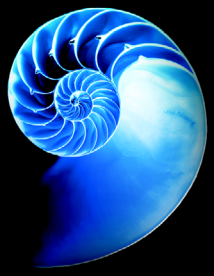

L

Labeled `break` and `continue` Statements

L.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.

L.2 Labeled `break` Statement

The `break` statement presented in Section 5.8.1 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 iteration 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 an iteration statement or a block in which an iteration statement appears. Figure L.1 demonstrates the labeled `break` statement in a nested `for` statement.

The block (lines 5–23 in Fig. L.1) begins with a **label** (an identifier followed by a colon) at line 5; here we use the `stop:` label. The block is enclosed in braces (lines 6 and 23) and includes the nested `for` (lines 8–19) and the output statement at line 22. When the `if` at line 11 detects that `row` is equal to `5`, the `break` statement at line 12 executes. This statement terminates both the `for` at lines 10–16 and its enclosing `for` at lines 8–19. 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 executes its body only four times. The output statement at line 22 never executes, because it's in the labeled block's body, and the outer `for` never completes.

Good Programming Practice L.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.

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

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

Fig. L.1 | Labeled `break` statement exiting a nested `for` statement.

L.3 Labeled `continue` Statement

The `continue` statement presented in Section 5.8.2 proceeds with the next iteration 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 iteration statements and proceeds with the next iteration of the enclosing **labeled iteration 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 increment expression is executed and the loop-continuation test is evaluated. Figure L.2 uses a labeled `continue` statement in a nested `for` to enable execution to continue with the next iteration of the outer `for`.

The labeled `for` (lines 5–19) starts at the `nextRow` label. When the `if` at line 13 in the inner `for` detects that `column` is greater than `row`, the `continue` statement at line 14 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.

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

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

Fig. L.2 | Labeled `continue` statement terminating a nested `for` statement.