

CAPÍTULO 5

1 / 1

» Control de flujo, excepciones y afirmaciones

Declaraciones if y switch

Las declaraciones if y switch se denominan comúnmente como declaraciones de decisión.

Cuando se usan declaraciones de decisión en un programa, se le pide al mismo programa que evalúe una expresión dada para determinar qué curso de acción se debe tomar.

Ramificación if-else

El formato básico de declaración if es la siguiente:

```
if (booleanExpression) {  
    System.out.println("Inside if statement");  
}
```

La expresión entre paréntesis debe evaluarse como (un valor booleano) verdadero o falso. Por lo general, está probando algo para ver si es cierto y luego ejecuta un bloque de código (una o más declaraciones) si es verdadera, y (opcionalmente) otro bloque de código si no lo es.

Hay un par de reglas para usar else-if:

- Puede tener cero o un else para cada if y debe ir después de cualquier else ifs.
- Puede tener de cero a muchos otros ifs para un if dado y deben venir antes del (opcional) else
- Una vez que un else if tiene éxito, ninguno de los demás ifs o elses restantes ser probado.

Expresiones Legales para declaraciones if

La expresión en una instrucción if debe ser una expresión booleana (cualquier expresión que se resuelve en un booleano está bien, y algunas de las expresiones pueden ser complejas).

Declaraciones Switch

Una forma de simular el uso de múltiples sentencias if es con la sentencia switch.

Las declaraciones de ruptura (break) son opcionales y su inclusión o la exclusión provoca grandes cambios en la forma en que se ejecutara una instrucción de cambio.

Expresiones legales para switch y case

La forma general de la declaración de cambio es:

```
switch (expression) {
    case constant1: code block
    case constant2: code block
    default: code block
}
```

La expresión de un switch debe evaluarse en un char, byte, short, int o a partir de Java 6, una enumeración. Esto significa que si no se está usando una enumeración, solo las variables y los valores que se pueden promover automáticamente (en otras palabras, emitir implicitamente) a un int son aceptables. No podrá compilar si usa cualquier otra cosa, incluidos los tipos numéricos restantes de largo, float y double.

Una constante de caso (case) debe evaluarse al mismo tipo que la expresión de cambio que se pueda usar, con una restricción adicional (y grande): la constante de caso debe ser una constante de tiempo de compilación. Dado que el argumento del caso debe resolverse en tiempo de compilación, significa que puede usar solo una variable constante o final a la que se le asigne un valor literal. No basta con ser final, debe ser una constante de tiempo de compilación. Además, el switch (interruptor) solo puede verificar la igualdad. Esto significa que los otros operadores relacionales como '>' (mayor que) se vuelven inutilizables en un case.

Rotura y caída en bloques de interruptores

Break and Fall-Through in switch Blocks

Lo más importante que se debe recordar sobre el flujo de ejecución a través de una declaración de cambio es:

- Las constantes de casos se evalúan de arriba hacia abajo, y la primera constante de casos que coincide con la expresión del comutador es el punto de entrada de ejecución.

En otras palabras, una vez que se empieza una constante de caso(case), la JVM ejecutará el bloque de código asociado y todos los bloques de código subsiguientes (salvo una declaración de interrupción) también se ejecutarán.

El caso predeterminado

The Default Case

Una instancia default, cuando es declarada, es ejecutada si el valor de la expresión no coincide con cualquiera de las otras instancias case.

Bucles e Iteradores

Loops and Iterators

Los bucles de Java vienen en tres versiones: while, do y for (a partir de Java 6, el bucle for tiene dos variaciones). Los tres tipos de bucle permiten repetir un bloque de código siempre que alguna condición sea verdadera o para un número específico de iteraciones.

Usando bucles While

Using While Loops

El bucle while es bueno para escenarios en los que no sabe cuántas veces debe repetirse un bloque o declaración, pero desea continuar con el bucle siempre que alguna condición sea verdadera. Una declaración while se ve así:

```
while (expression) {
    //do stuff
}
```

o

```
int x=2;
while (x==2) {
    System.out.println(x);
    ++x;
}
```

En este caso, como en todos los bucles, la expresión (prueba) debe evaluar a un resultado booleano. El cuerpo del ciclo while solo se ejecutará si la expresión (a veces llamada "condición") da como resultado un valor verdadero. Una vez dentro del ciclo, el cuerpo del ciclo se repetirá hasta que la condición ya no se cumpla porque se evalúa como false.

Usando do loops

Using do loops

El bucle do es similar al bucle while, excepto que la expresión no se evalúa hasta que se ejecuta el código del bucle do. Por lo tanto, se garantiza que el código en un bucle de ejecución se ejecutara al menos una vez.

El bucle do siempre ejecutará el código en el cuerpo del bucle al menos una vez.

Uso de bucles for

Using for loops

A partir de Java 6, el bucle for adquirió una segunda estructura, en donde el antiguo bucle for se conoce como el "bucle for básico" y el nuevo como "bucle for mejorado".

El bucle for básico es más flexible que el bucle for mejorado, pero el bucle for mejorado fue diseñado para hacer que la iteración a través de matrices y colecciones sea más fácil de codificar.

Lo básico para bucles (BpB)

True basic for loop

El bucle for es especialmente útil para el control de flujo cuando ya sabe cuántas veces necesita ejecutar las declaraciones en el bloque del bucle. La declaración de bucle for tiene tres partes principales, además del cuerpo del bucle:

- * Declaración e inicialización de variables
- * La expresión booleana (prueba condicional)
- * La expresión de iteración

↳ BpB Declaración e inicialización

La primera parte de la instrucción for, permite declarar e inicializar cero, una o varias variables del mismo tipo dentro del parentesis después de la palabra clave for. Si se declaran más de una variable del mismo tipo, se deberán separar con comas (,) como se muestra a continuación:

```
for (int x=10, y=3; y>3; y++) {}
```

↳ BpB Expresión condicional (Boolean)

La expresión condicional, como todas las demás pruebas, debe de evaluar a un valor booleano. Puede tener solo una expresión lógica, pero puede ser muy compleja.

La regla a recordar es la siguiente:

Solo se puede tener una expresión de prueba.

↳ BpB Expresión de Iteración

Después de cada ejecución del cuerpo del bucle for, se ejecuta la expresión de iteración. Aquí es en donde se puede decir lo que se quiere que suceda con cada iteración del ciclo.

Hay que tener en cuenta que salvo una salida forzada, evaluar la expresión de iteración y luego evaluar la expresión condicional son siempre las dos últimas cosas que suceden en un bucle for.

El bucle for mejorado (para matrices)

El bucle for mejorado, es un bucle for especializado que simplifica el bucle a través de una matriz o una colección. En lugar de tener tres componentes, el bucle for mejorado solo tiene dos componentes, las cuales son:

* Declaración:

La variable de bloque recién declarada, de un tipo compatible con los elementos de la matriz a la que está accediendo. Esta variable estará disponible dentro del bloque for y su valor será el mismo que el del elemento de matriz actual.

* Expresión:

Esto debe evaluar la matriz que desea recorrer. Puede ser una variable de matriz o una llamada a un método que devuelve una matriz. La matriz puede ser de cualquier tipo: primitivas, objetos, incluso matrices de matrices.

Usando Break y Continue

Las palabras clave `break` y `continue` se utilizan para detener todo el bucle (`break`) o solo la iteración actual (`continue`). Por lo general, si se está usando `break` o `continue`, se hará una prueba `if` dentro del ciclo, y si alguna condición se vuelve verdadera (o falsa según el programador), querrá salir de inmediato. La diferencia entre ellos es si continúa o no con una nueva iteración o salta a la primera declaración debajo del ciclo y continúa desde allí.

La sentencia `break` hace que el programa detenga la ejecución del bucle más interno y comience a procesar la siguiente línea de código después del bloque.

La sentencia `continue` hace que cese solo la iteración actual del bucle más interno y que comience la siguiente iteración del mismo bucle si se cumple la condición del bucle. Cuando se utilice una sentencia `continue` con un bucle `for`, se debe de considerar los efectos que tiene `continue` en la iteración del bucle.

Declaraciones sin Etiquetas

Tanto la instrucción `break` como la instrucción `continue` pueden estar etiquetadas o sin etiquetar; aunque es mucho más común usar `break` y `continue` sin etiquetar. Una declaración de interrupción (sin etiquetar) saldrá de la construcción de bucle más interna y continuará con la siguiente línea de código más allá del bloque de bucle.

```
boolean problem = true;
while (true) {
    if (problem) {
```

```
System.out.println("There was a problem");  
break;  
}  
}
```

Declaraciones Etiquetadas

Aunque se pueden etiquetar muchas sentencias, es más común usar etiquetas con sentencias de bucle como for o while, junto con sentencias break y continue. Una declaración de etiqueta debe colocarse justo antes de la declaración que se está etiquetando y consiste en un identificador válido que termina con dos puntos (:).

Las variedades etiquetadas son necesarias solo en situaciones en las que tiene un bucle anidado y necesita indicar de cuál de los bucles anidados desea romper o cuál de los bucles anidados desea continuar con la siguiente iteración. Una declaración de ruptura saldrá del bucle etiquetado, a diferencia del bucle más interno, si la palabra clave de ruptura se combina con una etiqueta.

Detector una excepción usando try y catch

Antes que nada, el término "excepción" significa "condición excepcional" y es una ocurrencia que altera el flujo normal del programa.

Un montón de cosas pueden dar lugar a excepciones, incluidas fallas de hardware, agotamiento de recursos y viejos enoves. Cuando ocurre un evento excepcional en Java, se dice que se "lanza" una excepción. El código responsable de hacer algo sobre la excepción se denomina "controlador de excepciones" y "detecta" la excepción lanzada.

El manejo de excepciones funciona transfiriendo la ejecución de un programa a un controlador de excepciones apropiado cuando ocurre una excepción.

Try se usa para definir un bloque de código en el que pueden ocurrir excepciones. Este bloque de código se denomina región protegida (que en realidad significa "aquí va el código de riesgo"). Una o más cláusulas catch hacen coincidir una excepción específica (o un grupo de excepciones, más sobre eso más adelante) con un bloque de código que lo maneja.

Usando Finally

Aunque try y catch proporcionan un excelente mecanismo para capturar y manejar excepciones, aún queda el problema de cómo limpiar si llega a ocurrir una excepción.

Los controladores de excepciones son un lugar inadecuado para limpiar después del código en el bloque try porque cada controlador requiere su propio código de limpieza.

Un bloque finally incluye código que se ejecuta en algún momento después del bloque try, ya sea que se haya lanzado una excepción o no. Incluso si hay una instrucción de retorno en el bloque try, el bloque finalmente se ejecuta justo después de que se encuentra la instrucción de retorno y, antes de que se ejecute el retorno.

Si el bloque try se ejecuta sin excepciones, el bloque finally se ejecuta inmediatamente después de que se completa el bloque try. Si se lanza una excepción, el bloque finally se ejecuta inmediatamente después de que se completa el bloque de captura adecuado.