# DESARROLLO DE APLICACIONES WEB

# Unidad 5

## Estructuras de Control

*1r DAW*

*IES La Mola de Novelda*

*Departament d'informàtica*

Profesor: Joan Carbonell Picó

# ÍNDEX

# Unidad 5: Estructuras de Control

## 1.- ESTRUCTURA SECUENCIAL

El orden en que se ejecutan por defecto las sentencias de un programa es secuencial. Es decir, **las sentencias se ejecutan una después de otra**, en el orden en que aparecen escritas dentro del programa.

Cada una de las instrucciones están **separadas por el carácter punto y coma** (**;**). Las instrucciones se suelen agrupar en bloques.

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

```
{
    instrucción 1;
    instrucción 2;
    instrucción 3;
}
```

En Java si el bloque de sentencias está constituido por **una única sentencia no es obligatorio** el uso de las llaves de apertura y cierre (**{ }**), aunque **sí recomendable**.

- Ejemplo: Realiza un programa que pida dos números enteros. Visualízalos. A continuación, realiza un **intercambio de** esas dos **variables** (en la variable del primero pones el segundo y al contrario) y visualízalas para comprobar que se ha realizado correctamente el intercambio.

  En el ejemplo, todas la instrucciones ejecutadas son secuenciales.

## 2.- ESTRUCTURA DE SELECCIÓN

La estructura de selección o condicional permite al programa bifurcar el flujo de ejecución de instrucciones dependiendo del valor de una expresión (condición). Podemos diferenciar las siguientes:

### 2.1.-SELECCIÓN SIMPLE


Se evalúa la condición y si ésta se cumple se ejecuta una determinada acción o grupo de acciones. En caso contrario se saltan dicho grupo de acciones.

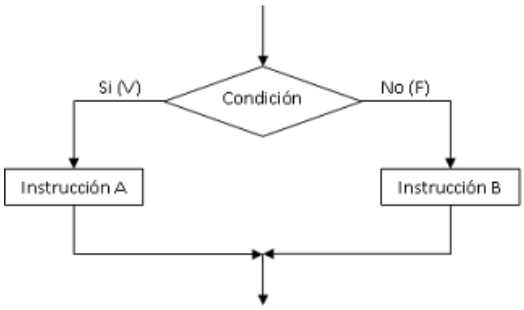
```
if (expresión_booleana) {
        instrucción 1
        instrucción 2
        .....
}
```

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

- Ejemplo1: Realiza un programa que pida la edad del usuario y que determine si es mayor de edad o no.

- Ejemplo2: Realiza un programa que pida el precio de un televisor. Los televisores de más de 999€ tienen un descuento del 10%. En pantalla tiene que aparecer el precio del televisor total y si se le ha aplicado un descuento.

## 2.2.-SELECCIÓN DOBLE




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

```
if (expresión booleana) {
        instrucciones 1
}
else{
        instrucciones 2
}
```

### 2.2.1.- Operador condicional ? :

Se puede utilizar en sustitución de la sentencia de control **if-else**. Los forman los caracteres ? i :

Se utiliza de la forma siguiente: *expresión1* **?** *expresión2* **:** *expresión3;*

Si **expresión1 es cierta entonces se evalúa expresión2** y éste será el valor de la expresión condicional. Si **expresión1 es falsa, se evalúa expresión3** y éste será el valor de la expresión condicional.

- Ejemplo1: Programa que pide por teclado un número y devuelve si es par o impar. Realiza dos versiones del programa. Una con el **if-else** y la otra con el **?**.

- Ejemplo2: Realiza un programa que pida una un valor secreto. El programa ha de visualizar si el valor introducido es el valor secreto o no. El valor secreto lo almacenaremos en una variable que no se podrá modificar

## 2.3.-SELECCIÓN MÚLTIPLE

Se obtiene anidando sentencias **if ... else**. Permite construir estructuras de selección más complejas. Cada **else** se corresponde con el **if** más próximo que no haya sido emparejado.

```
if (expresion_booleana1){
        instrucciones;
}
else if (expresion_booleana2){
        instrucciones;
}
else{
        instrucciones;
}
```

Una vez que se ejecuta un bloque de instrucciones, la ejecución continúa en la siguiente instrucción que aparezca después de las sentencias **if .. else** anidadas.

- Ejemplo1: Programa que muestra un saludo según la hora indicada por el usuario (Buenos días, Buenas tardes, Buenas noches, Hora no válida).
- Ejemplo2: Realiza un programa que lea una nota numérica de 1 a 10 y visualice su nota en forma textual (insuficiente, suficiente, bien, notable y sobresaliente).

## 2.4.-SELECCIÓN MÚLTIPLE. INSTRUCCIÓN SWITCH.

Se utiliza para seleccionar una de entre múltiples alternativas. La forma general de la instrucción **switch** en Java es la que muestra la figura.

```
switch (expresión){
        case valor 1:
                instrucciones;
                break;
        case valor 2:
                instrucciones;
                break;
        . . .
        default:
                instrucciones;
}
```

La instrucción **switch** se puede usar con datos de tipo *byte*, *short*, *char* e *int*. También con tipos enumerados y con las clases envolventes *Character*, *Byte*, *Short* e *Integer*.

A partir de **Java 7** también pueden usarse datos de tipo *String* en un **switch**.

- Ejemplo: Programa que pide el número del mes y muestra el nombre correspondiente.
- Vuelve a hacer el mismo ejemplo pero con el condicional múltiplo **if ... else**
- Indica las diferencias que hay entre hacer el problema con un **switch** o con condicionales anidados.
- Ejemplo2: Programa que solicite el número de un mes y visualice el números de días que tiene.
- Ejemplo3: Un uso muy típico de la sentencia *switch* es para generar de manera sencilla

menús en que el usuario puede elegir entre diferentes opciones. Según la opción elegida, se ejecutan directamente las instrucciones asociadas a cada una. Haz un programa en que, a partir de dos números enteros, se pida qué operación se quiere hacer entre estas: sumar, restar, multiplicar o dividir.

## 3.- ESTRUCTURAS DE REPETICIÓN

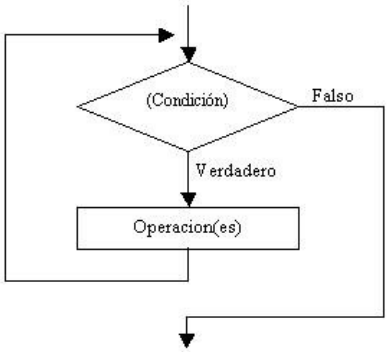
Permiten ejecutar de forma repetida un bloque específico de instrucciones. Las instrucciones se repiten mientras o hasta que se cumpla una determinada condición. Esta condición se conoce como **condición de salida**.

### 3.1.-TIPOS DE ESTRUCTURAS REPETITIVAS.

#### 3.1.1.- while

```
while(condicion){
    sentencias;
}
```

Las instrucciones se repiten mientras la condición sea cierta. La **condición** se comprueba **al principio** del bucle por lo que las acciones se pueden ejecutar **0 o más veces**.

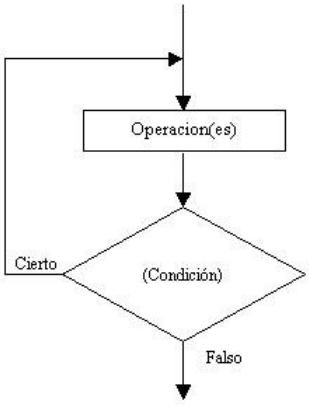



- Ejemplo1: Realiza un programa que visualice los 10 primeros números enteros positivos.
- Ejemplo2: Realiza un programa que escriba una línea horizontal con un carácter determinado. El programa tendrá que solicitar la longitud de la línea y el carácter a utilizar para dibujar la línea.

#### 3.1.2.- do .. while

```
do{
    sentencias;
}while(condicion);
```

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



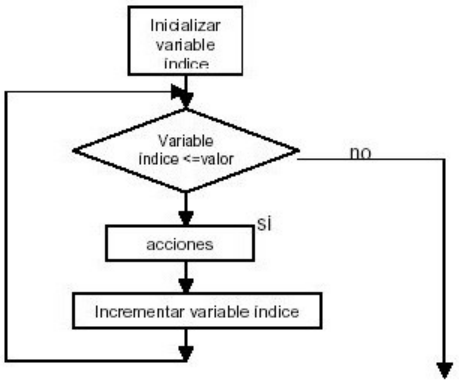

- Ejemplo1: Realiza un programa que calcule el perímetro, el área y el diámetro de un círculo. El usuario tiene que introducir el radio i este tiene que ser positivo.

- Ejemplo2: Realiza un programa que solicite un número que está dentro del rango [1,10]. Si no es el caso, tendrá que volverlo a solicitar hasta que sea un valor válido. Además, el programa tendrá que visualizar la cantidad de intentos que ha probado el usuario hasta que ha introducido un valor válido.

### 3.1.3.- for

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

```
for( inicialización; condición; incremento/decremento){
        instrucción 1;
        .....
        instrucción N;
}
```




- Ejemplo1: Realiza un programa que visualice los 10 primeros números enteros positivos.
- Ejemplo2: Realiza un programa que solicite un número y que visualice la tabla de multiplicar de ese número del 1 al 10.

### 3.1.4.- foreach o for extendido

Esta forma de uso del **for** facilita el recorrido de objetos existentes en una colección sin necesidad de definir el número de elementos a recorrer. La sintaxis que se emplea es:

```
for ( TipoARecorrer nombreVariableTemporal : nombreDeLaColección ) {
        instrucciones
}
```

Esta estructura la utilizaremos cuando estudiemos el tema de estructuras estáticas y dinámicas.

## 3.2.-BUCLES INFINITOS

Java permite la posibilidad de construir bucles infinitos, los cuales se ejecutarán indefinidamente, a no ser que provoquemos su interrupción. Tres ejemplos:

```
for(;;){
    instrucciones
}
```

```
for(;true;){
    instrucciones
}
```

```
while(true){
    instrucciones
}
```

Los bucles infinitos también suelen producirse de forma involuntaria al no modificar correctamente la condición de salida de un bucle. Es un ERROR común al iniciarse en Programación.

### 3.3.-BUCLES ANIDADOS

Bucles anidados son aquellos que incluyen instrucciones ***for***, ***while*** o ***do-while*** unas dentro de otras. Debemos tener en cuenta que las variables de control que utilicemos deben ser distintas.

Los anidamientos de estructuras tienen que ser correctos, es decir, que una estructura anidada dentro de otra lo debe estar totalmente.

- Ejemplo: Realiza un programa que visualice las tablas de multiplicar del 1 al 5.

## 4.- ESTRUCTURAS DE SALTO

Las estructuras de salto son instrucciones que nos permiten romper con el orden natural de ejecución de nuestros programas, pudiendo saltar a un determinado punto o instrucción.

### 4.1.-BREAK

La sentencia ***break*** se utiliza para interrumpir la ejecución de una **estructura de repetición** o de un ***switch***. Cuando se ejecuta el ***break***, el flujo del programa continúa en la sentencia inmediatamente posterior a la estructura de repetición o al *switch.*

```
for (int i = 0; i < 10; i++) {
    System.out.println("Dentro del bucle");
    break;
    System.out.println("Nunca lo escribira");
}
System.out.println("Tras el bucle");
```

Cuando el programa alcance la sentencia ***break***, detendrá la ejecución del bucle y saldrá de él. Por lo tanto el resultado de la ejecución del programa será:

**Dentro del Bucle**
**Tras el bucle**

## 4.2.-CONTINUE

La sentencia ***continue*** únicamente puede aparecer dentro de una **estructura de repetición**. El efecto que produce es que se deja de ejecutar el resto del bucle para volver a evaluar la condición del bucle, continuando con la siguiente iteración, si el bucle lo permite.

```
for (int i = 0; i < 10; i++) {
    System.out.println("Dentro del bucle");
    continue;
    System.out.println("Nunca lo escribira");
}
```

Cada vez que el programa alcance la sentencia ***continue***, saltará a una nueva iteración del bucle y por lo tanto la última sentencia **System.out.println** nunca llegará a escribirse.

El resultado del programa será escribir 10 veces “**Dentro del bucle**

## 4.3.-RETURN

La sentencia ***return*** se utiliza para terminar la ejecución de un método o función. De esta forma volvemos a la instrucción que hizo la llamada a dicho método.

Con la instrucción ***return*** podemos devolver un valor o no:

- Devuelvo valor: ***return*** variable;
- No devuelvo nada: ***return***;