QUEUE

Creating a queue using array

```
public class QueueUSingARRAYS {
    static class Queue{
        static int arr[];
        static int size;
        static int rear;
        // constructer calling
         public Queue(int n){
            arr=new int[n];
            size=n;
            rear=-1;
         }


    public static boolean isEmpty(){
        return rear==-1;
    }

    // adding an element in queue
    public static void add(int data){
        if(rear==size-1){
            System.out.println("Queue is full");
            return;
        }
        rear=rear+1;
        arr[rear]= data;
    }

    // removing an element in Queue
    public static int remove(){
        if(isEmpty()){
            System.out.println(" que isempty");
            return -1;
        }
        int front=arr[0];
        for(int i=0;i<rear;i++){
            arr[i]=arr[i+1];
        }
```

```
        rear=rear-1;
        return front;

    }

    // peek()
    public static int peek(){
        if(isEmpty()){
            System.out.println(" que isempty");
            return -1;
        }
        return arr[0];
    }
  }

  public static void main(String[] args) {
    Queue q= new Queue(5);
    q.add(1);
    q.add(2);
    q.add(3);

    while(!q.isEmpty()){
        System.out.println(q.peek());
        q.remove();
    }
  }

}
Output:
1
2
3
```

Code for circular Queue:

```
public class CircularQueue {
    static class Queue{
        static  int arr[];
        static int size;
        static int rear;
```

```
        static int front;

        public Queue(int n){
        arr= new int[n];
        size=n;
        rear=-1;
        front=-1;
        }


        public static boolean isempty(){
            return rear==-1&&front==-1;

        }

        public static boolean isFull(){
            return (rear+1)%size==front;
        }

        public static void add(int data){
            if(isFull()){
                System.out.println("Queue is full");
                return;
            }
            if(front==-1){
                front=0;
            }
            rear= (rear+1)%size;
            arr[rear]=data;
        }

        public static int remove(){
            if(isempty()){
                System.out.println("queue is empty");
                return -1;

            }


            int result= arr[front];

            if(rear==front){
                rear=front=-1;
            }else{
                front=(front+1)%size;
```

```
            }
            return result;

    }
    public static int peek(){
        if(isempty()){
            System.out.println(" que isempty");
            return -1;
        }
        return arr[front];
    }

    }
    public static void main(String[] args) {
        Queue q= new Queue(3);
        q.add(1);
        q.add(2);
        q.add(3);
        System.out.println(q.remove());
        q.add(4);
        System.out.println(q.remove());
        q.add(5);
        while(!q.isempty()){
            System.out.println(q.peek());
            q.remove();
        }

    }

}
Output:
1
2
3
4
5
```

Queue using Linked List:

```
import java.util.*;
public class QueUsingLL {
    static class Node{
```

```
        int data;
        Node next;
     Node(int data){
        this.data= data;
        this.next=null;

    }}
    static Node head=null;
    static Node tail=null;
    static class Queue{




        public static boolean isempty(){
            return head==null && tail==null;
        }

        public static void add(int data){
            Node newnode =new Node(data);
            if(head==null){
                head=tail=newnode;
                return;
            }
            tail.next=newnode;
            tail=newnode;
        }

        public static int remove(){
            if(isempty()){
                System.out.println("empty");
                return -1;
            }
            int front=head.data;
            if(tail==head){
                tail=head=null;

            }else{

            head=head.next;}
            return front;
        }

        public static int peek(){
            if(isempty()){
```

```
                System.out.println("empty");
                return -1;
            }
            return head.data;
        }

    }
    public static void main(String[] args) {
        Queue q= new Queue();
        q.add(1);

        q.add(2);
        q.add(2);

        while(!q.isempty()){
            System.out.println(q.peek());
            q.remove();
        }
    }

}
Output:
1
2
2
```

Queue Using Java Collections frameworks:

```
import java.util.Queue;
import java.util.LinkedList;
public class QueusingJCF {
    public static void main(String[] args) {
        Queue q= new LinkedList<>() ;
        q.add(1);
        q.add(2);
        q.add(3);

        while(!q.isEmpty()){
            System.out.println(q.peek());
            q.remove();

        }
    }
}
Output:
```

```
1
2
3
```

Queue using two Stacks:

```
import java.util.*;
public class queusingtwostacks {

    static class Queue{
        static Stack <Integer> s1= new Stack<>();
        static Stack <Integer> s2= new Stack<>();

        // is empty
        public static boolean isEmpty(){
            return s1.isEmpty();
        }

        // add
        public static void add(int data){
            while(!s1.isEmpty()){
                s2.push(s1.pop());// transfer from s1 to s2
            }
            s1.push(data); // push data
            while(!s2.isEmpty()){
                s1.push(s2.pop());
            }
        }
        //remove
        public static int remove(){
            if(isEmpty()){
                System.out.println("empty");
                return -1;
            }
           return  s1.pop();
        }
        //peek
        public static int peek(){
            if(isEmpty()){
                System.out.println("empty");
                return-1;
            }
            return s1.peek();
        }
```

```
    }
    public static void main(String[] args) {
        Queue q= new Queue();
        q.add(1);
        q.add(2);
        q.add(3);
        while(!q.isEmpty()){
            System.out.println(q.peek());
            q.remove();
        }
    }

}
Output:
1
2
3
```

Stack using two queues:

```
import java.util.*;
public class stackusingtwoQues {
    static class stack{
        static Queue<Integer>q1= new LinkedList<>();
        static Queue<Integer>q2= new LinkedList<>();

        // is empty
        public static boolean isempty(){
            return q1.isEmpty() && q2.isEmpty();
        }

        // add or  push

        public static void push(int data){
            if(!q1.isEmpty()){
                q1.add(data);

            }else{
                q2.add(data);
            }
        }

        //pop  or remove
```

```
    public static int pop(){
        if(isempty()){
            System.out.println("empty");
            return -1;
        }
        int top=-1;
        if(!q1.isEmpty()){
            while(!q1.isEmpty()){
                top=q1.remove();

            if(q1.isEmpty()){
                break;

            }
            q2.add(top);
        }
        }else{
            while(!q2.isEmpty()){
                top=q2.remove();
                if(q2.isEmpty()){
                    break;
                }
                q1.add(top);
            }
        }
        return top;
    }
    public static int peek(){
        if(isempty()){
            System.out.println("empty");
            return -1;
        }
        int top=-1;
        if(!q1.isEmpty()){
            while(!q1.isEmpty()){
                top=q1.remove();

            q2.add(top);
        }
        }else{
            while(!q2.isEmpty()){
                top=q2.remove();

                q1.add(top);
```

```
                }
            }
            return top;
        }
    }
    public static void main(String[] args) {
        stack s= new stack();
        s.push(1);
        s.push(2);
        s.push(3);
        while(!s.isempty()){
            System.out.println(s.peek());
            s.pop();
        }
    }
}output:
3
2
1
```

Code for non repeating element in a string using queue:

```
import java.util.*;
public class nonrepeatingLetter {
    public static void nonrepeatingLetters(String str){
        //create a freaquency alphabets array
        int frequency[]= new int[26];
        // queue  creation
        Queue <Character> q= new LinkedList<>();
        // insert characters of a string into queue
        for(int i=0;i<str.length();i++){
            char ch= str.charAt(i);
            q.add(ch);
            frequency[ch-'a']++;
            while(!q.isEmpty() && frequency[q.peek()-'a']>1){
                q.remove();
            }
            if(q.isEmpty()){
                System.out.println(-1+" ");
            }else{
```

```
                System.out.println(q.peek());
            }
        }

    }
    public static void main(String[] args) {
        String str= "aabccxb";
        nonrepeatingLetters(str);
    }
}
Output:
a
-1
b
b
b
b
x
```

reverse a Queue:

```
import java.rmi.Remote;
import java.util.*;
public class reverseAQUEUE {
    public static void reverse(Queue<Integer> q){
        Stack<Integer> s= new Stack<>();
        // int size=q.size();

        // for(int i=0;i<size;i++){
        //     s.add(q.remove());
        // } OR
        while(!q.isEmpty()){
            s.push(q.remove());
        }

        while(!s.isEmpty()){
            q.add(s.pop());
        }
    }
```

```
    public static void main(String[] args) {
        Queue<Integer> q= new LinkedList<>();
        q.add(1);
        q.add(2);
        q.add(3);
        q.add(4);
        reverse(q);
        while(!q.isEmpty()){
            System.out.print(q.peek()+" ");
            q.remove();
        }

    }


}
Output:
4 3 2 1
```

Deque operations:

```
import java.util.*;
public class DequeUsingJCF {
    public static void main(String args[]){
        Deque<Integer> dq= new LinkedList<>();
        dq.addFirst(3);
        dq.addFirst(2);
        dq.addFirst(1);
        dq.addLast(4);
        System.out.println(dq);
        dq.removeLast();
        System.out.println(dq);
        dq.removeFirst();
        System.out.println(dq);
        System.out.println("First el =" +dq.getFirst());
        System.out.println("last el =" +dq.getLast());
      }

}
Output:
[1, 2, 3, 4]
[1, 2, 3]
```

```
[2, 3]
First el =2
last el =3
```

Queue using deque: `import java.util.*;`

```
public class QueueUsingDeque {
    static class Queue{
        Deque <Integer> dq= new LinkedList<>();
        // add method
        public  void add(int data){
            dq.addLast(data);
        }
        // remove method
        public int remove(){
            return dq.removeFirst();
        }
        // peek method
        public int peek(){
            return dq.getFirst();
        }
    }
    public static void main(String[] args) {
        Queue q= new Queue();
        q.add(1);
        q.add(2);
        q.add(3);
        System.out.println("peek -" +q.peek());
        System.out.println(q.remove());
        System.out.println(q.remove());
        System.out.println(q.remove());



    }

}
Output:
peek -1
1
2
3
```

Stack using deque:

```
import java.util.LinkedList;

import java.util.*;
public class StackUsingDeque {
    public class Satck{
        Deque<Integer> dq= new LinkedList<>();

        // method for push in stack
        public void push(int data){
            dq.addLast(data);
        }

        // method for pop in stack
        public int pop(){
            return dq.removeLast();
        }
        // method for peek in stack
        public int peek(){
            return dq.getLast();
        }

    }
    public static void main(String args[]){
        Stack s= new Stack();
        s.push(1);
        s.push(2);
        s.push(3);
        System.out.println("peek-"+s.peek());
        while(!s.isEmpty()){
            System.out.println(s.peek());
            s.pop();
        }
    }
}
Output:
peek-3
3
2
1
```

Genetating binary number for given number:

```
import java.util.*;
public class generateBINarayNUMber {

    static void getBinNum(int n){
        Queue<String> q= new LinkedList<>();
        q.add("1");

        while(n-->0){
        String s1= q.peek();
        q.remove();
        System.out.println(s1);
        String s2= s1;
        q.add(s1+"0");
        q.add(s2+"1");

    }
}
public static void main(String[] args) {
    int n=5;
    getBinNum(n);
}
}
Output:
1
10
11
100
101
```

Finding minimum cost:

```
import java.util.*;

import javax.print.attribute.Size2DSyntax;
public class findingMinCost {
    static int getMinCost(int arr[], int n){
        PriorityQueue<Integer> pq= new PriorityQueue<Integer>();
        for(int i=0;i<arr.length;i++){
            pq.add(arr[i]);
        }
        int res=0;
        while(pq.size()>1){
```

```
            int first=pq.poll();
            int second=pq.poll();
            res+=first+second;
            pq.add(first+second);
        }
        return res;
    }
    public static void main(String[] args) {
        int arr[]={4,3,2,6};
        int size=arr.length;
        System.out.println("min cost="+getMinCost(arr, size));
    }

}
Output:
min cost=29
```