Practice Script 3

Summary:
- Native concurrent programming in Java. (cont.)
- Conditional synchronization.
- Implementation of some elementary synchronization mechanism.

Exercise 3.1

Using Java's native concurrency and synchronization mechanisms, solve the *producers-consumers* problem. There are P *producers* that add values to a shared queue (for the purposes of this exercise, a string queue could be used). On the other hand, there are C *consumers* that wait for values from the queue to be consumed. Both producers and consumers should wait a random spaced time (for a better simulation of the problem).

Study and implement a solution for the problem of program termination. You may try to combine the possibility of making consumers *daemon threads*, and synchronize both the end of producers, and an empty shared queue, in the main thread).

Exercise 3.2

Using Java's native concurrency mechanisms, implement as objects the following synchronization mechanisms: semaphores, and binary semaphores.

Please use the following services as the interface for these objects:

- `aquire`
- `release`

Exercise 3.3

Solve the problem 3.1 using only the semaphores developed in the previous exercise.

As a suggestion for devising a working solution, consider that there are three separate synchronization problems to solve: mutual exclusion (binary semaphore), empty and full queue (counting semaphores, the first initialized with 0, and the second initialized with the maximum size of the queue).

You can (and should) reuse the test program of exercise 3.1.

Exercise 3.4
Using Java's native mechanisms implement as objects the following synchronization abstractions:

- Mutex (recursive and non-recursive)
  - `lock`
  - `unlock`
  - `lockIsMine`
  - `isRecursive`
  - `recursiveCount`
  - `newConditionVariable` (after implementing the next abstraction)
- Condition variable (over the previous Mutex abstraction)
  - `await`
  - `signal`
  - `broadcast`
- Readers-writer exclusion (given priority to writers)
  - `lockReader`
  - `lockWriter`
  - `unlockReader`
  - `unlockWriter`
  - `lockIsMine`
  - `readerLockIsMine`
  - `writerLockIsMine`
  - `isLockedByReader`
  - `isLockedByWriter`
  - `isRecursive`
  - `recursiveCount`

You can make use of existing exercises to test these classes.