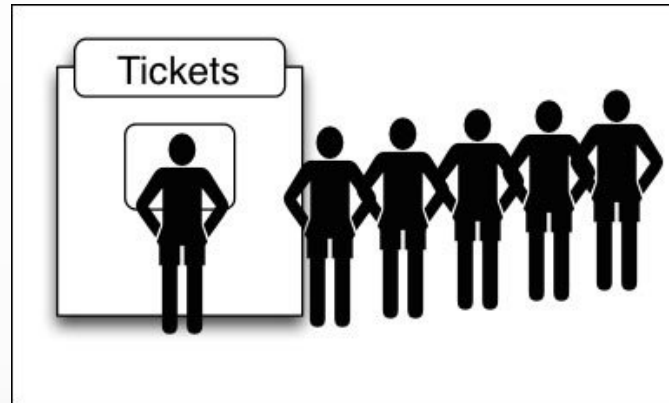

IT2070 – Data Structures and Algorithms

Lecture 02

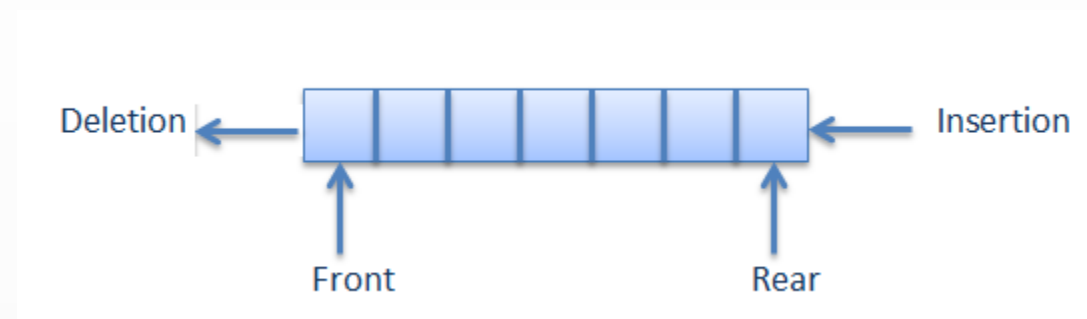
Introduction to Queue

Queues



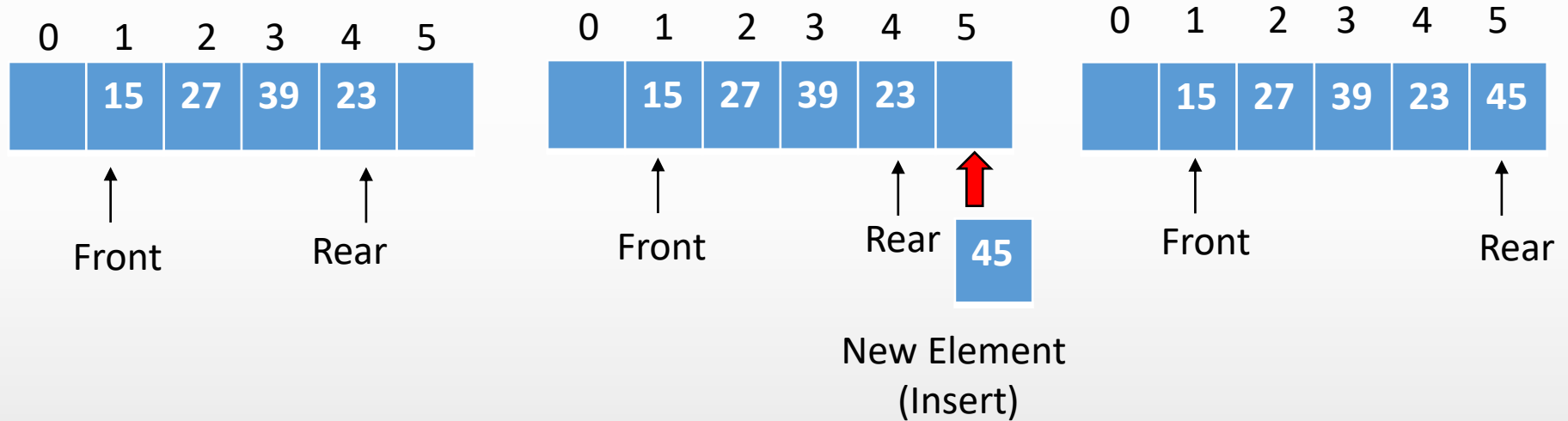
- Imagine a queue in real life
- The first item inserted is the first item to be removed

Queues



- 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 - Insert

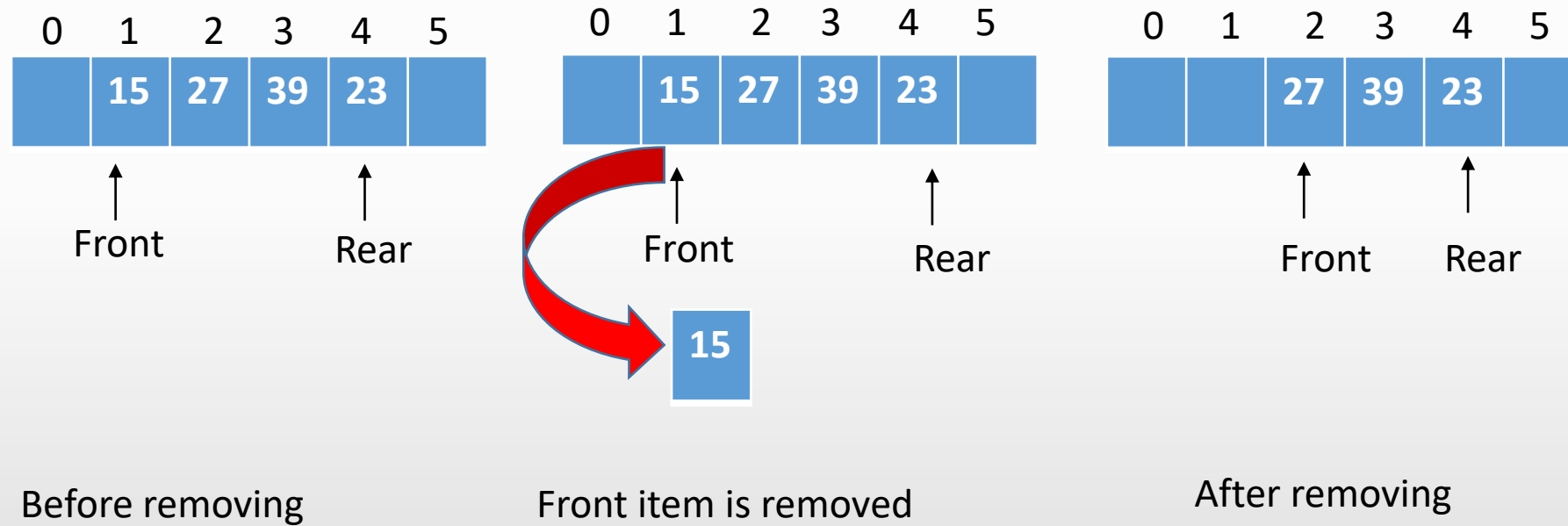


Before inserting

Item 45 is inserted to the rear

After inserting

Queue - Remove



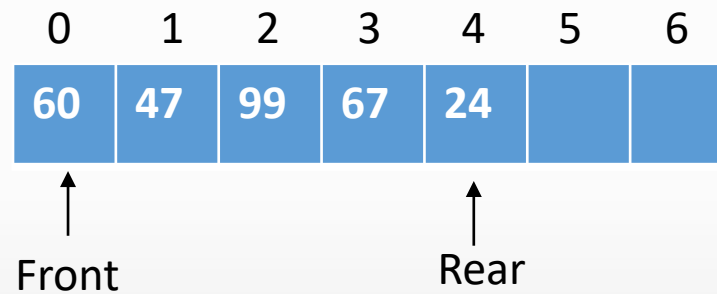
Queue - PeekFront



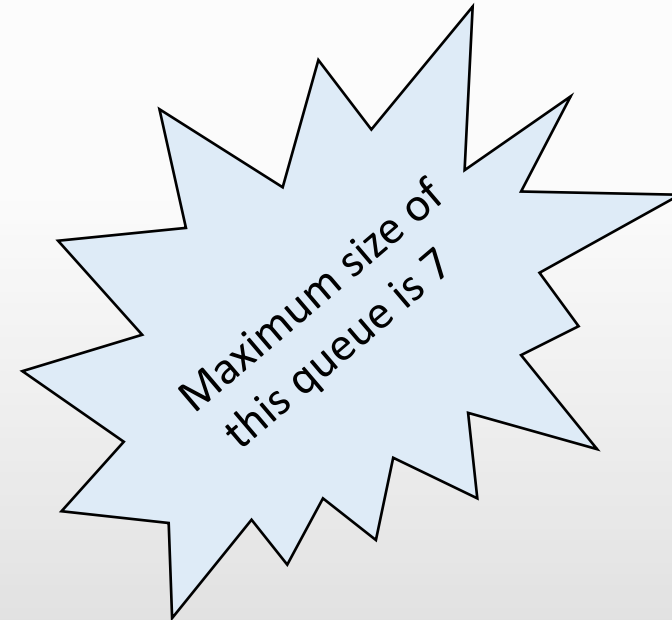
Peek is used to read the value from the Front of the queue without removing it. You can peek only the Front item, all the other items are invisible to the queue user.

Question 01

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



- i) Insert item 33
- ii) Insert item 53
- iii) peekFront
- iv) remove
- v) remove
- vi) remove
- vii) remove
- viii) remove



Uses of Queue

- There are various queues quietly doing their job in a computer's operating system.
 - printer queue
 - stores keystroke data as you type at the keyboard
 - pipeline

Queue - Implementation

Queue implementation using an **array** with restricted access

- Constructor creates a new Queue of a size specified in its argument.
- Variable **front**, which stores the index of the item on the front of the queue.
- Variable **rear**, which stores the index of the item on the end of the queue.
- Variable *nItems*, which stores the total number of the items in the queue.


```
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  
    }  
    .....  
    .....  
}
```

Queue – Implementation - insert

```
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  
    }  
    public void insert(int j) {  
        // increment rear  
        // insert an item  
    }  
}
```

Queue – Implementation - insert

```
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  
    }  
  
    public void insert(int j) {  
        // increment rear and insert an item  
        queArray[++rear] = j;  
        nItems++;  
    }  
}
```



What if the queue is full ?

Queue – Implementation - insert

```
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  
    }  
}
```

```
    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++;  
        }  
    }  
}
```

Queue – Implementation – remove/peekFront

```
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
    }

    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++;

        }
    }
}
```

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

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

Queue – Implementation – remove

```
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
    }

    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++;

        }

    }
}
```

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

Queue – Implementation – peekFront

```
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
    }

    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++;

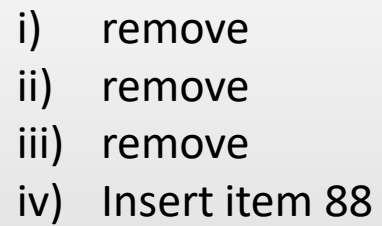
        }
    }
}
```

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

Question 02

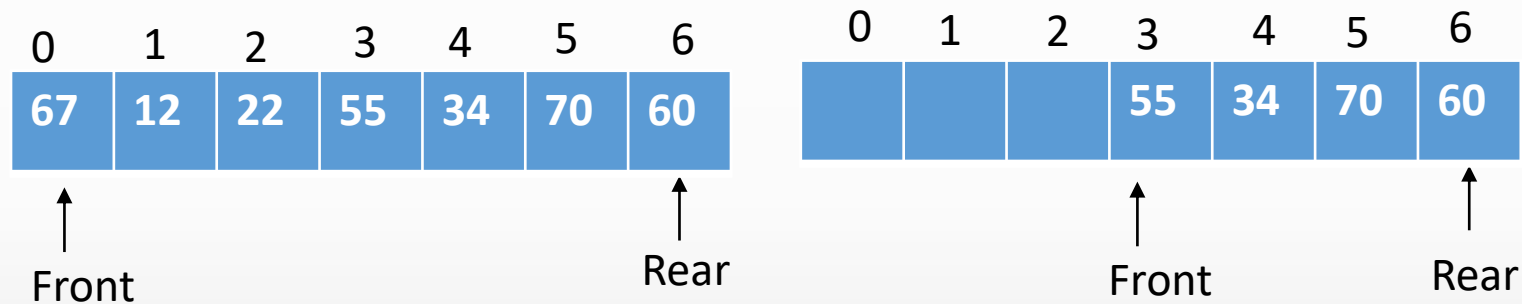
isEmpty() method of the Queue class returns true if the Queue is empty and isFull() method returns true if the Queue is full.

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



Question 03 Contd..

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



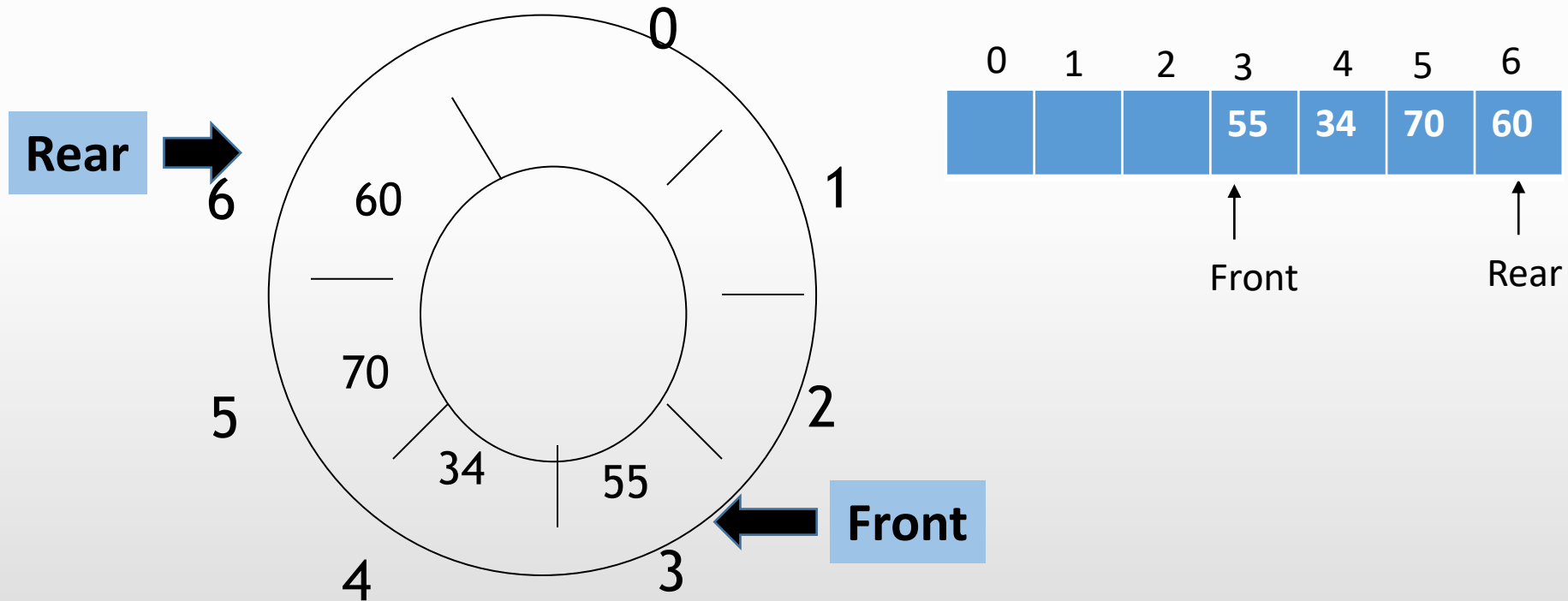
- i) remove
- ii) remove
- iii) remove
- iv) Insert item 88

Although the queue is not full we cannot insert more elements.

Any Suggestions?

How to overcome this situation??

We can use a Circular Queue



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.

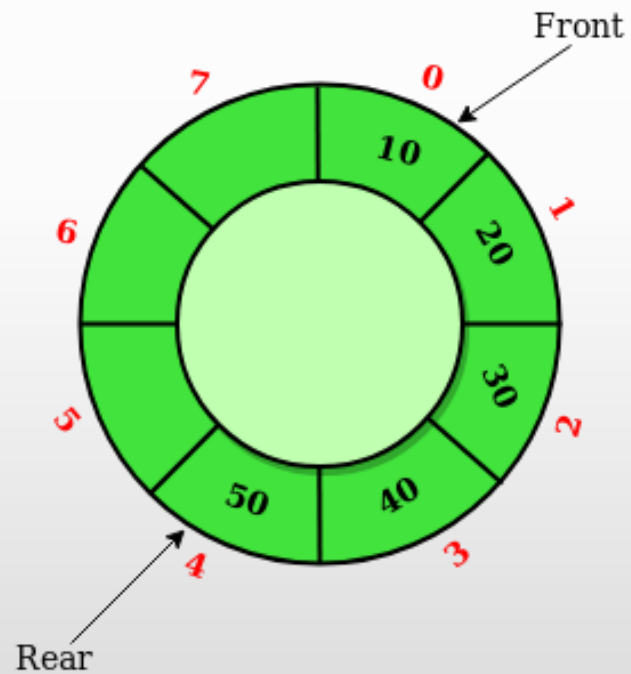
i.e. if the queue is not full we can make the rear start from the beginning by wrapping around

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();

Inserting an element to Linear Queue

```
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++;  
    }  
}
```

Inserting an element to Circular Queue

```
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;  
  
        queArray[++rear] = j;  
  
        nItems++;  
    }  
}
```

Removing an element from Linear Queue

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

Removing an element from Circular Queue

```
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;  
    }  
}
```

Question 05

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

Question 06

Creating a Queue

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.

Creating a Queue

```
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
```

References

1. Mitchell Waite, Robert Lafore, Data Structures and Algorithms in Java, 2nd Edition, Waite Group Press, 1998.

