University of California at Berkeley
Department of Electrical Engineering and Computer Sciences
Computer Science Division

Spring 2004 **Jonathan Shewchuk**

CS 61B: Midterm Exam I

This is an open book, open notes exam. (However, laptop computers, cellphones, or anything with a wireless connection are not allowed.) **Do not open your exam until you are told to do so!**

Name:_____

Login:_____

Lab TA:_____

Lab day and time:_____

Do not write in these boxes.

Problem #	**Possible**	**Score**
1. Quickies	6	
2. Inheritance	5	
3. Box diagram	8	
4. Counting runs	6	
Total	25	

Problem 1. (6 points) **Quickies.**

a. (1 point) In Java, a `final` field is usually declared `static`. Why do you think that is?

b. (2 points) Fill in the blanks.
The following line of code will produce a _____ error if _____.
To give a different example, it will produce a _____ error if _____.

```
Object o = s.subString(0);
```

(Note: there are more than two possibilities. You only need to give two. The errors must occur at *this line of code*, not inside the `subString` method.)

c. (1 point) What is the bug in the following code fragment? Assume `i` is never negative.

```
switch (i % 3) {
case 1:
  System.out.println("i/3 has a remainder of 1.");
case 0:
  System.out.println("i is divisible by 3.");
default:
  System.out.println("i/3 has a remainder of 2.");
}
```

d. (2 points) The `moveToEnd` method moves the first node of a doubly-linked list to the end of the list. Fill in the blanks.

```
public class DList {          // Doubly-linked list without sentinel node
  protected DListNode head;
  protected DListNode tail;

  public void moveToEnd() {
    // Moves the first node of the list to the end of the list.
    if (head != null) {
      tail.next = _____;
      head.prev = _____;
      tail = _____;
      head = _____;
      tail.next = null;
      head.prev = null;
    }
  }
}
```

Problem 2. (5 points) **Inheritance.**
`DateAndPlace` is a subclass of the `Date` class you wrote for Homework 2.

```
class DateAndPlace extends Date {
  protected String place;

  public DateAndPlace(String date, String place) {
    super(date);
    this.place = place;
  }

  public boolean isSamePlace(DateAndPlace dp) {
    return place.equals(dp.place);
  }
}
```

a. (1 point) Must the fields inside the `Date` class be changed from `private` to `protected` for this subclass to compile? Explain.

b. (2 points) Recall that your `Date` class has a public `isBefore` method to compare two `Date` objects. Which line of code below causes an error? Is it a compile-time or run-time error? Why is it an error?

```
Date d = new DateAndPlace("2/23/2004", "Berkeley");
boolean b1 = d.isBefore(d);
boolean b2 = d.equals(d);
boolean b3 = d.isSamePlace(d);
```

c. (2 points) For convenience, we add another constructor to the `DateAndPlace` class to accept arguments given in reverse order. What's wrong with this new constructor?

```
public DateAndPlace(String place, String date) {
  this(date, place);   // swap arguments and call the first constructor
}
```

Problem 3. (8 points) **Fields, objects, and the stack.**
For the following code, draw a box-and-pointer diagram with a box for every activation record, field, and object in memory at the instant *before* the `callMe` method *returns* for the *first* time. Draw your diagram in the style used in the lectures and lecture notes. Include the activation records for all methods in progress, and illustrate which entities are inside those activation records. For every field, draw an arrow from its box showing what it references, or put an X in the box if the field is null. Don't forget the `this` field wherever it's appropriate. Entities (fields, objects, activation records) on the stack should be on the left-hand side of the page, and entities not on the stack should be on the right-hand side.

```
public class StringThing {
  public String[] tokens;

  public static void main(String[] args) {
    StringThing st = new StringThing();
    st.tokens = new String[5];
    st.tokens[2] = "Hello";
    st.tokens[0] = st.tokens[3];
    st.tokens[3] = "Goodbye";
    st.tokens = st.callMe(st.tokens, st.tokens[3]);
  }

  private String[] callMe(String[] sarray, String s) {
    if (s == null) {
      sarray[3] = "Hello";
      s = tokens[3];
      // Draw a picture of memory at THIS POINT IN TIME.
    } else {
      String[] others = sarray;
      sarray = new String[4];
      tokens[1] = s;
      sarray[2] = others[1];
      tokens = callMe(sarray, others[0]);
    }
    return sarray;
  }
}
```

Draw stuff on the stack on the left side | Draw stuff not on the stack on the right

Problem 4. (6 points) **Counting runs.**

In a list of `int`s, a *run* is a subsequence of `int`s that are all the same. For instance, there are three runs in the list 3 3 3 3 1 7 7 7: a run of 3's, a run of 1's (with just one member), and a run of 7's. Implement the `ArrayList.countRuns` method below, which returns the number of runs in an array `myArray` of `int`s. You may assume that `myArray` is not null, but it might reference an array of length zero.

Also, edit the two classes as necessary to ensure that they compile. (Besides implementing the method `ArrayList.countRuns`, you need to make one other change. Deleting a method is not an acceptable change.)

```
public abstract class List {
  protected int lastCount;

  public void printRunCount(List l) {
    lastCount = countRuns();
    System.out.println(lastCount);
  }

}

public class ArrayList extends List {
  protected int[] myArray;

  public int countRuns() {
    // Implement this method.









  }
                                          Check here if your answer |---|
}                                         is continued on the back. |___|
```