Concurrent Programming

COMP 409, Winter 2025

Assignment 2

Due date: Tuesday, February 18, 2025
Midnight (23:59:59)

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.**

Blocking synchronization only. In this assignment all problems must be solved with *blocking* synchronization—threads must not spin when acquiring locks, and any attempt at lock acquisition should either succeed or imply going to sleep until woken by another thread's actions. Note that dealing with spurious wakeups or necessarily broad `notify(All)`'s when waiting on condition variables is not considered spinning, and despite their optimized hybrid design you can consider Java's *synchronized* to be blocking.

There must be no data races. This means all shared variable access must be properly protected by synchronization: any memory location 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. Solving a word search puzzle involves searching a grid of letters to find actual words. Many game designs exist, with the most flexible allowing words to be formed by any non-repeating consecutive sequence of letters, with each letter adjacent to the previous by one of the 8 grid directions. **12**

   First, based on an input random seed value, create a grid by initializing an $n \times n$ grid with random letters. To improve the likelihood of words, choose letters according to the letter frequences in English shown in the `freq.txt` file (which shows counts for each letter out of 100000). Once the grid is created output your grid, one line of letters per row, no spaces.

   Now, launch t threads to conduct a random search for words. Each thread picks a random starting cell, and pre-selects a sequence of up to 7 random moves by incrementally creating a plan of moves to make. Move choices are random, but the sequence must avoid visiting the same cell more than once. This may limit the feasible length of the sequence, although a 2-move (3-letter) sequence is always possible ($n > 1$).

   Once a candidate sequence is identified, the thread checks whether a 3-letter, 4-letter, etc. word is formed by starting at the first cell and following the move sequence. Each potential word should be verified to exist in the provided dictionary (`dict.txt`). If so, it adds the word to a list associated with each cell in the sequence, if it is not already there. Once finished processing the sequence, the thread sleeps for 20 ms before picking a new starting cell.

To avoid conflicts, each cell must be associated with a separate lock, using blocking synchronization. A thread must acquire and own the individual locks for every cell in its movement sequence before/while checking for any valid words and while updating the associated cell→word lists.

Let the simulation run until each thread has tested k starting cells. Once done, (sequentially) iterate through all cells, and for each emit a line of text, consisting of the coordinate (as 0-based x, y values) a space, and a space-separated list of words to which that cell/letter contributed.

Your program, `q1.java`, should accept integer parameters s, $n > 4$, t, and k in that order (s is a random (integer) seed value). Include a sample of the resulting output $n = 5$, $t = 3$ and a large k with your submission, along with a text file `q1.txt` which briefly explains how you guarantee your program does not deadlock.

2. Professor P currently has $k > 3$ TAs at the university, and also 5 new international grad students who are each making their own plans to eventually arrive at the university. P's normal behaviour is to sleep in their office, only waking when TAs have questions. To allow P to sleep for longer periods, P will only wake and answer questions from a group of exactly 3 TAs—if fewer than 3 TAs have questions they must wait until more do, and while 3 TAs are talking to P other TAs with questions must wait. **8**

   As the grad students arrive they head off to sleep in the lab. If P wakes for questions and all grad students have finally returned, the last returning grad student has the task of interrupting the TA question session to fetch the professor, who immediately abandons any ongoing TA questions and goes to the lab to wake their grad students and begin research. The simulation then terminates.

   Modelling each of P, the 5 grad students, and k TAs as separate threads, build a simulation of this scenario based on *monitors,* and using multiple condition variables. Each TA has a $q\%$ chance each second to come up with a question, and it takes 0.5s to address a set of TA questions. Each new grad student arrives at a random time, within 10–60s.

   Your program, `q2.java`, should accept integer parameters $k > 3$ and $0 < q < 100$, and should output a short but descriptive line of text every time a significant event happens and which TAs or grad students are involved:

   (a) a TA comes up with a question,
   (b) a group of TAs starts to be seen by P,
   (c) a group of TAs questions have been answered,
   (d) P goes to sleep,
   (e) P wakes,
   (f) a grad student arrives,
   (g) a grad student interrupts a TA session,
   (h) P wakes their grad students
   (i) all grad students have been woken

   Include a sample of the resulting output for $k = 9$, and $q = 10$ with your submission. How would your design change if Java had `SIGNAL_AND_WAIT` semantics? Include a separate file `q2.txt` or `q2.pdf` which describes what would need to be changed in your existing solution to make a correct solution under the different semantics (but you do not need to actually code the change).

What to hand in

Submit your declaration and `q1.java`, `q1.txt`, `q2.java`, `q2.txt` files to *MyCourses*. Note that clock accuracy varies, and late assignments will not be accepted without a special permission: **do not wait until the last minute**. Assignments must be submitted on the due date **before midnight**.

Where possible hand in only **source code** files containing code you write. Do not submit compiled binaries or .class files, but do include a `readme.txt` of how to execute your program if it is not trivial and obvious. 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. $\overline{\mathbf{20}}$

Programming assessment

	Mastery	Proficient	Developing	Beginning
Correctness	The solution works correctly on all inputs and meets all specifications.	The solution meets most of the specifications; minor errors exist.	The solution is incorrect in many instances.	The solution does not run or is mostly incorrect.
Readability	Well organized according to course expectations and easy to follow without additional context.	Mostly organized according to course expectations and easy to follow for someone with context.	Readable only by someone who knows what it is supposed to be doing.	Poorly organized and very difficult to read.
Algorithm Design	The choice of algorithms, data structures, or implementation techniques is very appropriate to the problem.	The choice of algorithms, data structures, or implementation techniques is mostly appropriate to the problem.	The choice of algorithms, data structures, or implementation techniques is mostly inappropriate to the problem.	Fails to present a coherent algorithm or solution.
Documentation	The solution is well documented according to course expectations.	The solution is well documented according to course expectations.	The solution documentation lacks relevancy or disagrees with course expectations.	The solution lacks documentation.
Performance	The solution meets all performance expectations	The solution meets most performance expectations	The solution meets few performance expectations	The solution is not performant.