COS 226 Practical Assignment 3

- Date Issued: Monday, 15 August 2016, 13:30h
- Date Due: Friday, 26 August 2016, 17:00h
- Submission Procedure: Upload the specified required files for a given task.
- This assignment consists of **2 tasks** with marks stated alongside each question.

---

1. Introduction

From here on forward, the practical assignments assume that you know the basics of how threads work. You must complete this assignment individually.

You may ask the Teaching Assistants for help but they will not be able to give you the solutions. They will be able to help you with coding, debugging and understanding the concepts explained both in the textbook and during lectures.

2. Mark Allocation

This assignment is divided into **two (2) tasks**. Each task will be marked by an assistant manually and not in the sessions so upload all the necessary files to the correct upload slots. Your program must adhere to the following:

1. Your program must produce the expected output
2. Your program must not throw any exceptions
3. Your program must implement the correct mechanisms or algorithms, as specified by the assignment.

3. Source code

Before you begin, you should download the source code from the CS website. You are allowed to make changes to some of the files in the source code.

4. Task 1 – Single Writer, Multiple Readers (6 marks)

In this task, you will have to study the code that is in the *Task1* folder. The files provided are **PricesInfo.java**, **Main.java, Reader.java** and **Writer.java**. You will have to implement the Read/Write lock. You will have to study the Java language *ReadWriteLock* interface and the *ReentrantReadWriteLock* class. Furthermore, useful details for understanding on *single writer, multiple readers* can be found in Chapter 4 of the prescribed textbook.

In this task your will have to implement a simple introductory to *single writer, multiple readers*. Different reader threads read two price variables and a single writer thread set these variables at some set intervals. You must use read/writer locks to ensure that values are accessed appropriately.

Read price values are displayed after each price have been read, and the writer displays a message before it modify the values and it display a message after the price values have been set.

Your task is to edit the **PricesInfo.java** file only and implement the following member functions. The function prototypes have been provided for you.

- The *constructor*, initializing price1 as 1.0 and price2 as 2.0 and creating a new *ReadWriteLock* object.
- Two get methods for the reading the prices.
- A set method that sets the values of *price1* and *price2*. The arguments of this member method are two integer values to be set as *price1* and *price2*, respectively.
- Read or write locks must be implemented where necessary.

Compress all your files (especially **PricesInfo.java** file) and upload tar-ball (or *.zip* file) to the CS website to the **Practical3 Task1** submission box. Make sure that the uploaded files do not belong to any package.

5. Task 2 –Binary Tree Peterson Locks (10 marks)

The prescribed textbook states that the Peterson lock can be generalized by a binary tree implementation.

Another way to generalize the two-thread Peterson lock is to arrange a number of 2-thread Peterson locks in a binary tree. Suppose *n* is a power of two. Each thread is assigned a leaf lock which it shares with one other thread. Each lock treats one thread as thread 0 and the other as thread 1.

In the tree-lock acquire method; the thread acquires every two-thread Peterson lock from that thread's leaf to the root. The tree-lock's release method for the tree-lock unlocks each of the 2-thread Peterson locks that thread has acquired, from the root back to its leaf.

In this task you must implement the Peterson lock algorithm that will provides the full implementation of the Lock interface and you must implement a Peterson binary tree lock. You are provided with the following files: **Peterson.java, PetersonTree.java** and **ThreadID.java**.

Peterson class

The primary methods from the Lock interface which you must implement fully include the Paterson *constructor*, the *lock()* member function and the *unlock()* method.
Furthermore, two more overloading *lock(int)* and *unlock(int)* methods that each takes an integer argument must be implemented. The argument represents the input thread ID value in this case.
Other irrelevant methods of the Lock interface may be implemented with default values or provide relevant exceptions methods.

PetersonTree class

A binary tree that contains Peterson objects must be implemented. You can choose to implement your own structure of the binary tree implementation. You can have your own required member variables for this class.

The *PetersonTree constructor* must take an integer value as an argument for the number of threads.

The *PTreeLock()* – Given that each thread is assigned at the leaf level, then each thread tries to get the parent level lock until it have achieved the root lock. The thread that has obtained the root lock eventually enters the critical section.

The *PTreeUnlock()* – In the path from the root to the leaf through the tree, unlock all the Peterson objects.

You are expected to test if your functions work appropriately. You must use any lock based implementation of **your choice** to test your Peterson binary tree lock functions.

Compress the files and upload the tar-ball to the CS website to the **Practical3 Task2** submission box. Make sure that the uploaded file does not belong to any package.