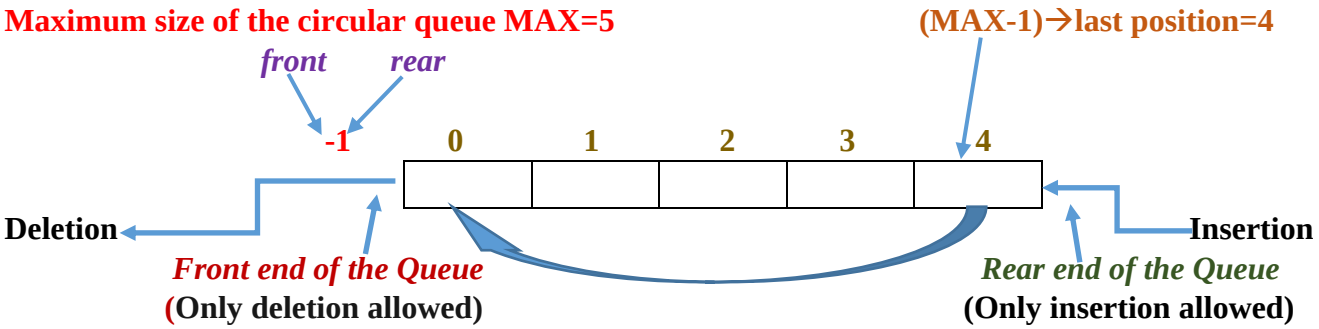
```
import java.util.Scanner;

public class CIRCULAR_QUEUE_DEMO
{
        static int MAX= 3; /* MAX is maximum size of the circular Q*/
        static int front = -1;  /*creates an empty circular Q*/
        static int rear = -1;  /*creates an empty circular Q*/
        static char Que[]= new char[MAX]; /*Que is the array to hold Q elements*/
        /*The above Que will hold character type data items*/
```

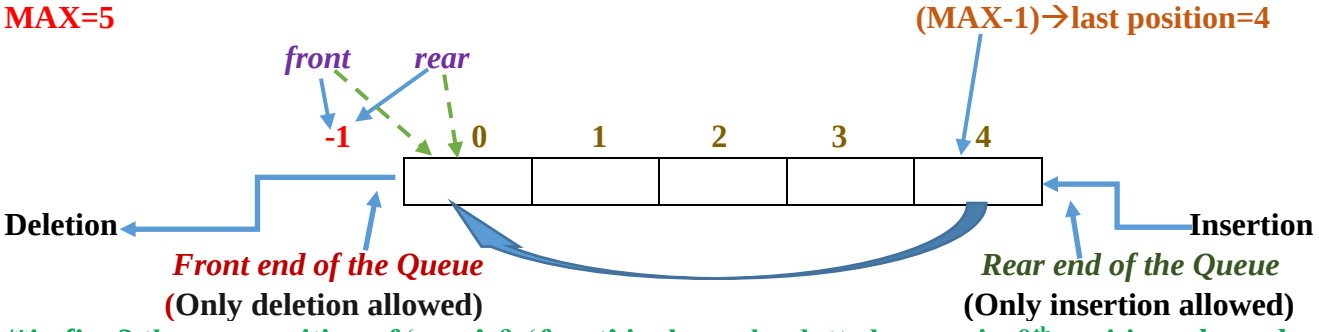
/*Fig: 1 An empty Circular Queue created after the above statements */




```
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                char ch;
                int opt;
                char data;

                do
                {
                        System.out.println("\n****CIRCULAR QUEUE***** ");
                        System.out.println("1.En-Queue 2.De-Queue 3");
                        System.out.println("3.DISPLAY/TRAVERSE");

                        System.out.print("\nEnter your option: ");
                        opt=sc.nextInt();

                        switch(opt)
                        {
                           case 1:
                                System.out.println("\nEnter the item to insert into the Q:");
                                data=sc.next().charAt(0);
                                 enQueue(data);
                                 break;
                           case 2:
                                deQueue();
                                break;
                           case 3:
                                TRAVERSE_Q();
```

```
                    break;
                default:
                    System.out.println("invalid option");
            } /*End of switch*/
            System.out.println("\nDo you want to perform another operation(y/n)");
            ch=sc.next().charAt(0);

        }while(ch=='y'|| ch== 'Y'); /*End of do…while loop*/

    } /*End of main method*/
/*insert_Q method to insert a new item at rear end of the circular Q*/
    public static void enQueue(char new_item)
    {
        if ( front == ( rear + 1) % MAX ) /*checks if the Q is full*/
        {
            System.out.println("The Queue is full");
            System.out.println("You can't insert more items");
            return;
        }
        else
        {   /*this else clause will be executed if the queue is not full*/

            if ( front < 0 )  /*checks if the queue is empty*/
            {
                front = rear = 0; /*store  the new item at 0th location of Q*/
```

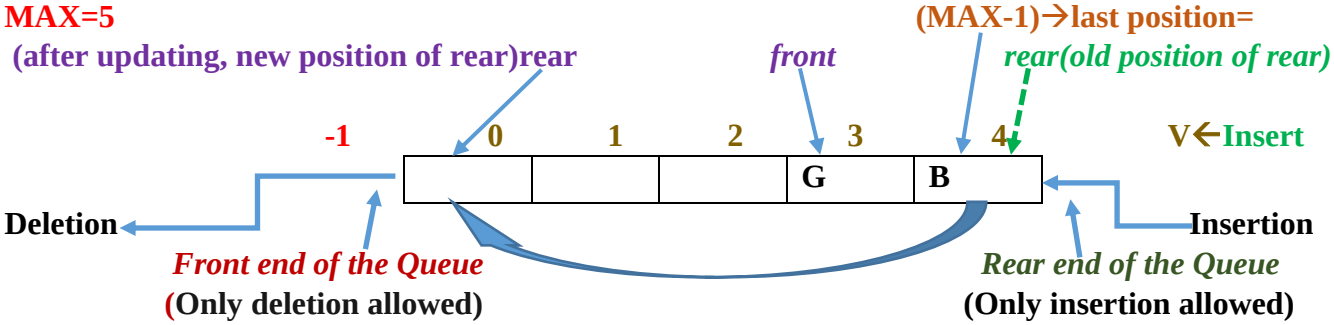
/*Fig: 2 When the circular Q is empty, before insertion of a new item */



/*in fig: 2 the new position of 'rear' & 'front' is shown by dotted arrow i.e 0th position where the new item will be inserted */

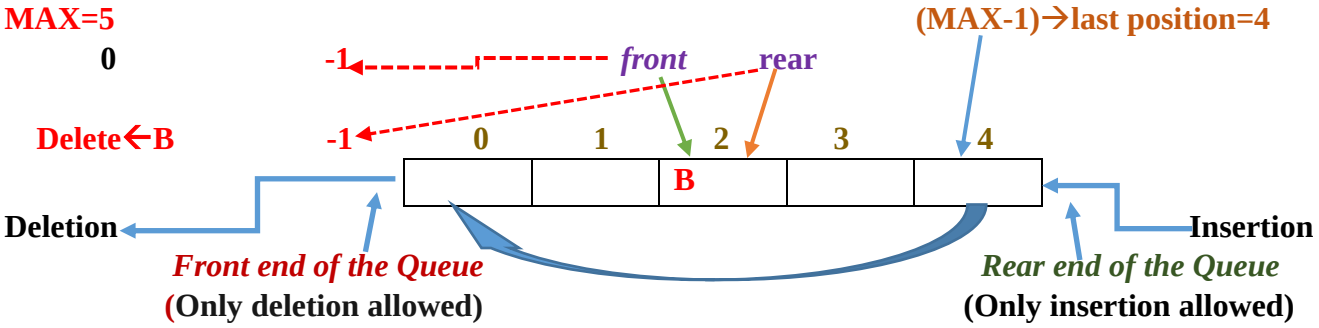
```
            } /*end of if clause*/
            else
            {  rear = ( rear + 1 ) % MAX ; /*move the 'rear' to next empty position*/
```

/*Fig: 3 when the Q is not empty, after updating rear pointer to next empty position i.e. to 0 w*/

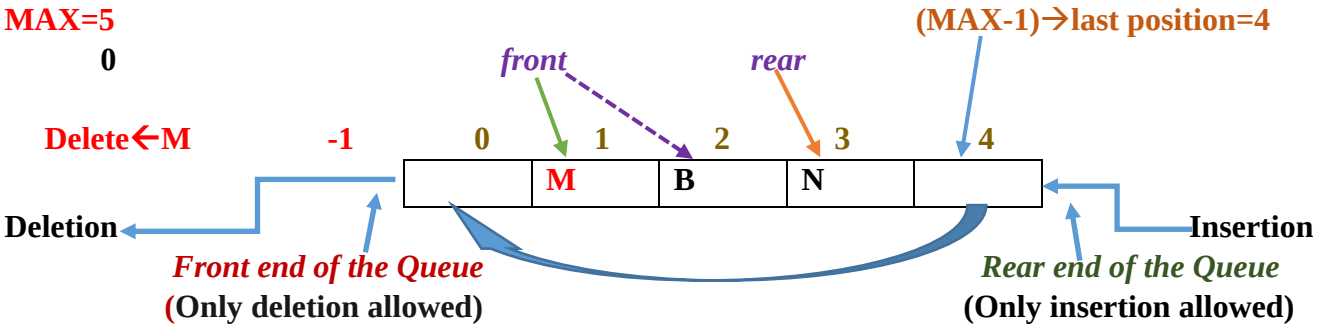



```
            } /* end of inner else clause*/
```

```
                Que[rear] = new_item; /*stores the  new item at new position of rear*/

            } /*End of outer else clause*/
        } /*End of insert queue method*/

/* Delete queue method that deletes the front most element from the  circular Q*/
        public static void deQueue()
        {
            if ( front < 0 ) /*checks whether the Q is empty?*/
            {
                System.out.println("The Queue is empty..you can't delete");
                return;
            }
            else
            {       char t = Que[front]; /*store the front element in t that is to be deleted*/

                if ( front == rear ) /*if the queue contains only one element? */
                {
                    front=rear= -1 ; /*the Q will be empty after deletion of one item*/
        /*Fig: 4 When the Q has only one element that is to be deleted */
```



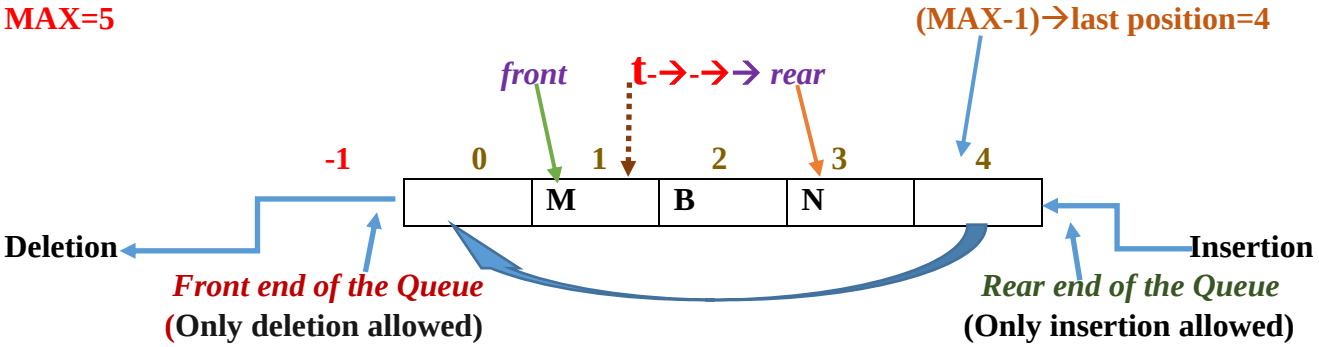

```
            /*after setting the 'rear' & 'front' to -1 as show by dotted arrow*/

                } /*End of inner if clause*/
                else
                {
                 front =  ( front + 1 ) % MAX ; /* move 'front' to next front element*/
/*Fig: 5 after updating rear pointer only to the next empty position when Q is not empty */
```




```
/*after deleting the front most item M, the new position of front pointer as shown by dotted arrow*/
                } /* End of inner else clause */
                System.out.print("\n The deleted node is : " + t);
            } /* End of outer else clause */
        } /*End of delete queue method*/
```

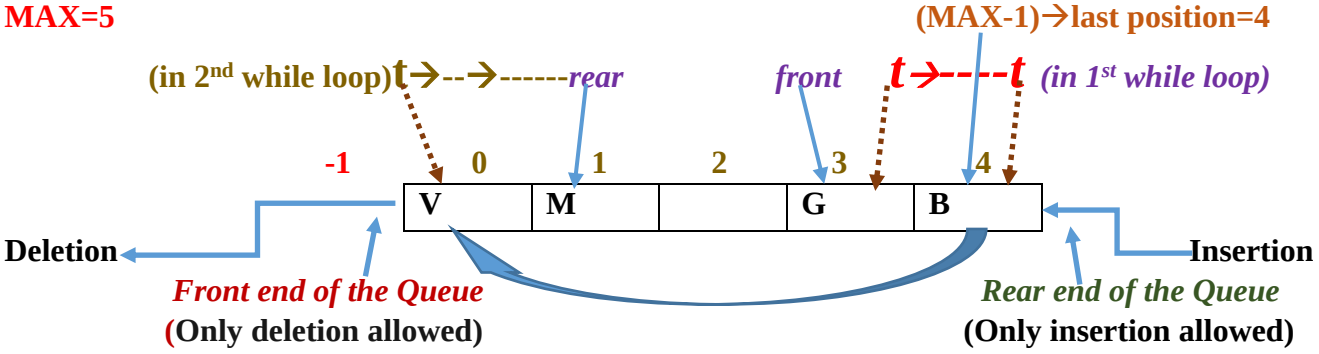
```
/* Traverse a circular Q*/
    public static void TRAVERSE_Q()
    {
        if ( front < 0 )
        {
            System.out.println("The Queue is empty..");
            return ;
        }
        else
        {   System.out.println("The Q elements from front to rear end are");

            int  t = front ;  /* t starts from 'front' pointers position*/

            if ( front <= rear ) /* if front element is behind the rear pointer*/
            {
```

/*Fig: 6 A circular Q when front element is behind the rear most element*/




```
                while( t <= rear ) /* t moves from 'front' up to the 'rear' pointers position*/
                {
                    System.out.print(Que[t] + " -> ");
                    t++;
                }
            }
            else /*if the rear pointer is behind the front pointer as show in fig:7*/
            {
```

/*Fig: 7 A non-empty circular Queue when rear element is behind front element */




```
                while ( t <= MAX - 1 ) /*this while…loop moves from G to B*/
                {
                    System.out.print( Que[t] + " -> " );
                    t++;
                }
```

```
                    t=0;
                    while( t <= rear ) /*this while…loop moves again from V to M*/
                    {
                        System.out.print(Que[t] + " -> ");
                        t++;
                    }
                } /*End of inner else clause */
            } /*End of outer else clause */
        }  /* End of traverse queue method*/

} /* END OF CIRCULAR_QUEUE_DEMO CLASS*/
```