# CMSC 433 – Spring 2013 – Exam 1

**Name: _____**

Directions:

Test is closed book, closed notes, closed electronics.
Answer every question; write your answers in the spaces provided.
If you need extra sheets, raise your hand.
Good luck!

"I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination."

_____

_____

_____

_____

_____

_____

Please do not write below this line on this page.

Grading:

| | |
|---|---|
| 1. | 4. |
| 2. | 5. |
| 3. | 6. |

1. (30 Points) Answer each of the following assuming Java and the Java Memory Model is our context. Provide one or two sentences to support your answer. A yes/no answers without a corresponding written answer will not receive points.

(a) `i++;` is an atomic operation on an `int` YES NO

Why/why not?

(b) It is fine to guard values stored in an `Integer` object as a way to synchronize on an integer value while its value is updated.

YES NO

Why/why not?

(c) Give shared integers **`X,Y,Z,errorCount`** all initialized to 0 and the following two threads:

Thread T1 runs: **`while(true){X=Y; Y=Z+1; Z=X+1;}`**
Thread T2 runs: **`while(true){if (X>Z) errorCount++;}`**

The value in **`errorCount`** will *always* be 0. YES NO

Why/why not?

(d) Regardless of the programmer, a **`private`** field in a class can never escape.

YES NO

Why/why not?

(e) Assuming that **`SafeClass`** is a thread-safe class, is **`IsItSafe`** thread-safe as a class definition or are there any data races or deadlock risks possible?

```
public class IsItSafe {
   private  SafeClass latest;
   public  SafeClass  getter() {
      synchronized (latest) {return latest;}
   }
   public void setter(SafeClass latestIn) {
      synchronized (latest) {latest = latestIn;}
   }
}
```

YES NO

Why/why not?

(f) Would objects of the following class be ***immutable*** according to the rules of Java?

```
public class Point2D {
   private int xVal;
   private int yVal;

   Point2D (int x, int y) { xVal = x; yVal = y; }
   int getX() = { return xVal; }
   int getY() = { return yVal; }
}
```

YES NO

Why/why not?

2. (10 Points)
Main Method: White House defines:
`LinkedList<BudgetItems> cutCandidates = new LinkedList<BudgetItems>();`
and creates two objects that instantiate thread-based classes House and Senate that are each passed a reference to this shared list via the constructor and have a `run()` method as given below. The Main Method then calls `.start()` on both thread objects. From the starter code for the two `run()` methods below, ***fill in the code required*** (if any) to guard the shared data structure in a way that maintains good opportunities for concurrency. You may assume that the helper method `pickRandomProgram` is thread-safe.

In the **House** `run()` method, we start with the following code:

```
for (int i=0; i<5; i++) {

      BudgetItem oneToCut = House.pickRandomProgram();

      cutCandidates.add(oneToCut);

}
```

In the **Senate** `run()` method, we start with the following code:

```
for (int i=0; i<5; i++) {

      BudgetItem oneToCut = Senate.pickRandomProgram();

      cutCandidates.add(oneToCut);

}
```

3. (10 Points) Answer each of the following with two to three sentences.

(a) What is the difference between “a data race” and “non-determinism” in Java?

(b) Assume you had a class **`Worker`** that had a static Integer field keeping track of how many **`Worker`** objects were created. Your program will start an arbitrary number of threads, and each wants to be able to have its **`Worker`** objects know how many were created within its own thread, but not how many were created by other threads. What Java mechanism would most easily support this and how?

4. (20 Points)

(a) Assuming that **`SafeClass`** is a thread-safe class and we have a single shared static **`SafeClass`** object referred to by **`value`** in our class, and three methods named **`A,B,C`** each of which contains a block of code surrounded by **`synchronized(value)`**.

From within its synchronized block of code Method A calls Method B.
From within its synchronized block of code Method C calls Method B.

Given this context, answer each of the following:
(i) Assume only one thread is calling Method A. Why doesn't Method B block on value seeing as Method A already has its lock?

(ii) Assume you have two threads; T1 which repeatedly calls Method A and T2 which repeatedly calls Method C. Can the threads ever deadlock? Briefly explain why or why not.

(b) Assume the following three threads share objects Java, Fortran, Perl:

| Thread T1 | Thread T2 | Thread T3 |
|---|---|---|
| synchronized(Java) {<br>  synchronized(Fortran) {<br>  System.out.println("X");<br>  }<br>} | synchronized(Fortran) {<br>  synchronized(Perl) {<br>  System.out.println("Y");<br>  }<br>} | synchronized(Perl) {<br>  synchronized(Java) {<br>  System.out.println("Z");<br>  }<br>} |

fs

(i) Provide a sequence of events showing how this system could deadlock.

(ii) What principle of lock acquisition can be applied to prevent this? Show how that principle can be utilized to fix the above three threads.

5. (20 points)

(a) Explain how the **`volatile`** keyword impacts how the Java Memory Model treats a class variable and explain whether a variable in a class is marked as **`volatile`**, does that remove the need to use locks if multiple threads might be setting and getting that variable?

(b) In the box to the right, give the program-order ***event sequence*** for the following code segment running in a thread named **TZ**.
Recall that events are things such as **read(T2, x, _)** or **write(T1, x, _)**.

```
int x;
int y = 10;
int z = 100;

   x = 1;
   y++;
   z += x;
```

(c) Consider the following **event sequence trace** for part of a two-threaded program that share variables x and y where ***neither*** is marked as **`volatile`**.

lock(T1, sharedLock)
write(T1, y, _)
write(T2, x, _)
unlock(T1, sharedLock)
lock(T2, sharedLock)
read(T1, x, _)
read(T2, y, _)
unlock(T2, sharedLock)

Are there any volatility-based data races in this trace? Whether you say "yes" or "no" you need to explain your answer using the "happens-before" logic we've been using in class. If there is more than one, identify them all.

5. (15 points)
Image we are writing a program for a 1-Dimensional world. We want a thread-safe class called **LeftArrow** that contains two floating point values along the length of the Universe where values to the left are smaller than those to the right; one that records the coordinate of the head of the arrow and one that records the coordinate of the tail of the arrow. The class invariant is that the spot marking the head is required to be to the left of the spot marking the tail by at least 1 unit of measurement.

The class needs to have a constructor that takes in two floats and if the values passed in meet the above criteria, sets the spots of the head and tail. If the values do not meet the above criteria it needs to use the given spot for the head and then set the tail such that it is at least 1 unit of measurement to the left of the head.

There need to be two methods (**setTail** and **setHead**) that each get passed a floating point value. They need to work such that they only alter a coordinate if doing so does not break the invariant. They will return "true" if it did alter the coordinate and "false" is it did not.

There needs to be a method **toString** that returns a String of the form:
Left Arrow: From _ to _ .
where the _ spaces are filled in (respectively) with the coordinate of the head and the coordinate of the tail.

Implement the class by filling in the skeleton below:

```
public class LeftArrow {
//DATA GOES HERE









//CONSTRUCTOR
   public LeftArrow(float headIn, float tailIn) {









   }
```

```
//SETTERS AND GETTERS
   public boolean setTail(float newCoord) {









   }


   public boolean setHead(float newCoord) {









   }

   public String toString() {









   }
```

Solution working space…

```
public class RightArrow {
   int x1, y1; //tail
   int x2, y2; //head
   //Invariant: The point marking the tail must never be to
   //           the right of the point marking the head.

   public RightArrow(int x1in, int y1in, int x2in, int y2in) {
      y1 = y1in;  y2 = y2in;
      x2 = x2in;
      if (x1in<=x2) {x1 = x1in;}
      else {x1 = x2in;
   }

   //Returns true if and only if it was able to set the new point.
   public boolean setTail(int newX1, int newY1) {
      if (newX1<=x2) {x1=newX1; y1=newY1; return true;}
      else {return false;}
   }

   //Returns true if and only if it was able to set the new point.
   public boolean setHead(int newX2, int newY2) {
      if (x1<=newX2) {x2=newX2; y2=newY2; return true;}
      else {return false;}
   }

   public String toString() {
      String returnValue = "Right Arrow: ";
      returnValue += "(Tail= " + x1 +"," y1 + ") ";
      returnValue += "(Head= " + x2 +"," y2 + ") ";
      return returnValue;
   }
```

Leftovers….

| Thread T1 | Thread T2 |
|---|---|
| synchronized(sharedLock) {<br>x = 1;<br>} | y = 2;<br>System.out.println(x); |