Name: Vijay Misal
Div: C Batch: C3
Roll No: 233073
PRN No: 22320079

Practical No: 3

Title: Construct an inorder threaded binary search tree. Traverse threaded binary tree it in inorder and preorder.

Code:

```java
import java.util.LinkedList;
import java.util.Queue;

class Node {
    Node left, right;
    int info;
    boolean lthread, rthread;

    public Node(int key) {
        this.info = key;
        this.left = null;
        this.right = null;
        this.lthread = true;
        this.rthread = true;
    }
}

public class ThreadedBST {
    private Node root;

    private Node insert(Node root, int key) {
        Node ptr = root;
        Node par = null;

        while (ptr != null) {
            if (key == ptr.info) {
                System.out.println("Duplicate Key !");
                return root;
            }
            par = ptr;
```

```
            if (key < ptr.info) {
                if (!ptr.lthread)
                    ptr = ptr.left;
                else
                    break;
            } else {
                if (!ptr.rthread)
                    ptr = ptr.right;
                else
                    break;
            }
        }

        Node tmp = new Node(key);
        tmp.lthread = true;
        tmp.rthread = true;

        if (par == null) {
            root = tmp;
            tmp.left = null;
            tmp.right = null;
        } else if (key < par.info) {
            tmp.left = par.left;
            tmp.right = par;
            par.lthread = false;
            par.left = tmp;
        } else {
            tmp.left = par;
            tmp.right = par.right;
            par.rthread = false;
            par.right = tmp;
        }

        return root;
    }

    private Node inorderSuccessor(Node ptr) {
        if (ptr.rthread)
            return ptr.right;

        ptr = ptr.right;
        while (!ptr.lthread)
```

```
            ptr = ptr.left;

        return ptr;
    }

    private void inorder(Node root) {
        if (root == null) {
            System.out.println("Tree is empty");
            return;
        }

        Node ptr = root;
        while (!ptr.lthread)
            ptr = ptr.left;

        while (ptr != null) {
            System.out.print(ptr.info + " ");
            ptr = inorderSuccessor(ptr);
        }
    }

    private void preorder(Node root) {
        if (root == null) {
            System.out.println("Tree is empty");
            return;
        } else {
            while (root != null) {
                System.out.print(root.info + " ");
                if (!root.lthread)
                    root = root.left;
                else if (!root.rthread)
                    root = root.right;
                else {
                    while (root != null && root.rthread) {
                        root = inorderSuccessor(root);
                    }
                    if (root != null) {
                        root = root.right;
                    }
                }
            }
        }
```

```
    }

    private void levelOrderTraversal(Node root) {
        if (root == null)
            return;

        Queue<Node> q = new LinkedList<>();
        q.add(root);
        q.add(null);

        while (!q.isEmpty()) {
            Node current = q.poll();

            if (current == null) {
                System.out.println();
                if (!q.isEmpty())
                    q.add(null);
            } else {
                System.out.print(current.info + " ");

                if (current.left != null && !current.lthread)
                    q.add(current.left);
                if (current.right != null && !current.rthread)
                    q.add(current.right);
            }
        }
    }

    public static void main(String[] args) {
        ThreadedBST threadedBST = new ThreadedBST();

        threadedBST.root = threadedBST.insert(threadedBST.root, 20);
        threadedBST.root = threadedBST.insert(threadedBST.root, 10);
        threadedBST.root = threadedBST.insert(threadedBST.root, 30);
        threadedBST.root = threadedBST.insert(threadedBST.root, 5);
        threadedBST.root = threadedBST.insert(threadedBST.root, 16);
        threadedBST.root = threadedBST.insert(threadedBST.root, 14);
        threadedBST.root = threadedBST.insert(threadedBST.root, 17);
        threadedBST.root = threadedBST.insert(threadedBST.root, 13);

        System.out.print("Inorder Traversal: ");
        threadedBST.inorder(threadedBST.root);
```

```
        System.out.println();

        System.out.print("Preorder Traversal: ");
        threadedBST.preorder(threadedBST.root);
        System.out.println();

        System.out.print("Level-wise Display: ");
        threadedBST.levelOrderTraversal(threadedBST.root);
    }
}
```

Output:

Inorder Traversal: 5 10 13 14 16 17 20 30

Preorder Traversal: 20 10 5 16 14 13 17 30

Level-wise Display: 20

10 30

5 16

14 17

13