EE 360P: Concurrent and Distributed Systems
Assignment 2

Instructor: Professor Vijay Garg (email: garg@ece.utexas.edu)

Deadline: 11:59 PM Feb. 14th, 2017

The submissions (adhering to the programming homework guidelines) must be uploaded to the canvas by the deadline mentioned above. Please submit one zip file per team, with the name format [EID1_EID2].zip.

1. **(15 points)** (a) A CyclicBarrier is a synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point. CyclicBarriers are useful in programs involving a fixed sized party of threads that must occasionally wait for each other. The barrier is called cyclic because it can be reused after the waiting threads are released. You only need to implement the following methods:

```
public class CyclicBarrier {
  public CyclicBarrier(int parties) {
    // Creates a new CyclicBarrier that will trip when
    // the given number of parties (threads) are waiting upon it
  }
  int await() throws InterruptedException {
    // Waits until all parties have invoked await on this barrier.
    // If the current thread is not the last to arrive then it is
    // disabled for thread scheduling purposes and lies dormant until
    // the last thread arrives.
    // Returns: the arrival index of the current thread, where index
    // (parties - 1) indicates the first to arrive and zero indicates
    // the last to arrive.
  }
}
```

   (b) **(15 points)** Implement `MonitorCyclicBarrier` using Java Monitor. Its interface is identical to `CyclicBarrier`. You are not allowed to use semaphores for its implementation.

2. **(40 points)** Implement a Java class `FairReadWriteLock` that synchronizes reader and writer threads using monitors with `wait, notify` and `notifyAll` methods. The class should provide the following methods: `void beginRead()`, `void endRead()`, `void beginWrite()`, and `void endWrite()`. A reader thread only invokes `beginRead()` and `endRead()` while a writer thread only invokes `beginWrite()` and `endWrite()`. In addition, the lock (instance of this class) should provide the following properties:

(a) There is no read-write or write-write conflict.

(b) A writer thread that invokes `beginWrite()` will be blocked until all preceding reader and writer threads have acquired and released the lock.

(c) A reader thread that invokes `beginRead()` will be blocked until all preceding writer threads have acquired and released the lock.

(d) A reader thread cannot be blocked if any preceding writer thread has acquired and released the lock.

The precedence of threads is determined by the timestamp (sequence number) that threads obtain on arrival.

3. **(30 points)** Newton, Benjamin, and Mary are planting seeds of apple trees. They have decided that Newton digs the holes and Benjamin places a seed into each hole. Then Mary fills the holes. There are some constraints to synchronize their progress:

   (a) Benjamin cannot plant a seed unless at least one empty hole exists and Mary cannot fill a hole unless at least one hole exists in which Benjamin has planted a seed.

   (b) Newton has to wait for Benjamin if there are 4 holes dug which have not been seeded yet. He also has to wait for Mary if there are 8 unfilled holes. Mary does not care how far Benjamin gets ahead of her.

   (c) There is only one shovel that can be used to dig and fill holes, and thus Newton and Mary need to coordinate between themselves for using the shovel; ie. only one of them can use the shovel at any point of time.

   The pseudocode of the threads Newton, Benjamin, and Mary is as follows.

Pseudocode

```
Thread Newton {
  while (true) {
    garden.startDigging();
    dig();
    garden.doneDigging();
  }
}
Thread Benjamin {
  while (true) {
    garden.startSeeding();
    plantSeed();
    garden.doneSeeding();
  }
}
Thread Mary {
  while (true) {
    garden.startFilling();
    Fill();
    garden.doneFilling();
  }
}
```

The template of Garden

```
public class Garden {
  // Implement the following function

  public Garden();
  public void startDigging();
  public void doneDigging();
  public void startSeeding();
  public void doneSeeding();
  public void startFilling();
  public void doneFilling();
  public int totalDug(); // holes dug
  public int totalSeeded(); // holes seeded
  public int totalFilled(); // holes filled
}
```

Provide an implementation for the Garden class (template given above) which uses `ReentrantLock` and `Condition` (from Java's `java.util.concurrent` package) to synchronize between the three threads while observing the constraints (a) to (c).