UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science
COS 226 - Concurrent Systems

Practical 2

- **Date issued:** 04 August 2022
- **Deadline:** 18 August 2022, 8:00 PM
- This practical consists of 2 task. Read each task **carefully!**

1 Introduction

1.1 Objectives and Outcomes

This practical aims to further explore locking via implementation of new locks when 3 or more threads are involved.

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

1.2 Submission and Demo Bookings

You are provided with some skeleton code to aid in the assignment, consisting of the following Java classes: **Main, Tansport, Venue, Filter, Bakery**

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 for **locking** apart from those already provided.
- 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

5 buses are used to transport people from various locations to **venue A**. At venue A, the buses will have to wait in line before they can get to the drop-off point. Only one bus is allowed on the drop-off point, at all times.

2.1 Task 1 - Filter Lock

For this task you will need to implement the simulation of the above mentioned scenario as well as implement a **FilterLock** to enforce mutual exclusion.

The following must be completed:

- The **run()** method of the **Transport** class needs to simulate 4 buses accessing the drop-off point through the **dropOff()** method of the **Venue** class. Each bus will take 5 loads, i.e. calls **dropOff()** 5 times.
- The **dropOff()** method needs to simulate a bus dropping off people at the destination venue. To do this, once at the drop-off point, the thread representing a bus, i.e. Transport, will need to sleep for a randomly selected amount of time between 200 and 1000 milliseconds. Remember only one bus is allowed at the drop-off at any time!
- A FilterLock will need to be implemented inside the Filter class. i.e. A **lock()** and **unlock()** method.
- The following output is expected:
  - When a bus ATTEMPTS to drop-off commuters, the following will need to be output:
    BUS ([Thread-Name]) is waiting to drop-off: Load [Load-Number]
    Example: BUS (Thread-1) is waiting to drop-off: LOAD 1.
  - When a bus ENTERS the drop-off point, the following will need to be output:
    BUS ([Thread-Name]) is dropping-off: Load [Load-Number]
  - When a bus LEAVES the drop-off point, the following will need to be output:
    BUS ([Thread-Name]) has left: Load [Load-Number]

2.2 Task 2 - Bakery Lock

Some of the bus drivers and commuters are complaining about the drop-off method of being unfair. To solve this problem a new drop-off method is devised. For the next task you will need to modify your previous implementation to make use of a BakeryLock.

The following needs to be completed:

- The FilterLock from the previous task needs to be replaced by a BakeryLock.
- Implement your BakeryLock inside the **Bakery** class.
- Change the simulation you have created to make use of the BakeryLock instead of the FilterLock.