CS-162 Lab #8: Recursion

General

At the top of the every document that you create (word processing files or source files) include:

```
/**
 * Description of the class or document.
 *
 * @author YOUR NAME
 * @version CS162 Lab #, mm/dd/yyyy (replace with the last edit date)
*/
```

Submit in this lab via Moodle using the “Lab #” link. Your assignment must be uploaded by the assigned due date and time. Moodle will automatically close the link at that time. It is strongly highly recommend you do not wait until the very last minute to submit your work.

Concepts

This lab focuses on gaining some experience with programming and tracing recursive code in Java. You will also continue your work with exception handling and file I/O.

Background

1) Review the on line tutorials on Recursion (#70 - #74).

2) A **prime number** is an integer that cannot be divided by any integer other than one and itself. For example, 7 is a prime number because its only divisors are 1 and 7. The integer 8 is not a prime number because its divisors are 1, 2, 4, and 8.

   A common way to define prime testing is (for positive integers):

   ```
   prime(1)    = false
   prime(N)    = prime(N, N-1) //for N > 1
   prime(N, 1) = true
   prime(N, D) = if D divides N, false
                 else prime(N, D-1)
   ```

   For example,

   ```
   prime(4)   = prime(4,3)
   prime(4,3) = prime(4,2)
   prime(4,2) = false
   ```

   Another example,

   ```
   prime(7)   = prime(7,6)
   prime(7,6) = prime(7,5)
   prime(7,5) = prime(7,4)
   prime(7,4) = prime(7,3)
   prime(7,3) = prime(7,2)
   prime(7,1) = true
   ```

Assignment

Create a new project and a class called “PrimeFinder”. Translate the above definition of prime number testing into two Java methods that return a boolean using the signatures shown below. Use the **% (modulus)** operator to test divisibility. Create a second class with a “main” method checking a series of numbers using a for loop to test your program. (Look at FactorialTester.java in the tutorial chapters.)

Your prime testing methods must have these signatures:

```
public boolean prime( int num);

private boolean primeHelper( int num, int testDivisor);
```

Try to run your program to test integers larger than about 15,000; you will likely run out of memory (if not, try a slightly larger number). Your program will stop running and report a *StackOverflowError*. This is because each method activation in the activation chain requires memory on the run-time stack, and around 15,000 activations or so uses up all the memory that has been reserved for this use. Think about why this is so..

Lab Requirements:

1) Write a complete Java program (including a “main”) that implements the recursive “prime” method described above. You are to use two methods to accomplish this; one with a single integer parameter, and one with 2 integer parameters. The method with 2 parameters is commonly called a “helper” method and should be private. The single parameter method simply calls the private, RECURSIVE method with N and N-1.
2) Write a “betterPrime(int N)” method that uses the same helper method from the 1st problem. However, there are better possible values to use for the second parameter than N-1. Think about this and come up with something to algorithmically generate the second parameter that will make this more efficient than the implementation from problem #1.
3) Write a method “generatePrimes()” that uses the betterPrime(N) method in a loop to generate all of the prime numbers between 1 and 1000. The method must print these prime numbers to an output file (review Chapter #12 sections on file I/O). You must also write the exception handling code around the file I/O calls to trap problems that might happen when creating a file.

Turn in your entire BlueJ project folder electronically into the CS-162 InBox Lab3 folder (FTP site) by midnight of the due date.