**Homework 2**

As part of this assignment, you will have to write a multi-threaded program that takes as input a text file and computes the distribution of letters (after being converted to lower case) in the input file. Your program must be implemented in Java, but you are limited to using only the standard libraries included with the SDK. You should consider three different approaches to solve the problem, as described below. Note that the number of child threads N should be a parameter in all your solutions:

* Using Monitors and Condition Variables: In the main thread, load the text file into memory. Give each of the N child threads a chunk of size 1/N of the data to process. Each child process will produce a local distribution of letters. As children complete, they will add their distributions to compute the distribution over the entire file.
* Using Monitors and Condition Variables and Atomics: In the main thread, load the text file into memory. Give each of the N child threads a chunk of size 1/N of the data to process. For each letter, its frequency is stored as an atomic variable (<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html>) . When a child considers a letter, it will increment the associated atomic variable.
* Using a Producer-Consumer Paradigm: In the main thread, load R bytes from the text file until reaching the end-of-the file. Each time R bytes are read, the program should generate a work request to process those bytes using one of the N child consumer threads.

Turn in your implementation for each of the three versions of the problem. Additionally, generate a graph that includes three lines (one per solution) showing how each solution increases with the number of threads. Start with N=1 and increase it until there is no further benefit to adding more threads.

Based on the above information and your experience implementing each solution, answer the following questions:

1. Is there one implementation that is a clear winner from a performance perspective? If so, why do you think that is the case.
2. Is there one implementation that was easier to implement than another? Explain what made it easier to implement.
3. How confident are you that your implementations are correct? Please rate each one of the solutions. Describes what steps you took to make sure that your implementations were correct.