UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

COS226 - Concurrent Systems
Practical 1 Specification

Release Date: 28/07/2025

Due Date: 04/08/2025 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 Task 1(20)

For this task you are required to implement the classic **Peterson's algorithm** to achieve mutual exclusion between two threads. Skeleton code has been provided. Constraints are as follows: No use of synchronized, ReentrantLock, AtomicInteger is allowed, and you make only use volatile, boolean, and int for shared variables.

Example Output:

```
Thread-1 (ID 1) is running...

Thread-0 (ID 0) is running...

Thread-0 (ID 0) has incremented to 2

Thread-1 (ID 1) has incremented to 1

Thread-0 (ID 0) has incremented to 4

Thread-1 (ID 1) has incremented to 3

Thread-0 (ID 0) has incremented to 6

Thread-1 (ID 1) has incremented to 5

Thread-0 (ID 0) has incremented to 8

Thread-1 (ID 1) has incremented to 7

Thread-0 (ID 0) has incremented to 9

Thread-1 (ID 1) has incremented to 10

Final sheep counted: 10
```

Note: For testing you may include output such as the above but remove any output before submission

3 Task 2 (80)

Now that you are familiar with Peterson's algorithm for 2 threads. You will now extend this concept to support more threads, using a structured locking scheme.

You are required to implement a PattersonLock.java (named after the farmer who designed the program) which can lock up to n threads where n is any integer number. Your program must demonstrate correct mutual exclusion, no deadlock and fairness.

Example Output:

```
Thread-1 (ID 1) is running...

Thread-0 (ID 0) is running...

Thread-3 (ID 3) is running...

Thread-2 (ID 2) is running...
```

```
Thread-2 (ID 2) has incremented to 3
Thread-1 (ID 1) has incremented to 0
Thread-3 (ID 3) has incremented to 2
Thread-0 (ID 0) has incremented to 1
Thread-2 (ID 2) has incremented to 6
Thread-3 (ID 3) has incremented to 7
Thread-1 (ID 1) has incremented to 4
Thread-0 (ID 0) has incremented to 5
Thread-2 (ID 2) has incremented to 9
Thread-1 (ID 1) has incremented to 10
Thread-3 (ID 3) has incremented to 8
Thread-2 (ID 2) has incremented to 12
Thread-0 (ID 0) has incremented to 14
Thread-3 (ID 3) has incremented to 13
Thread-1 (ID 1) has incremented to 11
Thread-0 (ID 0) has incremented to 15
Thread-1 (ID 1) has incremented to 18
Thread-2 (ID 2) has incremented to 17
Thread-3 (ID 3) has incremented to 16
Thread-0 (ID 0) has incremented to 19
Expected total cows counted: 20
Actual total cows counted:   20
Test PASSED.
```

6 Marking

The marking for this practical will work as follows:

- You will implement the tasks and upload your solution to Fitchfork.
- The test classes will be overridden by FitchFork and will then be used to mark the output that is produced by your code.
- You will receive a mark on Fitchfork.

- A mark on Fitchfork of greater than 0 will allow you to go to a practical session to be marked by a tutor.
- In the practical session a tutor will ensure that your implementation has kept within the restrictions (i.e no copy-pasting, not using libraries to do the whole task). The tutor will then assess your understanding of your implementation and the practical content.
- You will receive a final mark which will be some weighted combination of your Fitchfork mark and your "understanding" mark.

7 Upload Checklist

The following files should be in the root of your archive

- Task1Test.java and Task2Test.java will be overridden
- PetersonLock.java
- PattersonLock.java
- CustomThread.java
- FlockOfSheep.java
- HerdOfCows.java
- TreeNode.java

8 Submission

You need to submit your source files on the FitchFork website (https://ff.cs.up.ac.za/). 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 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!**