COS226 - Concurrent Systems Assignment 2 Specification

Release Date: 22/09/2025

Due Date: 19/10/2025 at 23:59

Total Marks: 100

1 General Instructions

- Read the entire assignment thoroughly before you start coding.
- This assignment should be completed individually; no group effort is allowed.
- To prevent plagiarism, every submission will be inspected with the help of dedicated software.
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experiences a runtime error, you will be awarded a zero mark. Runtime errors are considered unsafe programming.
- Ensure your code compiles with Java 8
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent), and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). If you have any form of question regarding this, please ask one of the lecturers to avoid any misunderstanding. Also note that the OOP principle of code reuse does not mean that you should copy and adapt code to suit your solution.

3 Overview

In this assignment, you will design and implement three classic concurrency problems using multi-threading. The focus is on correctness, synchronization, and demonstrating understanding of concurrency control.

4 Tasks

Task 1: Barrier Synchronization

You and four friends are making a pizza together. You make the dough, Jen kneads the dough, Steve spreads the sauce, Mark adds the cheese, and Jessica adds toppings. You must each finish your part before the pizza can be put in the oven. Only once everyone is ready can you put the pizza into the oven. Then this can repeat for multiple pizzas.

Implement a reusable barrier for n friends (threads) where all friends must finish their task before any can proceed. The barrier should be reusable across multiple pizzas (i.e., cyclic). Ensure through testing in your Main.java that a program where n friends "make a pizza", synchronize at the barrier, and then continue.

You may use any lock if it ensures mutual exclusion and ensures exclusive access to the barrier state (i.e. baking the pizza) and a way to wait using the await() function.

As a test you can use the () -> System.out.println("All friends finished! Time to bake the pizza!\n") for the task variable so that once all threads have finished the task, this should print.

In your report explain which lock you chose and why

Task 2: Concurrent Data Structure

You and your friends open a pizzeria and need to decide on how many cashiers to hire.
You want the cashiers to ensure fairness at the counter where pizzas are served. Your options are:

1. Hire one cashier to manage the counter.
   They must ensure that whenever a waiter wants to place an order, then **no one** can have access to the counter until the order is placed. If a customer wants to get a dish, then they must wait for the cashier to finish monitoring the waiter before they can show the cashier their receipt and collect their food.

2. Hire two cashiers for each side of the counter.
   When the waiter wants to place a dish then they need to speak to the cashier on the kitchen side to let them place it. Similarly, if a customer wants to pick up their order they need to speak to the second cashier to be allowed to pick it up. Since they can talk to different cashiers, the waiter can place a dish at the same time a customer takes one.

To get an idea of which approach is better you decide to implement a thread-safe bounded blocking queue with the following operations:

- enqueue(item) — add a pizza to the counter (wait if full).
- dequeue() — picking up a pizza (wait if empty).

A coarse-grained lock and fine-grained lock must be implemented. The pizzas can be represented as objects, and you must decide on which lock you use. The lock must ensure mutual exclusion. Your program must support multiple threads and ensure no race conditions or deadlocks.

In your report describe the following:

- which lock was chosen and why
- compare coarse and fine-grained implementations
- explain which option would be better for the pizzeria and why in terms of the pizzeria analogy

Task 3: Producer–Consumer

Your pizzeria is doing very well and you are trying to decide whether a delivery service would be a feasible option. You decide to implement a producer-consumer scenario. With the producer being the pizzeria and the consumer being the delivery driver that will then deliver the pizzas. The system must be able to use multiple producer threads (generate pizzas) and multiple consumer threads (delivers pizzas).

Producers and consumers should generate items at random intervals. The producer should have a queue where the queue is the list of pizza orders and the consumer should also have a queue where they wait and take pizzas out of the queue as they become available. To stop the drivers from waiting forever, a “poison pill” is given to them so that they know that they can stop driving back to the pizzeria. The “poison pill” is represented as a -1 value.

In your report describe the following:

- discuss a method to use in place of the “poison pill” and implement it
- discuss how you ensure that the producers don’t overwrite and consumers don’t read empty queues.

Ensure that each file contains a class called e.g. Task1 which contains the classes needed for the task such that if one would want the barrier from task1 you would call it using Task1.Barrier(params).

4 Marking

The marking for this practical will work as follows:

- You will implement the tasks and upload your solution to Fitchfork and Clickup.
- You will receive a marks as follows:
  - Correctness of Implementation (20%) – From Fitchfork
  - Understanding of Implementation (40%)
  - Justification of Choices Made (20%)
  - Depth of discussion (10%)
  - Clarity of discussion (10%)

5 Upload Checklist

The following files should be in the root of your archive.

- Main.java will be overridden
- Task1.java
- Task2.java
- Task3.java

6 Submission

You need to submit your source files on the FitchFork and Clickup. All methods need to be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. Your code must be able to be compiled with the Java 8 standard.

For this practical, you will have 5 upload opportunities on Clickup and 20 upload opportunities and your best mark will be your final mark. Upload your archive to the appropriate slot on the FitchFork website well before the deadline. **No late submissions will be accepted!**