

Bootcamp Java Developer

Fase 1 - Java Analyst
Módulo 7



Ciclos

Introducción

Los ciclos, también conocidos como “estructuras repetitivas”, **permiten ejecutar varias veces los mismos bloques o sentencias mientras se cumplan una o varias condiciones.**

A cada entrada en el ciclo (bucle) se le conoce como ***iteración***.

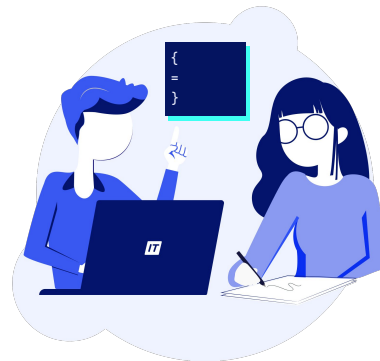
Existen **dos clasificaciones**:

Indeterminados

No sabemos a ciencia cierta la cantidad de iteraciones que realizará, estos ciclos son el **`while` y `do while`**.

Determinados

Son aquellos ciclos que sabemos de antemano la cantidad de iteraciones máximas que va a tener por ejemplo el ciclo **`for`**.



Ejemplo

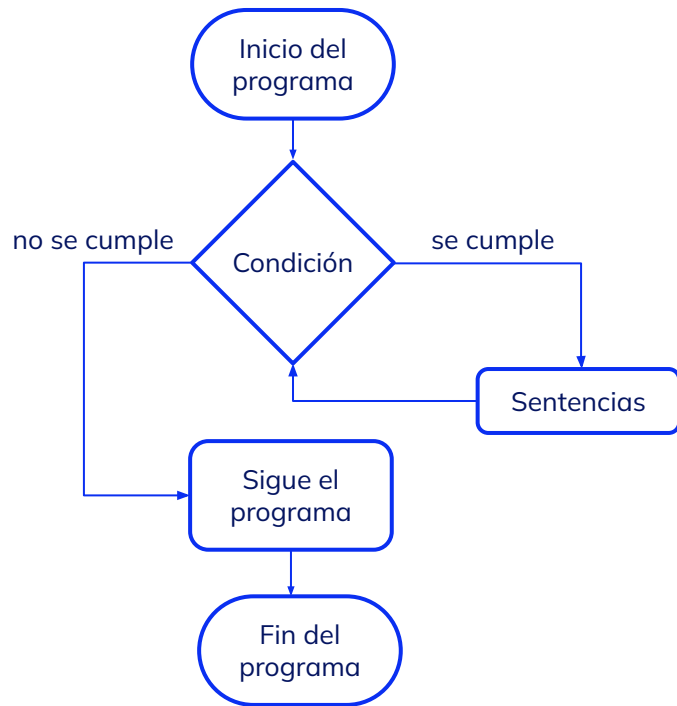
```
int num1, num2;
num1 = 0;
num2 = 5;

while (num1 <= num2) {
    System.out.println("Veces dentro del ciclo:" + (num1 + 1));
    num1++;
}

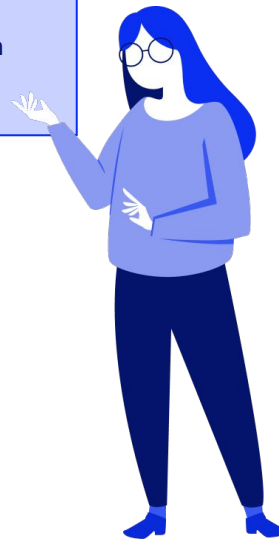
int num = 0;
boolean seguir = true;

while(seguir){
    System.out.println("Veces dentro del ciclo:" + (num + 1));
    if (num == 3) {
        seguir = false;
    }
}
```

Ciclo



Algo muy importante en los ciclos es que se debe **garantizar que en algún momento deje de cumplirse la condición**. De esa manera, se podrá salir de él; de lo contrario quedaría en un bucle infinito.



While

Esta estructura es muy similar a la que vimos en el condicional `if`, la diferencia radica en que **si se cumple la condición ejecutará las sentencias y/o bloques que se encuentren dentro**, volverá al inicio y evaluará la condición nuevamente. Esto se realizará hasta que la condición deje de cumplirse.

```
// con llaves
while (condicion) {
    sentencia1;
    sentencia2;
}

// sin llaves
while (condicion)
    sentenciaUnica;
```

Do while

Trabaja de forma muy similar al `while`. La única diferencia es que la **condición se evalúa al final del bloque**.

Esto permite que las **sentencias se ejecuten al menos una vez**, a diferencia del “`while`” que evalúa la condición al principio y puede que nunca se ejecuten las sentencias que contiene.

```
// con llaves
do {
    sentencia1;
    sentencia2;
} while (condicion);

// sin llaves
do
    sentenciaUnica;
while (condicion);
```

Ejemplo

D

```
int num1, num2;  
num1 = 0;  
num2 = 5;  
  
do {  
    System.out.println("Veces dentro del ciclo:" + (num1 + 1));  
    num1++;  
} while (num1 <= num2);  
  
int num = 0;  
boolean seguir = true;  
  
do {  
    System.out.println("Veces dentro del ciclo:" + (num + 1));  
    if (num == 3) {  
        seguir = false;  
    }  
} while (seguir);
```


For

En el siguiente código se observa que, aunque sabemos que el **while** está entre los ciclos indeterminados, **la cantidad máxima de iteraciones es 10**, además, para que el bucle no se vuelva infinito, la variable a comparar en el condicional se incrementa al final del bloque.

```
int num1 = 0;

while (num1 < 10) {
    System.out.println("Veces dentro del ciclo:" + (num1 + 1));
    num1++;
}
```



Para este tipo de situaciones tenemos el ciclo **for**, que posee una forma más cómoda de hacer lo mismo.

```
// con llaves
for (inicio; condicion; despuesSentencias) {
    sentencia1;
    sentencia2;
}

// sin llaves
for (inicio; condicion; despuesSentencias)
    sentenciaUnica;
```

Esta estructura tiene algo en particular: se puede omitir lo que **contiene entre los paréntesis**, pero sin dejar de colocar los **punto y coma**.

```
// con llaves
for ( ; ; ) {
    sentencia1;
    sentencia2;
}
```

Ejemplo

```
for (int num2 = 0; num2 < 10; num2++) {  
    System.out.println("Veces dentro del ciclo:" + (num2 + 1));  
}
```



De esta manera, se logra que las sentencias que están dentro del bloque se ejecuten infinitamente, ¿por qué la estructura permite esto?

Algunas veces, según la solución, necesitamos que el bloque de incremento o bloque **“despuesSentencias”** se ejecute antes, otras veces, que la variable inicial sea usada después de haber terminado el ciclo.

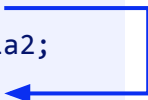
```
int num2 = 0
for ( ; num2 < 10; ) {
    num2++;
    System.out.println("Veces dentro del ciclo:" + (num2 + 1));
}
System.out.println("Valor final de la variable" + num2);
```



Break y Continue


Como ya se mencionó en la definición del condicional switch, la sentencia “**break**” provoca que la ejecución salga de la estructura. En este caso, hace exactamente lo mismo: **le indica al programa que salga del ciclo inmediato al que pertenece.**

```
ciclo {  
    sentencia1;  
    break;  
    sentencia2;  
}
```



La otra sentencia “**continue**” le indica al programa que **omita todas las instrucciones que se encuentren después de esta palabra dentro del bucle** y salte a la siguiente iteración.

```
ciclo {  
    sentencia1;  
    continue;  
    sentencia2;  
}
```



**¡Sigamos
trabajando!**