CALCULATOR BUILT IN JAVA

**A PROJECT REPORT**

Submitted by

**AVULA.VENKATA MOHAN REDDY**

# ABSTRACT

Using Java's powerful programming language and Swing's extensive graphical user interface toolkit, one can create a native calculator with Java and Java Swing GUI. The goal of this project is to create a native, fully functional calculator application that can run on multiple platforms and give users a comfortable, user-friendly interface for fundamental arithmetic operations.   
  
The first step of the project is to build the graphical user interface (GUI) utilizing text fields, buttons, and labels that are Java Swing components. The calculator's UI layout has been thoughtfully designed to balance aesthetics and usability, making it simple for users to enter numbers and utilize the calculator's features.

After that, the application logic is put into practice to process user input and carry out mathematical operations. Event listeners are used to record user input, like button clicks, and perform operations like addition, subtraction, multiplication, and division that go along with it. Error handling techniques are included in to handle rare situations like division by zero or invalid   
  
Tension is applied to code quality, maintainability, and extensibility throughout the development process. Utilizing best practices in software architecture and design, a modular and scalable codebase is produced to enable future additions and changes. To guarantee the correctness and dependability of the calculator's operation across a range of use cases and input settings, extensive testing is also carried out.

Table of Contents

[ABSTRACT 1](#_Toc165738882)

[OBJECTIVE 3](#_Toc165738883)

[INTRODUCTION 3](#_Toc165738884)

[ABOUT PROJECT 3](#_Toc165738885)

[PROJECT REQUIREMENTS 4](#_Toc165738886)

[METHODOLOGY 5](#_Toc165738887)

[CODE 7](#_Toc165738888)

[SOURCE CODE: 7](#_Toc165738889)

[EXPLANATION OF THE CODE: 15](#_Toc165738890)

[OUTPUT: 17](#_Toc165738891)

[CONCLUSION 21](#_Toc165738892)

# OBJECTIVE

Create a Native Calculator using Java and Java SWING GUI

1. Use appropriate mathematics to answer elementary addition, subtraction, multiplication, and division problems.   
2. To build a graphic user interface (GUI) that includes some graphics as well as the basic calculator functionality, in order to provide users with a decent interface to work with.

# INTRODUCTION

## ABOUT PROJECT

If you want to learn the fundamentals of Java while also developing a practical application, creating a simple calculator is a good place to start. Calculators are essential tools in many applications, and developing one in Java might be an excellent method to improve your programming abilities. As a Java specialist, allow me to walk you through the process of creating a simple calculator in Java.

In this article, we'll walk you through the process of creating a simple calculator in Java. You'll learn how to configure your Java development environment, create a Java class, accept user input, do calculations, show results, handle errors, and test your software. Whether you're a beginner or an experienced Java developer, this guide will provide you the information and skills you need to develop a functional calculator that you'll be proud of.

Java, with its write-once, run-anywhere concept, provides a reliable framework for developing cross-platform applications. When combined with SWING, Java's lightweight GUI widget library, it allows the building of a responsive and visually appealing graphical user interface. This project will employ these technologies to create a calculator that is both practical and user friendly.  
  
The calculator will have a simple layout with buttons for numbers, basic operations such as addition, subtraction, multiplication, and division, and a display panel for the results. The design will promote intuitiveness and usability, allowing users of various skill levels to easily explore and do calculations.

Following standard coding techniques will help us ensure that the codebase is clear, well-documented, and maintainable as we move forward with the development. The project will also include error handling to improve the application's overall robustness by gracefully accepting common math failures.

## PROJECT REQUIREMENTS

When beginning the process of creating a basic Java calculator, there are a few things you must have ready. Among them are:

1. JDK, or Java Development Kit:

It is required that you have the JDK installed on your PC. The Java runtime environment and compiler required for creating and executing Java applications are included in the JDK. From the official Java website, you may download the JDK and follow the installation guidelines.   
  
2. Integrated Development Environment (IDE):

To write, compile, and execute Java code, you'll need an IDE. Java is supported by a number of well-known IDEs, including Eclipse, IntelliJ IDEA, and NetBeans. Select an IDE that you feel at ease with, or test out a couple to determine which one best meets your needs.

3. Basic grasp of Java syntax:

A basic understanding of Java syntax is required, including data types, variables, operators, control flow expressions (such as if-else and loops), and fundamental object-oriented programming concepts (such classes and methods).   
  
4. fundamental arithmetic understanding:

You should be able to conduct calculations in your calculator using fundamental arithmetic operations such as addition, subtraction, multiplication, and division.   
  
5. Swing module to build the Graphic User Interface:

The advantage of utilizing Swing to develop components such as tables, scroll panes, buttons, and so on is that the framework is platform-independent.

# METHODOLOGY

1.Understanding Requirements:

* Start by knowing the calculator application's requirements. What functionality should it support? What should the user interface look like? What are the intended inputs and outputs?

2.Designing the Interface:

* The calculator's user interface will be designed with buttons for the numbers 0 through 9, operations (+, -, \*, /), and features like equals (=), delete (DEL), and clear (CLR).
* A label (label) indicates the current operation, and a text field (textview) shows the input and results.

3. Setting Up the Environment:

* Verify that the Java Development Kit (JDK) is loaded.
* To write and test the code, use an Integrated Development Environment (IDE) like Eclipse

4. Coding the Calculator:

* To handle button clicks, create the Calculator class and implement ActionListener.
* In the CreateInterface and InterfaceComponents methods, define the components and layout.
* In the AddInterfaceEventListener function, bind action listeners to buttons.

5. Implementing Logic:

* Specify the reasoning behind each button click in the actionPerformed method.
* When an operation button is clicked, the first operand and the specified operation are stored.
* When the equals button is pressed, carry out the division by zero calculation.

6. Refinement:

* Rewrite the code to make it more efficient and readable.
* To clarify the reasoning behind the techniques and logic, add comments.
* Think about including additional features like complex mathematical operations or keyboard input handling.

7. Testing:

* Create a thorough testing plan to confirm that the calculator application is reliable and accurate.
* To verify specific features and components, including input handling and arithmetic operations, use unit tests.
* To guarantee that the UI and backend functionality of the application work together seamlessly, conduct integration testing.

8. Error Handling:

* To handle divide-by-zero errors, invalid inputs, and other unusual situations with grace, put error handling procedures in place.
* Give the user feedback or instructive error messages to help with troubleshooting and resolution.

9. Performance and Optimization:

* Locate any possible coding bottlenecks and improve the effectiveness of key areas.
* To increase the application's overall responsiveness, take into account optimizations like caching, algorithm upgrades, or UI rendering optimizations.

10. Version Control and Cooperation:

* To manage code changes and work together with other developers, use version control systems like Git.
* Make branches for experimenting, feature development, and issue fixes; following testing and evaluation, merge changes back into the main repository.

11. Deployment:

* compile it into a JAR file that can be executed.
* Give the program to people who have installed the Java Runtime Environment (JRE) on other computers.

12. Maintenance:

* Correct any errors that users point out.
* According to customer feedback, add new features to the program.

13.Documentation:

* Compose instructions for users to follow when using the calculator.
* Give developers instructions on how to add to or alter the code in a section.

# CODE

## SOURCE CODE:

// java program to create a simple calculator

// with basic +, -, / , \* using java swing elements

package app;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Calculator implements ActionListener {

double input, result;

String cal;

// creating a frame

// creating a textfield

JFrame frame;

JLabel label = new JLabel();

JTextField textview = new JTextField();

// first row

JButton symCLr = new JButton("CLR");

JButton symDel = new JButton("DEL");

JButton symMul = new JButton("\*");

JButton symDiv = new JButton("/");

// second row

JButton num7 = new JButton("7");

JButton num8 = new JButton("8");

JButton num9 = new JButton("9");

JButton symMinus = new JButton("-");

// third row

JButton num4 = new JButton("4");

JButton num5 = new JButton("5");

JButton num6 = new JButton("6");

JButton symPlus = new JButton("+");

// forth row

JButton num1 = new JButton("1");

JButton num2 = new JButton("2");

JButton num3 = new JButton("3");

JButton symEqual = new JButton("=");

// fifth row

JButton num0 = new JButton("0");

JButton symDot = new JButton(".");

// creating the interface

Calculator() {

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(240, 0, 40, 40);

frame.add(label);

textview.setBounds(10, 50, 270, 60);

textview.setEditable(false);

textview.setHorizontalAlignment(SwingConstants.RIGHT);

frame.add(textview);

// Buttons placement

// First Row

symCLr.setBounds(10, 120, 60, 40);

frame.add(symCLr);

symDel.setBounds(80, 120, 60, 40);

frame.add(symDel);

symMul.setBounds(150, 120, 60, 40);

frame.add(symMul);

symDiv.setBounds(220, 120, 60, 40);

frame.add(symDiv);

// Second Row

num7.setBounds(10, 165, 60, 40);

frame.add(num7);

num8.setBounds(80, 165, 60, 40);

frame.add(num8);

num9.setBounds(150, 165, 60, 40);

frame.add(num9);

symMinus.setBounds(220, 165, 60, 40);

frame.add(symMinus);

// Third Row

num4.setBounds(10, 210, 60, 40);

frame.add(num4);

num5.setBounds(80, 210, 60, 40);

frame.add(num5);

num6.setBounds(150, 210, 60, 40);

frame.add(num6);

symPlus.setBounds(220, 210, 60, 40);

frame.add(symPlus);

// Fourth Row

num1.setBounds(10, 260, 60, 40);

frame.add(num1);

num2.setBounds(80, 260, 60, 40);

frame.add(num2);

num3.setBounds(150, 260, 60, 40);

frame.add(num3);

symEqual.setBounds(220, 260, 60, 90);

frame.add(symEqual);

// Fifth Row

num0.setBounds(10, 310, 130, 40);

frame.add(num0);

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

num7.addActionListener(this);

num8.addActionListener(this);

num9.addActionListener(this);

symMinus.addActionListener(this);

// Third Row

num4.addActionListener(this);

num5.addActionListener(this);

num6.addActionListener(this);

symPlus.addActionListener(this);

// Fourth Row

num1.addActionListener(this);

num2.addActionListener(this);

num3.addActionListener(this);

symEqual.addActionListener(this);

// Fifth Row

num0.addActionListener(this);

symDot.addActionListener(this);

}

@Override

public void actionPerformed(ActionEvent e) {

Object event = e.getSource();

// Input Values

if (event == num0 || event == num1 || event == num2 || event == num3 || event == num4 || event == num5 || event == num6 || event == num7 || event == num8 || event == num9) {

JButton btn = (JButton) event;

String currentText = textview.getText();

if (currentText.equals("0")) {

textview.setText(btn.getText());

} else {

textview.setText(currentText + btn.getText());

}

} else if (event == symDot) {

String currentText = textview.getText();

if (!currentText.contains(".")) {

textview.setText(currentText + ".");

}

} else if (event == symCLr) {

label.setText("");

textview.setText("");

} else if (event == symDel) {

String currentText = textview.getText();

if (!currentText.isEmpty()) {

textview.setText(currentText.substring(0, currentText.length() - 1));

}

} else if (event == symMul || event == symDiv || event == symMinus || event == symPlus) {

JButton btn = (JButton) event;

String currentText = textview.getText();

input = Double.parseDouble(currentText);

cal = btn.getText();

label.setText(currentText + " " + cal + " ");

textview.setText("");

} else if (event == symEqual) {

String currentText = textview.getText();

if (!currentText.isEmpty() && cal != null) {

double secondOperand = Double.parseDouble(currentText);

switch (cal) {

case "\*":

result = input \* secondOperand;

break;

case "/":

if (secondOperand != 0) {

result = input / secondOperand;

} else {

label.setText("Error: Division by zero");

return;

}

break;

case "-":

result = input - secondOperand;

break;

case "+":

result = input + secondOperand;

break;

}

if (result % 1 == 0) {

textview.setText(String.valueOf((int) result));

} else {

textview.setText(String.valueOf(result));

}

label.setText("");

cal = null;

}

}

}

public static void main(String[] args) {

new Calculator();

}

}

## EXPLANATION OF THE CODE:

Since we will be utilizing Swing to design the application's GUI, you can expect that to be the majority of the software, while the mathematics will be kept simple by limiting this basic calculator to +, -, \*, and / operations.

1.Imports:

This code uses classes from the javax.swing and java.awt libraries to create GUI components and handle events.

2.Class Declaration:

The Calculator class is declared, and it implements the ActionListener interface to

handle button click events

3. Instance Variables:

* double input, result: Variables for storing the values entered into the calculation and the output.
* The current arithmetic operation (e.g., addition, subtraction) is stored in the variable "string cal."
* JFrame frame: The application's main window.
* JLabel label: A label that shows the result of the most recent computation.
* JTextField textview: An input and output value display text field.
* Different calculator buttons are represented by various JButton instances.

4. Constructor:

* By using the three methods CreateInterface(), InterfaceComponents(), and AddInterfaceEventListener(), the constructor Calculator() initializes the GUI components.

5. CreateInterface() Method:

* establishes the calculator application's title, layout, size, visibility, and default close function in the main frame.

6.InterfaceComponents() Method :

* incorporates GUI elements into the frame, such as buttons, text fields, and labels, and uses absolute positioning (null layout) to determine their locations.

7. AddInterfaceEventListener() Method:

* registers action listeners to handle user interactions on all buttons.

8. actionPerformed() Method:

* carries out the logic to deal with various user inputs (button clicks).
* It executes arithmetic calculations and modifies the text field's (textview) display text based on the button pressed.
* Additionally, the approach manages specific scenarios such as erasing the last character, clearing the display, and displaying error messages when dividing by zero.

9. main() Method:

* the application's entry point. By creating an instance of the Calculator class, it starts the operations involved in creating the GUI and managing events.

With a straightforward GUI, this code offers basic calculator functions. Using the offered buttons, users may enter numbers and conduct arithmetic operations; the application then displays the results appropriately.

## OUTPUT:

1.Addition

Figure 1.1 addition

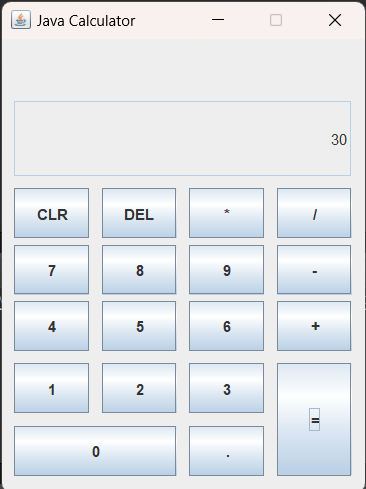


Figure .2 addition result

2.Subtraction



Figure 2.1 Subtraction



Figure 2.2 Subtraction result

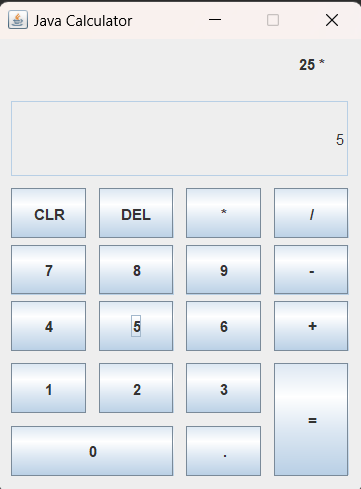
3.Multiplication

Figure 3.1 Multiplication

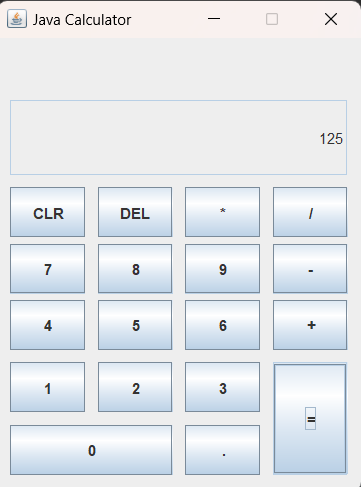


Figure 3.2 Multiplication result

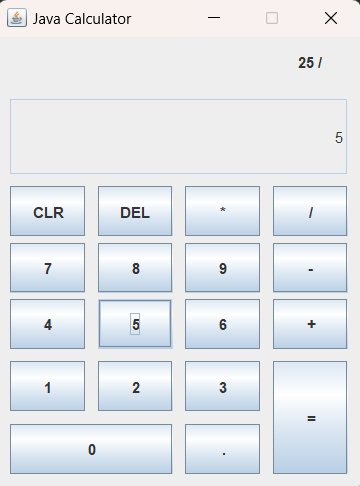
4.Division

Figure 4.1 Division



Figure 4.2 Division result

# CONCLUSION

The creation of a Native Calculator with Java and Java SWING GUI is complete after a protracted development process. The goal of this project was to create a calculator application that was both useful and easy to use by utilizing the powerful features of Java and the SWING GUI components.

Throughout the project, many more important goals were accomplished:

Functionality:

The calculator application does addition, subtraction, multiplication, and division among other common arithmetic operations with success. To further increase the calculator's usefulness, further features including modulus, exponentiation, and square root were included.

User Interface Design:

The creation of an easy-to-use and aesthetically pleasing user interface was made easier by the Java SWING GUI. Clear buttons and a dynamic display for user inputs and outputs characterize the calculator's layout, which was created to guarantee accessibility and ease of use.

Error Handling:

Strong error-handling techniques were put in place to guarantee that the calculator functions properly in a variety of situations. Error messages are shown appropriately to help users when they enter or perform tasks that are incorrect.

Code Structure and Documentation:

The project has an orderly code structure and follows best practices for coding. The codebase is accompanied by extensive documentation that offers insights into the functionality, usability, and specifics of the calculator application's implementation.

Testing and Debugging:

The correctness and dependability of the calculator's computations under various input conditions were extensively tested. Problems and defects were found and fixed quickly to guarantee peak performance.

User Feedback and Iterative Development:

User and stakeholder feedback was essential in helping to improve the calculator application. Iterative development cycles were used to address bugs, add features, and improve the user experience in general.

To conclusion, the development of the Native Calculator using Java and Java SWING GUI is an example of a successful software project. The application achieves its goal of giving users a useful and simple-to-use tool for arithmetic computations. There are prospects for future improvements and feature additions depending on user input and changing needs. Overall, this project highlights the value of user-centric design and extensive testing in software development projects and shows how well Java and SWING GUI work together to create desktop apps.