INFORMATORIO

Taller de Programación Web

Estructura de datos, Condicionales e Iterativas

Como se ha visto en los módulos anteriores, las estructuras de datos, estructuras condicionales y las de iteración son similares a los lenguajes anteriores vistos en el Informatorio. A continuación haremos una revisión de las mismas y su sintaxis en JAVA:

La Sentencia if-else:

El formato de la sentencia if-else es el siguiente:

```
if (condicion)
   sentencia1
else
   sentencia2
```

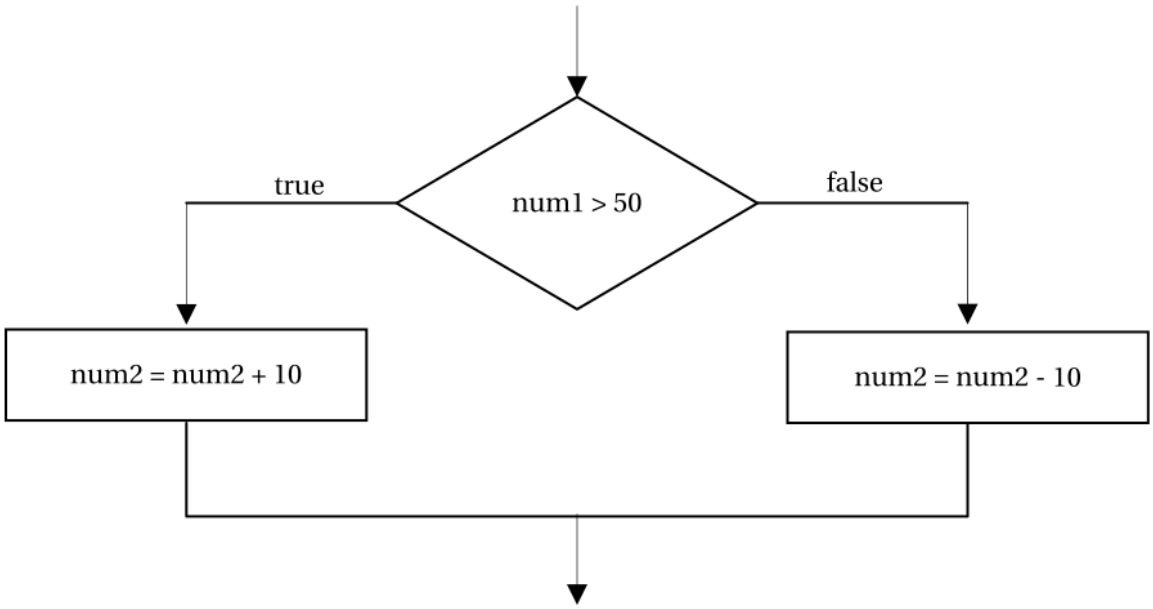
La **condición** debe ser una expresion booleana. Entonces, debe evaluarse si es **true** o **false**. Si la **condición** es true, la **sentencia1** es ejecutada. La parte de **else** es opcional. Se podría escribir lo siguiente:

```
if (condicion)
   sentencia1
```

Ejemplo

```
if (num1 > 50)
   num2 = num2 + 1;
else
   num2 = num2 -1;
```

Podríamos verlo como un diagrama de flujo de la siguiente manera:



Ahora supongamos que queremos ejecutar más de una sentencia dentro del if-else. Ejemplo:

```
if (num1 > 50)
   System.out.println("Ejecutando sentencias dentro del bloque if");
   num2 = num2 + 1;
else
System.out.println("Ejecutando sentencias dentro del bloque else");
   num2 = num2 -1;
```

El siguiente código producirá un error de compilación. ¿Por qué sucede esto? Solo se puede colocar una sentencia debajo de la sentencia if y la sentencia else. Si queremos realizar la ejecución de más de una sentencia deberemos encerrarlas en un bloque de sentencia.


```java
if (num1 > 50) {
   System.out.println("Ejecutando sentencias dentro del bloque if");
   num2 = num2 + 1;
} else {
   System.out.println("Ejecutando sentencias dentro del bloque else");
   num2 = num2 - 1;
}
```

La sentencia if-else permite la anidación de la misma. Esto quiere decir que dentro de una sentencia if o else, pueden haber otras sentencias if-else.

Términos:
Sentencia: Especifica la acción/ejecución de un programa de JAVA, como puede ser la asignación de una variable, la suma de x más y, imprimir un mensaje por pantalla.
Sentencia de Bloque: Es una secuencia de cero o más sentencias encerradas por llaves.

La Sentencia switch:

```java
switch (expresion_switch) {
   case etiqueta1:
       //sentencias
   case etiqueta2:
       //sentencias
   default:
       //sentencias
}
```

La sentencia `switch` debe evaluar a un tipo: `byte, short, char, int, enum` or `String`. Las etiquetas son valores que deben estar en el rango del tipo. Una sentencia `switch` se evalúa de la siguiente manera:

- La sentencia expresion_switch se evalúa.
- Si el valor de la expresion expresion_switch coincide (match) con la etiqueta de algún `case`, la ejecución comenzará desde la etiqueta del case que coincide y ejecutará todas las sentencias hasta el final de la sentencia `switch`.
- Si el valor de la expresion expresion_switch no coincide con alguna etiqueta de los `case`, la ejecución comienza desde la etiqueta `default` (que es opcional) y continuará hasta el final de la sentencia `switch`.

Ejemplo

```
int i = 10;
switch (i) {
   case 10: //Encuentra coincidencia
       System.out.println("Diez");
   case 20:
       System.out.println("Veinte");
   default:
       System.out.println("No hay coincidencias");
}
```

Salida del Programa (impresión en la consola):

```
Diez
Veinte
No hay coincidencias
```

El valor de `i` es 10. La ejecución comienza en la primera sentencia seguido de `case: 10` y continua a través de las etiquetas `case: 20` y `default` ejecutando todas sus sentencias. Si cambiamos el valor de `i` a 50, no habrá coincidencias

en ningún caso y la ejecución comenzará en la primera sentencia luego de la etiqueta **`default`**.

Ejemplo

```
int i = 50;
switch (i) {
   case 10:
       System.out.println("Diez");
   case 20:
       System.out.println("Veinte");
   default: //La ejecución comienza aqui
       System.out.println("No hay coincidencias");
}
```

Salida del Programa (impresión en la consola):

```
No hay coincidencias
```

La sentencia **`default`** es opcional y si la remoomitieramos del ejemplo anterior, no se producirían coincidencia ni tampoco habría sentencias a ejecutar y no habría ninguna salida o impresión por consola.

Ahora, volviendo al primer ejemplo donde **`i`** vale 10 y la sentencia switch ejecutaba todas las sentencias de las etiquetas. Podemos alterar el comportamiento para que cuando se produzca una coincidencia solo se ejecuten las sentencias de dicho **`case`** o **`default`**. Para eso agregaremos una sentencia **`break`**, que al ser ejecutada dentro de una sentencia **`switch`**, transfiere el control fuera de la sentencia **`switch`**.

Ejemplo:

```
int i = 10;
```

```
switch (i) {
   case 10:
       System.out.println("Diez");
       break;  //Transfiere el control fuera del switch
   case 20:
       System.out.println("Veinte");
       break; //Transfiere el control fuera del switch
   default:
       System.out.println("No hay coincidencias");
       break; //Transfiere el control fuera del switch
}
```

Salida del Programa (impresión en la consola):

```
Diez
```

La sentencia **switch** es una manera más clara de escribir sentencias if-else anidadas.

La Sentencia for:

La sentencia for es una sentencia de iteración (también llamada sentencia bucle for), que es usada para ejecutar sentencias dentro de un bucle por un determinado número de veces en base a una condición. La forma de representarlo es:

```
for (inicializacion; expresion_condicional;
lista-expresion)
   sentencia
```

La sentencia for posee 4 partes:

- Inicialización

- Expresión condicional
- Sentencia
- Lista de Expresiones

Ejemplo:

```
for(int num = 1; num <= 10; num++)
   System.out.println(num);
```

En el ejemplo de arriba se ejecuta primero la inicialización (`int num = 1`), que declara una variable `num` de tipo `int` y la inicializa con 1. Es importante saber, que la variable declarada en la inicialización, sólo podrá ser usada dentro del bucle `for`.

Luego, la expresión condición (`num <=10`) es evaluada, siendo al principio de la iteración o bucle `1 <= 10`, que devolverá true la primera vez. A continuación, la sentencia asociada al bucle for se ejecuta (en este caso `println`).
Finalmente, la expresión dentro de la lista de expresiones, `num++`, es evaluada, que incrementará el valor de `num` en 1. La expresion_condicional ahora será `2 <= 10`, que retorna `true` y luego se ejecutara la sentencia `println`. El proceso continuará hasta que la expresion_condicional `11 <= 10` sea `false`, que detendrá la ejecución de la sentencia de bucle for.

Tipo	Ejemplo	Observación
Operador Unario	contador++	El contenido de la variable contador se incrementa en 1
Operador Binario	4 + 5	4 sería el operando1 y 5 el operando2
Operador Ternario	**esDomingo ? noSeTrabaja : seTrabaja**	**operando1 operador1 operando2 operador2 operando3**

La Sentencia for-each:

La sentencia for-each es usada para iterar sobre los elementos de una colección o array (Colecciones y Array se verán en el próximo Tema). Y su sintaxis será:

```
for(Type element : una_collection_o__array) {
   // Este código sera ejecutado una vez por cada
   // elemento en la colección/array.
   // Cada vez que se ejecute este bloque, la variable
   // elemento contendrá la referencia
   // del elemento actual en la collection/array
}
```

```
int[] numList = {10, 20, 30, 40};
for(int num : numList) {
   System.out.println(num);
}
```

La Sentencia while:

La sentencia **`while`** es otra de las sentencias de iteración (o bucle), que es utilizada para ejecutar sentencias de manera repetida siempre y cuando la condición sea **`true`**. Su sintaxis será:

```
while (expresion_condicional)
   sentencia
```

Ejemplo: Si quisiéramos imprimir todos los números de 1 a 10

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

La Sentencia do-while:

La sentencia **`do-while`** es similar al bucle **`while`** con una pequeña diferencia. En la sentencia while puede que las sentencias encerradas no se ejecuten porque al evaluar la expresión por primera vez esta devuelve **`false`**. En cambio, la sentencia **`do-while`** es ejecutada al menos una vez.

```
do
   sentencia
while (expression_condicional);
```

Ejemplo: Si quisiéramos imprimir todos los numeros de 1 a 10

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

La Sentencia break:

Hemos visto en la sección de sentencia **`switch`** como se puede usar una sentencia **`break`**. Podemos decir, que una sentencia **`break`** es usada para salir de una sentencia en bloque. Puede ser usada también dentro de las sentencias de iteración.

La Sentencia continue:

La sentencia continua solo puede ser usada en las sentencias de iteración for, while y do-while. Cuando una sentencia **`continue`** es ejecutada dentro de una sentencia de iteración, las siguientes sentencias dentro del cuerpo de la sentencia de iteración son ignoradas y da lugar a la próxima ejecución del bucle.

Ejemplo: Si quisiéramos imprimir solo los números impares del 1 al 10

```
for (int i = 1; i < 10; i++) {
   if (i % 2 == 0) {
       continue;
   }
   System.out.println(i);
}
```