



Sentencias de control
del flujo del programa

Índice



Sentencias de control del flujo del programa

1 La sentencia if	3
2 If anidados	6
3 El operador ternario "? :"	7
4 La sentencia switch	9
5 El bucle for	11
6 El bucle while	14
7 El bucle do while	16
8 Sentencias de salto break y continue	18

1. La sentencia if

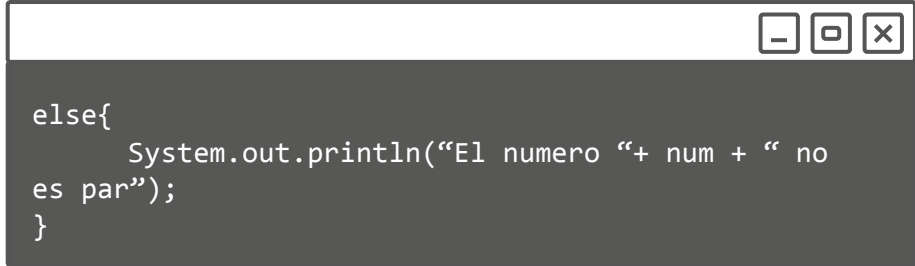
La sentencia **if** o de decisión simple evalúa una expresión condicional, si su resultado es **true** se ejecuta un bloque de sentencias, si su resultado es **false** se ejecuta otro resultado. Las formato de esta sentencia es la siguiente:

```
if (<expresion_condicional>){  
    <sentencias_true>  
}  
else{  
    <sentencias_false>  
}
```

La clausula else es opcional, por tanto la formato sin esta es la siguiente:

```
if (<expresion_condicional>){  
    <sentencias_true>  
}
```

En este caso, si la expresión condicional no se ejecuta ninguna sentencia, el programa sigue ejecutándose en la sentencia siguiente al bloque del if.



```
else{  
    System.out.println("El numero "+ num + " no  
es par");  
}
```

Ejemplo que partiendo de un número aleatorio, igual o mayor que 2, examina si es par ($\text{num} \% 2 == 0$):

```
Random aleatorio = new Random();
int num = aleatorio.nextInt()+2;
if( num % 2 == 0){
    System.out.println("El numero "+ num + " es
par");
}
```

Ejemplo que partiendo de un número aleatorio, igual o mayor que 0 y menor que 5, y otro mayor que 0 y menor que 2500, comprueba que le primero es cero, si lo es lo incrementa en una unidad y continuación lo divide por el segundo:

```
Random aleatorio = new Random();
int num = 0,
    num1 = aleatorio.nextInt(5), num2 =
aleatorio.nextInt(2500);
if (num1 == 0){
    num++;
}
System.out.println(num2 / num1);
```



En cualquiera de los bloques de sentencias, si estos están formados por una sólo no es necesario encerrarla entre `{ }`. Los ejemplos anteriores quedarían de la forma siguiente:

```
Random aleatorio = new Random();
    int num = aleatorio.nextInt()+2;
    if( num % 2 == 0)System.out.println("El numero "+
                                     num + " es par");

    else System.out.println("El numero "+
                           num + " no es par");

    int num1 = aleatorio.nextInt(5),
        num2 = aleatorio.nextInt(2500);
    if (num1 == 0)num++;
    System.out.println(num2 / num1);
```



2. If anidados

Cualquiera de los dos bloques de sentencias pueden contener a su vez otras sentencias **if**. La precaución que hay que tener en estos casos es que una instrucción **else** siempre hace referencia a la instrucción **if** más cercana dentro de un mismo bloque y que no esté asociada con ninguna otra instrucción **else**. Por ejemplo:

```
if( i <= 100){
    if( j > 10) incto = - 0.5F;
    if( k < 5) incto = 0.5F;
    else incto = .0F; //este else hace referencia a if( k < 5 )
}else incto = 1.5F; //este else hace referencia a if( i <= 100)
```

Una construcción muy típica con if anidados es la llamada if-else-if, en la parte else se sitúa un if; si no se cumple una condición se evalúa si se cumple otra condición que en cierta medida estará relacionada con la anterior. Su construcción sigue este formato:

```
if (<expresion_condicional1>){
    <sentencias_true1>
}
else if (<expresion_condicional2>){<sentencias_true2>
}
else if (<expresion_condicional3>){<sentencias_true3>
}
...
else if (<expresion_condicionalN>){<sentencias_trueN>
}
```

Por ejemplo:

```
if( i < 0 ) marcha = "frenar";
else if ( i == 0 ) marcha = "continuar";
else marcha = "acelerar";
```

```
if( i < 0 ) marcha = "frenar";
else if ( i == 0 ) marcha = "continuar";
else marcha = "acelerar";

if (edad > 55 ) categoria = "veterano C";
else if (edad > 45 ) categoria = "veterano B";
else if (edad > 35 ) categoria = "veterano A";
else if (edad > 19 ) categoria = "senior";
else if (edad > 16 ) categoria = "junior";
else categoria = "no puede participar";
```

3. El operador ternario "? :"

El operador ternario, no es una sentencia de control de flujo del programa, como cualquier otro operador se relaciona con sus operandos y produce un resultado, que puede ser almacenado en una variable, ser el valor de un parámetro de una función o ser el valor que retorna una función.

Su formato es la siguiente:

```
<expresion_condicional> ?
<expresion_verdadera> :
<expresion_falsa>;
```

Su funcionamiento es el siguiente:

- Se evalúa <expresion_condicional>, cuyo resultado será true o false.
- Si el resultado es true, se evalúa <expresion_verdadera>, cuyo resultado podrá ser un valor de cualquier tipo.
- Si el resultado es false, se evalúa <expresion_falsa>, cuyo resultado podrá ser un valor de cualquier tipo.

Por ejemplo:

```
resultado = divisor == 0 ? 0 : dividendo / divisor;
```

En este ejemplo se impide la división por cero, en "resultado" se almacena un cero si "divisor" es igual que 0, sino se almacena el resultado de dividir "dividendo" entre "divisor".

Otro ejemplo:

```
menor = num1 < num2 ? num1 : num2;
```

En "menor" se almacena el menor de "num1" y "num2"

Otro ejemplo:

```
public static boolean esMayorEdad(int edad){  
    return edad >= 18 ? true : false;  
}
```

La función "esMayorEdad" devuelve true o false, según sea el valor del parámetro "edad".



4. La sentencia switch

La sentencia switch es una sentencia de decisión múltiple, permite elegir entre varias alternativas, es equivalente a la utilización de una secuencia de sentencias if anidadas if-else-if.

Su formato es el siguiente:

```
switch (<expresion_switch>){  
    case <constante_1>:  
        <sentencias_1>  
        break;  
    case <constante_2>:  
        <sentencias_2>  
        break;  
    . . .  
    case <constante_N>:  
        <sentencias_3>  
        break;  
    [default:  
        <sentencias_default>]  
}
```

Las palabras reservadas que se utilizan para la formación de la sentencia **switch** son, además de ella misma, **case**, **break** y **default**.

La expresión a la derecha de **switch** (<expresion_switch>) podrá ser de tipo char, o cualquier tipo entero o una cadena (String).

La clausula **default** es opcional.

El funcionamiento de switch es el siguiente:

- Se evalúa **<expresion_switch>**, produce un valor de los indicados anteriormente.
- Se ejecutan las sentencias que se encuentren a continuación de la etiqueta **case** cuya **constante** sea exactamente igual al resultado de <expresion_switch>
- Las sentencias que se ejecutan son todas las que se encuentren a la derecha del carácter ":" de la etiqueta **case** correspondiente, según lo anterior, hasta la sentencia **break**.
- Si se omite un **break**, se ejecutan las sentencias hasta el próximo **break** o si no hay ninguno hasta la llave de cierre del **switch**.

- Si no se encuentra ninguna constante en ninguna clausula case que coincida con el resultado de **<expresion_switch>**, se ejecutan las sentencias de la clausula **default**.
- Si no hay clausula **default**, se ejecuta la sentencia siguiente a la llave "}" de cierre del bloque **switch**, por tanto no se ejecuta ningún grupo de sentencias dentro del bloque switch.

Ejemplo, que según valor de la cadena almacenada en "opcion", ejecuta una función u otra, si no coincide con ninguna constante case, en variable "opcionvalida" se almacena "false":

```
System.out.println("Teclea una opcion");
String opcion = teclado.next(); boolean opcionvalida=
true;
switch(opcion){
    case "A":          abrirArchivo();
                        break;
    case "B":          borrarArchivo();
                        break;
    case "C":          crearArchivo();
                        break;
    case "L":          listarArchivo();
                        break;
    case "X":          terminar();
                        break;
    default: opcionvalida=false;
}
```

Como se puede comprobar la sentencia switch se puede implementar con sentencias if-else-if, pero se utiliza esta última construcción cuando las condiciones que controlan el proceso de selección no dependen de un mismo valor.

La sentencia **switch** por tanto se utiliza cuando cualquiera de sus alternativas depende del valor de una única condición, igualdad del valor de la expresión con una de las constantes, para cualquier otro caso se deberá utilizar **if-else-if**, como por ejemplo en el caso siguiente:

```
if(edad < 18) //sentencias
else if (peso > maxpeso) //sentencias
else if (altura > maxaltura) //sentencias
```

En el ejemplo cada condición depende de una variable distinta, por tanto no se puede transformar este grupo de sentencias **if** en una sentencia **switch**.

Dentro de cualquier grupo de sentencias, de cualquier etiqueta case, se pueden anidar otras sentencias **switch**, **if** o como las que se estudian a continuación.

El grupo de sentencias de cada etiqueta case o de default, o tienen que ir encerradas entre " { }", formando un bloque, el grupo esta delimitado por el carácter ":" y la sentencia break o el final del bloque **switch**.

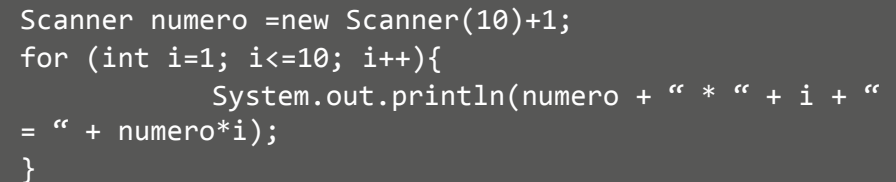
5. El bucle for

La sentencia **for** en su formato más general se utiliza para que se repita la ejecución de un bloque de sentencias un número concreto y determinado de veces, cuyo número vendrá determinado por una variable, normalmente llamada contador que desde un valor inicial, se incrementa en cada iteración en un valor concreto.

El formato de **for** para esta funcionalidad es:

```
for( <expresion_inicializacion>; <expresion_
condicional>; <expresion_incremento>{
    <sentencias>
}
```

Ejemplo:

A screenshot of a Java IDE window with a dark background. The code inside is as follows:

```
Scanner numero =new Scanner(10)+1;
for (int i=1; i<=10; i++){
    System.out.println(numero + " * " + i + "
= " + numero*i);
}
```

The window has standard OS controls (minimize, maximize, close) in the top right corner.

Este ejemplo ejecuta 10 veces el bloque de sentencias. La variable contadora *i*, se crea e inicializa con valor 1, se evalúa, si es menor o igual a 10 se ejecuta la sentencia del bloque y a continuación se incrementa en una unidad. Después de la decima iteración, *i* tiene un valor de 11, con lo cual la expresión condicional es false, y ya no se ejecuta la sentencia del bloque.

Para entender el alcance de las diferentes posibilidades que tiene la utilización del bucle for, se pueden dividir las diferentes acciones que realiza en 4 grupos, siendo el orden en las que se ejecutan el siguiente:

- **Se ejecuta <expresion_inicializacion>.** Es opcional, puede no indicarse. En realidad es una lista de expresiones separadas por el carácter “;”. En su funcionamiento “clásico” es una expresión que inicializa la variable contadora a un valor.
- **Se evalúa <expresion_condicional>.** esta expresión siempre dará un valor booleano. si el resultado es false acaba el bucle for. Es también opcional, por lo que si no se indica se tomaría como si su resultado fuera true, por tanto se estaría en un bucle infinito. Para la formación de esta expresión condicional no es obligatorio utilizar la variable contadora.
- **Si la <expresion_condicional> es true se ejecutan las sentencias del bloque,** se ejecutarán estas sentencias, siempre que esta expresión sea true.
- **Se ejecuta la expresión <expresion_incremento>.** Es opcional. Es una lista de expresiones separadas por el carácter “;”. En su funcionamiento “clásico” es una expresión que incrementa o decrementa el valor de la variable contadora.

En la imagen se muestra el orden de ejecución de estas acciones del for según un organigrama.

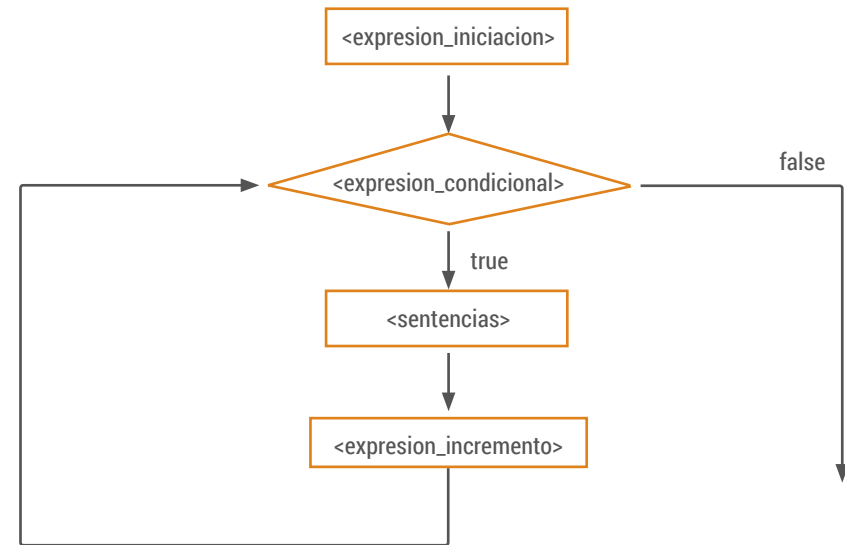


FIGURA 3.1: ORGANIGRAMA BUCLE FOR



Tal y como se puede deducir por las diferentes variantes que puede tomar, for es la sentencia más versátil de Java. Algunas de ellas son:

- **Varias variables contadoras o de control.**

```
int i=0, j=0;
for (i=10, j=1; i>j; i--, j++) //dos variables contadoras
    System.out.println("i: " + i + " j: " + j); //
una sentencia
//i y j siguen existiendo fuera del bucle for
System.out.println("i: " + i + " j: " + j); //después del
for
```

- **Elementos vacíos.**

```
int k=0;
for( ;k<10; ){
    k++;
    System.out.println("Iteración numero:
    "+ k);
}
```

- **Bucle infinito.**

```
for( ; ; ){
    //Sentencias que se están ejecutando
    siempre
}
```

- **La expresión condicional puede ser cualquier expresión booleana**, no tiene porque depender de la variable o variables de la expresión de inicialización.

```
System.out.println("Teclea x para parar");
//System.in.read lee un carácter y lo retorna
como int
//un int son 4 char, se leen los 4 del buffer
for(i=1; ((char)System.in.read()) != 'x'; i++)
    System.out.println("Iteración número: "
    + i);
```

- **Bucle sin cuerpo.**

```
numero= aleatorio.nextInt(200)+1;
long suma=0;
for(i=1; i<numero; suma += i++);
System.out.println("i : " + i + " suma: " +
suma);
```

6. El bucle while

El bucle **while** ejecuta un bloque de sentencias si se cumple una condición determinada, cuando la condición deja de cumplirse ya no se ejecutan dichas sentencias y termina el bucle.

Su formato es:

```
while (<expresion_condicional>){  
    <sentencias>  
}
```

En **<sentencias>** tendrá que ejecutarse alguna instrucción que haga que la **<expresion_condicional>** en alguna iteración sea false, porque si no sería un bucle infinito.

El grupo **<sentencias>** puede que no se ejecute ninguna vez si la primera vez que se evalúa **<expresion_condicional>** es false.

La ejecución del bucle **while** es cómo muestra la imagen:

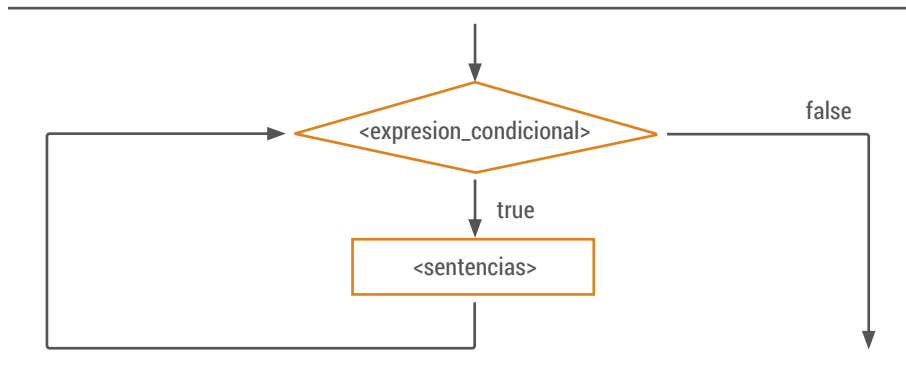
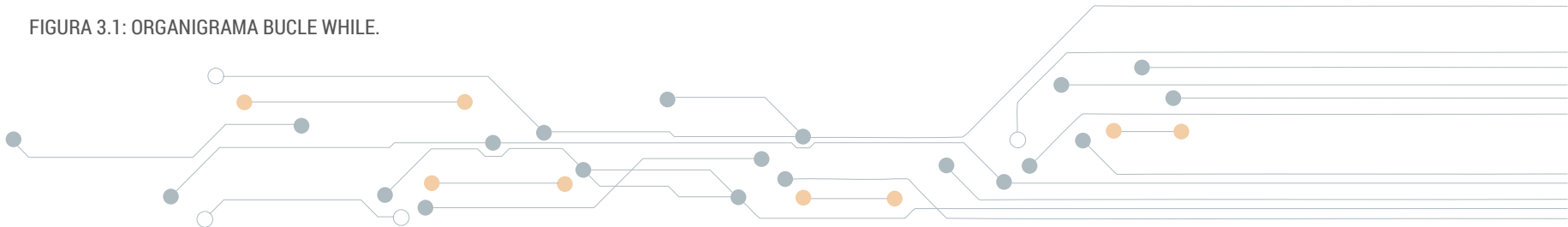


FIGURA 3.1: ORGANIGRAMA BUCLE WHILE.



Ejemplo que implementa un bucle while, en el que se esta iterando mientras que no se teclee un carácter que no sea "s" o "S", en cada iteración se genera un número aleatorio comprendido en el rango de 1 a 49:

```
Random aleatorio= new Random();
Scanner teclado= new Scanner(System.in);
int menor=1, mayor=49, numero=0;
boolean seguir=true;
while(seguir){
    numero= aleatorio.nextInt(mayor-menor+1) + menor;
    System.out.println("Numero: "+ numero);
    System.out.print("Quieres otro número?");
    String opcion= teclado.next();
    seguir = opcion.equals("s") || opcion.equals("S")
        ? true : false;
}
System.out.println("FIN");
```



7. El bucle do while

El bucle **do while** ejecuta un bloque de sentencias mientras se cumpla una condición determinada, cuando la condición deje de cumplirse ya no se ejecutan dichas sentencias y termina el bucle.

Su formato es:

```
do {  
    <sentencias>  
} while (<expresion_condicional>);
```

La ejecución del bucle **do while** es cómo muestra la imagen:

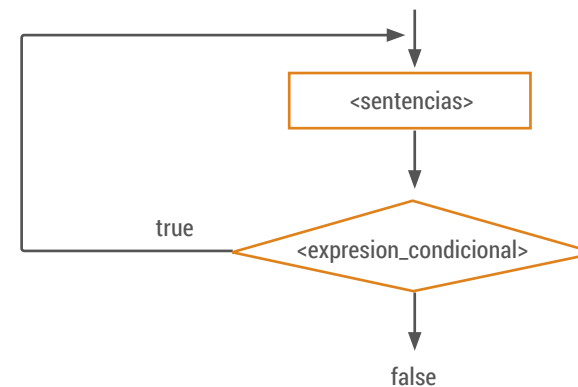


FIGURA 3.1: ORGANIGRAMA BUCLE DO WHILE.

En **<sentencias>** tendrá que ejecutarse alguna instrucción que haga que la **<expresion_condicional>** en alguna iteración sea false, porque si no sería un bucle infinito.

El grupo **<sentencias>** se ejecuta al menos una vez.

Ejemplo que implementa un bucle do-while, en el que se esta iterando mientras que no se teclee un carácter que no sea "s" o "S", en cada iteración se genera un número aleatorio comprendido en el rango de 1 a 49:

```
Random aleatorio= new Random();
Scanner teclado= new Scanner(System.in);
int menor=1, mayor=49, numero=0;
boolean seguir=false;
do{
    numero= aleatorio.nextInt(mayor-menor+1) + menor;
    System.out.println("Numero: "+ numero);
    System.out.print("Quieres otro número?");
    String opcion= teclado.next();
    seguir = opcion.equals("s") || opcion.equals("S")
        ? true : false;
}while( seguir );
System.out.println("FIN");
```



8. Sentencias de salto break y continue

En Java se utilizan las sentencias break y continue para implementar saltos ocasionales en el flujo de control del programa. La sentencia break es utilizada en la estructura de switch, para interrumpir la ejecución de sentencias de cada una de las alternativas.

La funcionalidad de break y continue es salir del bucle que se está ejecutando. La diferencia es:

- **break**, interrumpe la ejecución de las sentencias del bucle saliendo de este y ejecutando la sentencia siguiente ya fuera del bucle. Fuerza la salida inmediata de un bucle.
- **continue**, interrumpe la ejecución de las sentencias del bucle y vuelve a que se evalúe de nuevo la <expresion_condicional> que controla las iteraciones del bucle. Fuerza la siguiente iteración del bucle.

Ejemplo sentencia **break**, en el que se esta iterando siempre (forever), a menos que de forma aleatoria se genere un 0 para "num1" o para "num2":

```
for( ; ; ){  
    num1= aleatorio.nextInt(50);  
    if (num1 == 0) break;  
    num2= aleatorio.nextInt(1000);  
    if (num2 == 0) break;  
    System.out.print("num2: "+ num2 + "  
num1: " + num1 );  
    System.out.print(" num2/num1 = "+  
num2 / num1);  
    System.out.println(" num2%num1 = "+  
num2 % num1);  
}
```



Ejemplo sentencia **continue** que en cuanto se genere un número aleatorio con valor 0 para "num1" o para "num2", no se ejecutan las sentencias que vienen a continuación y se vuelve a comprobar la condición "i<=veces":

```
for( int i=1; i<= veces ; i++){
    System.out.print("\niteración número: "+ i + " ==>
");

    num1= aleatorio.nextInt(50);
    if (num1 == 0) continue;
    num2= aleatorio.nextInt(1000);
    if (num2 == 0) continue;
    System.out.print("num2: "+ num2 + " num1: " + num1
);

    System.out.print(" num2/num1 = "+ num2 / num1);
    System.out.println(" num2%num1 = "+ num2 % num1);
}
```



Telefonica

EDUCACIÓN DIGITAL