UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS 226 - Concurrent Systems

Practical 1

- **Date issued:** 28 July 2022
- **Deadline:** 4 August 2022 (Midnight)
- This practical consists of 2 task. Read each task **carefully!**

1 Introduction

1.1 Objectives and Outcomes

This practical aims to introduce the topic of Java threads. It is already assumed you are able to create programs within the Java language. Please familiarize yourself with all aspects of Java threading as from here onwards, all future practicals will assume you are familiar with them.

You must complete this assignment individually. Copying will not be tolerated.

1.2 Submission and Demo Bookings

While future practicals may contain starting code for you to build on, for the purposes of this practical, all code must be written from scratch.

Submit your code to **clickup** before the deadline.

You will have to demonstrate each task of this practical during the **physical** practical lab session. So be sure to create copies of your source code for each task separately. Booking slots will be made available for the practical demo.

1.3 Mark Allocation

For each task in this practical, in order to achieve any marks, the following must hold:

- Your code must produce console output. (As this is not marked by fitchfork, formatting is not that strict)
- Your code must not contain any errors. (No exceptions must be thrown)
- Your code may not use any external libraries.
- You must be able to explain your code to a tutor and answer any questions asked.

The mark allocation is as follows:

Task Number	**Marks**
Task 1	5
Task 2	5
Total	10

2 Practical Requirements

Given 2 array data structures one for todo tasks and other for completed tasks. Implement 2 threads that illustrate task allocation and execution of the task by moving the tasks from the todo list and adding them to completed list.

2.1 Task 1 - Thread Creation

For the following task you will need to implement threading in Java based on the above information. You may use either method for your implementation (thread or Runnable). The following must be completed:

- A class named **Scrumboard** must be created, this class must consist of:
  - Two array list of type string, named **Todo** and **Completed**. Todo array must be initialized with values A-J and Completed must be empty
  - A method to return the next item in the Todo list
  - A method to store a single task into Completed list via passed parameters
  - You can add helper variables to access the two lists
- A class that implements a thread, which must consist of: TDemo
  - A variable to hold a reference to a **Scrumboard** object.
  - A constructor that takes in a **Scrumboard** object and stores it in the variable.
  - The run() method, which should call the **Scrumboard** object methods, 5 times, in this order:
    1. get next task from **Todo**
    2. display output as follow (NB! to illustrate task execution before marking as completed):
       * [Thread Name] Task: [Task]
       * **Example:** Thread-0 Task: E
    3. add task to **Completed**

- Two threads need to be initialized and run using the class you have created and the shared Scrumboard object passed to them.
  Example of creation if the MyThread class extends Thread:

```
Scrumboard s = new Scrumboard();
Thread t1 = new MyThread(s);
Thread t2 = new MyThread(s);
```

- If the methods of the **Scrumboard** object are called 10 times, the Scrumboard should have no task remaining in the **Todo** list and 10 tasks in the **Completed** list, however notice this may not be the case in your code.

2.2 Task 2 - Peterson Lock

For the next task you will need to extend your previous implementation to make use of a lock. Implement your own **Peterson lock**.

You will be using this lock to enforce mutual exclusion to the **critical section** of the previous task, so that only one thread may access the critical section at a time. The content of the **Todo** and **Completed** lists will now be logically correct. i.e. if the methods of the Scrumboard object are called 10 times, the Scrumboard should have no task remaining in the Todo list and 10 tasks in the Completed list.

The following code may assist in creation and use of the lock: (You may also consult the documentation online)

```
Lock l = ....;
.....
l.lock();
try{
    //critical section....
}
finally{
    l.unlock();
}
```

Notes:

- Create a new class for the Peterson Lock that **implements** the **Lock** interface from Java, similar to how you would implement the **Runnable** interface in thread creation.
- Copying the code directly from the textbook will probably not work, as Java handles Thread IDs differently, you will have to find a way to use the **Thread Name** of a thread to differentiate the threads.

so in java a thread has an id, in the textbook they assume that the thread ids are numerical order hence they use -1, however in practice the threads might have a large difference in thread ids