Middle East Technical University Department of Computer Engineering

CENG 443

Introduction to Object Oriented Programming Languages and Systems

Spring 2019-2020

Homework 2 - A Synchronization Object for Hard(ly) Working CENG Students

Due date: April 16, 2020, Thursday, 23:55

1 Introduction

In this homework you will practice the synchronization concepts you have learned in class by solving a variant of the Unisex Bathroom problem in Little Book of Semaphores, p.170 (Do not mind the name, you can use both Java monitors and conditions with `synchronized` blocks, or the `Semaphore, ReentrantLock, Condition` objects, the problem is open to both approaches).

Students in CENG tend to form study groups among each other. These groups prefer to study exclusively in a particular lab, and nowhere else. Different groups also dislike studying in the same lab at the same time. Moreover, each lab has a certain capacity and can not admit more students than that. In this assignment, you will implement a synchronization object that student threads can use for accessing the labs.

2 Implementation

2.1 `Lab` class

represents a lab. Has the following:

- `public Lab(String name, int capacity):` `capacity` is the maximum number of students that this lab allows. The name of the lab is used for debugging purposes.
- `public String getName(), public int getCapacity()` as public getters.

2.2 `StudyGroup` class

represents a study group. Has the following:

- `StudyGroup(String name, Lab lab):` `lab` is the laboratory this study group prefers. The name is used for debugging purposes.
- `public String getName(), public Lab getLab()` as public getters.
- `public void startStudyingWith():` A student associated with this study group will call this method to study in the registered lab. If the lab is occupied by another study group or the lab is completely full, the student should be blocked.
- `public void stopStudyingWith():` A student associated with this study group will call this method to signal that he has left the lab.

3 Other Specifications

- The problem should be solvable with the synchronization objects/language features mentioned in the introduction, but if you want to use something else, make a request to do so in ODTUCLASS.
- Thread safety for getters and constructors is not required.
- You may assume that the values given during the constructions of the objects are `final`, that is, the name and capacity of the labs / the name and the registered lab of the study groups will not change after construction.
- A `Main` class is provided for you at ODTUCLASS to test your implementation. **But do not take the results as absolute truth. It may miss cases with faulty implementation if threads context switch at the right spots.** (*Can you see how?*) Therefore your implementations will also be heavily white-box checked this time. In that regard, please be generous with your comments, especially on the long chunks of code (that could be code smell for bad OO), or risky acquisition of multiple locks for synchronization black magic.
- It is ok that your implementation is prone to starvation. But a starvation resilient implementation where study groups take turns is highly recommended as an exercise. You would also observe the performance drop along with it.
- If something you use has the checked `InterruptedException`, just wrap it with try-catch as the example `Main` file does.
- Your OO design will also affect your grade in a minor way. Do not take `Lab` as a wrapper of just an integer and a string. Fill it in a way that it affects the study groups it admits. If you are following the reference in the introduction, you may also want to create a `LightSwitch` class.
- **This is an individual assignment. Using any piece of code that is not your own is strictly forbidden and constitutes as cheating. This includes friends, previous homeworks, or the Internet. The violators will be punished according to the department regulations.**
- **Late Submission: As indicated in the syllabus, $5day^2$ for at most 3 days.**
- Follow the course page on ODTUClass for any updates and clarifications. Please ask your questions on ODTUClass instead of e-mailing if the question does not contain code or solution.

4 Submission

Put all of your classes and interface definitions under package "`hw2`". Submission will be done via ODTUClass. You will submit a single tar file called "`hw2.tar.gz`". This should contain the `hw2` directory, containing all your source files. Your homework should compile with

```
> tar -xf hw2.tar.gz
> cd hw2
> javac *.java
```