Problem 1. For the LockFreeHashSet, when an uninitialized bucket is accessed in a table of size $N$, it might be necessary to recursively initialize (i.e., split) as many as $O(\log N)$ of its parent buckets to allow the insertion of a new bucket. Show an example of such a scenario.
**Problem 2.** The ReentrantReadWriteLock class provided by the java.util.concurrent.locks package does not allow a thread holding the lock in read mode to then access that lock in write mode (the thread will block). Justify this design decision by sketching what it would take to permit such lock upgrades.
Problem 3. Consider an application with distinct sets of active and passive threads, where we want to block the passive threads until all active threads give permission for the passive threads to proceed.

A CountDownLatch encapsulates a counter, initialized to be $n$, the number of active threads. When an active method is ready for the passive threads to run, it calls countDown(), which decrements the counter. Each passive thread calls await(), which blocks the thread until the counter reaches zero.

Provide a CountDownLatch implementation using locks and conditions.

(Note: This is NOT the problem you’re expected to turn in for this week’s programming assignment. As such you do not need to go the full mile and actually have an implementation that you can compile and test—just provide code and turn it in as apart of your Gradescope handin.)
Problem 4. Both Alice and Bob have acquired multiple pets. Their pets still do not get along, but they must share a yard. We will build a monitor to help them out. There are two classes of threads, called **ALICEPET** and **BOBPET**. There is a single **Yard** resource that must be used in the following way:

1. Mutual exclusion: pets of different owners may not occupy the yard simultaneously,
2. Starvation-freedom: every pet who wants to enter the yard eventually enters.

The protocol is implemented via the following four procedures:

- **enterAlicePet()** delays the caller until it is ok for one of Alice’s pets to enter the yard
- **leaveAlicePet()** is called when one of Alice’s pets leaves the yard,
- **enterBobPet()** and **leaveBobPet()** do the same for Bob’s pets

The figure shows an example.

Implement this class using locks and condition variables. Explain why it satisfies mutual exclusion and starvation-freedom.

**Programming Assignment:**

The support code for this can be found at `/course/cs1760/pub/yard`. Implement the **Yard** class in **Yard.java** to satisfy the specifications listed above and provide a README with your answer to the second part of the question.

As always, email zipfile containing your solution code to this problem at `cs1760tas@lists.brown.edu`