A MINOR PROJECT

ON

JAVA CALCULATOR:

Building a basic arithmetic tool.

Using JAVA, SWING GUI

Submitted in partial fulfillment of the requirements for the award of the certificate

In

PROGRAMMING IN JAVA

Submitted By

KASULA NITHYA SRI

Vignan's Institute of Management and Technology for Women (VMTW),

3rd Year (2021-2025),

CSE (Computer Science and Engineering),

kasulanithyasri@gmail.com

Under the Guidance of

1STOP COMPANY

CONTENTS

LIST OF FIGURES

ABSTRACT

Java Swing is a GUI (graphical user Interface) widget toolkit for Java. Java Swing is a part of Oracle's Java foundation classes . Java Swing is an API for providing graphical user interface elements to Java Programs. Swing was created to provide more powerful and flexible components than Java AWT (Abstract Window Toolkit). GUI, which stands for Graphical User Interface, is a user-friendly visual experience builder for Java applications. It comprises graphical units like buttons, labels, windows, etc. via which users can connect with an application. Swing and JavaFX are two commonly used applications to create GUIs in Java.

This project presents a Java-based calculator application employing Graphical User Interface (GUI) and applet technologies. The application offers basic arithmetic functionalities such as addition, subtraction, multiplication, and division, providing users with a convenient interface for performing mathematical operations. Developed using Java's applet framework, the calculator features a user-friendly GUI designed to enhance usability and accessibility. The implementation leverages Java's event-driven programming model to enable interactive input and output handling, ensuring smooth user interaction. This abstract provides an overview of the design and functionality of the Java calculator applet, emphasizing its role in facilitating mathematical computations through intuitive graphical interfaces.

The calculator offers a wide range of functionalities, including basic arithmetic operations such as addition, subtraction, multiplication, and division computation. It provides a user-friendly interface with intuitive input methods and clear output displays, ensuring ease of use for both novice and experienced users. The calculator's robust design ensures accurate results and efficient performance, making it a versatile tool for everyday calculations, educational purposes, and professional applications. This abstract highlights the comprehensive capabilities and user-centric design of the advanced calculator, underscoring its importance as a valuable computational resource in various contexts.

SUBMITTED BY:

KASULA NITHYA SRI

OBJECTIVE

A calculator program in Java allows performing arithmetic operations like addition, subtraction, multiplication, division, and modulo on two numbers. This article explores two methods: If-Else and Switch-Case, to implement the calculator program. It provides code examples and explanations for each method. The time and space complexity of the program are analyzed, with both being constant (O(1)). By understanding the purpose, methods, and implementation details, readers can enhance their understanding of algorithmic thinking and decision-making structures in Java programming.

Purpose of Calculator Program in Java:

A calculator program can be put to use to perform arithmetic operations like addition, subtraction, division, multiplication, and modulo of two numbers by getting user input and giving the output as the result of the computation.

Methods to solve:

1. If-Else

2. Switch-Case

Dry Run of Calculator Program in Java:

Suppose we have two operands, a and b, and we have to perform the following arithmetic operations discussed, namely, addition, subtraction, multiply, divide and modulo, hence we assign 5 to a while 3 to b, tracing out for each operation,

Addition can be stated as, a+b which results in 5+3 = 8.

Subtraction can be stated as, a-b which results in 5-3 = 2.

Multiplication can be stated as, ab which results in 53 = 12.

Division can be stated as, a/b which results in 5/3 = 1.

As suggested above, any of the operations will be performed, with the result stored in a variable. On printing the result, we end the program successfully.

SUBMITTED BY:

KASULA NITHYA SRI

1.INTRODUCTION

1.1 Building a Calculator in Java: An Introduction

In the realm of computer programming, building a calculator serves as a fundamental exercise that not only hones one's coding skills but also provides insight into the core principles of software development. Java, a versatile and widely-used programming language, offers an excellent platform for crafting such applications due to its simplicity, robustness, and object-oriented nature. In this introduction, we embark on a journey to construct a basic calculator using Java, exploring key concepts, design considerations, and implementation strategies along the way.

1.2 Understanding the Requirements

Before delving into code implementation, it is crucial to outline the requirements of our calculator application. At its core, a calculator must possess the ability to perform basic arithmetic operations such as addition, subtraction, multiplication, and division. Additionally, it should facilitate user interaction through a user-friendly interface, enabling input of numerical values and operators, as well as displaying the computed results.

1.3 Setting Up the Development Environment

To commence our project, we need to ensure that our development environment is properly configured. This involves installing the Java Development Kit (JDK), an integrated development environment (IDE) such as Eclipse or IntelliJ IDEA, and setting up a project directory to organize our code files.

1.4 Creating the Calculator Class

In Java, we encapsulate functionalities within classes, which serve as blueprints for objects. Our calculator application can be represented by a **Calculator** class, which contains methods for performing arithmetic operations and handling user input.

Code = import java.util.Scanner;

```
public class Calculator {
    // Class implementation goes here
}
```

1.5 Handling User Input

To interact with our calculator, users need to input numerical values and operators. We can utilize the **Scanner** class in Java to capture input from the command line.

Code = Scanner scanner = new Scanner(System.in);

With the **Scanner** object instantiated, we prompt the user to enter the numbers and operator required for the desired calculation.

1.6 Performing Arithmetic Operations

Once we have obtained the necessary input from the user, we proceed to perform the arithmetic operation based on the provided operator.

Code = double result = 0;

```
switch (operator) {
    case '+':
        result = num1 + num2;
        break;
    case '-':
        result = num1 - num2;
        break;
    case '*':
        result = num1 * num2;
        break;
    case '/':
        if (num2 != 0) {
            result = num1 / num2;
        } else {
            System.out.println("Error: Division by zero!");
            return;
        }
        break;
    default:
```

```
        System.out.println("Error: Invalid operator!");
        return;
}
```

1.7 Displaying the Result

After computing the result, we display it to the user for their reference.

Code = System.out.println("Result: " + result);

1.8 Conclusion

In this introductory exploration, we have embarked on the journey of constructing a basic calculator application using Java. Through the implementation of essential functionalities such as user input handling, arithmetic operation execution, and result display, we have gained insights into the foundational aspects of software development. As we progress further, we can enhance our calculator with additional features, refine its user interface, and optimize its performance to create a more sophisticated and versatile tool. Through continual learning and experimentation, we unlock the full potential of Java as a programming language, empowering us to innovate and create impactful solutions in the ever-evolving landscape of technology.

METHODOLOGY

Java Swing is a GUI (graphical user Interface) widget toolkit for Java. Java Swing is a part of Oracle's Java foundation classes . Java Swing is an API for providing graphical user interface elements to Java Programs. Swing was created to provide more powerful and flexible components than Java AWT (Abstract Window Toolkit).

Online calculators are something every single one of us is familiar with. To implement this in Java in JFrame, we have a special package named Java Swing which in short is a toolkit for the graphical user interface (GUI) in Java.

This is nothing but the buttons and view of the calculator we get in order to perform the arithmetic calculations.

JFrame is one such feature that provides us the complete frame to develop our online calculator with components like buttons, labels, textfield, etc. This JFrame class can be inherited in our class and be made us of. Let us split our code and understand it better.

Inside our class (JavaCalculator), we need to first creates all the objects and variables required by us for making our online calculator. Initially, as a part of GUI,

we will require 10 buttons for the digits 0-9 and also require 6 more buttons for '+', '-', '*', '/', '=' and 'C'. So we create JButton objects so all the above required buttons.

So, to perform these operations we create a constructor in class (Calculator) which will contain all these.

Creating a GUI :

The process of creating a GUI in Swing starts with creating a class that represents the main GUI. An article of this class acts as a container which holds all the other components to be displayed.

In most of the projects, the main interface article is a frame, i.e., the JFrame class in javax.swing package. A frame is basically a window which is displayed whenever a user opens an application on his/her computer. It has a title bar and buttons such as minimize, maximize and close along with other features.

The JFrame class consists of simple constructors such as JFrame() and JFrame(String). The JFrame() leaves the frame's title bar empty, whereas the JFrame(String) places the title bar to a specified text.

Apart from the title, the size of the frame can also be customized. It can be established by incorporating the setSize(305, 400) method by inserting the width and height desired for the frame. The size of a frame is always designated in pixels.

For example, calling **setBounds(10,40,270,60)** that would create a frame .

Usually, frames are invisible at the time of their creation. However, a user can make them visible by using the frame's **setVisible(boolean)** method by using the word 'true' as an argument.

The following are the steps to create GUI in Java :

STEP 1: The following code is to be copied into an editor

```
public void CreateInterface() {
        frame = new JFrame();
        frame.setTitle("Java Calculator");
        frame.getContentPane().setLayout(null);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);
        frame.setSize(305,400);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

STEP 2: Save and compile the code as mentioned above and then run it.

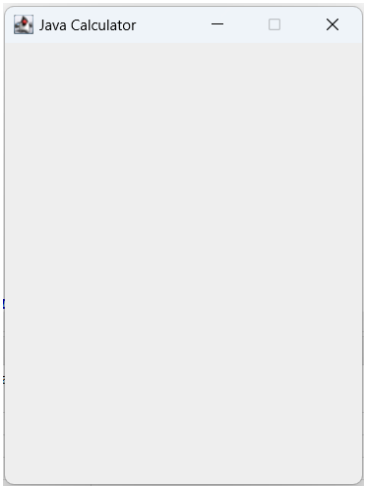


Fig 1: Represents the Frame.

STEP 3:

Adding buttons to the above frame. To create a component in Java, the user is required to create an object of that component's class. We have already understood the container class JFrame.

One such component to implement is JButton. This class represents the clickable buttons. In any application or program, buttons trigger user actions. Literally, every action begins with a click; like to close an application, the user would click on the close button.

A swing can also be inserted, which can feature a text, a graphical icon or a combination of both. A user can use the following constructors:

- **JButton(String):** This button is labelled with a specified text.
- **JButton(Icon):** This button is labelled with a graphical icon.
- **JButton(String,Icon):** This button is labelled with a combination of text and icon.

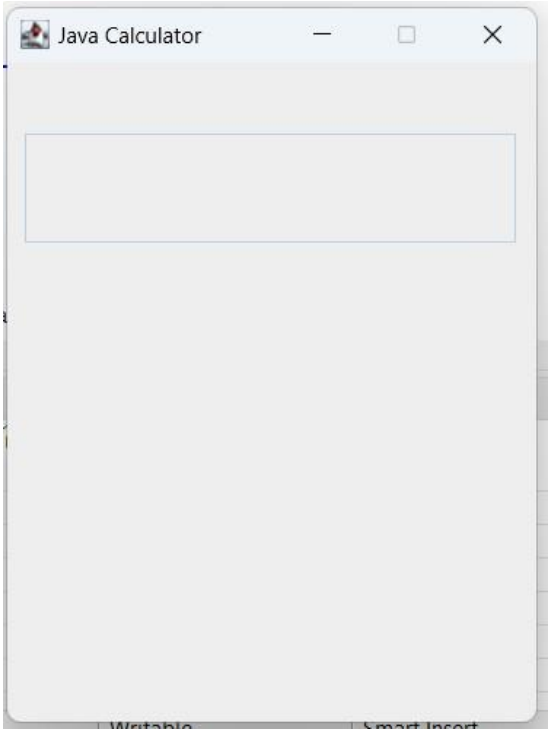


Fig 2: Adding Textfield to the Frame.

STEP 4:

Step 3 is to be executed. A big button will appear on the screen.

STEP 5: A user can add two buttons to the frame as well. Copy the code given below into an editor.

STEP 6: Save, compile and run the above code.

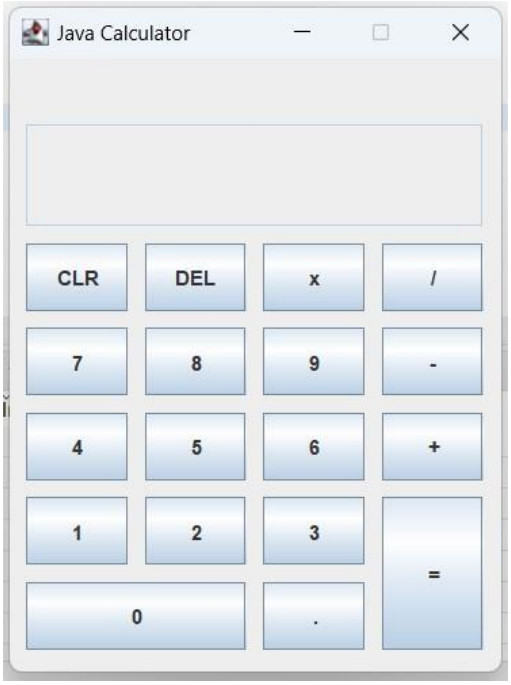


Fig 3: Represents the output Window.

CODE

CODE FOR CREATING CALCULATOR USING JAVA AND JAVA SWING GUI :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class JavaCalculator implements ActionListener{

    double input,result;
    String cal;
    JFrame frame;
    JLabel label = new JLabel();
    JTextField textView = new JTextField();



    JButton symClr = new JButton("CLR");
    JButton symDel = new JButton("DEL");
    JButton symMul = new JButton("x");
    JButton symDiv= new JButton("/");

    JButton numSeven = new JButton("7");
    JButton numEight = new JButton("8");
    JButton numNine = new JButton("9");
    JButton symMinus = new JButton("-");

    JButton numFour = new JButton("4");
    JButton numFive = new JButton("5");
```

```
JButton numSix = new JButton("6");
JButton symPlus = new JButton("+");

JButton numOne = new JButton("1");
JButton numTwo = new JButton("2");
JButton numThree = new JButton("3");
JButton symEqual = new JButton("=");

JButton numZero = new JButton("0");
JButton symDot = new JButton(".");



JavaCalculator(){
        CreateInterface();
        InterfaceComponents();
        AddInterfaceEventListener();

}
public void CreateInterface() {
        frame = new JFrame();
        frame.setTitle("Java Calculator");
        frame.getContentPane().setLayout(null);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);
        frame.setSize(305,400);
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public void InterfaceComponents() {
```

```
label.setBounds(200,0,70,40);
frame.add(label);

textView.setBounds(10, 40, 270, 60);
textView.setEditable(false);
textView.setHorizontalAlignment(SwingConstants.RIGHT);
frame.add(textView);

//First row
symClr.setBounds(10,110,60,40);
frame.add(symClr);
symDel.setBounds(80,110,60,40);
frame.add(symDel);
symMul.setBounds(150,110,60,40);
frame.add(symMul);
symDiv.setBounds(220,110,60,40);
frame.add(symDiv);

//Second row
numSeven.setBounds(10,160,60,40);
frame.add(numSeven);
numEight.setBounds(80,160,60,40);
frame.add(numEight);
numNine.setBounds(150,160,60,40);
frame.add(numNine);
symMinus.setBounds(220,160,60,40);
frame.add(symMinus);

//third row
numFour.setBounds(10,210,60,40);
```

```
        frame.add(numFour);
        numFive.setBounds(80,210,60,40);
        frame.add(numFive);
        numSix.setBounds(150,210,60,40);
        frame.add(numSix);
        symPlus.setBounds(220,210,60,40);
        frame.add(symPlus);

        //fourth row
        numOne.setBounds(10,260,60,40);
        frame.add(numOne);
        numTwo.setBounds(80,260,60,40);
        frame.add(numTwo);
        numThree.setBounds(150,260,60,40);
        frame.add(numThree);
        symEqual.setBounds(220,260,60,90);
        frame.add(symEqual);

        //Fifth row
        numZero.setBounds(10,310,130,40);
        frame.add(numZero);
        symDot.setBounds(150,310,60,40);
        frame.add(symDot);

}
public void AddInterfaceEventListener() {
        //First row
        symClr.addActionListener(this);
        symDel.addActionListener(this);
        symMul.addActionListener(this);
```

```
        symDiv.addActionListener(this);
        //Second row
        numSeven.addActionListener(this);
        numEight.addActionListener(this);
        numNine.addActionListener(this);
        symMinus.addActionListener(this);
        //Third row
        numFour.addActionListener(this);
        numFive.addActionListener(this);
        numSix.addActionListener(this);
        symPlus.addActionListener(this);
        //Four row
        numOne.addActionListener(this);
        numTwo.addActionListener(this);
        numThree.addActionListener(this);
        symEqual.addActionListener(this);
        //fifth row
        numZero.addActionListener(this);
        symDot.addActionListener(this);
}

@Override
public void actionPerformed(ActionEvent e) {
        Object event = e.getSource();

        if(event == numOne){
                textView.setText(textView.getText()+"1");
        }else if(event == numTwo){
                textView.setText(textView.getText()+"2");
        }else if(event == numThree){
```

```
            textView.setText(textView.getText()+"3");
        }else if(event == numFour){
            textView.setText(textView.getText()+"4");
        }else if(event == numFive){
            textView.setText(textView.getText()+"5");
        }else if(event == numSix){
            textView.setText(textView.getText()+"6");
        }else if(event == numSeven){
            textView.setText(textView.getText()+"7");
        }else if(event == numEight){
            textView.setText(textView.getText()+"8");
        }else if(event == numNine){
            textView.setText(textView.getText()+"9");
        }else if(event == numZero){
            if(textView.getText().equals("0")) {
                return;
            }else {
                textView.setText(textView.getText()+"0");
            }
        }else if(event == symDot) {
            if(textView.getText().equals(0)) {
                return;
            }else {
            textView.setText(textView.getText()+".");
            }
        }else if(event == symClr) {
            label.setText("");
            textView.setText("");
        }else if(event == symDel) {
            int length = textView.getText().length();
```

```
            int number = length-1;
            if(length>0) {
            StringBuilder numString = new StringBuilder(textView.getText());
                    numString.deleteCharAt(number);
                    textView.setText(numString.toString());
            }
            if(textView.getText().endsWith("")) {
                    label.setText("");
            }
    }else if(event == symMul) {
            String presentNumber = textView.getText();
            input = Double.parseDouble(textView.getText());
            textView.setText("");
            label.setText(presentNumber+" * ");
            cal = "*";
    }else if(event == symDiv) {
            String presentNumber = textView.getText();
            input = Double.parseDouble(textView.getText());
            textView.setText("");
            label.setText(presentNumber+" / ");
            cal = "/";
    }else if(event == symMinus) {
            String presentNumber = textView.getText();
            input = Double.parseDouble(textView.getText());
            textView.setText("");
            label.setText(presentNumber+" - ");
            cal = "-";
    }else if(event == symPlus) {
            String presentNumber = textView.getText();
            input = Double.parseDouble(textView.getText());
```

```
            textView.setText("");
            label.setText(presentNumber+" + ");
            cal = "+";
        }else if(event == symEqual) {
            switch(cal) {
            case "*" : result = input * (Double.parseDouble(textView.getText()));
                    if(Double.toString(result).endsWith(".0")) {
                       textView.setText(Double.toString(result).replace(".0",""));
                    }else {
                       textView.setText(Double.toString(result));
                    }
                    label.setText("");
                    break;
                case "/" : result = input / (Double.parseDouble(textView.getText()));
            if(Double.toString(result).endsWith(".0")) {
               textView.setText(Double.toString(result).replace(".0",""));
            }else {
               textView.setText(Double.toString(result));
            }
            label.setText("");
            break;
                 case "-" : result = input - (Double.parseDouble(textView.getText()));
        if(Double.toString(result).endsWith(".0")) {
            textView.setText(Double.toString(result).replace(".0",""));
        }else {
            textView.setText(Double.toString(result));
        }
           label.setText("");
           break;
                case "+" : result = input + (Double.parseDouble(textView.getText()));
```

```
                if(Double.toString(result).endsWith(".0")) {
                        textView.setText(Double.toString(result).replace(".0",""));
                }else {
                        textView.setText(Double.toString(result));
                }
                   label.setText("");
                   break;
                        }
                 }
        }

        public static void main(String [] args) {
                 new JavaCalculator();
        }
}
```

RESULT

Screenshot of output:

Fig 4: Output of the code.

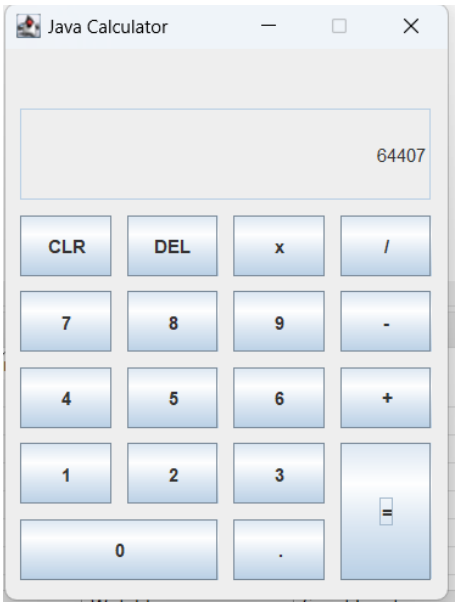
OUTPUT

Performing addition operation:

Giving input to the Calculator:

4444 + 59963 = 64407

Output for the Addition Operation:



Performing subtraction operation:

Giving input to the Calculator:

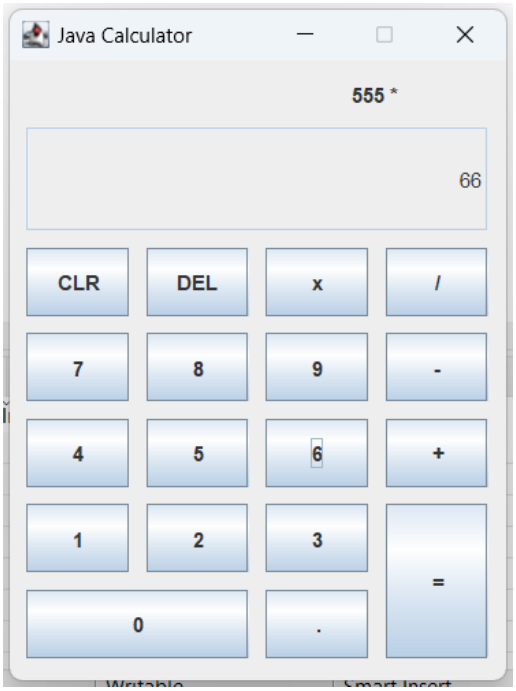
58723 – 96852 = -38129

Output for the Subtraction Operation:

Performing Multipplication operation:

Giving input to the Calculator:

555 * 66 = 36630

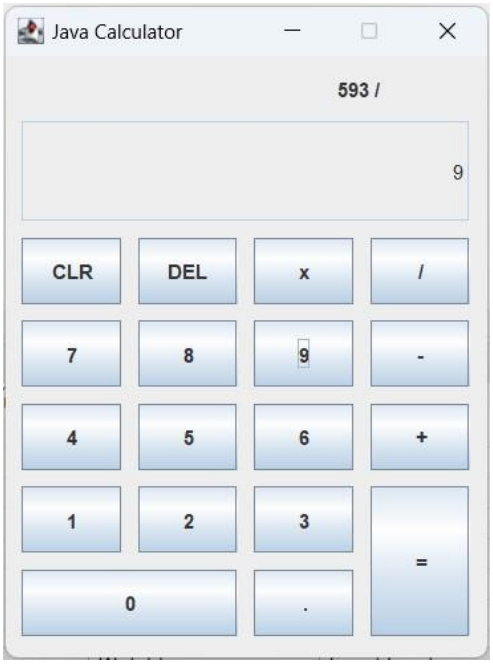


Output for the Multiplication Operation:

Performing Division operation:

Giving input to the Calculator:

593 / 9 = 65.888888888888889

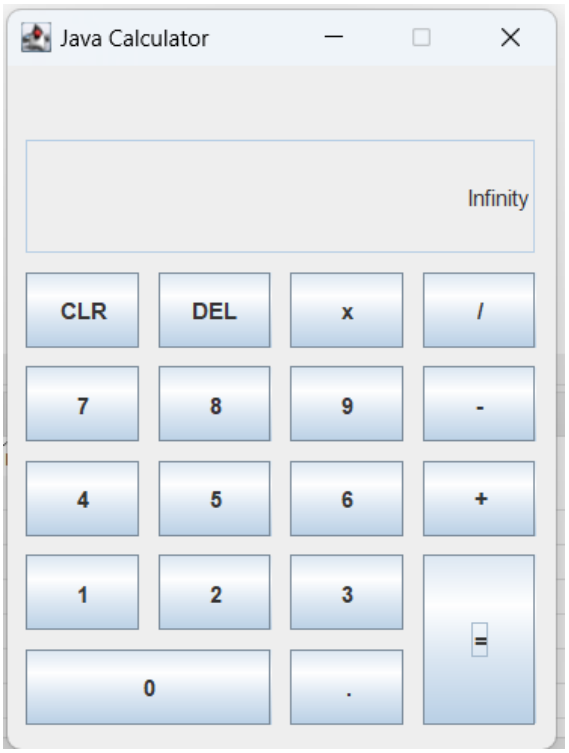


Output for the Division Operation:

Giving input to the Calculator:

6398 / 0 = infinite

Output for the Division Operation:



CONCLUSION

In conclusion, the calculator program in Java serves as a valuable tool for performing basic arithmetic operations on two numbers. This article explored two popular methods, If-Else and Switch-Case, for implementing the calculator program. By providing code examples and explanations, readers gained insights into the logic and structure of each method. Furthermore, the analysis of time and space complexity revealed that the calculator program operates with constant complexity, making it efficient for calculations. By understanding the purpose, methods, and implementation details of the calculator program, readers have enhanced their understanding of algorithmic thinking and decision-making structures in Java programming. This knowledge can be applied to various scenarios, empowering programmers to build more sophisticated applications and solve complex mathematical problems.

Java Calculator is used to calculating operations like addition, subtraction, division and multiplication. This can be done in 2 ways by using a switch case statement and by using swing API. Implementing a calculator in Java Swing provides a robust and user-friendly graphical user interface (GUI) for desktop applications. Java's object-oriented nature and Swing's extensive library of components make it relatively straightforward to create a visually appealing and functional calculator.

Java Swing offers cross-platform compatibility, allowing the calculator to run on any system with a Java Runtime Environment (JRE) installed. Additionally, Swing provides features like event handling, layout management, and customization options, enabling developers to create responsive and interactive user interfaces. However, while Java Swing offers a rich set of features, it may require more code compared to other frameworks or languages, resulting in a longer development time. Additionally, the appearance of Swing applications may not always match the native look and feel of the underlying operating system.

Overall, Java Swing is a solid choice for building calculators and other desktop applications that require a GUI. Its reliability, cross-platform compatibility, and extensive feature set make it a popular choice among developers for creating Java-based GUI applications.