**Binary Search Tree Operations and Views:  
CODE:**

import java.util.\*;

class Node {

    int key;

    Node left, right;

    public Node(int item) {

        key = item;

        left = right = null;

    }

}

class Pair{

    int hd;

    Node node;

    Pair(Node node,int hd){

        this.node=node;

        this.hd=hd;

    }

}

class BST {

    static Node insert(Node root, int key) {

        if (root == null) return new Node(key);

        if (key < root.key) root.left = insert(root.left, key);

        else if (key > root.key) root.right = insert(root.right, key);

        return root;

    }

    static Node search(Node root, int key) {

        if (root == null || root.key == key) return root;

        return key < root.key ? search(root.left, key) : search(root.right, key);

    }

    static void inorder(Node root) {

        if (root != null) {

            inorder(root.left);

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

            inorder(root.right);

        }

    }

    static Node delete(Node root, int key) {

        if (root == null) return root;

        if (key < root.key) root.left = delete(root.left, key);

        else if (key > root.key) root.right = delete(root.right, key);

        else {

            if (root.left == null) return root.right;

            else if (root.right == null) return root.left;

            Node succ = findMin(root.right);

            root.key = succ.key;

            root.right = delete(root.right, succ.key);

        }

        return root;

    }

    static boolean validateBST(Node root) {

        return helper(root, Integer.MIN\_VALUE, Integer.MAX\_VALUE);

    }

    private static boolean helper(Node root, int min, int max) {

        if (root == null) return true;

        if (root.key <= min || root.key >= max) return false;

        return helper(root.left, min, root.key) && helper(root.right, root.key, max);

    }

    private static Node findMin(Node node) {

        while (node.left != null) node = node.left;

        return node;

    }

    public static List<Integer>  rightView(Node root){

        List<Integer> r= new ArrayList<>();

        if(root==null){

            return r;

        }

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

        q.offer(root);

        while(!q.isEmpty()){

            int l=q.size();

            for(int i=0;i<l;i++){

                Node curr=q.poll();

                if(i==l-1){

                    r.add(curr.key);

                }

                if(curr.left!=null){

                    q.offer(curr.left);

                }

                if(curr.right!=null){

                    q.offer(curr.right);

                }

            }

        }

        return r;

    }

    public static List<Integer>  leftView(Node root){

        List<Integer> l= new ArrayList<>();

        if(root==null){

            return l;

        }

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

        q.offer(root);

        while(!q.isEmpty()){

            int n=q.size();

            for(int i=0;i<n;i++){

                Node curr=q.poll();

                if(i==0){

                    l.add(curr.key);

                }

                if(curr.left!=null){

                    q.offer(curr.left);

                }

                if(curr.right!=null){

                    q.offer(curr.right);

                }

            }

        }

        return l;

    }

    public static List<Integer>  topView(Node root){

        List<Integer> top=new ArrayList<>();

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

        Map<Integer,Integer> map=new TreeMap<>();

        if(root==null){return top;}

        q.offer(new Pair(root,0));

        while(!q.isEmpty()){

            Pair curr=q.poll();

            Node node=curr.node;

            int hd=curr.hd;

            if(!map.containsKey(curr.hd)){

                map.put(hd,node.key);

            }

            if(node.left!=null){

                q.offer(new Pair(node.left,hd-1));

            }

            if(node.right!=null){

                q.offer(new Pair(node.right,hd+1));

            }

        }

        for(int key:map.keySet()){

            top.add(map.get(key));

        }

        return top;

    }

    public static List<Integer>  bottomView(Node root){

        List<Integer> bot=new ArrayList<>();

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

        Map<Integer,Integer> map=new TreeMap<>();

        if(root==null){return bot;}

        q.offer(new Pair(root,0));

        while(!q.isEmpty()){

            Pair curr=q.poll();

            Node node=curr.node;

            int hd=curr.hd;

            map.put(hd,node.key);

            if(node.left!=null){

                q.offer(new Pair(node.left,hd-1));

            }

            if(node.right!=null){

                q.offer(new Pair(node.right,hd+1));

            }

        }

        for(int key:map.keySet()){

            bot.add(map.get(key));

        }

        return bot;

    }

    public static void main(String args[]) {

        Node root = null;

        root = insert(root, 50);

        root = insert(root, 30);

        root = insert(root, 20);

        root = insert(root, 40);

        root = insert(root, 70);

        root = insert(root, 60);

        root = insert(root, 80);

        System.out.println("Inorder traversal:");

        inorder(root);

        System.out.println();

        System.out.println(search(root, 60) != null ? "Found" : "Not Found");

        root = delete(root, 70);

        System.out.println("Inorder traversal after deletion:");

        inorder(root);

        System.out.println();

        System.out.println("Is the tree a valid BST? " + validateBST(root));

        System.out.println("Right View : "+rightView(root));

        System.out.println("Left View : "+leftView(root));

        System.out.println("Top View : "+topView(root));

        System.out.println("Bottom View : "+bottomView(root));

    }

}

**OUTPUT:**

