Program for Sum Root To Leaf Numbers

```
class TreeNodesum {
    int val;
    TreeNodesum left;
    TreeNodesum right;
    TreeNodesum(int val) {
        this.val = val;
    }
}

public class SumRootToLeafNumbers {
    public int sumRootNumbers(TreeNodesum root) {
        return sumNumbers(root, 0);
    }

    public int sumNumbers(TreeNodesum node, int sum) {
        if (node == null) {
            return 0;
        }
        // Calculate the new currentSum by appending the current node's value
        sum = sum * 10 + node.val;

        // If it's a leaf node, return the currentSum
        if (node.left == null && node.right == null) {
            return sum;
        }

        // Recursively calculate the sum for left and right subtrees
        int leftSum = sumNumbers(node.left, sum);
        int rightSum = sumNumbers(node.right, sum);

        return leftSum + rightSum;
    }

    public static void main(String[] args) {
        // Create a sample binary tree
        TreeNodesum root = new TreeNodesum(4);
        root.left = new TreeNodesum(9);
        root.right = new TreeNodesum(0);
        root.left.left= new TreeNodesum(5);
        root.left.right=new TreeNodesum(1);
        SumRootToLeafNumbers obj = new SumRootToLeafNumbers();
        int sum = obj.sumRootNumbers(root);
        System.out.println("Sum numbers: " + sum);
    }
}
```

OUTPUT

Sum numbers: 1026

Program for Evaluate Reverse Polish Notation

```
import java.util.Stack;

public class EvaluateReversePolishNotation {

    public static int evalRPN(String[] tokens) {
        Stack<Integer> stack = new Stack();
        String operators = "+-*/";

        for (String str : tokens) {

            if (operators.contains(str) && !stack.isEmpty()) {
                int op1 = stack.pop();
                int op2 = stack.pop();

                int ans = help(op2, str, op1); // Calculate the result
                stack.push(ans); // Push the result back onto the stack
            } else {
               stack.push(Integer.parseInt(str)); // Parse the token as an
integer and push it onto the stack
            }
        }
       return stack.pop(); //The final result is the top element of the stack
    }

    public static int help(int op2, String str, int op1) {
        if (str.equals("+")) {
            return op2 + op1; // Addition
        } else if (str.equals("-")) {
            return op2 - op1; // Subtraction
        } else if (str.equals("/")) {
            return op2 / op1; // Division
        }
        return op2 * op1; // Multiplication
    }

public static void main(String[] args) {
 String[] tokens = {"10","6","9","3","+","-11","*","/","*","17","+","5","+"};
 int result = evalRPN(tokens);
 System.out.println("Output: " + result);
}

}
```

OUTPUT

Output: 22