THE 3N+1 PROBLEM IN JAVA

Let's do another example, working this time with a program that you haven't already seen. The assignment here is an abstract mathematical problem that is one of my favorite programming exercises. This time, we'll start with a more complete specification of the task to be performed:

"Given a positive integer, N, define the '3N+1' sequence starting from N as follows: If N is an even number, then divide N by two; but if N is odd, then multiply N by 3 and add 1. Continue to generate numbers in this way until N becomes equal to 1. For example, starting from N = 3, which is odd, we multiply by 3 and add 1, giving N = 3*3+1 = 10. Then, since N is even, we divide by 2, giving N = 10/2 = 5. We continue in this way, stopping when we reach 1, giving the complete sequence: 3, 10, 5, 16, 8, 4, 2, 1.

"Write a program that will read a positive integer from the user and will print out the 3N+1 sequence starting from that integer. The program should also count and print out the number of terms in the sequence."

A general outline of the algorithm for the program we want is:

```
Get a positive integer N from the user.
Compute, print, and count each number in the sequence.
Output the number of terms.
```

The bulk of the program is in the second step. We'll need a loop, since we want to keep computing numbers until we get 1. To put this in terms appropriate for a `while` loop, we need to know when to **continue** the loop rather than when to stop it: We want to continue as long as the number is **not** 1. So, we can expand our pseudocode algorithm to:

```
Get a positive integer N from the user;
while N is not 1:
    Compute N = next term;
    Output N;
    Count this term;
Output the number of terms;
```

In order to compute the next term, the computer must take different actions depending on whether N is even or odd. We need an `if` statement to decide between the two cases:

```
Get a positive integer N from the user;
while N is not 1:
    if N is even:
       Compute N = N/2;
    else
       Compute N = 3 * N + 1;
    Output N;
    Count this term;
Output the number of terms;
```

We are almost there. The one problem that remains is counting. Counting means that you start with zero, and every time you have something to count, you add one. We need a variable to do the counting. (Again, this is a common pattern that you should expect to see over and over.) With the counter added, we get:

```
Get a positive integer N from the user;
Let counter = 0;
while N is not 1:
    if N is even:
       Compute N = N/2;
    else
       Compute N = 3 * N + 1;
    Output N;
    Add 1 to counter;
Output the counter;
```

We still have to worry about the very first step. How can we get a **positive** integer from the user? If we just read in a number, it's possible that the user might type in a negative number or zero. If you follow what happens when the value of N is negative or zero, you'll see that the program will go on forever, since the value of N will never become equal to 1. This is bad. In this case, the problem is probably no big deal, but in general you should try to write programs that are foolproof. One way to fix this is to keep reading in numbers until the user types in a positive number:

```
Ask user to input a positive number;
Let N be the user's response;
while N is not positive:
   Print an error message;
   Read another value for N;
Let counter = 0;
while N is not 1:
    if N is even:
       Compute N = N/2;
    else
       Compute N = 3 * N + 1;
    Output N;
    Add 1 to counter;
Output the counter;
```

The first `while` loop will end only when N is a positive number, as required. (A common beginning programmer's error is to use an `if` statement instead of a `while` statement here: "If N is not positive, ask the user to input another value." The problem arises if the second number input by the user is also non-positive. The `if` statement is only executed once, so the second input number is never tested, and the program proceeds into an infinite loop. With the `while` loop, after the second number is input, the computer jumps back to the beginning of the loop and tests whether the second number is positive. If not, it asks the user for a third number, and it will continue asking for numbers until the user enters an acceptable input.)

Here is a Java program implementing this algorithm. It uses the operators <= to mean "is less than or equal to" and != to mean "is not equal to." To test whether N is even, it uses "N % 2 == 0". All the operators used here were discussed in Section 2.5.

```
/**
 * This program prints out a 3N+1 sequence starting from a
positive
 * integer specified by the user.  It also counts the number of
 * terms in the sequence, and prints out that number.
 */
 public class ThreeN1 {

      public static void main(String[] args) {

         int N;       // for computing terms in the sequence
         int counter; // for counting the terms

         TextIO.put("Starting point for sequence: ");
         N = TextIO.getlnInt();
         while (N <= 0) {
            TextIO.put("The  starting  point  must  be  positive.
Please try again: ");
            N = TextIO.getlnInt();
         }
         // At this point, we know that N > 0

         counter = 0;
         while (N != 1) {
             if (N % 2 == 0)
                N = N / 2;
             else
                N = 3 * N + 1;
             TextIO.putln(N);
             counter = counter + 1;
         }

         TextIO.putln();
```

```
            TextIO.put("There were ");
            TextIO.put(counter);
            TextIO.putln(" terms in the sequence.");

        }  // end of main()

    }  // end of class ThreeN1
```

Two final notes on this program: First, you might have noticed that the first term of the sequence -- the value of N input by the user -- is not printed or counted by this program. Is this an error? It's hard to say. Was the specification of the program careful enough to decide? This is the type of thing that might send you back to the boss/professor for clarification. The problem (if it is one!) can be fixed easily enough. Just replace the line "counter = 0" before the while loop with the two lines:

```
TextIO.putln(N);   // print out initial term
counter = 1;       // and count it
```

Second, there is the question of why this problem is at all interesting. Well, it's interesting to mathematicians and computer scientists because of a simple question about the problem that they haven't been able to answer: Will the process of computing the 3N+1 sequence finish after a finite number of steps for all possible starting values of N? Although individual sequences are easy to compute, no one has been able to answer the general question. To put this another way, no one knows whether the process of computing 3N+1 sequences can properly be called an algorithm, since an algorithm is required to terminate after a finite number of steps! (This discussion assumes that the value of `N` can take on arbitrarily large integer values, which is not true for a variable of type int in a Java program. When the value of `N` in the program becomes too large to be represented as a 32-bit int, the values output by the program are no longer mathematically correct.

Source : http://math.hws.edu/javanotes/c3/s2.html