Queue:-

The queue is a linear data structure in which data will be inserted at one end, called “Rear” and delete from the other end called “Front”.

The queue data structure is work on First-In-First-Out(FIFO) principle, which means the data inserted first into the queue, that data will be removed first.

Example:- The ticket queue outside a cinema hall.

When a data item is inserted into the queue, the rear pointer will increase by one unit and when a data item is removed from the queue, the front pointer will increase by one unit.

The common operation performs on queue:-

1. Insert data into a queue.
2. Deleting data into a queue.
3. Display all data present in a queue.

Types of Queue:-

There are four types of the queue.

1. Simple Queue or Linear Queue.
2. Circular Queue.
3. Dequeue or Double ended queue.
4. Priority Queue.
    A. Ascending Priority Queue.
    B. Descending Priority Queue.

Simple Queue or Linear Queue:-In this queue we inserted the record from one end called rear and deleted the data from another end called front. It is strictly follow the FIFO rule.

The major drawback of this queue is once we deleted the data from the queue, we can’t insert more element in available space.

Circular Queue:-It is similar to linear queue except that last element or rear element is connected to the first element or front element.

Circular queue solves the problem of simple queue, when we delete the element from circular queue, deleted place we can insert the new record.

The main advantage of circular queue better memory utilization.

Dequeue or Double ended queue:- In Dequeue or Double ended queue insertion and deletion can be perform on both side front side and rear side.

There are two types of Dequeue or Double ended queue:-

1. Input restricted queue.
2. Output restricted queue.

Input restricted queue:-In this queue we perform insertion operation only rear side but deletion operation on the front and rear sides both.

Output restricted queue:- -In this queue we perform deletion operation only on the rear side but insertion operation on the front and rear sides both.

Priority Queue:-In this queue all elements or data will be store based on some priority.

There are two types of Priority queue:-

1. Ascending Priority Queue.
2. Descending Priority Queue.

Ascending Priority Queue:-In this queue all elements or data will be stored in ascending order.

Ascending Priority Queue:- In this queue all elements or data will be stored in descending order.

There are two ways to implement the Queue:-

1. implementation of Queue using arrays.
2. implementation of Queue using linked list.

Implementation of Queue using arrays:-

```
package com.ds;
public class NormalQueueDemo
{
int front,rear,size,queue[];
public NormalQueueDemo()
{

front = -1;
rear = -1;
size = 5;
queue = new int[size];
}
void insert(int value)
{
```

```
if(rear==size)
{
System.out.println("NQ is full");
return;
}
if(front==rear)
{
front=rear=0;
}
queue[rear++]=value;
}
void delete(){
if(front==rear){
System.out.println("NQ is empty");
return;
}
System.out.println("Deleted object is: "+queue[front]);
front++;
if(front==rear)
{
front=rear=-1;
}
}
void display(){
if(front==rear)
{
System.out.println("Normal Queue is empty");
return;
}
for(int i=front;i<rear;i++)
{
System.out.print(queue[i]+" ");
}
System.out.println();
}
public static void main(String[] args)
{
NormalQueueDemo nqd = new NormalQueueDemo();
nqd.insert(111);
nqd.insert(222);
nqd.insert(333);
nqd.insert(444);
nqd.insert(555);
nqd.display();
nqd.delete();
nqd.display();
}
}
```

Result:-

```
111 222 333 444 555
Deleted object is: 111
222 333 444 555
```

implementation of Queue using linked list:-

```
package com.ds;
public class NormalQueueList
{
Node front,rear;
int size;
class Node
{
int data;
Node next;
Node(int data,Node next)
{
this.data = data;
this.next = next;
}
}
public NormalQueueList()
{
front = null;
rear = null;
size = 0;
}
void display()
{
if(size==0)
{
System.out.println("q is empty");
return;
}
Node temp = front;
while(temp!=null)
{
System.out.print(temp.data+" ");
temp = temp.next;
}
System.out.println();
}
void insert(int value)
{
Node newNode = new Node(value,null);
if(front==null && rear==null)
{
```

```
front = newNode;
rear = newNode;
}
else
{
rear.next = newNode;
rear = newNode;
}
size++;
}
void delete()
{
if(size==0)
{
System.out.println("Queue is empty");
return;
}
System.out.println("Deleted item is: "+front.data);
front = front.next;
size--;
}
public static void main(String[] args)
{
NormalQueueList nql = new NormalQueueList();
nql.display();//q is empty
nql.insert(111);
nql.insert(222);
nql.insert(333);
nql.display();//111, 222, 333
nql.delete();//111
nql.display();//222, 333
nql.delete();//222
nql.display();//333
nql.delete();
nql.display();//q is empty
}
}
```

Result:-

```
Queue is empty
111 222 333
Deleted item is: 111
222 333
Deleted item is: 222
333
Deleted item is: 333
Queue is empty
```

implementation of Circular Queue using arrays:-

```
package com.ds;
public class CircularQueueArray
{int front,rear,size,count,circularQueue[];
CircularQueueArray()
{
front = -1;
rear = -1;
count = 0;
size = 5;
circularQueue = new int[size];
}
void insert(int value)
{
if(count==size)
{
System.out.println("Queue is full");
return;
}
if(front==-1)
front=rear=0;
else
rear = (rear+1)%size;
circularQueue[rear] = value;
count++;
}
void delete(){
if(count==0){
System.out.println("queue is empty");
return;
}
System.out.println("Deleted item is: "+circularQueue[front]);
if(front==rear)
front=rear=-1;
else
front = (front+1)%size;
count--;
}
void display(){
if(count==0){
System.out.println("queue is empty");
return;
}
int i=front;
if(front<=rear){
while(i<=rear)
System.out.print(circularQueue[i++]+" ");
}
else{
```

```
while(i!=rear){
System.out.print(circularQueue[i]+" ");
i=(i+1)%size;
}
System.out.print(circularQueue[i]);
}
System.out.println();
}
public static void main(String[] args)
{
CircularQueueArray cqa = new CircularQueueArray();
cqa.insert(111);
cqa.insert(222);
cqa.insert(333);
cqa.insert(444);
cqa.insert(555);
cqa.display();
cqa.insert(666);
cqa.delete();//111
cqa.display();
cqa.insert(666);
cqa.display();
}
}
```

Result:-

```
111 222 333 444 555
Queue is full
Deleted item is: 111
222 333 444 555
222 333 444 555 666
```

Implementation of Circular Queue using linked list:-

```
package com.ds;
public class CircularQueueList
{
Node front,rear;
class Node
{
int data;
Node next;
Node(int data,Node next)
{
this.data = data;
this.next = next;
}
}
void insert(int data)
{
```

```
Node newNode = new Node(data,null);
if(front==null)
{
front = newNode;
}
else
{
rear.next = newNode;
}
rear=newNode;
rear.next=front;
}
void delete(){
if(front==null){
System.out.println("queue is empty");
return;
}
System.out.println("Deleted item is: "+front.data);
if(front==rear){
front=null;
rear=null;
}
else{
front = front.next;
rear.next = front;
}
}
void display(){
Node temp = front;
if(temp==null){
System.out.println("queue is empty");
return;
}
while(temp.next!=front){
System.out.print(temp.data+" ");
temp=temp.next;
}
System.out.println(temp.data);
}
public static void main(String[] args)
{
CircularQueueList cql = new CircularQueueList();
cql.display();
cql.insert(111);
cql.insert(222);
cql.insert(333);
cql.insert(444);
cql.insert(555);
cql.display();
```

```
cql.delete();
cql.delete();
cql.display();
}
}
```

Result:-

```
queue is empty
111 222 333 444 555
Deleted item is: 111
Deleted item is: 222
333 444 555
```

Implementation of deque using arrays.

```
package com.ds;
public class DequeArray
{
int deque[],front,rear,size;
public DequeArray()
{
front = -1;
rear = -1;
size = 5;
deque = new int[size];
}
void insertAtFront(int value){
if((front==0 && rear==size-1)||(front==rear+1)){
System.out.println("Q is full");
return;
}
if(front==-1)
{
front=rear=0;
}
else if(front==0)
{
front=size-1;
}
else
{
front=front-1;
}
deque[front]=value;
}
void insertAtRear(int value){
if((front==0 && rear==size-1)||(front==rear+1)){
System.out.println("Q is full");
return;
```

```
}
if(front==-1)
{
front=rear=0;
}
else if(rear==size-1)
{
rear=0;
}
else
{
rear=rear+1;
}
deque[rear]=value;
}
void deleteFromFront(){
if(front==-1)
{
System.out.println("DQ is empty");
return;
}
System.out.println("deleted item is: "+deque[front]);
if(front==rear)
{
front=rear=-1;
}
else
{
if(front==size-1)
{
front=0;
}
else
{
front=front+1;
}
}
}
void deleteFromRear(){
if(front==-1){
System.out.println("dq is empty");
return;
}
System.out.println("Deleted object : "+deque[rear]);
if(front==rear)
{
front=rear=-1;
}
else
```

```
{
if(rear==0)
{
rear=size-1;
}
else
{
rear=rear-1;
}
}
}
void display(){
if(front==-1){
System.out.println("dq is empty");
return;
}
int left=front,right=rear;
if(left<=right){
while(left<=right)
{
System.out.print(deque[left++]+" ");
}
}
else{
while(left<=size-1)
System.out.print(deque[left++]+" ");
left=0;
while(left<=right)
System.out.print(deque[left++]+" ");
}
System.out.println();
}
public static void main(String[] args)
{
DequeArray dq = new DequeArray();
dq.insertAtRear(111);
dq.insertAtRear(222);
dq.insertAtRear(333);
dq.display();//111 222 333
dq.insertAtFront(777);
dq.insertAtFront(888);
dq.insertAtFront(999);//queue is full
dq.display();//888 777 111 222 333
dq.deleteFromFront();//888
dq.display();//777 111 222 333
dq.deleteFromRear();//333
dq.display();//777 111 222
}
}
```

Result:-

```
888 777 111 222 333
deleted item is: 888
777 111 222 333
Deleted object : 333
777 111 222
```

Implementation of deque using linked list:-

```
package com.ds;
public class DequeList
{
Node front,rear;
int size;//number of element in deque
DequeList(){
front = null;
rear = null;
size = 0;
}
class Node{
int data;
Node next;
Node(int data,Node next){
this.data = data;
this.next = next;
size++;
}
}
void insertAtFront(int value){
Node newNode = new Node(value,null);
if(front==null){
front = newNode;
rear = newNode;
return;
}
newNode.next = front;
front = newNode;
}
void insertAtRear(int value){
Node newNode = new Node(value,null);
if(front==null){
front = newNode;
rear = newNode;
return;
}
rear.next = newNode;
rear = newNode;
}
void deleteAtFront(){
if(front==null){
```

```
System.out.println("dq is empty");
return;
}
System.out.println("deleted obj is: "+front.data);
front = front.next;
size--;
}
void deleteAtRear(){
if(front==null){
System.out.println("dq is empty");
return;
}
System.out.println("deleted obj is: "+rear.data);
size--;
if(front==rear){
front=null;
rear=null;
return;
}
Node temp = front;
while(temp.next!=rear)
temp = temp.next;
rear = temp;
rear.next=null;
}
void display(){
if(front==null){
System.out.println("dq is empty");
return;
}
Node temp = front;
while(temp!=null){
System.out.print(temp.data+" ");
temp=temp.next;
}
System.out.println();
}
public static void main(String[] args)
{
DequeList dq = new DequeList();
dq.insertAtFront(333);
dq.insertAtFront(222);
dq.insertAtFront(111);
dq.display();//111 222 333
dq.insertAtRear(444);
dq.insertAtRear(555);
dq.insertAtFront(999);
dq.display();//999 111 222 333 444 555
dq.deleteAtFront();
```

```
dq.display();//111 222 333 444 555
dq.deleteAtRear();
dq.display();//111 222 333 444
}
}
```

Result:-

```
999 111 222 333 444 555
deleted obj is: 999
111 222 333 444 555
deleted obj is: 555
111 222 333 444
```