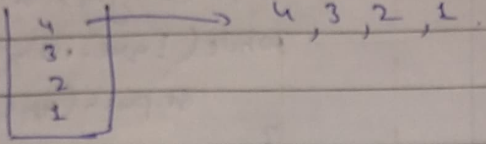
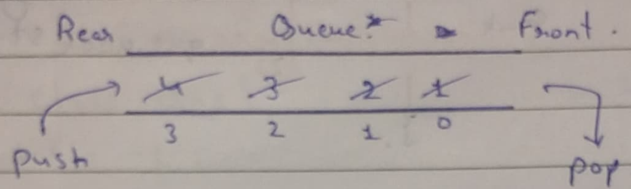
Queue

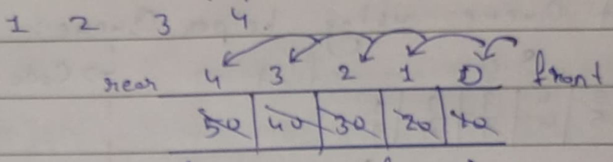
⇒ Stack :- LIFO or FILO [Last in First Out] or [First in Last Out].



⇒ Queue :- FIFO [First in First Out] or LILO [Last in Last Out].



size = 4



```
if ( rear == size) {
  // Overflow Condition
}
```

```
if ( rear == front) {
  // Underflow Condition
}
```

Basic ~~S/~~ Implementation :-

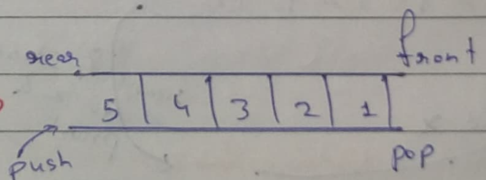
# Need to import :-
java.util.ArrayDeque;
java.util.LinkedList;
java.util.Queue;

=> Code :- psvm() {

```
Queue<Integer> qu = new ArrayDeque<>();
Queue<Integer> que = new LinkedList<>();
```

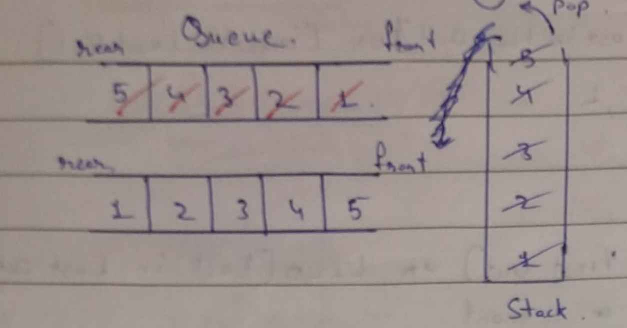
You can implement using this two methods

```
Sout(que.isEmpty());  ——→ True.

que.add(1);
que.add(2);
que.add(3);
que.add(4);
que.add(5);
Sout(que);  ——→ [1, 2, 3, 4, 5].
```



```
que.remove;  ——→ Remove ~~s~~ from front ——→ que => [2, 3, 4, 5].
que.poll();  ——→ Remove from front ——→ que => [3, 4, 5].
que.peek();  ——→ returns front. ——→ 3.

que.size();  ——→ 3
~~que~~ que.isEmpty();  ——→ False.
}
```

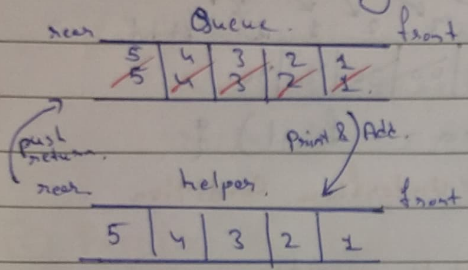
Reverse The Queue Using Stack :



=> Code :-

```
psv reverseQueue(que){
    Stack<Integer> st = new Stack<>();
    while(!que.isEmpty()){.
        st.push(que.remove());
    }
    while(!st.isEmpty()){.
        que.add(st.pop());
    }
}.
```

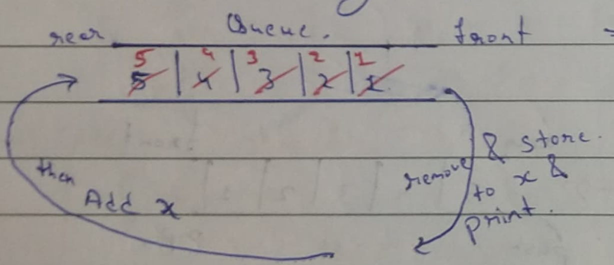
Print Queue using helper Queue :-



=> Code :-

```
psvm() {
    Queue<Integer> helper = new LinkedList<>();
    while (que.size() > 0){.
        Sout(que.peek() + " ");
        helper.add(que.remove());
    }.
    while(helper.size() >0){.
        que.add(helper.remove());
    }.
}.
```

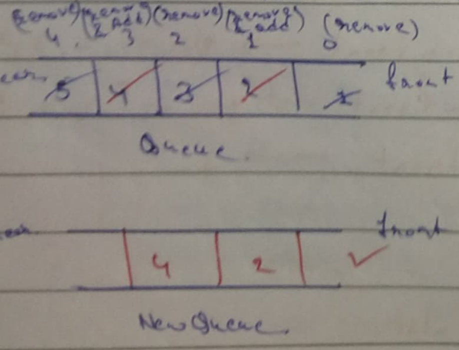
Print without using helper Queue :-



=>Code :-

```
psvm(){
    for(int i=0; i<que.size(); i++){.
        int x = que.peek();
        Sout(x + " ");
        que.remove();
        que.add(x);
    }.
```

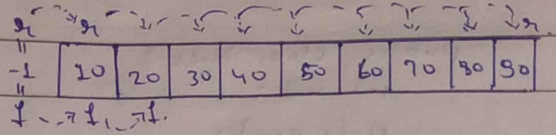
Remove Element At Even index in a Queue :-



=> Code :-

```
psv removeEven(que){
    newQueue = new LinkedList<>();
    while (!que.isEmpty()) {.
        que.remove();
        if(!que.isEmpty()){
            newQueue.add(que.remove());
        }
    }
    que = newQueue;
}
```

Array Implementation of Queue.



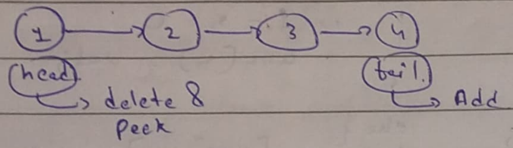
```
Add() & if(r.== arr.length -1) Queue is full.
if(r == -1 && f == -1)
    f = r = 0.
    arr[r] = val;
}
else { arr[~~r+1~~ ++r] = val;
}
```

```
to remove :- int x = arr[f];
             f++;
             return x;
```

```
=> Code :- public Class arrayImplementation {
   public class Que {
       int f = -1 ;
       int r = -1 ;
       int size = 0 ;
       int[] arr = new int[5];

       public void add(int val) {
           if(r == arr.length -1)
               Sout ("Queue is full"); } Overflow Condition.
               return;
           }
           if(f == -1 ) {
               f = r = 0 ;
               arr[r] = val;
           }
           else {
               arr[++r] = val;
           }
           size++;
       }.
       public int ~~void~~ remove() {.
           if( size == 0) {.
               Sout ("Queue is Empty"); } -> Underflow Condition
               return -1;
           }
           int val = arr[f++];
           size--;
           return val.
       }.

       public int peek() {
           if( size == 0) {
               Sout ("Queue is Empty");
           }   return -1;
           return arr[f];
       }.
```

```
public boolean isEmpty() {
    if(size == 0) return true;
    return false;
}

public void display() {.
    if( size == 0) {
        Sout("Queue is Empty");
    }.
    else {
        for(int i = f ; i<=r ; i++) {.
            Sout(arr[i] + " ");
        }.
    }.
    Sout();
}.

public int size() {.
    return size ;
}.
```

Linked List Implementation of Queue.



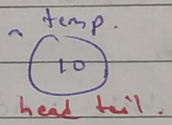
=> Code :- public ~~static~~ Class QueueLL{

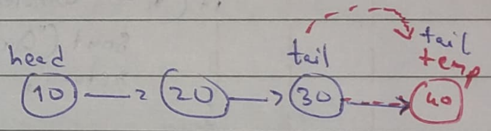
```
Node head ;
Node tail ;
int size = 0;
```

```
public class Node{
    ~~int val;~~
    Node next;
    Node(int val){
        this.val = val;
    }.
}
```

① Add Method :-

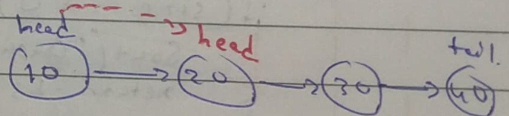
```
public void add(int val){
    Node temp = new Node(val)~~$~~;
    if(size == 0){
        head = tail = temp;
    }.
    else{
        tail.next = temp;
        tail = temp;
    }.
    size ++;
}
```





② Remove Method :-

```
public int remove( ) {
    if(size == 0){
        Sout("Queue is Empty");
        return -1;
    }
    int x = head.val;
    head = head.next ;
    size --;
    return x.
}.
```



③ Peek Method :-

```
public int peek(){
    if(size == 0){
        Sout ("Queue is Empty");
        return -1;
    }.
    return head.val ;
}
```

⑤ isEmpty Method :-

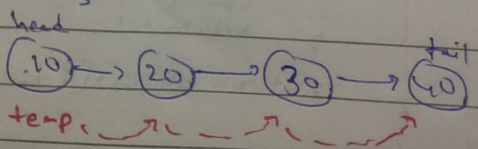
```
public boolean isEmpty(){
    return (size==0);
}
```

⑥ Size Method :-

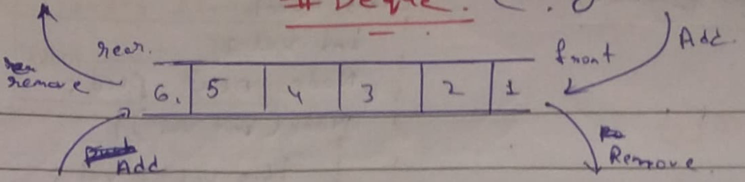
```
publis int size(){
    return size ;
}
```

④ Display Method :-

```
public void display(){
    Node temp = head;
    while(temp != null){
        Sout(temp.val + " ");
        temp = temp.next ;
    }.
    Sout();
}
```



Deque. (Doubly Ended Queue)



Deque<Integer> dq = new LinkedList<>(); } → Implementation

Operation:-

dq.addFirst(x); → Add element from front.

dq.add Last(x); → Add element from rear.

dq.add(x); → Add from rear Like Normal Queue.

dq.removeFirst(); → Remove ele from front.

dq.removeLast(); → Remove ele from Rear.

dq.remove(); → Remove ele from front Like Normal Queue

dq.getFirst(); → Get ele from front.

dq.getLast(); → Get ele from Rear.

dq.isEmpty(); → Check whether Deque is Empty or Not.

dq.size(); → Returns the size of Deque.

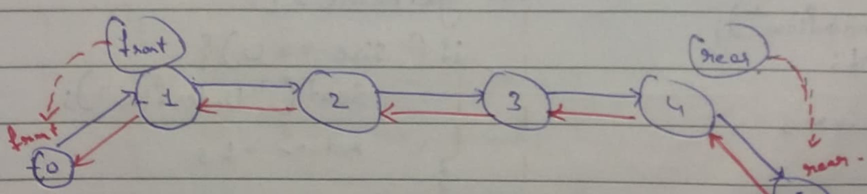
dq.removeAll(dq); → Remove All ele from Deque.

rear dq | 1 | 2 | 3 | 2 | 1 | front

dq.removeFirstOccurrence(2); → Rear | 1 | 2 | 3 | 1 | front

dq.removeLastOccurrence(1); → rear | 2 | 3 | 2 | 1 | front

Deque Implementation Using Doubly Linked List.



for deletefront :-
front = front.next.

for deleteRear :-
rear = rear.prev.

⟹ Code:- public Class dequeImplUsingDLL{.

```
Static Class Deque{
    Node front ;
    Node rear ;
    int Size = 0;
```

① Insert from front:-

```
void insertFront(int val){.
    Node temp = new Node(val);
    if(front == null){.
        rear = front = temp;
    }
    else{
        temp.next = front;
        front.prev = temp;
        front = temp;
    }
    size++;
}.
```

③ Delete from front:-

```
void deleteFront(){.
    if(size==0){.
        Sout("Underflow");
    }
    else{.
        front = front.next;
        if(front == null) rear = null;
        else{
            front.prev = null;
        }
        size--;
    }.
}
```

⑤ Get Front:-

```
int getFront(){
    ~~int~~if(size==0){.
        Sout("Underflow");.
        return -1;
    }
    return front.val;
}.
```

```
Class Node{
    int val;
    Node next, prev;
    Node(int val){.
        this.val = val;
    }
}.
```

② Insert from Rear:-

```
void insertRear(int val){.
    Node temp = new Node(val);
    if(rear == null){
        rear = ~~bean~~ front = temp;
    }
    else{.
        temp.prev = rear;
        rear.next = temp;
        rear = temp;.
    }
    size++;
}.
```

④ Delete from Rear:-

```
void deleteRear(){.
    if(size==0){.
        Sout("Underflow");
    }
    else{
        rear = rear.prev;
        if(rear == null)
            front = null;
        else{
            rear.next = null;
        }
        Size--;
    }
}.
```

⑥ Get Rear:-

```
int getRear(){
    if( size == 0){.
        Sout("Underflow");
        return -1;
    }.
    return rear.val;
}.
```

(7) Display from Front():-

```
void displayFront() {.
    Node temp = front;
    while(temp != null){.
        Sout(tem.val + " ");
        temp= temp.next;
    }.
    Sout();
}
```

~~(8) Display from Rear~~ (10) isEmpty:-

```
int isEmpty(){.
    return (size ==0);.
}
```

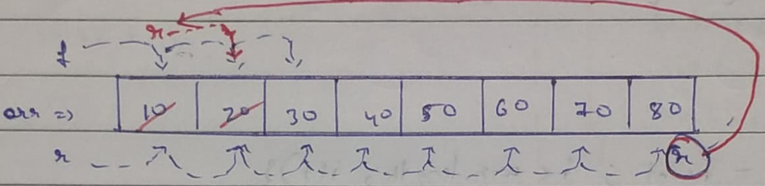
(9) size :-

```
int size(){.
    return size;
}.
```

(8) Display from Rear:-

```
Void displayRear(){
    Node temp = Rear;
    while(temp != null){
        Sout(temp.val + " ");
        temp = temp.~~next~~ prev;
    }
    Sout();
}.
```

Circular Queue Implementation.
Using Array.



=> Code :-

```
public Class CircularQueueArray {.
public static Class cQueArr{.
    int front = -1;
    int rear = -1;
    int size = 0;
    int[] arr = new int[5];
    public void add(int val){.
        if(size == arr.length){
            Sout(~~arr~~"Cirular Queue is full");
            return;
        }.
        else if( size == 0){.
            front = rear = 0;
            arr[rear] = val;
        }.
        else if(rear < arr.length-1){.
            arr[~~x~~ ++ rear] = val;
        }.
        else if(rear == arr.length ~~X~~ -1){.
            rear = 0;
            arr[rear] = val;
        }
        size ++;
```

```
    public int remove(){.
        if(size==0){.
            Sout("Queue is Empty");
            return -1;
        }.
        else{
            int val = arr[front];
            if(front == arr.length-1)
                front = 0;
            else front ++;.
            Size --;
            return val;
        }
    }.
```

```
public int peek(){
    if (size == 0){
        Sout ("Queue is Empty");
        return -1;
    }
    ~~else~~ return arr[front];
}.
```

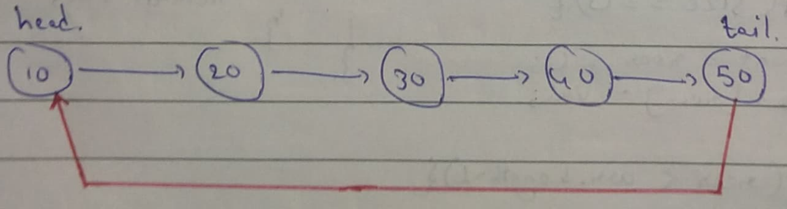
```
public void display(){
    if (size == 0){
        Sout ("Queue is Empty");
        return;
    }
    else if (front <= rear){
        for (int i = front ; i <= rear ; i++){
            Sout(arr[i] + " ");
        }.
    }
    else {
        for (int i = front ; i< arr.length ; i++){
            Sout(arr[i] + " ");
        }
        for (int i = 0 ; i <= rear ; i++){
            Sout (arr[i] + " ");
        }.
    }
    Sout();
}.
```

```
public boolean isEmpty(){
    return (size == 0);
}
```

```
public int size(){
    return size;
}
```

```
public boolean isFull(){
    return (size == arr.length);
}
```

Circular Queue Implementation Using ~~By~~ Linked List.



⟹ Code :- public Class CircularQueueLL{

```
public Class CircularLL{
    Node head;
    Node tail;
    int size = 0;
```

```
public statis Class ~~Node{~~ Node{
    int val;
    Node next;
    Node(int val){
        this.val = val;
    }
}
```

```
    public void add(int val){
        Node temp = new Node(val);
        if(size == 0){
            head = tail = temp;
        }
        else{
            tail.next = temp;
            tail = temp;
            tail.next = head;
        }
        size++;
    }
```

```
public int remove() throws Exception{
    if(size == 0){
        throw new Exception("Queue is Empty");
    }
    else{
        int x = head.val;
        head = head.next;
        tail.next = head;
        size--;
        return x;
    }
}
```

```
publis void display(){
    if(size == 0){
        Sout("Queue is Empty");
        return;
    }
    else{
        Node temp = head;
        while(true){
            Sout(temp.val + " ");
            if(temp.next == head)
                break;
            temp = temp.next;
        }
        Sout();
    }
}
```

```
public int peek() ~~{~~ throw Exception{
    if(size == 0){
        throw ~~B~~ new Exception("Queue is Empty");
    }
    return head.val;
}
```

```
public boolean isEmpty(){
    ~~R~~return (size == 0);
}
```