CSCI 1101 – Winter 2017
Lab. No. 7

This lab is on ArrayLists and an introduction to LinkedLists.

Notes:

1. *All submissions must be made on Brightspace (dal.ca/brightspace).*
2. ***Submission deadline is 11.55 p.m. (5 minutes to midnight) on SUNDAY, March 12, 2017. (Note: Only for this week, the submission deadline is a Sunday).***
3. *Put the java source code files and the text outputs for each exercise in a folder. Zip the folder into one file and submit the zip file.*

4. Marking Scheme:
Each exercise carries 10 points. Your final score will be scaled down to a value out of 10.
Working code, Outputs included, Efficient, Comments included: 10/10
No comments: subtract one point
Unnecessarily long code and inefficient program, improper use of variables: subtract one point
No outputs: subtract two points
Code not working: subtract up to six points depending upon the extent to which the program is incorrect.

***5. Error checking**: Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input.*

***6. Testing your code and generating the outputs:** If the test data is included in the question, use that to test your classes. In addition, test it with two more input sets. Otherwise, create your own test data and run your program for at least 3 input sets such that they cover the range of results expected.*

ARRAYLISTS

Exercise 1: Write a program that reads words into an ArrayList `list1` and then removes all the duplicates in `list1`.

A sample dialog is shown below:

```
Enter words on one line, end with -1
java c pascal ada java java ada c++ -1

Here's the original list:
[java, c, pascal, ada, java, java, ada, c++]

Here's the same list with no duplicates:
[java, c, pascal, ada, c++]
```

You need to do this efficiently, for example, without creating another ArrayList. Note that it can be tricky to scan an ArrayList and simultaneously remove items from it. This is due to the fact that when you remove or add an item, the list adjusts itself and hence the index used for scanning will not be pointing to the same location.

Exercise 2:

The objective of this exercise is to write a program that reads a number of Strings and sorts them by inserting each string into the appropriate place in an arraylist. For example, if the strings are:

Shai
Ralph
Hillary
Tom
Barbara
Fred

Then the arraylist should grow as follows:
[Empty]
[Shai]
[Ralph, Shai]
[Hillary, Ralph, Shai]
[Hillary, Ralph, Shai, Tom]
[Barbara, Hillary, Ralph, Shai, Tom]
[Barbara, Fred, Hillary, Ralph, Shai, Tom]

The algorithm to sort is simple. As you read each name (say name1), compare it with each name (say name2) stored in the arraylist starting from the index 0. If (name1.compareTo(name2) <0), you found the right index to put name1. Be sure that you do not cross the ArrayList boundary.

LINKED LISTS

Before you solve Exercises 3 and 4, test the Node class and the LinkedList class given below with the LinkedListDemo1 class (also given below). Understand the basics of Linked Lists.

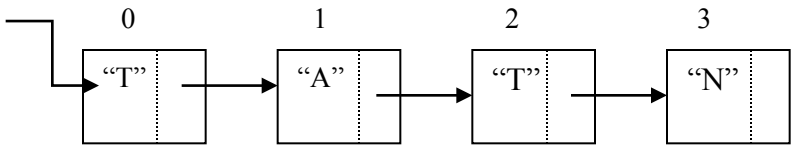
```
//class Node
public class Node{
   private String data;
   private Node next;
   public Node(String d, Node n){
      data = d;
      next = n;
   }
   public String getData(){
      return data;
   }
   public Node getNext(){
      return next;
   }
   public void setData(String d){
      data = d;
   }
   public void setNext(Node n){
      next = n;
   }
   public String toString(){
      return data + "-->";
   }
}

//class LinkedList (only the first few methods are given here)
public class LinkedList{
   private Node front;
   private int count;
```

```
   //constructor
   public LinkedList(){
      front = null;
      count = 0;
   }

   //add a node to the front of the linked list
   public void addToFront(String d){
      Node n;
      n = new Node(d, front);
      front = n;
      count++;
   }

   //get the current size of the list
   public int size(){
      return count;
   }

   //check if the list is empty
   public boolean isEmpty(){
      return (count==0);
   }

   //clear the list
   public void clear(){
      front = null;
      count=0;
   }

   //get the content of the first node
   public String getFrontData()
   {
      if (front==null)
         return "Empty list";
      else
         return front.getData();
   }

   //new method added - get the first node
   public Node getFront()
   {
     return front;
   }

   //scan the list and print contents
   public void enumerate()
   {
      Node curr = front;
      while (curr!=null)
      {
         System.out.print(curr);
         curr = curr.getNext();
     }
      System.out.println(".");
  }
}

//class LinkedListDemo1
public class LinkedListDemo1
{
   public static void main(String[] args)
   {
     LinkedList myList = new LinkedList();
      myList.addToFront("Toronto");
```

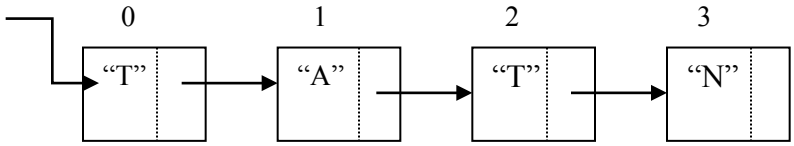
```
        myList.addToFront("New York");
       myList.addToFront("Calgary");
        myList.addToFront("Halifax");
        myList.addToFront("St.John's");
        System.out.println("Number of nodes in the list: "+ myList.size());
        myList.enumerate();
        myList.addToFront("Vancouver");
        myList.addToFront("Montreal");
        myList.enumerate();
    }
}
```

Exercise 3: Add the following method to LinkedList.java.
Method `public void enumerateOddNodes()` that prints the contents of all the nodes with odd indices. For example, for the list shown below, the method should display
A → N →

Test your method by including appropriate test statements to your demo program.



Exercise 4: Add the following method to LinkedList.java.
Method `public void listAllNodesWith(String d)` that displays the indices of all the nodes that have the String d in them. For example, for the list shown below, the method listAllNodesWith(“T”) should display
0 2



Test your method by including appropriate test statements to your demo program.