Concurrent Programming
COMP 409, Winter 2020

Assignment 2

Due date: Thursday, February 27, 2020
7pm

General Requirements

These instructions require you use Java. All code should be well-commented, in a professional style, with appropriate variables names, indenting, etc. Your code must be clear and readable. **Marks will be very generously deducted for bad style or lack of clarity.**

There must be no data races: all shared variable access must be properly protected by synchronization. Any variable that is written by one thread and read or written by another should only be accessed within a synchronized block (protected by the same lock), or marked as volatile. At the same time, avoid unnecessary use of synchronization or use of volatile. Unless otherwise specified, your programs should aim to be efficient, and exhibit high parallelism, maximizing the ability of threads to execute concurrently. Please stick closely to the described input and output formats.

Your assignment submission **must** include a separate text file, `declaration.txt` stating "This assignment solution represents my own efforts, and was designed and written entirely by me". Assignments without this declaration, or for which this assertion is found to be untrue are not accepted. In the latter case it will also be referred to the academic integrity office.

Questions

1. These two sub-questions simulate spiders(-ish) jumping around a web(-like thing). You should only use blocking sychronization—once a lock acquisition is attempted, it must complete. Do not use `volatile` or spin.

   (a) The web is formed of some number of edges between points in a 2D space. **12**

   Provide a program, `q1a.java` that accepts command-line arguments, n, t, and k. First, sequentially create n random points. Use floating point coordinates (floats or doubles are ok), and strictly bound the region in which these points are created to the 1000.0×1000.0 square. Include the corners of the outer square in the point set. Ensure no two points overlap (note that since you are using floating point numbers this means the distance between them should be greater than some small, but non-0 epsilon value).

   Each point should be represented by an object containing the coordinates, and a normal `java.util.ArrayList` for storing adjacent points. Access to the ArrayList should be protected by a lock, (or synchronized—using blocking synchronization with no spinning).

   Now, launch n threads to triangulate the region. Each thread picks a randomly selected pair of points and tries to add an edge between them (represented by adding an adjacency for each point in the other), if there is not one already. This process must be done atomically, in the sense that while you attempt to add an edge a thread requires exclusive access to each point's ArrayList, which it acquires by using the blocking synchronization protecting each such list.

   The program should terminates as soon as any one thread has failed to add a total of k edges. Print out the total number of edges that were successfully added.

   (b) On your web/graph, spiders jump around. To simplify things, spiders here have just 3 legs, and a **12**

body. The body occupies a point in the graph, and the three legs occupy 3 different points that are adjacent to the body. Each point in the graph can only hold a spider body or one leg.

Provide a program, `q2a.java` that accepts command-line arguments, n, t, k, and m. The first three parameters apply to your solution to the question above.

After the web has been constructed by your t threads, t spiders should be instantiated at random locations in the graph, non-overlapping. Your object structure for points will need to store which spider (and spider-part) it currently holds. Each spider is controlled by a separate thread.

Spiders sleep for a time, and then jump around to a different, randomly selected location (i.e., 4 points that hold the body and its legs) on the web. A spider must do so (effectively) atomically, however, verifying that the new random locations are not otherwise occupied before any part of it moves from its current location. Associate a lock (using blocking synchronization) with each point in order to protect the state of each point (you can reuse the locks protecting your ArrayList from the previous question, or use new ones). A spider jumping must then acquire appropriate locks to ensure its destination and departure points are not modified by other spiders, verify it can reach its destination and if so move, or not move if the destination is not suitable.

Spiders attempt to move every 40–50ms, sleeping between jumps. The simulation should run for m seconds, and then all threads should terminate. As output, after threads terminate emit the number of moves each spider has made.

Provide your source code for both questions, and as a *separate document* include a brief textual explanation of why your programs do not deadlock (for reasonable choices of input parameters).

2. Javas synchronized construct allows for trivial construction of monitors based on SIGNAL_AND_CONTINUE semantics. Define monitors to allow a choice of semantics. Define an abstract class `Monitor` which declares `enter()` and `exit()` methods, as well as `await() throws InterruptedException` and `signal()` for managing condition semantics (only one condition queue is required). Then define subclasses `MonitorSC`, `MonitorSW`, and `MonitorSUW` which provide SIGNAL_AND_CONTINUE, SIGNAL_AND_WAIT, and SIGNAL_AND_URGENT_WAIT semantics, respectively. **12**

What to hand in

Submit your declaration and assignment files to *MyCourses*. Note that clock accuracy varies, and late assignments will not be accepted without a medical note: **do not wait until the last minute**. Assignments must be submitted on the due date **before 6pm**.

Where possible hand in only **source code** files containing code you write. Do not submit compiled binaries or .class files. For any written answer questions, submit either an ASCII text document or a .pdf file. Avoid .doc or .docx files. Images (plots or scans) are acceptable in all common graphic file formats.

This assignment is worth 10% of your final grade. **36**