**Assignment 8 Turn in Sheet Name:**

Lab Questions (**Total 50 Pts. + 25 Extra**)

Big Java, Late Objects / Java for Everyone, 2e

**Course Topics Wrap Up**

The exercise is about to create a multithreaded application that implements the producer – consumer pattern. It is a good preparation for the upcoming final exam.

Background: A prime integer is a positive number that can be divided without remainder by exactly 2 numbers: by itself and by 1. 1 is not a prime number because it can only be divided by one number: 1. So the first prime is 2. And this is the only even number that is a prime. Every other even number 4,6,8,10,… is not a prime because it can be divided by 1, itself and 2.

We want to create a list of as many prime numbers as possible. The bigger the numbers get the longer it takes to check if a (odd) number is a prime or not. You will use BigInteger Class wherever applicable (please refer to Java API for BigInteger Class).

The producer in our system is a thread that creates a list of prime candidates (basically of odd numbers).

The consumer(s) in our system take one prime candidate from the list at one time and test if it is a prime or not. **DO NOT USE the BigInteger methods to check for primes, only use the methods below:**

1. Create a Class that provides the following algorithms as static methods:
2. (5 pts.)   
   public static BigInteger EuclidGCD(BigInteger a, BigInteger b);  
   You have programmed this in Assignment 3 Exercise 2. Rewrite it that is uses BigInteger instead of int. It must not use division, you only use subtraction like in the original exercise!
3. (5 pts.)   
   public static Boolean isPrime(BigInteger p);  
   Given a positive number (integer) p > 2, p is NOT a prime if it can be divided by any prime q, where 2<= q <= sqrt(p). The sqrt() function uses a lot of computing power, and we do not need to have the exact value as you can imagine, it is sufficient to test all prime r’s, where 4 <= r2 <= p.  
   The test if p can be divided by r can be done by checking if EuclidGCD(p,r) !=1. So you need to call EuclidGCD(p,r) until it either is !=1 or until for all r’s, where 4 <= r2 <= p EuclidGCD(p,r)==1 (Then you found a prime).
4. (5 pts.)   
   public static Boolean isDividableBy3(BigInteger p);  
   To test you can divide a number p >3 by 3 (meaning it is not a prime) is relatively simple. You add all digits and if the sum of the digits is dividable by 3 then the original number is too. If the sum has several digits you add them again and if the result is dividable by 3 then the original number is and so forth. So you can do this algorithm until your sum consists of just one digit. If this (the final sum) digit is 3, 6 or 9 the original number was dividable by 3.  
   Example: 39 => 3+9 =12 (dividable by 3) do it again => 1+2 = 3   
   123 => 6 (Div by 3)  
   867 => 8+6+7 = 21 => 2+1=3 (Div by 3)

868 => 8+6+8=22 => 4 (Not div by 3)  
Hint: Use the toString method of BigInteger along with Character methods to calculate the sum of the digits.

1. (10 pts.) Create a Runnable (this is the “generator”) that runs the following loop: It takes uneven numbers and adds it to the candidates ArrayList if they are not dividable by 3. The candidates ArrayList needs to carry the following information: BigInteger candidate and Date timeCreated (which is the time that the candidate is added to the ArrayList). TAKE CARE OF possible race conditions and deadlocks!
2. (10 pts.) Create a Runnable (this is the “tester”)that runs the following loop: It takes and removes the first candidate of the candidate ArrayList and tests it for being a prime. If it is a prime the square of that prime will be calculated too and it will be added along with other information to the prime ArrayList: BigInteger prime, BigInteger primeSquare, Date timeCandidateCreated (from the candidate list), Date timeCandidateFetched, Date timePrimeFound. TAKE CARE OF possible race conditions and deadlocks!
3. (20 pts.) The main thread in your program does the following:
4. It initializes the candidates ArrayList
5. It initializes the primes ArrayList
6. It tries to open and read the file prime\_numbers.dat and fills the primes ArrayList with the records found in that file. If that file does not exist the first element will be set to the BigInteger with the value 2 and the square of it (4) and the Date records set to the current time. The file will be created at that time if it did not exist yet.
7. It starts one thread that runs the generator. This thread needs to pick up its task behind the end of the primes ArrayList. (So it does not candidates that have already been tested).
8. It starts at least(!) 2 threads that run the tester.
9. Update the fields in the GUI every 5 seconds (configurable). The Last fields should show the toString of the BigInteger the threads worked on. The In Queue should show the number of candidates that are in the candidates list (the number of candidates that have been created but not tested yet). The In Work should show the the toString of the BigInteger the thread is working on right now.
10. Update and write the file prime\_numbers.dat to disk every minute (configurable)
11. TAKE CARE OF possible race conditions and deadlocks!
12. (20 pts.) Create the GUI below and implement the following functionality:
13. When the “Pause” Button is pressed it changes its text to “Run” and the thread is put to sleep (or whatever you decide to be able to wake it up and pickup its task when the now “Run” button is pressed).
14. When the “Terminate” button is pressed, the thread will be terminated, DO NOT USE the depreciated stop method. After the Terminate button is pressed the text “Terminate” will become gray (grayed out) and any further pressing of that button will not have any effect.

