Name: ______________________________

CMSC 433 Section 0101
Spring 2016
Midterm Exam

Directions: Test is open book, open notes, open electronics; however, all electronic devices must have all wireless capability disabled ("airplane mode"). Answer every question; write solutions in spaces provided. Use backs of pages for scratch work. Good luck!

Please do not write below this line.

1. ________

2. ________

3. ________

4. ________

5. ________

SCORE ________

1. (20 points) TRUE / FALSE
   Answer each question below by writing "T" if you believe the statement is true and "F" if you think it is false. Each question is worth 2 points.

   (a) Processes share no memory with other processes.

   false： 共享内存来通讯

   (b) You would never consider having a multi-threaded application on a single-processor machine because the only purpose of multi-threading is to make programs run faster.

   false

   (c) Programs that have data races are also guaranteed to have race conditions.

   false

   (d) Programs that have race conditions are also guaranteed to have data races.

   false

   (e) If a thread object has been created, but its `start()` method not yet invoked, then the object's state is `NEW`.

   True

   (f) Monitor locks in Java are re-entrant.

   True

   (g) Memory accesses of all 32-bit types in Java are atomic.

   (h) If a class is thread-safe, then no special care is ever required to publish objects in that class.

   (i) If an object's fields are all declared to be final, then the object is immutable.

   (j) Balking and guarded suspension may be used to implement state-dependent actions in thread-safe classes.

2. (20 points) INTERLEAVINGS AND SYNCHRONIZATION

   (a) (5 points) Give the number of interleavings of a sequence of 3 elements and a sequence of 5 elements. For full credit, you must give a number!

   (b) (5 points) Explain why the Java Monitor Pattern requires fields to be private.

   (c) (5 points) Do immutable objects require their instance methods to be synchronized? Why or why not?

   (d) (5 points) Why is client-side locking frequently used when implementing compound actions?

Thread safety guarantees individual method invocations preserve correctness

3. (20 points) DEADLOCK AND VISIBILITY

   (a) (10 points) Suppose we have objects a, b and c, and threads T_1, T_2 and T_3 given as follows.

$\underline{T_1}$	$\underline{T_2}$	$\underline{T_3}$
`synchronized (a) {`	`synchronized (a) {`	`synchronized (b) {`
`    synchronized (b) {`	`    synchronized (c) {`	`    synchronized (c) {`
`        ...`	`        ...`	`        ...`
`    }`	`    }`	`    }`
`}`	`}`	`}`

   Can this program deadlock? Explain your answer, either by describing a deadlock scenario that can arise or by arguing that no such scenario is possible.

(b) (5 points) Is a properly constructed object from a thread-safe class always considered to be safely published if it is assigned to a volatile field immediately after its construction? Explain.

(c) (5 points) What is "stack confinement," and how may it be used to simplify the development of multi-threaded programs?

4. (20 points) CONCURRENT COLLECTIONS

   (a) (5 points) Explain the difference between "fail-fast" and "weakly consistent" collections.

   (b) (5 points) Explain why the `ConcurrentHashMap` class includes implementations of some compound actions. In particular, why does it not just implement the `HashTable` primitives, and let client code worry about compound actions?

(c) (10 points) In class we talked about the class `CopyOnWriteArrayList<T>`. Objects in this class implement a thread-safe `List<T>` interface while not requiring synchronization on any read operations.

Give an implementation of the `set()` operation for `CopyOnWriteArrayList<T>` by filling in the code below. Given an integer `i` and object `elt` of type `T`, `set(i,elt)` replaces the object at position `i` by `elt`. If the position `i` is not valid, then an `IndexOutOfBoundsException` is thrown. The following `ArrayList<T>` operations may be useful.

`add()` Adds argument to the end of an `ArrayList<T>`

`get(i)` Returns the object at position

`size()` Returns the number of elements in an `ArrayList<T>`

Please include your code in the following template.

```
public class CopyOnWriteArrayList<T> {

    private volatile ArrayList<T> backingList;  // The internal list
    ...

    public synchronized void set(int i, T elt) {










    }
    ...
}
```

5. (20 points) CODING

In this question you are asked to implement instance methods for a thread-safe class of *binary semaphores*. Here is the interface the class, `BinarySemaphoreThreadSafe`, must implement.

```
public interface BinarySemaphore {
    public void acquire() throws InterruptedException;
    public void release() throws InterruptedException;
    public boolean tryAcquire();
    public boolean tryRelease();
}
```

The meaning of these operations is as follows. A binary semaphore object maintains a field indicating whether or not it is available. `acquire()`, which is a blocking operation, should wait until the object is available, then change its status to unavailable and return. `release()` is also blocking; it waits until the object is unavailable, then changes its status to available and returns. `tryAcquire()` and `tryRelease()` are non-blocking versions of `acquire()` and `release()`, respectively; each operation attempts to change the availability status appropriately, returning a boolean indicating whether the operation succeeded (because the availability status was correct for the operation to succeed) or not (because the availability status was not correct).

On the next page, complete your implementation of the class by providing code for each of these four operations. You may use whatever strategy you wish. Note there is no need to implement a constructor, as the default constructor is adequate in this case.

```
public class BinarySemaphoreThreadSafe implements BinarySemaphore {

    private boolean available = true; // Initially, semaphore is available

} // end of BinarySemaphoreThreadSafe
```