

SYSC 3303 A Summer 2014 Midterm Exam

No Calculators

First Name: _____ Last Name: _____

Student Number: _____ **Sample Solutions** _____

Q1 (/9)	Q2 (/16)	Q3 (/13)	Q4 (/17)	Q5 (/10)	Total (/65)

This exam is 1 hour 15 minutes (75 min) long. There are 5 questions worth 65 marks on 9 pages. Answer all questions in the space provided. Page 10 may be used for rough work. You may remove the last page. Ensure that your answers fully demonstrate your understanding of the material covered. Comments are required for full marks on programming questions. **Do not ask questions during the exam, unless you believe that you have found an error.** If you think something about a question is unclear or ambiguous, make a reasonable assumption and answer the question.

Question 1. (9 marks; 3 marks each part)

- a) Briefly describe mutual exclusion and condition synchronization, in general, without any reference to Java.

Mutual exclusion is used to protect critical sections of code. Critical sections are pieces of code that must be executed indivisibly to ensure that data is not corrupted. At most one thread of control can be in any of the critical sections associated with the same data.

Condition synchronization is used to enforce the order in which operations happen and/or ensure that certain operations only occur in certain states. This ensures that data is not changed or retrieved when the operations are not applicable (pre-conditions not met).

- b) How do we implement mutual exclusion in Java? Be sure to list all ways discussed in the lectures, and briefly explain how they ensure mutual exclusion.

In Java, mutual exclusion is implemented by the synchronized keyword/modifier in a method header, and by the synchronized (object) { } statement. When executing a synchronized method or statement (i.e. a critical section), the thread must first get the lock on the object ("this" in a synchronized method). No other threads needing to access a synchronized block (method or statement) requiring the same object lock may proceed until the first thread gives up the lock. This ensures that at most one thread of control may be in any of the critical sections associated with the same data.

- c) How do we implement condition synchronization in Java? Be sure to list all ways discussed in the lectures, and briefly explain how they ensure **condition synchronization**.

In Java, condition synchronization is implemented by the methods `wait()`, `wait(millis)`, `wait(millis,nanos)`, `notify()`, and `notifyAll()`. These methods may only be used inside critical sections (synchronized methods/statements). The wait methods are used to give up the lock if the state is not appropriate for the thread to proceed. Notify/notifyAll are used to let waiting threads know that the state has changed, and that they may now be able to proceed. This ensures that the operations proceed only in the correct state(s).

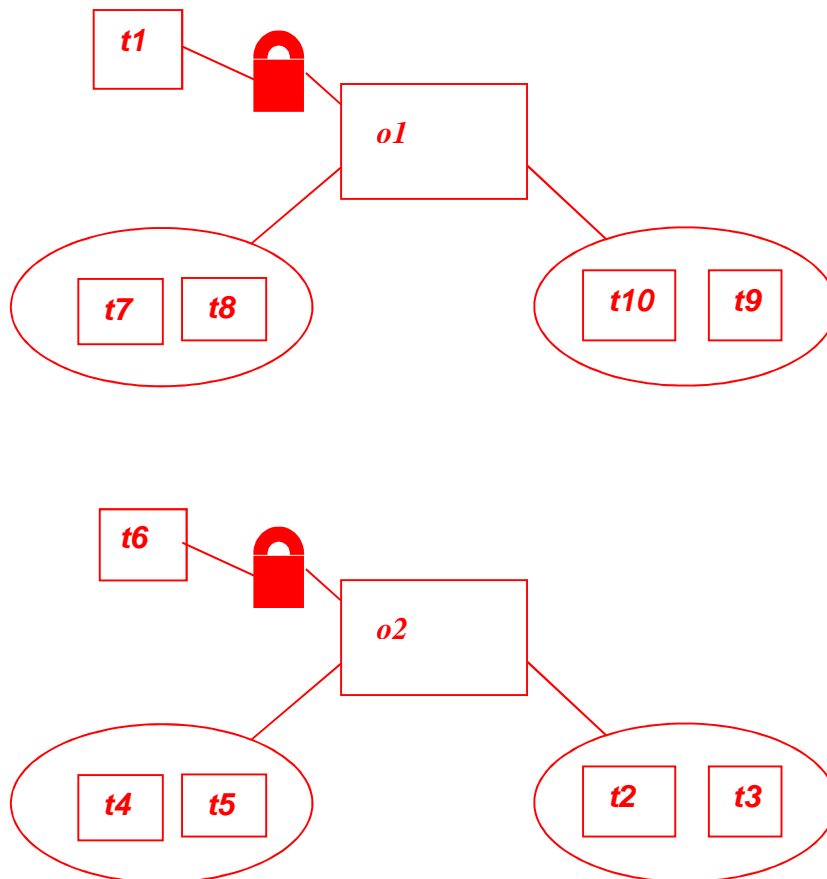
Question 2. (16 marks, 4 marks for part a, 2 marks each for the remaining parts)

Assume that we have a system made up of **twelve** threads (t_1, t_2, \dots, t_{12}). This system is using **two** object locks for objects o_1 and o_2 . At the current point in time, t_1 has the lock on o_1 , and t_6 has the lock on o_2 . Threads t_2 and t_3 are in o_2 's wait set, and threads t_4 and t_5 are in o_2 's blocked set. Threads t_7 and t_8 are in o_1 's blocked set, and threads t_9 and t_{10} are in o_1 's wait set. (Threads t_{11} and t_{12} do not need either lock at the moment.)

All parts of this question assume this starting point. They are **not** cumulative.

- a) Draw the **two** lock diagrams, one each for o_1 and o_2 , showing the above scenario.

(2 marks each)



- b) How many and which of our **12** threads **could** be running (i.e. have the processor) at this moment in time, assuming a uni-processor machine? Why?

With a uni-processor, at most one thread could be running. Thus, it could be that none of our threads are running. If one thread is running it is one of: t1, t6, t11 and t12. Threads t2-5, and t7-10 cannot be running as they are blocked or waiting.

- c) Which of our **12** threads **must** have already had the processor (at some time in the past or present)? Why?

All of threads t1-10 must have had the processor at some time in the past (or present). Threads t2, 3, 9 and 10 voluntarily invoked wait(), and they must have been executing to do that. Threads t1 and t6 currently have the lock and thus are executing critical sections (in state running or ready to run). Threads t4, 5, 7 and 8 are waiting for the lock, which means that they were executing and reached a critical section at which point they moved to the blocked set.

- d) If t1 invokes notify(), briefly explain what happens next.

One of the threads in o1's wait set (t9 or t10) will receive the notify() message and move to the blocked set. The JVM will pick the thread notified (either t9 or t10) at random. Thread t1 still has the lock at this point, as notify() must be invoked inside a critical section.

- e) If t6 gives up the lock, briefly explain what happens next.

Thread t6 will disappear from our lock diagram, and one of the threads in o2's blocked set will be chosen at random (either t4 or t5) by the JVM to get the lock next. That thread will then move to t6's position on the diagram and be removed from the blocked set.

- f) If t1 invokes notifyAll(), briefly explain what happens next.

All of the threads in o1's wait set, i.e. t9 and t10 will receive the notifyAll() message and move to the blocked set. Thread t1 still has the lock at this point, as notifyAll() must be invoked inside a critical section.

- g) If t6 invokes wait(), briefly explain what happens next.

Thread t6 will move to the wait set of o2's lock diagram, joining t2 and t3. Then one of the threads in o2's blocked set will be chosen at random (either t4 or t5) by the JVM to get the lock next. That thread will then move to t6's position on the diagram and be removed from the blocked set.

Question 3. (13 marks: 5+5+3 each part)

In this question, you are going to add three new methods to the Box class given on the last page of this exam. As mentioned at the top of the exam, you may remove the last page from the exam. The new methods are `replaceContents`, `replaceContentsWhenFull`, and `isFull`.

- a) `replaceContents` has one parameter, an `Object`, and does not return anything (i.e. it's a void method). Regardless of the current Box state (i.e. whether it is empty or not), this method sets the contents of the Box to the `Object` parameter.
- b) `replaceContentsWhenFull` has one parameter, an `Object`, and does not return anything. It is similar to method `replaceContents`, above, but it may only proceed if the Box is not empty. In other words, when the Box is full it replaces the contents with the `Object` parameter.
- c) `isFull` has no parameters and returns a `Boolean`. If the Box is empty it returns `false`, otherwise it returns `true`.

Space is provided on the next two pages for your answers. Clearly identify parts a, b, and c of your answers. Ensure that you provide sufficient internal synchronization for your Box methods to work properly in a multi-threaded environment.

```
// a) Replaces the box contents. This acts like put if the box is
// empty. If it is full, the contents are replaced. (5 marks)
public synchronized void replaceContents(Object obj) { // 1 mark
    contents = obj; // put or replace contents: 1 mark
    if (empty) { // box was empty: 1 mark
        empty = false; // no longer empty: 1 mark
        notifyAll(); // threads waiting to get need to know: 1 mark
    }
}

// b) Replaces the box contents when it is full. Once it is full,
// the contents are replaced. (5 marks)
public synchronized void replaceContentsWhenFull(Object obj) {
    // 1 mark
    while (empty) { // wait until it is full: 1 mark
        try {
            wait(); // 1 mark
        } catch (InterruptedException e) {return;} // 1 mark
    }
    contents = obj; // replace contents: 1 mark
    // don't change empty or notify as box was already full (-1 mark)
}

// c) Returns true if the box is full, false otherwise. (3 marks)
public synchronized boolean isFull() { // 1 mark
    return !empty; // 2 marks
}
```

// In addition, 0.5 to 1 mark deducted for inadequate / no comments.

// this page deliberately left blank so that the page numbers will match the question paper

Question 4. (17 marks: 10+2+5 for each part)

- a) In this question, you are going to rewrite the Box class (see the last page of the exam) with some changes. In addition to providing an object, the put method now provides a count indicating how many consumers will remove this object from the box before the box becomes empty. For example, if a producer invoked `box1.put(o1, 3)`, then the next three consumers to invoke `box1.get()` would all get `o1`. Only after `o1` has been removed 3 times will `box1` become empty. At that point, another object may be put in `box1` (with its count specifying how many consumers will get that object), etc. Ensure that your class contains sufficient internal synchronization to work properly in a multi-threaded environment. Call your new class `MultiBox`. (Write the entire class, including all fields and both methods.)

```
public class MultiBox // -1 mark if no comments
{
    private Object contents = null; // box contents
    private int count = 0; // 0 implies empty (empty no longer
        // needed); no marks deducted if you kept empty as long
        // as it's consistent with count: 1 mark

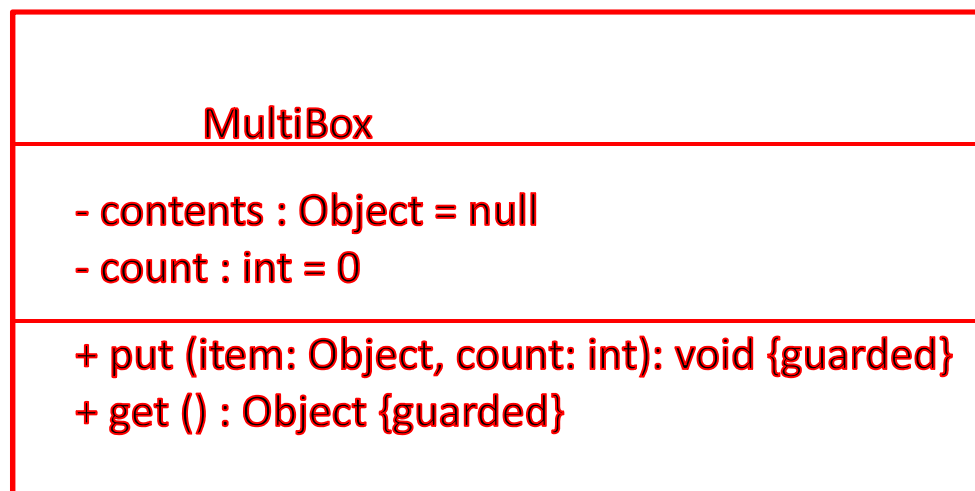
    public synchronized void put(Object item, int count)
    {
        // 5 marks total
        if (count<=0) count = 1; // do something reasonable: 1 mark
        while (this.count>0) { // box is not empty (or !isEmpty())
            try { // 2 marks
                wait();
            } catch (InterruptedException e) {return;}
        }
        contents = item;
        this.count = count; // 1 mark
        notifyAll(); // as count has gone from 0 to non-zero: 1 mark
    }

    public synchronized Object get()
    {
        // 4 marks total
        while (count==0) { // box is empty (or isEmpty())
            try { // 1 mark
                wait();
            } catch (InterruptedException e) {return null;}
        }
        Object obj = contents;
        count--; // 1 mark
        if (count==0) { // 1 mark
            contents = null; // optional
            notifyAll(); // count non-zero -> 0: 1 mark
        } // (Note: 2.5 marks deducted here if count not mentioned)
        return obj;
    }
}
```

- b) What is the level of thread safety of the `MultiBox` class? Briefly explain your answer, demonstrating that you understand the difference between the levels.

`MultiBox` is thread-safe. (1 mark) All methods contain sufficient internal synchronization to work properly in a multi-threaded environment. (1 mark)

- c) Draw a UML Class Diagram for the `MultiBox` class. For each attribute, give full details including the visibility, type and initial value, if applicable. For each operation, give full details including the signature, visibility, and synchronization property (even if it is the default).



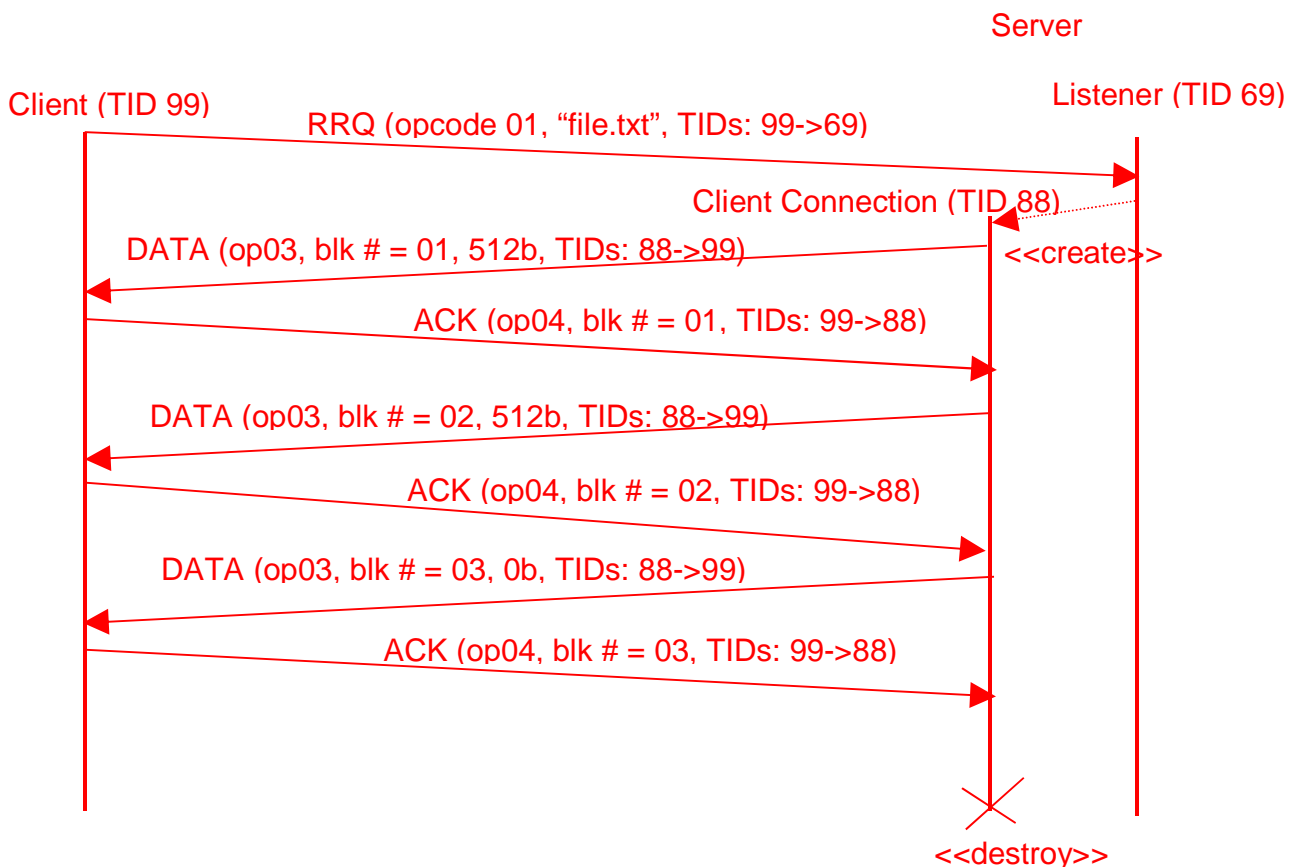
Question 5. (10 marks)

Draw a **Timing Diagram** showing the transfer of file “*file.txt*” containing 1024 bytes from the server to the client. The diagram should start with the initial "Read Request" by the client, and continue until the entire file has been transferred, showing all the packets sent during this transfer (from client to server and vice versa).

Your diagram must have **3 (three) vertical timelines**: one for the client, one for the server thread that waits for request packets (i.e. listener thread), and one for the client connection thread. **Clearly specify the TID for each of your three threads, either by using variable names or picking a specific TID.** Also, ensure that you show when threads are created/destroyed, if this happens during the time shown on your diagram. Assume that no errors occur during the file transfer, and that no packets are lost, delayed, or duplicated.

For each TFTP packet, you must clearly show the TFTP packet type, block number (if applicable), and number of bytes of data (if applicable).

1 mark per vertical line; 1 mark per message; port 69 must be used for the listener; any other ports including "x" and "y" for the other two. Arrows should slope down to show time passing, and should connect with the vertical lines to show where the messages come from / go to. (You don't have to show the TIDs on each message, but it makes things clearer.)



Here is the Box class referred to in questions 3 and 4.

```
public class Box
{
    private Object contents = null; // box contents
    private boolean empty = true; // is box empty?

    public synchronized void put(Object item)
    {
        while (!empty) {
            try {
                wait();
            } catch (InterruptedException e) {return;}
        }
        contents = item;
        empty = false;
        notifyAll();
    }

    public synchronized Object get()
    {
        while (empty) {
            try {
                wait();
            } catch (InterruptedException e) {return null;}
        }
        Object obj = contents;
        empty = true;
        contents = null;
        notifyAll();
        return obj;
    }
}
```