



UD2

LENGUAJE DE PROGRAMACIÓN JAVA

MP_0485
Programación

2.2 Instrucciones
de control

Introducción

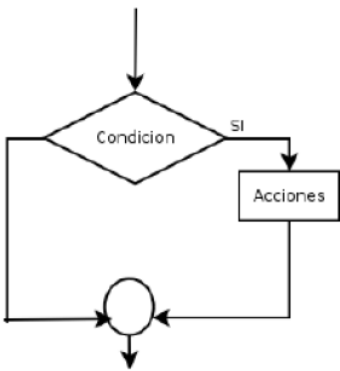
En este documento se verán las instrucciones de control más importantes del lenguaje de programación Java. Las estructuras de control son construcciones que permiten alterar el flujo secuencial de un programa de forma que, en función de una condición o el valor de una expresión, el mismo pueda ser desviado en una u otra alternativa de código o repetido un número de veces.

Las estructuras de control disponibles en Java son:

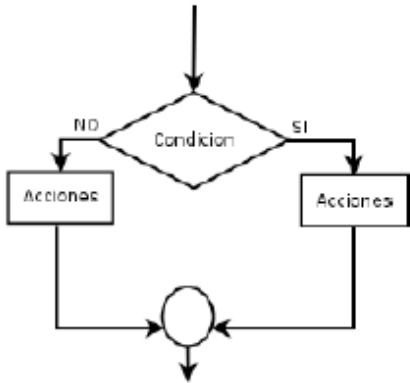
- ⊕ Condición Simple (if)
- ⊕ Condición Doble (if-else)
- ⊕ Condición Multiple (switch)
- ⊕ Bucle while
- ⊕ Bucle do – while
- ⊕ Bucle for

Estructuras condicionales

Condicional simple

Código	Ordinograma
<pre>if (condición) { // Acciones; }</pre> <p>El bloque de Acciones se ejecuta si la condición se evalúa a true (es verdadera).</p> <pre>if (cont == 0) { System.out.println("cont es 0"); // más instrucciones... }</pre> <p>Si dentro del if solo hay una instrucción, no es necesario poner las llaves.</p> <pre>if (cont == 0) System.out.println("cont es 0");</pre>	

Condicional doble

Código	Ordinograma
<pre> if (condición) { // AccionesSI; } else { // AccionesNO; } </pre> <p>El bloque AccionesSI se ejecuta si la condición se evalúa a true (verdadera). En caso contrario, se ejecuta el bloque de AccionesNO.</p> <pre> if (cont == 0) { System.out.println("cont es 0"); // más instrucciones... } else { System.out.println("cont no es 0"); // más instrucciones... } </pre> <p>Si dentro del if o el else solo hay una instrucción, no es necesario poner las llaves.</p> <pre> if (cont == 0) System.out.println("cont es 0"); else System.out.println("cont no es 0"); </pre>	 <pre> graph TD Start(()) --> Condicion{Condicion} Condicion -- NO --> Acciones1[Acciones] Condicion -- SI --> Acciones2[Acciones] Acciones1 --> Join(()) Acciones2 --> Join Join --> Exit(()) </pre>

Recordad que el operador relaciona para comprobar si son iguales es ==, no un solo =, este corresponde con el operador de asignación. Este error no lo detecta el compilador y es difícil de averiguar.

En muchas ocasiones, se anidan estructuras alternativas *if-else*, de forma que se pregunte por una condición si anteriormente no se ha cumplido otra sucesivamente. Por ejemplo: supongamos que realizamos un programa que muestra la nota de un alumno en la forma

(insuficiente, suficiente, bien, notable o sobresaliente) en función de su nota numérica. Podría codificarse de la siguiente forma:

```

1
2 import java.util.Scanner;
3
4 public class Nota {
5
6     public static void main(String[] args) {
7         Scanner entrada = new Scanner(System.in);
8         int nota;
9         //Suponemos que el usuario introduce el número correctamente.
10        //No hacemos comprobación
11        System.out.println("Dame un número entre 0 y 10");
12
13        nota = entrada.nextInt();
14
15        if (nota < 5) {
16            System.out.println("Insuficiente");
17        } else if (nota < 6) {
18            System.out.println("Suficiente");
19        } else if (nota < 7) {
20            System.out.println("Bien");
21        } else if (nota < 9) {
22            System.out.println("Notable");
23        } else {
24            System.out.println("Sobresaliente");
25        }
26    }
27 }
28
29
30
31
32
33
34
35
36
37
38

```

Siendo la salida:

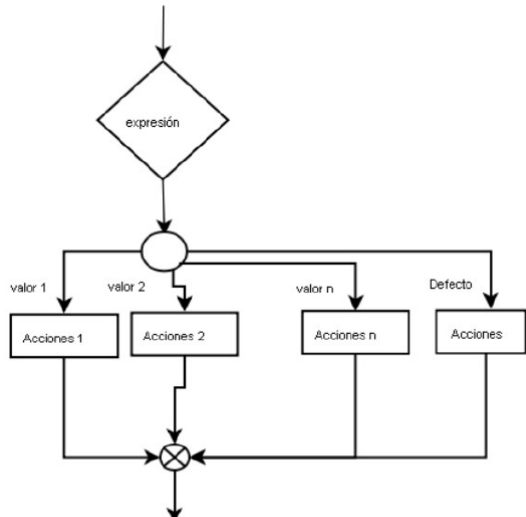
```

run:
Dame un número entre 0 y 10
8
Notable
BUILD SUCCESSFUL (total time: 11 seconds)

```

Es muy recomendable usar la tecla tabulador en las instrucciones de cada bloque. Como se puede ver en el ejemplo, cada **else** esta alineado con su **if** asociado, de esta forma es más fácil leer el código.

Condicional múltiple (Switch)

Código	Ordinograma
<pre> switch (expresión) { case valor1: // Acciones1; break; case valor2: // Acciones2; break; case valorN: // AccionesN; break; default: // Acciones por defecto; } </pre>	 <pre> graph TD Start(()) --> Exp{expresión} Exp --> Circle(()) Circle -- "valor 1" --> Acc1[Acciones 1] Circle -- "valor 2" --> Acc2[Acciones 2] Circle -- "valor n" --> Accn[Acciones n] Circle -- "Defecto" --> AccDef[Acciones] Acc1 --> Join((X)) Acc2 --> Join Accn --> Join AccDef --> Join Join --> End(()) </pre>

Es muy importante entender que en el switch se evalúa una expresión (un valor concreto como 0, 5, 1...) **no una condición** (verdadera o falsa) como en el *if* y el *ifelse*.

El programa comprueba el valor de la expresión y saltara al 'case' que corresponda con dicho valor (valor1 o valor2 o ...) ejecutando el código de dicho 'case' (Acciones1 o Acciones2 o ...). Si no coincide ningún valor, saltara al 'default' y ejecutara las acciones por defecto. Es importante añadir la sentencia **break;** al final de cada 'case', ya que de lo contrario el programa seguirá ejecutando el código de las demás acciones y normalmente no queremos que haga eso (aunque Java permite hacerlo, es confuso y por ello esta desaconsejado).

Un ejemplo sería el siguiente:

```
1
2 import java.util.Scanner;
3
4 public class Alternativa_Multiple {
5
6     public static void main(String[] args) {
7         Scanner entrada = new Scanner(System.in);
8         int dia;
9
10        System.out.println("Dame un número entre 1 y 7:");
11
12        dia = entrada.nextInt();
13
14        switch (dia) {
15            case 1:
16                System.out.println("Lunes");
17                break;
18            case 2:
19                System.out.println("Martes");
20                break;
21            case 3:
22                System.out.println("Miércoles");
23                break;
24            case 4:
25                System.out.println("Jueves");
26                break;
27            case 5:
28                System.out.println("Viernes");
29                break;
30            case 6:
31                System.out.println("Sábado");
32                break;
33            case 7:
34                System.out.println("Domingo");
35                break;
36            default:
37                System.out.println("Error el número debe estar entre 0 y 7");
38        }
39    }
40 }
41
42 }
```

Y la salida:

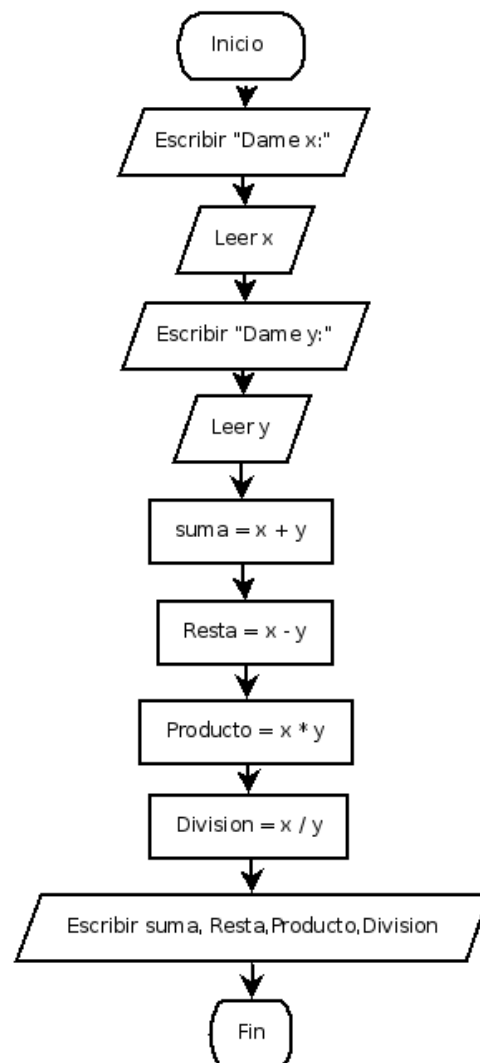
```
run:
Dame un número entre 1 y 7:
4
Jueves
BUILD SUCCESSFUL (total time: 2 seconds)
```

Ejemplos de estructuras condicionales

Ejemplo 1

Programa que lea dos números, calcule y muestre el valor de sus suma, resta, producto y división.

Ordinograma:



```

1  package ejemplo1;
2
3  import java.util.Scanner; // Importamos la clase Scanner
4
5  public class Ejemplo1 {
6
7      public static void main(String[] args) {
8
9          // Declaramos las variables que vamos a necesitar
10         int x, y, suma, resta, mult, div;
11
12         // Creamos el objeto Scanner para leer por teclado
13         Scanner reader = new Scanner(System.in);
14
15         // Pedimos y leemos x
16         System.out.print("Dame x:");
17         x = reader.nextInt();
18
19         // Pedimos y leemos y
20         System.out.print("Dame y:");
21         y = reader.nextInt();
22
23         // Realizamos los cálculos necesarios
24         suma = x + y;
25         resta = x - y;
26         mult = x * y;
27         div = x / y;
28
29         // Mostramos los cálculos por pantalla
30         System.out.println("Suma: " + suma);
31         System.out.println("Resta: " + resta);
32         System.out.println("Multiplicación: " + mult);
33         System.out.println("División: " + div);
34     }
35 }

```

Y su salida:



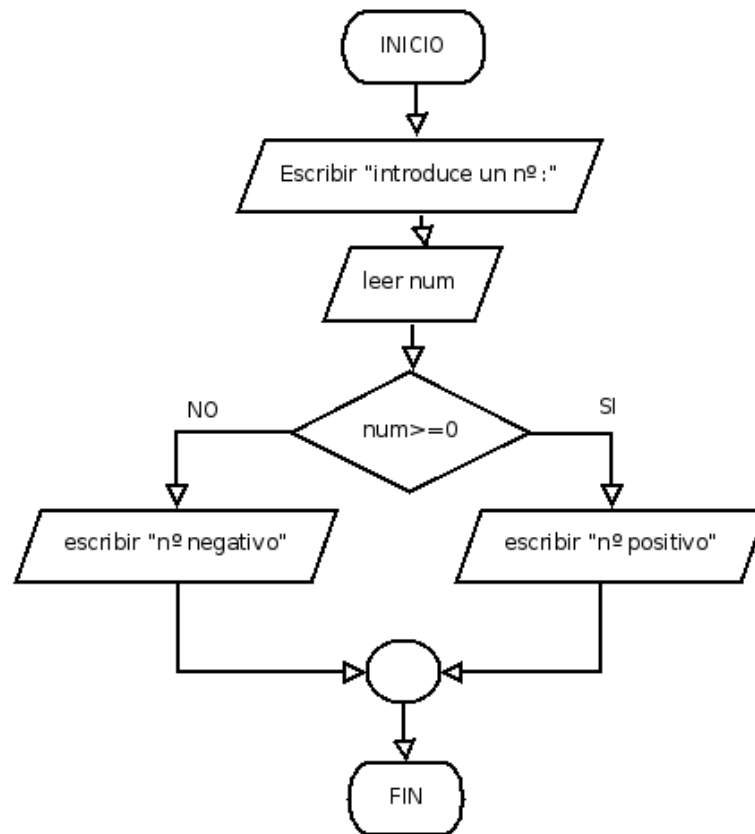
```

run:
Dame x:4
Dame y:2
Suma: 6
Resta: 2
Multiplicación: 8
División: 2

```


Ejemplo 2

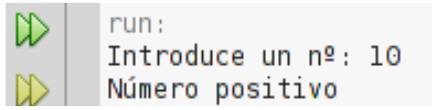
Programa que lee un número y me dice si es positivo o negativo. Consideraremos el cero como positivo.



```

1  package ejemplo2;
2
3  import java.util.Scanner; // Importamos la clase Scanner
4
5  public class Ejemplo2 {
6
7      public static void main(String[] args) {
8
9          // Declaramos la variable num
10         int num;
11
12         // Creamos el objeto Scanner para leer por teclado
13         Scanner reader = new Scanner(System.in);
14
15         // Pedimos y leemos x
16         System.out.print("Introduce un nº: ");
17         num = reader.nextInt();
18
19         // Astructura alternativa doble
20         if (num >= 0)
21             System.out.println("Número positivo");
22         else
23             System.out.println("Número negativo");
24     }
25 }
  
```

Y su salida:



```
run:  
Introduce un nº: 10  
Número positivo
```

Estructuras de control de repetición (bucles)

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.

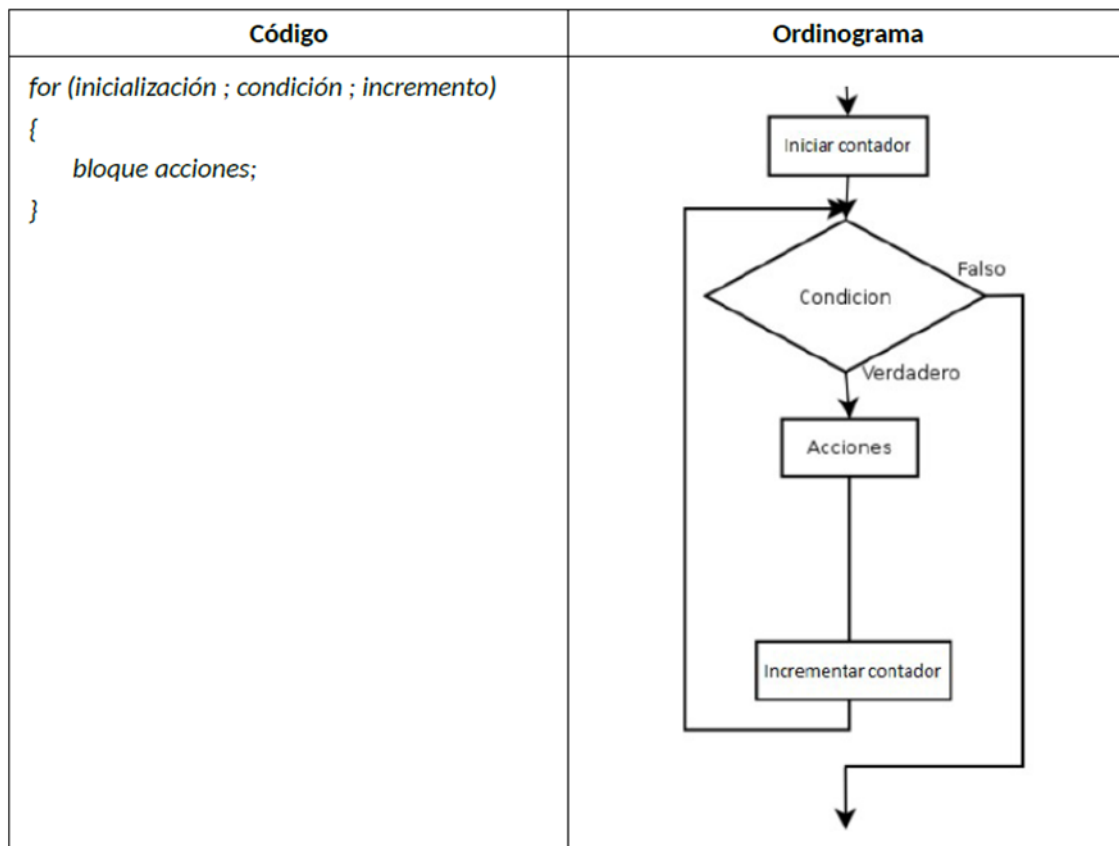
Utilizar el bucle **while** cuando no sabemos el número de veces que ha de repetirse el bloque y es posible que no deba ejecutarse ninguna.

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.

Bucle for

El bucle for se codifica de la siguiente forma:



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 0. 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.

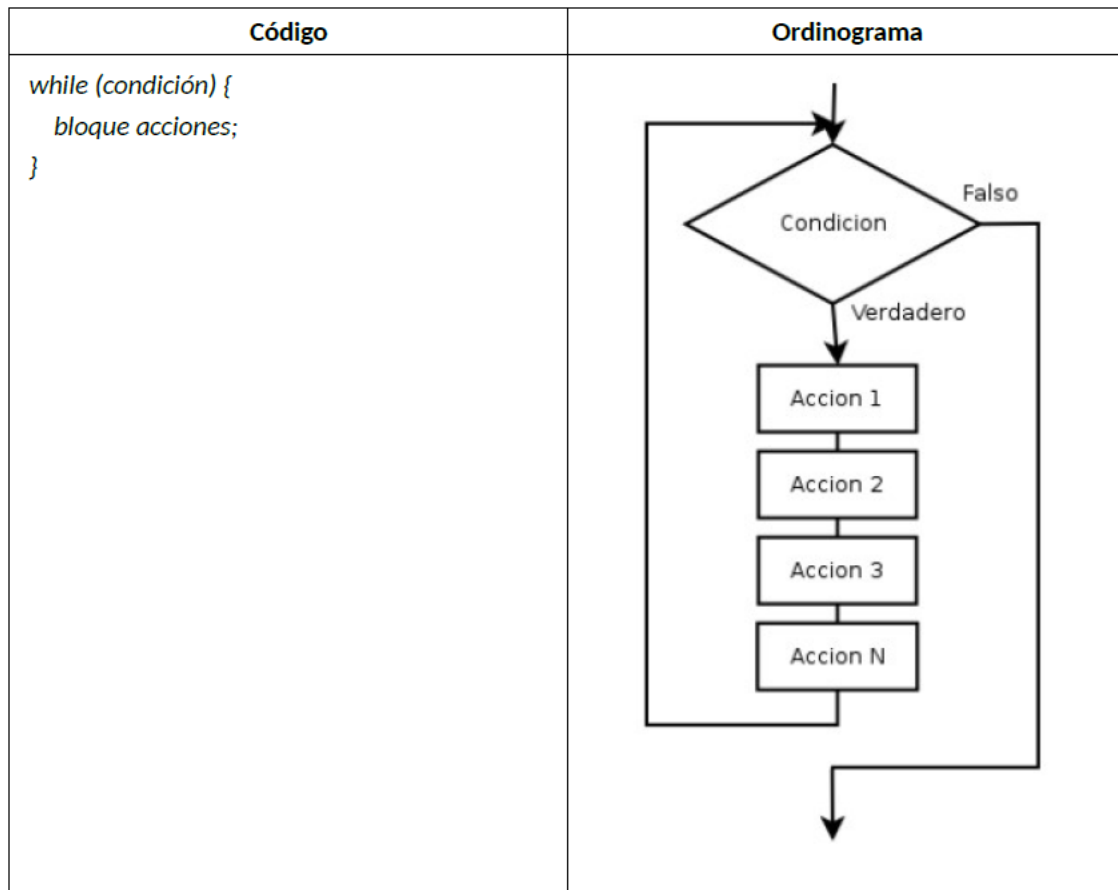
```
6 public static void main(String[] args) {  
7     Scanner sc = new Scanner(System.in);  
8     int max;  
9     System.out.print("Introduce el número máximo: ");  
10    max = sc.nextInt();  
11    for (int i = 1; i <= max; i++) {  
12        System.out.println("Número: " + i);  
13    }  
14 }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

Siendo su salida:

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

Bucle while

El bucle while se codifica de la siguiente forma:



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:

```

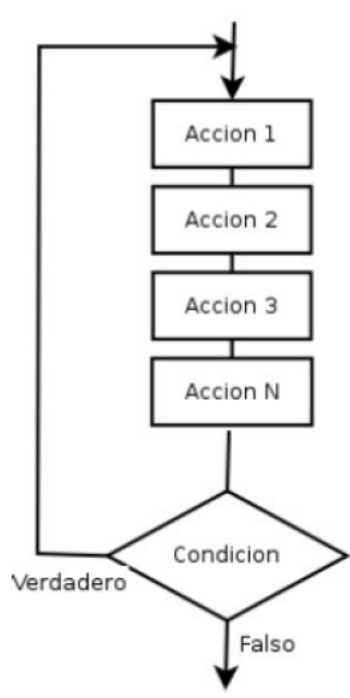
7 public static void main(String[] args) {
8     Scanner sc = new Scanner(System.in);
9     int max, cont;
10    System.out.print("Introduce el número máximo: ");
11    max = sc.nextInt();
12    cont = 1;
13    while (cont <= max) {
14        System.out.println("Número: " + cont);
15        cont++;
16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

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)
```

Bucle do-while

El bucle do-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 --> C{Condicion} C -- Verdadero --> Entry C -- 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 instrucción que, en algún momento haga que la *condición* se evalúe a *false*. ¡Si no el bucle no acabaría nunca!

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

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

Y la salida sería:

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

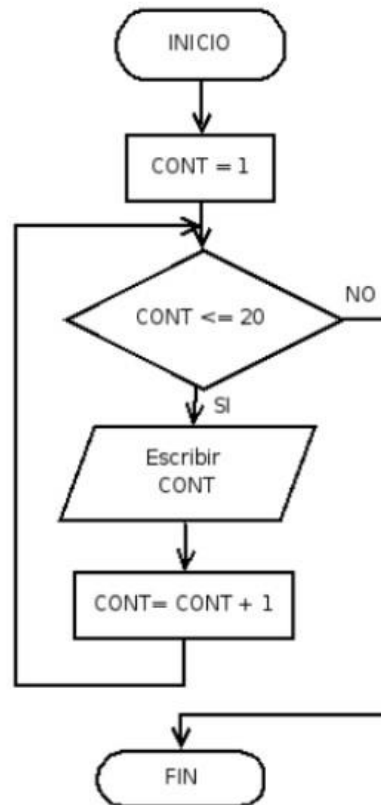
Ejemplos

Ejemplo 1

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

boolean activado = false;

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:



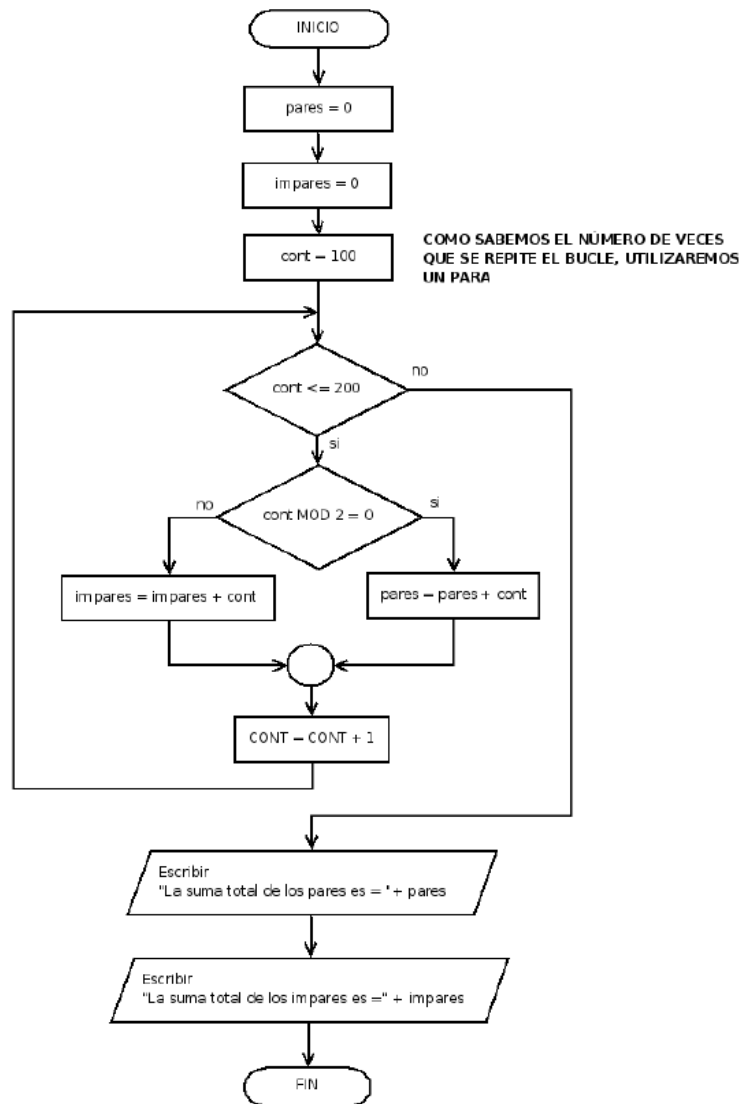
```

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)
  
```


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)
```

Referencias

Apuntes elaborados a partir de la siguiente documentación:

- [1] Apuntes Fernando Barber y Ricardo Ferris. Universidad de Valencia.
- [2] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.
- [3] Apuntes de Programación de Carlos Cacho y Raquel Torres. Ceedcv.
- [4] Apuntes de Programación Edix Digital Workers.

Licencia



[CC BY-NC-SA 3.0 ES](https://creativecommons.org/licenses/by-nc-sa/3.0/es/) Reconocimiento - No Comercial - Compartir Igual (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 original realizada por Carlos Cacho y Raquel Torres.