CSE3004 (DAA) Lab-7

KHAN MOHD OWAIS RAZA
20BCD7138

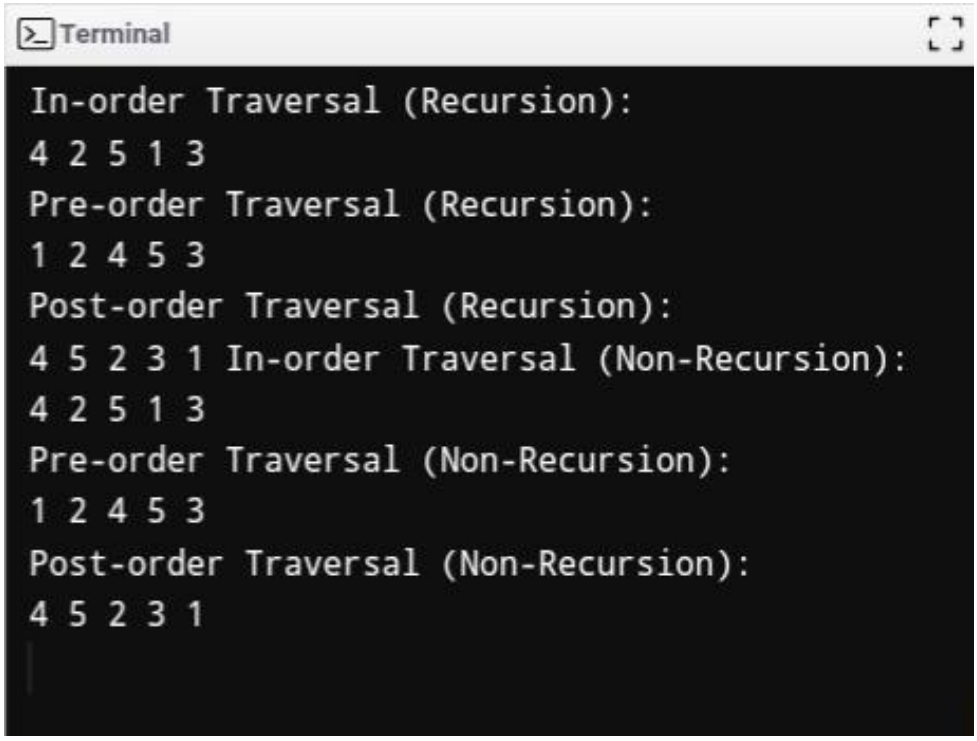
Write a Java program to implement Binary Tree Traversals (In-order, Pre-order, and Post-order) using both recursion and non-recursion.

```
import java.util.Stack;
class Node {
    int data;
    Node left, right;
    public Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}
public class BinaryTree {
    Node root;
    public BinaryTree() {
        root = null;
    }
    // In-order Traversal using recursion
    public void inorderRecursion(Node node) {
        if (node == null) {
            return;
        }
        inorderRecursion(node.left);
        System.out.print(node.data + " ");
        inorderRecursion(node.right);
    }
    // Pre-order Traversal using recursion
    public void preorderRecursion(Node node) {
        if (node == null) {
            return;
```

```
    }
    System.out.print(node.data + " ");
    preorderRecursion(node.left);
    preorderRecursion(node.right);
}
// Post-order Traversal using recursion
public void postorderRecursion(Node node) {
    if (node == null) {
        return;
    }
    postorderRecursion(node.left);
    postorderRecursion(node.right);
    System.out.print(node.data + " ");
}
// In-order Traversal without recursion
public void inorderNonRecursion() {
    if (root == null) {
        return;
    }
    Stack<Node> stack = new Stack<Node>();
    Node node = root;
    while (!stack.isEmpty() || node != null) {
        if (node != null) {
            stack.push(node);
            node = node.left;
        } else {
            node = stack.pop();
            System.out.print(node.data + " ");
            node = node.right;
        }
    }
}
// Pre-order Traversal without recursion
public void preorderNonRecursion() {
    if (root == null) {
        return;
    }
```

```
    Stack<Node> stack = new Stack<Node>();
    stack.push(root);
    while (!stack.isEmpty()) {
        Node node = stack.pop();
        System.out.print(node.data + " ");
        if (node.right != null) {
            stack.push(node.right);
        }
        if (node.left != null) {
            stack.push(node.left);
        }
    }
}
// Post-order Traversal without recursion
public void postorderNonRecursion() {
    if (root == null) {
        return;
    }
    Stack<Node> stack1 = new Stack<Node>();
    Stack<Node> stack2 = new Stack<Node>();
    stack1.push(root);
    while (!stack1.isEmpty()) {
        Node node = stack1.pop();
        stack2.push(node);
        if (node.left != null) {
            stack1.push(node.left);
        }
        if (node.right != null) {
            stack1.push(node.right);
        }
    }
    while (!stack2.isEmpty()) {
        System.out.print(stack2.pop().data + " ");
    }
}
public static void main(String[] args) {
    BinaryTree tree = new BinaryTree();
```

```
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);
        System.out.println("In-order Traversal (Recursion):");
        tree.inorderRecursion(tree.root);
        System.out.println();
        System.out.println("Pre-order Traversal (Recursion):");
        tree.preorderRecursion(tree.root);
        System.out.println();
        System.out.println("Post-order Traversal (Recursion):");
        tree.postorderRecursion(tree.root);
        System.out.println();
        System.out.println("In-order Traversal (Non-Recursion):");
        tree.inorderNonRecursion();
        System.out.println();
        System.out.println("Pre-order Traversal (Non-Recursion):");
        tree.preorderNonRecursion();
        System.out.println();
        System.out.println("Post-order Traversal (Non-Recursion):");
        tree.postorderNonRecursion();
        System.out.println();
}}
```


Terminal
In-order Traversal (Recursion):
4 2 5 1 3
Pre-order Traversal (Recursion):
1 2 4 5 3
Post-order Traversal (Recursion):
4 5 2 3 1 In-order Traversal (Non-Recursion):
4 2 5 1 3
Pre-order Traversal (Non-Recursion):
1 2 4 5 3
Post-order Traversal (Non-Recursion):
4 5 2 3 1