UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS 226 - Concurrent Systems

Practical 1

- **Date issued:** 27 July 2023
- **Deadline:** 3 August 2023 (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 an array of integers from n to m. Apply multi-threading to search for prime numbers in array using the methods highlighted in task 1 and 2. For the tasks you will need to implement threading in Java. You may use either method, thread or Runnable, for your implementation.

2.1 Task 1 - Divide and conquer

Balance the load between the threads by dividing the search space. That is, each thread is allocated a specific range, within the array, to search.

The following must be completed:

- An array of integers must be defined as follow:
  - size = m-n
  - initialize with integers from n-to-m
- A class that implements a thread, which must consist of:
  - A variable to hold a reference to the **array** object and a two variables to hold the search range.
  - A constructor that takes in an **array** object and the search range, stores them in their respective variables.
  - The run() method, which should search the **array** object and display output, when a prime number is found, as follow:
    - {Thread Name} {Range}: { prime number }
    - **Example:** Thread-0 [0-10]: 2
- A number of threads need to be initialized and run using the class you have created and the shared array object passed to them.
  Example of creation if the MyThread class extends Thread:

```
Integer array = new Integer[size];
Thread t1 = new MyThread(array, range 1... );
Thread t2 = new MyThread(array, range 2... );
Thread t3 = new MyThread(array, range 3... );
```

2.2 Task 2 - Shared Counter

For the next task you will need to update your previous implementation to make use of a shared counter, instead of dividing the search space, as follow:

- The threads use a shared counter object to access the array. The threads access the counter using the **getAndIncrement** method to allow well coordinated search among the threads. (Refer to chapter 1 and 2 of the textbook for more information)
- In addition, you are required to make use of a lock mechanism. Use your own implementation of the **Peterson lock** to enforce mutual exclusion to the critical section, so that only one thread may access the critical section at a time.
- Your program must produce output as in Task 1.

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

```
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.