Lab 10: Queue (Using Linked List)

Objective(s)

- Queue implementation using LinkedList (Linear and Circular)
- Double Ended Queue (DEQue) implementation using DoublyLinkedList.

Tool(s)/Software

Java programming language with NetBeans IDE.

Description

Implementing a Linear **Queue** using a Singly **Linked-List** and Implementing a Circular **Queue** using a Circular **Linked-List**:

- The **Front** of the **Queue** is the **head.** Dequeue off the Queue at front (**deleteFirst**).
- The **Rear** of the **Queue** is the **tail**. Enqueue onto the Queue at rear (**addLast**).

Implementing a Double-Ended Queue (DEQueue) using a **Doubly Linked-List:**

- Can add and delete from both front and rear
- EnQueueFront, EnQueueRear, DeQueueFront, DeQueueRear

A. Linear Queue implementation using Singly Linked List (*QueueLinkedList.java*):

```
public class QueueLinkedList {

    private class Node{...15 lines }

    private int count = 0;
    Node front = null;
    Node rear  = null;

    public int getSize(){...3 lines }

    public boolean isEmpty(){...3 lines }

    public String getFront(){...5 lines }

    public String getRear(){...5 lines }

    public void EnQueue(int value){...14 lines }

    public  int  DeQueue(){...12 lines }

    public void display(){...8 lines }
```

```
private class Node{
    private int element;
    private Node link;

    public Node(int value, Node n){
        this.element = value;
        link = n;
    }
    public void setLink(Node n){
        link = n;
    }
    public Node getLink(){
        return link;
    }
}
```

EnQueue and DeQueue in ***QueueLinkedList*** *class*:

```
public void EnQueue(int value){
    Node newNode = new Node(value, null);
    if (isEmpty())//first node to be added
    {
        front = rear = newNode;
    } else {
        rear.next = newNode;
        rear = newNode;
    }
    count++;
}
```

```
public void DeQueue()
{
    if(isEmpty())
    {
        System.out.println("Underflow"); return;
    }
    System.out.println(front.element+" - Deleted from Queue");
    front=front.next;
    count--;

}
```

In the main, create object and try the queue operations:

```
    public static void main(String[] args) {
        QueueLinkedList Q = new QueueLinkedList();
        Q.EnQueue(4);
        Q.EnQueue(20);
        Q.EnQueue(50);
        Q.EnQueue(10);
        Q.EnQueue(60);
        Q.EnQueue(90);
        Q.display();
        Q.DeQueue();

//          checking other methods
        System.out.println("\n--------- Checking other methods ---------");
        System.out.println(" getFront(): " + Q.getFront());
        System.out.println(" getRear(): " + Q.getRear());
        System.out.println(" getSize(): " + Q.getSize());
//
        Q.DeQueue();
        Q.DeQueue();

        // See the output of this operation
        Q.EnQueue(90);

        Q.DeQueue();
        Q.DeQueue();
        Q.DeQueue();
        Q.DeQueue();
        Q.display();

        // checking other methods
        System.out.println("\n--------- Checking other methods ---------");
        System.out.println(" getFront(): " + Q.getFront());
        System.out.println(" getRear(): " + Q.getRear());
        System.out.println(" getSize(): " + Q.getSize());
    }

}
```

B. Double-Ended Queue (DEQueue) using a Doubly Linked-List (*DoubleEndedQueue.java*):

```java
package queueapp;
public class DoubleEndedQueue {
     private class Node{
        private int element;
        private Node next;
        private Node prev;

        public Node(int element, Node next,Node prev) {
            this.element = element;
            this.next = next;
            this.prev = prev;
        }

        public int getElement() {
            return element;
        }

        public Node getNext() {
            return next;
        }

        public Node getPrev() {
            return prev;
        }

    }
```

```java
 Node front=null;
 Node rear=null;
 int count=0;
 public int size(){ return count;}
 public boolean isEmpty(){ return (count==0);}
 public String getFront(){
     if(isEmpty())
         return "Queue is Empty";
     else
         return front.element+"";
 }
  public String getRear(){
     if(isEmpty())
         return "Queue is Empty";
     else
         return rear.element+"";
 }
public void display()
 {
    if(isEmpty())
     {System.out.println("Empty Queue.");return;}
     Node current = front;
     while(current!=null){
         System.out.print(current.element + "    ");
         current = current.next;
     }
     System.out.println();
 }
```

EnQueueRear and DeQueueFront in ***DoubleEndedQueue*** *class*:

```
public void EnQueueRear(int value){
    Node newNode = new Node(value, null,null);
    if (isEmpty())//first node to be added
    {
        front = rear = newNode;
    } else {
        rear.next = newNode;
        newNode.prev=rear;
        rear = newNode;
    }
    count++;
}
```

```
public void DeQueueFront() {
    if (isEmpty()) {
        System.out.println("Underflow");
        return;
    }
    System.out.println(front.element + " - Deleted from Queue");
    if (count == 1) {
        front = rear = null;
        count = 0;
        return;
    }
    front = front.next;
    front.prev = null;
    count--;

}
```

In the main, create object and try the *DoubleEndedQueue* operations:

```java
package queueapp;
public class QueueApp {
    public static void main(String[] args) {

        DoubleEndedQueue DQ = new DoubleEndedQueue();
        DQ.EnQueueRear(10);
        DQ.EnQueueRear(20);
        DQ.EnQueueRear(30);
        DQ.EnQueueRear(40);
        while(! DQ.isEmpty())
            DQ.DeQueueFront();
    }
}
```

```
Output - QueueApp (run)
run:
10 - Deleted from Queue
20 - Deleted from Queue
30 - Deleted from Queue
40 - Deleted from Queue
BUILD SUCCESSFUL (total time: 0 seconds)
```

Tasks/Assignments(s)

- Write your own program to implement ***CircularQueue*** using the Circular Linked-List and implement the ***Enqueue*** and ***Dequeue*** methods.
- Modify ***QueueDoubleEnded*** class to add ***EnQueueFront*** and ***DeQueueRear*** methods.

Deliverables(s)

You are required to implement and deliver a Java program(s) as described in the previous section.