树前中后序遍历

```java
package tree;

import java.util.Stack;

public class PreInPosTraversal {

    public static class Node {
        public int value;
        public Node left;
        public Node right;

        public Node(int data) {
            this.value = data;
        }
    }

    public static void preOrderRecur(Node head) {
        if (head == null) {
            return;
        }
        System.out.print(head.value + " ");
        preOrderRecur(head.left);
        preOrderRecur(head.right);
    }

    public static void inOrderRecur(Node head) {
        if (head == null) {
            return;
        }
        inOrderRecur(head.left);
        System.out.print(head.value + " ");
        inOrderRecur(head.right);
    }

    public static void posOrderRecur(Node head) {
        if (head == null) {
            return;
        }
        posOrderRecur(head.left);
        posOrderRecur(head.right);
        System.out.print(head.value + " ");
    }

    public static void preOrderUnRecur(Node head) {
        System.out.print("pre-order: ");
        if (head != null) {
            Stack<Node> stack = new Stack<Node>();
            stack.add(head);
            while (!stack.isEmpty()) {
                head = stack.pop();
                System.out.print(head.value + " ");
                if (head.right != null) {
                    stack.push(head.right);
                }
                if (head.left != null) {
                    stack.push(head.left);
                }
            }
        }
        System.out.println();
    }

    public static void inOrderUnRecur(Node head) {
        System.out.print("in-order: ");
        if (head != null) {
            Stack<Node> stack = new Stack<Node>();
            while (!stack.isEmpty() || head != null) {
                if (head != null) {
                    stack.push(head);
                    head = head.left;
                } else {
                    head = stack.pop();
                    System.out.print(head.value + " ");
                    head = head.right;
                }
            }
        }
        System.out.println();
    }

    public static void posOrderUnRecur1(Node head) {
        System.out.print("pos-order: ");
        if (head != null) {
            Stack<Node> s1 = new Stack<Node>();
            Stack<Node> s2 = new Stack<Node>();
            s1.push(head);
            while (!s1.isEmpty()) {
                head = s1.pop();
                s2.push(head);
                if (head.left != null) {
                    s1.push(head.left);
                }
                if (head.right != null) {
                    s1.push(head.right);
                }
            }
            while (!s2.isEmpty()) {
                System.out.print(s2.pop().value + " ");
            }
        }
        System.out.println();
    }

    public static void posOrderUnRecur2(Node h) {
        System.out.print("pos-order: ");
        if (h != null) {
            Stack<Node> stack = new Stack<Node>();
            stack.push(h);
            Node c = null;
            while (!stack.isEmpty()) {
                c = stack.peek();
                if (c.left != null && h != c.left && h != c.right) {
                    stack.push(c.left);
                } else if (c.right != null && h != c.right) {
                    stack.push(c.right);
                } else {
                    System.out.print(stack.pop().value + " ");
                    h = c;
                }
            }
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Node head = new Node(5);
        head.left = new Node(3);
        head.right = new Node(8);
        head.left.left = new Node(2);
        head.left.right = new Node(4);
        head.left.left.left = new Node(1);
        head.right.left = new Node(7);
        head.right.left.left = new Node(6);
        head.right.right = new Node(10);
        head.right.right.left = new Node(9);
        head.right.right.right = new Node(11);

        // recursive
        System.out.println("==============recursive==============");
        System.out.print("pre-order: ");
        preOrderRecur(head);
        System.out.println();
        System.out.print("in-order: ");
        inOrderRecur(head);
        System.out.println();
        System.out.print("pos-order: ");
        posOrderRecur(head);
        System.out.println();

        // unrecursive
        System.out.println("============unrecursive=============");
        preOrderUnRecur(head);
        inOrderUnRecur(head);
        posOrderUnRecur1(head);
        posOrderUnRecur2(head);

    }

}
```