Special Topic 6.3

The "Loop and a Half" Problem

Reading input data sometimes requires a loop such as the following, which is somewhat unsightly:

```
boolean done = false;
while (!done)
{
   String input = in.next();
   if (input.equalsIgnoreCase("Q"))
      done = true;
   else
   {
      Process data
   }
}
```

The true test for loop termination is in the middle of the loop, not at the top. This is called a "loop and a half", because one must go halfway into the loop before knowing whether one needs to terminate.

Some programmers dislike the introduction of an additional Boolean variable for loop control. Two Java language features can be used to alleviate the "loop and a half" problem. I don't think either is a superior solution, but both approaches are fairly common, so it is worth knowing about them when reading other people's code.

You can combine an assignment and a test in the loop condition:

```
while (!(input = in.next()).equalsIgnoreCase("Q"))
{
   Process data
}
```

The expression

```
(input = in.next()).equalsIgnoreCase("Q")
```

means, "First call `in.next()`, then assign the result to `input`, then test whether it equals `"Q"`". This is an expression with a side effect. The primary purpose of the expression is to serve as a test for the `while` loop, but it also does some work—namely, reading the input and storing it in the variable `input`. In general, it is a bad idea to use side effects, because they make a program hard to read and maintain. In this case, however, that practice is somewhat seductive, because it eliminates the control variable `done`, which also makes the code hard to read and maintain.

The other solution is to exit the loop from the middle, either by a `return` statement or by a `break` statement (see Special Topic 6.4 on page 246).

```
public void processInput(Scanner in)
{
   while (true)
   {
      String input = in.next();
      if (input.equalsIgnoreCase("Q"))
         return;
      Process data
   }
}
```