CS162

ASSIGNMENT 7

NAME:

ARCHIT AGRAWAL

ROLL NO. :

202052307

SECTION:

A

Question

1. **Implement a Queue using ArrayList**
    a. The name of the class should be “QueueAL”
    b. The class should have all the functions as described in the lecture.

2. **Implement a Queue using LinkedList**
    a. The name of the class should be “QueueLL”
    b. The class should have all the functions as described in the lecture.

3. **Create a Driver/Application class to run the Queue**
    a. The driver class should have a main method.
    b. While creating the list either insert the letters of your name as input or the digits of your roll number as the input.
    c. The driver class should create an object of the **QueueAL** class and **QueueLL** class.
    d. The driver class should call all the functions of the **QueueAL** class and **QueueLL** class.

4. **Bonus question (optional)**
    a. Implement stack using queue
    b. Implement queue using stack

CODE

```
import java.util.*;

class ArrayList <T>{

    protected T[] arr;
    protected int size;

    public ArrayList(int initialCapacity){
        if(initialCapacity < 1){
            throw new IllegalArgumentException("Initial Capacity must be >= 1"
);
        }
        arr = (T[])new Object[initialCapacity];
        size = 0;
    }

    public ArrayList(){
        this(10);
    }

    public void addToFront(T obj){
        add(obj, 0);
    }

    public void addToRear(T obj){
        add(obj, size);
    }

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

    public int size(){
        return size;
    }

    public T removeFront(){
        if(size == 0){
            throw new IllegalArgumentException("Array List is Empty");
        }
        T obj = remove(0);
```

```
        return obj;
    }

    public T removeRear(){
        if(size == 0){
            throw new IllegalArgumentException("Array List is Empty");
        }
        T obj = remove(size - 1);
        return obj;
    }

    public void checkIndex(int index){
        if(index < 0 || index > size - 1){
            throw new IndexOutOfBoundsException("Index = "+ index + " Size = "
+size);
        }
    }

    public T get(int index){
        checkIndex(index);
        return arr[index];
    }

    public Integer indexOf(T obj){
        for(int i = 0; i < size; i++){
            //the 'if' condition used is just to make sure that there will be
no 'NullPointerException'
            if((arr[i] == null && obj == null) || (arr[i] != null && arr[i].eq
uals(obj))){
                return i;
            }
        }
        return -1;
    }

    public T remove(int index){
        checkIndex(index);
        T temp = arr[index];
        for(int i = index + 1; i < size; i++){
            arr[i - 1] = arr[i];
        }
        size--;
        arr[size] = null;
        return temp;
    }
```

```
    public void changeSize(){
        T []newArray = (T[])new Object[2 * size];
        int i = 0;
        for(T o : arr){
            newArray[i++] = o;
        }
        this.arr = newArray;
    }

    protected void add(T obj, int index){
        if(index < 0 || index > size){
            throw new IndexOutOfBoundsException("Index = "+ index +" Size = "+
size);
        }
        //checking capacity
        if(size == arr.length){
            changeSize();
        }
        //shift elements to right
        for(int i = size; i > index; i--){
            arr[i] = arr[i - 1];
        }
        arr[index] = obj;
        size++;
    }

    public String toString(){

        StringBuilder sb = new StringBuilder();
        sb.append("[");
        for(int i = 0; i < size; i++){
            sb.append((arr[i] != null ? arr[i].toString() : "null"));
            if(i < size - 1){
                sb.append(", ");
            }
        }
        sb.append("]");
        return sb.toString();
    }

}

class QueueAL<T> extends ArrayList<T> {
    QueueAL(){
        super();
    }
```

```
public void enqueue(T data){
    if(size == 100000){
        throw new QueueOverflowException();
    }
    super.add(data, size);
}

public T dequeue(){
    if(super.size() == 0){
        throw new QueueUnderflowException();
    }
    return super.remove(0);
}

public T getFrontElement(){
    if(super.isEmpty()){
        throw new QueueEmptyException();
    }
    return super.get(size - 1);
}

public T getRearElement(){
    if(super.isEmpty()){
        throw new QueueEmptyException();
    }
    return super.get(0);
}

public int search(T data){
    int i = super.indexOf(data);
    return i == -1 ? -1 : (size - i);
}

public String display(){
    return super.toString();
}

public void addToFront(T obj){}

public void addToRear(T obj){}

public T remove(int index){
    return null;
}
```

```
    public T removeFront(){
        return null;
    }

    public T removeRear(){
        return null;
    }

    public T get(int index){
        return null;
    }

    public void checkIndex(int index){}

    public String toString(){
        return null;
    }

    public Integer indexOf(T obj){
        return null;
    }
}

class Node<T>{
    T data;
    Node<T> next;

    public Node(){}

    public Node(T element){
        this.data = element;
    }

    public Node(T element, Node addr){
        this.data = element;
        this.next = addr;
    }
}

class LinkedList<T>{

    protected Node<T> head;
    protected int size;

    public LinkedList(){
        head = null;
```

```
        size = 0;
    }

    //The method append adds an element at the end (index = size) of the linke
d list.
    public void append(T data){
        Node<T> newNode = new Node<T>(data);
        if (head == null) {
            head = newNode;
        } else {
            Node <T> lastNode = head;
            while (lastNode.next != null) {
                lastNode = lastNode.next;
            }
            lastNode.next = newNode;
        }
        size++;
    }

    //The method add adds a given element at the given index.
    public void add(int index, T data){

        if(index < 0 || index > size){
            throw new IndexOutOfBoundsException("Index = "+index +" Size = "+s
ize);
        }

        if(index == 0){
            Node<T> newNode = new Node<T>(data);
            newNode.next = head;
            head = newNode;
        } else {
            Node<T> newNode = new Node<T>(data);
            Node<T> prevNode = head;
            int i = 0;
            while(i < index - 1){
                prevNode = prevNode.next;
                i++;
            }
            newNode.next = prevNode.next;
            prevNode.next = newNode;
        }
        size++;
    }
```

```
    //The method removeKey removes the first occurence of the element passed t
o it from the linked list.
    public void removeKey(T key){
        if(size == 0){
            return;
        }
        if(head.data == key){
            head = head.next;

        } else {
            Node<T> currNode = head.next;
            Node<T> prevNode = head;

            while(currNode != null){
                //to prevent NullPointerException
                if(currNode.data.equals(key)){
                    prevNode.next = currNode.next;
                    break;
                } else {
                    prevNode = currNode;
                    currNode = currNode.next;
                }
            }
        }
        size--;
    }

    //The method removeIndex removes the element from the index passed to it a
nd returns the removed element.
    public T removeIndex(int index){
        checkIndex(index);
        T removed;

        if(index == 0){
            removed = head.data;
            head = head.next;
        } else {
            Node<T> currNode = head.next;
            Node<T> prevNode = head;
            int i = 0;
            while(i < index - 1){
                prevNode = prevNode.next;
                currNode = currNode.next;
                i++;
            }
            removed = currNode.data;
```

```
            prevNode.next = currNode.next;
        }
        size--;
        return removed;
    }

    //This method prints the linked list in formatted way
    public void printLinkedList() {
        Node<T> currentNode = head;
        while(currentNode != null){
            System.out.print(currentNode.data + " -> ");
            currentNode = currentNode.next;
        }
        System.out.println();
    }

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

    public int size(){
        return size;
    }

    public void checkIndex(int index){
        if(index < 0 || index >= size){
            throw new IndexOutOfBoundsException("Index = "+index +" size = "+
size);
        }
    }

    public T get(int index){
        checkIndex(index);
        Node<T> currNode = head;
        int i = 0;
        while(i != index){
            currNode = currNode.next;
            i++;
        }
        return currNode.data;
    }

    public Integer indexOf(T data){
        Node<T> temp = head;
        int index = 0;
        while(temp != null){
```

```
            if(temp.data.equals(data)){
                return index;
            }
            index++;
            temp = temp.next;
        }
        return -1;
    }

    public String toString(){
        if(head == null){
            return "[]";
        }
        StringBuilder sb = new StringBuilder();
        sb.append("[");
        Node<T> currentNode = head;
        while(currentNode.next != null){
            sb.append(currentNode.data.toString() + ", ");
            currentNode = currentNode.next;
        }
        sb.append(currentNode.data.toString() + "]");
        return sb.toString();
    }

}

class QueueLL<T> extends LinkedList<T>{

    public QueueLL(){
        super();
    }

    public void enqueue(T data){
        if(size == 100000){
            throw new QueueOverflowException();
        }
        super.append(data);
    }

    public T dequeue(){
        if(super.isEmpty()){
            throw new QueueUnderflowException();
        }
        return super.removeIndex(0);
    }
```

```
    public T getFrontElement(){
        if(super.isEmpty()){
            throw new QueueEmptyException();
        }
        return super.get(size - 1);
    }

    public T getRearElement(){
        if(super.isEmpty()){
            throw new QueueEmptyException();
        }
        return super.get(0);
    }

    public int search(T data){
        int i = super.indexOf(data);
        return i == -1 ? -1 : (size - i);
    }

    public String display(){
        return super.toString();
    }

    public void append(T data){}

    public void add(int index, T data){}

    public void removeKey(T key){}

    public T removeIndex(int index){
        return null;
    }

    public void printLinkedList(){}

    public void checkIndex(int index){}

    public T get(int index){
        return null;
    }

    public String toString(){
        return null;
    }

    public Integer indexOf(T data){
```

```
        return null;
    }

}

class QueueOverflowException extends RuntimeException{
    public QueueOverflowException(){
        super("Queue Overflow");
    }
}

class QueueUnderflowException extends RuntimeException{
    public QueueUnderflowException(){
        super("Queue Underflow");
    }
}

class QueueEmptyException extends RuntimeException{
    public QueueEmptyException(){
        super("Queue is Empty");
    }
}

public class Driver{
    public static void main(String []args){

        System.out.println("Initializing a queue using ArrayList");
        QueueAL <Integer> queueALObject = new QueueAL<Integer>();
        System.out.println();

        System.out.print("Checking if Queue is empty... :");
        if(queueALObject.isEmpty()){
            System.out.println("Queue is empty");
        } else {
            System.out.println("Queue is not empty");
        }


        System.out.println("The size of queue is : "+queueALObject.size());
        System.out.println();

        System.out.println("The current queue is(front element is at rightmost
) : "+queueALObject.display());

        System.out.println();
        System.out.println("Enqueuing few elements in the queue..");
```

```
        queueALObject.enqueue(8);
        queueALObject.enqueue(9);
        queueALObject.enqueue(9);
        queueALObject.enqueue(2);
        queueALObject.enqueue(0);

        System.out.println();
        System.out.println("The current queue is(front element is at rightmost
) : "+queueALObject.display());
        System.out.println("The size of queue is : "+queueALObject.size());
        System.out.print("Checking if Queue is empty... : ");
        if(queueALObject.isEmpty()){
            System.out.println("Queue is empty");
        } else {
            System.out.println("Queue is not empty");
        }
        System.out.println();

        System.out.println("Enqueuing few more elements in the queue...");
        queueALObject.enqueue(2);
        queueALObject.enqueue(0);
        queueALObject.enqueue(5);
        queueALObject.enqueue(2);
        System.out.println();
        System.out.println("The current queue is(front element is at rightmost
) : "+queueALObject.display());
        System.out.println();

        System.out.println("The front element in the queue is : "+queueALObjec
t.getFrontElement());
        System.out.println();
        System.out.println("The rear element in the queue is : "+queueALObject
.getRearElement());
        System.out.println();
        System.out.println("The location of element 5 from front of queue is :
 "+queueALObject.search(5));
        System.out.println();
        System.out.println("Enqueuing few more elements in the queue..");

        queueALObject.enqueue(3);
        queueALObject.enqueue(0);
        queueALObject.enqueue(7);

        System.out.println("The current queue is(front element is at rightmost
) : "+queueALObject.display());
        System.out.println();
```

```
        System.out.println("Dequeuing few elements from the queue...");
        queueALObject.dequeue();
        queueALObject.dequeue();
        System.out.println();

        System.out.println("The current queue is(front element is at rightmost
) : "+queueALObject.display());
        System.out.println();
        System.out.println("Dequeuing another element from the queue, the dequ
eued element is : "+queueALObject.dequeue());
        System.out.println();
        System.out.println("The current queue is(front element is at rightmost
) : "+queueALObject.display());

        System.out.println("_____________________________________________
_____________________________________________
_____________________");
        System.out.println("_____________________________________________
_____________________________________________
_____________________");

        System.out.println();

        System.out.println("Initializing a queue using LinkedList");
        QueueLL <Character> queueLLObject = new QueueLL<Character>();
        System.out.println();

        System.out.print("Checking if Queue is empty... :");
        if(queueLLObject.isEmpty()){
            System.out.println("Queue is empty");
        } else {
            System.out.println("Queue is not empty");
        }

        System.out.println("The size of queue is : "+queueLLObject.size());
        System.out.println();

        System.out.println("The current queue is(front element is at rightmost
) : "+queueLLObject.display());

        System.out.println();
        System.out.println("Enqueuing few elements in the queue..");
        queueLLObject.enqueue('A');
        queueLLObject.enqueue('G');
        queueLLObject.enqueue('R');
        queueLLObject.enqueue('A');
```

```
        queueLLObject.enqueue('W');
        queueLLObject.enqueue('A');
        queueLLObject.enqueue('L');

        System.out.println();
        System.out.println("The current queue is(front element is at rightmost
) : "+queueLLObject.display());
        System.out.println("The size of queue is : "+queueLLObject.size());
        System.out.print("Checking if Queue is empty... :");
        if(queueLLObject.isEmpty()){
            System.out.println("Queue is empty");
        } else {
            System.out.println("Queue is not empty");
        }
        System.out.println();

        System.out.println("Enqueuing few more elements in the queue...");
        queueLLObject.enqueue('A');
        queueLLObject.enqueue('R');
        queueLLObject.enqueue('C');
        queueLLObject.enqueue('H');
        queueLLObject.enqueue('I');
        queueLLObject.enqueue('T');

        System.out.println("The current queue is(front element is at rightmost
) : "+queueLLObject.display());
        System.out.println();

        System.out.println("The element at the front of queue : "+queueLLObjec
t.getFrontElement());
        System.out.println();
        System.out.println("The element at the rear of queue : "+queueLLObject
.getRearElement());
        System.out.println();
        System.out.println("The position of the element 'G' from front is : "+
queueLLObject.search('G'));

        System.out.println();
        System.out.println("Dequeuing few elements from the queue...");

        for(int i = 0; i < 6; i++){
            queueLLObject.dequeue();
        }

        System.out.println("The current queue is(front element is at rightmost
) : "+queueLLObject.display());
```

```
        System.out.println();

        System.out.println("Dequeuing another element from the queue, the dequ
eued element is : "+queueLLObject.dequeue());
        System.out.println();
        System.out.println("The current queue is(front element is at rightmost
) : "+queueLLObject.display());

    }
}
```

OUTPUT

```
Initializing a queue using ArrayList

Checking if Queue is empty... : Queue is empty
The size of queue is : 0

The current queue is(front element is at rightmost) : []

Enqueuing few elements in the queue..

The current queue is(front element is at rightmost) : [8, 9, 9, 2, 0]
The size of queue is : 5
Checking if Queue is empty... : Queue is not empty

Enqueuing few more elements in the queue...

The current queue is(front element is at rightmost) : [8, 9, 9, 2, 0, 2, 0, 5, 2]

The front element in the queue is : 2

The rear element in the queue is : 8

The location of element 5 from front of queue is : 2

Enqueuing few more elements in the queue..
The current queue is(front element is at rightmost) : [8, 9, 9, 2, 0, 2, 0, 5, 2, 3, 0, 7]

Dequeuing few elements from the queue...

The current queue is(front element is at rightmost) : [9, 2, 0, 2, 0, 5, 2, 3, 0, 7]

Dequeuing another element from the queue, the dequeued element is : 9

The current queue is(front element is at rightmost) : [2, 0, 2, 0, 5, 2, 3, 0, 7]
```

```
Initializing a queue using LinkedList

Checking if Queue is empty... :Queue is empty
The size of queue is : 0

The current queue is(front element is at rightmost) : []

Enqueuing few elements in the queue..

The current queue is(front element is at rightmost) : [A, G, R, A, W, A, L]
The size of queue is : 7
Checking if Queue is empty... :Queue is not empty

Enqueuing few more elements in the queue...
The current queue is(front element is at rightmost) : [A, G, R, A, W, A, L, A, R, C, H, I, T]

The element at the front of queue : T

The element at the rear of queue : A

The position of the element 'G' from front is : 12

Dequeuing few elements from the queue...
The current queue is(front element is at rightmost) : [L, A, R, C, H, I, T]

Dequeuing another element from the queue, the dequeued element is : L

The current queue is(front element is at rightmost) : [A, R, C, H, I, T]
PS C:\Users\Archit\Desktop\cprog>
```

Code for throwing Exceptions -:

The changes required in the code above in order to throw exceptions are to be done in Driver class only.

Hence, instead of pasting the whole code again and again, only the class Driver code is pasted.

- *For throwing QueueOverflowException while adding more elements than the capacity of Queue using ArrayList*
  (Just to show this exception the Queue capacity was reduced to 5, in the code given at starting this capacity is 100,000.)

```
public class Driver{
    public static void main(String []args){

        System.out.println("Initializing a queue using ArrayList");
        QueueAL <Integer> queueALObject = new QueueAL<Integer>();
        System.out.println();
        System.out.println("Enqueuing few elements in the queue..");
        queueALObject.enqueue(8);
        queueALObject.enqueue(9);
        queueALObject.enqueue(9);
        queueALObject.enqueue(2);
        queueALObject.enqueue(0);
        System.out.println("The current queue is(front element is at ri
ghtmost) : "+queueALObject.display());
        System.out.println("Enqueuing another element in the queue..");
        queueALObject.enqueue(25);

    }
}
```

Runtime Errors:

```
Exception in thread "main" QueueOverflowException: Queue Overflow
        at QueueAL.enqueue(Driver.java:132)
        at Driver.main(Driver.java:464)
```

Output:

Copy

```
Initializing a queue using ArrayList

Enqueuing few elements in the queue..
The current queue is(front element is at rightmost) : [8, 9, 9, 2, 0]
Enqueuing another element in the queue..
```

- *For throwing QueueUnderflowException when dequeue method is called on empty Queue using ArrayList*

```
public class Driver{
    public static void main(String []args){

        System.out.println("Initializing a queue using ArrayList");
        QueueAL <Integer> queueALObject = new QueueAL<Integer>();

        System.out.print("Checking if Queue is empty... : ");
        if(queueALObject.isEmpty()){
```

```
            System.out.println("Queue is empty");
        } else {
            System.out.println("Queue is not empty");
        }

        queueALObject.dequeue();
    }
}
```

Runtime Errors:

```
Exception in thread "main" QueueUnderflowException: Queue Underflow
        at QueueAL.dequeue(Driver.java:139)
        at Driver.main(Driver.java:463)
```

Output:

Copy

```
Initializing a queue using ArrayList
Checking if Queue is empty... : Queue is empty
```

- *For throwing QueueEmptyException when getFrontElement or getRearElement method is called on empty Queue using ArrayList*

```
public class Driver{
    public static void main(String []args){

        System.out.println("Initializing a queue using ArrayList");
        QueueAL <Integer> queueALObject = new QueueAL<Integer>();

        System.out.print("Checking if Queue is empty... : ");
        if(queueALObject.isEmpty()){
            System.out.println("Queue is empty");
        } else {
            System.out.println("Queue is not empty");
        }

        queueALObject.getFrontElement(); // or queueALObject.getRearEle
ment();
    }
}
```

Runtime Errors:

```
Exception in thread "main" QueueEmptyException: Queue is Empty
        at QueueAL.getFrontElement(Driver.java:146)
        at Driver.main(Driver.java:463)
```

Output: Copy

```
Initializing a queue using ArrayList
Checking if Queue is empty... : Queue is empty
```

- *For throwing QueueOverflowException while adding more elements than the capacity of Queue using LinkedList*

(Just to show this exception the Queue capacity was reduced to 6, in the code given at starting this capacity is 100,000.)

```
public class Driver{
    public static void main(String []args){

        System.out.println("Initializing a queue using LinkedList");
        QueueLL <Character> queueLLObject = new QueueLL<Character>();
        System.out.println();

        System.out.println("Enqueuing few more elements in the queue...
");
        queueLLObject.enqueue('A');
        queueLLObject.enqueue('R');
        queueLLObject.enqueue('C');
        queueLLObject.enqueue('H');
        queueLLObject.enqueue('I');
        queueLLObject.enqueue('T');

        System.out.println("The current queue is(front element is at ri
ghtmost) : "+queueLLObject.display());
        System.out.println("Enqueuing another element in the queue...")
;
        queueLLObject.enqueue('T');
    }
}
```

Runtime Errors:

```
Exception in thread "main" QueueOverflowException: Queue Overflow
        at QueueLL.enqueue(Driver.java:387)
        at Driver.main(Driver.java:467)
```

Output:

Copy

```
Initializing a queue using LinkedList

Enqueuing few more elements in the queue...
The current queue is(front element is at rightmost) : [A, R, C, H, I, T]
Enqueuing another element in the queue...
```

- *For throwing QueueUnderflowException when dequeue method is called on empty Queue using LinkedList*

```
public class Driver{
    public static void main(String []args){

        System.out.println("Initializing a queue using LinkedList");
        QueueLL <Character> queueLLObject = new QueueLL<Character>();
        System.out.println();

        System.out.print("Checking if Queue is empty... : ");
        if(queueLLObject.isEmpty()){
            System.out.println("Queue is empty");
        } else {
            System.out.println("Queue is not empty");
        }

        queueLLObject.dequeue();
    }
}
```

Runtime Errors:

```
Exception in thread "main" QueueUnderflowException: Queue Underflow
        at QueueLL.dequeue(Driver.java:394)
        at Driver.main(Driver.java:464)
```

Output:

Copy

```
Initializing a queue using LinkedList

Checking if Queue is empty... : Queue is empty
```

- *For throwing QueueEmptyException when getFrontElement or getRearElement method is called on empty Queue using LinkedList*

```
public class Driver{
    public static void main(String []args){

        System.out.println("Initializing a queue using LinkedList");
        QueueLL <Character> queueLLObject = new QueueLL<Character>();
        System.out.println();

        System.out.print("Checking if Queue is empty... : ");
        if(queueLLObject.isEmpty()){
            System.out.println("Queue is empty");
        } else {
            System.out.println("Queue is not empty");
        }

        queueLLObject.getRearElement();
    }
}
```

Runtime Errors:

```
Exception in thread "main" QueueEmptyException: Queue is Empty
        at QueueLL.getRearElement(Driver.java:408)
        at Driver.main(Driver.java:464)
```

Output:

Copy

```
Initializing a queue using LinkedList

Checking if Queue is empty... : Queue is empty
```