Assignment 7. Functional implementation of linked listed

Due on Sunday, 11 November, by midnight (submitted electronically). Download a skeleton class List from the assignment subpage of the course website. Class List MUST have the two constructors shown below.

Introduction. In this assignment, you will write a functional implementation of linked lists --none of your methods will use the assignment statement.

Setting the stage. Class List provides two fields: a value and the name of an instance of List. As usual, this can represent a linked list.

```
/** An instance is a linked list. Its value is field
element; the rest of the list is field next. An empty
list is represented by null */
public class List {
   public Object element;
   public List next;

    /** Constructor: an instance with one node,
        which contains the value null */
    public List() { element= null; next= null; }

    /** Constructor: a list with first element d and
                   rest of the list  n */
    public List(Object e, List n) {
      element= e;
      next= n;
    }

    // =  "o1 = o2" (they could both be null)
    public static boolean equals
                         (Object o1, Object o2) {
      if (o1==null  &&  o2==null)
          return true;
      if (o1==null  || o2==null)
          return false;
      return  o1.equals(o2);
    }

    // = l1 with l2 appended to it
    public static List append(List l1, List l2) {
        if (isEmpty(l1))
            return l2;
        return prepend(first(l1), append(rest(l1),l2));
    }
}
```

The first constructor is there for technical reasons. Don't worry about it.

Next, we define some primitive methods:

```
/** #1: = List l with x prepended to it */
public static List prepend(Object x, List l)

/** #2: = the element of the first node of l.
       Precondition: l is not null  */
public static Object first(List l)

/** #3:  = List l but with its first element removed */
public static List rest(List l)

/** #4:  = "list l is empty --i.e. null"  */
public static boolean isEmpty(List l)
```

Part I. Implement these methods (in class List) and test them until you are positive that they are correct. Do it incrementally. Write one at a time and test it. Remember: don't use an assignment statement.

Method append is a good one to look at to get the idea of writing methods in a functional manner.

Part II. Implement the methods that are described below (place them in class List). They should be static. They should be recursive; they should not use the assignment statement. Of course, you can use the methods of Part 1. And, use method equals (given above) to test equality of objects. THESE METHODS SHOULD NOT REFERENCE FIELDS element and next!! Also, each method better have a suitable specification.

5. printList(List l). Print all the elements of list l on a single line, with a blank after each element. For an empty list, print a blank line. Return **null**.

6. isMember(Object o, List l). Return a boolean: the value of "Object o is a member of list l".

7. deleteFirst(Object o, List l). Return a list that is like l except that the first occurrence of o has been deleted. If o does not occur in l, return a list with the same elements as l.

8. deleteAll(Object o, List l). Return a list that is like l except that all occurrences of o have been deleted. If o doesn't occur in l, return a list with the same elements as l.

9. reverse(List l). Return a list that is the reverse of l.

Please turn over and read the other side

Part III. A set is a collection of elements (but with no duplicates). We can implement a set in an instance of class List. In implementing a set in a List, we require:

• The List does not contains duplicates.

• The order of the elements in the List does not matter.

Write (in class List) the following methods --without using the assignment statement. They should be static.

Below, since the arguments of calls to these functions are sets, they do not contain duplicates and the order in which the values appear in them does not matter. If a method returns a List that represents a set, it should also satisfy these properties.

10. difference(List l1, List l2). Lists l1 and l2 contains sets. Return a list that represents the difference l1-l2 of these two sets: this is the set that contains the elements of l1 that are not in l2. For example, the difference of sets {2,5,3} and {3,6,4}, written as {2,5,3} - {3,6,4} is the set {2,5}.

11. union(List 1, List l1). Lists l1 and l2 contain sets. Return a list that represents the union of these two sets: this is the set of all elements that are in at least one of l1 and l2.

12. intersection(List 1, List l1). Lists l1 and l2 contain sets. Return a list that represents the intersection of these two sets: this is the set of all elements that are in both l1 and l2.

Submitting your assignment

At the top of your file List.java, put a comment that contains your name an netid.

Submit your assignment electronically, as Assignment 7. Submit a folder that contains ONLY file List.java.

Again. Submit a folder that contains ONLY a file List.java.

We don't want to see anything else.

While this is not mandatory, it will help us if you put the 12 methods that you have to write in the order in which they were presented.

We will preselect 5 of the 12 methods that you have to write to test. The same 5 methods will be used for all students. For each method, your grade will depend on:

(1) A good Javadoc specification for the method. If you still don't know what a good specification is (which means that you haven't been listening or studying), look in the grading guide on the web page.

(2) Whether the method works on our test cases. We will try all sorts of test cases, e.g. empty lists, lists with 1 element, lists with 2 elements, etc. So be sure you test your method well.

(3) The presentation of your method --is it well indented, etc.

In our sample solution, the bodies of the 12 methods to be written take under 50 lines. So the average number of lines per method body is just over 4. Work on one method at a time, testing it well before you move on to the next, and this assignment should not take too much time.