Name: _____

CMSC 433 Section 0101
Fall 2012
Midterm Exam #1

Directions: Test is closed book, closed notes. Answer every question; write solutions in spaces provided. Use backs of pages for scratch work. Good luck!

Please do not write below this line.

---

1. _____ 2. _____ 3. _____ 4. _____ 5. _____
SCORE _____

1. (20 points) Answer each of questions in 1-2 sentences

   (a) (5 points) What is the difference between a "data race" and a "race condition?"

   (b) (5 points) What is the difference between a user thread and a daemon thread in Java?

   (c) (5 points) What does it mean for a (correct) class to be thread-safe?

   (d) (5 points) Why is it a bad idea to publish `this` in the constructor for a class?

2. (20 points) LOCKING, DEADLOCK

(a) (5 points) Explain what it means for locks to be *reentrant*.

(b) (10 points) Suppose three objects `A`, `B` and `C` have been declared, and consider the following threads.

T1

```
synchronized(A){
  synchronized(B){
  ...
  }
}
```

T2

```
synchronized(B){
  synchronized(C) {
  ...
  }
}
```

T3

```
synchronized(C){
  synchronized(A){
  ...
  }
}
```

Show that this system can deadlock by drawing an appropriate waits-for graph that can arise during an execution of the system.

(c) (5 points) How can the system in the previous part of this problem be fixed so that deadlock is impossible?

3. (20 points) JAVA MEMORY MODEL, HAPPENS-BEFORE

   (a) (6 points) Explain how the Java Memory Model treats volatile variables.

   (b) (7 points) Give the program-order event sequence for the following program.

```
public class Simple {
  public static void main (String[] args) {
    int x;
    int y = 1;
    x = 1;
    y += x;
  }
}
```

(c) (7 points) Consider the following (partial) event sequence for a program.

$\langle T_0, \text{write}, \texttt{x}, 0\rangle$
$\langle T_1, \text{write}, \texttt{y}, 1\rangle$
$\langle T_0, \text{unlock}, \texttt{lockA}\rangle$
$\langle T_1, \text{read}, \texttt{x}, 0\rangle$
$\langle T_1, \text{lock}, \texttt{lockA}\rangle$

Is there a data race in this program? Explain. (Hint: use "happens-before" in your explanation.)

4. (20 points) VISIBILITY, PUBLISHING

   (a) (6 points) Explain the difference between publishing an object and letting an object escape.

   (b) (7 points) Consider the following class definition.

```
public class Point {
  private double x;
  private double y;

  Point (double x, double y) { this.x = x; this.y = y; }

  double getX() = { return x; }
  double getY() = { return y; }
}
```

   Are the objects in this class immutable? Explain.

(c) (7 points) Consider the following class, which was studied in lecture.

```
public class Holder {
  private int n;
  Holder (int n) { this.n = n; }

  public void assertSanity () {
    if (n != n) throw new AssertionError ("BAD CONSTRUCTION!");
  }
}
```

Suppose we now publish an object in this class using the following.

```
public volatile Holder h = new Holder(42);
```

Can `h.assertSanity()` ever throw an `AssertionError`? Explain.

5. (20 points) CODING
Many compilers use a so-called *string table* to store the variable names that a programmer uses in her / his program. The idea is to replace (expensive) string comparisons in the compiler with (efficient) integer comparisons. The table may be thought of as an array. A compiler, upon seeing a variable name, would look the string up in the table and use the array index for that variable name in place of the actual string of characters.

Complete the following *thread-safe* implementation of a class `StringTable` by providing implementations of two public methods.

- `int lookup (String s)`: returns the position of `s`, adding it into the table if necessary
- `String getString (int i)`: returns the string stored at position `i`, or `null` if `i` is not a valid position in the table.

Here are some useful methods for `ArrayList<E>` that you may use.

- `boolean add(E e)`: adds `e` to the end of the list (boolean return value may be ignored)
- `E get (int index)`: returns the element at the specified position in the list
- `int indexOf (Object o)`: returns the index of the first occurrence of `o` in the list, or -1 if `o` is not in the list
- `int size ()`: returns the number of elements in the list

When comparing strings be sure to use the `boolean equals(String s)` method, not ==!

```
public class StringTable {

// Invariants:
// 1.  Any string appears at most once in the table.
// 2.  Once a string is added to the table, its index never changes.

  private final ArrayList<String> table;

  public StringTable () { table = new ArrayList<String> (); }

  // Spec for int lookup (String s)
  // Precondition: none
  // Postcondition:  returns index of s in table, adding s to end if necessary
  // Exception:  none









   // Spec for String getString (int i)
   // Precondition:  none
   // Postcondition:  return the string at position i, or null if i is not a position









} // end of StringTable
```