SFDV3006 Concurrent Programming
Midterm Exam 2011

Name:
ID:
Answer all questions and ALL parts of a question in your answer sheets
Time: 110 mins Marks:40

Long Answer Questions / Problems [20 marks]

Q1. Consider the following class which is not thread safe:

```
public class IDGenerator { //not thread safe
	private static long nextID = 0; //initial value

	public static long getNextID() {
		nextID = nextID + 1;
		return nextID;
	}
}
```

a) [2 marks] Assume that at a certain time the value of *nextID* is 7 and three concurrent threads are executing the *getNextID()* method. What could be *all the possible* values of *nextID* after the three concurrent threads finish executing the *getNextID()* method?

b) [1 mark] Rewrite the *getNextID()* method to prevent interference among threads such that only one thread is executing the *getNextID()* method.

Q2. [3 marks] The following class *FixedCounter* is not a monitor. Rewrite the class such that it is a monitor with the following condition synchronizations:
i) The *currentValue* of *FixedCounter* must never exceed MAX_VALUE on *increment()*
ii) The *currentValue* of *FixedCounter* must never reduce to less than MIN_VALUE on *decrement()*

```
public class FixedCounter {
	public static final int MAX_VALUE = 10;
	public static final int MIN_VALUE = 0;
	private int currentValue = 0;

	public int increment() {
		return currentValue++;
	}

	public int decrement() {
		return currentValue--;
	}
}
```

Q3. [2 + 2 marks] In Java *boolean* assignments are always atomic - there is no need to use *synchronized* even in presence of concurrent threads. Using this fact a smart Java programmer has written a class *ResourceLock* to control access to one single resource as follows:

```
class ResourceLock {        //for controlling access to one resource only
	private boolean taken = false;

	//acquire the lock
	public void acquireLock() { taken = true; }

	//release the lock
	public void releaseLock() { taken = false; }
}
```

Assume that a shared object of this *ResourceLock* class will be used concurrently by many threads - will the *ResourceLock* class work correctly to allow access to one thread at a time? i) Explain why this may not work correctly to allow access to only one thread at a time and how would you correct the above class to prevent this. ii) Rewrite the Java for the correct implementation of the *ResourceLock*.

Q4. The following is the code for an unbounded buffer which allows producers to run continuously - this can lead to starvation of consumers.

```
import java.util.*;
public class UnBoundedBuffer {
	private ArrayList buffer = new ArrayList();//to buffer items

	/**
	 * put() is called by Producer thread
	 */
	public synchronized void put(Object o) {
		buffer.add(o);
	}

	/**
	 * take() is called by Consumer thread
	 */
	public synchronized Object take() throws InterruptedException {
		while(buffer.size() == 0) wait;
		return buffer.remove(0); //remove first item
	}
}
```

a) [5 marks] Rewrite the above *UnBoundedBuffer* class such that a consumer runs every time there are at least 10 items in the buffer and the consumer should notify the producer only after it has taken 10 items from the buffer.

Q5. Consider the following *UnfairDatabase* class which controls access to a database for both readers and writers. Readers call *endRead()* and *startRead()* methods and writers call *startWrite()* and *endWrite()* methods. This implementation is unfair to writers because they have wait whenever readers are reading continuously one after another.

```
public class UnfairDatabase {

	private boolean writing = false;
	private int readerCount = 0;

	//reader methods
	public synchronized void startRead() throws InterruptedException {
		while (writing) wait();
		readerCount++;
	}

	public synchronized void endRead() {
		readerCount--;
		if (readerCount == 0) notifyAll();
	}

	//writer methods
	public synchronized void startWrite() throws InterruptedException {
		while(writing || readerCount > 0) wait();
		writing = true;
	}

	public synchronized void endWrite() {
		writing = false;
		notifyAll();
	}
}
```

[5 marks] Rewrite the above *UnfairDatabase* class such that it is fair to both readers and writers by giving them alternate turns to run so that both readers and writers get a fair and equal chance to run.

Short Answer Questions [5 x 2 marks each = 10 marks]:
1. In what situation would you implement the *Runnable* interface to create a thread class?

2. What are the similarities and differences between deadlocks and livelocks?

3. Define safety property and liveness property.

4. How do you decide when to use *notify()* and when to use *notifyAll()*?

5. What are the different uses of a semaphore? Give an example for each.

MCQs. [10 x 1 mark each = 10 marks]
1. In which of the following situations one or more threads are blocked?
a) deadlock
b) livelock
c) starvation
d) both a) and c)

2. Two threads using two locks will go into a deadlock if _____
a) they acquire the locks in same order
b) they acquire locks one after another
c) they acquire locks in different order
d) none of the above

3. Race condition is an example of violation of _____
a) progress property
b) safety property
c) liveness property
d) none of the above

4. Which of the following happens when *wait()* is called on a thread?
a) thread moves from RUNNING to RUNNABLE state
b) thread moves from RUNNING to NON-RUNNABLE state
c) threads move from RUNNABLE to NON-RUNNABLE state
d) thread moves from RUNNING to TERMINATED state

5. When a thread changes the condition of the monitor it should _____
a) do nothing
b) wait for the other thread to enter the monitor
c) call notify() or notifyAll()
d) execute its critical section

6. *notify()* will always select the _____ thread in the wait set
a) first
b) last
c) middle
c) any random thread

7. What happens when a thread which calls a *synchronized* methods cannot acquire the object lock?
a) The thread sleeps
b) The thread goes into a busy loop
c) The thread is blocked and put into the entry set
d) The thread is blocked and put into the wait set

8. How can the performance of a multi threaded program be increased when its threads are accessing data of a shared object?
a) Do no make the methods of the class synchronized
b) Increase the scope of the synchronized from the method to the class
c) reduce the scope of synchronized from method to the block around the critical section
d) Increase the number of CPUs

9. Which of the following is the correct way to create a thread in Java?
a) Implement Runnable and override the start() method
b) Implement the Thread interface and override the run() method
c) Overload the run method in the Thread class
d) Implement the Runnable interface and override the run() method

10. If *MyThread* implements the *Runnable* interface, what is the correct way to create an instance of it as thread?
a) Thread t = new MyThread();
b) MyThread t = new MyThread();
c) MyThread t = new Thread();
d) Thread t = new Thread(new MyThread());

END OF EXAM