COP 3530
Data Structures
Midsemester Exam

Name: ________________________________
Email: ______________________________

February 26, 2002

This exam has 3 questions. Each question starts on a new page. Please answer each question on its page. You may write on the back of a page. You may assume java.util has been imported.
1. **[60 points]** Consider the following method, whose implementation is not shown:

```java
// Precondition: Collection c represents a Collection of
// other Collections.
// c is not null; none of the collections are null
// str is not null
// Postcondition: returns the number of occurrences of
// String str in c.
public static int count( Collection c, String str )
```

An example of a collection that satisfies the stated precondition is Collection `c1` defined as follows:

```java
Collection c1 = new ArrayList();
Collection o1 = new TreeSet(); o1.add( "hello" ); o1.add( "world" ); o1.add( "foo" );
Collection o2 = new HashSet(); o2.add( "foo" ); o2.add( "bar" ); o2.add( new Integer( 5 ) );
Collection o3 = new LinkedList(); o3.add( "hello" ); o3.add( "foo" ); o3.add( "hello" );
c1.add( o1 ); c1.add( o2 ); c1.add( o3 );
```

The call to `count(c1,"hello")` returns 3.

(a) Provide an implementation of `count` *(approx. 10 lines)*.

(b) Assume that Collection `c` contains `N` collections, and that each of those collections contains `N` objects. What is the running time of `count`, as written in part (a)?

(c) Suppose it takes 2 milliseconds to run `count` when `N` (specified above) is 100. How long will it take to run `count` when `N` is 300?
2. [60 points] Assume that a singly linked list is declared as shown, by storing a reference to the first node in the list (first is null if the list is empty):

```java
class LinkedList {
    private static class Node {
        Object data;
        Node next;
    }

    private Node first; // represents first node

    public void trim() {
        /* YOU MUST WRITE TWO VERSIONS */
    }

    private Node trim(Node p) {
        /* YOU MUST WRITE ONE RECURSIVE VERSION */
    }

    /* OTHER METHODS NOT SHOWN */
}
```

(a) The public trim instance method is intended to remove from this list all nodes whose data field is null. Implement trim non-recursively (apx. 10 lines).

(b) An alternate implementation of trim would be to have trim call a recursive private routine. The private trim routine starts at Node p and removes all subsequent nodes whose data field is null, returning the resulting sublist (the return value is the first node from p onward that was not removed). Implement both the public driver trim and the private recursive trim routines (apx. 10 lines total).
3. [80 points]

(a) Static method `computeCounts` takes as input an array of strings and returns a map that stores the strings as keys, and the number of occurrences of each string as values. Implement `computeMap` as started below (appx 10 lines) AND PROVIDE THE RUNNING TIME OF YOUR ROUTINE:

```java
public static Map computeCounts( String[] strings )
{
```

(b) Write a routine that takes the map generated in part (a) and returns a list of the strings that occur most often (i.e. if there are k strings that are tied as the most common, the return list will have size k). Complete `mostCommonStrings` below (appx 20 lines) AND PROVIDE THE RUNNING TIME OF YOUR ROUTINE:

```java
public static List mostCommonStrings( Map stringsAndCounts )
{
```