CSE 143 SUMMER 2010 FINAL EXAM SOLUTION (PART 1)

1. (12 points) **Using recursion**, write a method `mod` that takes two integers `a` and `b` as parameters and returns the remainder of `a` divided by `b`, *i.e.*, `a % b`. You may use `+`, `-`, comparison operators (`>`, `>=`, `<`, `<=`, `==`, `!=`), and Boolean operators (`&&`, `||`). You may NOT use `*`, `/`, `%`, any loops (*i.e.*, `for` or `while`) or any method from the Java library. You may assume that `a` is greater than or equal to zero and `b` is greater than zero.

   You may find the following equality useful: `a % b = (a - b) % b`

   ```
   public static int mod(int a, int b) {
      if (a < b) {
         return a;
      } else {
         return mod(a - b, b);
      }
   }
   ```

   A bunch of people did everything properly except to say "return" in the recursive call. That was worth 3 points.

2. (12 points) **Using recursion**, write a method `parity` that takes a string as a parameter and returns the string `"odd"` if the string has an odd number of characters and `"even"` if the string has an even number of characters. For example, the call `parity("abcdef")` will return `"even"`, while `parity("JS&")` will return `"odd"`. You may NOT use any loops (*i.e.*, `for` or `while`).

   You are limited to the following `String` methods (notice the absence of `length`):

   [METHODS AND EXAMPLES CUT]

   ```
   public static String parity(String s) {
      if (s.equals("")) {
         return "even";
      } else if (s.substring(1).equals("")) {
         return "odd";
      } else {
         return parity(s.substring(2));
   }
   ```

```
public static String parity(String s) {
   if (s.equals("")) {
      return "even";
   } else if (s.equals(s.substring(0,1)) {
      return "odd";
   } else {
      if (parity(s.substring(1)).equals("even")) {
         return "odd";
     } else {
         return "even";
      }
   }
}

public static String parity(String s) {
   if (s.equals("") ||
       parity(s.substring(1)).equals("odd")) {
      return "even";
   } else {
      return "odd";
   }
}

public static String parity(String s) {
   if (s.equals("")) {
      return "even";
   } else {
      String answer = parity(s.substring(1));
      if (answer.equals("even")) {
         return "odd";
      } else {
         return "even";
      }
   }
}
```

There are many more possible solutions.

3. (12 points) Recall the definitions of the `ListNode` and `LinkedIntList` class:

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

public class LinkedIntList {
     private ListNode front;

     <methods>
}
```

Write a method `interleave` for the `LinkedIntList` class that takes another `LinkedIntList` as a parameter and alternates the nodes of both lists. For example, if `list1` contained [1,2,3] and `list2` contained [4,5,6], then the call `list1.interleave(list2)` would result in `list1` becoming [1,4,2,5,3,6].

- You may assume that `list1` and `list2` initially have the same number of nodes.
- After a call `interleave`, you do NOT need to preserve the structure of the parameter list (i.e., `list2`).
- You may NOT create any new nodes or overwite the `data` variable of any node.
- You must solve this problem by manipulating the `next` variable in the `ListNode` class. Do NOT use recursion.

```
public void interleave(LinkedIntList other) {
   ListNode l1 = front;
   ListNode l2 = other.front;

   while (l1 != null && l2 != null) {
      ListNode temp1 = l1.next;
      ListNode temp2 = l2.next;
      l1.next = l2;
      l2.next = temp1;
      l1 = temp1;
      l2 = temp2;
   }
}
```

4. (12 points) Define a class `Box` for storing objects of the class `Item` (described below). Your class should include the following methods:

`add(Item)`	Adds the given item to this box.
`getWeight()`	Returns the weight of this box and contents. An empty box weighs 0.25 pounds. This method must run in O(1) time.

`getNumItems()`	Returns the number of items in the box.

The `Item` class (which you do NOT need to write) has the following accessor methods:

`getName()`	Returns the name of the item.
`getWeight()`	Returns the weight of the item.

Additionally, your `Box` class should implement the `Comparable<E>` interface. Lighter boxes should be considered "less" than heavier boxes.

- Your class should NOT have any unnecessary fields.
- Avoid redundancy where possible and use proper encapsulation.

```
public class Box implements Comparable<Box> {
   private List<Item> items;
   private double totalWeight;

   public Box() {
      items = new ArrayList<Item>();
      totalWeight = 0.25;
   }

   public void add(Item i) {
      items.add(i);
      totalWeight += i.getWeight();
   }

   public double getWeight() {
      return totalWeight;
   }

   public int getNumItems() {
      return items.size();
   }

   public int compareTo(Box other) {
      double otherWeight = other.getWeight();
      if (totalWeight == otherWeight) {
         return 0;
      } else if (totalWeight < otherWeight) {
         return -1;
      } else {
         return 1;
      }
   }
}
```