```
::::::::::::::
Driver.java
::::::::::::::
/*

 * Purpose: Data Structure and Algorithms Lab 3 Problem 1

 * Status: Complete and thoroughly tested

 * Last update: 02/11/20

 * Submitted:  02/11/20

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.02.11

 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.LinkedList;

public class Driver {
    static BufferedReader stdin = new BufferedReader (new InputStreamReader(System
.in));

    public static void main(String[] args) throws NumberFormatException, IOExcepti
on {


        LinkedList<Object> items = new LinkedList<Object>();
        boolean switchOn = true;

        System.out.println("\nSelect from the following menu:"
                           + "\n\t1. Insert item to list\n\t2. Remove item from li
st"
                           + "\n\t3. Get item from list"
                           + "\n\t4. Clear list\n\t5. Print size and content of li
st"
                           + "\n\t6. Delete largest item in the list."
                           + "\n\t7. Reverse list" + "\n\t8. Exit program");

        while(switchOn == true)
        {
            System.out.print("\nMake your selection now: ");
            int selection = Integer.parseInt(stdin.readLine().trim());
            System.out.println(selection);

            switch(selection)
            {

            /*
             * Case 0 exists solely to test Problem 3.
             * It takes input from the Command Line, then
             * prints
             */
            case 0:
                System.out.print("1st String to test: ");
                String test1 = stdin.readLine();
                System.out.print(test1 + "\n2nd String to test: ");
                String test2 = stdin.readLine();
                System.out.println(test2);

                System.out.println("\n" + test1 + " compared to " + test2 + ": "
                                   + test1.compareTo(test2));

                break;

            case 1:
                System.out.print("\nYou are now inserting an item into the list.\n
\tEnter item: ");
                Object item = stdin.readLine().trim();
                System.out.println(item);

                System.out.print("\tEnter position to insert item in: ");
                int index = Integer.parseInt(stdin.readLine().trim());
                System.out.println(index);

                if(index >= items.size()+1)
                {
                    System.out.println("\nPosition specified is out of range!");
                }
                else
                {
                    items.add(index, item);
                    System.out.println("\nItem " + item + " inserted at position "
 + index + " in the list.");
                }
                break;
            case 2:
                System.out.print("\tEnter position to remove item from: ");
                int toRemove = Integer.parseInt(stdin.readLine().trim());
                System.out.println(toRemove);

                if((toRemove >= items.size()) || (toRemove < 0))
                {
                    System.out.println("\nPosition specified is out of range!");
                }
                else
                {
                    System.out.println("\nItem " + items.get(toRemove) + " removed
 from position " + toRemove + " in the list.");
                    items.remove(toRemove);
                }
                break;
            case 3:
                System.out.print("\t\nEnter position to retrieve item from: ");
                int toRetrieve = Integer.parseInt(stdin.readLine().trim());
                System.out.println(toRetrieve);

                if((toRetrieve >= items.size()) || (toRetrieve < 0))
                {
                    System.out.println("\nPosition specified is out of range!");
                }
                else
                {
                    System.out.println("\nItem " + items.get(toRetrieve) + " retri
eved from position " + toRetrieve + " in the list.");
                }
                break;
```

```
            case 4:
                items.removeAll(items);
                break;
            case 5:

                if(items.size() == 0)
                {
                    System.out.println("List is empty.");
                }
                else
                {
                    System.out.print("List of size " + items.size() + " has the fo
llowing items: ");

                    for(int i = 0; i < items.size(); i++)
                    {
                        System.out.print(items.get(i) + " ");
                    }
                }
                break;

            case 6:
                if(items.size() == 0)
                {
                    System.out.println("List empty, nothing to delete!");
                }
                else
                {
                    String compare = "";
                    int removeIndex = 0;
                    for(int i = 0; i < items.size(); i++)
                    {
                        if(items.get(i).toString().compareTo(compare) > 0)
                        {
                            compare = items.get(i).toString();
                            removeIndex = i;
                        }
                    }
                    System.out.println("Largest item " + items.get(removeIndex).to
String() + " deleted.");
                    items.remove(removeIndex);
                }
                break;

            case 7:

                if(items.size() == 0)
                {
                    System.out.println("List is empty... nothing to reverse!");
                }
                else
                {

                    for(int i = 0, k = items.size()-1; i < items.size()/2; i++, k-
-)
                    {
                        Object toFront = items.get(k);
                        Object toBack = items.get(i);

                        items.add(i, toFront);
                        items.remove(i+1);
                        items.add(k, toBack);
                        items.remove(k+1);
                    }

                    System.out.println("List reversed");
                }

                break;

            case 8:
                switchOn = false;
                System.out.print("Exiting program...Good Bye");
                break;


            default:

                break;
            }
        }

    }
}
```

```
::::::::::::::
ListIndexOutOfBoundsException.java
::::::::::::::
public class ListIndexOutOfBoundsException
    extends IndexOutOfBoundsException
{
    public ListIndexOutOfBoundsException(String s)
    {
        super(s);
    }  // end constructor
}  // end ListIndexOutOfBoundsException::::::::::::::
ListInterface.java
::::::::::::::
// ****************************************************
// Interface ListInterface for the ADT list.
// ****************************************************
public interface ListInterface
{
    boolean isEmpty();
    int size();
    void add(int index, Object item) throws ListIndexOutOfBoundsException;
    Object get(int index) throws ListIndexOutOfBoundsException;
    void remove(int index) throws ListIndexOutOfBoundsException;
    void removeAll();
    String toString();
}  // end ListInterface::::::::::::::
MyListReferenceBased.java
::::::::::::::
/*

 * Purpose: Data Structure and Algorithms Lab 3 Problem 1

 * Status: Complete and thoroughly tested

 * Last update: 02/11/20

 * Submitted:  02/11/20
```

```
 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.02.11

 */

// Please note that this code is slightly different from the textbook code
//to reflect the fact that the Node class is implemented using data encapsulation


// ************************************************
// Reference-based implementation of ADT list.
// ************************************************
public class MyListReferenceBased implements ListInterface
{
    // reference to linked list of items
    private Node head;

    public MyListReferenceBased()
    {
        head = null;
    }  // end default constructor

    public boolean isEmpty()
    {
        return size() == 0;
    }  // end isEmpty

    public int size()
    {
        int numItems = 0;
        Node next = head;

        while(next != null)
        {
            numItems++;
            next.getNext();
        }
        return numItems;
    }  // end size

    private Node find(int index)
    {
        // --------------------------------------------------
        // Locates a specified node in a linked list.
        // Precondition: index is the number of the desired
        // node. Assumes that 0 <= index <= numItems
        // Postcondition: Returns a reference to the desired
        // node.
        // --------------------------------------------------
        Node curr = head;
        for (int skip = 0; skip < index; skip++)
        {
            curr = curr.getNext();
        } // end for
        return curr;
    } // end find

    public Object get(int index)
    throws ListIndexOutOfBoundsException
    {
        if (index >= 0 && index < size())
        {
            // get reference to node, then data in node
            Node curr = find(index);
            Object dataItem = curr.getItem();
            return dataItem;
        }
        else
        {
            throw new ListIndexOutOfBoundsException(
                "List index out of bounds exception on get");
        } // end if
    } // end get

    public void add(int index, Object item)
    throws ListIndexOutOfBoundsException
    {
        if (index >= 0 && index < size()+1)
        {
            if (index == 0)
            {
                // insert the new node containing item at
                // beginning of list
                Node newNode = new Node(item, head);
                head = newNode;
            }
            else
            {
                Node prev = find(index-1);
                // insert the new node containing item after
                // the node that prev references
                Node newNode = new Node(item, prev.getNext());
                prev.setNext(newNode);
            } // end if
        }
        else
        {
            throw new ListIndexOutOfBoundsException(
                "List index out of bounds exception on add");
        } // end if
    }  // end add

    public void remove(int index)
    throws ListIndexOutOfBoundsException
    {
        if (index >= 0 && index < size())
        {
            if (index == 0)
            {
                // delete the first node from the list
                head = head.getNext();
            }
            else
            {
                Node prev = find(index-1);
                // delete the node after the node that prev
                // references, save reference to node
                Node curr = prev.getNext();
                prev.setNext(curr.getNext());
            } // end if
```

```
        } // end if
        else
        {
            throw new ListIndexOutOfBoundsException(
                "List index out of bounds exception on remove");
        } // end if
    }   // end remove

    public void removeAll()
    {
        // setting head to null causes list to be
        // unreachable and thus marked for garbage
        // collection
        head = null;
    } // end removeAll

    public String toString()
    {
        Node next = head;
        StringBuilder builder = new StringBuilder();
        String toReturn = "";

        while(next != null)
        {
            String name = next.getItem().toString() + " ";
            builder.append(name);
            next.getNext();
        }
        toReturn = builder.toString();

        return toReturn;
    }

} // end ListReferenceBased::::::::::::::
Node.java
:::::::::::::::
//please note that this code is different from the textbook code, because the data
 is encapsulated!

public class Node
{
    private Object item;
    private Node next;

    public Node(Object newItem)
    {
        item = newItem;
        next = null;
    } // end constructor

    public Node(Object newItem, Node nextNode)
    {
        item = newItem;
        next = nextNode;
    } // end constructor

    public void setItem(Object newItem)
    {
        item = newItem;
    } // end setItem

    public Object getItem()
    {
        return item;
    } // end getItem

    public void setNext(Node nextNode)
    {
        next = nextNode;
    } // end setNext

    public Node getNext()
    {
        return next;
    } // end getNext

} // end class Node::::::::::::::
output.txt
::::::::::::::

Select from the following menu:
        1. Insert item to list
        2. Remove item from list
        3. Get item from list
        4. Clear list
        5. Print size and content of list
        6. Delete largest item in the list.
        7. Reverse list
        8. Exit program

Make your selection now: 5
List is empty.

Make your selection now: 6
List empty, nothing to delete!

Make your selection now: 7
List is empty... nothing to reverse!

Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Data
        Enter position to insert item in: 0

Item Data inserted at position 0 in the list.

Make your selection now: 5
List of size 1 has the following items: Data
Make your selection now: 7
List reversed

Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Beverly
        Enter position to insert item in: 0

Item Beverly inserted at position 0 in the list.

Make your selection now: 5
List of size 2 has the following items: Beverly Data
Make your selection now: 1
```

```
You are now inserting an item into the list.
        Enter item: Jean-Luc
        Enter position to insert item in: 5

Position specified is out of range!

Make your selection now: 5
List of size 2 has the following items: Beverly Data
Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Jean-Luc
        Enter position to insert item in: 2

Item Jean-Luc inserted at position 2 in the list.

Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Geordi
        Enter position to insert item in: 2

Item Geordi inserted at position 2 in the list.

Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Worf
        Enter position to insert item in: 3

Item Worf inserted at position 3 in the list.

Make your selection now: 5
List of size 5 has the following items: Beverly Data Geordi Worf Jean-Luc
Make your selection now: 7
List reversed

Make your selection now: 7
List reversed

Make your selection now: 6
Largest item Worf deleted.

Make your selection now: 5
List of size 4 has the following items: Beverly Data Geordi Jean-Luc
Make your selection now: 7
List reversed

Make your selection now: 7
List reversed

Make your selection now: 2
        Enter position to remove item from: 9

Position specified is out of range!

Make your selection now: 2
        Enter position to remove item from: 3

Item Jean-Luc removed from position 3 in the list.

Make your selection now: 5
List of size 3 has the following items: Beverly Data Geordi
Make your selection now: 2
        Enter position to remove item from: 0

Item Beverly removed from position 0 in the list.

Make your selection now: 1

You are now inserting an item into the list.
        Enter item: Will
        Enter position to insert item in: 0

Item Will inserted at position 0 in the list.

Make your selection now: 5
List of size 3 has the following items: Will Data Geordi
Make your selection now: 3

Enter position to retrieve item from: 2

Item Geordi retrieved from position 2 in the list.

Make your selection now: 3

Enter position to retrieve item from: 0

Item Will retrieved from position 0 in the list.

Make your selection now: 3

Enter position to retrieve item from: 8

Position specified is out of range!

Make your selection now: 5
List of size 3 has the following items: Will Data Geordi
Make your selection now: 6
Largest item Will deleted.

Make your selection now: 5
List of size 2 has the following items: Data Geordi
Make your selection now: 6
Largest item Geordi deleted.

Make your selection now: 5
List of size 1 has the following items: Data
Make your selection now: 4

Make your selection now: 5
List is empty.

Make your selection now: 7
List is empty... nothing to reverse!

Make your selection now: 8
Exiting program...Good Bye::::::::::::::
P3_Output.txt
:::::::::::::::
LAB 3 PROBLEM 3 OUTPUT SAMPLES + WRITE-UP

Select from the following menu:
        1. Insert item to list
```

```
        2. Remove item from list
        3. Get item from list
        4. Clear list
        5. Print size and content of list
        6. Delete largest item in the list.
        7. Reverse list
        8. Exit program

Make your selection now: 0
1st String to test: a
2nd String to test: a

a compared to a: 0

Make your selection now: 0
1st String to test: a
2nd String to test: b

a compared to b: -1

Make your selection now: 0
1st String to test: !
2nd String to test: a

! compared to a: -64

Make your selection now: 0
1st String to test: ~
2nd String to test: !

~ compared to !: 93

Make your selection now: 0
1st String to test: A
2nd String to test: a

A compared to a: -32

Make your selection now: 0
1st String to test: a
2nd String to test: A

a compared to A: 32

Make your selection now: 0
1st String to test: aeiou
2nd String to test: aeeiou

aeiou compared to aeeiou: 4

Make your selection now: 0
1st String to test: Aeiou
2nd String to test: Aeeiou

Aeiou compared to Aeeiou: 4

Make your selection now: 0
1st String to test:
2nd String to test: a

  compared to a: -65

Make your selection now: 0
1st String to test: -
2nd String to test: +

- compared to +: 2

Make your selection now: 0
1st String to test: -
2nd String to test: =

- compared to =: -16

Make your selection now: 0
1st String to test: +
2nd String to test: =

+ compared to =: -18

Make your selection now: 8
Exiting program...Good Bye
```

---

For sake of ease of testing, I wrote a simple Case 0 in my Driver that took in 2 S
tring values, then compared them. (i.e. StringA.compareTo(StringB)).

Through **this** process, I was able to ascertain a couple of things, although my test
ing is not 100% thorough and concrete findings would require more examples to
run through.

For one, the larger the gap between two characters, the 'bigger' the output - ! an
d a have a massive 64 characters between them, **for** instance.

Additionally, and **while** admidtedly obvious, compareTo is **case** sensitive. If you co
mpare a and A, you get -32 or 32 depending on the order you compare to represent t
he 32 spaces between them.

Something interesting that I don't know how to explain, exactly, is how it determi
ned that **"aeeiou"** and **"Aeeiou"** are bigger than **"aeiou"** and **"Aeiou"** respectively. T
hey both gave an output of 4, which tells me that lexographically, they're further
 down.

Further experiments required to figure that out.

What caught me by surprise is that lexographically, the special characters aren't
*after* the alphabet but before them based off the output.

Stranger still is that the blank space created by hitting the spacebar is one char
acter before the exclamation point.

There's a lot I still don't understand about compareTo, which leads to me to concl
ude that I need to **do** more testing regarding compareTo.

However, something to definitely take going forward is that **if** I want to compare t
wo strings lexographically ***while** ignoring **case*** is to use the ignore **case** version
of compareTo.

```
::::::::::::::
Lab3Conclusions.txt
::::::::::::::
LAB 3 CONCLUSIONS
```

Lab 3 was interesting – it really sold how inefficient Arrays and ArrayLists can b
e, even when they **do** have their uses occasionally.

I've never had to reverse a collection myself, honestly, or at least never a
LinkedList. Figuring that out was cool but I feel like there's a much, MUCH
more efficient way of going about that.

My greatest takeaway is that LinkedLists are slightly more complex to work with
than an Array but the benefits are much greater. Additionally, **if** you can get
the Collection working with an Array/ArrayList, its incredibly easy to convert
it over to a LinkedList by minor tweaks.