****Linked list implementation and it’s operations like****

**addfirst, addAt index ,addLast , deleteFirst ,deleteAt index ,delete last ,reverse list , print list ,search in list**

class linkedlist{

    Node head;

    static int size=0;

    class Node{

        int data;

        Node next;

        Node(int data){

            this.data=data;

            next=null;

        }

    }

    public void addfirst(int data){

        Node newnode=new Node(data);

        if(head==null){

            head=newnode;

            size++;

            return;

        }

        newnode.next=head;

        head=newnode;

        size++;

    }

    void addAt(int idx,int data){

        Node newnode=new Node(data);

        Node currnode=head;

        for(int i=0;i<idx-1;i++){

            currnode=currnode.next;

        }

        newnode.next=currnode.next;

        currnode.next=newnode;

        size++;

    }

    public void addlast(int data){

        Node newnode=new Node(data);

        if(head==null){

            head=newnode;

            size++;

            return;

        }

        Node currnode=head;

        while(currnode.next!=null){

            currnode=currnode.next;

        }

        currnode.next=newnode;

        size++;

    }

    public void printlist(){

        Node currnode=head;

        if(head==null){

            System.out.println("list is empty");

            return;

        }

        while(currnode!=null){

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

            currnode=currnode.next;

        }

        System.out.println("Null");

    }

    public void deletefirst(){

        if(head==null){

            System.out.println("List is empty");

            return ;

        }

        head=head.next;

        size--;

    }

    void deleteAt(int idx){

        Node currnode=head;

        Node newnext=currnode.next;

        for(int i=0;i<idx-1;i++){

            newnext=newnext.next;

            currnode=currnode.next;

        }

        currnode.next=newnext.next;

        size--;

    }

    public void deletelast(){

        if(head==null){

            System.out.println("list is empty");

            return;

        }

        Node last=head;

        while(last.next.next!=null){

            last=last.next;

        }

        last.next=null;

    }

    public void reverselist(){

        if(head==null || head.next==null){

            System.out.println("Empty");

            return;

        }

        Node previous=head;

        Node currnode=head.next;

        while(currnode!=null){

            Node nextnode=currnode.next;

            currnode.next=previous;

            previous=currnode;

            currnode=nextnode;

        }

        head.next=null;

        head=previous;

    }

    public static void main(String args[]){

        linkedlist list=new linkedlist();

        list.addfirst(1);

        list.addlast(2);

        list.addlast(3);

        list.addlast(4);

        list.addAt(2,5);

        list.printlist();

        list.deleteAt(2);

        list.reverselist();

        list.deletefirst();

        list.printlist();

        list.deletelast();

        list.printlist();

    }

}

Queue implememtation using linked list And its operations like enque(add) , deque(remove),isEmpty,Peek,size

public class Qusinglinkedlist {

    static Node head=null;

    static Node tail=null;

    static class Node{

        int data;

        Node next;

        Node(int data){

            this.data=data;

            next=null;

         }

    }

    static class queue{

        public boolean isempty(){

            return tail==null;

        }

        public void add(int data){

            Node newnode=new Node(data);

            if(isempty()){

                head=tail=newnode ;

            }

            else{

            tail.next=newnode;

            tail=newnode;

            }

        }

        public int remove(){

            if(isempty()){

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

                return -1;

            }

            int res=head.data;

            if(head==tail){

                tail=null;

            }

            head=head.next;

            return res;

        }

        public int peek(){

            if(isempty()){

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

                return -1;

            }

            return head.data;

        }

    }

    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();

        }

    }

}

Queue implementation using Array And its operations like enque(add) , deque(remove), isEmpty , isFull ,Peek, size

class qusingarray{

    static class queue{

        static int size;

        static int rear;

        static int arr[];

        queue(int size){

            this.size=size;

            arr=new int[size];

            rear=-1;

        }

        public static boolean isempty(){

            return rear==-1;

        }

        public boolean isfull(){

            return rear==size-1;

        }

        public void add(int data){

            if(isfull()){

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

                return ;

            }

            rear++;

            arr[rear]=data;

        }

        public int remove(){

            if(isempty()){

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

                return -1;

            }

            int front=arr[0];

            for(int i=0;i<rear;i++){

                arr[i]=arr[i+1];

            }

            rear--;

            return front;

        }

        public int peek(){

            if(isempty()){

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

                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();

        }

    }

}

Queue implementation using Array list And its operations like enque (add) , deque(remove), isEmpty, Peek, size

public class queueusingarraylist {

    static class queue{

        static int arr[];

        static int size;

        static int front=0;

        static int rear=-1;

        public queue(int n){

            arr=new int[n];

            this.size=n;

        }

        public boolean isempty(){

            return rear==-1;

        }

        public void add(int a){

            if(rear==size-1){

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

                return ;

            }

            rear++;

            arr[rear]=a;

        }

        public int remove(){

            if(isempty()){

                return -1;

            }

            int removeitem=arr[front];

            for(int i=0;i<rear;i++){

                arr[i]=arr[i+1];

            }

            front++;

            rear--;

            return removeitem;

        }

        public int peek(){

            if(isempty()){

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

                return -1;

            }

            return arr[front];

        }

    }

    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();

        }

    }

}

# Stack implementation using linked list And its operations like push , pop ,peek ,isEmpty,size

  public class stackusinglinkedlist{

         public static class Node{

        int data;

        Node next;

        Node(int data){

            this.data=data;

            next=null;

        }

    }

        public static Node head;

        int size=0;

        public boolean isempty(){

            return head==null;

        }

        public void push(int data){

            Node newnode=new Node(data);

            if(isempty()){

                head=newnode;

                size++;

                return;

            }

            newnode.next=head;

            head=newnode;

            size++;

        }

        public int pop(){

            if(isempty()){

                return -1;

            }

            int top=head.data;

            head=head.next;

            size--;

            return top;

        }

        public int peek(){

            if(isempty()){

                return -1;

            }

            return head.data;

        }

    public static void main(String args[]){

        stackusinglinkedlist st=new stackusinglinkedlist();

        st.push(1);

        st.push(2);

        st.push(3);

        st.push(4);

        System.out.println(st.size);

        while(!st.isempty()){

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

            st.pop();

        }

        System.out.println(st.size);

    }

}

# Stack implementation using Array list And its operations like push , pop ,peek ,isEmpty,size

import java.util.ArrayList;

class stackusingarraylist{

    static class stack{

       ArrayList<Integer> list=new ArrayList<>();

       int size=0;

       public boolean isempty(){

        return list.size()==0;

       }

       public void push(int data){

        list.add(data);

        size++;

       }

       public int pop(){

        if(isempty()){

            return -1;

        }

        int top=list.get(list.size()-1);

        list.remove(list.size()-1);

        size--;

        return top;

       }

       public int peek(){

        if(isempty()){

            return -1;

        }

        return list.get(list.size()-1);

       }

    }

    public static void main(String args[]){

        stack st=new stack();

        st.push(1);

        st.push(2);

        st.push(3);

        st.push(4);

        System.out.println("size of stack is before pop :"+st.size);

        while(!st.isempty()){

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

            st.pop();

        }

        System.out.println("size of stack is after pop :"+st.size);

    }

}

# Binary tree implementation and traversal techniques like Inorder , preorder , post order ,level order

import java.util.\*;

public class tree{

    static class Node{

        int data;

        Node left,right;

        Node(int d){

            data=d;

            left=right=null;

        }

    }

    static class binary{

        static int idx=-1;

        public  Node build(int nodes[]){

            idx++;

            if(nodes[idx]==-1){

                return null;

            }

            Node newnode=new Node(nodes[idx]);

            newnode.left=build(nodes);

            newnode.right=build(nodes);

            return newnode;

        }

        public void inorder(Node root){

            if(root==null){

                return ;

            }

            inorder(root.left);

            System.out.print(root.data+" ");

            inorder(root.right);

        }

        public void preorder(Node root){

            if(root==null){

                return;

            }

            System.out.print(root.data+" ");

            preorder(root.left);

            preorder(root.right);

        }

        public void postorder(Node root){

            if(root==null){

                return;

            }

            postorder(root.left);

            postorder(root.right);

            System.out.print(root.data+" ");

        }

        public void levelorder(Node root){

            if(root==null){

                return ;

            }

            Queue<Node> q=new LinkedList<>();

            q.add(root);

            q.add(null);

            while(!q.isEmpty()){

                Node curr=q.remove();

                if(curr==null){ System.out.println();

                    if(q.isEmpty()){break;}

                    else{q.add(null);}

                }

                else{System.out.print(curr.data+" ");

                if(curr.left!=null) q.add(curr.left);

                if(curr.right!=null) q.add(curr.right);

                }

            }

        }

    }

    public static void main(String args[]){

        binary t=new binary();

        int nodes[]={1, 2, 4, -1, -1, 5, -1, -1, 3, -1, 6, -1, -1};

        Node root=t.build(nodes);

// tree tree=new tree();

    // tree.root=new Node(5);

    // tree.root.left=new Node(2);

    // tree.root.right=new Node(6);

    // tree.root.right.left=new Node(7);

    // tree.root.right.left.right=new Node(8);

    // tree.print(tree.root);

        System.out.println(root.data);

        t.inorder(root);

        System.out.println();

        t.preorder(root);

        System.out.println();

        t.postorder(root);

        System.out.println();

        t.levelorder(root);

    }

}

# Graph implementation using adjacency list

import java.util.\*;

class graph{

    int v;

    LinkedList<Integer> adj[];

    graph(int v){

        this.v=v;

        adj=new LinkedList[v];

        for(int i=0;i<v;i++){

            adj[i]=new LinkedList<>();

        }

    }

    void addEdge(int s,int d){

        adj[s].add(d);

        adj[d].add(s);

    }

    void bfs(int s){

        Queue<Integer> q=new LinkedList<>();

        boolean vis[]=new boolean[v];

        q.add(s);

        vis[s]=true;

        while(!q.isEmpty()){

            s=q.poll();

            System.out.print(s+" ");

            for(int i:adj[s]){

                if(!vis[i]){

                    q.add(i);

                    vis[i]=true;

                }

            }

        }

    }

}

public class graph\_using\_adjList {

    public static void main(String args[]){

        graph g=new graph(4);

        g.addEdge(0,1);

        g.addEdge(0,2);

        g.addEdge(1,2);

        g.addEdge(2,3);

        g.bfs(0);

    }

}

# Graph implementation using adjacency matrix

public class graph\_adj\_matrix {

    static int vertices;

    int matrix[][];

    graph\_adj\_matrix(int v){

        this.vertices=v;

        matrix=new int[v][v];

    }

    void addEdge(int s,int d){

        matrix[s][d]=1;

        matrix[d][s]=1;

    }

    void printgraph(){

        boolean vis[]=new boolean[vertices];

        for(int i=0;i<vertices;i++){

            for(int j=0;j<vertices;j++){

                if(matrix[i][j]==1 && matrix[j][i]==1 && !vis[i]){

                    vis[i]=true;

                    System.out.print(i+" ");

                }

            }

            System.out.println();

        }

    }

    public static void main(String args[]){

        graph\_adj\_matrix g=new graph\_adj\_matrix(4);

        g.addEdge(0,1);

        g.addEdge(0,2);

        g.addEdge(1,2);

        g.addEdge(2,3);

        g.printgraph();

    }

}