# Lab 8

**Task 1,2,3,5 using linked list**

public class Queue\_2\_LL

{

    public static class Node

    {

        int data;

        Node next;

        Node(int x)

        {

            data = x;

            next = null;

        }

    }

    public static class Queue

    {

        static Node head = null;

        static Node tail = null;

        public static boolean isEmpty()

        {

            return head == null && tail == null;

        }

        public static void add(int x)

        {

            Node nnode = new Node(x);

            if(head == null)

            {

                head = tail = nnode;

                return;

            }

            tail.next = nnode;

            tail = tail.next;

        }

        public static int remove()

        {

            if(tail == null)

            {

                System.out.println("The queue is empty");

                return -1;

            }

            int ndata = 0;

            if(head == tail)

            {

                ndata = head.data;

                head = null;

                tail = null;

                return ndata;

            }

            ndata = head.data;

            head = head.next;

            return ndata;

        }

        public static int peek()

        {

            if(head == null)

            {

                System.out.println("The queue is empty.");

                return -1;

            }

            // int front = head.data;

            return head.data;

        }

        // Task 1

        // sum of all integers in a queue

        public static int printSum()

        {

            if(isEmpty()) //Calling isEmpty() method

            {             //to check whether queue is empty

                System.out.println("The queue is Empty.");

                return -1;

            }

            int sum = 0; //taking sum variable as zero

            Node node = head;   //declaring a temporary node

            while(node != null) //for traversing node till null

            {

                sum += node.data;  //addition of node data

                node = node.next;

            }

            return sum;     //returning final sum

        }

        // Task 2

        public static Node find\_Kth\_Node(int k)

        {

            Node cNode = head;

            int i = 1;

            while(i < k)

            {

                cNode = cNode.next;

                i++;

            }

            return cNode;

        }

        // Reversing till nth node and returning the nth node

        public static void reverse\_K\_Ele(int k)

        {

            if(isEmpty())

            {

                System.out.println("The queue is Empty.");

                return;

            }

            Node cNode = find\_Kth\_Node(k);

            // Saving the reference of nodes after nth node

            Node temp1 = cNode.next;

            cNode.next = null;  // Breaking the Queue

            Node pNode = null;

            Node curNode = head;

            Node nextNode;

            while(curNode != null)

            { // reversing by swaping two nodes places with eachother

                nextNode = curNode.next; //saving current node

                curNode.next = pNode; // making previous node next

                pNode = curNode;

                curNode = nextNode;

            }

            // When queue is reversed the last element will

            // become the head of the queue

            head = pNode;

            //Traverse till the end of first half

            Node tempNode = head;

            while(tempNode.next != null)

            {

                tempNode = tempNode.next;

            }

            tempNode.next = temp1;

        }

        // Task 3

        public static double average(int k)

        {

            Node node = head;

            int sum = 0;

            Node node2 = find\_Kth\_Node(k); //kth element

            while(node != node2.next)

            {

                sum += node.data;

                node = node.next;

            }

            //Directly returning average by dividing by k

            return (double)sum/k;

        }

        public static void printQueue()

        {

            if(isEmpty())

            {

                System.out.println("The queue is Empty.");

                return;

            }

            Node cNode = head;

            while(cNode != null)

            {

                System.out.print(cNode.data + " -> ");

                cNode = cNode.next;

            }

            System.out.println(" Null");

        }

        // Task 5

        public static boolean checkPalindrome()

        {

            if(head == null)

            {

                System.out.print("The list is empty");

                return  false;

            }

            int[] arr = new int[10];

            Node cuNode = head;

            int i = 0;

            while(cuNode != null)

            {

                arr[i++] = cuNode.data;

                cuNode = cuNode.next;

            }

            reverse\_K\_Ele(5);

            boolean ispalindrome = true;

            Node cNode = head;

            i = 0;

            while(cNode != null)

            {

                if(arr[i++] != cNode.data)

                {

                    ispalindrome = false;

                }

            }

            return ispalindrome;

        }

    }

    public static void main(String[] args) {

        Queue.add(3);

        Queue.add(4);

        Queue.add(7);

        Queue.add(5);

        System.out.println(Queue.peek());

        Queue.printQueue();

        System.out.println(Queue.remove());

        Queue.printQueue();

        Queue.add(12);

        Queue.add(9);

        Queue.add(0);

        Queue.printQueue();

        Queue.reverse\_K\_Ele(3);

        // System.out.println(node.data);

        Queue.printQueue();

        System.out.println(Queue.printSum());

        System.out.println(Queue.average(3));

        System.out.println(Queue.checkPalindrome());

    }

}

**Task 4 using array**

public class Circular\_Que\_1\_array

{

    static class Queue

    {

        static int arr[];

        static int size;

        static int rear = -1;

        static int front = -1;

        //we don't need to make front as it will always be zero in array

        Queue(int n) //size

        {

            arr = new int[n];

            // this.size = n;

            size = n;

        }

        public static boolean isEmpty()

        {

            return rear == -1 && front == -1;

        }

        public static boolean isFull()

        {

            return (rear + 1) % size == front;

        }

        //enqueue

        public static void add(int data)

        {

            if(isFull())

            {

                System.out.println("full queue");

                return;

            }

            if(front == -1)

            {

                front = 0;

            }

            rear = (rear + 1) % size;

            arr[rear] = data;

        }

        //dequeue

        public static int remove()

        {

            if(isEmpty())

            {

                System.out.println("Empty queue");

                return -1;

            }

            int result = arr[front];

            if(rear == front)

            {

                rear = front = -1;

            }

            else{

                front = (front + 1) % size;

            }

            return result;

        }

        //peek

        public static int peek()

        {

            if(isEmpty())

            {

                System.out.println("Empty queue");

                return -1;

            }

            return arr[front];

        }

    }

    public static void main(String[] args) {

        new Queue(5);

        Queue.add(1);

        Queue.add(2);

        Queue.add(3);

        Queue.add(4);

        Queue.add(5);

        System.out.println(Queue.remove());

        Queue.add(6);

        System.out.println(Queue.remove());

        Queue.add(7);

        while(!Queue.isEmpty())

        {

            System.out.print(Queue.peek() + " ");

            Queue.remove();

        }

    }

}