```
::::::::::::::
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!
```