COSC 237

Assignment # 4

DUE DATE: 12/07/2021 (Tuesday)

For this assignment, you will work in teams (as assigned), but all teammates should contribute seriously - please let me know if that's not the case. In principle, you should not discuss the homework with anyone, except your partners and me. If you do discuss it with someone else (better don't), you should say this in the preamble of the assignment. You should only submit work that is completely your own and you should be able to explain your solutions if asked to do so.
Same type of submission as in previous assignments, both email and hard copy.
You have one programming problem. Make sure that you test it with my client. Email the class DoubleLinkedList and 2 different clients (with 2 different types of objects; for example, DoubleLinkedList<Integer> in one client and DoubleLinkedList<Time> in another client.) Make sure you handle **invalid input**, **including type mismatch** - input validation is a requirement! Focus on **good design**, **good style (**the format of your programs is very important and bad style will be penalized with up to 50%), and, most importantly, **reuse of code**. Start early and remember the rules about late assignments/syllabus → **no late assignments!**

NOTE: Try to use this quote as a guide when writing code (I do!): *"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."*
--- Martin Fowler, Refactoring: Improving the Design of Existing Code

HONORS STATEMENT: "I will not cheat, fabricate materials, plagiarize, or help another to undertake such acts of academic dishonesty, nor will I protect those who engage in acts of academic dishonesty. I pledge on my honor as a student, that I have not received or provided any unauthorized help on this assignment."

This is the last assignment. Make it your best.

Doubly linked list programming assignment: Rewrite the implementation of a sorted list using a doubly linked list. Use as a guide the implementation with a single linked list (see lecture notes).

Definition: A doubly linked list is a list in which each node is linked to both its successor and its predecessor.

Example:

In this definition, a node has 2 link members; one is a reference to the next node in the list--immediate successor, and the other is a reference to the previous node in the list--immediate predecessor. The first node in the list has no predecessor and the last node has no successor. The advantage of using a double linked list is that the list can be traversed from first node to last, as any linked list, AND ALSO from last node to first (very convenient when we need to traverse a list backwards). The disadvantage is the extra storage required for the second reference in each node. Also, the inserting and deleting operations are a bit slower (both references have to be updated).

Hint: Read the Doubly Linked List section, pg. 1072 - 1086 [Malik]

Operations to be implemented on a double linked list:

1. Initialize the list
2. Determine whether the list is empty
3. Retrieve the first element in the list
4. Retrieve the last element in the list
5. Find the length of the list
6. Display the list (direct and reverse order)
7. Search the list for a given item
8. Insert an item in the list
9. Delete an item from the list
10. toString (direct and reverse order, 2 versions for each, one using recursion)
11. Compare 2 lists for equality (equals)
12. Make a copy of the list
13. Make a copy of the list in reverse order

Use this interface:

```
public interface DoubleLinkedListADT<T>  {
    public void initializeList();
    public boolean isEmptyList();
    public T front();
    public T back();
    public int length();
    public void print();
    public void reversePrint();
    public boolean search(T searchItem);
    public void insertNode(T insertItem);
    public void deleteNode(T deleteItem);
    public String toString();
    public String recursiveToString();
    public String backwardsString();
    public String recursiveBackwardsString();
    public boolean equals(Object o);
    public void copy(DoubleLinkedList<T> otherList);
    public void reversedCopy(DoubleLinkedList<T> otherList);
}
```

Complete this class definition:

```
public class DoubleLinkedList<T> implements DoubleLinkedListADT<T> {
    //Double linked list node class
    public class DoubleLinkedListNode<T>  {
        T info;
        DoubleLinkedListNode<T> next;
        DoubleLinkedListNode<T> back;

        public DoubleLinkedListNode() {
            info = null;
            next = null;
            back = null;
        }

        public String toString() {
            return info.toString();
        }
    }
```

```
    protected int count;  //number of nodes
    protected DoubleLinkedListNode<T> first; //reference to first node
    protected DoubleLinkedListNode<T> last;  //reference to last node
    ...
    ...
    ...
}
```

Use this client program to do the testing (full testing → these values, and more; change the data, check if you get the expected output):

```
public class Client_DLLString {
    public static void main(String[] args) {
       DoubleLinkedList<String>list_1 = new DoubleLinkedList<String>();
       DoubleLinkedList<String>list_2 = new DoubleLinkedList<String>();
       String item;
       list_1.insertNode("John");
       list_1.insertNode("Ann");
       list_1.insertNode("Paul");
       list_1.insertNode("Joshua");
       list_1.insertNode("Will");
       list_1.insertNode("Emma");
       list_1.insertNode("Peter");
       list_1.insertNode("Linda");
       list_1.insertNode("Joan");
       list_1.insertNode("David");
       list_1.insertNode("Miriam");
       list_1.insertNode("Leah");
       list_1.insertNode("Jane");
       System.out.println("Inserted in first list the names: John, Ann, Paul, Joshua,
Will, Emma, Peter, Linda, Joan, David, MIriam, Leah, Jane");
       System.out.println("(Testing toString) First list sorted is: " + list_1);
       System.out.println("(Testing recursive toString) First list sorted is: [head] - "
+ list_1.recursiveToString());
       System.out.println("(Testing backwards) First list reversed (print) is: " +
list_1.backwardsString());
       System.out.print("(Testing recursive backwards) First list reversed (print) is: "
+ list_1.recursiveBackwardsString());
       System.out.println(" - [head]");
       System.out.println("Will copy in second list only names that don't start with J.
List one destroyed.");
       while (!list_1.isEmptyList()) {
           item = list_1.front();
           list_1.deleteNode(item);
           if(item.charAt(0) != 'J')
               list_2.insertNode(item);
       }
       System.out.println("Second list should hold names that don't start with J
(sorted): " + list_2);
       System.out.println("First list should be empty. Nothing printed. ");
       list_1.print();
       System.out.print("(Testing equals) ");
       if(list_1.equals(list_2))
           System.out.println("The 2 lists are equals");
       else
           System.out.println("The 2 lists are NOT equals");
       System.out.print("(Testing copy) ");
       list_1.copy(list_2);
       System.out.println("First list after copy is: " + list_1);
       System.out.print("(Testing reversed copy) ");
       list_1.reversedCopy(list_2);
       System.out.println("First list as the copy of the second list reversed is: " +
```

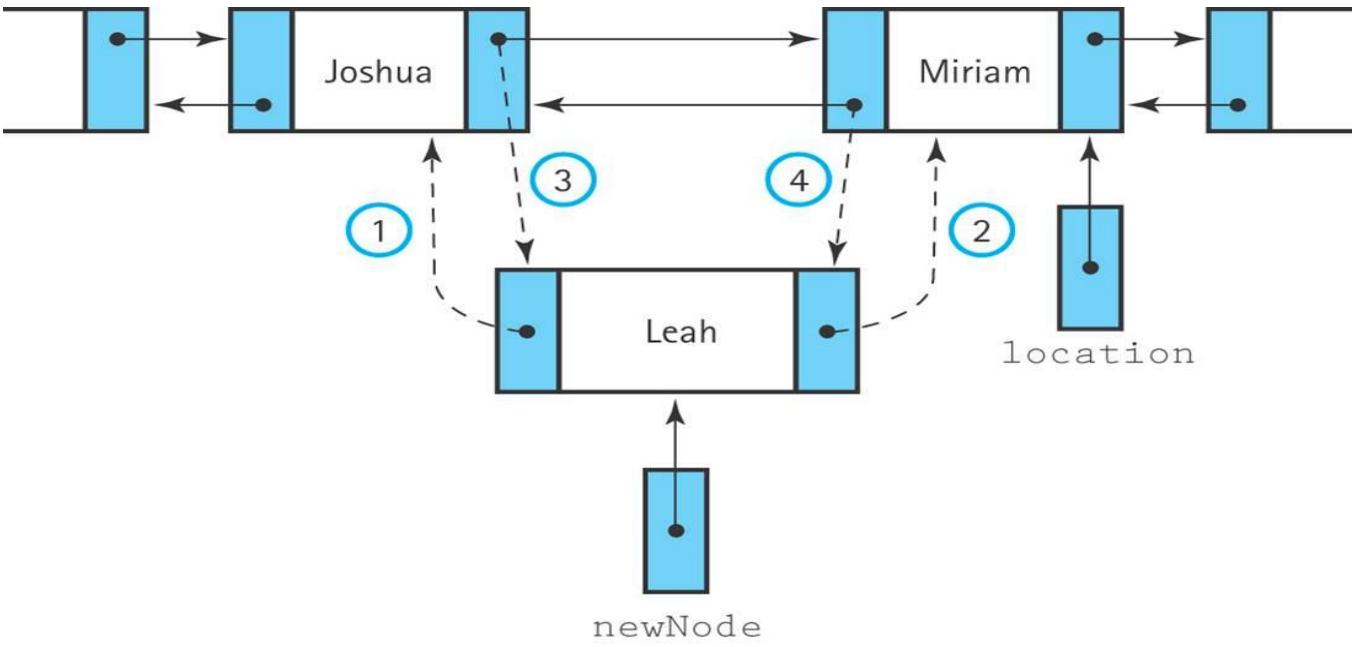
```
list_1);
    }
}
```

OUTPUT:

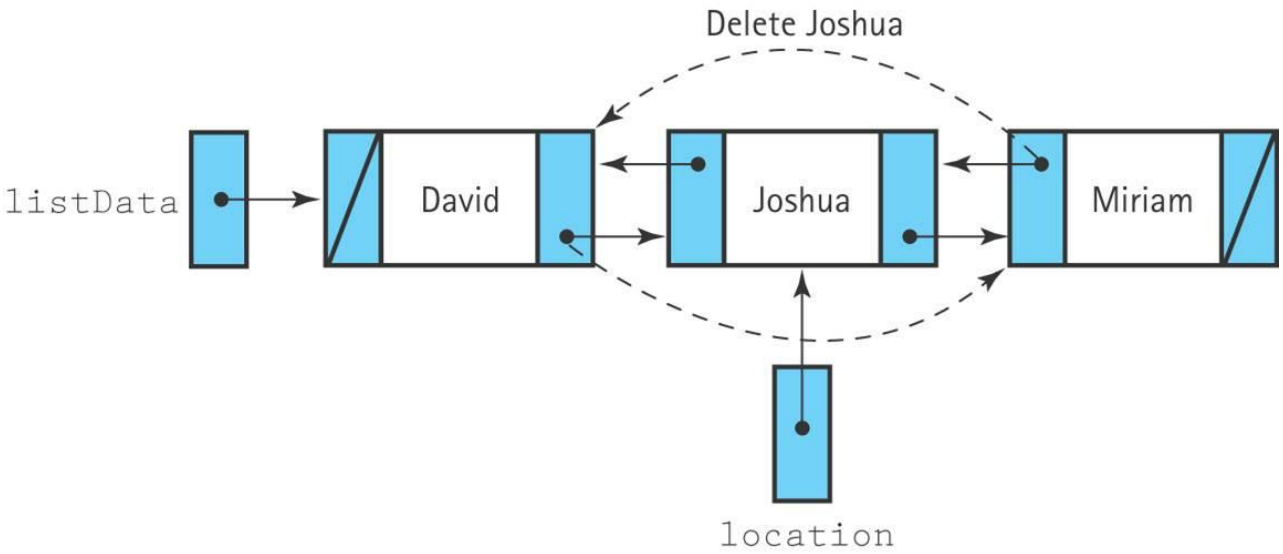
```
Inserted in first list the names: John, Ann, Paul, Joshua, Will, Emma, Peter, Linda,
Joan, David, Miriam, Leah, Jane
(Testing toString) First list sorted is: [head] - Ann - David - Emma - Jane - Joan - John
- Joshua - Leah - Linda - Miriam - Paul - Peter - Will - [tail]
(Testing recursive toString) First list sorted is: [head] - Ann - David - Emma - Jane -
Joan - John - Joshua - Leah - Linda - Miriam - Paul - Peter - Will - [tail]
(Testing backwards) First list reversed (print) is: [tail] - Will - Peter - Paul - Miriam
- Linda - Leah - Joshua - John - Joan - Jane - Emma - David - Ann - [head]
(Testing recursive backwards) First list reversed (print) is: [tail] - Will - Peter -
Paul - Miriam - Linda - Leah - Joshua - John - Joan - Jane - Emma - David - Ann - [head]
Will copy in second list only names that don't start with J. List one destroyed.
Second list should hold names that don't start with J (sorted): [head] - Ann - David -
Emma - Leah - Linda - Miriam - Paul - Peter - Will - [tail]
First list should be empty. Nothing printed.
(Testing equals) The 2 lists are NOT equals
(Testing copy) First list after copy is: [head] - Ann - David - Emma - Leah - Linda -
Miriam - Paul - Peter - Will - [tail]
(Testing reversed copy) First list as the copy of the second list reversed is: [head] -
Will - Peter - Paul - Miriam - Linda - Leah - Emma - David - Ann - [tail]
```

Hints:

Take a look at the following picture when you write the method to insert a node:



Take a look at the following picture when you write the method to delete a node:



---

[1] Java Programming: From Problem Analysis to Program Design, by D.S. Malik, Thomson Course Technology
[2] C++ Plus Data Structures, by Nell Dale, Jones and Bartlett Publishers.