```
::::::::::::::
Bag.java
::::::::::::::

public class Bag <T> {
    private int packages;
    private float weight;
    private StackInterface<T> bag;

    public Bag()
    {
        bag = new StackSLS<T>();
        packages = 0;
        weight = 0;
    }

    public void pickUpOrder(Package pack)
    {
        bag.push((T) pack);
        packages++;
        weight += (pack.getItemWeight() * pack.getItemAmount());

    }


    public void displayPackageBag()
    {
        System.out.println("Bag has " + packages + " packages and weighs " + weigh
t + " lbs.");
    }

    public void displaySampleBag(Bag<T> samples)
    {
        System.out.println("Bag has " + samples.packages + " packages and weighs "
 + samples.weight + " lbs.");
    }



    public StackInterface<T> getBag()
    {
        return bag;
    }

    public int getPackages()
    {
        return packages;
    }

    public float getWeight()
    {
        return weight;
    }

    public void setPackages(int newPackages)
    {
        packages = newPackages;
    }

    public void setWeight(float newWeight)
    {
        weight = newWeight;
    }

}
::::::::::::::
Driver.java
::::::::::::::

/*

 * Purpose: Data Structure and Algorithms Lab 5 Problem 2

 * Status: Complete and thoroughly tested

 * Last update: 02/24/20

 * Submitted:  02/24/20

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.02.24

 */

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


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

    public static <T> void main(String[] args) throws NumberFormatException, IOExc
eption {


        Bag<T> bag = new Bag<T>();
        Bag<T> samples = new Bag<T>();

        boolean switchOn = true;

        System.out.println("\nSelect from the following menu:"
                           + "\n\t0. Exit.\n\t1. Pick up an order."
                           + "\n\t2. Drop off an order."
                           + "\n\t3. Display number of packages and weight of bag.
"
                           + "\n\t4. Display number of items and weight of the bag
 of samples."
                           + "\n\t5. Enjoy an item from the bag of samples."
                           + "\n\t6. Enjoy all the samples in the bag of samples.\
r\n");

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

            case 1:

                String name = "";
                String sender = "";
                String recipient = "";
                float weight = 0;
                int amount = 0;

                System.out.println("Please specify info: ");

                System.out.print("Item name: ");
                name = stdin.readLine().trim();
                System.out.println(name);

                System.out.print("\nItem weight: ");
                weight = Float.parseFloat(stdin.readLine().trim());
                System.out.println(weight);

                System.out.print("\n# of items: ");
                amount = Integer.parseInt(stdin.readLine().trim());
                System.out.println(amount);

                System.out.print("\nSender: ");
                sender = stdin.readLine().trim();
                System.out.println(sender);

                System.out.print("\nRecipient: ");
                recipient = stdin.readLine().trim();
                System.out.println(recipient);

                Package pack = new Package(name, weight, amount, sender, recipient
);
                bag.pickUpOrder(pack);
                System.out.println("A package of " + name + " each weighing " + we
ight + " lbs are now in the bag.");
                break;
            case 2:

                if(bag.getBag().isEmpty())
                {
                    System.out.println("No deliveries to process!");
                }
                else
                {
                    Package dropOff = (Package) bag.getBag().pop();


                    float newWeight = bag.getWeight() - (dropOff.getItemWeight() *
 dropOff.getItemAmount());
                    int newPackages = bag.getPackages()-1;

                    bag.setWeight(newWeight);
                    bag.setPackages(newPackages);


                    System.out.print("Here is your package " + dropOff.getItemRece
iver() + ". May I keep a sample (Y/N)? ");
                    String response = stdin.readLine();
                    System.out.print(response);

                    while(!((response.toUpperCase().equals("Y") || (response.toUpp
erCase().equals("N")))))
                    {
                        System.out.print("Please say (Y)es or (No)! ");
                        response = stdin.readLine().trim();
                        System.out.println(response);
                    }

                    System.out.println("\nYour package contains: ");

                    if(dropOff.getItemAmount() == 1)
                    {
                        System.out.println("A " + dropOff.getItemName() + " weighi
ng " + dropOff.getItemWeight()
                                                + " from " + dropOff.getItemSender() +
" to " + dropOff.getItemReceiver());
                    }
                    else
                    {
                        System.out.println(dropOff.getItemAmount() + " " + dropOff
.getItemName()
                                                + "s each weighing " + dropOff.getItemW
eight()
                                                + " from " + dropOff.getItemSender() +
" to " + dropOff.getItemReceiver());
                    }

                    if((response.toUpperCase().equals("Y")))
                    {
                        System.out.println(" Thanks for letting me keep a " + drop
Off.getItemName() + "!");
                        Package sample = new Package(dropOff.getItemName().toStrin
g(), dropOff.getItemWeight(), 1, dropOff.getItemSender(), dropOff.getItemReceiver(
));
                        samples.pickUpOrder(sample);
                    }
                    else
                    {
                        System.out.println(" Thanks anyway.");
                    }
                }

                break;
            case 3:
                System.out.println("Bag has " + bag.getPackages() + " packages and
 weights " + bag.getWeight() + " lbs.");
                break;
            case 4:
                System.out.println("Sample bag has " + samples.getPackages() + " p
ackages and weights " + samples.getWeight() + " lbs.");

                break;
            case 5:
                if(bag.getBag().isEmpty())
                {
                    System.out.println("No samples to enjoy!");
                }
                else
```

```
                {
                    Package sample = (Package) samples.getBag().pop();

                    float newWeight = samples.getWeight() - sample.getItemWeight()
;
                    int newPackages = samples.getPackages()-1;

                    samples.setWeight(newWeight);
                    samples.setPackages(newPackages);

                    System.out.println("This " + sample.getItemName() + " is amazi
ng! I love free stuff!");
                }
                break;
            case 6:
                if(samples.getBag().isEmpty())
                {
                    System.out.println("Sample bag is already empty.");
                }
                else
                {
                    samples.getBag().popAll();
                    samples.setPackages(0);
                    samples.setWeight(0);
                    System.out.println("Sample bag has been emptied.");
                }
                break;

            default:
                break;
            }
        }
    }
}
```

```
::::::::::::::
MenuHandling.java
::::::::::::::
import java.util.LinkedList;

public class MenuHandling {

    public static <T> T peek(StackInterface<T> items) {
        return items.peek();
    }

    public static <T> StackInterface<T> push(T item, StackInterface<T> items) {
        items.push(item);
        return items;
    }

    public static <T> T pop(StackInterface<T> items)
    {
        return items.pop();
    }

    public static <T> String showAll(StackInterface<T> items)
    {
        return items.toString();
    }
}
```

```
::::::::::::::
Node.java
::::::::::::::
public class Node<T> {
    private T item;
    private Node<T> next;

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

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

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

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

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

    public Node<T> getNext() {
        return next;
    } // end getNext
} // end class Node::::::::::::::
Package.java
::::::::::::::

public class Package {

    private String itemName;
    private float itemWeight;
    private int itemAmount;
    private String itemSender;
    private String itemReceiver;

    public Package(String name, float weight, int amount, String sender, String re
ceiver)
    {
        itemName = name;
        itemWeight = weight;
        itemAmount = amount;
        itemSender = sender;
        itemReceiver = receiver;
    }

    public String getItemName() {
        return itemName;
    }

    public void setItemName(String itemName) {
        this.itemName = itemName;
    }
```

```
    public float getItemWeight() {
        return itemWeight;
    }

    public void setItemWeight(float itemWeight) {
        this.itemWeight = itemWeight;
    }

    public int getItemAmount() {
        return itemAmount;
    }

    public void setItemAmount(int itemAmount) {
        this.itemAmount = itemAmount;
    }

    public String getItemSender() {
        return itemSender;
    }

    public void setItemSender(String itemSender) {
        this.itemSender = itemSender;
    }

    public String getItemReceiver() {
        return itemReceiver;
    }

    public void setItemReceiver(String itemReceiver) {
        this.itemReceiver = itemReceiver;
    }


}
::::::::::::::
Sample.java
::::::::::::::

public class Sample extends Package {

    private String itemName;
    private float itemWeight;
    private int itemAmount;
    private String itemSender;
    private String itemReceiver;

    public Sample(String name, float weight, int amount, String sender, String rec
eiver)
    {
        super(receiver, weight, amount, receiver, receiver);
    }

    public String getItemName() {
        return itemName;
    }

    public void setItemName(String itemName) {
        this.itemName = itemName;
    }

    public float getItemWeight() {
        return itemWeight;
    }

    public void setItemWeight(float itemWeight) {
        this.itemWeight = itemWeight;
    }

    public int getItemAmount() {
        return itemAmount;
    }

    public void setItemAmount(int itemAmount) {
        this.itemAmount = itemAmount;
    }

    public String getItemSender() {
        return itemSender;
    }

    public void setItemSender(String itemSender) {
        this.itemSender = itemSender;
    }

    public String getItemReceiver() {
        return itemReceiver;
    }

    public void setItemReceiver(String itemReceiver) {
        this.itemReceiver = itemReceiver;
    }
}
::::::::::::::
StackException.java
::::::::::::::
public class StackException
    extends java.lang.RuntimeException {
    public StackException(String s) {
        super(s);
    }  // end constructor
}  // end StackException::::::::::::::
StackInterface.java
::::::::::::::
public interface StackInterface<T> {
    public boolean isEmpty();
    // Determines whether the stack is empty.
    // Precondition: None.
    // Postcondition: Returns true if the stack is empty;
    // otherwise returns false.

    public void popAll();
    // Removes all the items from the stack.
    // Precondition: None.
    // PostCondition: Stack is empty.

    public void push(T newItem) throws StackException;
    // Adds an item to the top of a stack.
    // Precondition: newItem is the item to be added.
    // Postcondition: If insertion is successful, newItem
    // is on the top of the stack.
    // Exception: Some implementations may throw
    // StackException when newItem cannot be placed on
    // the stack.
```

```
    public T pop() throws StackException;
    // Removes the top of a stack.
    // Precondition: None.
    // Postcondition: If the stack is not empty, the item
    // that was added most recently is removed from the
    // stack.
    // Exception: Throws StackException if the stack is
    // empty.

    public T peek() throws StackException;
    // Retrieves the top of a stack.
    // Precondition: None.
    // Postcondition: If the stack is not empty, the item
    // that was added most recently is returned. The
    // stack is unchanged.
    // Exception: Throws StackException if the stack is
    // empty.
    public String toString();
}  // end StackInterface::::::::::::::
StackRAB.java
::::::::::::::

public class StackRAB<T> implements StackInterface {

    private int top;
    private T[] items;

    public StackRAB()
    {
        items = (T[]) new Object[3];
        top = -1;
    }

    @Override
    public boolean isEmpty() {
        if(top == -1)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    @Override
    public void popAll() {
        items = (T[]) new Object[3];
        top = -1;
    }

    @Override
    public void push(Object newItem) throws StackException {
        if(top == items.length-1)
        {
            resize();
        }
        items[++top] = (T) newItem;
    }

    private void resize() {
        T[] newItems = (T[]) new Object[(items.length * (3/2)) + 1];
        for(int i = 0; i < items.length; i++)
        {
            newItems[i] = items[i];
        }

        items = newItems;
    }


    @Override
    public Object pop() throws StackException {
        T result = null;
        if(top != -1)
        {
            result = items[top];
            System.out.print("Item to be popped: " + result);
            items[top--] = null;
        }
        else
        {
            System.out.println("Stack is empty!");
        }
        return result;
    }

    @Override
    public T peek() throws StackException {

        T result = null;
        if(top != -1)
        {
            result = items[top];
        }
        else
        {
            System.out.println("List empty!");
        }
        return result;
    }

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

        for(int i = 0; i <= top; i++)
        {
            String name = items[i].toString() + " ";
            builder.append(name);
        }

        return toReturn = builder.toString();

    }


}
::::::::::::::
StackSLS.java
::::::::::::::
```

```
public class StackSLS<T> implements StackInterface {

    private Node top;

    public <T> StackSLS()
    {
        top = null;
    }

    @Override
    public boolean isEmpty() {
        if(top == null)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    @Override
    public void popAll() {
        top = null;
    }

    @Override
    public void push(Object newItem) throws StackException {
        top = new Node(newItem, top);
    }

    @Override
    public T pop() throws StackException {
        T result = null;
        if(top != null)
        {
            result = (T) top.getItem();
            top = top.getNext();
        }
        return result;
    }

    @Override
    public T peek() throws StackException {
        T result = null;
        if(top != null)
        {
            result = (T) top.getItem();
        }
        return result;
    }

    public String toString()
    {
        Node<T> next = top;
        StringBuilder builder = new StringBuilder();
        String toReturn = "";

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

        return toReturn;

    }
}
```

```
::::::::::::::
output.txt
::::::::::::::

Select from the following menu:
        0. Exit.
        1. Pick up an order.
        2. Drop off an order.
        3. Display number of packages and weight of bag.
        4. Display number of items and weight of the bag of samples.
        5. Enjoy an item from the bag of samples.
        6. Enjoy all the samples in the bag of samples.


Make your selection now: 3
Bag has 0 packages and weights 0.0 lbs.

Make your selection now: 4
Sample bag has 0 packages and weights 0.0 lbs.

Make your selection now: 5
No samples to enjoy!

Make your selection now: 6
Sample bag is already empty.

Make your selection now: 2
No deliveries to process!

Make your selection now: 1
Please specify info:
Item name: apple

Item weight: 0.6

# of items: 10

Sender: Pickachu

Recipient: Mew
A package of apple each weighing 0.6 lbs are now in the bag.

Make your selection now: 3
Bag has 1 packages and weights 6.0 lbs.

Make your selection now: 4
Sample bag has 0 packages and weights 0.0 lbs.

Make your selection now: 1
Please specify info:
Item name: orange

Item weight: 0.85
```

```
# of items: 14

Sender: Bulbasaur

Recipient: Abra
A package of orange each weighing 0.85 lbs are now in the bag.

Make your selection now: 1
Please specify info:
Item name: pear

Item weight: 0.9

# of items: 7

Sender: Abra

Recipient: Kadabra
A package of pear each weighing 0.9 lbs are now in the bag.

Make your selection now: 3
Bag has 3 packages and weights 24.2 lbs.

Make your selection now: 4
Sample bag has 0 packages and weights 0.0 lbs.

Make your selection now: 2
Here is your package Kadabra. May I keep a sample (Y/N)? Y
Your package contains:
7 pears each weighing 0.9 from Abra to Kadabra
 Thanks for letting me keep a pear!

Make your selection now: 4
Sample bag has 1 packages and weights 0.9 lbs.

Make your selection now: 3
Bag has 2 packages and weights 17.900002 lbs.

Make your selection now: 1
Please specify info:
Item name: cookie

Item weight: 0.1

# of items: 50

Sender: Charizard

Recipient: Squirtle
A package of cookie each weighing 0.1 lbs are now in the bag.

Make your selection now: 1
Please specify info:
Item name: banana

Item weight: 0.5

# of items: 22

Sender: Clefairy

Recipient: Vulpix
A package of banana each weighing 0.5 lbs are now in the bag.

Make your selection now: 4
Sample bag has 1 packages and weights 0.9 lbs.

Make your selection now: 3
Bag has 4 packages and weights 33.9 lbs.

Make your selection now: 2
Here is your package Vulpix. May I keep a sample (Y/N)? N
Your package contains:
22 bananas each weighing 0.5 from Clefairy to Vulpix
 Thanks anyway.

Make your selection now: 4
Sample bag has 1 packages and weights 0.9 lbs.

Make your selection now: 3
Bag has 3 packages and weights 22.900002 lbs.

Make your selection now: 2
Here is your package Squirtle. May I keep a sample (Y/N)? Y
Your package contains:
50 cookies each weighing 0.1 from Charizard to Squirtle
 Thanks for letting me keep a cookie!

Make your selection now: 4
Sample bag has 2 packages and weights 1.0 lbs.

Make your selection now: 3
Bag has 2 packages and weights 17.900002 lbs.

Make your selection now: 5
This cookie is amazing! I love free stuff!

Make your selection now: 3
Bag has 2 packages and weights 17.900002 lbs.

Make your selection now: 4
Sample bag has 1 packages and weights 0.9 lbs.

Make your selection now: 6
Sample bag has been emptied.

Make your selection now: 1
Please specify info:
Item name: granola

Item weight: 0.5

# of items: 25

Sender: Jigglypuff

Recipient: Meowth
A package of granola each weighing 0.5 lbs are now in the bag.

Make your selection now: 1
Please specify info:
Item name: watermelon

Item weight: 3.7
```

```
# of items: 3

Sender: Slowpoke

Recipient: Slowbro
A package of watermelon each weighing 3.7 lbs are now in the bag.

Make your selection now: 2
Here is your package Slowbro. May I keep a sample (Y/N)? Y
Your package contains:
3 watermelons each weighing 3.7 from Slowpoke to Slowbro
 Thanks for letting me keep a watermelon!

Make your selection now: 3
Bag has 3 packages and weights 30.4 lbs.

Make your selection now: 4
Sample bag has 1 packages and weights 3.7 lbs.

Make your selection now: 5
This watermelon is amazing! I love free stuff!

Make your selection now: 0
Exiting program...Good Bye
::::::::::::::
Lab3Generic.txt
::::::::::::::
LAB 3 CONVERTED TO A GENERIC COLLECTION


/*

 * 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(Sy
stem.in));

        public static <T> void main(String[] args) throws NumberFormatException, I
OException {

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

                System.out.println("\nSelect from the following menu:"
                                + "\n\t1. Insert item to list\n\t2. Remove item fr
om list"
                                + "\n\t3. Get item from list"
                                + "\n\t4. Clear list\n\t5. Print size and content
of list"
                                + "\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 i
s out of range!");
                                }
                                else
                                {
                                items.add(index, (T) item);
                                System.out.println("\nItem " + item + " inserted a
t position " + index + " in the list.");
```

```
                        }
                        break;
                case 2:
                        System.out.print("\tEnter position to remove item
from: ");
                        int toRemove = Integer.parseInt(stdin.readLine().t
rim());
                        System.out.println(toRemove);

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

                        if((toRetrieve >= items.size()) || (toRetrieve < 0
))
                        {
                                System.out.println("\nPosition specified i
s out of range!");
                        }
                        else
                        {
                                System.out.println("\nItem " + items.get(t
oRetrieve) + " retrieved 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 following 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(compa
re) > 0)
                                        {
                                        compare = items.get(i).toString();
                                        removeIndex = i;
                                        }
                        }
                        System.out.println("Largest item " + items.get(rem
oveIndex).toString() + " deleted.");
                        items.remove(removeIndex);
                        }
                        break;

                case 7:

                        if(items.size() == 0)
                        {
                                System.out.println("List is empty... nothi
ng to reverse!");
                        }
                        else
                        {

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

                                        items.add(i, (T) toFront);
                                        items.remove(i+1);

                                        items.add(k, (T) 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;
                }
```

```
                }

        }
}


/*

 * 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

 */

public class ListIndexOutOfBoundsException
    extends IndexOutOfBoundsException
{
    public ListIndexOutOfBoundsException(String s)
    {
        super(s);
    }  // end constructor
}  // end ListIndexOutOfBoundsException



/*

 * 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

 */

// ****************************************************
// 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



/*

 * 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



/*

 * 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 different from the textbook code, because the data
 is encapsulated!

public class Node<T>
{
  private T item;
  private Node next;

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

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

  public void setItem(T 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
::::::::::::::
Lab5Conclusions.txt
::::::::::::::
Generics are *funky.*

They are incredibly powerful, but *incredibly* funky at the same time; there was
a bit of a learning curve to converting from a straight Object type to the
generic T type, but once I got that, it was smooth sailing.

Really, the thing that I got the most of out of this lab (outside of the use of
Queues and Generics)?

REMEMBER TO UPDATE YOUR VARIABLES!

I spent *so long* trying to figure out what was causing the math to bug out,
only to realize after a good night's sleep, *I* was the cause of the bad math!

Such a novice error to make and somehow, I did it.
```