Queues and Priority Queues

Objective

- To learn how Queue data structure can be implemented and how to perform basic queue operations using both array and linked list.
- To learn how priority Queue data structure can be implemented and how to perform basic queue operations using both array and linked list.

Task

1. Implement a queue using both array and linked list as shown below.

   a. Array based Queue (circular queue)

```
Class ArrQueue<T>{
            T[] Q;     int F;     int R;
   ArrQueue(){
   Q=(T[])new Object[10];
    F=0;    R=0;
      }
  Arr Queue(int size){
   Q=(T[])new Object[size];
   F=0;   R=0;
}
```

```
public void Enqueue(T obj){….}
Public T Dequeue(){….}
public boolean isEmpty(){…..}
public boolean isFull(){…..}
}// class end
```

   b. Linked list based queue (note: no need a circular linked list to build a queue)

```
class node<T>{
T data;
 node<T> next;

     //constructor
     …
}
```

```
class ListQueue<T>{
  node<T> Front;
  node<T>  Rear;
       public void Enqueue(T obj){… }
       Public T Dequeue(){ … }
       public boolean isEmpty(){ … }
}
```

   c. To test the implementation of the queue, demonstrate the operations of printing jobs in a queue.

```
public static void main(String[] args) {
   ArrQueue<String> Q1=new ArrQueue<String>(10);
    Q1.Enqueue("job1");
    Q1.Enqueue("job2");
    System.out.println(Q1.Dequeue());
    System.out.println(Q1.Dequeue());
    System.out.println(Q1.Dequeue());

//=================

ListQueue<String> Q2=new ListQueue<String>(10);
    Q2.Enqueue("job1");
    Q2.Enqueue("job2");
    System.out.println(Q2.Dequeue());
    System.out.println(Q2.Dequeue());
    System.out.println(Q2.Dequeue());
}
```

2. Implement a Priority Queue using both array and linked list as shown below.

   a. Array based implementation of Priority Queue

```
Class ArrPriorityQueue<T extends Comparable<T>>{
        T[] Q;
        int F;
         ArrPriorityQueue(){
         Q=(T[])new Comparable[10];
         F=-1;
        }
         ArrPriorityQueue(int size){
         Q=(T[])new Comparable[size];
         F=-1;
        }
            public void Enqueue(T obj){…}
            Public T Dequeue(){…}
            public boolean isEmpty(){…}
            public boolean isFull(){…}
            public String toString() {…}
        }
```

   b. Link list based implementation of priority Queue

```
class node<T>{
T data;
 node<T> next;

        //constructor
        …
       }
```

```
class ListPriorityQueue<T>{
  node<T> Front;
  node<T>  Rear;
        public void Enqueue(T obj){… }
        Public T Dequeue(){ … }
        public boolean isEmpty(){ … }
       public String toString() {…}

      }
```

   c. Test the above implementation of priority queues to demonstrate the operations on the printing job queue. Set a priority on the file size of the printing jobs in KBs. You must release the *shortest job first* (take file size in kb as input) from the priority queue to be assigned to the printer. On the other hand, if jobs are of the same size then delete on a first come first serve basis (FIFO).

```
    public static void main(String[] args) {
       ArrPriorityQueue<Integer> Q1=new ArrPriorityQueue<Integer>(10);
        Q1.Enqueue(21); // 21 kb file size
        Q1.Enqueue(54);
        System.out.println(Q1.Dequeue());
        System.out.println(Q1.Dequeue());
        System.out.println(Q1.Dequeue());

    //=================

    ListPriorityQueue<Integer> Q2=new ListPriorityQueue<Integer>(10);
        Q2.Enqueue(44);
        Q2.Enqueue(34);
        System.out.println(Q2.Dequeue());
        System.out.println(Q2.Dequeue());
        System.out.println(Q2.Dequeue());
    }
```