UNIDAD 3

Estructuras repetitivas o bucles

Contenido

- **Sentencias de repetición o bucles**
- **Bucles controlados por condición**
  - **while**
  - **do-while**
- **Bucles controlados por contador: for**
- **Sentencias o Salidas anticipadas: break**
- **Sentencias o Salidas anticipadas: continue**
- **Bucles anidados: independientes**
- **Bucles anidados: dependientes**

Sentencias de repetición o bucles

Los bucles o sentencias repetitivas modifican el flujo secuencial de un programa permitiendo la ejecución reiterada de una sentencia o bloque de sentencias.

En Java hay tres tipos diferentes de bucles:

- while
- do-while
- for

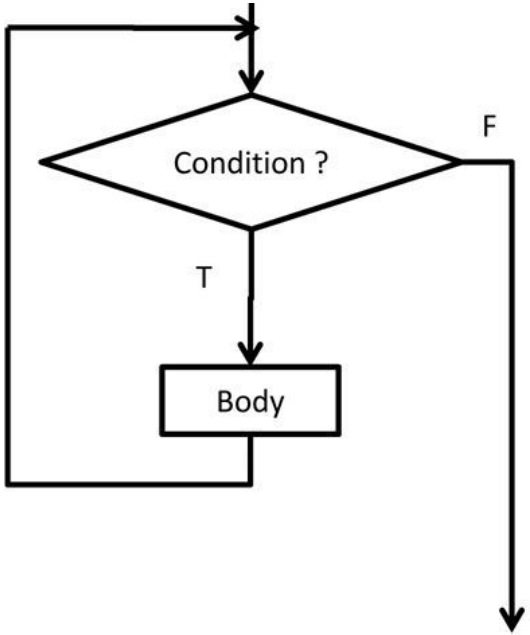
Iteración: cada vez que se repite el bucle, es decir, cada vez que se ejecuta el bloque de sentencias incluidas dentro del bucle.

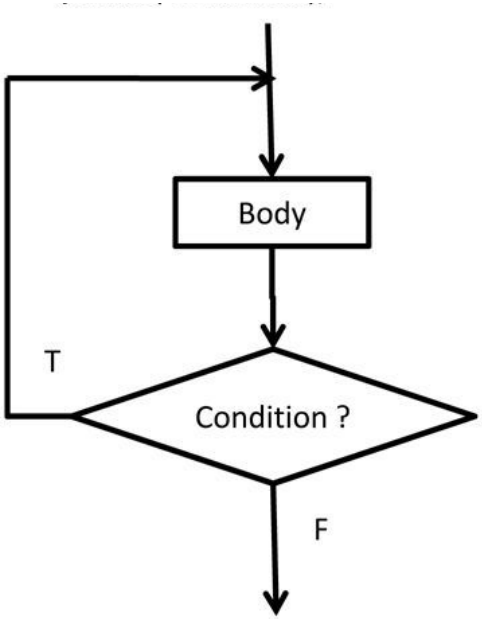
Bucles controlados por condición

Es como un if en el que una condición al evaluarse como cierta, indica que se debe ejecutar un bloque de código.

La diferencia es que cuando acaba ese bloque, vuelve a evaluar la condición y si es cierta, vuelve a ejecutar el mismo bloque de código y así hasta que la condición sea falsa.

Existen dos formas, una en el que el bloque está detrás de la condición y otra en la que está antes.

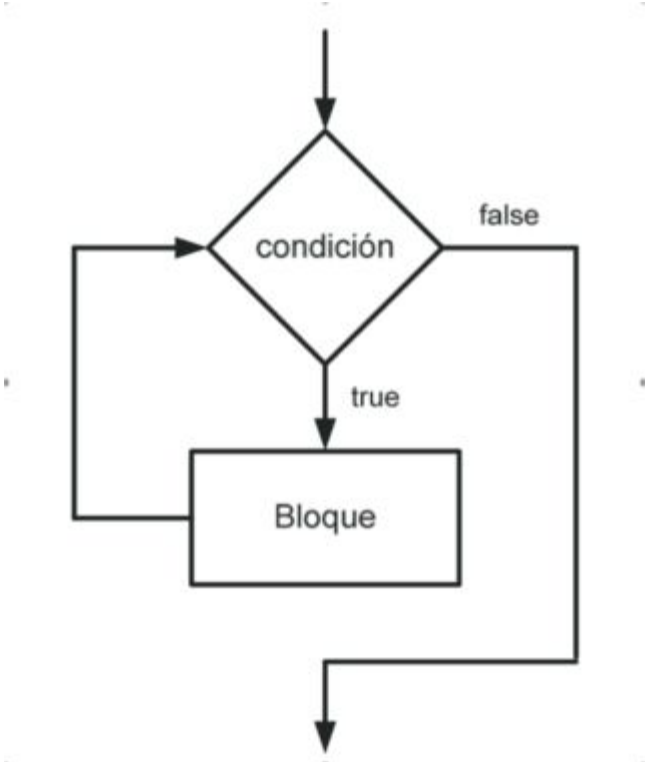




while

Es un bucle o sentencia repetitiva con una condición al principio. Se ejecuta una sentencia mientras sea cierta una condición. La sentencia puede que no se ejecute ni una sola vez.

```
while (condición) {
    sentencias;   //Bloque de instrucciones
    ...
}
```



Ejemplo while

```
// Contamos hasta 5
int contador = 1;
while (contador < =5) {
    System.out.println(contador);
    contador++;
}
```

```
// Suma números hasta que el usuario introduce un 0.
Scanner sc = new Scanner(System.in);
int suma = 0;
int numero;
System.out.println("Introduces números a sumar. Pon un 0 para finalizar");
System.out.print("Número: ");
numero = sc.nextInt();
while (numero > 0) {
        suma += numero;
        System.out.print("Número: ");
        numero = sc.nextInt();
}
System.out.println("La suma es " + suma);
```

Ejemplo while

Ejemplo de while que nunca llega a ejecutarse, se realizan **cero iteraciones**.

```
int cuentaAtras = -8; //valor negativo
   while (cuentaAtras >= 0) {
   ...
}
```

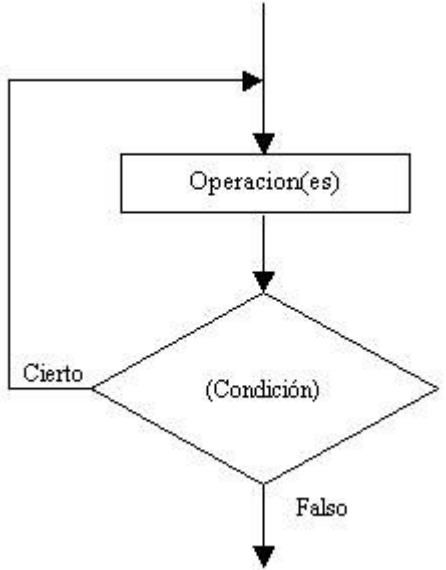
Ejemplo de while que se conoce como **bucle infinito**, ya que nunca para.

```
int cuentaAtras = 10;
while (cuentaAtras >= 0) {
   System.out.println(cuentaAtras);
}
```

do-while

Primero se ejecuta el bloque de instrucciones y después se evalúa la condición para decidir si se vuelve a ejecutar el bloque de instrucciones o continúa fuera del bucle.

```
do {
    sentencias;   //Bloque de instrucciones
} while (condición);
```



Nota: se ejecuta al menos una vez.

Ejemplo do-while

```
// Contamos hasta 5
int contador = 1;
do {
    System.out.println(contador);
    contador++;
} while (contador < =5);
```

```
// Suma números hasta que el usuario introduce un 0.
Scanner sc = new Scanner(System.in);
int suma = 0;
int numero;
System.out.println(“Introduces números a sumar. Pon un 0 para finalizar”);
do {
        System.out.print(“Número:  “);
        numero = sc.nextInt();
        suma += numero;
} while (numero > 0);
System.out.println(“La suma es “ + suma);
```

Cosas importantes sobre do-while

- Es el único bucle que termina en punto y coma ;

- Mientras el bucle while se puede ejecutar de 0 a infinitas veces, el do-while lo hace de 1 a infinitas veces

- La única diferencia con el bucle while es que do-while se ejecuta, al menos, una vez.

Bucle controlado por contador. for

El bucle for permite controlar el número de iteraciones mediante una variable que suele recibir el nombre de contador que se pone en la parte de la inicialización. La sintaxis de la estructura for es la siguiente:

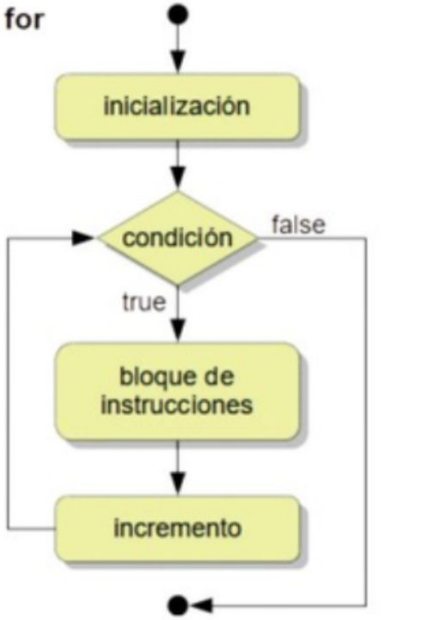
```
for (inicialización; condición; incremento) {
   bloque de instrucciones
   ...
}
```

Inicialización: lista de instrucciones separadas por coma donde generalmente se inicializan las variables (normalmente sólo una) que controlan el bucle. Se ejecuta una sola vez antes de comenzar el bucle.

Condición: expresión booleana que se evalúa para hacer otra iteración (true) o finalizar el bucle (false). Se evalúa antes de cada iteración.

Incremento: lista de instrucciones separadas por coma donde se modifican las variables que controlan el bucle. Se ejecuta al final de cada iteración.

Bucle controlado por contador. for



Ejemplo for

```
// Contamos hasta 5
for (int contador = 1; contador < =5; contador++) {
    System.out.println(contador);
}

// Números pares menores que 100
for (int contador = 2; contador < 100; contador+=2) {
    System.out.println(contador);
}

// Doble contador, uno incrementando y otro decrementando.
for (int i = 0, j = 10; i < 10 && j > 0; i++, j--) {
    System.out.println("i = " + i + " :: " + "j = " + j);
}
```

¡Cuidado!

Si declaramos la variable contador dentro del for, solo estará disponible dentro del bucle. Pero ya no funciona fuera de él.

```
for (int contador = 1; contador < =5; contador++) {
    System.out.println(contador);
}
System.out.println(contador);  // Da un error de que no encuentra la variable.
```

Esto es debido al ámbito de las variables que se verá en temas posteriores.

¡Truco!

Aunque en un for **lo deseable es declarar la variable dentro del for**. Si queremos no perder la variable cuando acabe el bucle, podemos declararla antes e iniciarla en el for.

```
int contador;
for (contador = 1; contador < =5; contador++) {
    System.out.println(contador);
}
System.out.println(contador);  // Ahora funciona.
```

Sentencias o salidas anticipadas: break

En algunas ocasiones es preferible acabar un bucle sin tener que esperar a comprobar la condición. Es decir, interrumpir (romper) el bucle.

Para ello usamos la palabra reservada **break** que interrumpe completamente la ejecución de un bucle.

```
i = 1;
while (i <= 10) {
   System.out.println("La i vale" + i);
   if (i == 2) {
      break;
   }
   i++;
}
```

El bucle para cuando i vale 2.

Sentencias o salidas anticipadas: continue

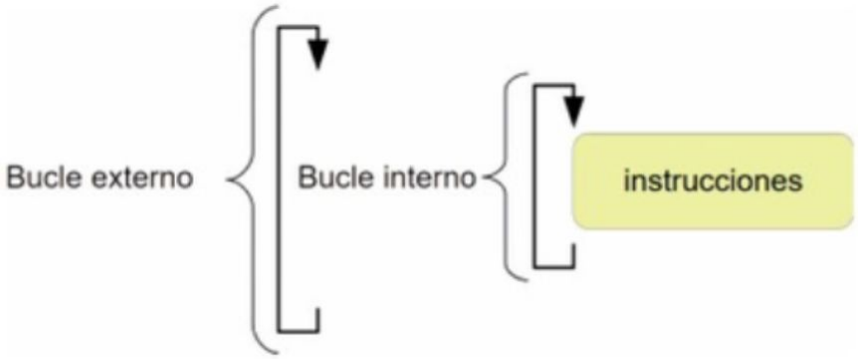
También es posible interrumpir la iteración actual y pasar a la siguiente (siempre que la condición sea cierta). Para ello usamos la palabra reservada **continue** que detiene la iteración actual y no sigue ejecutando las sentencias del bloque.

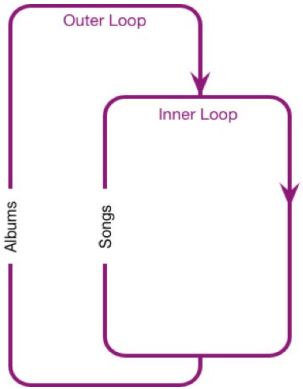
```
i = 0;
while (i < 10) {
   i++;
   if (i % 2 == 0) { //si i es par
      continue;
   }
   System.out.println("La i vale " + i);
}
```

En este caso, si el número es par, se interrumpe antes de ejecutar imprimir por consola. Por lo que sólo se mostrarán los impares.

Bucles anidados

- La anidación de bucles es cuando un bucle se encuentra dentro de otro.
- Se realiza cuando queremos recorrer una secuencia que internamente tienen otra secuencia repetitiva.
- Por ejemplo si queremos mostrar todas las canciones de todos los álbumes que tenemos en una colección, primero iremos seleccionando uno a uno cada álbum e imprimiendo cada canción.
- Al realizar los bucles anidados el número de iteraciones se multiplica.

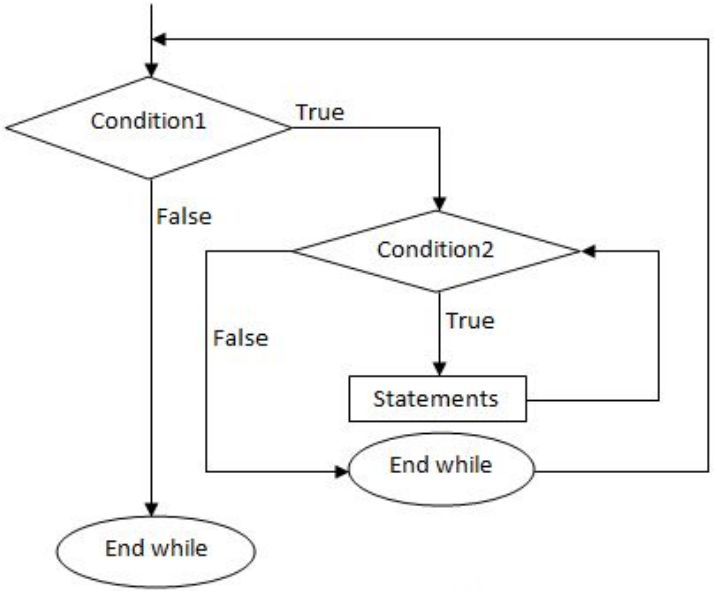




Bucles anidados

Este es el esquema básico de dos bucles anidados en las que hay dos condiciones, una para controlar el bucle externo y otra para el interno.

Pueden anidarse tantos bucles como se quiera, pero esto aumenta la complejidad del programa. (Se puede incrementar el número de iteraciones exponencialmente).



bucles anidados independientes

Cuando los bucles anidados no dependen, en absoluto, unos de los otros para determinar el número de iteraciones, se denominan blucles anidados independientes. Por ejemplo:

```
for (i = 1; i <= 4; i++) {
   for (j = 1; j <= 3; j++) {
      System.out.println("Ejecutando...");
   }
}
```

El bucle externo, controlado por la variable `i`, realizará cuatro iteraciones, donde `i` toma los valores 1, 2, 3 y 4. En cada una de ellas, el bucle interno, controlado por `j`, realizará tres iteraciones, tomando `j` los valores 1, 2 y 3. En total, el bloque de instrucciones se ejecutará doce veces.

bucles anidados dependientes

Puede darse el caso de que el número de iteraciones de un bucle interno no sea independiente de la ejecución de los bucles exteriores, y dependa de sus variables de control. Decimos entonces que son bucles anidados dependientes.

En el siguiente ejemplo, la variable utilizada en el bucle externo i se compara con la variable j que controla el bucle más interno.

```
for (i = 1; i <= 3; i++) {
   System.out.println("Bucle externo, i=" +i);
   j = 1;
   while (j <= i) {
      System.out.println("...Bucle interno, j=" + j);
      j++;
   }
}
```