Condicionales y ciclos en JAVA

Dentro de cualquier lenguaje de programación existen una serie de estructuras que nos permiten poder definir una lógica no lineal para trabajar en la aplicación.

En cualquier programa aparecen a menudo preguntas que necesitamos responder para que la aplicación realice una acción determinada u otra. Mediante las sentencias podemos realizar dicha lógica y responder a las preguntas que vayan apareciendo dentro de un programa para hacer una secuencia o una evaluación de variables.

Sentencia if-else

Permite seleccionar la ejecución de distintos fragmentos de código dependiendo del resultado de la valoración de una expresión booleana.

Su sintaxis es la siguiente:

if (expresion_booleana 1) { // La expresión booleana expresa condiciones

. . .

// código que se ejecuta si la expresión devuelve true

. . .

} else if (expresión boolean 2) {

. . .

// código que se ejecuta si la expresión devuelve true

. . .

} else {

// código que se ejecuta si la expresión devuelve false

}

El uso de else es opcional.

La expresión booleana puede contener operadores relacionales para encadenar condiciones.

Vamos a visualizar un ejemplo práctico en el que compararemos el valor de una variable utilzando If, Else y Else If:

```
public class ClaseIfElse{

           public static void main(String args[])      +

          {

                    int i=0;

                    String mensaje="";

                    if (i>0)

                    {

                              mensaje = "Positivo";
```

```
                    }else if (i<0)
                    {
                            mensaje = "Negativo";
                    }else
                    {
                            mensaje = "Cero";
                    }
                    System.out.println("El valor de i es " + mensaje);
            }
}
```

Sentencia switch-case

Se utiliza cuando se quieren analizar muchas posibilidades de una única variable o expresión. Case es el caso de control y determina los valores posibles de la variable.

Su sintaxis es la siguiente:

```
switch (expresión)
{
case valor1:
//sentencias1;
break;
case valor2:
//sentencias2;
break;
case valor3:
//sentencias3;
break;
case valor4:
//sentencias4;
break;
default:
//sentencias_default
break;
}
```

La sentencia default es opcional y se ejecutará en todos los casos en el que el valor devuelto por expresión no coincida con los especificados en los case. El valor de cada case debe ser un entero positivo loq ue limita en algo esta sentencia.

La sentencia break (sin etiqueta) hace que se salga del dominio de la sentencia switch. Con ello se evita que se valoren todas las opciones que quedan una vez que se ha encontrado la correcta. Es opcional.

Vamos a visualizar un ejemplo en el evaluaremos una variables entera y mostraremos un mensaje dependiendo del valor obtenido en los diferentes casos que escribamos en el programa.

```
public class EjemploSwitch{
          public static void main(String args[])
          {
                    String mensaje="";
                    int i=2;
                    switch (i)
                    {
                              case 0:
                                        mensaje = "El número es cero";
                                        break;
                              case 1:
                                        mensaje = "El número es uno";
                                        break;
                              case 2:
                                        mensaje = "El número es dos";
                                        break;
                              case 3:
                                        mensaje = "El número es tres";
                                        break;
                              case 4:
                              case 5:
                              case 6:
                                        mensaje = "El número es cuatro, cinco o seis";
                                        break;
                              default:
                                        mensaje = "El número es otro no evaluado";
                     }
                     System.out.println("He salido del Switch");
                     System.out.println("El mensaje es " + mensaje);
          }
}
```

Sentencia while

Ejecuta repetidamente un conjunto de sentencias mientras una expresión booleana sea verdadera.

Su sintaxis es:

while (Expresión_booleana) { //Condición de permanencia

...

sentencias;

...

Dentro del bucle se deberá alterar la variable que controla el bucle de lo contrario nunca se dejará de cumplir la condición.

Se puede realizar un bucle infinito si no se incluye una salida dentro del bucle para finalizar su ejecución.

Para visualizar el comportamiento del bucle while vamos a crearnos un ejemplo práctico en el que incrementaremos una variable entera.

```
public class EjemploWhile{
          public static void main(String args[])
          {
                    String mensaje="";
                    int i=0;
                    while (i <= 10)
                    {
                              i++;
                              System.out.println("Valor de i: " + i);
                    }
          }
}
```

Podemos comprobar que la ejecución del programa permanece en el bucle hasta que la variable alcanza la condición deseada.

Sentencia do-while

Igual que la sentencia **while**, pero las sentencias se ejecutan al menos una vez antes de valorar las expresión, eso nos permite saber que al menos se ejecutará la sentencia del bucle una vez, permitiéndonos evaluar posteriormente.

Su sintaxis:

do {

...

sentencias;

...

} while (Expresión_booleana);

Vamos a realizar un ejemplo práctico para visualizar el comportamiento de éste tipo de sentencia. Lo que haremos será crearnos una variable con valor superior a la condición del bucle y podremos comprobar que primero entra en el bucle y después evalua el valor de la variable creada.

```
public class EjemploDoWhile{
          public static void main(String args[])
          {
                    String mensaje="";
                    int i=20;
                    System.out.println("Estoy al inicio del bucle");
                    do
                    {
                              i++;
                              System.out.println("Valor de i: " + i)
                    }while (i <= 10);
                    System.out.println("Fin del bucle");
          }
}
```

La acción se realiza al menos una vez, que es la principal característica de éste tipo de bucle.

Sentencia for

Este bucle permite establecer la inicialización de la variable que controla el bucle, su variación y la condición de permanencia, en la propia definición del for.

Es un bucle numérico, por lo que su condición de salida se establece en el número de veces que estará dentro del bucle, permitiéndonos controlar totalmente el número exacto de repeticiones en las que se realizará una misma acción.

Su sintaxis:

for (inicialización; condición de permanencia; variación){

...

sentencias

...

}

La inicialización hace referencia a la variable de control del bucle.

La condición de permanencia es una expresión booleana donde debe encontrarse la variable de control.

La actualización hace referencia a la variable de control.

Vamos a visualizar un ejemplo en el que mostraremos los números pares que hay entre 0 y 20. Para ello utilizaremos un bucle contador debido a que sabemos el número exacto de veces que tiene que repetir la acción:

```
public class EjemploFor{
          public static void main(String args[])
          {
                    System.out.println("Estoy al inicio del bucle");
                    System.out.println("Numeros pares");
                    for (int i=0; i<20;i++)
                    {
                              if (i%2==0)
                              {
                                        System.out.println("Valor de i: " + i);
                              }
                    }
                    System.out.println("Fin de programa");
          }
}
```

Existen múltiples combinaciones, también podríamos cambiar el incremento del bucle para que realice la acción de dos en dos y no evalue el valor de la variable.

```
public class EjemploFor{
          public static void main(String args[])
          {
                    System.out.println("Estoy al inicio del bucle");
                    System.out.println("Numeros pares");
                    for (int i=0; i<20;i+=2)
                    {
                              System.out.println("Valor de i: " + i);
                    }
                    System.out.println("Fin de programa");
          }
}
```

CONTINUE

Permite volver al inicio del bucle para comenzar una nueva iteración sin que la iteración actual se lleve a cabo del todo.

Ejemplo

```
public class Ejemplo {
          public static void main(String[] args) throws IOException{
                    bf: for ( int i=1; i <5; i++ ) {
```

```
                              for ( int j=1; j<5; j++ ) {
                                        if ( j>i ) {
                                                  System.out.println(" ");
                                                  continue bf;
                                        }
                                        System.out.print( " " + (i*j) );
                              }
                    }
          }
}
```

BREAK

Permite salir del bucle aunque la condición de permanencia todavía se cumpla.

Ejemplo

```
public class Ejemplo {
          public static void main(String[] args) throws IOException{
                    bf:  for( int i=1; i<10; i++ ){
                              System.out.print(" i="+i);
                              for( int j=1; j<10; j++ ){
                                        if ( j==5 ){
                                                  break bf; //Con break salimos de los dos bucles
                                                  System.out.print(" j="+j);
                                        }
                                        System.out.print("Este mensaje nunca lo imprimirá");
                              }
                     }
}
```

Bibliografía:

- Extraído de <<Condicionales y ciclos en JAVA>>. 9 de octubre de 2018. Consultado el 28 de noviembre del 2022