Problem Set 4

(8 points) Cohort Exercise 1:

Write a program to multiply two square matrices A and B using two threads (in addition to the main thread).

(8 points) Cohort Exercise 2:

Continue with cohort exercise 2; refactor your matrix multiplication programs to be based on Runnable.

(8 points) Cohort Exercise 3:

Modify your matrix multiplication program in cohort exercise 2 so that it uses 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 threads and measure the performance with a square matrix with 1,2,…,1000 rows.

(8 points) Cohort Exercise 5:

Write a program to search for an element is in an integer array using two threads. Hint: You can use a static variable to allow communication between searching threads and the main thread; and use interrupt() to stop a thread when the element is found.

(8 points) Cohort Exercise 6:

Write a multi-threaded program to factor semi-prime. Stop all threads as soon as the problem is solved.

(20 points) Homework Question 1:

Compile and run BadThreads.java:

```
1. public class BadThreads {
2.
3.     static String message;
4.
5.     private static class CorrectorThread
6.         extends Thread {
7.
8.         public void run() {
9.             try {
10.                     sleep(1000);
11.                } catch (InterruptedException e) {}
12.                 // Key statement 1:
13.                 message = "Mares do eat oats.";
14.            }
15.         }
16.
```

```
17.         public static void main(String args[])
18.              throws InterruptedException {
19.
20.              (new CorrectorThread()).start();
21.              message = "Mares do not eat oats.";
22.              Thread.sleep(2000);
23.              // Key statement 2:
24.              System.out.println(message);
25.         }
       }
```

The application should print out "Mares do eat oats." Is it guaranteed to always do this? If not, why not? Would it help to change the parameters of the two invocations of Sleep? How would you guarantee that all changes to message will be visible in the main thread?

(20 points) Homework Question 2:

Given a semi-prime, use multiple computers (processes) to find the factors. Stop all computers as soon as the problem is solved. Hint: on the client-side, use a separate thread to do the actual factoring and interrupt the thread when necessary.

(20 points) Homework Question 3:

Write a multi-threaded FTP server such that a new thread is used to handle a new client each time. That is, at the client side, the client transfers a text file (one line at a time) to the server and expects an acknowledgment from the server. Only after receiving an acknowledgment from the server, the client transmits the next line. If the acknowledgment is not received within a timeout period (choose your own value depending on your network delay), the client retransmits the unit. The above process is repeated until all the contents of the file are transferred. Multiple clients should be able to connect and transfer at the same time.