Scores:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Total (50 max)</td>
<td></td>
</tr>
</tbody>
</table>
1. (14 points) Pages 7 to 9 define a simplified version of the Java Collection interface (the SimpleCollection and SimpleIterator interfaces), together with a concrete implementation using linked lists (the class LinkedSimpleCollection and the auxiliary class Node). These definitions are identical to those in the review handout, except that the LinkedSimpleIterator class has been slightly simplified.

Suppose we execute the following program:

```java
public class Main {
    public static void main(String[] args) {
        SimpleCollection<String> c = new LinkedSimpleCollection<String>();
        c.add("c"); c.add("b"); c.add("a");
        SimpleIterator<String> i = c.iterator();
        i.next();
        c.removeAll("b");
        String s = i.next();
        // HERE
    }
}
```

In the space below, draw a diagram of the memory configuration (both stack and heap) when execution reaches the line marked HERE. (You can omit the parameter args to the main method.) Use the same conventions as the sample stack/heap drawing on page 10.
2. (6 points) Briefly (in two or three sentences) compare the way mutable state is treated in OCaml and in Java.
class MyExn extends RuntimeException {
}

class AnotherExn extends MyExn {
}

class C {
    public void foo() {
        try {
            System.out.println("a");
            bar();
            System.out.println("b");
        } catch (AnotherExn e) {
            System.out.println("c");
        } catch (MyExn e) {
            System.out.println("d");
        } finally {
            System.out.println("e");
        }
    }

    public void bar() {
        try {
            baz();
        } catch (AnotherExn e) {
            System.out.println("f");
            throw e;
        } catch (MyExn e) {
            System.out.println("g");
            throw new AnotherExn();
        } finally {
            System.out.println("h");
        }
        System.out.println("i");
    }

    public void baz() {
        try {
            System.out.println("j");
            if (true) throw new MyExn();
            System.out.println("k");
        } catch (AnotherExn e) {
            System.out.println("l");
        } finally {
            System.out.println("m");
        }
        System.out.println("n");
    }
}
4. (14 points) On the following page, complete the following definition of the class \texttt{IterMax}. Instances of \texttt{IterMax} are \textit{iterator transformers}—iterators built from other iterators. The \texttt{IterMax} constructor takes a \texttt{String} iterator \texttt{wrapped} (an instance of the \texttt{SimpleIterator} class defined on page 9) as an argument, and the \texttt{hasNext} and \texttt{next} methods use the corresponding methods of \texttt{wrapped} to do their job. Each time \texttt{next} is called on the outer \texttt{IterMax} iterator, it should return the first element from the inner \texttt{wrapped} iterator that is longer than all the elements previously returned. For example if the inner Iterator produces the sequence

\begin{verbatim}
"a" "bb" "c" "ddd" "ab" "abc" "eeee"
\end{verbatim}

then the outer one would produce:

\begin{verbatim}
"a" "bb" "ddd" "eeee"
\end{verbatim}

The outer iterator should never return \texttt{null} (even if the inner iterator does).
class IterMax implements SimpleIterator<String> {
    // Fields (fill in as needed)

    // Constructor (fill in body)
    IterMax (SimpleIterator<String> wrapped) {

    }

    // SimpleIterator methods (fill in bodies)
    public boolean hasNext () {

    }

    public String next () {

    }

    // Auxiliary methods (if needed)
}
5. (12 points) This problem again refers to the definitions of SimpleCollection and related classes on pages 7 to 9. Your job is to complete the definition of a method interleave that combines the contents of two LinkedSimpleCollections “in-place,” in alternating order. For example, if c contains "a", "c", and "e" (in that order) and c2 contains "b", and "d" (in that order), then after c.interleave(c2),

- c should contain "a", "b", "c", "d", and "e", and
- c2 should be empty.

If the two initial collections are not the same size, the extra elements should appear at the end of the interleaved collection. Your solution should not create any new objects (i.e., don’t use new anywhere).

```java
class LinkedSimpleCollection<E> implements SimpleCollection<E> {
    // ...
    public void interleave(LinkedSimpleCollection<E> other) {
        // ...
    }
}
```
For Reference:
SimpleCollection, SimpleIterator, and Node

interface SimpleCollection<E> {
    boolean removeAll(E element);
    boolean add(E element);
    boolean contains(Object o);
    SimpleIterator<E> iterator();
}

interface SimpleIterator<E> {
    boolean hasNext();
    E next();
}

class Node<E> {
    public E element;
    public Node<E> next;

    public Node (E element, Node<E> next) {
        this.element = element;
        this.next = next;
    }
}
For Reference: LinkedSimpleCollection class

class LinkedSimpleCollection<E> implements SimpleCollection<E> {
    private Node<E> first = null;

    public boolean removeAll(E element) {
        boolean changed = false;
        while (first != null && first.element.equals(element)) {
            first = first.next;
            changed = true;
        }
        if (first == null) return changed;
        Node<E> prev = first;
        for (Node<E> current = first.next; current != null; current = current.next) {
            if (current.element.equals(element)) {
                prev.next = current.next;
                changed = true;
            }
            prev = prev.next;
        }
        return changed;
    }

    public boolean add(E element) {
        Node<E> newnode = new Node<E>(element, first);
        first = newnode;
        return true;
    }

    public boolean contains(Object o) {
        for (Node<E> current = first; current != null; current = current.next) {
            if ((current.element == null && o == null)
                || (current.element != null && current.element.equals(o))) {
                return true;
            }
        }
        return false;
    }

    public SimpleIterator<E> iterator() {
        return new LinkedSimpleIterator<E>(first);
    }
}

8
For Reference: LinkedSimpleIterator class

class LinkedSimpleIterator<E> implements SimpleIterator<E> {
  private Node<E> current;

  public LinkedSimpleIterator (Node<E> first) {
    current = first;
  }

  public boolean hasNext() {
    return (current != null);
  }

  public E next() {
    if (current == null) {
      throw new java.util.NoSuchElementException();
    } else {
      E n = current.element;
      current = current.next;
      return n;
    }
  }
}

For Reference: Sample Stack/Heap Drawing

Stack

- coll1
- coll2
- c
- e

Heap

- LinkedSimpleCollection
  - first
- Node
  - element
  - next: null
- String
  - "foo"
- LinkedSimpleCollection
  - first
- Node
  - element
  - next: null
- String
  - "bar"