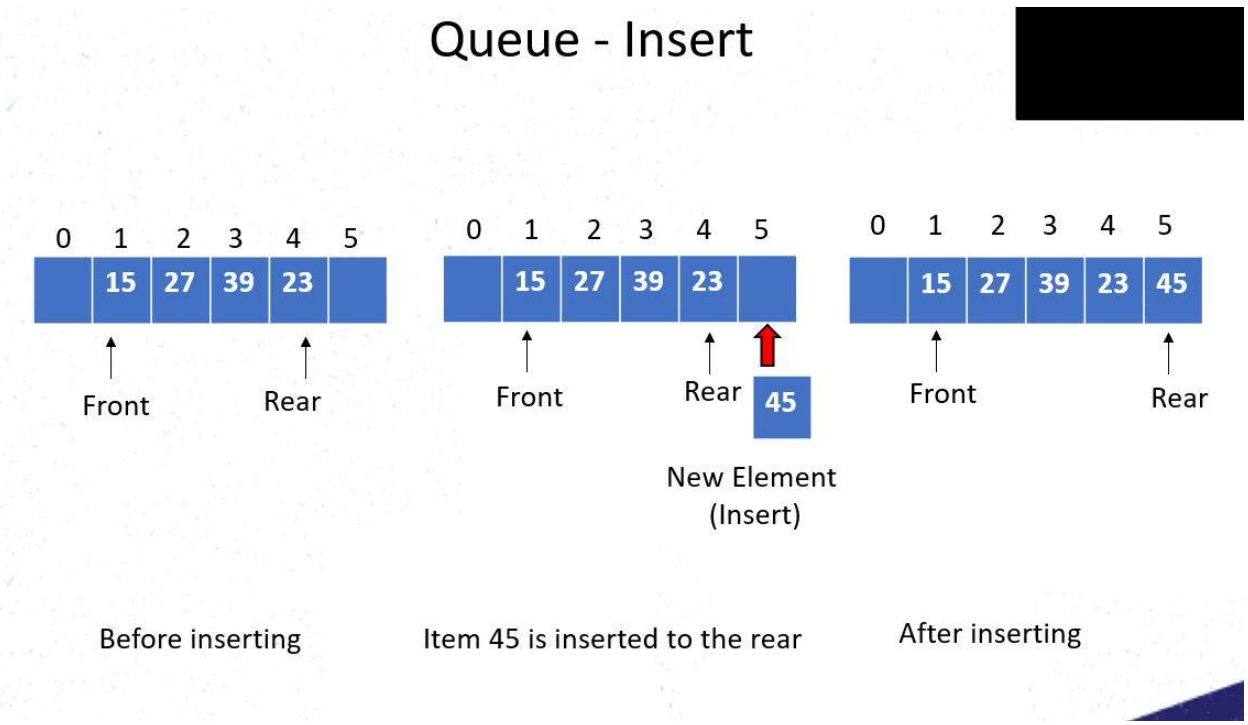
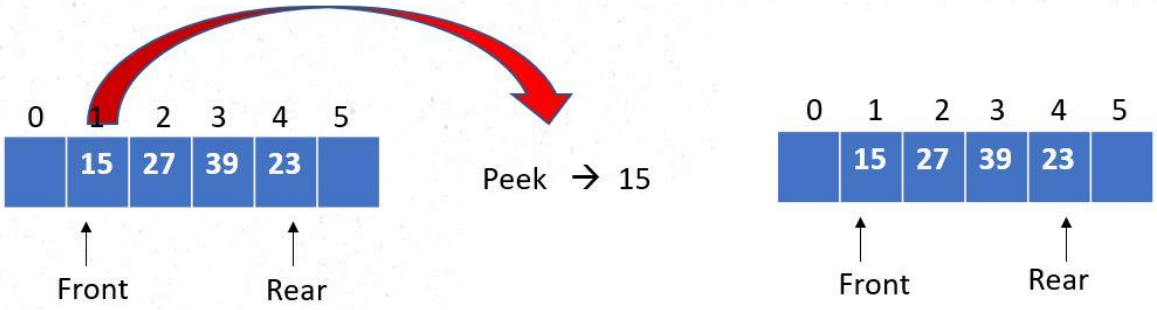
Queue

- In a queue all insertions are made at the Rear end and deletions are made at the Front end.
- Insertions and deletions are restricted from the Middle of the Queue.
    - Adding an item is called insert
    - Removing an item is called remove
    - The elements are inserted and removed according to the First-In-First-Out (FIFO) principle.



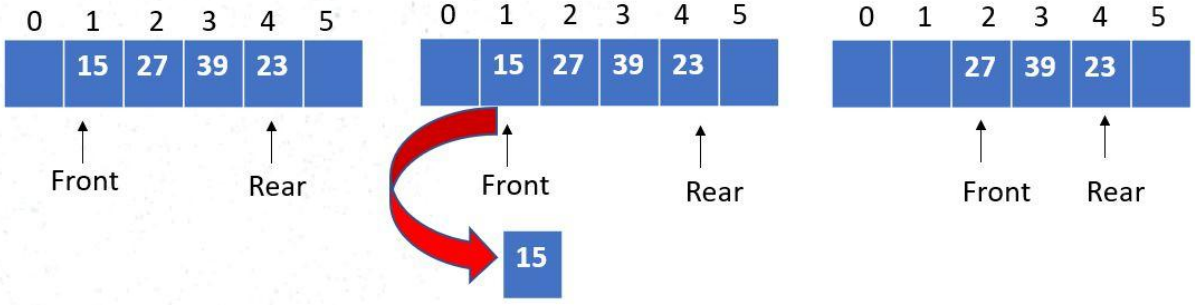
Queue - PeekFront



Queue remains the same

Peek is used to read the value from the Front of the queue without removing it. You can peek only the Front item, all the other items are invisible to the queue user.

Queue - Remove



Before removing

Front item is removed

After removing

Queue implementation

```
class QueueX {
    private int maxSize;    // size of queue array
    private int [] queArray;
    private int front;           //front of the queue
    private int rear;           //rear of the queue
    private int nItems;  //no of items of the queue

    public QueueX (int s) {// constructor

   maxSize = s;       // set array size
   queArray = new int [maxSize];
   front = 0;
   rear = -1;
   nItems = 0;             // no items
     }
```

The constructor initializes the max size of the queue.then creates a new array and assigns it to the array pointer.
Then the front variable is initialized to 0 and the rear is initialized to -1 because there are no variables at the start and no items is 0.

Inserting items

```
public void insert(int j)  {
//check whether the queue is full
  // increment rear
  // insert an item

 }
```

```
public void insert(int j) {
        // check whether queue is full
         if (rear == maxSize - 1)
               System.out.println("Queue is full");
        else {
                queArray[++rear] = j;
                nItems++;

        }
          }
    }
```

Removing an element

```
public int remove() {
        // check whether queue is empty
        // if not
        // access item and increment front
}
```

```
public int remove() {
        if (nItems == 0)  {
            System.out.println("Queue is empty");
            return -99;
        }
        else   {
            nItems--;
            return queArray[front++];
        }
}
```

Peeking an element

public int peekFront() {
  // check whether queue is empty
  // if not
  // access item
}

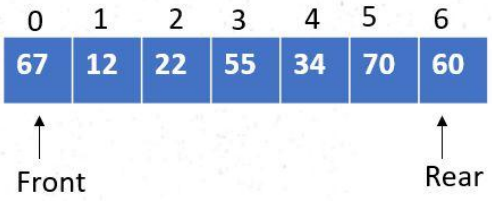
```
public int peekFront() {
    if (nItems == 0)  {
         System.out.println("Queue is empty");
         return -99;
     }
    else {
         return queArray[front];
    }
}
```

Checking whether the queue is empty or full

```
public boolean isEmpty() // true if queue is empty
{
return (nItems==0);
}
//--------------------------------------------------------------
public boolean isFull() // true if queue is full
{
return (nItems==maxSize);
}
```

Question 03

Draw the Queue frame after performing the below operations to the queue given below.



i) remove
ii) remove
iii) remove
iv) Insert item 88

1)

67	12	22	55	34	70	60

2)

	12	22	55	34	70	60

3)

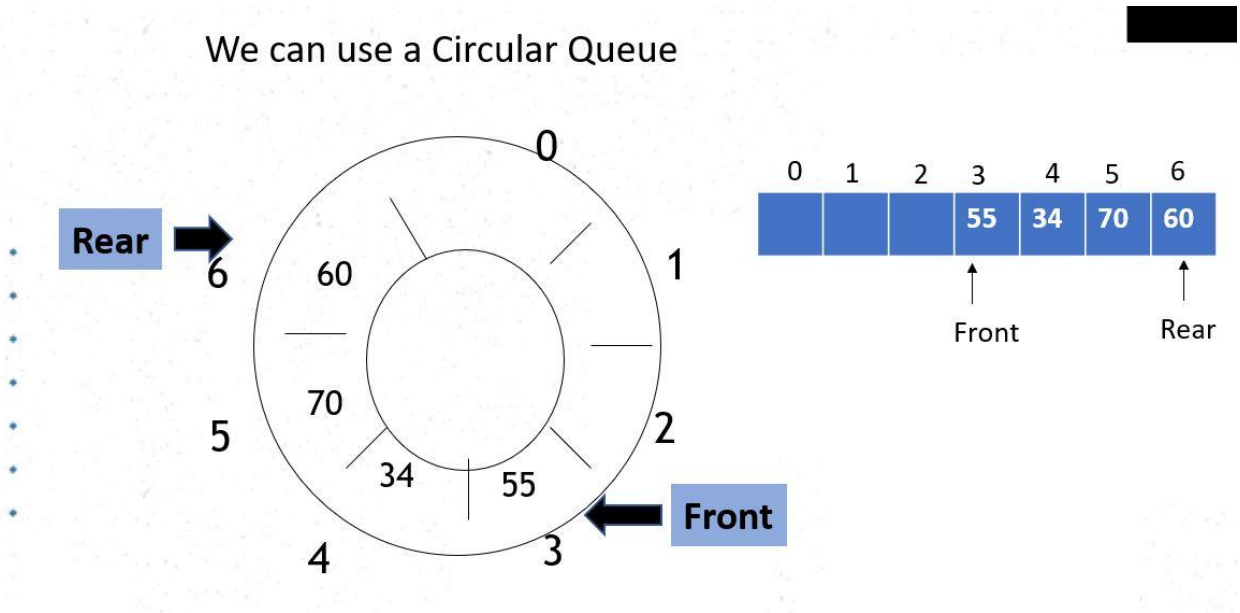
			55	34	70	60

4)
If the implementation was like if(rear ==maxSize -1)
Although there is space in the queue, since the elements are not shifting , the queue is shown as full

			55	34	70	60

To overcome this problem the circular queue can be used.

Circular queue



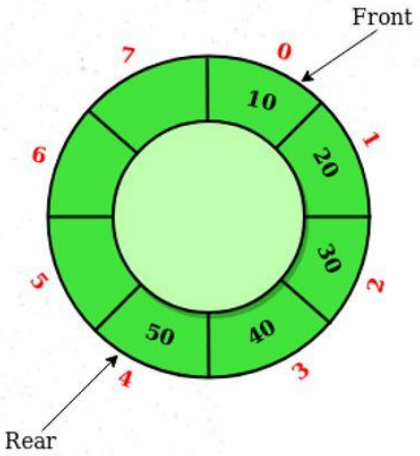
- Circular queues are queues that wrap around themselves.
- These are also called ring buffers.
- The problem in using the linear queue can be overcome by using circular queue.
- When we want to insert a new element we can insert it at the beginning of the queue.
  i.e. if the queue is not full we can make the rear start from the beginning by wrapping around

  If rear was 3 then the next element should be stored in index 4

  If rear was 7 then the next element should be stored in index 0

Question 04

Draw the Queue frame after performing the below operations to the circular queue given below.



i) insert(14);
ii) insert(29);
iii) insert(33);
iv) insert(88);
v) peekFront();
vi) remove();
vii) remove();
viii) insert(90);
ix) insert(100);
x) peekFront();

1)

10	20	30	40	50	14		
0	1	2	3	4	5	6	8

2)

10	20	30	40	50	14	29	
0	1	2	3	4	5	6	8

3)

10	20	30	40	50	14	29	33
0	1	2	3	4	5	6	8

4)

Array is full

10	20	30	40	50	14	29	33
0	1	2	3	4	5	6	8

5) 10

10	20	30	40	50	14	29	33
0	1	2	3	4	5	6	8

6)
Front = 1

	20	30	40	50	14	29	33
0	1	2	3	4	5	6	8

7)
Front = 2

		30	40	50	14	29	33
0	1	2	3	4	5	6	8

8)
Rear = 0

90		30	40	50	14	29	33
0	1	2	3	4	5	6	8

9)
Rear = 1
Front = 2

90	100	30	40	50	14	29	33
0	1	2	3	4	5	6	8

10)30
Front = 2

10	20	30	40	50	14	29	33
0	1	2	3	4	5	6	8

Implementation of the circular queue

Inserting elements

```
public void insert(int j)  {
    // check whether queue is full
    if (nItems == maxSize)
         System.out.println("Queue is full");
   else {
        if(rear == maxSize - 1)
            rear = -1;//like in the start initialization since there are
no elements

        queArray[++rear] = j;

        nItems++;
   }
}
```

Removing elements

```
public int remove() {
    // check whether queue is empty
    if ( nItems == 0)
         System.out.println("Queue is empty");
   else  {
        int temp = queArray[front++];
        if (front == maxSize)
            front  = 0;
        nItems--;
        return temp;

   }
}
```

Lets consider the following example

10	20						33
0	1	2	3	4	5	6	8

In this instance the rear is equal to 1 and the front is equal to 8
maxSize = 8 and no of items is 3
When remove is called

```
    // check whether queue is empty
    if ( nItems == 0)
         System.out.println("Queue is empty");

No of items != 0 therefore the else is executed
   else  {

        int temp = queArray[front++]; // temp = 33 and front = 8
        if (front == maxSize) // front = maxsize = 8
  front  = 0; // front = 0
        nItems--; //nItems = 2
        return temp; // return 33

   }
```

Question 05

Implement isFull(), isEmpty() and peekFront() methods of the Circular Queue class.

```
public long peekFront() // peek at front of queue
{
```

```
return queArray[front];
}
//--------------------------------------------------------------
public boolean isEmpty() // true if queue is empty
{
return (nItems==0);
}
//--------------------------------------------------------------
public boolean isFull() // true if queue is full
{

return (nItems==maxSize);
}
//--------------------------------------------------------------
```

Q6)

Using the implemented QueueX class, Write a program to create a queue with maximum size 10 and insert the following items to the queue.

10 25 55 65 85

Delete all the items from the queue and display the deleted items.

```
class QueueApp {
    public static void main(String[] args) {
        QueueX  theQueue = new QueueX(10);  // create a queue with max
size 10

         theQueue. insert(10);  // insert given items
         theQueue. insert(25);
         theQueue. insert(55);
         theQueue. insert(65);
         theQueue. insert(85);
```

```
        while( !theQueue.isEmpty() ) {    // until it is empty, delete
item from queue

            int val = theQueue.remove();
            System.out.print(val);
            System.out.print(" ");
        }
    }
} // end of class
```