The University of Birmingham
School of Computer Science

Exercise 03 - Mutual Exclusion

This exercise is to implement the five different locks discussed on the `n05_mutual_exclusion` slide deck and, for those with flaws, to run experiments that demonstrate those flaws:

- LockOne: Demonstrate mutual exclusion, concurrent deadlock (if possible), sequential non-deadlock
- LockTwo: Demonstrate mutual exclusion, sequential deadlock, concurrent non-deadlock (if possible)
- Peterson: Demonstrate mutual exclusion, no deadlock, starvation free
- Filter: Demonstrate mutual exclusion, no deadlock, starvation free, non-fairness
- Bakery: Demonstrate mutual exclusion, no deadlock, starvation free, Fairness

Use any suitable application to exercise the use of the locks, e.g. the `Counter` class from the slide deck or the `Queue` class from previously.

Do not use the `java.util.concurrency.Lock` interface, but make your own interface like the one in the slide deck

The important part of this exercise is too first predict exactly what bad symptoms these locks will exhibit when they misbehave, and set up the application to make those symptoms obvious when they occur and verify that you do not see any symptoms of misbehavior in the properties that each lock is supposed to hand correctly.

Note that observations cannot prove the *absence* of some behaviour: it can only prove the *presence* of a behaviour. All you can do is design your code to make the observation of a behaviour more probable. For example, proving that deadlock occurs requires setting your code up so that you do observe a deadlock incident. Since this might be an unlikely event even if the lock is flawed and allows deadlock, you can design your code to try many different sequences of locking and unlocking in different configurations many times over and identify and report deadlock when it occurs. You can not prove by observation that deadlock never occurs because no matter how long your program runs without deadlocking, it may still deadlock at some time in the future. All you can do is make it statistically more likely that a deadlock will be observed if the lock is flawed.