Lab 8: Threads

Objectives

- to gain experience using threads in a program.
- to understand competitive and cooperative synchronization.

You can run *getcs* to get a minimal template for this assignment, called `PrimeCount.java`.

Write a Java program that checks to see whether each number is prime and displays, once per second, how many primes it has found so far to `System.out`. The user should be able to tell the program to pause, continue, or quit the program by typing commands in the console (which the program will read via `System.in`).

For example, one run of the program might proceed thus.

```
55911 primes <= 691530
88425 primes <= 1137456
115690 primes <= 1521570
pause
continue
133725 primes <= 1780613
153258 primes <= 2062705
172066 primes <= 2337820
quit
181126 primes <= 2471207
```

The first line, for example, which is displayed one second after the computation begins, says that the program found 55,911 prime numbers less than or equal to 691,530.

You should accomplish this task using three threads: One to perform the computation, one to display the current values each second, and another to respond to commands from the user. When paused, your program should use cooperative synchronization so that the computation and displaying threads will take no CPU time during the pause but will both immediately restart once the user says to continue.

You should pay careful attention to ensure that the results displayed are always correct. In particular, if you have a variable representing how many numbers have been tested, and another representing how many primes have been found, you should prevent the displaying thread from displaying its information between incrementing one variable and incrementing the other.

When the user types “quit,” the program should complete its computation on the current number before displaying the final result.

Run *handincs* to submit your solutions electronically. Your lab report’s body need contain only your Java code, commented appropriately.