciones. En otro caso se puede crear el if sin llaves:

```

if(expresión lógica) sentencia;

// Ejemplo:

if(nota<5)
    System.out.println("Suspenso");

```

## IF - Sentencia condicional compuesta

Es igual que la anterior, sólo que se añade un apartado else que contiene instrucciones que se ejecutarán si la expresión evaluada por el if es falsa. Sintaxis:

```

if(expresión lógica)
{
    instrucciones
    ...
}
else
{
    instrucciones
    ...
}

```

Como en el caso anterior, las llaves son necesarias sólo si se ejecuta más de una sentencia. Ejemplo de sentencia if-else:

```

if(nota>=5)
{
    System.out.println("Aprobado");
    aprobados++;
}
else
{
    System.out.println("Suspenso");
}

```

```
suspensos++;
}
```

## IF - Anidación

Dentro de una sentencia if se puede colocar otra sentencia if. A esto se le llama anidación y permite crear programas donde se valoren expresiones complejas. La nueva sentencia puede ir tanto en la parte if como en la parte else.

Las anidaciones se utilizan muchísimo al programar. Sólo hay que tener en cuenta que siempre se debe cerrar primero el último if que se abrió. Es muy importante también tabular el código correctamente para que las anidaciones sean legibles.

El código podría ser:

```
if (x==1)
{
    Instrucciones
    ...
}
else
{
    if(x==2)
    {
        instrucciones
        ...
    }
    else
    {
        if(x==3)
        {
            instrucciones
            ...
        }
    }
}
```

Una forma más legible de escribir ese mismo código sería con la instrucción if-else-if:

```
if (x==1)
{
    instrucciones
    ...
}
else if (x==2)
{
    instrucciones
    ...
}
```

```

else if (x==3)
{
    instrucciones
    ...
}

```

## EJERCICIOS

 Hoja de ejercicios 1

### SWITCH

Se la llama estructura condicional compleja porque permite evaluar varios valores a la vez. En realidad sirve como sustituta de algunas expresiones de tipo if-else-if.

Su sintaxis es la siguiente:

```

switch (expresiónEntera)
{
    case valor1:
        instrucciones del valor 1
        break;
    case valor2:
        instrucciones del valor 2
        break;
    ...
    default:
        /*instrucciones que se ejecutan si la expresión no toma
        ninguno de los valores anteriores*/
}

```

Otra sintaxis de la expresión es:

```

switch (expresiónEntera){
    case valor1 -> instrucciones del valor 1;
    case valor2 -> instrucciones del valor 2;
    ....
    default -> instrucciones que se ejecutan si la expresión no toma ninguno de
los valores anteriores
}

```

Esta instrucción evalúa una expresión (que debe ser short, int, byte o char), y según el valor de la misma ejecuta instrucciones. Cada case contiene un valor de la expresión; si efectivamente la expresión equivale a ese valor, se ejecutan las instrucciones de ese case y de los siguientes.

La instrucción **break** se utiliza para salir del switch. De tal modo que si queremos que para un determinado valor se ejecuten las instrucciones de un apartado case y sólo las de ese apartado, entonces habrá que

finalizar ese case con un break.

El bloque default sirve para ejecutar instrucciones para los casos en los que la expresión no se ajuste a ningún case. Vemos un ejemplo:

```
int diasemana=1;
String texto;
switch (diasemana)
{
    case 1:
        texto="Lunes";
        break;
    case 2:
        texto="Martes";
        break;
    case 3:
        texto="Miércoles";
        break;
    case 4:
        texto="Jueves";
        break;
    case 5:
        texto="Viernes";
        break;
    case 6:
        texto="Sábado";
        break;
    case 7:
        texto="Domingo";
        break;
    default:
        texto="?";
}
```

Utilizando la otra sintaxis

```
int diasemana=1;
String texto;
texto = switch (diasemana) {
    case 1 -> "Lunes";
    case 2 -> "Martes";
    case 3 -> "Miércoles";
    case 4 -> "Jueves";
    case 5 -> "Viernes";
    case 6 -> "Sábado";
    case 7 -> "Domingo";
    default -> "?";
};
```

```

switch (diasemana)
{
    case 1:
    case 2:
    case 3:
    case 4:
    case 5:
        laborable=true; break;
    case 6:
    case 7:
        laborable=false;
}

```


Utilizando la otra sintaxis

```

switch (diasemana)
{
    case 1, 2, 3, 4, 5 -> laborable=true;
    case 6, 7 -> laborable=false;
}

```

## EJERCICIOS

 Hoja de ejercicios 2

### WHILE - Sentencia repetitiva mientras

La instrucción while permite crear bucles. Un bucle es un conjunto de sentencias que se repiten mientras se cumpla una determinada condición. Los bucles while agrupan instrucciones las cuales se ejecutan continuamente hasta que una condición que se evalúa sea falsa.

La condición se mira antes de entrar dentro del while y cada vez que se termina de ejecutar las instrucciones del while. Sintaxis:

```

while (expresión lógica)
{
    sentencias que se ejecutan si la condición es true
}

```

El programa se ejecuta siguiendo estos pasos:

1. Se evalúa la expresión lógica
2. Si la expresión es verdadera ejecuta las sentencias, sino el programa abandona la sentencia while
3. Tras ejecutar las sentencias, volvemos al paso 1

Ejemplo (escribir números del 1 al 100):

```
int i=1;
while (i<=100)
{
    System.out.println(i);
    i++;
}
```

## Bucles con contador

Se llaman así a los bucles que se repiten una serie determinada de veces. Están controlados por un contador (o incluso más de uno). El contador es una variable que va variando su valor (de uno en uno, de dos en dos,...) en cada vuelta del bucle. Cuando el contador alcanza un límite determinado, entonces el bucle termina.

En todos los bucles de contador necesitamos saber:

1. Lo que vale la variable contadora al principio. Antes de entrar en el bucle
2. Lo que varía (lo que se incrementa o decrementa) el contador en cada vuelta
3. Las acciones a realizar en cada vuelta del bucle
4. El valor final del contador. En cuanto se rebase el bucle termina. Dicho valor se pone como condición del bucle, pero a la inversa; es decir, la condición mide el valor que tiene que tener el contador para que el bucle se repita y no para que termine.

Vemos un ejemplo:

```
i=10; //Valor inicial del contador, empieza valiendo 10 (por supuesto i debería
estar declarada como entera, int)
while (i<=200)
{ //condición del bucle, mientras i sea menor de 200, el bucle se repetirá,
  cuando i rebase este valor, el bucle termina
    System.out.println(i); //acciones que ocurren en cada vuelta del bucle.
                           //En este caso simplemente escribe el valor
del contador
    i+=10;                //Variación del contador, en este caso cuenta de
10 en 10
}
```

## Bucles con centinela

Es el segundo tipo de bucle básico. Se trata de preguntar a cada ciclo del bucle por una condición lógica. Si esta condición se cumple, continuamos otro ciclo más en el bucle. Cuando la condición lógica deja de cumplirse, se sale del bucle.

A la condición lógica, que se la llama centinela, puede ser desde una simple variable booleana hasta una expresión lógica más compleja, y sirve para decidir si el bucle se repite o no. De modo que cuando la condición lógica se incumpla, el bucle termina.

Esa expresión lógica a cumplir es lo que se conoce como centinela y normalmente la suele realizar una variable booleana.

Ejemplo:

```
boolean salir=false; /* En este caso el centinela es una variable booleana que
inicialmente vale falso */
int n;
while(salir==false)    // Condición de repetición: que salir siga siendo falso.
Ese es el centinela.
{    //También se podía haber escrito simplemente: while(!salir)
    n=(int)Math.floor(Math.random()*499+1); // Lo que se repite en el bucle:
    System.out.println(i); // calcular un número aleatorio de 1 a 500 y
    escribirlo
    salir=(i%7==0);    //El centinela vale verdadero si el número es múltiplo
de 7
}
```

Un bucle podría ser incluso mixto: de centinela y de contador.

Ejercicio para hacer: un programa que escriba números aleatorios de uno a 500 y se repita hasta que llegue un múltiplo de 7, pero que como mucho se repite cinco veces. ¡¡CODIFÍCALO!!

NOTA: Hoja de ayuda para los números aleatorios. [Generación de números aleatorios.](#)

## DO WHILE - Sentencia repetitiva hacer mientras

La única diferencia respecto a la anterior está en que la expresión lógica se evalúa después de haber ejecutado las sentencias. Es decir el bucle al menos se ejecuta una vez. Es decir los pasos son:

1. Ejecutar sentencias
2. Evaluar expresión lógica
3. Si la expresión es verdadera volver al paso 1, sino continuar fuera del while

Sintaxis:

```
do
{
    instrucciones
} while (expresión lógica)
```

Ejemplo (contar de uno a 1000):

```
int i=0;
do
{
    i++;
```



```
System.out.println(i);
} while (i<1000);
```

Se utiliza cuando al menos las sentencias del bucle se van a repetir una vez (en un bucle while puede que incluso no se ejecuten las sentencias que hay dentro del bucle si la condición fuera falsa, ya desde un inicio).

De hecho cualquier sentencia do..while se puede convertir en while.

Ejercicio para hacer: Intenta codificar el ejemplo anterior utilizando una estructura while

## EJERCICIOS



Hoja de ejercicios 3

### FOR - Sentencia repetitiva para

Es un bucle más complejo especialmente pensado para rellenar arrays (estructura que veremos más adelante) o para ejecutar instrucciones controladas por un contador. Es decir, es un bucle que se utiliza cuando se sabe a priori cuantos ciclos va a ejecutarse. Una vez más, se ejecutan una serie de instrucciones en el caso de que se cumpla una determinada condición.

Sintaxis:

```
for(inicialización;condición;incremento)
{
    sentencias
}
```

Antes de entrar en el bucle, se ejecuta la instrucción de inicialización. Se entra en el bucle mientras la condición sea verdadera. Y en cada vuelta se ejecuta la instrucción de incremento. Es decir, el funcionamiento es:

1. Se ejecuta la instrucción de inicialización
2. Se comprueba la condición
3. Si la condición es cierta, entonces se ejecutan las sentencias. Si la condición es falsa, abandonamos el bloque for
4. Tras ejecutar las sentencias, se ejecuta la instrucción de incremento y se vuelve al paso 2

Ejemplo (contar números del 1 al 1000):

```
for(int i=1;i<=1000;i++)
{
    System.out.println(i);
}
```

La ventaja que tiene es que el código se reduce. La desventaja es que el código es menos comprensible. El bucle anterior es equivalente al siguiente bucle while:

```
int i=1; /*sentencia de inicialización*/
while(i<=1000) /*condición*/
{
    System.out.println(i);
    i++; /*incremento*/
}
```

Como se ha podido observar, es posible declarar la variable contadora dentro del propio bucle for. De hecho es la forma habitual de declarar un contador. De esta manera se crea una variable que muere en cuanto el bucle acaba.

## EJERCICIOS



Hoja de ejercicios 4

## ANEXO DEL TEMA. BUCLES ANIDADOS

### ¿EN QUE CONSISTEN?

Se trata de usar una estructura de bucle dentro de otra ya existente. Estas pueden ser cualesquiera de las estructuras repetitivas vistas hasta ahora.

- While
- Do
- For

Puedo anidar cualquier número de ellas unas dentro de otras, aunque lo más normal es no pasar de 3.

Veamos este ejemplo. Calcular el factorial de n números introducidos por teclado. La introducción finaliza al introducir el 0.

```
Scanner entrada = new Scanner(System.in);
int num, fact;

//Intro datos
System.out.print("Introduce el número: ");
num=entrada.nextInt();

//Calculo
while (num !=0) {
    fact=1;
    for (int i = num; i > 0; i--) {
        fact*=i;
    }
    System.out.println("El factorial del número "+ num+" es: " +fact);
}
```

```

        //Vuelvo a pedir el número
        System.out.print("Introduce el número: ");
        num=entrada.nextInt();
    }
    System.out.println("*****FIN*****");

```

Veamos otro ejemplo. Dibujamos un cuadrado con \* cuyo número de \* por lado será pedido por teclado.

```

Scanner entrada = new Scanner(System.in);
int numAsteriscosLado;
System.out.print("Introduce el número de asteriscos por lado: ");
numAsteriscosLado=entrada.nextInt();

//Dibujamos la parte de arriba del cuadrado
for(int cont=0;numAsteriscosLado>cont;cont++){
    System.out.print("*");
}
System.out.println("");

//Usamos un bucle anidado para dibujar los asteriscos del medio
//Calcula las filas intermedias poniendo un * al inicio y final de llas.
for(int cont=1;(numAsteriscosLado-2)>=cont;cont++){
    System.out.print("*");
    //Este bucle dibuja los espacio entre el primer y ultimo asterisco
    //de cada una de las filas.
    for (int i=0;(numAsteriscosLado-2)>i;i++){
        System.out.print(" ");
    }
    System.out.print("*");
    System.out.println("");
}

//Dibujamos la parte de abajo del cuadrado
for(int cont=0;numAsteriscosLado>cont;cont++){
    System.out.print("*");
}
System.out.println("");

```

## EJERCICIOS



Hoja de ejercicios 5