Special Topic 6.4

The `break` and `continue` Statements

You already encountered the `break` statement in Special Topic 5.2, where it was used to exit a `switch` statement. In addition to breaking out of a `switch` statement, a `break` statement can also be used to exit a `while`, `for`, or `do` loop. For example, the `break` statement in the following loop terminates the loop when the end of input is reached.

```
while (true)
{
   String input = in.next();
   if (input.equalsIgnoreCase("Q"))
      break;
   double x = Double.parseDouble(input);
   data.add(x);
}
```

In general, a `break` is a very poor way of exiting a loop. In 1990, a misused `break` caused an AT&T 4ESS telephone switch to fail, and the failure propagated through the entire U.S. network, rendering it nearly unusable for about nine hours. A programmer had used a `break` to terminate an `if` statement. Unfortunately, `break` cannot be used with `if`, so the program execution broke out of the enclosing `switch` statement, skipping some variable initializations and running into chaos (*Expert C Programming*, Peter van der Linden, Prentice-Hall 1994, p.38). Using `break` statements also makes it difficult to use *correctness proof* techniques (see Special Topic 6.5 on page 255).

However, when faced with the bother of introducing a separate loop control variable, some programmers find that `break` statements are beneficial in the "loop and a half" case. This issue is often the topic of heated (and quite unproductive) debate. In this book, we won't use the `break` statement, and we leave it to you to decide whether you like to use it in your own programs.

In Java, there is a second form of the `break` statement that is used to break out of a nested statement. The statement `break` *label*; immediately jumps to the *end* of the statement that is tagged with a label. Any statement (including `if` and block statements) can be tagged with a label—the syntax is

label: *statement*

The labeled `break` statement was invented to break out of a set of nested loops.

```
outerloop:
while (outer loop condition)
{  . . .
   while (inner loop condition)
   {  . . .
      if (something really bad happened)
         break outerloop;
   }
}
Jumps here if something really bad happened
```

Naturally, this situation is quite rare. We recommend that you try to introduce additional methods instead of using complicated nested loops.

Finally, there is the `continue` statement, which jumps to the end of the *current iteration* of the loop. Here is a possible use for this statement:

```
while (!done)
{
   String input = in.next();
   if (input.equalsIgnoreCase("Q"))
   {
```

```
            done = true;
            continue; // Jump to the end of the loop body
         }
         double x = Double.parseDouble(input);
         data.add(x);
         // continue statement jumps here
      }
```

By using the `continue` statement, you don’t need to place the remainder of the loop code inside an `else` clause. This is a minor benefit. Few programmers use this statement.