EE 360P **Vijay Garg**

Solution to Sample Problems

1. **(10 points)**

   (a) No, the program does not satisfy mutual exclusion. The following execution sequence allows both threads to enter the critical section:
   P0.1 → P1.1 → P1.2 → P1.3 → P0.2 → P0.3

   (b) No, it is not correct. The following sequence results in a deadlock: P0.1 → P1.1 → P0.2 → P1.2

2. **(22 points)**

```
public class FifoLock {
    private int turn = 0;
    private int ticket = 0;
    public synchronized int getTicket() {
        return ticket++;
    }
    public synchronized void requestCS(int ticketNumber) {
        try {
            while (turn != ticketNumber) wait();
        } catch (Exception e) {}
    }
    public synchronized void releaseCS() {
        turn++;
        notifyAll();
    }
}
```

3. **(23 points)**

```
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;
public class Bridge {
      private final ReentrantLock lock = new ReentrantLock();
      private final Condition laneAvailable = lock.newCondition();
      private int dir = 0;
      private int cars = 0;
      public void arriveBridge(int direction) throws InterruptedException {
          lock.lock();
          try {
              while (cars >= 4 || (direction != dir && cars != 0) )
                  laneAvailable.await();
              dir = direction;
              ++cars;
          } finally {
              lock.unlock();
          }
}

    public void exitBridge() {
        lock.lock();
    try { --cars;
        laneAvailable.signalAll();
    } finally {
        lock.unlock();
    }
    }
}
```

4. **(23 points)**

```
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;
class MultiLock {
      private final ReentrantLock lock = new ReentrantLock();
      final Condition roomAvailable = lock.newCondition();
      int reservedRoom = 0;
      int count = 0;
      public void request (int roomNumber) throws InterruptedException {
          lock.lock();
          try {
              while ((reservedRoom == 1) || ((reservedRoom != roomNumber) &&
                          (reservedRoom != 0)))
                 roomAvailable.await();
              count++;
              if (count == 1) reservedRoom = roomNumber;
          } finally {
              lock.unlock();
          }
      }
      public void release (int roomNumber) {
          lock.lock();
          try {
              count--;
              if (count == 0) reservedRoom = 0;
              roomAvailable.signalAll();
          } finally {
              lock.unlock();
          }
      }
}
```