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

Spring 2012 **Jonathan Shewchuk**

CS 61B: Midterm Exam I

This is an open book, open notes exam. Electronic devices are forbidden on your person, including cell phones, iPods, headphones, and laptops. Turn your cell phone off and leave all electronics at the front of the room, or risk getting a zero on the exam. **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. Miscellaneous	8	
2. Inheritance	10	
3. Run-length encoding	7	
Total	25	

Problem 1. (8 points) **A miscellany.**

a. (2 points) Explain why binary search on a doubly-linked list is no faster than searching the list from one end to the other.

b. (2 points) When the Java Virtual Machine throws a `NullPointerException`, how do you determine which part of your code threw the exception? How do you determine which operations in that code might have thrown the exception?

c. (2 points) Circle the lines of code that can never cause a run-time error in any context (not counting system errors such as running out of memory). Assume that all these lines compile without errors.

```
boolean b1 = (head != null) && (head.next != null);
boolean b2 = (head == null) || (head.next == null);
Object o1 = (java.io.InputStream) System.in;
Object o2 = myObject.toString();
Comparable c = (Comparable) myObject;
int ia[] = {3, 7};
myStrings[1] = "Hello";
```

d. (1 point) Java has built-in arrays with a `length` field. But imagine if arrays were not built in to the language, and were instead somehow written out as a Java class definition. What modifiers would appear before `length` in its field declaration?

e. (1 point) Fill in the blanks: A cast changes the _________________________ type of an expression, and its effect on dynamic method lookup is ____.

Problem 2. (10 points) **Inheritance.**

On the next page, fill in the blanks so that the code compiles and runs without throwing an exception. Some blanks may require more than one word, or fewer. Make sure the class `Rational` and its subclasses `Integr` and `WholeNum` correctly implement rational numbers (fractions), integers, and whole numbers (nonzero integers), respectively. In particular, the method `Rational.compareTo` should return a negative value if `this` number is less than `o`, zero if the two numbers are equal, and a positive value otherwise, and it should correctly compare numbers among all three classes.

In the large box, override the `compareTo` method with an implementation that checks whether `o` is an `Integr`, calls the superclass `compareTo` if it is not, and otherwise does a faster comparison that does not look at the `denom` field.

```
public _____________ Rational _____________ Comparable {
  protected int numer;                       // Numerator of a fraction (rational number).
  protected int denom;                       // Denominator.  Invariant:  denom != 0.
  private ___________________ compare;

  public int compareTo(Object o) {
    _____________ other = (_____________) o;
    long myProduct = (long) numer * (long) other.denom; // High precision; no bits lost.
    long otherProduct = (long) other.numer * (long) denom;
    if (myProduct == otherProduct) {
      compare = ________;
    } else if (myProduct < otherProduct ^ __________________________________________) {
      compare = -1;                          //    This blank is a bonus point!
    } else {                                 //    (It's tricky.  Save it for last.)
      compare = 1;                           //    "^" means exclusive-or.
    }                                        //    false ^ false == true ^ true == false.
    return compare;                          //    false ^ true == true ^ false == true.
  }

  public _____________ Rational(int n, int denom) {
    numer = n;
    _____________ = denom;
    if (denom == 0) System.exit(1);          // Enforce the invariant.
  }

  protected ___________________ lastCompareTo() {
    return compare;
  }
}

public class Integr _________ Rational {     // Invariant:  denom == 1.
  public Integr(int n) {                     // Construct the integer n/1.
    ___________________;
  }

  // Override compareTo to be fast if o is an Integr, call superclass otherwise.






}

public class WholeNum _________ Integr {     // Invariants:  denom == 1, numer != 0.
  public WholeNum(int n) {                   // Construct the whole number n/1.
    ___________________;
    if (n == 0) System.exit(1);              // Enforce the invariant.
  }

  public short makesNoSense() {
    return Integr.lastCompareTo();
  }
}
```

Problem 3. (7 points) **Run-length encoding an array of strings.**

Write a method called `rle` in the `SListNode` class below that takes as **input** an array of `Strings` (the input parameter `strings`) and **returns** the head of a singly-linked list representing a run-length encoding of that array. Two adjacent `Strings` should be considered the same and compressed into a single run if they represent the same sequence of characters, even if they're different objects.

There is no `SList` class; just an `SListNode` class. Each `SListNode` has three fields: an `item` reference that points to a `String`, an `int` named `count` that records the number of occurrences of that `String`, and a reference to the `next` `SListNode` in the list.

Feel free to manipulate references directly. Do **not** assume that any methods are available unless you write them, except the constructor provided below. Assume that `strings` references an array whose length is at least one, so the run-length encoding will have at least one node and `rle` never returns `null`. The run-length encoding is allowed to reference the same `String` objects as the array, but you can copy the `Strings` if you prefer. (Hint: the code is simpler if you start at the end of the array and encode backward.)

```
public class SListNode {
  public String item;
  public int count;
  public SListNode next;

  public SListNode(String i, int c, SListNode n) {
    item = i;    count = c;    next = n;
  }

  public SListNode rle(String[] strings) {      // Run-length encode an array.
















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