Lab 06

Queue

Objective:

This lab will introduce you the concept of Queue data structure

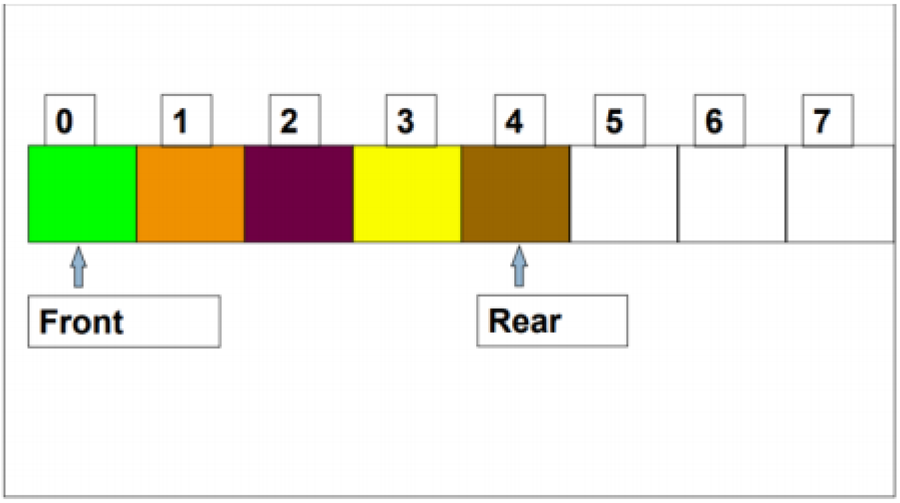
Activity Outcomes:

This lab teaches you the following topics:

- How to access the front and rear
- How to enqueuer/insert the data
- How to dequeuer/ delete the data

1) Useful Concepts

A queue is ordered collection of items which works according to FIFO (First In First Out) algorithm. Of the two ends of the queue, one is designated as the front − where elements are extracted (operation called dequeue), and another is the rear, where elements are inserted (operation called enqueue). A queue may be depicted as in Figure below:



The main operations are:
Enqueue − place an element at the tail of the queue;
Dequeue − take out an element form the front of the queue;
Delete − delete the whole queue

Activity 1:

Implement a Queue in Array, use class in the implementation so that Queue can be treated as an object and its operations as public interface for the user.

Interface in java:

```
public interface Queue {

	void enqueue(int x);
	int dequeue();
	int front();
	boolean isEmpty();
	boolean isFull();
	void display();
}




public class ArrayQueue implements Queue{

//Circular Array

	int front,rear,size,numElement;

	Integer[] arr;


	public ArrayQueue(int x) {
		this.front = 0;
		this.rear = 1;
		this.size = x;
		arr = new Integer[x];
		this.numElement = 0;
	}
```

```
    public static void main(String[] args) {
        ArrayQueue queue = new ArrayQueue(8);

        queue.enqueue(5);
        queue.enqueue(2);
        queue.enqueue(6);
        queue.enqueue(8);
        queue.enqueue(9);
        queue.enqueue(12);
        queue.enqueue(21);
        queue.enqueue(7);

        queue.display();
        System.out.println("Dequeue:");
        System.out.println(queue.dequeue());
        System.out.println(queue.dequeue());
        queue.display();

    }//end main

    @Override
    public void enqueue(int x) {

        rear = (rear+1)%size;
        arr[rear] = x;

        numElement++;

        if(front==0)
            front = rear;

    }

    @Override
    public int dequeue() {
        int x = arr[front];
        arr[front]=null;
        front = (front+1)%size;
        numElement--;
        return x;
    }

    @Override
    public int front() {

        return arr[front];
    }

    @Override
    public boolean isEmpty() {

        return (numElement == 0);
    }

    @Override
    public void display() {
        System.out.println("Display Queue: ");


        for(int i=0;i<size;i++)
        {


            System.out.println(arr[i]+" @ index= "+i);
        }

        System.out.println();
        System.out.println("front = index "+front);
        System.out.println("rear = index "+rear);
    }

    @Override
    public boolean isFull() {

        return (numElement==size);
    }
```

```
}//end class
```

Activity 2:

Implement Dynamic Queue.

```
public class Node {

    char value;
    Node nextNode;

    public Node(char value) {
        this.value = value;
        this.nextNode = null;
    }
}

public class LinkedListQueue implements Queue {

     private Node front;
     private Node rear;



     public LinkedListQueue() {
          this.front = null;
          this.rear = null;
     }

     public static void main(String[] args) {

          LinkedListQueue queue = new LinkedListQueue();

          System.out.println("is Empty: "+queue.isEmpty());

          System.out.println("Enqueue values 1 7 5 2");
          queue.enqueue(1);
        queue.enqueue(7);
        queue.enqueue(5);
        queue.enqueue(2);

        queue.display();

        System.out.println("\nDequeue");

        System.out.println(queue.dequeue());

        queue.display();

        System.out.println("\nFront value of queue:");
        System.out.println(queue.front());

        queue.display();

    }//end main

    @Override
    public void enqueue(int x) {

        if(front==null) {
            front = new Node(x);
            rear = front;
        }
        else
        {
            Node newNode = new Node(x);
            rear.nextNode = newNode;
            rear = newNode;
        }

    }//end enqueue

    @Override
    public int dequeue() {
        int x = -1;
        if(front!=null)
        {
            x = front.value;
            front = front.nextNode;

        }

        return x;
    }

    @Override
    public int front() {

        return front.value;
    }

    @Override
    public boolean isEmpty() {

        return (front==null);
    }

    @Override
    public void display() {
        System.out.println("Display Queue: ");
        Node current = front;
        while(current!=null) {
            System.out.print(current.value+" ");
            current = current.nextNode;
        }

    }//end display

    @Override
    public boolean isFull() {
        // TODO Auto-generated method stub
        return false;
    }

}//end class
```

Task

- Implement the dynamic queue data structure in a circular manner
    - Must know the front
    - Must know the rear
    - Must perform all queue operations such as enqueuer, dequeuer, delete all