**Documentation for programming assignment #1**

**Primes.java:**

First off, in the main thread, we ignore all even numbers except 2 thru out the entire program.The 8 threads start at a unique prime number under 100 and will expand in increments of 16 (because we try to avoid as much thread value overlaps as possible) up to 10,000. 10,000 is the upper bound for these threads because it’s the square root of our original goal of 100,000, meaning that once we sieve the largest prime number under 10,000, its multiples will extend out to 100,000 (at least near it). Knowing this, we continue our operations by looking at one thread at a time and sieving out (mark as false in bool array) the multiples of each number one by one up to 10,000. Once all threads have been processed, we avoid bottlenecking by going through only the odd numbers from 3 to 100,000 and check for primality based on the Boolean array created earlier, given that the number, 2, is already recorded down as prime in our list.

**Versions 1 thru 4:**

Given that the number of philosophers and the number of chopsticks are the same, we assign each philosopher and chopstick an id so that they can be differentiated thru out the process. For version1 we synchronized each chopstick method of picking up and putting down the chopstick so that only one philosopher can access the bool value at a time, but since we didn’t manage the state of the philosopher, there was no guard in the run method for whether or not a philosopher should pick up the chopstick, so a deadlock would almost always occur for every execution. In the slightly enhanced version2, we started to manage the states of when each philosopher should eat. This guarantees that no deadlock will occur but there is a possibility that at least one philosopher will starve because there’s no fairness imposed by our thread scheduler. However, version3 fixes this by using a reentrant lock, and we set the parameter upon instantiation of the lock to true (i.e. new ReentrantLock(true)), so that the thread scheduler can give priority to threads that have waited longer. In the final version4, we pass an extra piece of data, the number of philosophers down from the main thread to our philosopher class which will provide us with the ability to handle any number of philosophers given that the number of chopsticks are the same.