Lab 4.2: Thread Synchronization 2

Total points: 100

(Using `ReentrantLock` and `Condition` classes)

Assignment 1 (25 points) - Building H_2O

Create a monitor with methods `hydrogen()` and `oxygen()`, which wait until a water molecule can be formed and then return. Don't worry about explicitly creating the water molecule; just wait until two hydrogen threads and one oxygen thread can be grouped together. For example, if two threads call `hydrogen()`, and then a third thread calls `oxygen()`, the third thread should wake up the first two threads and they should then all return.

The constructor for this H2O class accepts an optional *fairness* parameter. When set `true`, the solution should take care the access order of hydrogen or oxygen threads.

(1. Identify the correctness constraints of the problem, 2. Specify the conditions that each method must wait for, 3. Write down the shared state that you will use to check these conditions)

Assignment 2 (25 points) - Project 1- Designing a Thread Pool

Thread pools were introduced in Section 4.5.1. When thread pools are used, a task is submitted to the pool and executed by a thread from the pool. Work is submitted to the pool using a queue, and an available thread removes work from the queue. If there are no available threads, the work remains queued until one becomes available. If there is no work, threads await notification until a task becomes available.

Your thread pool will implement the following API:

- `ThreadPool()` - Create a default-sized thread pool.
- `ThreadPool(int size)` - Create a thread pool of size size.
- `void add(Runnable task)` - Add a task to be performed by a thread in the pool.
- `void shutdown()` - Stop all threads in the pool.

Implementation will involve the following activities:

1. The constructor will first create a number of idle threads that await work.

2. Work will be submitted to the pool via the `add()` method, which adds a task implementing the `Runnable` interface. The `add()` method will place the `Runnable` task into a queue.

3. Once a thread in the pool becomes available for work, it will check the queue for any `Runnable` tasks. If there is such a task, the idle thread will remove the task from the queue and invoke its `run()` method. If the queue is empty, the idle thread will wait to be notified when work becomes available.

4. The `shutdown()` method will stop all threads in the pool by invoking their `interrupt()` method. This, of course, requires that `Runnable` tasks being executed by the thread pool check their interruption status.

Assignment 3 (25 points) – The First Readers-Writers Problem

The readers–writers problem: Suppose that a database is to be shared among several concurrent processes. Some of these processes may want only to read the database (**readers**), whereas others may want to update (write) the database (**writers**).

- If two readers access the shared data simultaneously, no adverse affects will result.
- However, if a writer and some other thread (either a reader or a writer) accesses the database simultaneously, chaos may ensue.

To ensure that these difficulties do not arise, we require that the writers **have exclusive access to the shared database while writing to the database.**

The 1st Readers-Writers Problem: The 1st readers–writers problem requires that no reader be kept waiting unless a writer has already obtained permission to use the shared object.

Assignment 4 (25 points) – The Second Readers-Writers Problem

The 2nd Readers-Writers Problem: The 2nd readers–writers problem requires that, once a writer is ready, that writer perform its write as soon as possible. In other words, if a writer is waiting to access the object, no new readers may start reading.