COMP0008 Written Java Concurrency Coursework 2018/19

Introduction

The final lecture introduced a number of programs which demo'ed different Java concurrency classes: ReentrantLock, Semaphore, CyclicBarrier, CountDownLatch and ReadWriteLock.

This coursework will give you detailed understanding of how these concurrency classes work and also give you practice at using traditional Java concurrency mechanisms to implement thread-safe classes.

The key task is to implement your own versions of ReentrantLock, Semaphore, CyclicBarrier and CountDownLatch using only Java traditional concurrency mechanisms (i.e. synchronized methods/statements and the wait()/notify()/notifyAll() mechanisms for conditional synchronization). Each of these thread control classes should be implemented as a 'monitor' class using conditional synchronization.

To get started ...

You should unzip the "COMP0008.zip" framework and get it working with your favourite Java IDE (if you want to use an IDE).

The main demos (which have been slightly modified to simplify their use) are in the top-level default package. At the moment these use "your" concurrency classes that are within the comp0008 package. These classes are currently stubs and do not control the threads in any way. If you run the demos you will see that none of the threads are controlled properly compared to when you switch to using the Java concurrency package versions of these classes (you can switch in/out the real concurrency class implementations but commenting and uncommenting the relevant package include files at the top of the demo files).

Your key task is to add appropriate internal state and traditional concurrency mechanisms to these stub versions of ReentrantLock, Semaphore, CyclicBarrier and CountdownLatch so that they control the threads in the same way as the original classes. Note that you just need to implement the method signatures that are specified and not implement a full version of the orginal classes (for instance the constructors do not have a version with takes a Boolean to make the class fair so this does not need to be implemented. Similarly the ReentrantLock does not have a tryLock() method so this doesn't need to be implemented. But you should try to implement the methods that are given as faithfully as possible.) It should be noted that the demo's do not test all functionality required for the classes so you may have to test other aspects of these classes yourself.

Do use the Moodle Post if you have any (general) questions about this task.