Software Tools 4

Assignment 4

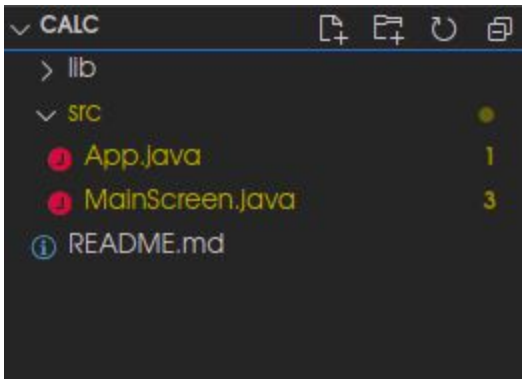
Krunal Rank
U18CO081

Write a java event handling program to create a scientific calculator.
NOTES :

- Calculator can take input from the keyboard as well.
- Add validations
  - e.g. for fractional number only one dot is allowed.
- Screenshot given below is for reference GUI .

Answer:

Directory Structure:



App.java:

```java
public class App {
    public static void main(String[] args) throws Exception {
        MainScreen main = new MainScreen();
    }
}
```

MainScreen.java:

```java
import javax.swing.*;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
```

```
import java.util.ArrayList;
import java.util.Stack;

public class MainScreen {

   private static class Symbol {
       int balancer;
       String id;
       double val = 0.0;
       String print = "";
       double multiplier = 1;

       public Symbol(String i, int b, double v, String p, double m) {
           this.id = i;
           this.balancer = b;
           this.val = v;
           this.print = p;
           this.multiplier = m;
       }

       public void debug() {
           System.out.println("DEBUG " + this.id + " " + this.balancer + " " +
this.val + " " + this.print + " "
                   + this.multiplier);
       }
   }

   private static class Literal {
       double val;
       String id, op;

       public Literal(double v, String i, String o) {
           this.val = v;
           this.id = i;
           this.op = o;
       }

       public void debug() {
           System.out.println("DEBUG " + this.id + " " + this.val + " " + this.op);
       }

   }

   private class CustomDispatcher implements KeyEventDispatcher {
```

```
        @Override
        public boolean dispatchKeyEvent(KeyEvent e) {
            if (e.getID() == KeyEvent.KEY_TYPED) {
                String key = "" + e.getKeyChar();
                if (isNum(key)) {
                    pressDigit(Integer.parseInt("" + e.getKeyChar()));
                } else if (isDot(key)) {
                    pressDot(".");
                } else if (isBinaryOp(key)) {
                    pressBinaryOp(key);
                } else if (isFactorialOp(key)) {
                    pressFactorial();
                } else if (isOpenBracket(key)) {
                    pressOpenBracket();
                } else if (isCloseBracket(key)) {
                    pressCloseBracket();
                }
            }
            if (e.getID() == KeyEvent.KEY_RELEASED) {
                if (e.getKeyCode() == KeyEvent.VK_BACK_SPACE) {
                    DEL();
                } else if (e.getKeyCode() == KeyEvent.VK_ENTER) {
                    ANS();
                }
            }
            return false;
        }
    }

    private JFrame frame;
    private JPanel mainPanel, subPanel, btnPanel, numPanel, toolPanel, textPanel,
btnLeftPanel, btnRightPanel;

    private JTextField textField;

    private ArrayList<Symbol> stack;

    ArrayList<JButton> btns, numButtons, toolButtons;

    private String ans = "";

    private void renderInfo() {
        ans = "";
        stack.forEach((sym) -> {
```

```
            ans += sym.print;
        });
        if (ans.length() == 0)
            ans = "0";

        textField.setText(ans);
        return;
    }

    private boolean isNum(String id) {
        return id.equals("1") || id.equals("2") || id.equals("3") || id.equals("4") ||
id.equals("5") || id.equals("6")
                || id.equals("7") || id.equals("8") || id.equals("9") ||
id.equals("0");
    }

    private boolean isBinaryOp(String id) {
        return id.equals("-") || id.equals("+") || id.equals("/") || id.equals("*") ||
id.equals("^") || id.equals("P")
                || id.equals("C");
    }

    private boolean isTrigonometryOp(String id) {
        return id.equals("tan") || id.equals("sin") || id.equals("cos") ||
id.equals("atan") || id.equals("acos")
                || id.equals("asin") || id.equals("sinh") || id.equals("cosh") ||
id.equals("tanh");
    }

    private boolean isLogOp(String id) {
        return id.equals("log") || id.equals("log10");
    }

    private boolean isDot(String id) {
        return id.equals(".");
    }

    private boolean isFactorialOp(String id) {
        return id.equals("!");
    }

    private boolean isOpenBracket(String id) {
        return id.equals("(");
    }
```

```
   private boolean isCloseBracket(String id) {
       return id.equals(")");
   }

   private int getPrecedence(String id) {
       if (id.equals("(") || id.equals(")"))
           return -1;
       if (id.equals("+"))
           return 0;
       if (id.equals("*"))
           return 1;
       if (id.equals("P") || id.equals("C"))
           return 2;
       if (id.equals("/"))
           return 3;
       if (id.equals("^"))
           return 4;
       if (isTrigonometryOp(id) || isLogOp(id))
           return 5;
       if(isFactorialOp(id) || id.equals("-"))
           return 6;
       return 7;
   }

   private boolean isDouble(double val) {
       return Math.round(val) != val;
   }

   private double gamma(double z) {
       double g = 7;
       double[] C = { 0.99999999999980993, 676.5203681218851, -1259.1392167224028,
771.32342877765313,
               -176.61502916214059, 12.507343278686905, -0.13857109526572012,
9.9843695780195716e-6,
               1.5056327351493116e-7 };

       if (z < 0.5)
           return Math.PI / (Math.sin(Math.PI * z) * gamma(1 - z));
       else {
           z -= 1;

           var x = C[0];
           for (var i = 1; i < g + 2; i++)
```

```
            x += C[i] / (z + i);

        var t = z + g + 0.5;
        return Math.sqrt(2 * Math.PI) * Math.pow(t, (z + 0.5)) * Math.exp(-t) * x;
    }
}

private double factorial(double x) {
    return gamma(x + 1);
}

private void CLR() {
    stack.clear();
    renderInfo();
}

private void DEL() {
    if (stack.isEmpty()) {
        return;
    }
    Symbol top = stack.get(stack.size() - 1);
    stack.remove(stack.size() - 1);
    if (top.id == "(") {
        if (stack.isEmpty()) {
            renderInfo();
            return;
        }
        top = stack.get(stack.size() - 1);
        if (isTrigonometryOp(top.id) || isLogOp(top.id)) {
            stack.remove(stack.size() - 1);
        }

    }
    renderInfo();
}

private void ANS() {
    if (stack.isEmpty()) {
        renderInfo();
        return;
    }
    Symbol top = stack.get(stack.size() - 1);
    int extraBrackets = top.balancer;
    while (extraBrackets > 0) {
```

```
            stack.add(new Symbol(")", top.balancer - 1, -1, ")", -1));
            extraBrackets--;
        }
        ArrayList<Literal> arr = new ArrayList<Literal>();
        for (int i = 0; i < stack.size(); i++) {
            Symbol t = stack.get(i);
            if (isNum(t.id) || isDot(t.id)) {
                int j = i;
                double val = 0;
                while (isNum(t.id) || isDot(t.id)) {
                    val = t.val;
                    j++;
                    if (j == stack.size())
                        break;
                    t = stack.get(j);
                }
                arr.add(new Literal(val, "number", "X"));
                i = j - 1;
            } else if (isOpenBracket(t.id)) {
                arr.add(new Literal(-1, "openBracket", "X"));
            } else if (isCloseBracket(t.id)) {
                arr.add(new Literal(-1, "closeBracket", "X"));
            } else if (isBinaryOp(t.id)) {
                if(t.id.equals("-")) arr.add(new Literal(-1,"binaryOp","+"));
                    arr.add(new Literal(-1, "binaryOp", t.id));
            } else if (isFactorialOp(t.id) || isTrigonometryOp(t.id) || isLogOp(t.id))
{
                arr.add(new Literal(-1, "unaryOp", t.id));
            }
        }
        try {
            ArrayList<Literal> postfix = new ArrayList<Literal>();
            Stack<Literal> st = new Stack<Literal>();

            for (int i = 0; i < arr.size(); i++) {
                Literal l = arr.get(i);
                if (l.id.equals("number"))
                    postfix.add(l);
                else if (l.id.equals("binaryOp") || l.id.equals("unaryOp")) {
                    if (st.isEmpty()) {
                        st.push(l);
                    } else {
                        while (!st.isEmpty() && getPrecedence(st.peek().op) >=
getPrecedence(l.op)) {
```

```
                    postfix.add(st.peek());
                    st.pop();
                }
                st.push(l);
            }
        } else if (l.id.equals("openBracket")) {
            st.push(l);
        } else if (l.id.equals("closeBracket")) {
            while (!st.isEmpty() && !st.peek().id.equals("openBracket")) {
                postfix.add(st.peek());
                st.pop();
            }
            st.pop();
        }
    }
    while (!st.empty()) {
        postfix.add(st.peek());
        st.pop();
    }
    Stack<Double> finalStack = new Stack<Double>();
    for (int i = 0; i < postfix.size(); i++) {
        Literal l = postfix.get(i);
        if (l.id.equals("number")){
            finalStack.push(l.val);
            continue;
        }
        double op1, op2, val;

        switch (l.op) {
            case "+":
                op1 = finalStack.peek();
                finalStack.pop();
                if(finalStack.size()==0){
                    finalStack.push(op1);
                    break;
                }
                op2 = finalStack.peek();
                finalStack.pop();
                val = op2 + op1;
                finalStack.push(val);
                break;
            case "-":
                op1 = finalStack.peek();
                finalStack.pop();
```

```
                    val = 0- op1;
                    finalStack.push(val);
                    break;
                case "*":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    op2 = finalStack.peek();
                    finalStack.pop();
                    val = op2 * op1;
                    finalStack.push(val);
                    break;
                case "/":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    op2 = finalStack.peek();
                    finalStack.pop();
                    val = op2 / op1;
                    finalStack.push(val);
                    break;
                case "^":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    op2 = finalStack.peek();
                    finalStack.pop();
                    val = Math.pow(op2, op1);
                    finalStack.push(val);
                    break;
                case "P":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    op2 = finalStack.peek();
                    finalStack.pop();
                    val = ((op2 < 0 ? -1 : 1) * factorial(op2)) / ((op2 < op1 ? -1
: 1) * factorial(op2 - op1));
                    finalStack.push(val);
                    break;
                case "C":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    op2 = finalStack.peek();
                    finalStack.pop();
                    val = ((op2 < 0 ? -1 : 1) * factorial(op2)) / (((op2 < op1 ? -1
: 1) * factorial(op2 - op1))
                            * ((op1 < 0 ? -1 : 1) * factorial(op1)));
```

```
                    finalStack.push(val);
                    break;
                case "!":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    val = (op1 < 0 ? -1 : 1) * factorial(op1);
                    finalStack.push(val);
                    break;
                case "sin":
                    op1 = finalStack.peek();
                    System.out.println(op1);
                    finalStack.pop();
                    finalStack.push(Math.sin(op1));
                    break;
                case "cos":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    finalStack.push(Math.cos(op1));
                    break;
                case "tan":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    finalStack.push(Math.tan(op1));
                    break;
                case "asin":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    finalStack.push(Math.asin(op1));
                    break;
                case "acos":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    finalStack.push(Math.acos(op1));
                    break;
                case "atan":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    finalStack.push(Math.atan(op1));
                    break;
                case "sinh":
                    op1 = finalStack.peek();
                    finalStack.pop();
                    finalStack.push(Math.sinh(op1));
                    break;
```

```
                    case "cosh":
                        op1 = finalStack.peek();
                        finalStack.pop();
                        finalStack.push(Math.cosh(op1));
                        break;
                    case "tanh":
                        op1 = finalStack.peek();
                        finalStack.pop();
                        finalStack.push(Math.tanh(op1));
                        break;
                    case "log":
                        op1 = finalStack.peek();
                        finalStack.pop();
                        finalStack.push(Math.log(op1));
                        break;
                    case "log10":
                        op1 = finalStack.peek();
                        finalStack.pop();
                        finalStack.push(Math.log10(op1));
                        break;
                    default:
                }
            }
            stack.clear();
            double finalAns = finalStack.peek();
            if(finalAns==0) return;
            String s = String.format("%.4f", finalAns);
            for (int i = 0; i < s.length(); i++) {
                String t = "" + s.charAt(i);
                if (isNum(t))
                    pressDigit(Integer.parseInt(t));
                if (isDot(t))
                    pressDot(".");
                if(isBinaryOp(t))
                    pressBinaryOp(t);
            }

        } catch (Exception e) {
            System.out.println(e.getMessage());
            JOptionPane.showMessageDialog(frame, "Invalid Expression: Please enter
valid Expression!", "CalC: Error",
                    JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
    private void EXIT() {
        System.exit(0);
    }

    private void pressDigit(int n) {
        if (n == 0) {
            if (stack.isEmpty())
                return;
            Symbol top = stack.get(stack.size() - 1);
            if (top.val == 0)
                return;
        }
        if (stack.isEmpty()) {
            stack.add(new Symbol("" + n, 0, (double) n, "" + n, 1));
            renderInfo();
            return;
        }
        Symbol top = stack.get(stack.size() - 1);
        if (isNum(top.id)) {
            if (top.val == 0)
                stack.remove(stack.size() - 1);
            stack.add(new Symbol("" + n, top.balancer,
                    top.multiplier >= 0 && top.multiplier < 1 ? top.val + n *
top.multiplier : top.val * 10 + n, "" + n,
                    top.multiplier >= 0 && top.multiplier < 1 ? top.multiplier / 10.0 :
1));
        } else if (isBinaryOp(top.id)) {
            stack.add(new Symbol("" + n, top.balancer, (double) n, "" + n, 1));
        } else if (isTrigonometryOp(top.id)) {
            stack.add(new Symbol("" + n, top.balancer, (double) n, "" + n, 1));
        } else if (isLogOp(top.id)) {
            stack.add(new Symbol("" + n, top.balancer, (double) n, "" + n, 1));
        } else if (isDot(top.id)) {
            stack.add(new Symbol("" + n, top.balancer,
                    top.multiplier >= 0 && top.multiplier < 1 ? top.val + n *
top.multiplier : top.val * 10 + n, "" + n,
                    0.01));
        } else if (isFactorialOp(top.id)) {
            stack.add(new Symbol("*", top.balancer, -1, "*", 1));
            stack.add(new Symbol("" + n, top.balancer, (double) n, "" + n, 1));
        } else if (isOpenBracket(top.id)) {
            stack.add(new Symbol("" + n, top.balancer, (double) n, "" + n, 1));
        } else if (isCloseBracket(top.id)) {
```

```
            stack.add(new Symbol("*", top.balancer, -1, "*", 1));
            stack.add(new Symbol("" + n, top.balancer, (double) n, "" + n, 1));
        }
        renderInfo();
    }

    private void pressBinaryOp(String n) {
        if (stack.isEmpty()) {
            if (n.equals("-")) {
                stack.add(new Symbol("" + n, 0, -1, "" + n, -1));
                renderInfo();
            }
            return;
        }
        Symbol top = stack.get(stack.size() - 1);
        if (isNum(top.id)) {
            stack.add(new Symbol("" + n, top.balancer, -1, "" + n, -1));
        } else if (isBinaryOp(top.id)) {
            stack.remove(stack.size() - 1);
            stack.add(new Symbol("" + n, top.balancer, -1, "" + n, -1));
        } else if (isTrigonometryOp(top.id)) {
            if (n.equals("-")) {
                stack.add(new Symbol("" + n, top.balancer, -1, "" + n, -1));
            }
        } else if (isLogOp(top.id)) {
            if (n.equals("-")) {
                stack.add(new Symbol("" + n, top.balancer, -1, "" + n, -1));
            }
        } else if (isDot(top.id)) {
            stack.remove(stack.size() - 1);
            stack.add(new Symbol("" + n, top.balancer, -1, "" + n, -1));
        } else if (isFactorialOp(top.id)) {
            stack.add(new Symbol("" + n, top.balancer, -1, "" + n, -1));
        } else if (isOpenBracket(top.id)) {
            if (n.equals("-")) {
                stack.add(new Symbol("" + n, 0, -1, "" + n, -1));
            }
        } else if (isCloseBracket(top.id)) {
            stack.add(new Symbol("" + n, top.balancer, -1, "" + n, -1));
        }
        renderInfo();
    }

    private void pressDot(String n) {
```

```
        if (stack.isEmpty()) {
            stack.add(new Symbol("0",0,0,"0",1));
            pressDot(".");
            renderInfo();
            return;
        }
        Symbol top = stack.get(stack.size() - 1);
        if (isNum(top.id)) {
            stack.add(new Symbol("" + n, top.balancer, top.val, "" + n, 0.1));
        } else if (isBinaryOp(top.id)) {
            pressDigit(0);
            pressDot(".");
        } else if (isTrigonometryOp(top.id)) {
            pressDigit(0);
            pressDot(".");
        } else if (isLogOp(top.id)) {
            pressDigit(0);
            pressDot(".");
        } else if (isDot(top.id)) {
        } else if (isFactorialOp(top.id)) {
            pressBinaryOp("*");
            pressDigit(0);
            pressDot(".");
        } else if (isOpenBracket(top.id)) {
            pressDigit(0);
            pressDot(".");
        } else if (isCloseBracket(top.id)) {
            pressBinaryOp("*");
            pressDigit(0);
            pressDot(".");
        }
        renderInfo();

    }

    private void pressInverseOp(String n) {
        if (stack.isEmpty())
            return;
        Symbol top = stack.get(stack.size() - 1);
        if (top.val == 0)
            return;
        ArrayList<Symbol> newStack = new ArrayList<Symbol>();
        newStack.add(new Symbol("1", 0, 1, "1", 1));
        newStack.add(new Symbol("/", 0, -1, "/", -1));
```

```
        newStack.add(new Symbol("(", 1, -1, "(", -1));
        for (int i = 0; i < stack.size(); i++) {
            Symbol sym = stack.get(i);
            sym.balancer += 1;
            newStack.add(sym);
        }
        stack.clear();
        for (int i = 0; i < newStack.size(); i++)
            stack.add(newStack.get(i));
        newStack.clear();
        renderInfo();
        return;
    }

    private void pressOpenBracket() {
        if (stack.isEmpty()) {
            stack.add(new Symbol("(", 1, -1, "(", -1));
            renderInfo();
            return;
        }
        Symbol top = stack.get(stack.size() - 1);
        if (isNum(top.id)) {
            pressBinaryOp("*");
            pressOpenBracket();
        } else if (isBinaryOp(top.id)) {
            stack.add(new Symbol("(", top.balancer + 1, -1, "(", -1));
        } else if (isTrigonometryOp(top.id)) {
            stack.add(new Symbol("(", top.balancer + 1, -1, "(", -1));
        } else if (isLogOp(top.id)) {
            stack.add(new Symbol("(", top.balancer + 1, -1, "(", -1));
        } else if (isDot(top.id)) {
            stack.remove(stack.size() - 1);
            pressBinaryOp("*");
            pressOpenBracket();
        } else if (isFactorialOp(top.id)) {
            pressBinaryOp("*");
            pressOpenBracket();
        } else if (isOpenBracket(top.id)) {
            stack.add(new Symbol("(", top.balancer + 1, -1, "(", -1));
        } else if (isCloseBracket(top.id)) {
            pressBinaryOp("*");
            pressOpenBracket();
        }
        renderInfo();
```

```
    }

    private void pressCloseBracket() {
        if (stack.isEmpty()) {
            return;
        }

        Symbol top = stack.get(stack.size() - 1);
        if (top.val == 0)
            return;
        if (top.balancer <= 0)
            return;
        if (isNum(top.id)) {
            stack.add(new Symbol(")", top.balancer - 1, -1, ")", -1));
        } else if (isBinaryOp(top.id)) {
        } else if (isTrigonometryOp(top.id)) {
        } else if (isLogOp(top.id)) {
        } else if (isDot(top.id)) {
            stack.remove(stack.size() - 1);
            stack.add(new Symbol(")", top.balancer - 1, -1, ")", -1));
        } else if (isFactorialOp(top.id)) {
            stack.add(new Symbol(")", top.balancer - 1, -1, ")", -1));
        } else if (isOpenBracket(top.id)) {
        } else if (isCloseBracket(top.id)) {
            stack.add(new Symbol(")", top.balancer - 1, -1, ")", -1));
        }
        renderInfo();
    }

    private void pressFactorial() {
        if (stack.isEmpty()) {
            return;
        }
        Symbol top = stack.get(stack.size() - 1);
        if (isNum(top.id)) {
            stack.add(new Symbol("!", top.balancer, -1, "!", -1));
        }
        renderInfo();
    }

    private void constantHelper(String n) {
        String s = "";
        if (n.equals("pi"))
            s = String.format("%.14f", Math.PI);
```

```
        if (n.equals("exp"))
            s = String.format("%.14f", Math.E);
        for (int i = 0; i < s.length(); i++) {
            if (isNum("" + s.charAt(i)))
                pressDigit(Integer.parseInt("" + s.charAt(i)));
            if (isDot("" + s.charAt(i)))
                pressDot(".");
            if(isBinaryOp(""+s.charAt(i)))
                pressBinaryOp(""+s.charAt(i));
        }
    }

    private void pressConstant(String n) {
        if (stack.isEmpty()) {
            constantHelper(n);
            return;
        }
        Symbol top = stack.get(stack.size() - 1);
        if (isNum(top.id)) {
            pressBinaryOp("*");
            pressConstant(n);
        } else if (isBinaryOp(top.id)) {
            constantHelper(n);
        } else if (isTrigonometryOp(top.id)) {
            constantHelper(n);
        } else if (isLogOp(top.id)) {
            constantHelper(n);
        } else if (isDot(top.id)) {
            stack.remove(stack.size() - 1);
            pressBinaryOp("*");
            constantHelper(n);
        } else if (isFactorialOp(top.id)) {
            pressBinaryOp("*");
            constantHelper(n);
        } else if (isOpenBracket(top.id)) {
            constantHelper(n);
        } else if (isCloseBracket(top.id)) {
            pressBinaryOp("*");
            constantHelper(n);
        }
    }

    private void pressTrigonometryOp(String t) {
        if (stack.isEmpty()) {
```

```
            stack.add(new Symbol(t, 0, -1, t, -1));
            pressOpenBracket();
            renderInfo();
            return;
        }
        Symbol top = stack.get(stack.size() - 1);
        if (isNum(top.id)) {
            pressBinaryOp("*");
            pressTrigonometryOp(t);
        } else if (isBinaryOp(top.id)) {
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isTrigonometryOp(top.id)) {
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isLogOp(top.id)) {
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isDot(top.id)) {
            stack.remove(stack.size() - 1);
            pressBinaryOp("*");
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isFactorialOp(top.id)) {
            pressBinaryOp("*");
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isOpenBracket(top.id)) {
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isCloseBracket(top.id)) {
            pressBinaryOp("*");
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        }
        renderInfo();
    }

    private void pressLogOp(String t) {
        if (stack.isEmpty()) {
            stack.add(new Symbol(t, 0, -1, t, -1));
            pressOpenBracket();
            renderInfo();
            return;
```

```
        }
        Symbol top = stack.get(stack.size() - 1);
        if (isNum(top.id)) {
            pressBinaryOp("*");
            pressLogOp(t);
        } else if (isBinaryOp(top.id)) {
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isTrigonometryOp(top.id)) {
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isLogOp(top.id)) {
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isDot(top.id)) {
            stack.remove(stack.size() - 1);
            pressBinaryOp("*");
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isFactorialOp(top.id)) {
            pressBinaryOp("*");
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isOpenBracket(top.id)) {
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        } else if (isCloseBracket(top.id)) {
            pressBinaryOp("*");
            stack.add(new Symbol(t, top.balancer, -1, t, -1));
            pressOpenBracket();
        }
        renderInfo();
    }

    public MainScreen() {

        // Initialise frame

        frame = new JFrame("CalC");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(900, 300);
        frame.setResizable(false);
        frame.setLocationRelativeTo(null);
```

```
        // Implement Key Events

        KeyboardFocusManager manager =
KeyboardFocusManager.getCurrentKeyboardFocusManager();
        manager.addKeyEventDispatcher(new CustomDispatcher());

        // Initialise all layouts

        mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));
        mainPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

        textPanel = new JPanel(new GridLayout(1, 1));
        textPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 10, 0));
        btnPanel = new JPanel(new GridLayout(1, 2));
        btnPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 10, 0));
        subPanel = new JPanel(new GridLayout(1, 2));

        numPanel = new JPanel(new GridLayout(4, 4, 10, 10));
        numPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 10));
        toolPanel = new JPanel(new GridLayout(5, 5, 10, 10));
        toolPanel.setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 0));

        btnLeftPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        btnRightPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

        // Initalise Stack
        stack = new ArrayList<Symbol>();

        // Initialise TextField

        textField = new JTextField("0");
        textField.setEditable(false);

        // Initialise Main Buttons

        btns = new ArrayList<JButton>();
        btns.add(new JButton("DEL"));
        btns.add(new JButton("CLR"));
        btns.add(new JButton("ANS"));
        btns.add(new JButton("EXIT"));

        for (JButton btn : btns) {
            btn.addActionListener(new ActionListener() {
```

```
            @Override
            public void actionPerformed(ActionEvent arg0) {
                String text = btn.getText();
                if (text.equals("CLR")) {
                    CLR();
                } else if (text.equals("DEL")) {
                    DEL();

                } else if (text.equals("ANS")) {
                    ANS();
                } else {
                    EXIT();
                }
            }

        });

    }

    // Initialise Num Buttons

    numButtons = new ArrayList<JButton>();
    String[] textArr = new String[] { "7", "8", "9", "+", "4", "5", "6", "-", "1",
"2", "3", "*", "0", ".", "+-",
            "/" };
    for (String t : textArr) {
        JButton btn = new JButton(t);
        btn.addActionListener(new ActionListener() {

            @Override
            public void actionPerformed(ActionEvent arg0) {
                if (isNum(btn.getText())) {
                    pressDigit(Integer.parseInt(btn.getText()));
                } else if (isBinaryOp(btn.getText())) {
                    pressBinaryOp(btn.getText());
                } else if (isDot(btn.getText())) {
                    pressDot(".");
                }
            }

        });
        numButtons.add(btn);
```

```
        }
        // Initialise Tool Buttons

        toolButtons = new ArrayList<JButton>();
        textArr = new String[] { "x^2", "x^3", "x^y", "1/x", "(", "sq-rt", "cb-rt",
"y-rt", "n!", ")", "sin", "cos",
                "tan", "exp", "nPr", "asin", "acos", "atan", "log", "nCr", "sinh",
"cosh", "tanh", "log10", "pi" };
        for (String t : textArr) {
            JButton btn = new JButton(t);
            btn.addActionListener(new ActionListener() {

                @Override
                public void actionPerformed(ActionEvent arg0) {
                    if (t.equals("x^2")) {
                        pressBinaryOp("^");
                        pressDigit(2);
                    } else if (t.equals("x^3")) {
                        pressBinaryOp("^");
                        pressDigit(3);
                    } else if (t.equals("x^y")) {
                        pressBinaryOp("^");
                    } else if (t.equals("1/x")) {
                        pressInverseOp("1/x");
                    } else if (t.equals("(")) {
                        pressOpenBracket();
                    } else if (t.equals(")")) {
                        pressCloseBracket();
                    } else if (t.equals("sq-rt")) {
                        pressBinaryOp("^");
                        pressOpenBracket();
                        pressDigit(1);
                        pressBinaryOp("/");
                        pressDigit(2);
                        pressCloseBracket();
                    } else if (t.equals("cb-rt")) {
                        pressBinaryOp("^");
                        pressOpenBracket();
                        pressDigit(1);
                        pressBinaryOp("/");
                        pressDigit(3);
                        pressCloseBracket();
                    } else if (t.equals("y-rt")) {
                        pressBinaryOp("^");
```

```
                    pressOpenBracket();
                    pressDigit(1);
                    pressBinaryOp("/");
                } else if (t.equals("n!")) {
                    pressFactorial();
                } else if (t.equals("nPr")) {
                    pressBinaryOp("P");
                } else if (t.equals("nCr")) {
                    pressBinaryOp("C");
                } else if (t.equals("exp")) {
                    pressConstant("exp");
                } else if (t.equals("pi")) {
                    pressConstant("pi");
                } else if (isTrigonometryOp(t)) {
                    pressTrigonometryOp(t);
                } else if (isLogOp(t)) {
                    pressLogOp(t);
                }
            }

        });
        toolButtons.add(btn);
    }
    // Adding all components to respective panels

    textPanel.add(textField);

    for (JButton button : btns) {
        if (button.getText().equals("DEL") || button.getText().equals("CLR"))
            btnLeftPanel.add(button);
        else
            btnRightPanel.add(button);
    }

    for (JButton button : numButtons)
        numPanel.add(button);

    for (JButton button : toolButtons)
        toolPanel.add(button);

    btnPanel.add(btnLeftPanel);
    btnPanel.add(btnRightPanel);

    subPanel.add(numPanel);
```

```
        subPanel.add(toolPanel);

        mainPanel.add(textPanel);
        mainPanel.add(btnPanel);
        mainPanel.add(subPanel);

        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        frame.setVisible(true);

    }

}
```