Estructuras de control en Java

Índice de contenidos

A. Introducción

Durante un programa existen acciones que se han de repetir un número determinado de veces. Por ejemplo, leer 3 caracteres de un flujo de entrada *in* se codificaría:

```
in.read();

in.read();

in.read();
```

Este código además de poco elegante sería inviable para una repetición de 3000 lecturas. Por eso aparecen las estructuras de control, que facilitan que determinadas acciones se realicen varias veces, mientras que una condición se cumpla, y, en definitiva, tomar decisiones de qué hacer en función de las condiciones que se den en el programa en un momento dado de su ejecución.

Así, nuestro ejemplo se podría indicar como:

```
int i=0;

for ( i=0 ; i <= 3 ; i++ )

in.read();
```

Donde bastaría cambiar el 3 por cualquier otro número para que la lectura se repitiese ese número de veces.

El lenguaje Java soporta las estructuras de control:

Sentencia	**Clave**
Alternativa	if-else, switch-case
Repetitiva	for, while, do-while
De Salto	break, continue, return

B. Las sentencias condicionales: if y switch

a.) La sentencia if - else

La sentencia *if-else* de Java dota a los programas de la habilidad de ejecutar distintos conjuntos de sentencias según algún criterio.

La sintaxis de la sentencia *if-else* es:

```
if ( condición ){
    Bloque de código a ejecutar si la condición es cierta
}
else{
    Bloque de código a ejecutar si la condición es falsa
}
```

La parte del *else* es opcional, y un bloque de código puede ser simplemente la sentencia vacía*;* para representar que en ese caso no se ha de ejecutar nada.

Supongamos que un programa debe realizar diferentes acciones dependiendo de si el usuario oprime el botón aceptar o el botón cancelar en una ventana de dialogo. Nuestro programa puede realizar esto usando la sentencia *if - else*:

```
String respuesta = "";
        // La respuesta es Aceptar o Cancelar
        if (respuesta.equals("Aceptar")) {
            // código para realizar la acción Aceptar
            System.out.println("Su petición está   atendida");
        } else {
            // código para realizar la acción Cancelar
            System.out.println("Cancelando acción");
        }
```

Se pueden anidar expresiones *if-else*, para poder implementar aquellos casos con múltiples acciones. Esto es lo que se suele denominar como sentencias *elseif*.

Por ejemplo, supongamos que se desea escribir un programa que clasifique según el contenido de una variable *valor*, asigne una letra a una variable *clasificacion*: A paraun valor del 100-91, B de 90-81, C para 80-71 y F si no es ninguno de los anteriores:

```
int valor = 0;

char clasificacion;

        if (valor > 90) {

                clasificacion = 'A';

        } else if (valor > 80) {

                clasificacion = 'B';

        } else if (valor > 70) {

                clasificacion = 'C';

        } else {

                clasificacion = 'F';

        }
```

Se pueden escribir los *if* en las mismas líneas que los *else*, pero se insta a utilizar la forma indentada (como se ha podido ver en el ejemplo), pues es más clara para el lector.

Este sistema de programación (*elseif)* no es demasiado recomendable, y por ello el lenguaje Java incluye la sentencia *switch,* que veremos a continuación, para dirigir el flujo de control de variables con múltiples valores.

b.) La sentencia switch

Mediante la sentencia *switch* se puede seleccionar entre varias sentencias según el valor de cierta expresión.

La forma general de *switch* es la siguiente:

switch (expresionMultivalor) {

case valor1 : conjuntoDeSentencias; break;

case valor2 : conjuntoDeSentencias; break;

case valor3: conjuntoDeSentencias; break;

default: conjuntoDeSentencias; break;

}

La sentencia *switch* evalúa la *expresiónMultivalor* y ejecuta el *conjuntoDeSentencias* que aparece junto a la cláusula *case* cuyo *valor* corresponda con el de la *expresiónMultivalor*.

Cada sentencia *case* debe ser única y el *valor* que evalúa debe ser del mismo tipo que el devuelto por la *expresiónMultivalor* de la sentencia *switch*.

Las sentencias *break* que aparecen tras cada *conjuntoDeSentencias* provocan que el control salga del *switch* y continúe con la siguiente instrucción al *switch*. Las sentencias *break* son necesarias porque sin ellas se ejecutarían secuencialmente las sentencias *case* siguientes. Existen ciertas situaciones en las que se desea ejecutar secuencialmente algunas o todas las sentencias *case*, para lo que habrá que eliminar algunos *break*.

Finalmente, se puede usar la sentencia *default* para manejar los valores que no son explícitamente contemplados por alguna de las sentencias *case*. Su uso es altamente recomendado.

Por ejemplo, supongamos un programa con una variable entera *meses* cuyo valor indica el mes actual, y se desea imprimir el nombre del mes en que estemos. Se puede utilizar la sentencia *switch* para realizar esta operación:

int meses;

switch (meses){

case 1: System.out.println("Enero"); break;

case 2: System.out.println("Febrero"); break;

```
case 3: System.out.println( "Marzo" ); break;

//Demás meses

    // . . .

case 12: System.out.println( "Diciembre" ); break;

default: System.out.println( "Mes no válido" ); break;

}
```

Por supuesto, se puede implementar esta estructura como una sentencia *if elseif*:

```
intmeses;

if ( meses == 1 ) {

System.out.println( "Enero" );

}

else
if ( meses == 2 ) {

System.out.println( "Febrero" );

}

    // Y así para los demás meses
```

El decidir si usar la sentencia *if* o *switch* depende del criterio de cada caso. Se puede decidir cuál usar basándonos en la legibilidad, aunque se recomienda utilizar *switch* para sentencias con más de tres o cuatro posibilidades.

C. Sentencias de iteración o bucles: while, do- while, for

a.) Bucle while

El bucle *while* es el bucle básico de iteración. Sirve para realizar una acción sucesivamente mientras se cumpla una determinada condición.

La forma general del bucle *while* es la siguiente:

```
while ( expresiónBooleana ) {

sentencias;

};
```

Las *sentencias* se ejecutan mientras la *expresiónBooleana* tenga un valor de
verdadero.

Se utiliza, por ejemplo para estar en un bucle del que no hay que salir hasta que nose cumpla una determinada condición.

Por ejemplo, multiplicar un número por 2 hasta que sea mayor que 100:

```
int i = 1;

while ( i <= 100 ) {

    i = i * 2;

    System.out.println(i);

}
```

b.) Bucle do-while

El bucle *do-while* es similar al bucle *while*, pero en el bucle *while* la expresión seevalúa al principio del bucle y en el bucle *do-while* la evaluación se realiza al final.

La forma general del bucle *do-while* es la siguiente:

```
do {

sentencias;

} while ( expresiónBooleana );
```

La sentencia *do-while* es el constructor de bucles menos utilizado en la programación, pero tiene sus usos, cuando el bucle deba ser ejecutado por lo menos una vez.

Por ejemplo, se controla una entrada por teclado:

```
do {
        System.out.println("Escriba un valor negativo: ");
        valor = teclado.nextDouble();
} while (valor > 0);
```

Por ejemplo, cuando se lee información de un archivo, se sabe que siempre se debeleer por lo menos un carácter, y si no hay se lee -1:

```
int c;

do {

   c = System.in.read( );

// Sentencias para tratar el carácter c

} while ( c != -1 ); // No se puede leer más (Fin fichero)
```

c.) Bucle for

Mediante la sentencia *for* se resume un bucle *do-while* con una iniciación previa. Es muy común que en los bucles *while* y *do-while* se inicien las variables de control de número de pasadas por el bucle inmediatamente antes de comenzar los bucles. Por eso el bucle *for* está tan extendido.

La forma general de la sentencia *for* es la siguiente:

for (iniciación ; condición ; incremento){

sentencias;}

La iniciación es una sentencia que se ejecuta una vez antes de entrar en el bucle.

La condición es una expresión que determina que mientras se cumpla cierta condición se debe continuar en el bucle. Esta expresión se evalúa al final de cada iteración del bucle. Cuando la expresión se evalúa a falso, el bucle termina.

El incremento es una expresión que es invocada en cada iteración del bucle. En realidad, puede ser una acción cualquiera, aunque se suele utilizar para incrementar una variable contador:

for (i = 0 ; i < 10 ; i++)

Algunos (o todos) estos componentes pueden omitirse, pero los puntos y coma siempre deben aparecer (aunque sea sin nada entre sí).

Se debe utilizar el bucle *for* cuando se conozcan las restricciones del bucle (su instrucción de iniciación, criterio de terminación e instrucción de incremento).

for (int i = 0; i <100; i++){

suma=suma+i;

System.out.println(suma);

}

D. Sentencias de salto: break, continue y return

a.) Sentencia break

La sentencia *break* provoca que el flujo de control salte a la sentencia inmediatamente posterior al bloque en curso. Ya se ha visto anteriormente la sentencia *break* dentro de la sentencia *switch*.

El uso de la sentencia break con sentencias etiquetadas es una alternativa al uso dela sentencia *goto*, que no es soportada por el lenguaje Java.

Se puede etiquetar una sentencia poniendo un identificador Java válido seguido por dos puntos antes de la sentencia:

nombreSentencia: sentenciaEtiquetada

La sentencia *break* se utiliza para salir de una sentencia etiquetada, llevando el flujo del programa al final de la sentencia de programa que indique:

break nombreSentencia2;

Un ejemplo de esto sería el programa:

```
//void gotoBreak() {

System.out.println("Ejemplo de break con etiqueta (como 'goto')");

a: for( int i=1; i<10; i++ ){

System.out.print(" i="+i);

for(int j=1; j<10; j++ ){

if ( j==5 )

break a; //Sale de los dos bucles!!!

System.out.print(" j="+j);

}

System.out.print("No llega aquí");

    }}
```

Al interpretar *break a*, no solo se rompe la ejecución del bucle interior (el de *j*), sino que se salta al final del bucle *i*, obteniéndose:

i=1 j=1 j=2 j=3 j=4

Nota: Se desaconseja esta forma de programación, basada en *go to*, y con saltos deflujo no controlados.

b.) Sentencia continue

Del mismo modo que en un bucle se puede desear romper la iteración, también se puede desear continuar con el bucle, pero dejando pasar una determinada iteración.

Se puede usar la sentencia *continue* dentro de los bucles para saltar a otra sentencia, aunque no puede ser llamada fuera de un bucle.

Tras la invocación a una sentencia *continue* se transfiere el control a la condición de terminación del bucle, que vuelve a ser evaluada en ese momento, y el bucle continúa o no dependiendo del resultado de la evaluación. En los bucles *for* además en ese momento se ejecuta la cláusula de incremento (antes de la evaluación).

Por ejemplo, el siguiente fragmento de código imprime los números del 0 al 9 no divisibles por 3:

```
for ( int i = 0 ; i < 10 ; i++ ){

if ( ( i % 3 ) == 0 )

continue;

System.out.print( " " + i );

}
```

Del mismo modo que *break*, en las sentencias *continue* se puede indicar una etiqueta de bloque al que hace referencia. Con ello podemos referirnos a un bloque superior, si estamos en bucles anidados. Si dicha etiqueta no es indicada, se presupone que nos referimos al bucle en el que la sentencia *continue* aparece.

Por ejemplo, el siguiente fragmento de código:

```
//void gotoContinue( ){

f: for ( int i=1; i <5; i++ ){

for ( int j=1; j<5; j++ ){

if ( j>i ){

System.out.println(" ");

continue f;                }
```

*System.out.print(" " + (i*j));} } }*

En este código la sentencia *continue* termina el bucle de *j* y continua el flujo en la siguiente iteración de i. Ese método imprimiría:

1

2 4

3 6 9

4 8 12 16

Nota: Se desaconseja esta forma de programación, basada en *goto*, y con saltos deflujo no controlados.

c.) Sentencia return

La última de las sentencias de salto es la sentencia *return*, que puede usar para salir del método en curso y retornar a la sentencia dentro de la cual se realizó la llamada.

Para devolver un valor, simplemente se debe poner el valor (o una expresión que calcule el valor) a continuación de la palabra *return*. El valor devuelto por *return* debe coincidir con el tipo declarado como valor de retorno del método.

Cuando un método se declara como *void* se debe usar la forma de *return* sin indicarle ningún valor. Esto se hace para no ejecutar todo el código del programa:

int contador;

boolean condicion;

int devuelveContadorIncrementado(){

return ++contador;

}

//void metodoReturn(){

//Sentencias

if (condicion == true)

return;

//Más sentencias a ejecutar si condición no vale true