Computer Science 483/580
Concurrent Programming
Midterm Exam
February 23, 2009

Your name ____

There are 6 pages to this exam printed front and back. Please make sure that you have all the pages now.

The exam is designed to take about 50 minutes.

This exam is **open book and open notes.** Cell phones, PDAs, computers, and other electronic devices are **not allowed** ***except*** you may use a laptop computer to refer to an electronic copy of the textbook if you do not have a hardcopy.

Honor statement: I have followed the above instructions regarding use of electronic devices while taking this exam.

Signature ____ Date _____________

1. Write a Java class called `Semaphore` that implements the basics of the semaphore abstraction described in section 5.5.3 (pp. 98-99) using Java intrinsic synchronization. In particular you need to implement: a constructor, taking as argument the initial number of permits; the `acquire()` method and the `release()` method. You do *not* need to implement methods that can time out nor show code related to exception handling. I just want to see the essence of the synchronization behavior.

2. “Joe Java” Programmer (JJP) has just learned about threads. He is writing a program that will add up all the space used by the files in a directory and its subdirectories (recursively) and he is excited to think how much faster it will be using threads because of all the parallelism he will achieve.

a) In Joe’s first attempt, the program enumerates the given, top-level, directory identifying the subdirectories and ordinary files. For each subdirectory a new thread is created that enumerates *that* subdirectory creating a thread for each of *its* subdirectories and so forth. Explain why this is a *really bad approach* to performing this computation.

b) Being disappointed with the results of the approach in part a) of the problem, Joe reads Chapter 6 and uses the Executor framework to manage threads in his program, using a fixed thread pool. What factors should Joe consider in choosing the size of the fixed thread pool and how will each of those factors influence his choice?

3. Java's intrinsic synchronization relies on data fields in all Objects to support `synchronized` blocks (and methods) and `wait/notify/notifyAll`. Given what you know about Java intrinsic synchronization, describe the data fields required in every object to support intrinsic synchronization including reentrancy; an unlimited number of threads waiting to enter a synchronized block; an unlimited number of threads waiting for `wait().` Don't forget that these fields may themselves be accessed concurrently.

Grads: Given the size overhead required in every Object to support intrinsic synchronization, what is your assessment of the design decision to enable every object to participate in intrinsic synchronization?

4. The Vector class below is intended to provide a mutable, thread-safe representation of 2-D vectors.

```
public class Vector {
    private int x, y;
    public Vector(int x, int y) {
        this.set(x, y);
    }
    public Vector(Vector v) {
        synchronized(v) { this.set(v.x, v.y); }
    }
    public synchronized int[] get() {
        return new int[] {x, y};
    }
    public synchronized void set(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public synchronized void add(Vector v) {
        synchronized(v) {
            this.x += v.x;
            this.y += v.y;
        }
    }
}
```

a) The code is subject to deadlock due to the `add` method. Give an example of client code that could cause a deadlock.

b) Suggest at least one way to fix the problem

c) The second constructor also acquires two locks. Even so, it is not subject to deadlock. Explain why.

5. “Joe Java” is back and is now concerned that his code will not perform well if it relies on the Java garbage collector. JJP decides to implement his own reference counting scheme for existing class `Foo` using the `RefCountedFooPool` class on the next page. You should assume that class `TSList` is a thread-safe list implementation with thread-safe methods `isEmpty(), add()` and `remove().`

Joe’s idea is that he will use the pool to recycle existing object instances whose ref-counts have reached 0.

Discuss the following:

a) are instances of `RefCountedFooPool` properly constructed? That is, does `this` escape during construction? Why or why not?

b) are instances of `RefCountedFoo` properly constructed? Why or why not?

c) is synchronization used correctly for the `refCount` field? Why or why not?

d) is synchronization used correctly for the operations on the `pool` object? Why or why not?

e) JJP should have in mind an invariant concerning the `refCount` field of objects contained in the pool list. What would a sensible invariant be? Is this what JJP implemented? Explain.

```
import java.util.*;
public final class RefCountedFooPool {
    private final class RefCountedFoo {
        public Foo o;
        private int refCount;
        public RefCountedFoo(Foo o) {
            this.refCount = 1;
            this.o = o;
            pool.add(this);
        }

        public synchronized int incrementRef() {
            return ++refCount;
        }
        public synchronized int decrementRef()
        {
            --refCount;
            if (refCount==0) pool.add(this);
            return refCount;
        }
    }

    public RefCountedFoo getFromPool() {
        if (!pool.isEmpty()) return pool.remove(0);
        return new RefCountedFoo(new Foo());
    }

    final TSList <RefCountedFoo> pool =
        new ArrayList<RefCountedFoo> ();
}
```