Módulo 1: Programación en Java 2023

Tema 5 Sentencias Condicionales, Bucles y Arrays

Tema 5

Contenido

1. Sentencias Condicionales, Bucles y Arrays

1.1. Objetivos

- Describir el funcionamiento de las sentencias selectivas o condicionales (if-else y switch).
- Interpretar el resultado de una secuencia de sentencias condicionales combinadas.
- Codificar una tarea sencilla convenientemente especificada, utilizando la secuencia y combinación de sentencias condicionales.
- Describir el funcionamiento de las sentencias iterativas o bucles (for, while y dowhile)
- Interpretar el resultado de una secuencia simple o combinada de sentencias de control
- Codificar una tarea sencilla convenientemente especificada, utilizando la secuencia y combinación de sentencias iterativas y condicionales.

1.2. Sentencias de control del flujo de un programa

Sin sentencias de control del flujo, el intérprete ejecuta las sentencias conforme aparecen en el programa de principio a fin. Las sentencias de control de flujo se emplean en los programas para ejecutar sentencias condicionalmente, repetir un conjunto de sentencias o, en general, cambiar el flujo secuencial de ejecución.

1.2.1. Sentencia if-else

La instrucción **if ... else** permite controlar qué procesos tienen lugar, típicamente en función del valor de una o varias variables, de un valor de cálculo o booleano, o de las decisiones del usuario.

Sintaxis de la instrucción if:

```
if (expresionLogica) {
sentencia_1;
}
```

Sintaxis de la instrucción if con la cláusula else:

```
if (expresionLogica){
sentencia_1;
}
else {
sentencia_2;
}
```

En el ejemplo anterior, primero se evalúa *expresionLogica*. Si el resultado es true, se ejecuta *sentencia_1*, si el resultado es false, se ejecuta *sentencia_2*. La *expresionLogica* debe ir entre paréntesis. Las llaves sólo son obligatorias si las sentencias son compuestas (las llaves sirven para agrupar varias sentencias simples).

La cláusula **else** no es obligatoria y sirve para indicar instrucciones a realizar en caso de no cumplirse la condición.

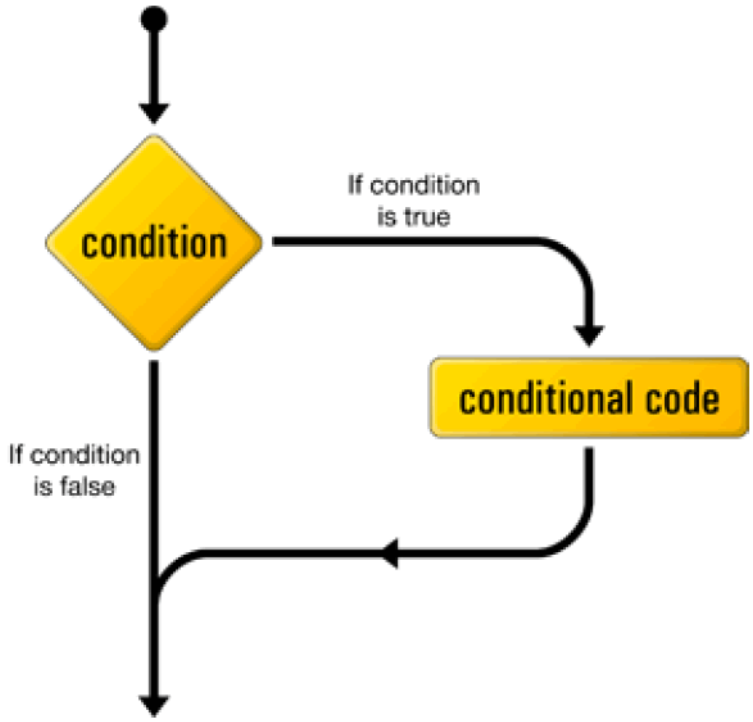

Ilustración 1 Diagrama de Flujo de una Sentencia if-else

Sintaxis:

```
if (expresionLogica_1){
sentencia_1;
}
else if (expresionLogica_2){
sentencia_2;
}
else if (expresionLogica_3){
sentencia_3;
}
else {
sentencia_4;
```

```
}
```

Ejemplo de uso de la sentencia **if-else**:

```
/**
* Ejemplo de uso de la sentencia if-else
*/
public class NumeroMayor {
        public static void main (String [ ] args) {
                int a = 5;
                int b = 10;
                System.out.println("a vale " + 5);
                System.out.println("b vale " + 10);
                if (a > b) {
                        System.out.println("a es mayor que b");
                }
                else {
                        System.out.println("a no es mayor que b");
                }
        }
}
```

La salida del programa anterior es:

```
a vale 5
b vale 10
a no es mayor que b
```

1.2.2. Sentencia if-else con múltiples condiciones

Cuando se quieren evaluar distintas condiciones una detrás de otra, se usa la expresión else if { }. De este modo, la evaluación que se produce es: si se cumple la primera condición, se ejecutan ciertas instrucciones; si no se cumple, comprobamos la segunda, tercera, cuarta... n condición. Si no se cumple ninguna de las condiciones, se ejecuta el else final en caso de existir.

Ejemplo de uso de la sentencia if-else con múltiples sentencias:

```
/**
* Ejemplo de uso de la sentencia if-else con multiples condiciones
```

```
*/
public class Caracter {
public static void main (String [ ] args) {
char c = 'a';
if (c == 'a') {
System.out.println("Es la vocal a");
}
else if (c == 'e') {
System.out.println("Es la vocal e");
}
else if (c == 'i') {
System.out.println("Es la vocal i");
}
else if (c == 'o') {
System.out.println("Es la vocal o");
}
else if (c == 'u') {
System.out.println("Es la vocal u");
}
else {
System.out.println("No es una vocal");
}
}
}
```

Ejemplos de ejecución del programa anterior:

```
Es la vocal a
```

1.2.3. Sentencia Switch

Es una sentencia condicional multiramificada o de selección multiple: dependiendo del valor de una variable o expresion entera permite ejecutar una o varias sentencias de entre muchas. La expresión puede ser de un tipo ordinal (de tipo entero byte, short o int o de tipo carácter char) pero no puede ser de un tipo real o de un tipo cadena.

Sintaxis:

```
switch (expresion) {
case valor_1: sentencias_1; break;
```

```
case valor_2: sentencias_2; break;
...
case valor_n: sentencias_n; break;
[default: sentencias_x;]
}
```

Cada sentencia **case** contiene un único valor distinto del de las demás sentencias case. A continuación del valor se introduce la sentencia o sentencias que se ejecutan en el caso de que el valor indicado coincida con el de la variable o expresión selector. Las sentencias que siguen a cada uno de los valores no se engloban entre llaves, pero suelen ir seguidas de un **break**.

Si la expresión no coincide con ningún valor se ejecuta la sentencia que sigue a default, aunque esta parte (default) no es obligatoria.

Nota: Si no existe algún break, continua la ejecución de la siguiente opción hasta el siguiente break o hasta el final de la sentencia switch.

Ejemplo de uso de la sentencia switch:

```
/**
* Ejemplo de uso de la sentencia switch
*/
public class Caracter {
public static void main (String [ ] args) {
char c = 'a';
switch (c) {
case 'a': System.out.println("Es la vocal a"); break;
case 'e': System.out.println("Es la vocal e"); break;
case 'i': System.out.println("Es la vocal i"); break;
case 'o': System.out.println("Es la vocal o"); break;
case 'u': System.out.println("Es la vocal u"); break;
default: System.out.println("No es una vocal"); break;
}
}
}
```

Ejemplos de ejecución del programa anterior:

```
Es la vocal a
```

1.2.4. Sentencias repetitivas o bucles

Los bucles, iteraciones o sentencias repetitivas modifican el flujo secuencial de un programa permitiendo la ejecución reiterada de una sentencia o sentencias. En Java hay tres tipos diferentes de bucles: **for**, **while** y **do-while**.

1.2.4.1. Sentencia for

Un **for** permite la ejecución de un bloque de código delimitado entre llaves un número determinado de veces. La sintaxis de un bucle for es la siguiente:

```
for (inicio; termino; iteracion)
sentencia;
o si se desean repetir un conjunto sentencias:
for (inicio; termino; iteracion) {
sentencia_1;
sentencia_2;
sentencia_n;
}
```

Es un bucle o sentencia repetitiva que:

1. Ejecuta la sentencia de inicio.
2. Verifica la expresión booleana de término.
    - Si es cierta, ejecuta la sentencia entre llaves y la sentencia de iteración para volver a verificar la expresión booleana de término.
    - Si es falsa, sale del bucle.

Flujo de un bucle **for**:

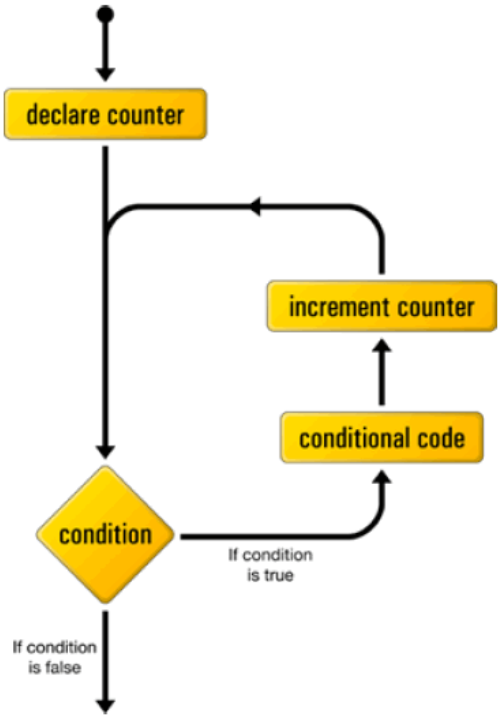

Ilustración 2Diagrama de Flujo de un bucle for

Las llaves sólo son necesarias si se quieren repetir varias sentencias, aunque se recomienda su uso porque facilita la lectura del código fuente y ayuda a evitar errores al modificarlo.

Habitualmente, en la expresión lógica de término se verifica que la variable de control alcance un determinado valor. Por ejemplo:

```
for (i = valor_inicial; i <= valor_final; i++) {
sentencia;
}
```

Es completamente legal en Java declarar una variable dentro de la cabecera de un bucle for. De esta forma la variable (local) sólo tiene ámbito dentro del bucle.

Ejemplo sencillo:

```
System.out.println("Tabla de multiplicar del 5");
for (int i =0; i <= 10; i++) {
System.out.println(5 + " * " + i + " = " + 5*i);
}
```

Salida por pantalla al ejecutar el código anterior:

```
5 * 0 = 0
```

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

1.2.4.2. Sentencia while

Es un **bucle** o **sentencia repetitiva** con una condición al principio. Se ejecuta una sentencia mientras sea cierta una condición. La sentencia puede que no se ejecute ni una sola vez.

Sintaxis:

```
[inicializacion;]
while(expresionLogica){
sentencias;
[iteracion;]
}
```

Flujo de un bucle **while**:

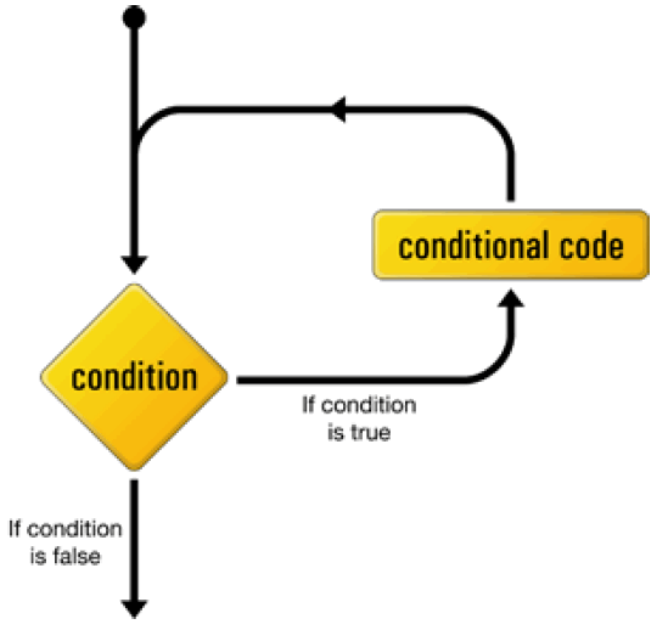

Ilustración 3Diagrama de Flujo de un bucle while

Ejemplo de programa:

```
/**
Ejemplo de sentencia while
*/
public class LeerNumero {
public static void main (String [ ] args) {
Scanner sc = new Scanner(System.in);
int numero = -1;
while (numero <= 0) {
System.out.println("Introduce un numero positivo: ");
numero = sc.nextInt();
}
System.out.println("El numero positivo es " + numero);
sc.close();
}
}
```

Ejemplo de ejecución y salida correspondiente por pantalla:

```
Introduce un numero positivo:
-5
Introduce un numero positivo:
5
El numer positivo es 5
```

1.2.4.3. Sentencia do-while

Es un **bucle** o **sentencia repetitiva con una condición al final**. Se ejecuta una sentencia mientras sea cierta una condición. La diferencia con respecto al bucle *while* es que la sentencia se ejecuta al menos una vez. La sintaxis es la siguiente:

Es un **bucle** o **sentencia repetitiva con una condición al final**. Se ejecuta una sentencia mientras sea cierta una condición. La diferencia con respecto al bucle *while* es que la sentencia se ejecuta al menos una vez. La sintaxis es la siguiente:

```
do {
sentencias;
[iteracion;]
} while (expresionLogica);
```

Flujo de un bucle **do-while**:

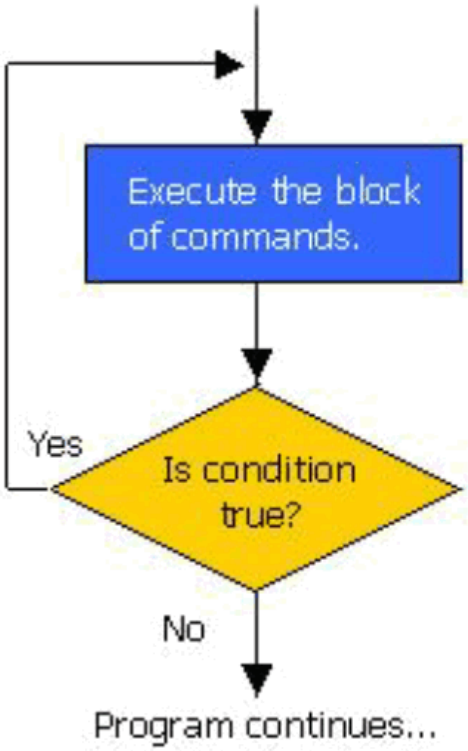

Ilustración 4 Diagrama de Flujo de un bucle do-while

Ejemplo de programa:

```
/**
* Ejemplo de sentencia while
*/
public class LeerNumero {
public static void main (String [ ] args) {
Scanner sc = new Scanner(System.in);
int numero = -1;
do {
System.out.println("Introduce un numero positivo: ");
numero = sc.nextInt();
} while (numero <= 0);
System.out.println("El numero positivo es " + numero);
sc.close();
}
}
```

1.3. Arrays Unidimensionales

Un **array** es una estructura para guardar un conjunto de objetos de la misma clase. Se accede a cada elemento individual del array mediante un número entero denominado **índice** (*index* en inglés). 0 es el índice del primer elemento y *n-1* es el índice del último elemento, siendo *n*, la dimensión del *array*.

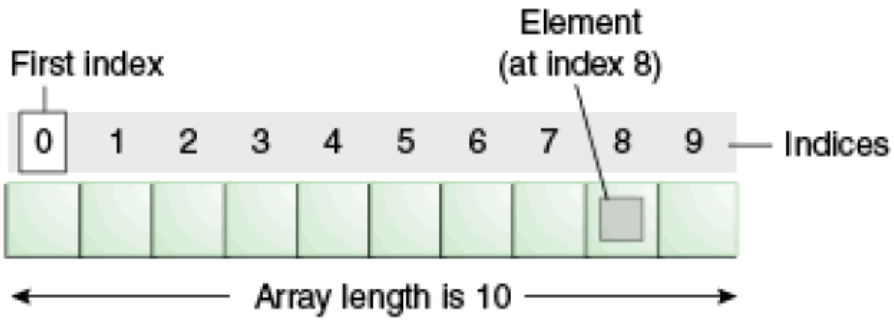

Ilustración 5 Representación gráfica de un array

Para declarar un array se usa la siguiente sintaxis:

```
tipo_de_dato[ ] nombre_del_array;
```

Por, ejemplo, para declarar un array de enteros escribimos lo siguiente:

```
int[ ] numeros;
```

Para crear un array de 4 números enteros escribimos lo siguiente:

```
numeros = new int[4];
```

La declaración y la creación del array se puede hacer en una misma línea:

```
int[ ] numeros = new int[4];
```

Para inicializar el array de 4 enteros escribimos lo siguiente:

```
int[ ] numeros = new int[4];
numeros[0] = 2;
numeros[1] = -4;
numeros[2] = 15;
numeros[3] = -25;
```

Los *arrays* se pueden declarar, crear e inicializar en una misma línea, de la siguiente manera:

```
int[ ] numeros = {2, -4, 15, -25};
String[ ] nombres = {"Juan", "José", "Miguel", "Antonio"};
```

Para imprimir a los elementos de array nombres se escribe

```
for(int i=0; i < nombres.length; i++){
System.out.println(nombres[ i ]);
}
```

Un *array* tiene la propiedad **length**, que retorna su número de elementos.

2. Bibliografía

- Material suministrado por CGA
- https://picodotdev.github.io/blog-bitix/2020/06/las-sentencias-de-control-de-flujo-en-java-if-switch-for-while-do-while-try-catch-break-continue-e-invocacion/
- http://www.edu4java.com/es/java/sentencia-while-variable-contador-acumulador.html