Project Report

Calculator using infix to postfix

Java Programming Language

GROUP MEMBERS

Student Name: **Abdul Raheem**
Student ID: **FA18-BSSE-0051**

Student Name: **ABDIRAHMAN AHMED KHALIF**
Student ID: **FA18-BSSE-0047**

FACULTY

Lab Teacher: **Ms. Rida Ayesha**

Theory Teacher: **Dr Syed Imran Jami**

Table of Contents

Introduction

We are group of two person Abdul Raheem, ID: FA18-BSSE-0051 and Abdirahman Ahmed Khalif, ID: FA18-BSSE-0047. And we made a calculator using the infix to postfix in Java Programming Language and we used IntelliJ IDEA as editor.

Software Description

We made a calculator is an application that performs arithmetic operations on numbers. The application performing the simplest calculator that can do only addition, subtraction, multiplication, and division.

In addition to that the software is performing infix to postfix data structure algorithm in every arithmetic operation performed.

In our software are consisting of five classes code, first class is the **Calculator**, this class is the construction of GUI of the software. The second class is **ButtonFunction**, this class is for the functions of the buttons in the calculator. The third class is the **ArithmeticOperations**, this class is for performing the arithmetic operation addition, subtraction, multiplication and division. The fourth class is **CalculatorExceptions**, this class is for the exception handling that can happen like dividing number by zero. And the fifth class is the **CalculatorDriver** or the Main method of the software.

Hardware/ Software Requirements

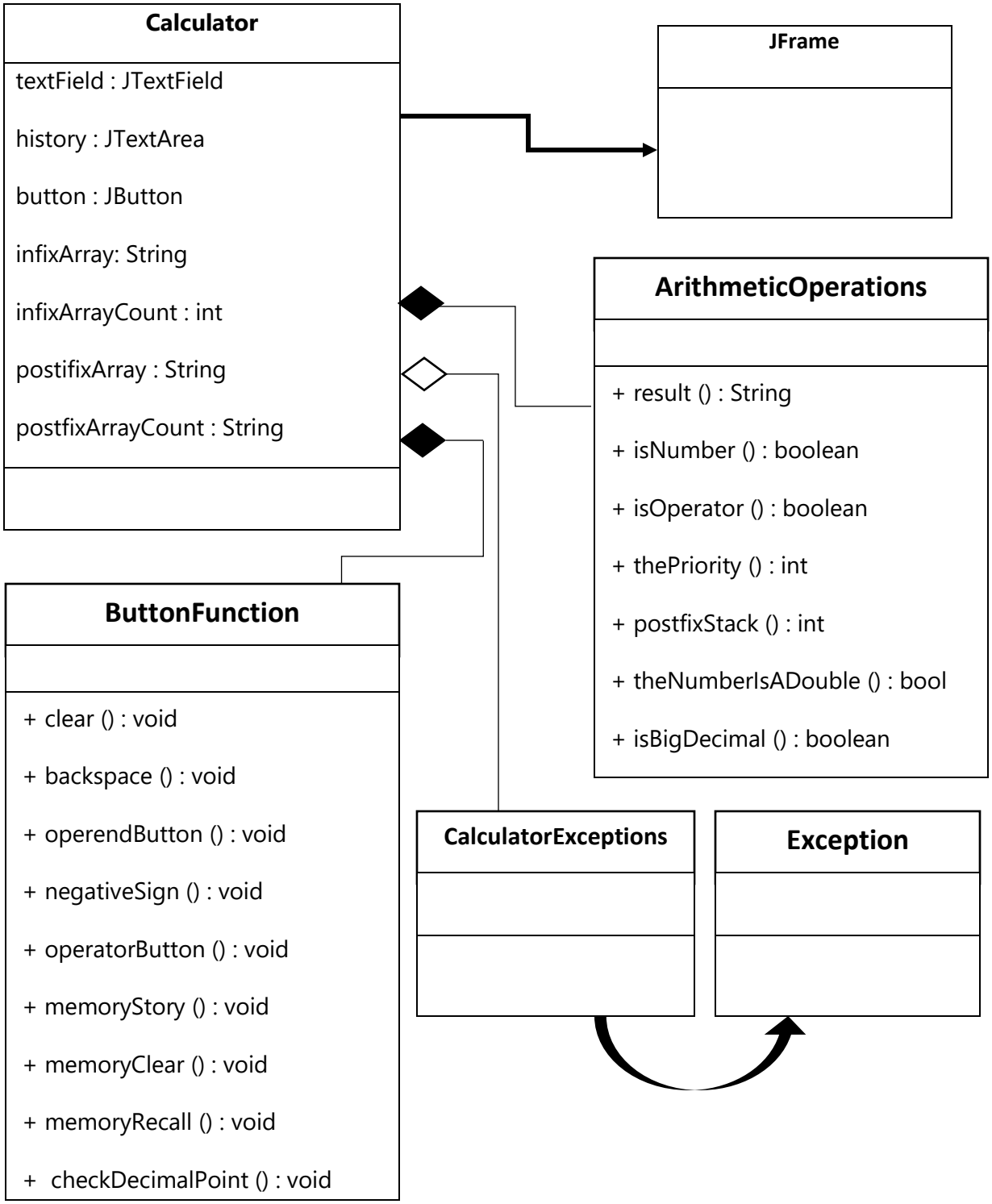
Hardware

- A minimum computer system that will help you access all the tools in the courses is a Pentium 166 or better
- 64 Megabytes of RAM or better
- Windows 2000 (or higher if possible)
- Java Virtual Machine

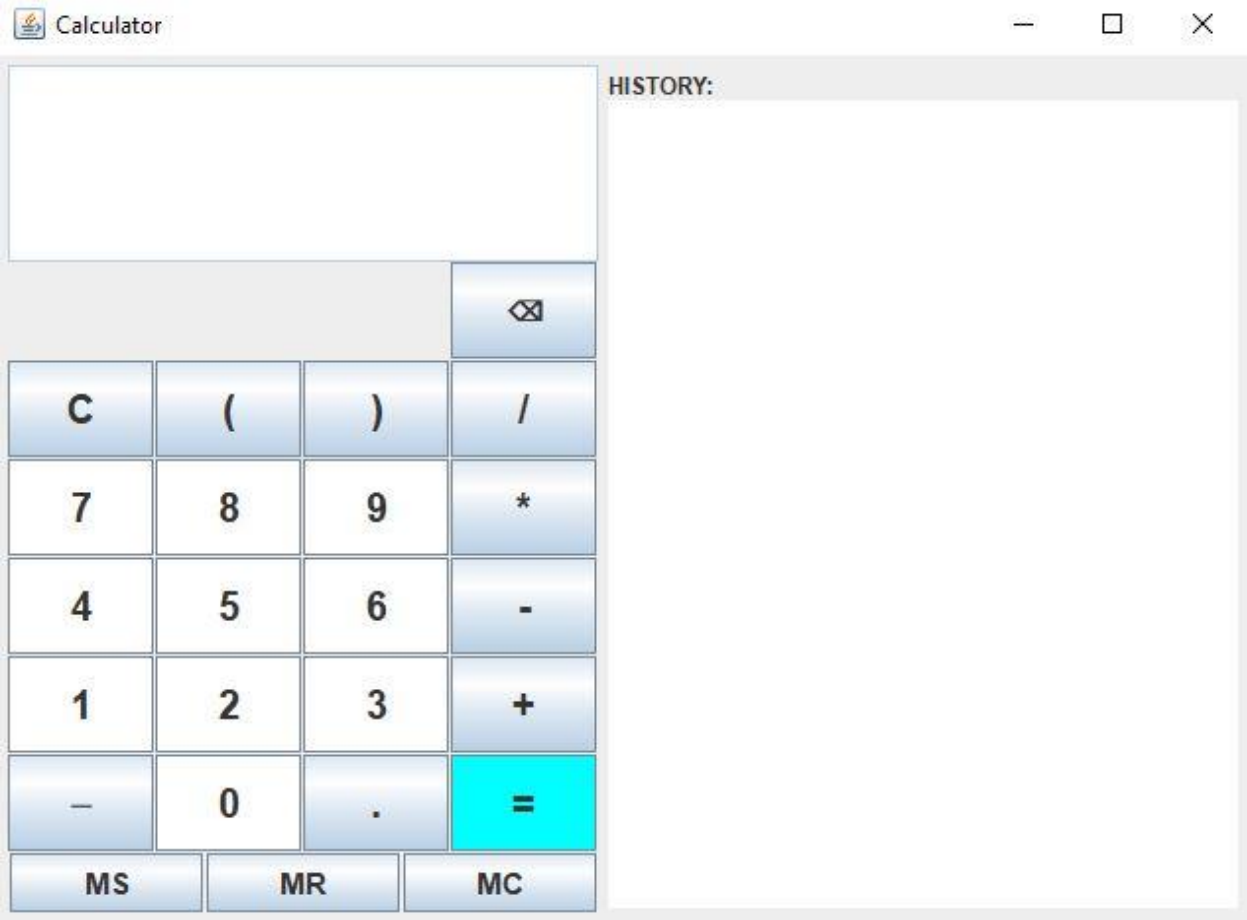
Software

- Notepad/Java editor/IntelliJ IDEA
- Jdk-13

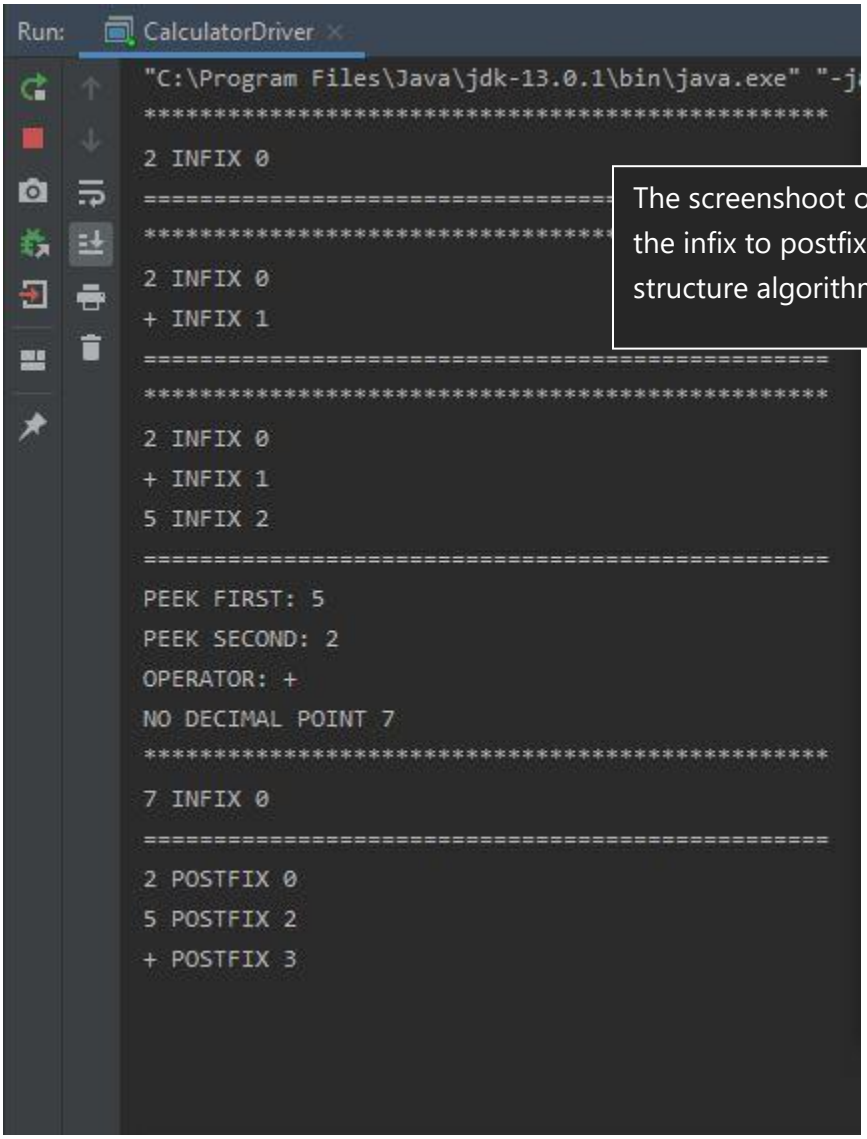
UML DIAGRAM


Calculator
textField : JTextField
history : JTextArea
button : JButton
infixArray: String
infixArrayCount : int
postifixArray : String
postfixArrayCount : String
JFrame
ArithmeticOperations
+ result () : String
+ isNumber () : boolean
+ isOperator () : boolean
+ thePriority () : int
+ postfixStack () : int
+ theNumberIsADouble () : bool
+ isBigDecimal () : boolean
ButtonFunction
+ clear () : void
+ backspace () : void
+ operendButton () : void
+ negativeSign () : void
+ operatorButton () : void
+ memoryStory () : void
+ memoryClear () : void
+ memoryRecall () : void
+ checkDecimalPoint () : void
CalculatorExceptions
Exception

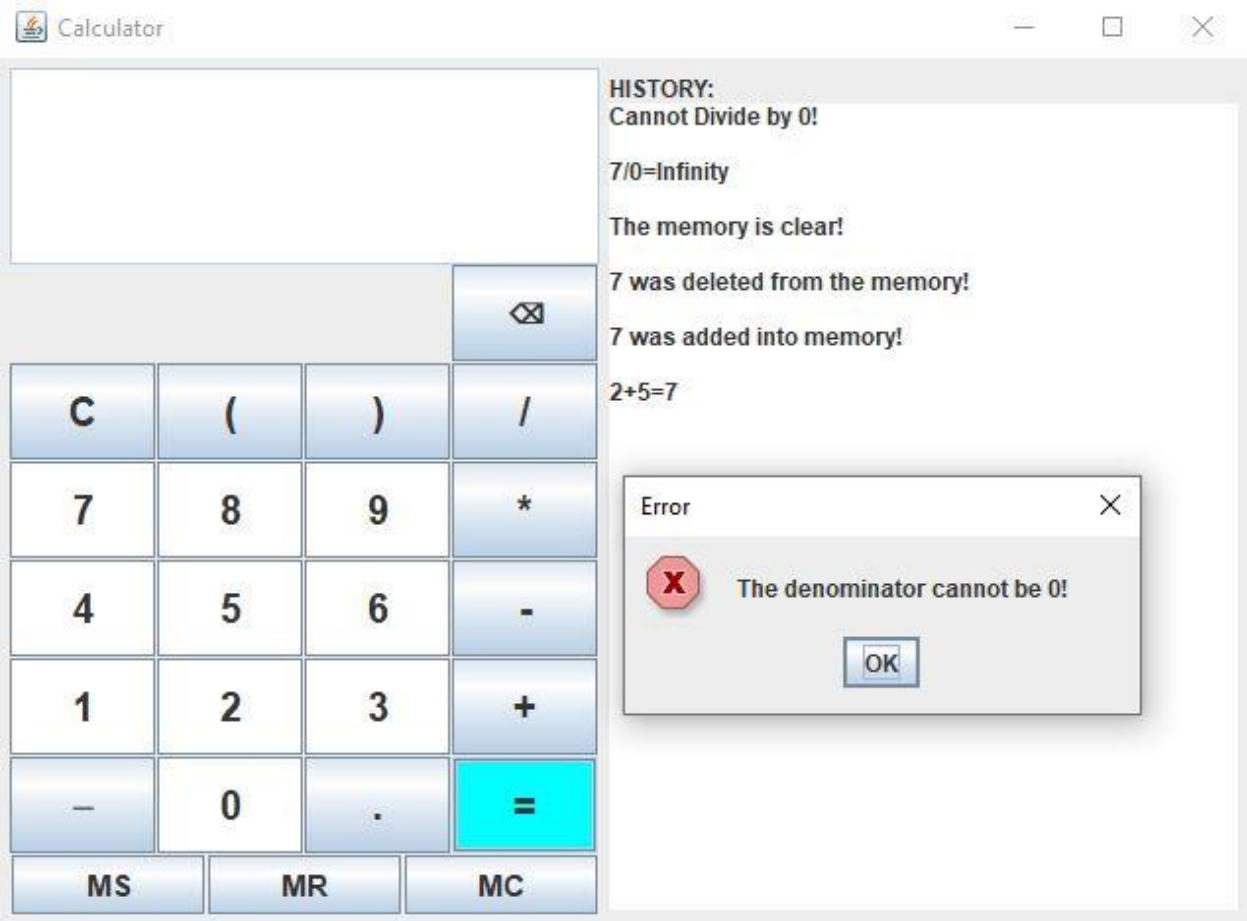
Calculator Snap Shots



The start window of the calculator application



The screenshoot of the process of the infix to postfix of data structure algorithm



Performing the arithmetic operations and storing into the memory "MS" , recalling the memory "MR" and clearing the memory "MC" and also checking the exception handling like dividing number by zero and shows popup massage that says "The denominator cannot be 0!"

DEVELOPER GUIDE
Calculator Code

1) Calculator class

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.JTextField;


public class Calculator extends JFrame
{

    protected static JTextField textField;


    protected static JTextArea history;


    private JButton button;
```

```
    private JLabel historyLabel;


    private JPanel forText;
    private JPanel mainButtons;
    private JPanel memoryButtons;
    private JPanel allButtons;
    private JPanel bigPanel;
    private JPanel space;
    private JPanel historyPanel;


    protected static String buttonLabel = "";
    protected static String elements = "";

    protected static int max = 100;

    protected static String infixArray[] = new String[max];
    protected static int infixArrayCount = 0;

    protected static String postfixArray[] = new String[max];
    protected static int postfixArrayCount;


    private ButtonListener readLabel = new ButtonListener();


    protected static String MS = "";


    public Calculator()
    {
        super("Calculator");

        // text field
        forText = new JPanel();
```

```
        textField = new JTextField("", 25);

        textField.setHorizontalAlignment(JTextField.RIGHT);
        textField.setEditable(false);
        textField.setBackground(Color.white);
        textField.setFont(new Font("Arial", Font.BOLD, 12));


        forText.setLayout(new GridLayout(1,1));
        forText.add(textField);
        forText.setPreferredSize(new Dimension(300,100));


        mainButtons = new JPanel();
        mainButtons.setLayout(new GridLayout(6,4,1,1));


        space = new JPanel();
        mainButtons.add(space);
        space = new JPanel();
        mainButtons.add(space);
        space = new JPanel();
        mainButtons.add(space);
        // delete button -> /u232b is the unicode for backspace symbol
        button = new JButton("\u232b");
        button.addActionListener(readLabel);
        mainButtons.add(button);
        button.setToolTipText("Backspace");
        button.setFont(button.getFont().deriveFont(15f));

        button = new JButton("C");
        button.addActionListener(readLabel);
        mainButtons.add(button);
        button.setToolTipText("Clear");
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("(");
```

```
        button.addActionListener(readLabel);
        button.setToolTipText("Open Parenthese");
        mainButtons.add(button);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton(")");
        button.addActionListener(readLabel);
        mainButtons.add(button);
        button.setToolTipText("Close Parenthese");
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("/");
        button.addActionListener(readLabel);
        mainButtons.add(button);
        button.setToolTipText("Division");
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("7");
        button.addActionListener(readLabel);
        button.setBackground(Color.white);
        mainButtons.add(button);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("8");
        button.addActionListener(readLabel);
        button.setBackground(Color.white);
        mainButtons.add(button);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("9");
        button.addActionListener(readLabel);
        button.setBackground(Color.white);
        mainButtons.add(button);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("*");
        button.addActionListener(readLabel);
```

```
        mainButtons.add(button);
        button.setToolTipText("Multiplication");
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("4");
        button.addActionListener(readLabel);
        button.setBackground(Color.white);
        mainButtons.add(button);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("5");
        button.addActionListener(readLabel);
        button.setBackground(Color.white);
        mainButtons.add(button);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("6");
        button.addActionListener(readLabel);
        button.setBackground(Color.white);
        mainButtons.add(button);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("-");
        button.addActionListener(readLabel);
        mainButtons.add(button);
        button.setToolTipText("Subtraction");
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("1");
        button.addActionListener(readLabel);
        button.setBackground(Color.white);
        mainButtons.add(button);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("2");
        button.addActionListener(readLabel);
        button.setBackground(Color.white);
```

```
        mainButtons.add(button);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("3");
        button.addActionListener(readLabel);
        button.setBackground(Color.white);
        mainButtons.add(button);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("+");
        button.addActionListener(readLabel);
        mainButtons.add(button);
        button.setToolTipText("Addition");
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("\u2212");
        button.addActionListener(readLabel);
        mainButtons.add(button);
        button.setToolTipText("Negative Number");
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("0");
        button.addActionListener(readLabel);
        mainButtons.add(button);
        button.setBackground(Color.white);
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton(".");
        button.addActionListener(readLabel);
        mainButtons.add(button);
        button.setToolTipText("Decimal Point");
        button.setFont(button.getFont().deriveFont(20f));

        button = new JButton("=");
        button.addActionListener(readLabel);
        button.setBackground(Color.CYAN);
        mainButtons.add(button);
```

```
        button.setToolTipText("Equal");
        button.setFont(button.getFont().deriveFont(20f));

        mainButtons.setPreferredSize(new Dimension(300,300));


        memoryButtons = new JPanel();
        memoryButtons.setLayout(new GridLayout(1,3,2,2));

        button = new JButton("MS");
        button.addActionListener(readLabel);
        button.setToolTipText("Store in memory");
        button.setFont(button.getFont().deriveFont(15f));
        memoryButtons.add(button);

        button = new JButton("MR");
        button.addActionListener(readLabel);
        memoryButtons.add(button);
        button.setToolTipText("Memory Recall");
        button.setFont(button.getFont().deriveFont(15f));

        button = new JButton("MC");
        button.addActionListener(readLabel);
        button.setToolTipText("Clear Memory");
        button.setFont(button.getFont().deriveFont(15f));
        memoryButtons.add(button);



        allButtons = new JPanel();
        allButtons.setLayout(new BorderLayout());
        allButtons.add(mainButtons, BorderLayout.NORTH);
        allButtons.add(memoryButtons, BorderLayout.SOUTH);


        history = new JTextArea();
        history.setPreferredSize(new Dimension(320,410));
```

```
        history.setEditable(false); // no input from the user, just display the history
        history.setFont(new Font("Arial", Font.BOLD, 12));
        historyLabel= new JLabel("HISTORY:");


        historyPanel = new JPanel();
        historyPanel.setLayout(new BorderLayout());
        historyPanel.add(historyLabel, BorderLayout.NORTH);
        historyPanel.add(history, BorderLayout.SOUTH);


        bigPanel = new JPanel();
        bigPanel.setLayout(new BorderLayout());
        bigPanel.add(forText, BorderLayout.NORTH);
        bigPanel.add(allButtons, BorderLayout.SOUTH);


        setLayout(new FlowLayout(FlowLayout.CENTER));
        add(bigPanel);
        add(historyPanel);
        pack();
    }

    class ButtonListener implements ActionListener
    {
        @Override
        public void actionPerformed(ActionEvent readLabel)
        {

            buttonLabel = readLabel.getActionCommand();
            textField.setText(textField.getText() + buttonLabel);


            if(buttonLabel.equals("C"))
                ButtonFunction.clear();
```

```
        if(buttonLabel.equals("\u232b"))
            ButtonFunction.backspace();


        if(buttonLabel.equals("0"))
            ButtonFunction.operandButton();

        else if(buttonLabel.equals("1"))
            ButtonFunction.operandButton();

        else if(buttonLabel.equals("2"))
            ButtonFunction.operandButton();

        else if(buttonLabel.equals("3"))
            ButtonFunction.operandButton();

        else if(buttonLabel.equals("4"))
            ButtonFunction.operandButton();

        else if(buttonLabel.equals("5"))
            ButtonFunction.operandButton();

        else if(buttonLabel.equals("6"))
            ButtonFunction.operandButton();

        else if(buttonLabel.equals("7"))
            ButtonFunction.operandButton();

        else if(buttonLabel.equals("8"))
            ButtonFunction.operandButton();

        else if(buttonLabel.equals("9"))
            ButtonFunction.operandButton();

        else if(buttonLabel.equals("."))
            ButtonFunction.checkDecimalPoint();
```

```
        else if(buttonLabel.equals("\u2212"))
            ButtonFunction.negativeSign();



        if(buttonLabel.equals("/"))
            ButtonFunction.operatorButton();


        else if(buttonLabel.equals("*"))
            ButtonFunction.operatorButton();


        else if(buttonLabel.equals("-"))
            ButtonFunction.operatorButton();


        else if(buttonLabel.equals("+"))
            ButtonFunction.operatorButton();


        else if(buttonLabel.equals("("))
            ButtonFunction.operatorButton();


        else if(buttonLabel.equals(")"))
            ButtonFunction.operatorButton();


        else if(buttonLabel.equals("="))
            ButtonFunction.equal();


        if(buttonLabel.equals("MS"))
            ButtonFunction.memoryStore();
```

```
            if(buttonLabel.equals("MC"))
                ButtonFunction.memoryClear();


            if(buttonLabel.equals("MR"))
                ButtonFunction.memoryRecall();



            System.out.println("************************************************");
            for(int i = 0; i < infixArrayCount+1; i++)
            {
                if(infixArray[i] != null)
                    System.out.println(infixArray[i] + " INFIX "  + i + "\t");
            }

            System.out.println("================================================");
            for(int i = 0; i < postfixArrayCount; i++)
            {
                if(postfixArray[i] != null)
                    System.out.println(postfixArray[i] + " POSTFIX " + i + "\t");
            }
        }
    }
}
```

2) ButtonFunctions class

```
import java.util.EmptyStackException;

import javax.swing.JFrame;
import javax.swing.JOptionPane;


public class ButtonFunction
{

    public static void clear()
    {

        for(int i = 0; i < Calculator.postfixArray.length; i++)
            Calculator.postfixArray[i] = null;
        for(int j = 0; j < Calculator.infixArray.length; j++)
            Calculator.infixArray[j]=null;


        Calculator.textField.setText("");
        Calculator.elements = "";
        Calculator.buttonLabel = "";
        Calculator.infixArrayCount = 0;
        Calculator.postfixArrayCount = 0;
    }

    public static void backspace()
    {
        try
        {

            String theText = Calculator.textField.getText();
            Calculator.textField.setText(theText.substring(0, theText.length()-2));

            Calculator.infixArray[Calculator.infixArrayCount] =
                    Calculator.infixArray[Calculator.infixArrayCount].substring(0,
```

```
Calculator.infixArray[Calculator.infixArrayCount].length()-1);

            Calculator.elements = Calculator.infixArray[Calculator.infixArrayCount];
        }


        catch(StringIndexOutOfBoundsException errorString )
        {
            try
            {
                Calculator.infixArrayCount--;

                Calculator.infixArray[Calculator.infixArrayCount] =
                        Calculator.infixArray[Calculator.infixArrayCount].substring(0,
Calculator.infixArray[Calculator.infixArrayCount].length()-1);

                Calculator.elements = Calculator.infixArray[Calculator.infixArrayCount];
            }


            catch(ArrayIndexOutOfBoundsException errorArray)
            {
                clear();
            }
        }

        catch(NullPointerException signError)
        {
            Calculator.infixArrayCount--;

            Calculator.infixArray[Calculator.infixArrayCount] =
                    Calculator.infixArray[Calculator.infixArrayCount].substring(0,
Calculator.infixArray[Calculator.infixArrayCount].length()-1);
        }
    }

    public static void operandButton()
```

```
    {
        Calculator.elements += Calculator.buttonLabel;
        Calculator.infixArray[Calculator.infixArrayCount] = Calculator.elements;


    }

    public static void negativeSign()
    {
        if(Calculator.elements.equals(""))
        {
            Calculator.elements += "-";
            Calculator.infixArray[Calculator.infixArrayCount] = Calculator.elements;
        }
        else
        {

            JOptionPane.showMessageDialog(new JFrame(), "You can not use the NEGATIVE sign for subtraction!",
"Calculation Error", JOptionPane.ERROR_MESSAGE);

            clear();
        }
    }

    public static void operatorButton()
    {
        Calculator.infixArrayCount++;

        Calculator.infixArray[Calculator.infixArrayCount] = Calculator.buttonLabel;

        Calculator.elements = "";

        Calculator.infixArrayCount++;
    }


    public static void memoryStore()
```

```
    {
        String theText = Calculator.textField.getText();
        Calculator.textField.setText(theText.substring(0, theText.length() - 2));


        if(Calculator.infixArrayCount==0 && Calculator.MS.length()==0 &&
Calculator.infixArray[Calculator.infixArrayCount]!=null)
        {
            Calculator.MS = Calculator.infixArray[Calculator.infixArrayCount];

            System.out.println("The " + Calculator.MS + " was added into memory!");
            Calculator.history.setText(Calculator.MS + " was added into memory!" + "\n\n"+
Calculator.history.getText());
        }

        else if(Calculator.infixArrayCount==0 && Calculator.MS.length()>0)
        {
            JOptionPane.showMessageDialog(new JFrame(), "Clear the memory first!", "Error",
JOptionPane.ERROR_MESSAGE);

            Calculator.history.setText(Calculator.MS + " is already in the memory!" + "\n\n"+
Calculator.history.getText());
        }
        else
        {
            JOptionPane.showMessageDialog(new JFrame(), "Please insert a valid number!", "Error",
JOptionPane.ERROR_MESSAGE);

            Calculator.history.setText(Calculator.textField.getText() + " is not a valid number to be added in
memory!" + "\n\n"+ Calculator.history.getText());
        }
    }

    public static void memoryClear()
    {

        String theText = Calculator.textField.getText();
```

```
        Calculator.textField.setText(theText.substring(0, theText.length() - 2));


        if(Calculator.MS.length()>0)
        {
            Calculator.history.setText(Calculator.MS + " was deleted from the memory!" + "\n\n"+
Calculator.history.getText());

            Calculator.MS = "";
        }
        else if(Calculator.MS.length()==0)
        {
            Calculator.history.setText(Calculator.MS + "The memory is clear! " + "\n\n"+
Calculator.history.getText());

            clear();
        }
    }


    public static void memoryRecall()
    {

        String theText = Calculator.textField.getText();


        if(Calculator.MS.equals(""))
        {

            Calculator.textField.setText(theText.substring(0, theText.length() - 2));

            Calculator.history.setText(Calculator.MS + "The memory is clear! " + "\n\n"+
Calculator.history.getText());
        }
        else
        {
```

```
            Calculator.textField.setText(theText.substring(0, theText.length() - 2));

            Calculator.infixArray[Calculator.infixArrayCount] = Calculator.MS;

            Calculator.textField.setText(Calculator.textField.getText() + Calculator.MS);
        }
    }

    public static void equal()
    {
        if(Calculator.infixArray[0]==null)
        {
            Calculator.infixArray[0]="0";

            Calculator.textField.setText("0" + Calculator.textField.getText());
        }


        Calculator.postfixArrayCount = ArithmeticOperations.postfixStack(Calculator.postfixArray,
Calculator.infixArray, Calculator.infixArrayCount);
        try
        {
            String total = ArithmeticOperations.result(Calculator.postfixArray, Calculator.postfixArrayCount);


            if(ArithmeticOperations.theNumberIsADouble(total))
            {
                Calculator.history.setText(Calculator.textField.getText() +  total + "\n\n"+
Calculator.history.getText());
                System.out.println("DECIMAL POINT " +  total);

                Calculator.textField.setText(total);

                Calculator.infixArrayCount = 0;

                Calculator.infixArray[Calculator.infixArrayCount] = total;
```

```
                try {
                    if(Double.parseDouble(Calculator.infixArray[0])==Double.POSITIVE_INFINITY ||
Double.parseDouble(Calculator.infixArray[0])==Double.NEGATIVE_INFINITY
                            ||Double.isNaN(Double.parseDouble(Calculator.infixArray[0])) )
                        throw new CalculatorExceptions();
                }

                catch(CalculatorExceptions InfinityResult)
                {
                    System.out.println("Division by 0, handeled");
                }
            }

            else if(ArithmeticOperations.isBigDecimal(total))// adding the whole calculation to the HISTORY TEXT AREA
            {  // if the TOTAL is very big and is written in scientific notation, display to the screen as a
bigDecimal
                Calculator.history.setText(Calculator.textField.getText() +  total + "\n\n"+
Calculator.history.getText());
                System.out.println("DECIMAL POINT " +  total); // TEST - CONSOLE READING
                Calculator.textField.setText(total);
                Calculator.infixArrayCount = 0;
                Calculator.infixArray[Calculator.infixArrayCount] = total;
            }

            else
            {
                total = total.substring(0, total.indexOf('.'));
                Calculator.history.setText(Calculator.textField.getText() +  total + "\n\n"+
Calculator.history.getText());
                System.out.println("NO DECIMAL POINT " +  total);
                Calculator.textField.setText(total);
                Calculator.infixArrayCount = 0;
                Calculator.infixArray[Calculator.infixArrayCount] = total;
            }
        }
```

```
        catch(EmptyStackException tooManySymbols)
        {
            Calculator.history.setText(Calculator.textField.getText() +  "Invalid Input" + "\n\n"+
Calculator.history.getText());
            Calculator.textField.setText("Invalid input");
            JOptionPane.showMessageDialog(new JFrame(), "Invalid input!", "Error", JOptionPane.ERROR_MESSAGE);

            ButtonFunction.clear();
        }

        catch(StringIndexOutOfBoundsException pressingEqualWithNoOP)
        {
            Calculator.history.setText(Calculator.textField.getText() +  "No Operators/Operands found!" + "\n\n"+
Calculator.history.getText());
            Calculator.textField.setText("Invalid input");
            JOptionPane.showMessageDialog(new JFrame(), "Invalid input!", "Error", JOptionPane.ERROR_MESSAGE);

            ButtonFunction.clear();
        }
    }


    public static void checkDecimalPoint()
    {
        if(Calculator.infixArray[Calculator.infixArrayCount]!=null)
        {
            if(Calculator.infixArray[Calculator.infixArrayCount].contains("."))
            {
                String theText = Calculator.textField.getText();
                Calculator.textField.setText(theText.substring(0, theText.length()-1));
                System.out.println("Decimal point ignored"); // TEST - CONSOLE READING
            }
            else
                operandButton();
        }
        else
        {
```

```
            String theText = Calculator.textField.getText();
            Calculator.textField.setText(theText.substring(0, theText.length()-1));
            System.out.println("Decimal point ignored"); // TEST - CONSOLE READING
        }
    }
}
```

3) ArithmeticOperations class

```
import java.util.EmptyStackException;
import java.util.Stack;

import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class ArithmeticOperations
{

    public static String result(String postfixArray[], int postfixArrayCount)
    {
        Stack <String> resultStack =  new Stack<String>();

        for(int i = 0; i < postfixArrayCount; i++)
        {
            if(postfixArray[i] != null)
            {

                if(isNumber(postfixArray[i]))
                    resultStack.push(postfixArray[i]);


                else if(isOperator(postfixArray[i]))
                {
                    System.out.println("PEEK FIRST: " + resultStack.peek());
                    double first = Double.parseDouble(resultStack.pop());
                    System.out.println("PEEK SECOND: " + resultStack.peek());
```

```
                    double second = Double.parseDouble(resultStack.pop());
                    System.out.println("OPERATOR: " + postfixArray[i]);
                    String operator = postfixArray[i];
                    double  calculation = 0;

                    switch(operator)
                    {
                        case"+": calculation = second + first; break;
                        case"-": calculation = second - first; break;
                        case"*": calculation = second * first; break;
                        case"/": calculation = second / first; break;
                    }

                    resultStack.push(calculation+"");
                }
            }
        }// TEST - CONSOLE READING

        return resultStack.pop();
    }

    public static boolean isNumber(String checkNumber)
    {
        try
        {
            Double.parseDouble(checkNumber);
            return true;
        }
        catch(NumberFormatException | NullPointerException nfe)
        {
            return false;
        }
    }

    public static boolean isOperator(String checkOperator)
    {
        switch (checkOperator)
```

```
        {
            case"+": return true;
            case"-": return true;
            case"*": return true;
            case"/": return true;
            default: return false;
        }
    }

    public static int thePriority(String operator)
    {
        int priority = 0;

        if(operator.equals("(") || operator.equals(")"))
            priority = 1; // last priority
        if(operator.equals("+") || operator.equals("-"))
            priority = 2; // middle priority
        if(operator.equals("*") || operator.equals("/"))
            priority = 3; // first priority

        return priority;
    }

    public static int postfixStack(String postfixArray[], String infixArray[], int infixArrayCount)
    {
        Stack <String> postfix =  new Stack<String>();
        int countPostFix = 0;

        for(int i = 0; i <= infixArrayCount; i++)
        {
            if(infixArray[i]==null || infixArray[i].equals(""))
                continue;

            if(infixArray[i].equals("("))
            {
                postfix.push("(");
            }
```

```
            else if(infixArray[i].equals(")"))
            {
                try
                {
                    while(!postfix.peek().equals("("))
                    {
                        postfixArray[countPostFix] = postfix.pop();
                        countPostFix++;
                    }

                    if(postfix.peek().equals("("))
                        postfix.pop();
                }

                catch(EmptyStackException noMatchingBraket)
                {
                    Calculator.history.setText(Calculator.textField.getText() +  "Open bracket missing!" + "\n\n"+
Calculator.history.getText());
                    Calculator.textField.setText("Matching bracket missing");
                    JOptionPane.showMessageDialog(new JFrame(), "Open bracket missing!", "Error",
JOptionPane.ERROR_MESSAGE);

                    ButtonFunction.clear();
                }
            }

            else if(isOperator(infixArray[i]))
            {

                if(postfix.isEmpty() || thePriority(infixArray[i]) > thePriority(postfix.peek()))
                {
                    countPostFix++;
                    postfix.push(infixArray[i]);
                }

                else
```

```
            {
                while(!postfix.isEmpty() && thePriority(infixArray[i]) <= thePriority(postfix.peek()))
                {
                    postfixArray[countPostFix] = postfix.pop();
                    countPostFix++;
                }

                postfix.push(infixArray[i]);
            }
        }

        else if(isNumber(infixArray[i]))
        {
            postfixArray[countPostFix] = infixArray[i];
            countPostFix++;
        }
    }

    while(!postfix.isEmpty())
    {
        postfixArray[countPostFix] = postfix.pop();
        countPostFix++;
    }

    return countPostFix;
}

public static boolean theNumberIsADouble(String result)
{

    boolean answer = true;

    double theResult = Double.parseDouble(result);
    if(theResult%1==0)
        answer = false;

    return answer;
```

```
    }

    public static boolean isBigDecimal(String result)
    {
        return result.contains("E");
    }

}
```

4) CalculatorExceptions class

```
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class CalculatorExceptions extends Exception
{
    public CalculatorExceptions()
    {
        super();

        ButtonFunction.clear();

        Calculator.history.setText(Calculator.textField.getText() +  "Cannot Divide by 0!" + "\n\n"+
Calculator.history.getText());

        Calculator.infixArray[Calculator.infixArrayCount]=null;

        JOptionPane.showMessageDialog(new JFrame(), "The denominator cannot be 0!", "Error",
JOptionPane.ERROR_MESSAGE);
    }
}
```

5) CalculatorDriver class

```
import javax.swing.JFrame;

public class CalculatorDriver
{
    public static void main(String[] args)
    {
        Calculator myWindow = new Calculator();
        myWindow.setLocation(400, 200);
        myWindow.setVisible(true);
        myWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```