THE WHILE STATEMENT IN JAVA

The `while` statement was already introduced in Section 3.1. A `while` loop has the form

```
while ( boolean-expression )
   statement
```

The **statement** can, of course, be a block statement consisting of several statements grouped together between a pair of braces. This statement is called the body of the loop. The body of the loop is repeated as long as the **boolean-expression** is true. This boolean expression is called the continuation condition, or more simply the test, of the loop. There are a few points that might need some clarification. What happens if the condition is false in the first place, before the body of the loop is executed even once? In that case, the body of the loop is never executed at all. The body of a while loop can be executed any number of times, including zero. What happens if the condition is true, but it becomes false somewhere in the **middle** of the loop body? Does the loop end as soon as this happens? It doesn't, because the computer continues executing the body of the loop until it gets to the end. Only then does it jump back to the beginning of the loop and test the condition, and only then can the loop end.

Let's look at a typical problem that can be solved using a `while` loop: finding the average of a set of positive integers entered by the user. The average is the sum of the integers, divided by the number of integers. The program will ask the user to enter one integer at a time. It will keep count of the number of integers entered, and it will keep a running total of all the numbers it has read so far. Here is a pseudocode algorithm for the program:

```
Let sum = 0     // The sum of the integers entered by the
user.
Let count = 0   // The number of integers entered by the
user.
while there are more integers to process:
    Read an integer
    Add it to the sum
    Count it
Divide sum by count to get the average
Print out the average
```

But how can we test whether there are more integers to process? A typical solution is to tell the user to type in zero after all the data have been entered. This will work because we are assuming that all the data are positive numbers, so zero is not a legal data value. The zero is not itself part of the data to be averaged. It's just there to mark the end of the real data. A data value used in this way is sometimes called asentinel value. So now the test in the while loop becomes "while the input integer is not zero". But there is another problem! The first time the test is evaluated, before the body of the loop has ever been executed, no integer has yet been read. There is no "input integer" yet, so testing whether the input integer is zero doesn't make sense. So, we have to do something **before** the while loop to make sure that the test makes sense. Setting things up so that the test in a `while` loop makes sense the first time it is executed is called priming the loop. In this case, we can simply read the first integer before the beginning of the loop. Here is a revised algorithm:

```
Let sum = 0
Let count = 0
Read an integer
while the integer is not zero:
    Add the integer to the sum
```

```
        Count it
        Read an integer
    Divide sum by count to get the average
    Print out the average
```

Notice that I've rearranged the body of the loop. Since an integer is read before the loop, the loop has to begin by processing that integer. At the end of the loop, the computer reads a new integer. The computer then jumps back to the beginning of the loop and tests the integer that it has just read. Note that when the computer finally reads the sentinel value, the loop ends before the sentinel value is processed. It is not added to the sum, and it is not counted. This is the way it's supposed to work. The sentinel is not part of the data. The original algorithm, even if it could have been made to work without priming, was incorrect since it would have summed and counted all the integers, including the sentinel. (Since the sentinel is zero, the sum would still be correct, but the count would be off by one. Such so-called off-by-one errors are very common. Counting turns out to be harder than it looks!)

We can easily turn the algorithm into a complete program. Note that the program cannot use the statement "`average = sum/count;`" to compute the average. Since `sum` and `count` are both variables of typeint, the value of `sum/count` is an integer. The average should be a real number. We've seen this problem before: we have to convert one of the int values to a double to force the computer to compute the quotient as a real number. This can be done by type-casting one of the variables to type double. The type cast "(double)sum" converts the value of `sum` to a real number, so in the program the average is computed as "`average = ((double)sum) / count;`". Another solution in this case would have been to declare `sum` to be a variable of type double in the first place.

One other issue is addressed by the program: If the user enters zero as the first input value, there are no data to process. We can test for this case by checking whether `count` is still equal to zero after the `while`loop. This might seem like a minor point, but a careful programmer should cover all the bases.

Here is the program and an applet that simulates it:

```
/**
 *  This program reads a sequence of positive integers
input
 * by the user, and it will print out the average of those
 * integers.  The user is prompted to enter one integer at
a
 * time.  The user must enter a 0 to mark the end of the
 * data.  (The zero is not counted as part of the data to
 * be averaged.)  The program does not check whether the
 * user's input is positive, so it will actually add up
 * both positive and negative input values.
 */

public class ComputeAverage {

   public static void main(String[] args) {

      int inputNumber;    // One of the integers input by
the user.
      int sum;               // The sum of the positive
integers.
      int count;              // The number of positive
integers.
      double average;     // The average of the positive
integers.
```

```
      /* Initialize the summation and counting variables.
*/

      sum = 0;
      count = 0;

      /* Read and process the user's input. */

      TextIO.put("Enter your first positive integer: ");
      inputNumber = TextIO.getlnInt();

      while (inputNumber != 0) {
         sum  +=  inputNumber;     //  Add  inputNumber  to
running sum.
         count++;                   // Count the input by
adding 1 to count.
         TextIO.put("Enter your next positive integer, or
0 to end: ");
         inputNumber = TextIO.getlnInt();
      }

      /* Display the result. */

      if (count == 0) {
         TextIO.putln("You didn't enter any data!");
      }
      else {
         average = ((double)sum) / count;
         TextIO.putln();
         TextIO.putln("You entered " + count + " positive
integers.");
         TextIO.putf("Their    average    is    %1.3f.\n",
average);
      }
```

```
    } // end main()

} // end class ComputeAverage
```

Source : http://math.hws.edu/javanotes/c3/s3.html