Binary Search Tree - Hands On

1. Check Binary Tree is BST or not

```
package usr.hands_on;

import java.util.*;

public class Qn_1 {
    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the Values of the list: ");
        String str = sc.nextLine();
        ArrayList<Integer> list1 = new ArrayList<>();
        for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) >= '0' && str.charAt(i) <= '9') {
                list.add(str.charAt(i) - '0');
                list1.add(str.charAt(i) - '0');
            }
        }
        Collections.sort(list);
        boolean flag = false;
        for (int i = 0; i < list.size(); i++) {
            if (!Objects.equals(list.get(i), list1.get(i))) {
                System.out.println("It is not a BST");
                flag = true;
                break;
            }
        }
        if (!flag) {
            System.out.println("It is a BST");
        }
    }
}
```

Output

```
Enter the Values of the list: 1 2 3 4 5
It is a BST
```

---

2. Insert into a Binary Search Tree

```
package usr.hands_on;

import usr.collections.BST;

import java.util.Scanner;

public class Qn_2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BST<Integer> tree = new BST<>();
        System.out.print("Enter the value of the tree: ");
        String str = sc.nextLine().trim();
        String[] arr = str.split(" ");
        for (String s : arr) {
            tree.insert(Integer.parseInt(s));
        }
        tree.display();

        System.out.print("Enter the value need to be inserted: ");
        int val = sc.nextInt();
        tree.insert(val);
        System.out.println();
        tree.display();
    }
}
```

Output

```
Enter the value of the tree: 1 2 3 4 5 6
Binary Search Tree:
                               -> 6
                         -> 5
                   -> 4
             -> 3
       -> 2
 -> 1

Enter the value need to be inserted: 3

Binary Search Tree:
                               -> 6
                         -> 5
                   -> 4
                         -> 3
             -> 3
       -> 2
 -> 1
```

---

3. Delete Node in a BST

```
package usr.hands_on;

import usr.collections.BST;

import java.util.Scanner;

public class Qn_3 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BST<Integer> tree = new BST<>();
        System.out.println("Enter the val of the tree: ");
        String str = sc.nextLine().trim();
        String[] arr = str.split(" ");
        for (String s : arr) {
            tree.insert(Integer.parseInt(s));
        }
        tree.display();
        System.out.println("Enter the value need to be removed: ");
        int val = sc.nextInt();
        tree.delete(val);
        tree.display();
    }
}
```

Output:

```
Enter the val of the tree:
1 2 3 4 5 6 7
Binary Search Tree:
                                     -> 7
                               -> 6
                         -> 5
                   -> 4
             -> 3
       -> 2
 -> 1

Enter the value need to be removed:
5
Binary Search Tree:
                               -> 7
                         -> 6
                   -> 4
             -> 3
       -> 2
 -> 1
```

4. BST Traversals

```java
package usr.hands_on;

import usr.collections.BST;

import java.util.Scanner;

public class Qn_4 {
    public static void main(String[] args) {
        BST<Integer> bst = new BST<>();
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the value of the tree: ");
        String str = sc.nextLine();
        String[] arr = str.split(" ");
        for (String s : arr) {
            bst.insert(Integer.parseInt(s));
        }


        bst.display();
        System.out.print("
In-Order: ");
        bst.getInOrder();
        System.out.print("
Pre-Order: ");
        bst.getPreOrder();
        System.out.print("
Post-Order: ");
        bst.getPostOrder();

    }
}
```

Output:

```
Enter the value of the tree: 10 20 30 40 50 60 70 80 90
Binary Search Tree:
                                                 -> 90
                                           -> 80
                                     -> 70
                               -> 60
                         -> 50
                   -> 40
             -> 30
       -> 20
 -> 10


In-Order: 10, 20, 30, 40, 50, 60, 70, 80, 90,

Pre-Order: 10, 20, 30, 40, 50, 60, 70, 80, 90,

Post-Order: 90, 80, 70, 60, 50, 40, 30, 20, 10,
```

5. Search in a BST

```java
package usr.hands_on;

import usr.collections.BST;

import java.util.Scanner;

public class Qn_5 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BST<Integer> bst = new BST<Integer>();
        System.out.println("Enter the values of the tree: ");
        String str = sc.nextLine();
        String[] strs = str.split(" ");
        for (String str1 : strs) {
            bst.insert(Integer.parseInt(str1));
        }
        bst.display();
        System.out.println("Enter the value to be searched: ");
        int search = sc.nextInt();
        if (!bst.search(search)){
            System.out.println("Element not found");
        }

    }
}
```

Output

```
Enter the values of the tree:
50 20 70 60 80 10 30
Binary Search Tree:
             -> 80
       -> 70
             -> 60
 -> 50
             -> 30
       -> 20
             -> 10
```

```
Enter the value to be searched:
70
       -> 80
 -> 70
       -> 60
```

6. Kth largest element in BST

```java
package usr.hands_on;

import usr.collections.BST;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Qn_6 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BST<Integer> bst = new BST<Integer>();
        System.out.println("Enter the values of the tree: ");
        String str = sc.nextLine();
        String[] strs = str.split(" ");
        for (String str1 : strs) {
            bst.insert(Integer.parseInt(str1));
        }
        System.out.println("Enter the k position: ");
        int k = sc.nextInt();
        List<Integer> inorder = new ArrayList<Integer>();
        bst.inorderTraversal(bst.getRoot(),inorder);
        int len = inorder.size();
        System.out.println("The Kth Largest element in tree: " + inorder.get(len-k));
    }
}
```

Output:

```
Enter the values of the tree:
50 20 70 60 80 10 30
Enter the k position:
4
The Kth Largest element in tree: 50
```

7. Minimum element in BST

```java
package usr.hands_on;

import usr.collections.BST;

import java.util.Scanner;

public class Qn_7 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BST<Integer> bst = new BST<Integer>();
        System.out.println("Enter the values of the tree: ");
        String str = sc.nextLine();
        String[] strs = str.split(" ");
        for (String str1 : strs) {
            bst.insert(Integer.parseInt(str1));
        }
        bst.display();

        System.out.println("The Minimum values in BST are: " + bst.getMin());
    }
}
```

Output:

```
Enter the values of the tree:
50 20 70 60 80 10 30
Binary Search Tree:
             -> 80
       -> 70
             -> 60
 -> 50
             -> 30
       -> 20
             -> 10

The Minimum values in BST are: 10
```

---

8. Median of BST

```
package usr.hands_on;

import usr.collections.BST;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Qn_8 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        BST<Integer> bst = new BST<Integer>();
        System.out.println("Enter the values of the tree: ");
        String str = sc.nextLine();
        String[] strs = str.split(" ");
        for (String str1 : strs) {
            bst.insert(Integer.parseInt(str1));
        }
        List<Integer> inorder = new ArrayList<Integer>();
        bst.inorderTraversal(bst.getRoot(),inorder);
        int len = inorder.size();
        int median;
        if (len%2 == 0) {
            median = (inorder.get(len/2)+inorder.get((len/2)-1))/2;
        } else {
            median = inorder.get(len/2);
        }
        System.out.println("Median: " + median);
    }
}
```

Output:

```
Enter the values of the tree:
50 20 70 60 80 10 30
Median: 50
```

9. Normal BST to Balanced BST

```java
package usr.hands_on;

import usr.collections.BST;

import java.util.Scanner;

public class Qn_9 {
    public static void main(String[] args) {
        BST<Integer> bst = new BST<Integer>();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the values of the tree: ");
        String str = sc.nextLine();
        String[] strArr = str.split(" ");
        for (String s: strArr) {
            bst.insert(Integer.parseInt(s));
        }
        System.out.print("Before Balancing ");
        bst.display();
        System.out.println("
After Balancing ");
        bst = bst.balanceBST(bst.getRoot());
        bst.display();
    }
}
```

Output:

```
Enter the values of the tree:
10 20 30 40 50 60 70 80 90 100
Before Balancing Binary Search Tree:
                                                       -> 100
                                                 -> 90
                                           -> 80
                                     -> 70
                               -> 60
                         -> 50
                   -> 40
             -> 30
       -> 20
 -> 10


After Balancing
Binary Search Tree:
                   -> 100
             -> 90
       -> 80
                   -> 70
             -> 60
 -> 50
                   -> 40
             -> 30
       -> 20
             -> 10
```

BST Implementation

```
package usr.collections;

import java.util.ArrayList;
import java.util.List;

class BSTNode<E extends Comparable<E>>{
    E data;
    BSTNode<E> left;
    BSTNode<E> right;

    public BSTNode(E data) {
        this.data = data;
        left = null;
        right = null;
    }


    public BSTNode(E data,BSTNode<E> left, BSTNode<E> right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }
}

public class BST<E extends Comparable<E>> {
    private BSTNode<E> _root;
    public BST() {
        _root = null;
    }

    public BST(E data, BSTNode<E> left, BSTNode<E> right) {
        _root = new BSTNode<E>(data, left, right);
    }

    public BST(BSTNode<E> root) {
        _root = root;
    }

    public void insert(E data) {
        BSTNode<E> newNode = new BSTNode<E>(data);
        if (_root == null) {
            _root = newNode;
        } else {
            BSTNode<E> current = _root;
            while (true) {
                if (current.data.compareTo(data) > 0) {
                    if (current.left == null) {
                        current.left = newNode;
                        break;
                    }
                    current = current.left;
                } else {
                    if (current.right == null) {
                        current.right = newNode;
                        break;
                    }
                    current = current.right;
                }
            }
        }
    }



    public void delete(E data) {
        this._root = _deleteNode(this._root, data);
    }

    private BSTNode<E> _deleteNode(BSTNode<E> root, E data) {
        if (root == null) {
            return null;
        }
        int compare = data.compareTo(root.data);
        if (compare < 0) {
            root.left = this._deleteNode(root.left, data);
        } else if (compare > 0) {
            root.right = this._deleteNode(root.right, data);
        } else {
            if (root.left == null) {
                return root.right;
            } else if (root.right == null) {
                return root.left;
            }
            BSTNode<E> minNodeForRight = minimumElement(root.right);
            root.data = minNodeForRight.data;
            root.right = _deleteNode(root.right, minNodeForRight.data);
        }
        return root;
    }

    private BSTNode<E> minimumElement(BSTNode<E> root) {
        if (root.left == null)
            return root;
        else {
            return minimumElement(root.left);
        }
    }

    public E getMin(){
        BSTNode<E> current = _root;
        while (current.left != null) {
            current = current.left;
        }
        return current.data;
    }

    public void display() {
        BSTNode<E> current = _root;
        System.out.println("Binary Search Tree: ");
        this._printTree(current, 0);
        System.out.println();
    }

    private void _printTree(BSTNode<E> node, int level) {
        if (node == null) {
            return;
        } else {
            _printTree(node.right, level + 1);
            for(int i=0; i<level; i++) {
                System.out.print("      ");
            }
            System.out.println(" -> " + node.data);
            _printTree(node.left, level + 1);
        }
    }

    public void getInOrder() {
        _getInOrder(this._root);
        System.out.println();
    }

    private void _getInOrder(BSTNode<E> node) {
        if (node == null) {
            return;
        }
        _getInOrder(node.left);
        System.out.print(node.data + ", ");
        _getInOrder(node.right);
    }

    public void getPreOrder() {
        _getPreOrder(this._root);
        System.out.println();
    }

    private void _getPreOrder(BSTNode<E> node) {
        if (node == null) {
            return;
        }
        System.out.print(node.data + ", ");
        _getPreOrder(node.left);
        _getPreOrder(node.right);
    }

    public void getPostOrder() {
        _getPostOrder(this._root);
        System.out.println();

    }

    private void _getPostOrder(BSTNode<E> node) {
        if (node == null) {
            return;
        }
        _getPostOrder(node.left);
        _getPostOrder(node.right);
        System.out.print(node.data + ", ");
    }

    public boolean contains(E data) {
        if (this._root == null) {
            return false;
        } else {
            BSTNode<E> current = _root;
            while (current != null) {
                if (current.data.compareTo(data) == 0) {
                    return true;
                } else if (current.data.compareTo(data) > 0) {
                    current = current.left;
                } else {
                    current = current.right;
                }
            }
            return false;
        }
    }

    public BST<E> balanceBST(BSTNode<E> root) {
        List<E> inorder = new ArrayList<>();
        inorderTraversal(_root, inorder);

        BSTNode<E> temp = createBalancedBST(inorder, 0, inorder.size() - 1);
        return new BST<>(temp);
    }
    public void inorderTraversal(BSTNode<E> root, List<E> inorder) {
        if (root == null) return;
        inorderTraversal(root.left, inorder);
        inorder.add(root.data);
        inorderTraversal(root.right, inorder);
    }

    private BSTNode<E> createBalancedBST(
            List<E> inorder,
            int start,
            int end
    ) {
        if (start > end) return null;
        int mid = start + (end - start) / 2;
        BSTNode<E> leftSubtree = createBalancedBST(inorder, start, mid - 1);
        BSTNode<E> rightSubtree = createBalancedBST(inorder, mid + 1, end);
        return new BSTNode<>(
                inorder.get(mid),
                leftSubtree,
                rightSubtree
        );
    }
    public BSTNode<E> getRoot(){
        return this._root;
    }

    public boolean search(E data){
        if (this._root == null) {
            return false;
        } else {
            BSTNode<E> current = _root;
            while (current != null) {
                if (current.data.compareTo(data) == 0) {
                    this._printTree(current, 0);
                    return true;
                } else if (current.data.compareTo(data) > 0) {
                    current = current.left;
                } else {
                    current = current.right;
                }
            }
        }
        return false;
    }

}
```