**Threads/Synchronization Exercise II**

1. A lock can be acquired by only one thread at a time. True or False (Explain)
2. Rewrite the following code fragment for a synchronized method foo( ) to an equivalent code providing mutual exclusion, but **without** using a synchronized method.

public synchronized void foo( ) { public void foo( ) { // YOUR CODE HERE

// mutual exclusion HERE // provide mutual exclusion HERE

} }

1. Consider the following code if several MaybeRace objects are created and multiple threads execute their increment methods in parallel:

public class MaybeRace {

static int x = 0;

Object y = new Object ( );

static Object z = new Object( );

public void inc1( ) {

synchronized(y) { x = x + 1; }

}

public void inc2( ) {

synchronized(z) { x = x + 1; }

}

public synchronized void inc3( ) { x = x + 1; }

}

* 1. Using inc1( ) will prevent data races True or False (Explain)
  2. Using inc2( ) will prevent data races True or False (Explain)
  3. Using inc3( ) will prevent data races True or False (Explain)

1. The following class simulates a bank processing requests (transactions). Each request is a String that must be added to the List requestLog. Rewrite the class so that each request is processed (added to requestLog) by a separate thread, so that requests may be processed concurrently (in parallel). The following restrictions are associated with this problem:

* Your code must process each request in a separate (and new) thread
* You must insert synchronization to prevent data races if needed
* You may not add any instance variables or methods to the Bank class
* You may add one inner class to Bank

public class Bank {

private List<String> requestLog = new ArrayList<String>( );

public void processRequests(String[] requests) {

for (String r : requests)

requestLog.add( r ); // Appends argument to list

}

}

1. The following class implements a queue.

public class MyQueue<E> {

private ArrayList<E> list = new ArrayList<E>();

public boolean isEmpty() {

synchronized(this) {

if (list.size() == 0)

return true;

return false;

}

}

public E getFirst() {

synchronized(this) {

return list.remove(0); // removes first element and shift

// elements to the right

}

}

public void offer(E data) {

synchronized(this) {

list.add(data);

}

}

/\*\* If queue not empty, remove value from queue and return it.   
 \* Otherwise, if queue is empty, return null \*/

public E dequeue() {

if (!isEmpty())

return getFirst();

return null;

}

}

Describe a possible scenario where the dequeue method will not work as documented when the dequeue is accessed by multiple threads.