



Final  
**EXAMINATION**  
Early Summer 2014

**DURATION: 3 HOURS**

**No. Of Students: 39**

**Department Name & Course Number:** Systems and Computer Engineering  
SYSC 3303 A Real-Time Concurrent Systems

**Course Instructor (s):** Lynn S. Marshall

**AUTHORIZED MEMORANDA**

**None: Closed book; No calculators**

**Students MUST count the number of pages in this examination question paper before beginning to write, and report any discrepancy to a proctor. This question paper has 9 pages + cover page = 10 pages in all.**

**This examination question paper may not be taken from the examination room.**

<b>In addition to this question paper, students require: an examination booklet</b>	<b>yes</b>
<b>Scantron Sheet</b>	<b>no</b>

---

**Name:** \_\_\_\_\_

**Student Number:** \_\_\_\_\_

**Instructions:**

1. This exam paper has **8** questions worth a total of **115** marks. The exam will be marked out of 100, so there are, in effect, **15** bonus marks. Questions 1-3 (worth 45 marks) are "Core Programming". In order to pass the course, you must get at least 15 marks on those three questions, in addition to getting at least 50 marks on the entire exam.
2. Read each question thoroughly before starting to write your answers. (Note that the Box class and some Java API reference material can be found in an appendix at the end of the exam.) Write your solutions in your answer booklet on the lined (right-hand) pages. Use the blank, left-hand pages in the answer booklet for your rough work. Solutions written on the blank pages will be assumed to be rough work and will not be graded unless you clearly indicate that you want them to be marked.
3. Some of the questions require you to write prose answers. You don't need to write pages and pages, but make sure that you provide enough detail to demonstrate that you understand the course material. Remember the "6 W's": "Who, What, When, Where, Why, and hoW."
4. Please, do not ask to have the questions explained to you. If you think something is unclear or ambiguous, make a reasonable assumption (one that does not contradict the question), write it at the start of your solution, and answer the question. However, if you think you have found an error in a question, feel free to ask for assistance!
5. ALL PAGES OF THIS EXAM PAPER MUST BE TURNED IN. DO NOT REMOVE ANY PAGES FROM THE EXAM ROOM.

**Appendix: Reference Material**

You may remove the appendix from the rest of the exam paper, but all pages must be handed in at the end of the exam.

**The Box Class**

```
1  public class Box
2  {
3      // The contents of the box.
4      private Object contents = null;

5      // If true, there is room for an object in the box.
6      private boolean empty = true;

7      // put: Puts an object (the parameter obj) into the box.
8      // Returns when the object has been put into the box.
9      public synchronized void put(Object obj) {
10         while (!empty) {
11             try {
12                 wait();
13             } catch (InterruptedException e) { return; }
14         }
15         contents = obj;
16         empty = false;
17         notifyAll();
18     }

19     // get: Removes and returns the object currently in the box.
20     // Returns when there's an object in the box to remove.
21     public synchronized Object get() {
22         while (empty) {
23             try {
24                 wait();
25             } catch (InterruptedException e) { return null; }
26         }
27         Object obj = contents;
28         contents = null;
29         empty = true;
30         notifyAll();
31         return obj;
32     }
33 }
```

**Java API Reference****Class Object**

```
// equals(Object) returns true if this object is equal to obj.
public boolean equals(Object obj);

// toString() returns a string representation of this object.
public String toString();

// wait() causes the current thread to wait until another thread invokes the
// notify() method or the notifyAll() method for this object. The interrupted
// status of the current thread is cleared if this method throws an
// InterruptedException. This method also has a version with a long millis
// parameter that will return after the given time in milliseconds, even if it
// has not been notified.
public final void wait() throws InterruptedException;

// notify() wakes up a single thread that is waiting on this object's lock.
public final void notify();

// notifyAll() wakes up all threads that are waiting on this object's lock.
public final void notifyAll();
```

**Class Thread**

```
// currentThread() returns a reference to the currently executing thread object.
public static Thread currentThread();

// Returns this thread's priority.
public final int getPriority();

// interrupt() interrupts this thread.
public void interrupt();

// interrupted() returns true if the current thread has been interrupted; false
// otherwise. The interrupted status of the thread is cleared by this method.
public static boolean interrupted();

// isInterrupted() returns true if this thread has been interrupted; false
// otherwise. The interrupted status of the thread is unaffected by this method.
public boolean isInterrupted();

// Causes this thread to begin execution; the Java Virtual Machine calls the run
// method of this thread.
public void start();

// Changes the priority of this thread to the specified newPriority.
public final void setPriority(int newPriority);

// Causes the currently executing thread to sleep (temporarily cease execution)
// for the specified number of milliseconds. The thread does not lose ownership
// of any monitors. Parameter millis is the length of time to sleep in
// milliseconds. The interrupted status of the current thread is cleared if this
// method throws an InterruptedException.
public static void sleep(long millis) throws InterruptedException;

// Causes the currently executing thread object to temporarily pause and allow
// other threads to execute.
public static void yield();
```