Busting Out of a Loop

Take a gander at the program in Listing 6-1. What's awkward about this program? Well, a few statements appear more than once in the program. Normally, a statement that's copied from one part of a program to another is no cause for concern. But in Listing 6-1, the overall strategy seems suspicious. You get a number from the user before the loop and (again) inside the loop.

```
out.print("Enter an int from 1 to 10: ");
int inputNumber = myScanner.nextInt();
numGuesses++;

while (inputNumber != randomNumber) {
    out.println();
    out.println("Try again...");
    out.print("Enter an int from 1 to 10: ");
    inputNumber = myScanner.nextInt();
    numGuesses++;
}
```

To be fair, I shouldn't badmouth this code. The code uses a standard trick for making loops work. It's called *priming* a loop. The pattern is

```
Get input
while the input you have isn't the last input
{
   Get more input
}
```

At the very start of the while loop, the computer checks a condition having to do with the user's input. So the computer doesn't enter the loop until the user gives some input. Then when the computer is inside the loop, the computer asks for more input to feed the loop's next iteration. The trick seems strange, but it works.

Professional programmers use this technique, priming a loop, all the time, so it can't be that bad. But there is another way, and that other way is illustrated in Listing A-1.

Listing A-1 A Break Statement in a Loop

```
import java.util.Scanner;
import java.util.Random;

public class BustingLoops {

    public static void main(String args[]) {
        int inputNumber, randomNumber, numGuesses = 0;
        Scanner myScanner = new Scanner(System.in);
        randomNumber = new Random().nextInt(10) + 1;
```

```
            System.out.println("       ************         ");
            System.out.println("Welcome to the Guessing Game");
            System.out.println("       ************         ");
            System.out.println();

            while (true) {
                System.out.print("Enter an int from 1 to 10: ");
                inputNumber = myScanner.nextInt();
                numGuesses++;
                if (inputNumber == randomNumber)
                    break;
                System.out.println();
                System.out.println("Try again...");
            }

            System.out.print("You win after ");
            System.out.println(numGuesses + " guesses.");
        }
    }
```

From the user's point of view, the code in Listing A-1 does exactly the same thing as the code in Listing 6-1. (To see the output of either program, refer to Figure 6-1.) The difference is, Listing 6-6 has only one call to `myScanner.netxtInt()`. That call is inside the loop, so the computer must enter the loop without testing any input.

If you look at the loop's condition, you can see how this works. The loop's condition is always true. (In Java, *true* is a keyword. It stands for a boolean value, which I cover in Chapter 4.) No matter what's going on, the loop's condition always passes its test. So the loop's condition is a big fraud. You never jump out of the loop by failing the test in the loop's condition. Instead, you jump out when you hit the break statement that's inside the loop (. . .and you hit that break statement when you get past the ~~if~~ `(inputNumber == randomNumber)` roadblock). The whole thing works very nicely.

Remember

When the computer executes a break statement that's in a loop, the computer jumps out of the loop (to the first statement that comes after the loop).

So why do programmers bother to prime their loops? Do break statements have any hidden drawbacks? Well, the answer depends on your point of view. Some programmers think that break statements in loops are confusing. All that jumping around makes them dizzy and reminds them of something from the 1960s called *spaghetti code*. (Spaghetti code uses *goto* statements to jump from one statement to another. In *Pascal by Example*, author B. Burd says "Programming with goto is like traveling around Paris by swimming through its sewer system. There are plenty of short cuts, but none of them are worth taking.") One way or another, break statements in loops are the exception, not the rule. Use them if you want, but don't expect to find many of them in other people's Java code.