Massachusetts Institute of Technology
6.005: Elements of Software Construction
Spring 2012

Quiz 2
April 30, 2012

Name: ________________________________________
Athena User Name: _____________________________________

Instructions
This quiz is 50 minutes long. It contains 10 pages (including this page) for a total of 100 points. The quiz is closed-book, closed-notes. Please check your copy to make sure that it is complete before you start. Turn in all pages, together, when you finish. Write your name on the top of every page. Please write neatly. No credit will be given if we cannot read what you write. Good luck!

<table>
<thead>
<tr>
<th>Questions</th>
<th>Page</th>
<th>Points per question</th>
<th>Total points</th>
<th>Your Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 and 2</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3 to 5</td>
<td>4</td>
<td>6</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>6 to 8</td>
<td>6</td>
<td>6</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>10 to 13</td>
<td>8</td>
<td>6</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>14 to 17</td>
<td>10</td>
<td>6</td>
<td>24</td>
<td></td>
</tr>
</tbody>
</table>

100
Question 1. **(circle the most appropriate answer)**
The following statement “When implementing a listener class, filter incoming events and only send an event to listeners when the data has changed from the last time.” is:

- **A.** False, because you should not filter the events and should send all to the listeners
- **B.** True, because otherwise there will be too many events send to the listeners
- **C.** False, because doing this will require keeping state in a normally stateless class
- **D.** True, because there could be cycles that will keep sending the same event around forever
- **E.** False, because the listener class does not involve sending and receiving messages

*D as cycles can create infinite looping when triggered by a single change*

Question 2. **(circle the most appropriate answer)**
Concurrent programs are hard to debug because:

- **A.** They are typically not fail-stop, thus, the effects of the bug becomes visible much later
- **B.** They can be not deterministic, thus, hard to reproduce the bug
- **C.** They can have complex control-flow making it difficult follow the program behavior
- **D.** They have multiple threads, making it impossible to single step through the execution in a debugger
- **E.** They can lead to deadlocks, which are impossible to detect in a debugger

*B. Non determinism is common due to multiple possible execution orderings of the threads. Non determinism makes it impossible to do iterative debugging and can lead to heisenbugs*
You are the new wiz programmer for a major league sports organization. Your first job is to write a seat reservation system for the team. As the team is popular, your job is to create a system where multiple people (using multiple threads) can simultaneously reserve the seats.

```java
public class Seat {
    int reservation_id; // 0 if not reserved
    ...
    public boolean isAvailable() { return reservation_id == 0; }
    public void reserve(int id) { this.reservation_id = id; }
}
```

```java
public class Reservations {
    Seat stadium[];
    ...
    <BODY>
}
```

Consider the `getTheSeat` method in the `<BODY>`. This method should reserve the requested seat and return true if it is available. Consider the following candidates for the next 3 questions.

<table>
<thead>
<tr>
<th>Option</th>
<th>Method Implementation</th>
</tr>
</thead>
</table>
| A      | ```java
    public synchronized boolean getTheSeat(int pos, int resid) {
        if(stadium[pos].isAvailable()) {
            stadium[pos].reserve(resid);
            return true;
        } else
            return false;
    }
``` |
| B      | ```java
    public boolean getTheSeat(int pos, int resid) {
        boolean available;
        synchronized(stadium[pos]) { available = stadium[pos].isAvailable(); }
        if(available) {
            synchronized(stadium[pos]) { stadium[pos].reserve(resid); }
            return true;
        } else
            return false;
    }
``` |
| C      | ```java
    public boolean getTheSeat(int pos, int resid) {
        synchronized(stadium[pos]) {
            if(stadium[pos].isAvailable()) {
                stadium[pos].reserve(resid);
                return true;
            } else
                return false;
        }
``` |
| D      | ```java
    public boolean getTheSeat(int pos, int resid) {
        synchronized(this) {
            if(stadium[pos].isAvailable()) {
                stadium[pos].reserve(resid);
                return true;
            } else
                return false;
        }
``` |
Question 3.  (circle all that apply)

Many of the fans will not be able to get a seat because the system will be slow to respond (i.e. will not scale).
What <BODY>(s) will lead to this problem?

(A)  (B)  (C)  (D)

A and D both lock the entire Reservations object

Question 4.  (Circle all that apply)

There will be pandemonium at the stadium because multiple people got tickets to the same seat.
What <BODY>(s) will lead to this problem?

(A)  (B)  (C)  (D)

B because the checking and reserving are not atomic. Thus, multiple checks can get true as an answer, before a single reserve goes through (only the last reservation will be survive)

Question 5.  (Circle all that apply)

All the fans will get their correct reservations on a timely manner.
What <BODY>(s) will lead to this nice solution?

(A)  (B)  (C)  (D)

C as synchronization is on each seat and check and reserve is done atomically
You want to add the ability to reserve multiple consecutive seats. You need to implement a `getSeats` method that returns true if you were able to find and reserve the indicated seat and \( n-1 \) adjacent seats to it. You come up with the following possible methods. For the next 4 questions, consider these methods.

```java
public boolean getSeats(int pos, int resid, int n) {
    if(n == 0) return true;
    synchronized(stadium[pos]) {
        if(stadium[pos].isAvailable()) {
            if(getSeats(pos+1, resid, n-1)) {
                stadium[pos].reserve(resid);
                return true;
            }
        }
    }
    return false;
}
```

A

```java
public boolean getSeats(int pos, int resid, int n) {
    if(n == 0) return true;
    synchronized(stadium[pos]) {
        if(stadium[pos].isAvailable()) {
            if(getSeats(pos+1, resid, n-1)) {
                stadium[pos].reserve(resid);
                return true;
            }
        }
    }
    return false;
}
```

B

```java
public boolean getSeats(int pos, int resid, int n) {
    if(n == 0) return true;
    synchronized(stadium[pos]) {
        if(stadium[pos].isAvailable()) {
            if(getSeats(pos+1, resid, n-1) || getSeats(pos-1, resid, n-1)) {
                stadium[pos].reserve(resid);
                return true;
            }
        }
    }
    return false;
}
```

C

```java
public boolean getSeats(int pos, int resid, int n) {
    if(n == 0) return true;
    synchronized(stadium[pos]) {
        if(stadium[pos].isAvailable()) {
            if(getSeats(pos+1, resid, n-1)) {
                stadium[pos].reserve(resid);
                return true;
            }
        }
    }
    return false;
}
```

D

```java
public synchronized boolean getSeats(int pos, int resid, int n) {
    if(n == 0) return true;
    if(stadium[pos].isAvailable()) {
        if(getSeats(pos+1, resid, n-1)) {
            stadium[pos].reserve(resid);
            return true;
        }
    }
    return false;
}
```

E
Question 6.  (Circle all that apply)
Pandemonium in the stadium as multiple people show up with tickets for the same seat. What <BODY>(s) will lead to this problem?

(A)  (B)  (C)  (D)  (E)

D as all calls to getSeats return the value from the base case, which is true. Thus, even if you get the reservation or not, you will always get true as an answer indicating you got all the seats.

Question 7.  (Circle all that apply)
Management is unhappy because multiple seats were left unsold (the reservation system either will inform the customers that the reservation failed when seats were reserved or the system froze and did not provide any feedback at all or was very slow and non-responsive). What <BODY>(s) will lead to?

(A)  (B)  (C)  (D)  (E)

B, C, and E. B will reserve subsequent seats if available but will return false if the first seat is not available, indicating to you that no seats are available but still reserving them. C will deadlock because some threads will try to acquire locks left to right and others right to left. E is slow as it locks the entire reservation object reducing parallelism.

Question 8.  (Circle all that apply)
Fans looking for multiple reservations quickly found the adjacent seats if they were available even during the peak usage. What <BODY>(s) will lead to this rosy scenario?

(A)  (B)  (C)  (D)  (E)

A as it locks one seat at a time and multiple locks are acquired in same order and check and reserve of a seat is done atomically and reserve is done after all checks are says all the seats are available.

Question 9.  (write less than four sentences)
Even though you found the correct solution(s), you are still not done as getSeats method may behave incorrectly for certain inputs. To help your fellow programmers who will be using your code, write the necessary preconditions to the getSeats method.

Assuming the last answer was A (change (1) if it is C)
1) pos >= 0 and pos <= stadium.length - n (otherwise array out of bounds access)
2) n > 0 (n >= 0 is OK)    (otherwise recursion will not bottom out)
3) resid != 0 (if resid == 0 is used to indicate no reservation in Seat class)
You want to record how the total number of seats reserved. You were not sure what methods in Aggregate need synchronization. So you wrote the following code for testing the different synchronization combinations.

```java
public class ReservationTest extends Thread {
    public static void main(String[] args) {
        for(int t=0; t<4; t++)
            (new ReservationTest()).start();
    }
    public void run() {
        Aggregate agg = Aggregate.getInstance();
        for(int i = 0; i < 1000; i++)
            agg.post(1);
        System.out.println(agg.get());
    }
}

public class Aggregate {
    static private Aggregate agg = null;
    int count;
    private Aggregate() { count = 0; }

    static public synchronized Aggregate getInstance() {
        if(agg == null) agg = new Aggregate();
        return agg;
    }
    public synchronized void post(int i) { count += i; }
    public synchronized int get() { return count; }
}
```

For different synchronization combinations, you observe different outputs. The following are four outputs, named A to D.

<table>
<thead>
<tr>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2148</td>
<td>1349</td>
<td>1000</td>
<td>1189</td>
</tr>
<tr>
<td>4000</td>
<td>2338</td>
<td>1486</td>
<td>3468</td>
</tr>
<tr>
<td>3213</td>
<td>2839</td>
<td>2000</td>
<td>4891</td>
</tr>
<tr>
<td>1324</td>
<td>1924</td>
<td>1000</td>
<td>2318</td>
</tr>
</tbody>
</table>

Note that when a variable is accessed or modified without synchronization by multiple threads, each read and write operation would still be atomic. For example, if a read and write to a variable occurs without synchronization, the read will either see the prior value or the value written.
Question 10.  (Circle all that apply)
If all three methods (getInstance, post and get) are synchronized, what of the above outputs are possible?

(A)  (B)  (C)  (D)

A as the last thread to exit will get all the updates reported, i.e. need to contain 4000.

Question 11.  (Circle all that apply)
If only the getInstance, and get are synchronized, what of the above outputs are possible?

(A)  (B)  (C)  (D)

A, B and C. When post method is not synchronized, updates can get lost. If you are lucky and get all the updates, thus A is valid. D is invalid since there can never be more than 4000 updates.

Question 12.  (Circle all that apply)
If only the post, and get are synchronized, what of the above outputs are possible?

(A)  (B)  (C)  (D)

A and C. In getInstance more than one Aggregate object can be created. A is valid if one object is created. C when 3 got created – two threads got its own and other two shared an object. Thus at end of the threads with the single object you get 1000 updates and one of the other two objects will see all their updates, i.e. 2000.

Question 13.  (Circle all that apply)
If only the getInstance, and post are synchronized, what of the above outputs are possible?

(A)  (B)  (C)  (D)

A Get method has no impact on the data and the last get we are interested happens after all the posts.
After the enormous success of Mr. Wozniak as a contestant, this season of Dancing with the Stars is all about computer industry luminaries. You, as the expert guest judge, are asked to give a programming challenge to the last four contestants Mr. Gage, Mr. Mallmer, Mr. Nastings and Mr. Ollison. For this test you write the following two versions of a Binary Tree class

The first version is:
```java
public class BinaryTree<V> {
    BinaryTree<V> left;
    BinaryTree<V> right;
    V value;

    BinaryTree(BinaryTree<V> left, BinaryTree<V> right, V value) {
        this.left = left;
        this.right = right;
        this.value = value;
    }

    V getVal() { return value; }
    BinaryTree<V> leftchild() { return this.left; }
    BinaryTree<V> rightchild() { return this.right; }
}
```

For the second version you add the following method
```java
void setVal(V value) { this.value = value; }
```

As the test, you ask the four contestants to write an equals method for your BinaryTree class. Note that a good equals method can be conservative (i.e. two objects can be reported as not equal when they might be equal). But if the method indicates that two objects are equal, all external observations of the objects, now and forever, should not be able to distinguish between them.

(A) Mr. Gage’s version:
```java
public boolean equals(Object obj) {
    return this == obj;
}
```

(B) Mr. Mallmer’s version:
```java
public boolean equals(Object obj) {
    if (! (obj instanceof BinaryTree)) return false;
    BinaryTree<V> that = (BinaryTree<V>) obj;
    return this.value == that.value;
}
```

(C) Mr. Nastings’s version:
```java
public boolean equals(Object obj) {
    if (! (obj instanceof BinaryTree)) return false;
    BinaryTree<V> that = (BinaryTree<V>) obj;
    if((this.left == null ^ that.left == null) || (this.left != null && !this.left.equals(that.left)))
        return false;
    if((this.right == null ^ that.left == null) || (this.right != null && !this.right.equals(that.right)))
        return false;
    return this.value == that.value;
```
Now your job is to grade the contestants in the following 4 questions.

Question 14. (circle one)
Who created the lamest solution that will throw an exception for certain valid inputs?

(A) Mr. Gage  (B) Mr. Mallmer  (C) Mr. Nastings  (D) Mr. Ollison

D as it does not check if this.left or this.right is not null

Question 15. (circle one)
Whose answer is plain wrong and violates the equals contract?

(A) Mr. Gage  (B) Mr. Mallmer  (C) Mr. Nastings  (D) Mr. Ollison

B because two different trees can have the same root value but different children and still be reported as equal

Question 16. (circle one)
What answer will work with the first version of the BinaryTree?

(A) Mr. Gage  (B) Mr. Mallmer  (C) Mr. Nastings  (D) Mr. Ollison

A and C. A reports equivalent when two trees are the same. C will report if the two trees have the same structure and each node has the same value. Since the tree is immutable, this is OK. Since we asked for cycle one cycling A or C or A and C all got full points.

Question 17. (circle one)
What answer will work with the both versions of the BinaryTree?

(A) Mr. Gage  (B) Mr. Mallmer  (C) Mr. Nastings  (D) Mr. Ollison

A as the trees are mutable only the same tree will be always equivalent. (thus C will not work)
The End!