Labeled break and continue Statements

O

O.1 Introduction

In Chapter 5, we discussed Java's break and continue statements, which enable programmers to alter the flow of control in control statements. Java also provides the labeled break and continue statements for cases in which a programmer needs to conveniently alter the flow of control in nested control statements. This appendix demonstrates the labeled break and continue statements with examples using nested for statements.

O.2 Labeled break Statement

The break statement presented in Section 5.7 enables a program to break out of the while, for, do...while or switch in which the break statement appears. Sometimes these control statements are nested in other repetition statements. A program might need to exit the entire nested control statement in one operation, rather than wait for it to complete execution normally. To break out of such nested control statements, you can use the **labeled break statement**. This statement, when executed in a while, for, do...while or switch, causes immediate exit from that control statement and any number of enclosing statements. Program execution resumes with the first statement after the enclosing **labeled statement**. The statement that follows the label can be either a repetition statement or a block in which a repetition statement appears. Figure O.1 demonstrates the labeled break statement in a nested for statement.

The block (lines 7–26 in Fig. O.1) begins with a **label** (an identifier followed by a colon) at line 7; here we use the stop: label. The block is enclosed in braces (lines 8 and 26) and includes the nested for (lines 10–22) and the output statement at line 25. When the if at line 15 detects that row is equal to 5, the break statement at line 16 executes. This statement terminates both the for at lines 13–19 and its enclosing for at lines 10–22. Then the program proceeds immediately to the first statement after the labeled block—in this case, the end of main is reached and the program terminates. The outer for fully exe-

cutes its body only four times. The output statement at line 25 never executes, because it's in the labeled block's body, and the outer for never completes.

> **Good Programming Practice O.1**
> *Too many levels of nested control statements can make a program difficult to read. As a general rule, try to avoid using more than three levels of nesting.*

```java
1   // Fig. O.1: BreakLabelTest.java
2   // Labeled break statement exiting a nested for statement.
3   public class BreakLabelTest
4   {
5      public static void main( String[] args )
6      {
7         stop: // labeled block
8         {
9            // count 10 rows
10           for ( int row = 1; row <= 10; row++ )
11           {
12              // count 5 columns
13              for ( int column = 1; column <= 5 ; column++ )
14              {
15                 if ( row == 5 ) // if row is 5,
16                    break stop;  // jump to end of stop block
17
18                 System.out.print( "* " );
19              } // end inner for
20
21              System.out.println(); // outputs a newline
22           } // end outer for
23
24           // following line is skipped
25           System.out.println( "\nLoops terminated normally" );
26        } // end labeled block
27     } // end main
28  } // end class BreakLabelTest
```

```
* * * * *
* * * * *
* * * * *
* * * * *
```

Fig. O.1 | Labeled break statement exiting a nested for statement.

O.3 Labeled continue Statement

The continue statement presented in Section 5.7 proceeds with the next iteration (repetition) of the immediately enclosing while, for or do…while. The **labeled continue statement** skips the remaining statements in that statement's body and any number of enclosing repetition statements and proceeds with the next iteration of the enclosing **labeled repetition statement** (i.e., a for, while or do…while preceded by a label). In labeled while and do…while statements, the program evaluates the loop-continuation test of the labeled loop immediately after the continue statement executes. In a labeled for, the in-

crement expression is executed and the loop-continuation test is evaluated. Figure O.2 uses a labeled continue statement in a nested for to enable execution to continue with the next iteration of the outer for.

```java
 1   // Fig. O.2: ContinueLabelTest.java
 2   // Labeled continue statement terminating a nested for statement.
 3   public class ContinueLabelTest
 4   {
 5      public static void main( String[] args )
 6      {
 7         nextRow: // target label of continue statement
 8
 9            // count 5 rows
10            for ( int row = 1; row <= 5; row++ )
11            {
12               System.out.println(); // outputs a newline
13
14               // count 10 columns per row
15               for ( int column = 1; column <= 10; column++ )
16               {
17                  // if column greater than row, start next row
18                  if ( column > row )
19                     continue nextRow; // next iteration of labeled loop
20
21                  System.out.print( "* " );
22               } // end inner for
23            } // end outer for
24
25         System.out.println(); // outputs a newline
26      } // end main
27   } // end class ContinueLabelTest
```

```
*
* *
* * *
* * * *
* * * * *
```

Fig. O.2 | Labeled continue statement terminating a nested for statement.

The labeled for (lines 7–23) actually starts at the nextRow label. When the if at line 18 in the inner for (lines 15–22) detects that column is greater than row, the continue statement at line 19 executes, and program control continues with the increment of the control variable row of the outer for loop. Even though the inner for counts from 1 to 10, the number of * characters output on a row never exceeds the value of row, creating an interesting triangle pattern.