```
::::::::::::::
PreLab_allfiles
::::::::::::::
::::::::::::::
Driver.java
::::::::::::::

/*

 * Purpose: Data Structure and Algorithms Lab 6 PreLab

 * Status: Complete and thoroughly tested

 * Last update: 03/03/2020

 * Submitted:  03/03/2020

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.03/03

 */


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

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

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

                QueueRAB<T> queue = new QueueRAB<T>();

                System.out.println("Select from the following: "
                                + "\n\t1. Insert item at back of queue"
                                + "\n\t2. Remove item from front of queue"
                                + "\n\t3. Display front item of queue"
                                + "\n\t4. Clear queue"
                                + "\n\t5. Display content of queue"
                                + "\n\t6. Exit");

                boolean switchOn = true;
                T result;

                while(switchOn == true)
                                {
                        System.out.print("\n\nMake your selection now: ");
                        int selection = Integer.parseInt(stdin.readLine());
                        System.out.println(selection);
                        switch(selection)
                        {
                        case 1:
                                System.out.print("\nItem to be added: ");
                                Object newItem = stdin.readLine().trim();
                                System.out.println(newItem);
                                queue.enqueue(newItem);
                                System.out.println(newItem + " has been added to b
ack of queue.");
                                break;

                        case 2:
                                result = queue.dequeue();
                                if(result == null)
                                {
                                        System.out.println("Queue empty!");
                                }
                                else
                                {
                                        System.out.println("\n" + result +
                                                        " has been removed from th
e front of the queue.");
                                }
                                break;

                        case 3:
                                result = queue.peek();
                                if(result == null)
                                {
                                        System.out.println("Queue empty!");
                                }
                                else
                                {
                                        System.out.println("\nFront item of queue:
 " + result);
                                }
                                break;

                        case 4:
                                queue.dequeueAll();
                                System.out.println("\nQueue cleared.");
                                break;

                        case 5:
                                System.out.println("\nContents of queue: " + queue
.toString());
                                break;

                        case 0:
                                switchOn = false;
                                System.out.println("\nExiting program... Goodbye!"
);
                        }

                                }


        }

}
::::::::::::::
QueueException.java
::::::::::::::
public class QueueException extends Throwable {

    public QueueException(String s) {
        super(s);
    }  // end constructor
}  // end QueueException::::::::::::::
```

QueueInterface.java
::::::::::::::

```
public interface QueueInterface<T> {

    public boolean isEmpty();
    // Determines whether a queue is empty.
    // Precondition: None.
    // Postcondition: Returns true if the queue is empty;
    // otherwise returns false.

    public void enqueue(T newItem) throws QueueException;
    // Adds an item at the back of a queue.
    // Precondition: newItem is the item to be inserted.
    // Postcondition: If the operation was successful, newItem
    // is at the back of the queue. Some implementations
    // may throw QueueException if newItem cannot be added
    // to the queue.

    public T dequeue() throws QueueException;
    // Retrieves and removes the front of a queue.
    // Precondition: None.
    // Postcondition: If the queue is not empty, the item that
    // was added to the queue earliest is removed. If the queue is
    // empty, the operation is impossible and QueueException is thrown.

    public void dequeueAll();
    // Removes all items of a queue.
    // Precondition: None.
    // Postcondition: The queue is empty.

    public T peek() throws QueueException;
    // Retrieves the item at the front of a queue.
    // Precondition: None.
    // Postcondition: If the queue is not empty, the item
    // that was added to the queue earliest is returned.
    // If the queue is empty, the operation is impossible
    // and QueueException is thrown.

    public String toString();
}  // end QueueInterface
```

::::::::::::::
QueueRAB.java
::::::::::::::

```
public class QueueRAB<T> implements QueueInterface<T>  {

        protected int numItems;
        protected int front;
        protected int back;
        protected T[] items;

        public QueueRAB()
        {
                numItems = 0;
                back = 0;
                front = 0;
                items = (T[]) new Object[3];
        }


        @Override
        public boolean isEmpty() {
                return (numItems == 0);
        }

        @Override
        public void enqueue(Object newItem) throws QueueException {
                if(numItems == items.length)
                {
                        resize();
                }
                items[back] = (T) newItem;
                back = (back+1)%items.length;
                numItems++;

        }

        @Override
        public T dequeue() throws QueueException {
                T result = null;
                if(numItems > 0)
                {
                        result = items[front];
                        items[front] = null;
                        front = (front + 1)%items.length;
                        numItems--;
                }
                return result;
        }

        @Override
        public void dequeueAll() {
                numItems = 0;
                back = 0;
                front = 0;
                items = (T[]) new Object[3];
        }

        @Override
        public T peek() throws QueueException {

                T result = null;
                if(numItems > 0)
                {
                        result = items[front];
                }
                else
                {
                        System.out.println("Queue empty!");
                }
                return result;

        }

        public String toString()
        {
                   StringBuilder builder = new StringBuilder();
                   String toReturn = "";
                int counter = 0;
                for(int i = front; counter < numItems; counter++)
                {

                        String name = items[(i)].toString() + " ";
```

```
                        builder.append(name);
                        i = (i+1) % items.length;
                }

                return toReturn = builder.toString();

        }

        private void resize() {
                T[] temp = (T[]) new Object[items.length + ((items.length/2) + 1)]
;
                System.out.println();
                int conuter = 0;
                for(int i = front; conuter < numItems; conuter++)
                {
                        temp[conuter] = items[i];
                        i = (i+1) % items.length;
                }

                items = temp;
                front = 0;
                back = numItems;

        }

}
::::::::::::::
output.txt
::::::::::::::
Select from the following:
        1. Insert item at back of queue
        2. Remove item from front of queue
        3. Display front item of queue
        4. Clear queue
        5. Display content of queue
        6. Exit


Make your selection now: 1

Item to be added: A-Assault Core
A-Assault Core has been added to back of queue.


Make your selection now: 1

Item to be added: B-Buster Drake
B-Buster Drake has been added to back of queue.


Make your selection now: 1

Item to be added: C-Crush Wyvern
C-Crush Wyvern has been added to back of queue.


Make your selection now: 3

Front item of queue: A-Assault Core


Make your selection now: 2

A-Assault Core has been removed from the front of the queue.


Make your selection now: 3

Front item of queue: B-Buster Drake


Make your selection now: 2

B-Buster Drake has been removed from the front of the queue.


Make your selection now: 3

Front item of queue: C-Crush Wyvern


Make your selection now: 2

C-Crush Wyvern has been removed from the front of the queue.


Make your selection now: 3
Queue empty!
Queue empty!


Make your selection now: 1

Item to be added: ABC-Dragon Buster
ABC-Dragon Buster has been added to back of queue.


Make your selection now: 1

Item to be added: VYXYZ-Dragon Catapult Cannon
VYXYZ-Dragon Catapult Cannon has been added to back of queue.


Make your selection now: 4

Queue cleared.


Make your selection now: 1

Item to be added: A-to-Z Dragon Cannon Buster
A-to-Z Dragon Cannon Buster has been added to back of queue.


Make your selection now: 1

Item to be added: Celtic Guardian
Celtic Guardian has been added to back of queue.


Make your selection now: 1

Item to be added: Winged Dragon of Ra
Winged Dragon of Ra has been added to back of queue.
```

```
Make your selection now: 1

Item to be added: Slifer, the Sky Dragon

Slifer, the Sky Dragon has been added to back of queue.


Make your selection now: 1

Item to be added: Obelisk the Tormentor
Obelisk the Tormentor has been added to back of queue.


Make your selection now: 3

Front item of queue: A-to-Z Dragon Cannon Buster


Make your selection now: 5

Contents of queue: A-to-Z Dragon Cannon Buster Celtic Guardian Winged Dragon of Ra
 Slifer, the Sky Dragon Obelisk the Tormentor


Make your selection now: 4

Queue cleared.


Make your selection now: 0

Exiting program... Goodbye!
::::::::::::::
Problem2_allfiles
::::::::::::::
::::::::::::::
DEQ.java
::::::::::::::
```

```
public class DEQ<T> extends Queue<T> implements ExtendedQueueInterface<T> {


    public DEQ()
    {
        super();
    }

    @Override
    public void enqueueFirst(Object newItem) throws ExtendedQueueException {
        if(numItems == items.length)
        {
            super.resize();
        }

        front = (front + items.length-1)%items.length;

        items[front] = (T) newItem;
        numItems++;
    }

    @Override
    public T dequeueLast() throws ExtendedQueueException {
        T result = null;
        if(numItems > 0)
        {
            back = (back + items.length-1)%items.length;
            result = items[back];
            items[back] = null;
            numItems--;
        }
        return result;
    }

    @Override
    public T peekLast() throws ExtendedQueueException {
        T result = null;
        if(numItems > 0)
        {
            result = items[(back + items.length-1)%items.length];
        }
        else
        {
            System.out.println("Queue empty!");
        }
        return result;
    }

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

/*

 * Purpose: Data Structure and Algorithms Lab 6 Problem 2

 * Status: Complete and thoroughly tested

 * Last update: 03/03/2020

 * Submitted:  03/03/2020

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.03/03

 */

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, QueueException {

        DEQ<T> queue = new DEQ<T>();
```

```
        System.out.println("Select from the following: "
                           + "\n\t0. Exit"
                           + "\n\t1. Insert item at back"
                           + "\n\t2. Insert item at front"
                           + "\n\t3. Remove item from front"
                           + "\n\t4. Remove item from back"
                           + "\n\t5. Display front item"
                           + "\n\t6. Display last item"
                           + "\n\t7. Clear collection"
                           + "\n\t8. Display content of collection");

        boolean switchOn = true;
        T result;

        while(switchOn == true)
        {
            System.out.print("\n\nMake your selection now: ");
            int selection = Integer.parseInt(stdin.readLine());
            System.out.println(selection);
            switch(selection)
            {
            case 0:
                switchOn = false;
                System.out.println("\nExiting program... Goodbye!");
                break;
            case 1:
                System.out.print("\nItem to be added at back: ");
                Object newItem = stdin.readLine();
                System.out.println(newItem);
                queue.enqueue(newItem);
                System.out.println(newItem + " has been added to back of queue.");
                break;

            case 2:
                System.out.print("\nItem to be added at front: ");
                Object frontItem = stdin.readLine();
                System.out.println(frontItem);
                queue.enqueueFirst(frontItem);
                System.out.println(frontItem + " has been added to front of queue.
");
                break;

            case 3:
                result = queue.dequeue();
                if(result == null)
                {
                    System.out.println("Queue empty!");
                }
                else
                {
                    System.out.println("\n" + result +
                                       " has been removed from the front of the qu
eue.");
                }
                break;

            case 4:

                result = queue.dequeueLast();
                if(result == null)
                {
                    System.out.println("Queue empty!");
                }
                else
                {
                    System.out.println("\n" + result +
                                       " has been removed from the back of the que
ue.");
                }

                break;

            case 5:
                result = queue.peek();
                if(result == null)
                {
                    System.out.println("Queue empty!");
                }
                else
                {
                    System.out.println("\nFront item of queue: " + result);
                }
                break;

            case 6:
                result = queue.peekLast();
                if(result == null)
                {
                    System.out.println("Queue empty!");
                }
                else
                {
                    System.out.println("\nBack item of queue: " + result);
                }
                break;
            case 7:
                queue.dequeueAll();
                System.out.println("\nQueue cleared.");
                break;

            case 8:
                if(queue.numItems == 0)
                {
                    System.out.println("Queue empty!");
                }
                else
                {
                    System.out.println("\nContents of queue: " + queue.toString())
;
                }
                break;

            default:
                break;
            }

        }


    }
}
::::::::::::::
```

ExtendedQueueException.java
::::::::::::::

```
public class ExtendedQueueException extends RuntimeException {

    public ExtendedQueueException(String s) {
        super(s);
    }  // end constructor
}  // end ExtendedQueueException::::::::::::::
```

ExtendedQueueInterface.java
::::::::::::::

```
public interface ExtendedQueueInterface<T> extends QueueInterface<T> {
    public void enqueueFirst(T newItem) throws ExtendedQueueException;
    public T dequeueLast() throws ExtendedQueueException;
    public T peekLast() throws ExtendedQueueException;
}  // end ExtendedQueueInterface
```

::::::::::::::
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::::::::::::::
```

QueueException.java
::::::::::::::

```
public class QueueException extends Throwable {

    public QueueException(String s) {
        super(s);
    }  // end constructor
}  // end QueueException::::::::::::::
```

QueueInterface.java
::::::::::::::

```
public interface QueueInterface<T> {

    public boolean isEmpty();
    // Determines whether a queue is empty.
    // Precondition: None.
    // Postcondition: Returns true if the queue is empty;
    // otherwise returns false.

    public void enqueue(T newItem) throws QueueException;
    // Adds an item at the back of a queue.
    // Precondition: newItem is the item to be inserted.
    // Postcondition: If the operation was successful, newItem
    // is at the back of the queue. Some implementations
    // may throw QueueException if newItem cannot be added
    // to the queue.

    public T dequeue() throws QueueException;
    // Retrieves and removes the front of a queue.
    // Precondition: None.
    // Postcondition: If the queue is not empty, the item that
    // was added to the queue earliest is removed. If the queue is
    // empty, the operation is impossible and QueueException is thrown.

    public void dequeueAll();
    // Removes all items of a queue.
    // Precondition: None.
    // Postcondition: The queue is empty.

    public T peek() throws QueueException;
    // Retrieves the item at the front of a queue.
    // Precondition: None.
    // Postcondition: If the queue is not empty, the item
    // that was added to the queue earliest is returned.
    // If the queue is empty, the operation is impossible
    // and QueueException is thrown.

    public String toString();
}  // end QueueInterface
```

::::::::::::::
Queue.java
::::::::::::::

```
public class Queue<T> implements QueueInterface<T>  {

    protected int numItems;
    protected int front;
    protected int back;
    protected T[] items;

    public Queue()
    {
        numItems = 0;
        back = 0;
        front = 0;
        items = (T[]) new Object[3];
    }


    @Override
    public boolean isEmpty() {
        return (numItems == 0);
    }
```

```
@Override
public void enqueue(Object newItem) throws QueueException {
    if(numItems == items.length)
    {
        resize();
    }
    items[back] = (T) newItem;
    back = (back+1)%items.length;
    numItems++;
}

@Override
public T dequeue() throws QueueException {
    T result = null;
    if(numItems > 0)
    {
        result = items[front];
        items[front] = null;
        front = (front + 1)%items.length;
        numItems--;
    }
    return result;
}

@Override
public void dequeueAll() {
    numItems = 0;
    back = 0;
    front = 0;
    items = (T[]) new Object[3];
}

@Override
public T peek() throws QueueException {

    T result = null;
    if(numItems > 0)
    {
        result = items[front];
    }
    else
    {
        System.out.println("Queue empty!");
    }
    return result;

}

public String toString()
{
    StringBuilder builder = new StringBuilder();
    String toReturn = "";
    int counter = 0;

    for(int i = front; counter < numItems; counter++)
    {
        String build = items[i].toString() + " ";
        builder.append(build);
        i = ((i+1)%items.length);
    }
        return toReturn = builder.toString();

    }

    protected void resize() {
        T[] temp = (T[]) new Object[items.length+1];
        System.out.println();
        int counter = 0;
        for(int i = front; counter < numItems; counter++)
        {
            temp[counter] = items[i];
            i = (i+1) % items.length;
        }

        items = temp;
        front = 0;
        back = numItems;

    }

}
::::::::::::::
QueueSLS.java
::::::::::::::

public class QueueSLS<T> implements QueueInterface {

    Node<T> front;
    Node<T> back;

    public QueueSLS()
    {
        front = null;
        back = null;
    }

    @Override
    public boolean isEmpty() {

        if(front == null)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    @Override
    public void enqueue(Object newItem) throws QueueException {
        if(back == null)
        {
            front = back = new Node(newItem);
        }
        else
        {
            Node temp = new Node(newItem);
            back.setNext(temp);
            back = temp;
        }
    }
```

```
    @Override
    public Object dequeue() throws QueueException {

        Object result = null;

        if(front.getNext() != null)
        {
            result = front.getItem();
            front = front.getNext();
            if(front == null)
            {
                back = null;
            }
        }

        return result;
    }

    @Override
    public void dequeueAll() {
        front = null;
        back = null;
    }

    @Override
    public Object peek() throws QueueException {
        return front.getItem();
    }

    public String toString()
    {
        Node<T> next = front;
        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:
        0. Exit
        1. Insert item at back
        2. Insert item at front
        3. Remove item from front
        4. Remove item from back
        5. Display front item
        6. Display last item
        7. Clear collection
        8. Display content of collection

Make your selection now: 3
Queue empty!


Make your selection now: 4
Queue empty!


Make your selection now: 7

Queue cleared.


Make your selection now: 1

Item to be added at back: Agumon
Agumon has been added to back of queue.


Make your selection now: 1

Item to be added at back: Gabumon
Gabumon has been added to back of queue.


Make your selection now: 1

Item to be added at back: Tentomon
Tentomon has been added to back of queue.


Make your selection now: 2

Item to be added at front: Veemon

Veemon has been added to front of queue.


Make your selection now: 2

Item to be added at front: Garurumon

Garurumon has been added to front of queue.


Make your selection now: 5

Front item of queue: Garurumon


Make your selection now: 6

Back item of queue: Tentomon


Make your selection now: 8

Contents of queue: Garurumon Veemon Agumon Gabumon Tentomon
```

```
Make your selection now: 3

Garurumon has been removed from the front of the queue.


Make your selection now: 4

Tentomon has been removed from the back of the queue.


Make your selection now: 7

Queue cleared.


Make your selection now: 8
Queue empty!


Make your selection now: 1

Item to be added at back: Omnimon
Omnimon has been added to back of queue.


Make your selection now: 2

Item to be added at front: WarGreymon
WarGreymon has been added to front of queue.


Make your selection now: 2

Item to be added at front: MetalGarurumon
MetalGarurumon has been added to front of queue.


Make your selection now: 2

Item to be added at front: Rosemon

Rosemon has been added to front of queue.


Make your selection now: 5

Front item of queue: Rosemon


Make your selection now: 3

Rosemon has been removed from the front of the queue.


Make your selection now: 5

Front item of queue: MetalGarurumon


Make your selection now: 3

MetalGarurumon has been removed from the front of the queue.


Make your selection now: 5

Front item of queue: WarGreymon


Make your selection now: 3

WarGreymon has been removed from the front of the queue.


Make your selection now: 8

Contents of queue: Omnimon


Make your selection now: 7

Queue cleared.


Make your selection now: 0

Exiting program... Goodbye!
::::::::::::::
Problem3_allfiles
::::::::::::::
::::::::::::::
Bag.java
::::::::::::::
public class Bag <T> {
    protected int packages;
    protected float weight;
    protected Queue<T> bag;

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

    public void pickUpOrder(Sample pack) throws QueueException
    {
        bag.enqueue(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 Queue<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;
    }

}
::::::::::::::
DEQ.java
::::::::::::::

public class DEQ<T> extends Queue<T> implements ExtendedQueueInterface<T> {


    public DEQ()
    {
        super();
    }

    @Override
    public void enqueueFirst(Object newItem) throws ExtendedQueueException {
        if(numItems == items.length)
        {
            super.resize();
        }

        front = (front + items.length-1)%items.length;

        items[front] = (T) newItem;
        numItems++;
    }

    @Override
    public T dequeueLast() throws ExtendedQueueException {
        T result = null;
        if(numItems > 0)
        {
            back = (back + items.length-1)%items.length;
            result = items[back];
            items[back] = null;
            numItems--;
        }
        return result;
    }

    @Override
    public T peekLast() throws ExtendedQueueException {
        T result = null;
        if(numItems > 0)
        {
            result = items[(back + items.length-1)%items.length];
        }
        else
        {
            System.out.println("Queue empty!");
        }
        return result;
    }

}
::::::::::::::
Driver.java
::::::::::::::
/*

 * Purpose: Data Structure and Algorithms Lab 6 Problem 3

 * Status: Complete and thoroughly tested

 * Last update: 03/03/2020

 * Submitted:  03/03/2020

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.03.03

 */

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, QueueException {


        Bag<T> bag = new Bag<T>();
        SampleBag<T> samples = new SampleBag<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().dequeue();


                    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;
            }
        }
    }
}


::::::::::::::
ExtendedQueueException.java
::::::::::::::

public class ExtendedQueueException extends RuntimeException {

    public ExtendedQueueException(String s) {
        super(s);
    }  // end constructor
}  // end ExtendedQueueException::::::::::::::
ExtendedQueueInterface.java
::::::::::::::

public interface ExtendedQueueInterface<T> extends QueueInterface<T> {
    public void enqueueFirst(T newItem) throws ExtendedQueueException;
    public T dequeueLast() throws ExtendedQueueException;
    public T peekLast() throws ExtendedQueueException;
}  // end ExtendedQueueInterface
::::::::::::::
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 extends Sample {

    public Package(String name, float weight, int amount, String sender, String re
ceiver)
    {
        super(receiver, weight, amount, receiver, receiver);
    }
}
::::::::::::::
QueueException.java
::::::::::::::
public class QueueException extends Throwable {

    public QueueException(String s) {
        super(s);
    }  // end constructor
```

```
}  // end QueueException::::::::::::::
QueueInterface.java
::::::::::::::
public interface QueueInterface<T> {

    public boolean isEmpty();
    // Determines whether a queue is empty.
    // Precondition: None.
    // Postcondition: Returns true if the queue is empty;
    // otherwise returns false.

    public void enqueue(T newItem) throws QueueException;
    // Adds an item at the back of a queue.
    // Precondition: newItem is the item to be inserted.
    // Postcondition: If the operation was successful, newItem
    // is at the back of the queue. Some implementations
    // may throw QueueException if newItem cannot be added
    // to the queue.

    public T dequeue() throws QueueException;
    // Retrieves and removes the front of a queue.
    // Precondition: None.
    // Postcondition: If the queue is not empty, the item that
    // was added to the queue earliest is removed. If the queue is
    // empty, the operation is impossible and QueueException is thrown.

    public void dequeueAll();
    // Removes all items of a queue.
    // Precondition: None.
    // Postcondition: The queue is empty.

    public T peek() throws QueueException;
    // Retrieves the item at the front of a queue.
    // Precondition: None.
    // Postcondition: If the queue is not empty, the item
    // that was added to the queue earliest is returned.
    // If the queue is empty, the operation is impossible
    // and QueueException is thrown.

    public String toString();
}  // end QueueInterface

::::::::::::::
Queue.java
::::::::::::::

public class Queue<T> implements QueueInterface<T>  {

    protected int numItems;
    protected int front;
    protected int back;
    protected T[] items;

    public Queue()
    {
        numItems = 0;
        back = 0;
        front = 0;
        items = (T[]) new Object[3];
    }

    @Override
    public boolean isEmpty() {
        return (numItems == 0);
    }

    @Override
    public void enqueue(Object newItem) throws QueueException {
        if(numItems == items.length)
        {
            resize();
        }
        items[back] = (T) newItem;
        back = (back+1)%items.length;
        numItems++;
    }

    @Override
    public T dequeue() throws QueueException {
        T result = null;
        if(numItems > 0)
        {
            result = items[front];
            items[front] = null;
            front = (front + 1)%items.length;
            numItems--;
        }
        return result;
    }

    @Override
    public void dequeueAll() {
        numItems = 0;
        back = 0;
        front = 0;
        items = (T[]) new Object[3];
    }

    @Override
    public T peek() throws QueueException {

        T result = null;
        if(numItems > 0)
        {
            result = items[front];
        }
        else
        {
            System.out.println("Queue empty!");
        }
        return result;

    }

    public String toString()
    {
        StringBuilder builder = new StringBuilder();
        String toReturn = "";
        int counter = 0;

        for(int i = front; counter < numItems; counter++)
        {
            String build = items[i].toString() + " ";
```

```
            builder.append(build);
            i = ((i+1)%items.length);
        }


        return toReturn = builder.toString();

    }

    protected void resize() {
        T[] temp = (T[]) new Object[items.length+1];
        System.out.println();
        int counter = 0;
        for(int i = front; counter < numItems; counter++)
        {
            temp[counter] = items[i];
            i = (i+1) % items.length;
        }

        items = temp;
        front = 0;
        back = numItems;

    }



}
::::::::::::::
QueueSLS.java
::::::::::::::

public class QueueSLS<T> implements QueueInterface {

    Node<T> front;
    Node<T> back;

    public QueueSLS()
    {
        front = null;
        back = null;
    }

    @Override
    public boolean isEmpty() {

        if(front == null)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    @Override
    public void enqueue(Object newItem) throws QueueException {
        if(back == null)
        {
            front = back = new Node(newItem);
        }
        else
        {
            Node temp = new Node(newItem);
            back.setNext(temp);
            back = temp;
        }
    }

    @Override
    public Object dequeue() throws QueueException {

        Object result = null;

        if(front.getNext() != null)
        {
            result = front.getItem();
            front = front.getNext();
            if(front == null)
            {
                back = null;
            }
        }

        return result;
    }

    @Override
    public void dequeueAll() {
        front = null;
        back = null;
    }

    @Override
    public Object peek() throws QueueException {
        return front.getItem();
    }

    public String toString()
    {
        Node<T> next = front;
        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;

    }

}
::::::::::::::
SampleBag.java
::::::::::::::

public class SampleBag<T> {
```

```
    private int packages;
    private float weight;
    private StackSLS<T> bag;

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

    public void pickUpOrder(Sample pack) throws QueueException
    {
        bag.push(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 StackSLS<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;
    }


}
```

```
::::::::::::::
Sample.java
::::::::::::::
public class Sample {

    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)
    {
        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;
    }
}
::::::::::::::
```

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::::::::::::::
```

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 Mew. May I keep a sample (Y/N)? Y
Your package contains:
10 Mews each weighing 0.6 from Mew to Mew
 Thanks for letting me keep a Mew!

Make your selection now: 4
Sample bag has 1 packages and weights 0.6 lbs.

Make your selection now: 3
Bag has 2 packages and weights 18.2 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 0.6 lbs.

Make your selection now: 3
Bag has 4 packages and weights 34.2 lbs.
```

```
Make your selection now: 2
Here is your package Abra. May I keep a sample (Y/N)? N
Your package contains:
14 Abras each weighing 0.85 from Abra to Abra
 Thanks anyway.

Make your selection now: 4
Sample bag has 1 packages and weights 0.6 lbs.

Make your selection now: 3
Bag has 3 packages and weights 22.3 lbs.

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

Make your selection now: 4
Sample bag has 2 packages and weights 1.5 lbs.

Make your selection now: 3
Bag has 2 packages and weights 16.0 lbs.

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

Make your selection now: 3
Bag has 2 packages and weights 16.0 lbs.

Make your selection now: 4
Sample bag has 1 packages and weights 0.6 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 Squirtle. May I keep a sample (Y/N)? Y
Your package contains:
50 Squirtles each weighing 0.1 from Squirtle to Squirtle
 Thanks for letting me keep a Squirtle!

Make your selection now: 3
Bag has 3 packages and weights 34.6 lbs.

Make your selection now: 4
Sample bag has 1 packages and weights 0.1 lbs.

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

Make your selection now: 0
Exiting program...Good Bye
::::::::::::::
Problem4_allfiles
::::::::::::::
::::::::::::::
Bag.java
::::::::::::::
public class Bag <T> {
    protected int packages;
    protected float weight;
    protected DEQ<T> bag;

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

    public void pickUpOrder(Sample pack) throws QueueException
    {
        bag.enqueue(pack);
        packages++;
        weight += (pack.getItemWeight() * pack.getItemAmount());

    }

    public void pickUpExpressOrder(Package expPack) {
        bag.enqueueFirst(expPack);
        packages++;
        weight += (expPack.getItemWeight() * expPack.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 Queue<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;
    }

}
::::::::::::::
DEQ.java
::::::::::::::

public class DEQ<T> extends Queue<T> implements ExtendedQueueInterface<T> {


    public DEQ()
    {
        super();
    }

    @Override
    public void enqueueFirst(Object newItem) throws ExtendedQueueException {
        if(numItems == items.length)
        {
            super.resize();
        }

        front = (front + items.length-1)%items.length;

        items[front] = (T) newItem;
        numItems++;
    }

    @Override
    public T dequeueLast() throws ExtendedQueueException {
        T result = null;
        if(numItems > 0)
        {
            back = (back + items.length-1)%items.length;
            result = items[back];
            items[back] = null;
            numItems--;
        }
        return result;
    }

    @Override
    public T peekLast() throws ExtendedQueueException {
        T result = null;
        if(numItems > 0)
        {
            result = items[(back + items.length-1)%items.length];
        }
        else
        {
            System.out.println("Queue empty!");
        }
        return result;
    }

}
::::::::::::::
Driver.java
::::::::::::::
/*

 * Purpose: Data Structure and Algorithms Lab 6 Problem 4

 * Status: Complete and thoroughly tested

 * Last update: 03/03/2020

 * Submitted:  03/03/2020

 * Comment: test suite and sample run attached

 * @author: Matthew Ryan

 * @version: 2020.03.03

 */

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, QueueException {


        Bag<T> bag = new Bag<T>();
        SampleBag<T> samples = new SampleBag<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."
                            + "\n\t7. Pick up an express order.");

        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 + "s each weighing " + w
eight + " 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().dequeue();


                    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;

            case 7:

                String expName = "";
                String expSender = "";
                String expRecipient = "";
                float expWeight = 0;
                int expAmount = 0;

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

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

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

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

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

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

                Package expPack = new Package(expName, expWeight, expAmount, expSe
nder, expRecipient);
                bag.pickUpExpressOrder(expPack);
                System.out.println("A package of " + expName + " each weighing " +
 expWeight + " lbs are now in the bag.");
                break;

            default:
                break;
            }
        }
    }
}


::::::::::::::
ExtendedQueueException.java
::::::::::::::

public class ExtendedQueueException extends RuntimeException {

    public ExtendedQueueException(String s) {
        super(s);
    }  // end constructor
}  // end ExtendedQueueException::::::::::::::
ExtendedQueueInterface.java
::::::::::::::

public interface ExtendedQueueInterface<T> extends QueueInterface<T> {
    public void enqueueFirst(T newItem) throws ExtendedQueueException;
    public T dequeueLast() throws ExtendedQueueException;
    public T peekLast() throws ExtendedQueueException;
}  // end ExtendedQueueInterface
::::::::::::::
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 extends Sample {

    public Package(String name, float weight, int amount, String sender, String re
ceiver)
    {
        super(name, weight, amount, sender, receiver);
    }
}
::::::::::::::
QueueException.java
::::::::::::::
public class QueueException extends Throwable {

    public QueueException(String s) {
        super(s);
    }  // end constructor
}  // end QueueException::::::::::::::
QueueInterface.java
::::::::::::::
public interface QueueInterface<T> {

    public boolean isEmpty();
    // Determines whether a queue is empty.
    // Precondition: None.
    // Postcondition: Returns true if the queue is empty;
    // otherwise returns false.

    public void enqueue(T newItem) throws QueueException;
    // Adds an item at the back of a queue.
    // Precondition: newItem is the item to be inserted.
    // Postcondition: If the operation was successful, newItem
    // is at the back of the queue. Some implementations
    // may throw QueueException if newItem cannot be added
    // to the queue.

    public T dequeue() throws QueueException;
    // Retrieves and removes the front of a queue.
    // Precondition: None.
    // Postcondition: If the queue is not empty, the item that
    // was added to the queue earliest is removed. If the queue is
    // empty, the operation is impossible and QueueException is thrown.

    public void dequeueAll();
    // Removes all items of a queue.
    // Precondition: None.
    // Postcondition: The queue is empty.

    public T peek() throws QueueException;
    // Retrieves the item at the front of a queue.
    // Precondition: None.
    // Postcondition: If the queue is not empty, the item
    // that was added to the queue earliest is returned.
    // If the queue is empty, the operation is impossible
    // and QueueException is thrown.

    public String toString();
}  // end QueueInterface

::::::::::::::
Queue.java
::::::::::::::

public class Queue<T> implements QueueInterface<T>  {

    protected int numItems;
    protected int front;
    protected int back;
    protected T[] items;

    public Queue()
    {
        numItems = 0;
        back = 0;
        front = 0;
        items = (T[]) new Object[3];
    }


    @Override
    public boolean isEmpty() {
        return (numItems == 0);
    }

    @Override
    public void enqueue(Object newItem) throws QueueException {
        if(numItems == items.length)
        {
            resize();
        }
        items[back] = (T) newItem;
        back = (back+1)%items.length;
        numItems++;
    }

    @Override
    public T dequeue() throws QueueException {
        T result = null;
        if(numItems > 0)
        {
            result = items[front];
            items[front] = null;
            front = (front + 1)%items.length;
            numItems--;
        }
        return result;
    }
```

```
@Override
public void dequeueAll() {
    numItems = 0;
    back = 0;
    front = 0;
    items = (T[]) new Object[3];
}

@Override
public T peek() throws QueueException {

    T result = null;
    if(numItems > 0)
    {
        result = items[front];
    }
    else
    {
        System.out.println("Queue empty!");
    }
    return result;

}

public String toString()
{
    StringBuilder builder = new StringBuilder();
    String toReturn = "";
    int counter = 0;

    for(int i = front; counter < numItems; counter++)
    {
        String build = items[i].toString() + " ";
        builder.append(build);
        i = ((i+1)%items.length);
    }


    return toReturn = builder.toString();

}

protected void resize() {
    T[] temp = (T[]) new Object[items.length+1];
    System.out.println();
    int counter = 0;
    for(int i = front; counter < numItems; counter++)
    {
        temp[counter] = items[i];
        i = (i+1) % items.length;
    }

    items = temp;
    front = 0;
    back = numItems;

}
```

```
}
::::::::::::::
QueueSLS.java
::::::::::::::

public class QueueSLS<T> implements QueueInterface {

    Node<T> front;
    Node<T> back;

    public QueueSLS()
    {
        front = null;
        back = null;
    }

    @Override
    public boolean isEmpty() {

        if(front == null)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    @Override
    public void enqueue(Object newItem) throws QueueException {
        if(back == null)
        {
            front = back = new Node(newItem);
        }
        else
        {
            Node temp = new Node(newItem);
            back.setNext(temp);
            back = temp;
        }
    }

    @Override
    public Object dequeue() throws QueueException {

        Object result = null;

        if(front.getNext() != null)
        {
            result = front.getItem();
            front = front.getNext();
            if(front == null)
            {
                back = null;
            }
        }

        return result;
    }

    @Override
```

```
    public void dequeueAll() {
        front = null;
        back = null;
    }

    @Override
    public Object peek() throws QueueException {
        return front.getItem();
    }

    public String toString()
    {
        Node<T> next = front;
        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;

    }

}
::::::::::::::
SampleBag.java
::::::::::::::

public class SampleBag<T> {

    private int packages;
    private float weight;
    private StackSLS<T> bag;

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

    public void pickUpOrder(Sample pack) throws QueueException
    {
        bag.push(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 StackSLS<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;
    }


}
::::::::::::::
Sample.java
::::::::::::::
public class Sample {

    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)
    {
        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;
    }
}
::::::::::::::
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::::::::::::::
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.
        7. Pick up an express order.

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 apples 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 oranges each weighing 0.85 lbs are now in the bag.

Make your selection now: 7
Please specify express package 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: 7
Please specify express package info:
Item name: mango

Item weight: 0.3

# of items: 12

Sender: Victini

Recipient: Meloetta

A package of mango each weighing 0.3 lbs are now in the bag.

Make your selection now: 3
Bag has 4 packages and weights 27.800001 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 Meloetta. May I keep a sample (Y/N)? Y
Your package contains:
12 mangos each weighing 0.3 from Victini to Meloetta
 Thanks for letting me keep a mango!

Make your selection now: 4
Sample bag has 1 packages and weights 0.3 lbs.

Make your selection now: 3
Bag has 3 packages and weights 24.2 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 cookies 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 bananas each weighing 0.5 lbs are now in the bag.

Make your selection now: 4
Sample bag has 1 packages and weights 0.3 lbs.

Make your selection now: 3
Bag has 5 packages and weights 40.2 lbs.

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

Make your selection now: 4
Sample bag has 1 packages and weights 0.3 lbs.

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

Make your selection now: 2
Here is your package Mew. May I keep a sample (Y/N)? Y
Your package contains:
10 apples each weighing 0.6 from Pickachu to Mew
 Thanks for letting me keep a apple!

Make your selection now: 4
Sample bag has 2 packages and weights 0.90000004 lbs.

Make your selection now: 3
Bag has 3 packages and weights 27.900002 lbs.

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

Make your selection now: 3
Bag has 3 packages and weights 27.900002 lbs.

Make your selection now: 4
Sample bag has 1 packages and weights 0.3 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 granolas each weighing 0.5 lbs are now in the bag.

Make your selection now: 7
Please specify express package 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 4 packages and weights 40.4 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
::::::::::::::
Lab6Conclusions.txt
::::::::::::::
Lab 6 was... in a word, challenging.

It was a genuine pain in the rear to get working at points, but I knew how the
lab was supposed to function once I got over my personal chokepoint in Problem 2
and that's what I had to aim for.

 And even then, my biggest challenge in Problem 2 was properly updating Front
 and Back for the DEQ, which reminded me that math is simultaneously complicated
 and incredibly simple.

I see the place for Queues; they're a smaller version of the CDLS while
retaining part of the functionality. That being said, the CDLS's functions are
simpler while giving up that space.

Its an interesting tradeoff for space VS ease of use, and something I'll have to h
eavily considering going forward.
```