

UNIDAD 5


BUCLES EN JAVA


**Programación
CFGS DAW**


Paco Aldarias
paco.aldarias@ceedcv.es
2020/2021
13/11/20 14:07:32

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

ÍNDICE DE CONTENIDO

1. Introducción.....	3
2. Bucle for.....	4
3. Bucle while.....	6
4. Bucle do-while.....	8
5. Ejemplos.....	10
5.1 Ejemplo 1.....	10
5.2 Ejemplo 2.....	12
6. Agradecimientos.....	13

UD05. BUCLES EN JAVA

1. INTRODUCCIÓN

Los bucles son estructuras de repetición, bloques de instrucciones que se repiten un número de veces mientras se cumpla una condición o hasta que se cumpla una condición.


Un bloque de instrucciones se encontrará encerrado mediante llaves {...} si existe más de una instrucción al igual que suceden las estructuras alternativas (if... else... etc).


Existen tres construcciones para estas estructuras de repetición:


- Bucle *for*
- Bucle *while*
- Bucle *do-while*

Todo problema que requiera repetición puede hacerse con cualquiera de los tres, pero según el caso suele ser más sencillo o intuitivo utilizar uno u otro.

Como regla general es recomendable:

 Utilizar el bucle **for** cuando se conozca de antemano el número exacto de veces que ha de repetirse el bloque de instrucciones.

 Utilizar el bucle **while** cuando no sabemos el número de veces que ha de repetirse el bloque y es posible que no deba ejecutarse ninguna vez.

 Utilizar el bucle **do-while** cuando no sabemos el número de veces que ha de repetirse el bloque y deberá ejecutarse al menos una vez.

Estas reglas son generales y algunos programadores se sienten más cómodos utilizando principalmente una de ellas. Con mayor o menor esfuerzo, puede utilizarse cualquiera de las tres indistintamente.

2. BUCLE FOR

El bucle *for* se codifica de la siguiente forma:

Código	Ordinograma
<pre>for (inicialización ; condición ; incremento) { bloque acciones; }</pre>	<pre>graph TD Start([Iniciar contador]) --> Condicion{Condicion} Condicion -- Verdadero --> Acciones[Acciones] Acciones --> Incrementar[Incrementar contador] Incrementar --> Condicion Condicion -- Falso --> Exit([Fin])</pre>

La cláusula ***inicialización*** es una instrucción que se ejecuta una sola vez al inicio del bucle, normalmente para inicializar un contador. Por ejemplo ***int i = 1;***

La cláusula ***condición*** es una expresión lógica que se evalúa al inicio de cada iteración del bucle. En el momento en que dicha expresión se evalúe a false se dejará de ejecutar el bucle y el control del programa pasará a la siguiente instrucción (a continuación del bucle *for*). Se utiliza para indicar la condición en la que quieres que el bucle continúe. Por ejemplo ***i <= 10;***

La cláusula ***incremento*** es una instrucción que se ejecuta al final de cada iteración del bucle (después del bloque de instrucciones). Generalmente se utiliza para incrementar o decrementar el contador. Por ejemplo ***i++;*** (incrementar i en 1).

Ejemplo 1: Bucle que muestra por pantalla los números naturales del 1 al 10:

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

En la inicialización utilizamos **int i=1** para crear la variable i con un valor inicial de 1.

La condición **i<=10** indica que el bucle debe repetirse mientras i sea menor o igual a 10.

La actualización **i++** indica que, al final de cada iteración, i debe incrementarse en 1.

Ejemplo 2: Programa que muestra los números naturales (1,2,3,4,5,6,...) hasta un número introducido por teclado.

```
6 public static void main(String[] args) {  
7     Scanner sc = new Scanner(System.in);  
8     int max;  
9     System.out.print("Introduce el número máximo: ");  
10    max = sc.nextInt();  
11    for (int i = 1; i <= max; i++) {  
12        System.out.println("Número: " + i);  
13    }  
14 }  
15 }  
16 }
```

Siendo la salida:

```
run:  
Introduce el número máximo: 5  
Número: 1  
Número: 2  
Número: 3  
Número: 4  
Número: 5  
BUILD SUCCESSFUL (total time: 6 seconds)
```

3. BUCLE WHILE

El bucle *while* se codifica de la siguiente forma:

Código	Ordinograma
<pre>while (condición) { bloque acciones; }</pre>	<pre>graph TD; Entry(()) --> Condicion{Condicion}; Condicion -- Verdadero --> Accion1[Accion 1]; Accion1 --> Accion2[Accion 2]; Accion2 --> Accion3[Accion 3]; Accion3 --> AccionN[Accion N]; AccionN --> Entry; Condicion -- Falso --> Exit(());</pre>

El bloque de instrucciones se ejecuta mientras se cumple una condición (mientras **condición** se evalúe a true). **La condición se comprueba ANTES de empezar** a ejecutar por primera vez el bucle, por lo que si se evalúa a false en la primera iteración, entonces el bloque de acciones no se ejecutará ninguna vez.

El mismo **ejemplo 2** anterior hecho con un bucle **while** sería:

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    int max, cont;  
    System.out.print("Introduce el número máximo: ");  
    max = sc.nextInt();  
    cont = 1;  
    while (cont <= max) {  
        System.out.println("Número: " + cont);  
        cont++;  
    }  
}
```

Y la salida:

```
run:  
Introduce el número máximo: 5  
Número: 1  
Número: 2  
Número: 3  
Número: 4  
Número: 5  
BUILD SUCCESSFUL (total time: 6 seconds)
```

4. BUCLE DO-WHILE

El bucle *while* se codifica de la siguiente forma:

Código	Ordinograma
<pre>do { bloque acciones; } while (condición);</pre>	<pre>graph TD Entry(()) --> A1[Accion 1] A1 --> A2[Accion 2] A2 --> A3[Accion 3] A3 --> AN[Accion N] AN --> Cond{Condicion} Cond -- Verdadero --> Entry Cond -- Falso --> Exit(())</pre>

En este tipo de bucle, **el bloque de instrucciones se ejecuta siempre al menos una vez**, y ese bloque de instrucciones se ejecutará mientras **condición** se evalúe a *true*.

⚡ Por ello en el bloque de instrucciones deberá existir alguna que, en algún momento, haga que *condición* se evalúe a *false*. ¡Si no el bucle no acabaría nunca!

El mismo **ejemplo 2** anterior hecho con un bucle **do-while** sería:

```
1 public static void main(String[] args) {  
2     Scanner sc = new Scanner(System.in);  
3     int max, cont;  
4     System.out.print("Introduce el número máximo: ");  
5     max = sc.nextInt();  
6     cont = 1;  
7  
8     do {  
9         System.out.println("Número: " + cont);  
10        cont++;  
11    } while (cont <= max);  
12  
13 }  
14  
15 }  
16  
17 }  
18  
19 }  
20 }
```

Y la salida:

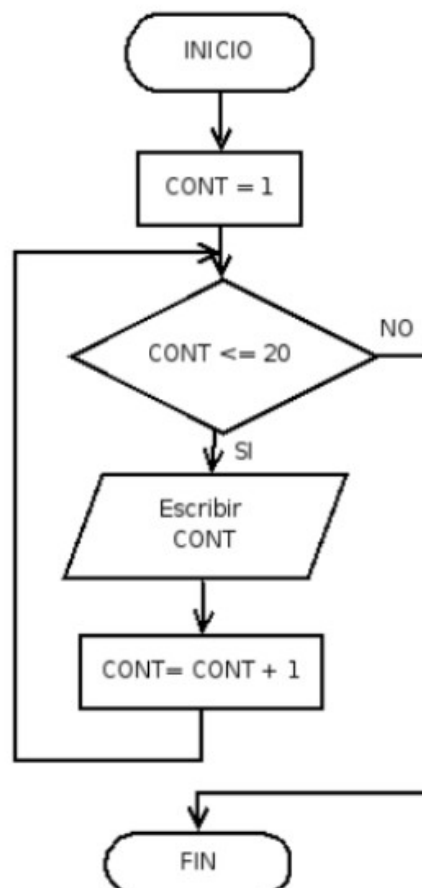
```
run:  
Introduce el número máximo: 5  
Número: 1  
Número: 2  
Número: 3  
Número: 4  
Número: 5  
BUILD SUCCESSFUL (total time: 6 seconds)
```

5. EJEMPLOS

5.1 Ejemplo 1

Programa que muestre por pantalla los 20 primeros números naturales (1, 2, 3... 20).

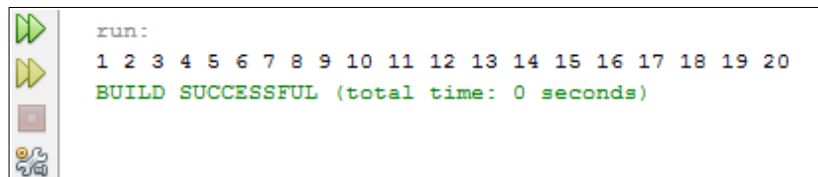
Ordinograma:



Código:

```
12     public class Ejercicio1 {  
13  
14         public static void main(String[] args) {  
15             int cont;  
16  
17             for(cont=1;cont<=20;cont++)  
18                 System.out.print(cont + " ");  
19  
20             System.out.print("\n");  
21         }  
22     }
```

Salida:

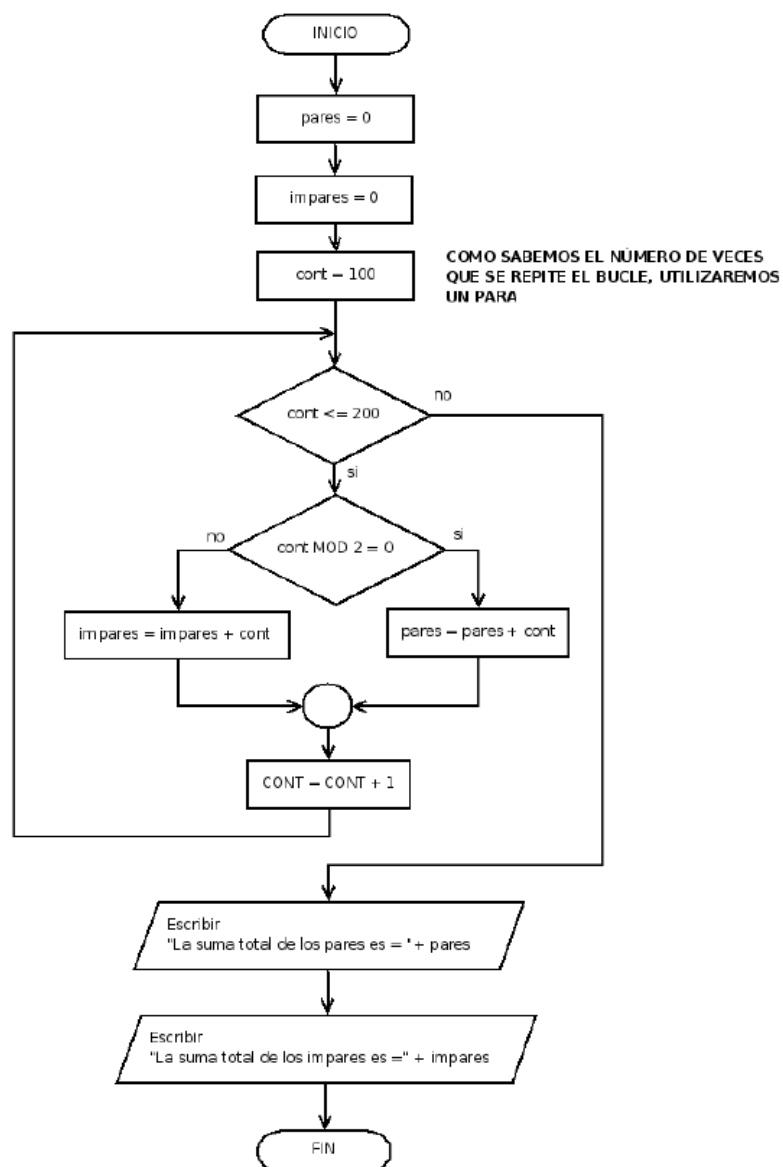


```
run:  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20  
BUILD SUCCESSFUL (total time: 0 seconds)
```

5.2 Ejemplo 2

Programa que suma independientemente los pares y los impares de los números comprendidos entre 100 y 200.

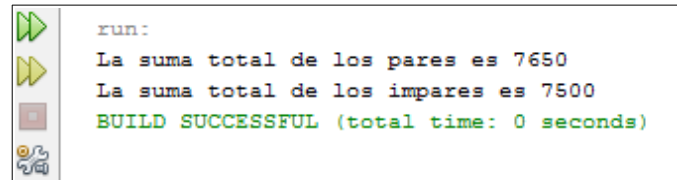
Ordinograma:



Código:

```
12 public class Ejercicio11 {  
13  
14     public static void main(String[] args) {  
15         int pares, impares, cont;  
16  
17         pares = 0;  
18         impares = 0;  
19  
20         for(cont=100; cont <= 200; cont++)  
21         {  
22             if(cont % 2 == 0)  
23                 pares = pares + cont;  
24             else  
25                 impares = impares + cont;  
26         }  
27  
28         System.out.println("La suma total de los pares es " + pares);  
29         System.out.println("La suma total de los impares es " + impares);  
30     }  
31  
32 }
```

Salida:



```
run:  
La suma total de los pares es 7650  
La suma total de los impares es 7500  
BUILD SUCCESSFUL (total time: 0 seconds)
```

6. AGRADECIMIENTOS

Apuntes actualizados y adaptados al CEEDCV a partir de la siguiente documentación:

- [1] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.

Licencia



Reconocimiento - NoComercial - CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original. NOTA: Esta es una obra derivada de la obra original realizada por Carlos Cacho, Raquel Torres, Lionel Tarazon, Fco. Javier Valero.