LJ University
University with a Difference

DATA STRUCTURE USING JAVA

LECTURE NOTES FOR UNIT - 4

UNIT -4

QUEUE - I

PREPARED BY:
MR. VISMAY SHAH

Table of Contents

1. Introduction To Queue

- ✓ A Queue is defined as a linear data structure that is open at both ends and the operations are performed in **First In First Out (FIFO) order.**
- ✓ Queue is a linear data structure where the first element is inserted from one end called **REAR** and deleted from the other end called as **FRONT**.
- ✓ Front points to the beginning of the queue and Rear points to the end of the queue.
- ✓ According to its FIFO structure, element inserted first will also be removed first.
- ✓ In a queue, one end is always used to **insert data (enqueue)** and the other is used to delete data (dequeue), because queue is open at both its ends.
- ✓ The enqueue() and dequeue() are two important functions used in a queue.

2. Basic Operations

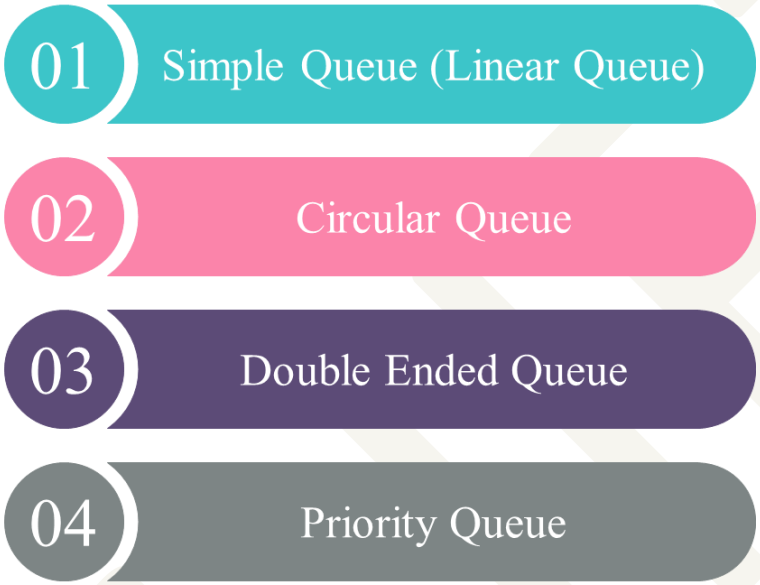
- ✓ Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory.
- ✓ Here we shall try to understand the basic operations associated with queues :

Operations	Description
enqueue ()	This function defines the operation for adding an element into queue.
dequeue ()	This function defines the operation for removing an element from queue.
init ()	This function is used for initializing the queue.
Front	Front is used to get the front data item from a queue.
Rear	Rear is used to get the last item from a queue.

3. FIFO Principle

- ✓ A Queue is like a line waiting to purchase tickets, where the first person in line is the first person served. (i.e. First come first serve).
- ✓ Position of the entry in a queue ready to be served, that is, the first entry that will be removed from the queue, is called the front of the queue, similarly, the position of the last entry in the queue, that is, the one most recently added, is called the rear of the queue.

4. Types of Queue



5. Simple Queue (Linear Queue)

- ✓ Array is the easiest way to implement a queue. Queue can be also implemented using Linked List or Stack.
- ✓ In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule.

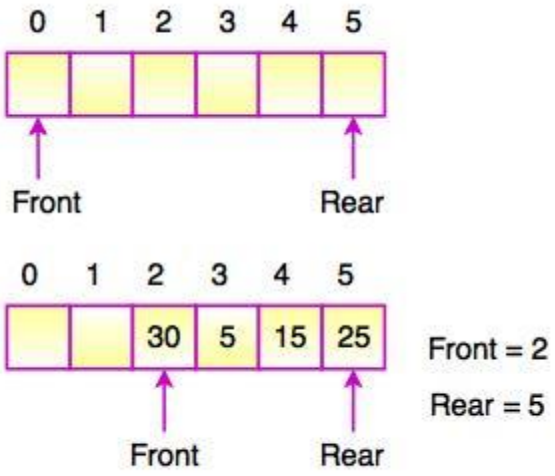


Fig. Implementation of Queue using Array

- ✓ In the above diagram, Front and Rear of the queue point at the first index of the array. (Array index starts from 0).
- ✓ While adding an element into the queue, the Rear keeps on moving ahead and always points to the position where the next element will be inserted. Front remains at the first index.

Example: Program to Implement a Queue using Array

In queue, insertion and deletion happen at the opposite ends, so implementation is not as simple as stack.

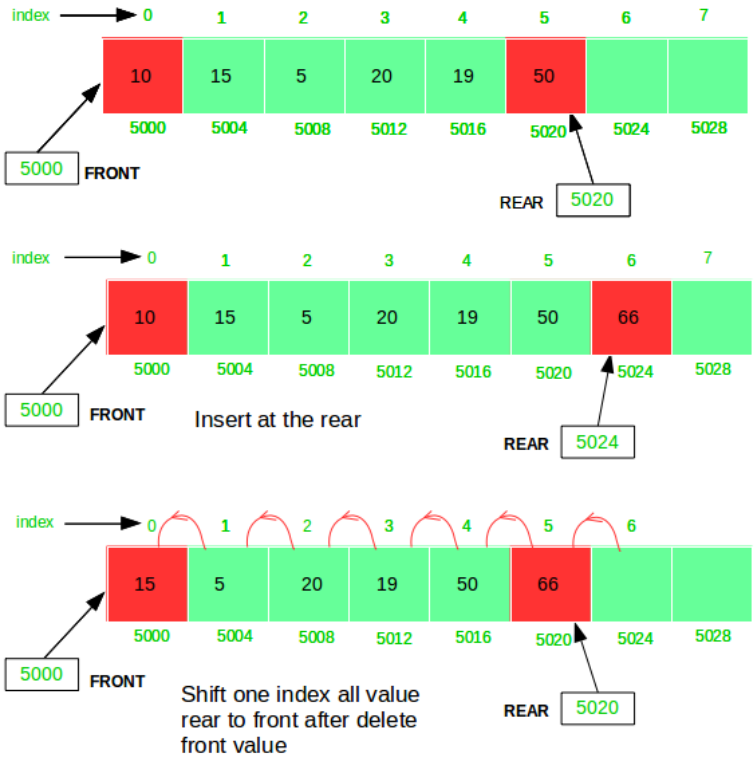
To implement a queue using array, create an array **arr** of **size n** and take two variables **front** and **rear** both of which will be initialized to 0 which means the queue is currently empty. Element rear is the index up to which the elements are stored in the array and front is the index of the first element of the array. Now, some of the implementation of queue operations are as follows:

Enqueue: Addition of an element to the queue. Adding an element will be performed after checking whether the queue is full or not. If **rear < n** which indicates that the array is not full then store the element at **arr[rear]** and increment rear by **1** but if **rear == n** then it is said to be an **Overflow Condition** as the array is full.

Dequeue: Removal of an element from the queue. An element can only be deleted when there is at least an element to delete i.e. **rear > 0**. Now, element at **arr[front]** can be deleted but all the remaining elements have to shifted to the left by one position in order for the dequeue operation to delete the second element from the left on another dequeue operation.

Front: Get the front element from the queue i.e. **arr[front]** if queue is not empty.

Display: Print all element of the queue. If the queue is non-empty, traverse and print all the elements from index front to rear.



6. Algorithm To Insert An Element in a Simple Queue

Procedure QINSERT (F, R, Q, N, Y)

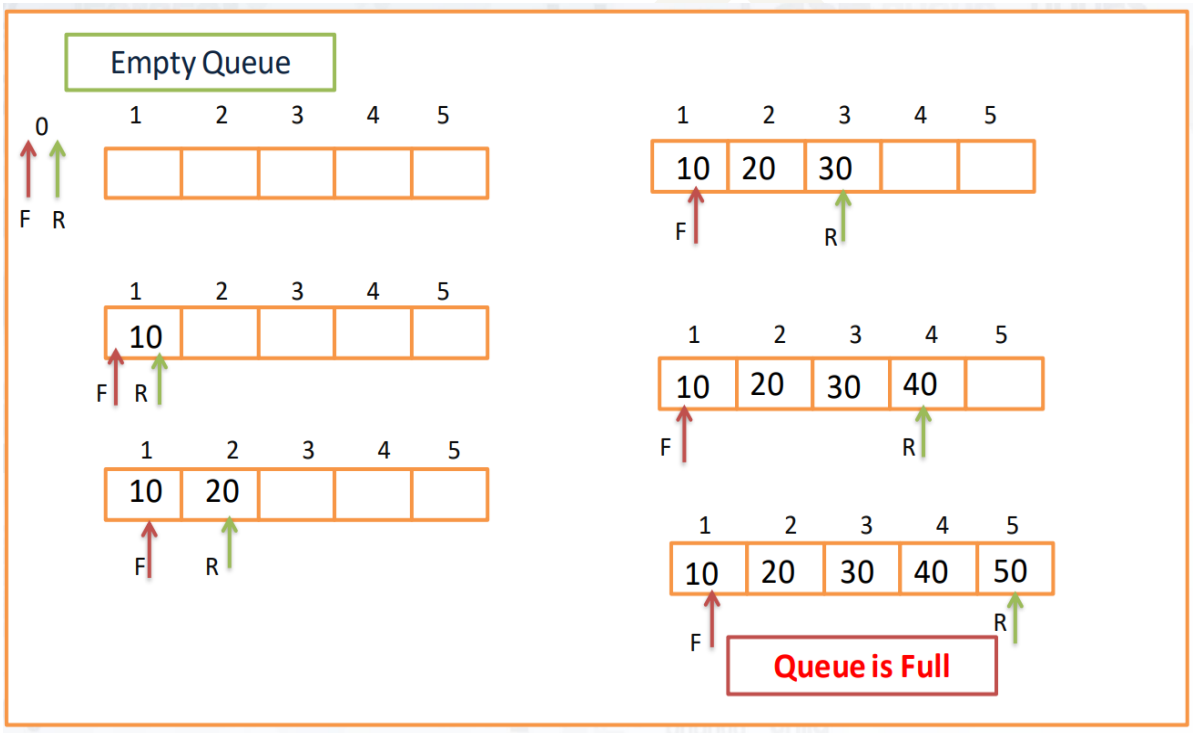
In this procedure, the F and R pointers to the front and rear elements of a queue, a queue Q consisting of N elements, and an element Y, this procedure inserts Y at rear of the queue. Prior to first invocation of the procedure, F and R have been set to zero.

Step 1: [Overflow ?]
If $R \geq N$
then Write('Overflow')
Return
Step 2: [Increment Rear Pointer]
$R \leftarrow R + 1$
Step 3: [Insert Element]
$Q[R] \leftarrow Y$
Step 4: [Is front pointer properly set]
If $F = 0$
then $F \leftarrow 1$
Return

Short Description:

Here, this algorithm is for insert the element in the queue Q. As we know that in the queue insertion is done at the Point Rear and Deletion is done at the point Front. So that for the insertion of the new element in the queue, we have to check for the overflow condition. Here the Queue's size is N.

So, if the Rear pointer position becomes greater or equal to N then queue becomes overflow. Now, if the overflow condition is become false, then increment the Rear pointer by 1. And insert the Element Y at that incremented Position of queue. Thus, we can insert the new elements in the Queue.



7. Algorithm To Delete An Element in a Simple Queue

Function QDELETE (Q, F, R)

Given F and R, the pointers to the front and rear elements of a queue, respectively, and the queue Q to which they correspond, this function deletes and returns the last element of the queue. Y is a temporary variable.

Step 1: [Underflow ?]

If F = 0

then Write('Underflow')

Return(0) (Here 0 denotes that the Queue is empty)

Step 2: [Delete the element]

Y← Q[F]

Step 3: [Queue Empty?]

If F = R

then F ← R ← 0

else F ← F + 1

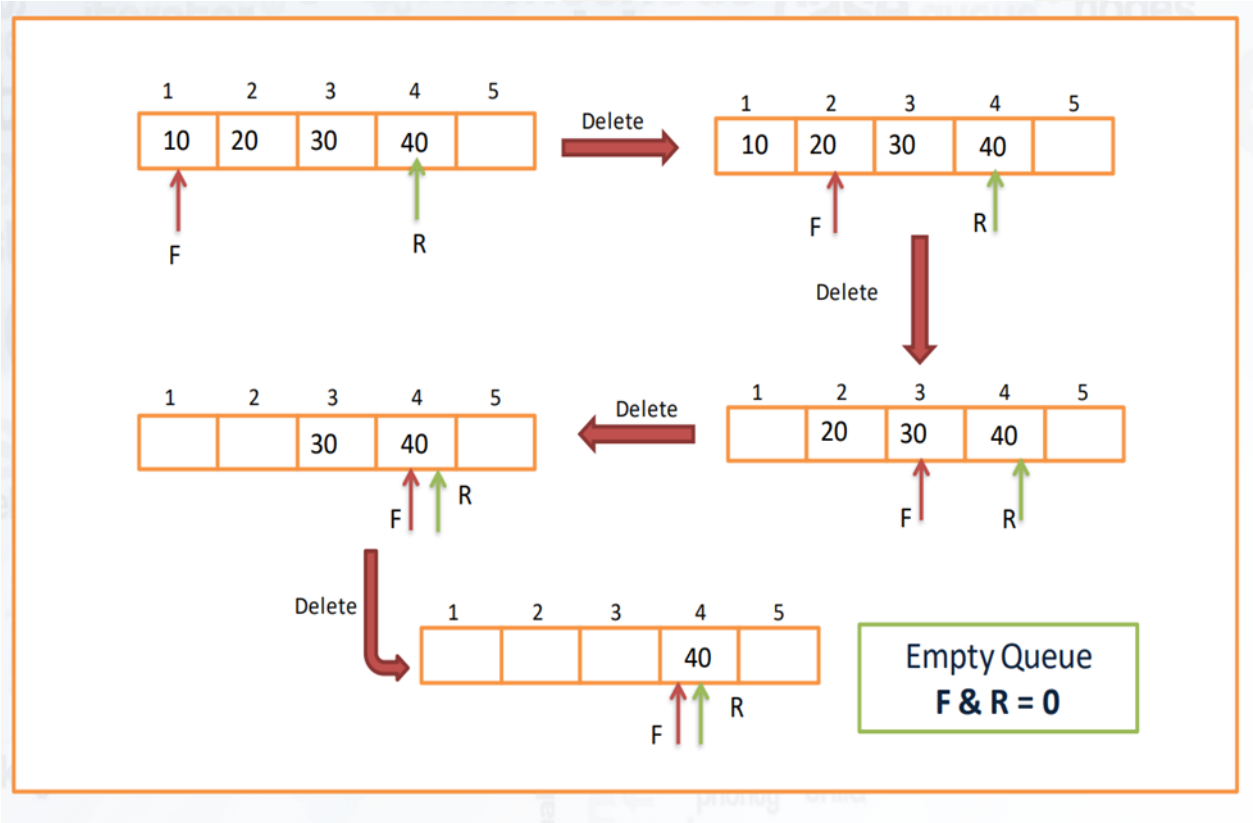
Step 4: [Return Element]

Return(Y)

Short Description:

Here, this algorithm is for delete the element in the queue Q. As we know that in the queue insertion is done at the Point Rear and Deletion is done at the point Front. So that for the deletion of the element from the queue, we have to check for the underflow condition. (Means whether the queue is empty or not?)

So, if the front pointer is at zero then queue becomes underflow. Now, if the underflow condition is become false, then just assign the Q[F] means the element on which F pointer is set, to the Y variable. Then check that whether the front and Rear are at the same position or not. If both are at same position then set Front and Rear at Zero. Because during deletion Front and Rear at the same position means the queue is empty. If Front and Rear are not at same position R then increment the Front pointer by 1. Thus, we can delete the element from the Queue.



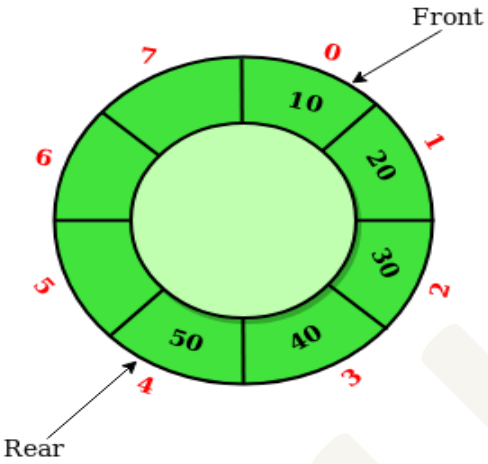
8. Write a Java Program to Implement The Basic Operation of Queue Using Switch Case.

```
1   //Write a Java program to implement QUEUE operations//
2   import java.util.*;
3   class Run
4   {
5       public static void main(String args[])
6       {
7           Scanner sc = new Scanner(System.in);
8           System.out.println("Enter the Size of Queue - "); //Here are we are asking the
            user to enter the queue size//
9           int size = sc.nextInt();
10          MyQueue q = new MyQueue(size); //created an object of class MyQueue & called
            parameterized constructor//
11          int choice = 0;
12          do{
13              System.out.println("Enter choice of your prefered operation - ");
14              choice = sc.nextInt();
15              switch(choice) //using a switch case we are asking user to enter the choice
                operation to be performed on queue//
16              {
17                  case 1: q.enqueue();
18                  break;
19
20                  case 2: q.dequeue();
21                  break;
22
23                  case 3: q.print();
24                  break;
25
26                  case 4: break;
27              }
28          }while(choice != 4); //until the user enters choice value 4 the loop will run
            continuously//
29      }
30  }
31  class MyQueue
32  {
33      Scanner sc = new Scanner(System.in);
34      int rear;
35      int front;
36      int size;
37      int a[];
38      MyQueue(int s) //created a parameterized constructor//
39      {
40          a = new int[s];
41          rear = -1;
42          front = -1;
43          size = s;
44      }
45      void enqueue() //created a method which will let us insert the elements//
46      {
47          if (rear == size - 1) //here we are checking the overflow condition//
48          {
49              System.out.println("Overflow");
50          }
51          else //if not overflow then//
52          {
53              if (front == - 1) //Are we adding an element for the 1st time? //
54              {
55                  front=0; //shifting front pointer to 0//
56              }
57              System.out.println("Enter no to be inserted - ");
58              int num = sc.nextInt();
59              rear = rear + 1; //increment rear pointer//
60              a[rear] = num;
61              System.out.println(num+ " is inserted successfully");
62          }
63      }
```

```
64        void dequeue() //created a method which will let us delete the element//
65        {
66            if(front == -1) //here we are checking the underflow condition//
67            {
68                System.out.println("Underflow");
69            }
70            else //if queue is not underflow then//
71            {
72                int num;
73                num = a[front]; //a temp variable will store the value where front is
                  pointing//
74                if (front == rear) // if queue has only one element, we need to reset it//
75                {
76                    front = -1;
77                    rear = -1;
78                }
79                else
80                {
81                    front = front + 1; //increment rear pointer//
82                }
83                System.out.println("Deleted element ->" + num);
84            }
85    }
86        void print() //created a method which will display the queue element//
87        {
88            if( front == -1) //checking if the queue is empty or not//
89            {
90                System.out.println("The queue is empty.");
91            }
92            else
93            {
94                for (int i = front; i<=rear; i++)
95                {
96                    System.out.println(a[i] + " - ");
97                }
98            }
99        }
100   }
```

9. Circular Queue

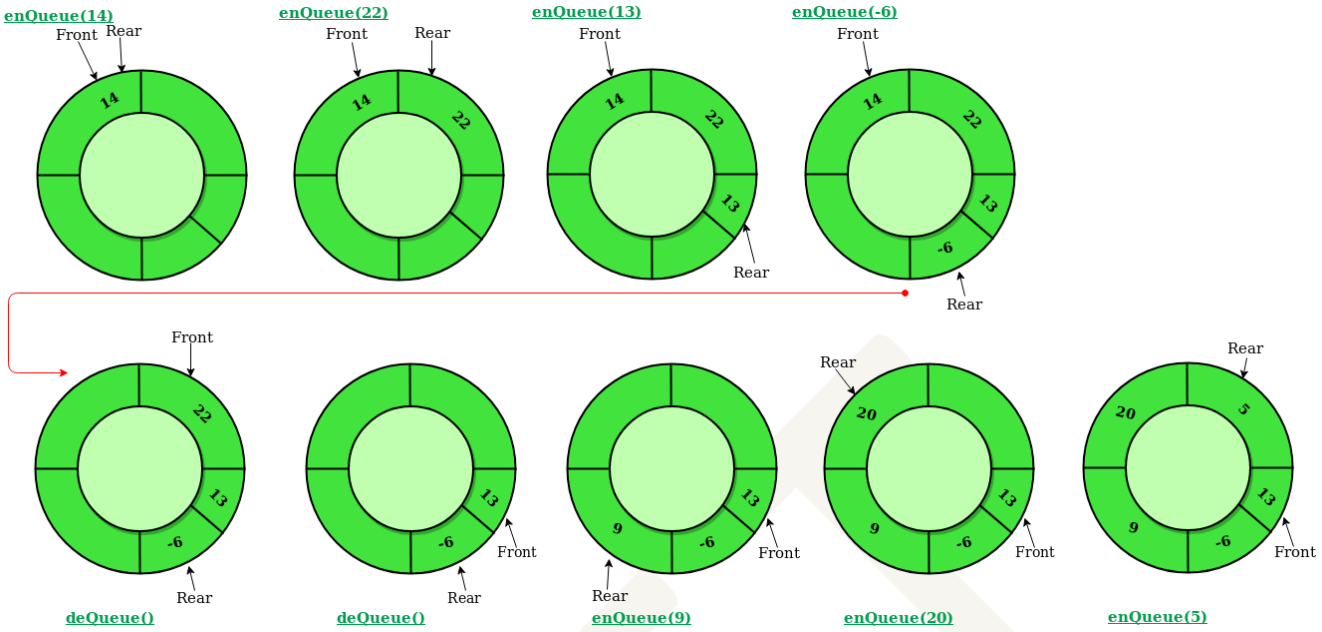
- ✓ Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. It is also called **'Ring Buffer'.**
- ✓ **Circular Queue is just a variation of the linear queue in which front and rear-end are connected to each other to optimize the space wastage of the Linear queue and make it efficient.**



Circular queue is primarily used in the following cases:

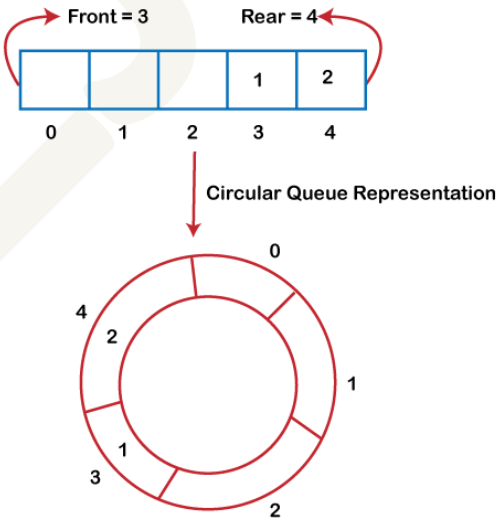
- ✓ **Memory Management:** The unused memory locations in the case of ordinary queues can be utilized in circular queues.
- ✓ **Traffic system:** In a computer-controlled traffic system, circular queues are used to switch on the traffic lights one by one repeatedly as per the time set.
- ✓ **CPU Scheduling:** Operating systems often maintain a queue of processes that are ready to execute or that are waiting for a particular event to occur.
- ✓ The **Time Complexity** for the Circular Queue is **O(1).**

- ✓ In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.



10. Basic Operations

Operations	Description
enQueue ()	This function defines the operation for adding an element into queue.
deQueue ()	This function defines the operation for removing an element from queue.
Front	Front is used to get the front data item from a queue.
Rear	Rear is used to get the last item from a queue.



11. Algorithm To Insert An Element In A Circular Queue

Method 1
Procedure CQINSERT (F, R, Q, N, Y)

Given pointers to the front and rear of a circular queue, F and R, a vector Q consisting of N elements, and an element Y, this procedure inserts Y at the rear of the queue, Initially, F and R are set to Zero.

Step 1: [Reset Rear Pointer]
If R = N
then R ← 1
else R ← R +1
Step 2: [Check for the Overflow ?]
If F = R
then Write(Queue is Overflow)
Return
Step 3: [Insert Element]
Q[R] ← Y
Step 4: [Is front Pointer Properly Set?]
If F = 0
then F ←1
Return

Method 2
Procedure CQINSERT (F, R, Q, N, Y)

Given pointers to the front and rear of a circular queue, F and R, a vector Q consisting of N elements, and an element Y, this procedure inserts Y at the rear of the queue, Initially, F and R are set to Zero.

Step 1: [Check for the Overflow Condition]
If (F=R+1) OR (F=1 AND R=N)
then Write(Queue is Overflow)
Return
Step 2: [Is front Pointer Properly Set?]
If F = 0
then F ←1
Return
Step 3: [Reset Rear Pointer]
If R = N
then R ← 1
else R ← R +1
Step 4: [Insert Element]
Q[R] ← Y

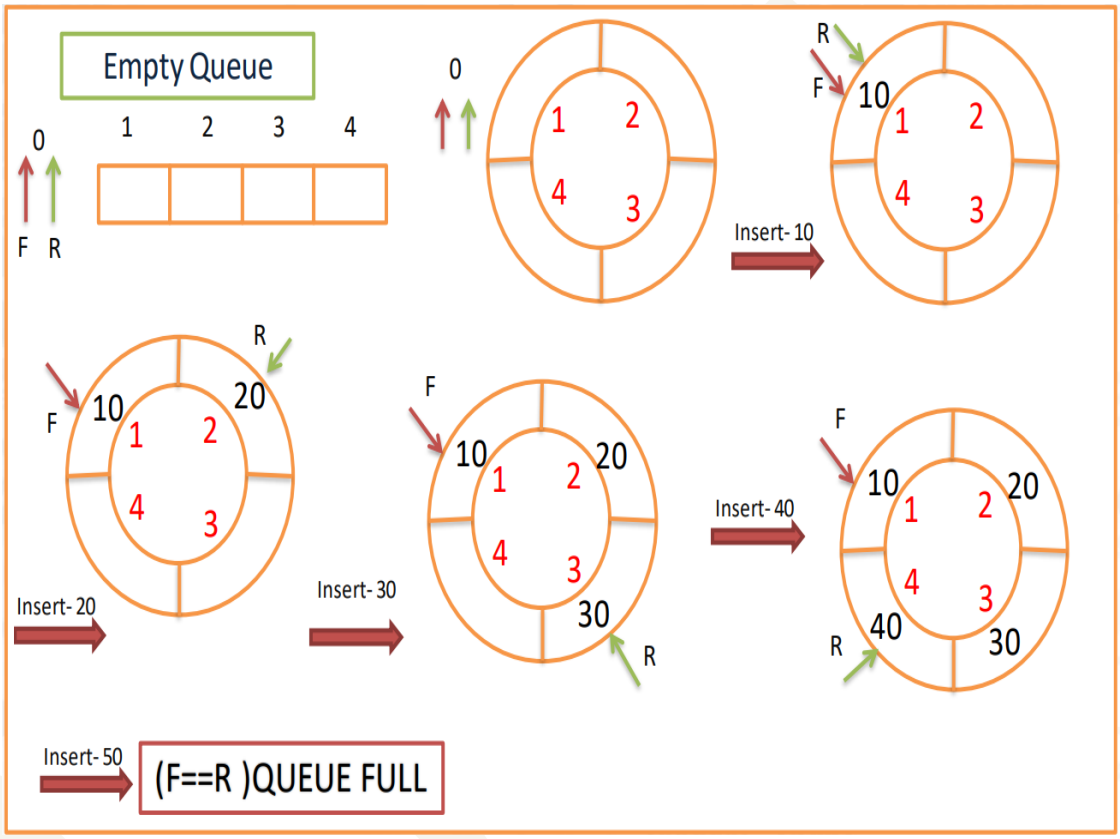
Short Description:
Here, this algorithm is for insert the element in the circular queue Q. As we know that in the queue insertion is done at the Point Rear and Deletion is done at the point Front. In circular queue if the Rear pointer comes at the N^{th} Position, then as the queue is circular, we have to set it at 1st position of the queue. Otherwise, we just need to increment the Rear Pointer by 1.

After the setting the Rear Pointer, we check for the Overflow condition. During the insertion operation in the circular queue, if the Front and Rear Pointer comes at the same position then it means that queue is become full.

So, if this condition become false then and then we can insert the new element Y at the Rear's Position. Q[R] = Y. Now check for the front pointer position, if the

front pointer is at zero then we set it at 1. Thus, the insertion is done in the circular queue.

The activities that place an excessive or repetitive load on the spine may increase the risks factors of Radiculopathy. This repetitive load can affect the herniated disc, degenerative disc and bone spurs.



12. Algorithm To Delete An Element In A Circular Queue

Procedure CQDELETE (F, R, Q, N)

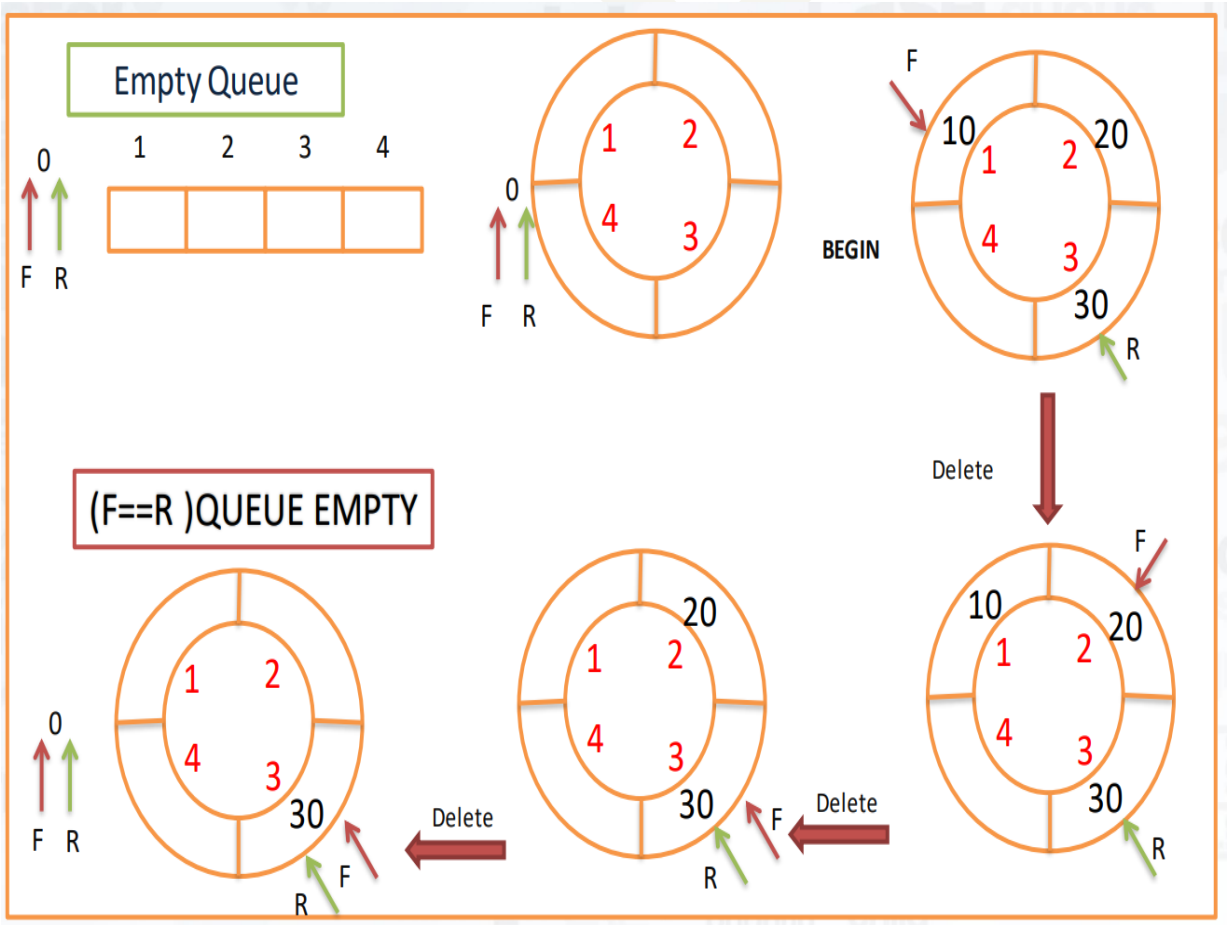
Step 1: [Underflow ?]
If F = 0
then Write('Underflow')
Return(0) (Here 0 denotes that the Queue is empty)
Step 2: [Delete the element]
Y← Q[F]
Step 3: [Queue Empty?]
If F = R
then F ← R ← 0
Return(Y)
Step 4: [Increment Front Pointer]
If F = N
then F ← 1
else F ← F + 1
Return(Y)

Short Description:

Here, this algorithm is for delete the element from the circular queue Q. As we know that in the queue insertion is done at the Point Rear and Deletion is done at the point Front. So that for the deletion of the element from the queue, we have to check for the underflow condition. (Means whether the queue is empty or not?)

So, if the front pointer is at zero then queue becomes underflow. Now, if the underflow condition is become false, then just assign the Q[F] means the element on which F pointer is set, to the Y variable. Then check that whether the front and Rear are at the same position or not. If both are at same position then set Front and Rear at Zero. Because during deletion Front and Rear at the same position means the queue is empty. Then Return Y.

Now if Front pointer is at position N., then set it to 1. Otherwise increment the front pointer by 1. Thus, we can delete the element from the circular queue.



13. Write a Java Program to Implement The Basic Operation of Circular Queue Using Switch Case.

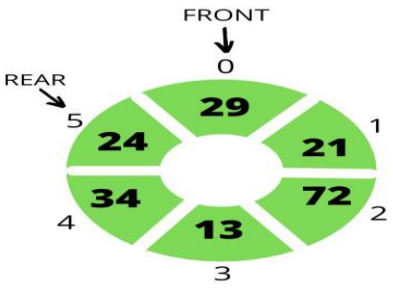
```
1   //Write a Java program to implement operations on CIRCULAR QUEUE//
2   import java.util.*;
3   class Run
4   {
5       public static void main(String args[])
6       {
7           Scanner sc = new Scanner(System.in);
8           System.out.println("Enter the Size of Queue - "); //Here are we are asking the
            user to enter the queue size//
9           int size = sc.nextInt();
10          MyQueue q = new MyQueue(size); //created an object of class MyQueue & called
            parameterized constructor//
11          int choice = 0;
12          do{
13              System.out.println("Enter choice of your prefered operation - ");
14              choice = sc.nextInt();
15              switch(choice) //using a switch case we are asking user to enter the choice
                operation to be performed on queue//
16              {
17                  case 1: q.enqueue();
18                  break;
19
20                  case 2: q.dequeue();
21                  break;
22
23                  case 3: q.print();
24                  break;
25
26                  case 4: break;
27              }
28          }while(choice != 4); //until the user enters choice value 4 the loop will run
            continuously//
29      }
30  }
31
32  class MyQueue
33  {
34      Scanner sc = new Scanner(System.in);
35      int rear;
36      int front;
37      int size;
38      int a[];
39      MyQueue(int s) //created a parameterized constructor//
40      {
41          a = new int[s];
42          rear = -1;
43          front = -1;
44          size = s;
45      }
46      void enqueue() //created a method which will let us insert the elements//
47      {
48          if ((front - 1== rear) || (front == 0 && rear == size - 1)) //here we are
            checking the overflow condition//
49          {
50              System.out.println("Overflow");
51          }
52          else //if queue is not overflow then//
53          {
54              if(front == -1) //Are we adding an element for the 1st time? //
55              {
56                  front=0; //shifting front pointer to 0//
57              }
58              System.out.println("Enter no to be inserted - ");
59              int num = sc.nextInt();
60              rear = (rear + 1) % size;
61              a[rear] = num;
62          }
```

```
63        }
64        void dequeue() //created a method which will let us delete the element//
65        {
66            if(front == -1) //here we are checking the underflow condition//
67            {
68                System.out.println("Empty");
69            }
70            else //if queue is not underflow then//
71            {
72                int num;
73                num = a[front];
74                if (front == rear) // if queue has only one element, we need to reset it//
75                {
76                    front = -1;
77                    rear = -1;
78                }
79                else
80                {
81                    front = (front + 1) % size;
82                }
83                System.out.println("Deleted element ->" + num);
84            }
85    }
86        void print() //created a method which will display the queue element//
87        {
88            int i;
89            if( front == -1) //checking if the queue is empty or not//
90            {
91                System.out.println("The queue is empty.");
92            }
93            else
94            {
95                for (i = front; i != rear; i = (i + 1) % size) //the loop will only run
                  untill it reaches rear//
96                {
97                    System.out.println(a[i] + " - "); //hence all values will be printed
                      except rear//
98                }
99                System.out.println(a[i]); //separate SOP to print last element (Rear)//
100           }
101       }
102   }
```

14. Why Circular Queue is Better Than Simple Queue?

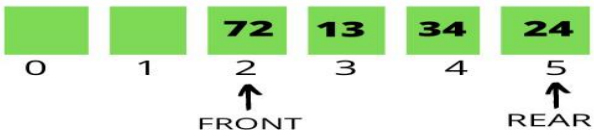
✓ When Enqueue operation is performed on both the queues: Let the queue is of size 6 having elements {29, 21, 72, 13, 34, 24}. In both the queues the front points at the first element 29 and the rear points at the last element 24 as illustrated below:

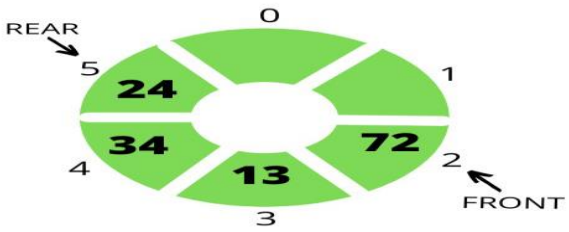
LINEAR QUEUE



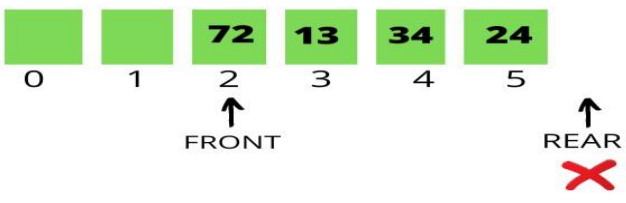
CIRCULAR QUEUE

✓ When the Dequeue operation is performed on both the queues: Consider the first 2 elements that are deleted from both the queues. In both the queues the front points at element 72 and the rear points at element 24 as illustrated below:

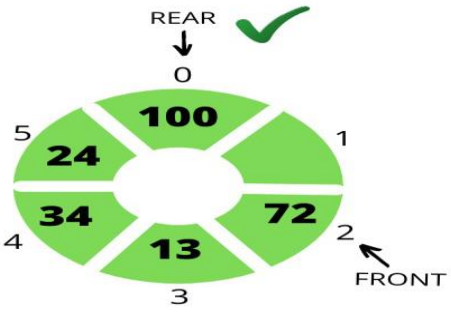




✓ Now again enqueue operation is performed: Consider an element with a value of 100 is inserted in both the queues. The insertion of element 100 is not possible in Linear Queue but in the Circular Queue, the element with a value of 100 is possible as illustrated below:



Insertion not possible !



Insertion possible !

Example

Show Position of F & R in Circular Queue of Size 6. Assume F at 2 and R at 4

QUEUE = _, L, M, N, _, _
Here Size = 6 and F at 2 R at 4
So Index Starts at 1

Operation	F	R	Queue
BEGIN	2	4	_, L, M, N, _, _
Add – O	2	5	_, L, M, N, O, _
Add – P	2	6	_, L, M, N, O, P
Delete	3	6	_, _, M, N, O, P
Delete	4	6	_, _, _, N, O, P
Add – Q, R	4	2	Q, R, _, N, O, P
Add – S	4	3	Q, R, S, N, O, P
Add – T	4	3	No INSERT Queue Full
Delete	5	3	Q, R, S, _, O, P
Delete	6	3	Q, R, S, _, _, P