



**CICLO 2**

[FORMACIÓN POR CICLOS]

# Sentencias de Control **DE FLUJO**




**Ingeni@**  
Soluciones TIC



**UNIVERSIDAD  
DE ANTIOQUIA**

Facultad de Ingeniería



Java es un lenguaje de programación imperativo, y como tal, con él debemos especificar paso a paso (mediante algoritmos) lo que queremos realizar. Es así, como los archivos de código fuente escritos en Java se ejecutan de arriba hacia abajo, sentencia por sentencia, en el orden en el que estas aparecen. Sin embargo, este orden se puede cambiar mediante sentencias de control de flujo. El control de flujo se puede realizar mediante la toma de decisiones, ciclos, entre otras instrucciones.

## Las sentencias `if-then` e `if-then-else`

### La sentencia `if-then`

La sentencia de control de flujo más básica es la sentencia `if-then`. Esta sentencia le dice al programa que se ejecuten una o más instrucciones sólo si una condición dada se cumple, es decir, si dicha condición se evalúa como verdadera. Si la condición no se cumple, el programa sigue su ejecución después del final del bloque `if-then`. Su sintaxis es esta:

```
if (condición) {  
    sentencia1;  
    sentencia2;  
    ...  
}
```

Después de la palabra clave `if` se expresa la condición. Si dicha condición se cumple, se ejecutan las sentencias 1 y 2, y cualquier otra dentro del bloque delimitado mediante llaves (`{ }`). Si la condición no se cumple, el flujo de ejecución pasa a estar debajo de la segunda llave que cierra el bloque. El siguiente ejemplo muestra el uso de `if` en el método `main`. Dada una nota, en la sentencia `if` se comprueba si esta es mayor o igual a 3. De ser así, el programa imprime el mensaje relacionado<sup>1</sup>.

---

<sup>1</sup>Los ejemplos de esta lectura se encuentran acá: [https://github.com/leonjaramillo/udea\\_ruta2\\_ciclo2/blob/main/main/java/co/edu/udea/udea\\_ruta2\\_ciclo2/ClaseControlFlujo.java](https://github.com/leonjaramillo/udea_ruta2_ciclo2/blob/main/main/java/co/edu/udea/udea_ruta2_ciclo2/ClaseControlFlujo.java)



```
public class ClaseControlFlujo {  
    public static void main (String[] args) {  
        double nota = 3.5;  
        if (nota >= 3) {  
            System.out.println("El estudiante aprobó.");  
        }  
    }  
}
```

En este caso, el programa imprimirá:

El estudiante aprobó.

Si el bloque de instrucciones que se ejecuta al cumplirse la condición de la sentencia `if` sólo consta de una línea (como en el caso anterior), las llaves que encierran dicho bloque son opcionales, tal como se ve en el ejemplo alternativo de abajo. Sin embargo, se recomienda usar siempre las llaves (`{ }`).

```
public class ClaseControlFlujo {  
    public static void main (String[] args) {  
        double nota = 3.5;  
        if (nota >= 3)  
            System.out.println("El estudiante aprobó.");  
    }  
}
```

## La sentencia `if-then-else`

La sentencia `if-then-else` funciona de manera similar a la sentencia `if-then`, pero provee una ruta de ejecución alternativa en caso de que la condición no se cumpla, es decir, en caso de que la cláusula se evalúe como falsa. Su sintaxis es esta:

```
if (condición) {  
    sentencia1;  
    sentencia2;  
} else {  
    sentencia3;  
    sentencia4;  
}
```



De acuerdo a lo anterior, si la condición se cumple, se ejecutarán las sentencias previas a la palabra clave `else` (en este caso las sentencias 1 y 2). Si la condición no se cumple, se ejecutan las sentencias dentro del bloque encerrado entre llaves posterior a la palabra clave `else`, es decir, las sentencias 3 y 4. En el siguiente ejemplo se puede observar el uso del camino alternativo mostrado por el `else`. Dada una edad, el programa imprimirá si se trata o no de un adolescente:

```
public class ClaseControlFlujo {
    public static void main (String[] args) {
        int edad = 8;
        if (edad >= 10 && edad <= 20) {
            System.out.println("Es un adolescente.");
        } else {
            System.out.println("No es un adolescente.");
        }
    }
}
```

En el caso anterior, el programa imprimirá lo siguiente:

No es un adolescente.

También es posible encadenar sentencias `if-then-else` de tal modo que se evalúen diferentes condiciones y, por consiguiente, se puedan seguir diferentes rutas. En el siguiente ejemplo, se da un concepto acerca del desempeño de un estudiante en una actividad de acuerdo a la nota que obtuvo en la misma:

```
public class ClaseControlFlujo {
    public static void main (String[] args) {
        double nota = 4.5;
        if (nota >= 0 && nota <= 1) {
            System.out.println("Deficiente");
        } else if (nota > 1 && nota <= 3) {
            System.out.println("Insuficiente");
        } else if (nota > 3 && nota <= 4) {
            System.out.println("Aprobado");
        } else if (nota > 4 && nota <= 5) {
            System.out.println("Excelente");
        } else {
            System.out.println("Nota inválida");
        }
    }
}
```

En el caso anterior, el programa imprimirá:

Excelente

## La sentencia switch

La sentencia `switch`, por su parte, permite que el programa tome diferentes rutas de ejecución, de acuerdo al valor de una variable determinada al momento en el que se llega al inicio de la sentencia. Los tipos de variables aceptados son:

- `byte`, `short`, `char`, `int`
- `String`
- Clases de empaquetado (`Character`, `Integer`, etc., abordadas posteriormente).

Su sintaxis es la siguiente:

```
switch (variable) {  
    case opción1:  
        sentencia1;  
        sentencia2;  
        break;  
    case opción2:  
        sentencia3;  
        sentenciaN;  
        break;  
    ...  
    default:  
        sentencias por defecto  
        break;  
}
```

En el caso anterior, se parte de una variable con un valor determinado. Si dicho valor es `opción1`, se ejecutarán las sentencias 1 y 2. Si el valor de variable es `opción2`, se ejecutarán las sentencias 3 y 4. Y así sucesivamente. Si el valor de variable no es ninguna de los relacionados arriba, se ejecutarán las sentencias por defecto, es decir, aquellas presentes en el bloque correspondiente a la etiqueta `default`. En este caso, la sentencia `break` es necesaria para que una vez se ejecuten las instrucciones asociadas a una opción, el control del programa pase al final del bloque de la sentencia `switch`, es decir, después de la llave de cierre. En el siguiente ejemplo, se parte de una variable numérica con el número del día de la semana. Posteriormente, el programa imprime el nombre del día:



```

public class ClaseControlFlujo {
    public static void main(String[] args) {
        int dia = 3;
        String nombreDia;
        switch (dia) {
            case 1:
                nombreDia = "Lunes";
                break;
            case 2:
                nombreDia = "Martes";
                break;
            case 3:
                nombreDia = "Miércoles";
                break;
            case 4:
                nombreDia = "Jueves";
                break;
            case 5:
                nombreDia = "Viernes";
                break;
            case 6:
                nombreDia = "Sábado";
                break;
            case 7:
                nombreDia = "Domingo";
                break;
            default:
                nombreDia = "Día inválido";
                break;
        }
        System.out.println(nombreDia);
    }
}

```

En este caso, el programa imprimirá:

Miércoles

También, es posible evaluar opciones asociadas a diferentes etiquetas `case` al mismo tiempo. En el siguiente ejemplo, se parte de una variable cuyo valor es el número de un mes (entre 1 y 12). Luego, el programa imprime el número de días que tiene dicho mes tomando en cuenta también el año:



```

public class ClaseControlFlujo {
    public static void main(String[] args) {
        int mes = 2;
        int anio = 2000;
        int numeroDias = 0;

        switch (mes) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                numeroDias = 31;
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                numeroDias = 30;
                break;
            case 2:
                if (((anio % 4 == 0)
                    && !(anio % 100 == 0))
                    || (anio % 400 == 0)) {
                    numeroDias = 29;
                } else {
                    numeroDias = 28;
                }
                break;
            default:
                System.out.println("Mes inválido");
                break;
        }
        System.out.println("Número de días = " + numeroDias);
    }
}

```

Notemos que, para diferentes meses, tenemos el mismo número de días por mes, y por eso se pueden encadenar las etiquetas `case`. En el caso anterior, el programa imprimirá:

```
Número de días = 29
```



## Sentencias while y do-while

Además de la toma de decisiones, los **ciclos** o bucles son otro mecanismo de control de flujo. Un ciclo permite la ejecución de un conjunto de instrucciones un número de veces determinada, o bajo una condición específica.

### Sentencia while

La sentencia `while` permite la ejecución continua de un bloque de sentencias mientras que una condición particular sea verdadera, su sintaxis es la siguiente:

```
while (condición) {  
    sentencias  
}
```

La sentencia `while` evalúa la condición del ciclo al inicio de cada iteración. Si la condición se cumple, procese a ejecutar las sentencias dentro del bloque, encerradas entre llaves (`{ }`). Si la condición no se cumple, o se deja de cumplir. El programa pasa a ejecutar la instrucción posterior al final del bloque. El siguiente ejemplo imprime la tabla de multiplicar del número 3, entre 0 y 10:

```
public class ClaseControlFlujo {  
    public static void main(String[] args) {  
        int numero = 3;  
        int i = 0;  
        while (i <= 10) {  
            System.out.println(numero + " * " + i + " = " +  
numero*i);  
            i = i+1;  
        }  
    }  
}
```





El anterior programa imprime lo siguiente:

```
3 * 0 = 0
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
```

## Sentencia do-while

La sentencia `do-while` funciona de manera similar a la sentencia `while`. La diferencia es que, en la sentencia `do-while`, la condición se evalúa al final de cada iteración y no al principio. Esto hace que las instrucciones dentro de un bloque asociado a un `do-while` se ejecute por lo menos una vez, antes de evaluar si se cumple o no la condición. Su sintaxis es la siguiente (en este caso, después de la condición se escribe punto y coma):

```
do {
    sentencias
} while (condición);
```

En el siguiente ejemplo, se realiza un conteo regresivo usando la sentencia `do-while`:

```
public class ClaseControlFlujo {
    public static void main(String[] args) {
        int i = 10;
        do {
            System.out.println(i);
            i--;
        } while (i > 0);
    }
}
```

Tanto `while` como `do-while` se pueden usar de forma anidada, es decir, un ciclo `while` puede ejecutarse dentro de otro ciclo `while`, y de igual manera ocurre con los ciclos `do-while`.

## Sentencia for

La sentencia `for`, también conocida como ciclo `for`, permite iterar sobre un conjunto de valores mientras se den una o varias condiciones. Se comporta de manera similar a un ciclo `while`, pero permite especificar algunos aspectos del ciclo de forma compacta, su sintaxis en la siguiente:

```
for (inicialización; condición; incremento) {  
    sentencias  
}
```

La anterior es la forma de uso más común del ciclo `for`, donde podemos observar lo siguiente:

- Una **inicialización**, donde se asigna el valor inicial a la o las variables que se iterarán durante el ciclo. Esta inicialización se realiza sólo una vez, al inicio del ciclo, es decir, antes de empezar la primera iteración.
- Una **condición**, que se evalúa al comienzo de cada iteración. Una vez la condición no se cumple (es decir, se evalúa como falsa), el ciclo finaliza.
- Un **incremento**, que se ejecuta al final de cada iteración, hasta que finaliza el ciclo.

El siguiente ejemplo muestra una cuenta entre 1 y 10:

```
public class ClaseControlFlujo {  
    public static void main(String[] args) {  
        for (int i = 1; i < 11; i++) {  
            System.out.println("La cuenta va en: " + i);  
        }  
    }  
}
```

Notemos lo siguiente:

- En la inicialización se define la variable `i`, sobre la que iteraremos empezando en 1. En este caso, no hubo necesidad de declarar la variable entera `i` antes del ciclo `for`. Esto, dado que en la inicialización de un ciclo `for` es posible declarar la variable sobre la cual vamos a iterar (en este caso entre 1 y 10). Cuando declaramos una variable en la inicialización de un ciclo, dicha variable sólo se podrá usar dentro de dicho ciclo.
- En la condición se establece que el ciclo se ejecutará mientras que la variable `i` sea menor que 11.
- El incremento será de a uno por iteración, usando el operador `++`.

El anterior programa imprime lo siguiente:

```
La cuenta va en: 1
La cuenta va en: 2
La cuenta va en: 3
La cuenta va en: 4
La cuenta va en: 5
La cuenta va en: 6
La cuenta va en: 7
La cuenta va en: 8
La cuenta va en: 9
La cuenta va en: 10
```

