CIS 211 Midterm 2, Winter 2011
1:00pm - 1:50pm, February 21, 2011

Instructions: Use the space provided for each answer. If necessary, use the back of the page or scratch paper (available at the front of the classroom). You must complete this exam independently, with no outside resources of any kind. This exam consists of 5 questions, worth a total of 100 points.

HINT: Look over all problems before you start and manage your time well. This is not an easy test. Do what you can in the time you have.

Below, we provide the Iterator and (partial) List interfaces, should you need them for reference:

interface Iterator<E>:

// Returns true if the iteration has more elements.
boolean hasNext();

// Returns the next element in the iteration.
E next();

// Removes from the underlying collection the last element returned by the iterator.
void remove();

interface List<E>:

boolean add(E e); // add to the end of the list
void add(int index, E element); // insert at specified index
void clear(); // remove all elements
boolean contains(Object o); // true if list contains the object
E get(int index); // get element at specified index
int indexOf(Object o); // index of first occurrence (or -1)
boolean isEmpty(); // list is empty (size() == 0)
Iterator<E> iterator(); // return iterator though this list
int lastIndexOf(Object o); // index of last occurrence (or -1)
E remove(int index); // remove at specified index
boolean remove(Object o); // remove first occurrence of object
E set(int index, E element); // set item at index to be given element
int size(); // number of elements in list
1. (15 points)

Suppose the following class has been defined:

```java
public class ListNode
{
    public int data;
    public ListNode next;

    public ListNode(int data) {
        this.data = data;
    }
    public ListNode(int data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}
```

What does the following code print out when run?

```java
ListNode x = new ListNode(1);
ListNode y = new ListNode(2, x);
ListNode z = new ListNode(3, y);
x.next = z;
y.next.next = y;
z = x;
x.data = 4;
y.data = 6;
z.next.data = 5;
System.out.println("x: " + x.data);
System.out.println("y: " + y.data);
System.out.println("z: " + z.data);
```

**ANSWER:**

1: 4
2: 5
3: 4
2. (a) (10 points) What does the following code print out for each input?
(HINT: You are free to refer to the List interface on the first page of the test.)

```java
public static void mystery(ArrayList<Integer> list)
{
    for (int i = 0; i < list.size() - 1; i++) {
        list.remove(i);
        list.add(i, list.get(i));
    }
    System.out.println(list);
}
```

[1, 2]  

[1, 2, 3, 4]  

ANSWER:

[2, 2]  

[2, 3, 4, 4]  

(b) (5 points) What is the runtime of this method? *Briefly* explain your answer (1-2 sentences at most).

Details: Use big-O notation, disregard constant factors and lower-order terms, and express your answer as the tightest bound you can. You may use $n$ to refer to the size of the input list, $l$.

ANSWER: This is $O(n^2)$, since it executes the remove operation $n$ times, and the remove operation in an ArrayList takes time $O(n)$. 


3. (20 points) Consider the following class definitions:

```java
public class One {
    public void b() {
        System.out.println("1B");
    }
}

public class Two extends One {
    public void a() {
        b();
        System.out.println("2A");
    }

    public void b() {
        super.b();
        System.out.println("2B");
    }
}

public class Three extends Two {
    public void b() {
        System.out.println("3B");
    }
}
```

(You may tear out this page for reference if you wish.)
Suppose the following variables are defined:

```java
One v1 = new One();
One v2 = new Two();
Two v3 = new Three();
```

Indicate on each line below the output produced by each statement shown. If the statement produces more than one line of output indicate the line breaks with slashes as in a/b/c to indicate three lines of output with a followed by b followed by c. If the statement causes an error (either at compile time or runtime), write the word “error” to indicate this.

```java
v1.a();
```

--- ERROR

```java
v2.b();
```

---

```java
v2.a();
```

--- ERROR

```java
v3.a();
```

---

```java
((Three)v2).b();
```

--- ERROR

(Problem instructions by Reges and Stepp.)

**ANSWER:**

```java
v1.a();  -- ERROR
v2.b();  -- 1B / 2B
v2.a();  -- ERROR
v3.a();  -- 3B / 2A
((Three)v2).m2();  -- ERROR
```
4. (25 points) Add the method add() to SortedIntList, which adds an integer to the list while maintaining sorted order. For instance, if the list contains the items [-2, 1, 13], then after calling add(4) it should contain [-2, 1, 4, 13]. You may assume that the list is already in sorted order before the method is called. The list is in ascending order, which means that the smallest value is at the front of the list and largest value is at the back.

You may not use an iterator. You may construct only one new ListNode object. However, you are free to use as many ListNode variables as you like. Your method must run in time $O(n)$, where $n$ is the length of the list.

Here is the definition of the SortedIntList classes:

```java
public class SortedIntList {
    // Reference to the first node in the list
    private ListNode front;

    ...
}
```

The definition of ListNode is the same as in Problem 1.

```java
public class ListNode {
    public int data;
    public ListNode next;

    public ListNode(int data) {
        this.data = data;
    }
    public ListNode(int data, ListNode next) {
        this.data = data;
        this.next = next;
    }
}
```

(Continued on the next page. You may tear out this page for reference if you wish.)
Here is the declaration of the method you are to implement, as a member of the SortedIntList class:

```java
/**
 * Add an integer to the list, maintaining sorted order.
 * @param item to add to the list
 */
public void add(int item)
```

**ANSWER:**

```java
{
  ListNode n = new ListNode(item);
  if (front == null || front.next == null || front.next.data > item) {
    n.next = front;
    front = n;
  } else {
    ListNode curr = front;
    while (curr.next != null && curr.next.data < item) {
      curr = curr.next;
    }
    n.next = curr.next;
    curr.next = n;
  }
}
```
5. (25 points) Write the method `listCompare`, which compares two lists of characters lexicographically (dictionary order). `listCompare` should return a negative integer if the first list should be ordered before the second, a positive integer if the first list should be ordered after the second, and zero if their contents are equal.

Do not use any String objects or methods. Your method must run in $O(n)$ time, where $n$ is the length of the longer list.

**Ordering example:**

\[
[] < ['a', 'b', 'c'] < ['c', 'a', 't'] < ['c', 'a', 't', 's'] < ['z']
\]

So your method must return a positive number if passed a first list containing `['c', 'a', 't', 's']` and a second list containing `['c', 'a', 't']`, since “cats” comes after “cat” in the dictionary.

If either List is null, throw an `IllegalArgumentException`.

(Continued on the next page. You may tear out this page for reference if you wish.)
/**
 * Compares two lists of characters lexicographically.
 *
 * @param l1 first list of characters to compare
 * @param l2 second list of characters to compare
 * @return negative number if l1 comes first; positive if l2 comes
 * first; 0 if l1 and l2 are equivalent
 */
public static int listCompare(List<Character> l1, List<Character> l2)
{
    if (l1 == null || l2 == null) {
        throw new IllegalArgumentException("null list");
    }

    Iterator<Character> i1 = l1.iterator();
    Iterator<Character> i2 = l2.iterator();
    while (i1.hasNext()) {
        if (!i2.hasNext()) {
            return 1;
        }
        int diff = i1.next() - i2.next();
        if (diff != 0) {
            return diff;
        }
    }
    if (i2.hasNext()) {
        return -1;
    } else {
        return 0;
    }
}