Shanto-Mariam University Of Creative Technology
Department of CSE

Project Name: Simple Calculator in JavaFX

Course Name: Object Oriented Software Development lab
Course Code: 3170

Submitted To: Md. Fazle Hassan Mihad
Lecturer: Shanto-Mariam University Of Creative Technology
Department: CSE

Submitted By: Abdur Rahaman Shishir
ID: 223071092
Department: CSE
Group: B
Semester: 7th

Index

1. Name of the Project:

Simple Calculator in JavaFX

2. Abstract:

The objective of this JavaFX calculator project is to create a simple, user-friendly desktop calculator that performs basic arithmetic operations such as addition, subtraction, multiplication, and division. The application features both mouse and keyboard input support, error handling (e.g., divide by zero), and additional functionalities like percentage calculation, sign toggling (±), and clear options (C and CE). The final application mimics the behavior of a standard digital calculator with a clean interface and interactive buttons.

3. Introduction:

A calculator is a fundamental tool used in daily life for performing mathematical operations quickly and accurately. From simple arithmetic to more complex calculations, calculators are essential in education, business, engineering, and personal finance. In the digital age, software-based calculators offer greater flexibility and accessibility, making them a common feature in computers and mobile devices.

Region to build a calculator?

To strengthen understanding of JavaFX and GUI development while creating a practical and familiar application that demonstrates core programming concepts.

Goals of this project?

- ✓ To implement a fully functional calculator with basic arithmetic and utility operations.
- ✓ To create a responsive and intuitive user interface.
- ✓ To practice event handling, layout management, and user input validation.

Technologies use in this project?

- ✓ Java as the main programming language.
- ✓ JavaFX for building the graphical user interface.
- ✓ Object-Oriented Programming (OOP) principles to structure and manage application logic cleanly and modularly.

4. Review / Background:

JavaFX is a Java library used to build rich desktop applications with graphical user interfaces. It is suitable for GUI applications because it offers built-in support for modern UI components, styling with CSS, and event handling, making it easy to design interactive and visually appealing interfaces.

5. System Design & Output (With Visual Represent):

The calculator is structured using a clean and intuitive layout built with JavaFX. The graphical user interface (GUI) follows a grid-based design, where buttons are arranged in a **GridPane** to resemble a traditional calculator layout. At the top, a non-editable **TextField** serves as the display area for inputs and results.

Components Used:

- A **TextField** for displaying numbers, operations, and results.
- Multiple **Button** elements representing digits, operations **(e.g., +, -, *, /)**, utility
- functions **(C, CE, %, ±)**, and the equals sign **(=)**.

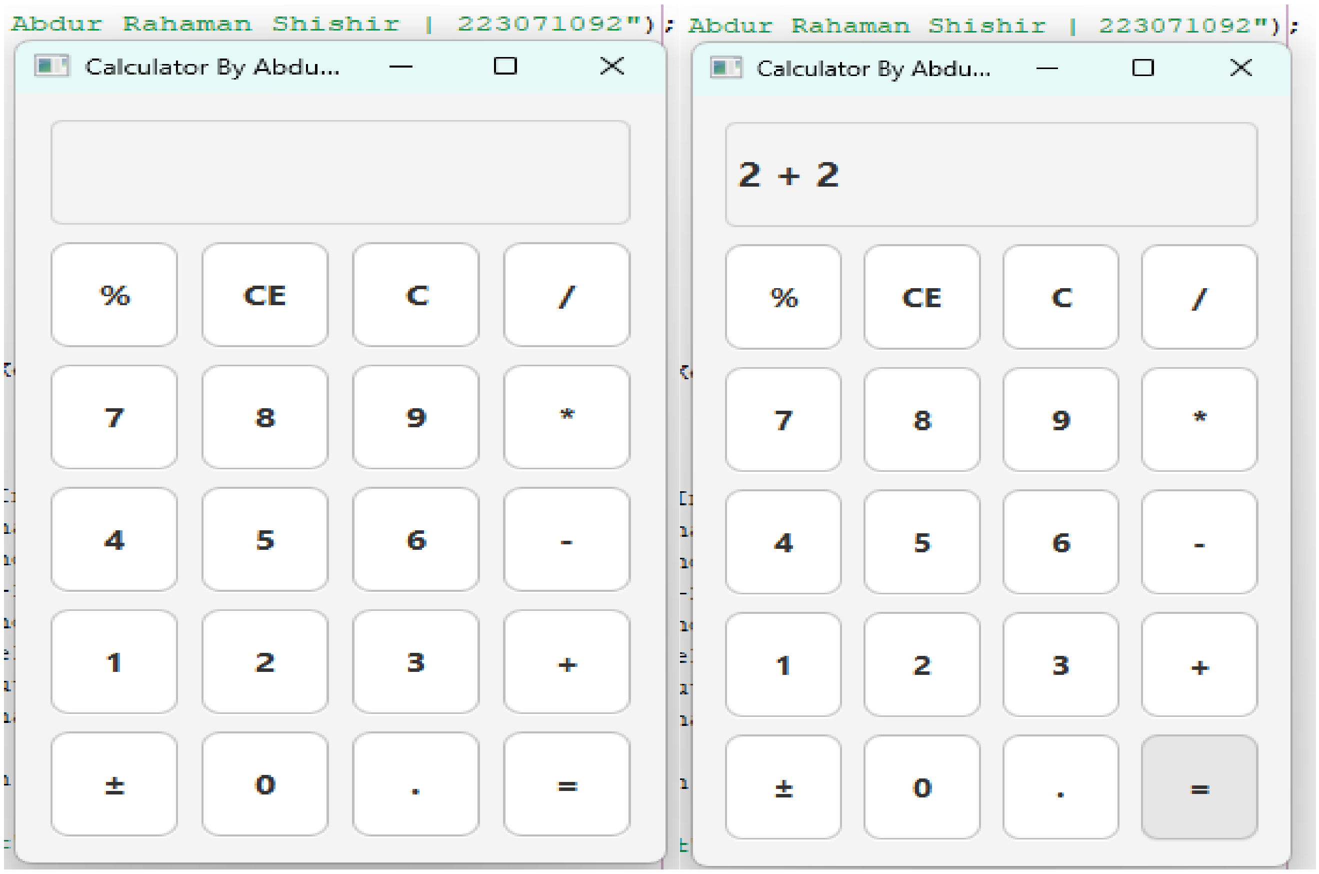
User Input Handling:

- Mouse Input: Each button is interactive and responds to mouse clicks, triggering the appropriate action.
- Keyboard Input: The application listens for key presses, allowing the user to input numbers and operations via the keyboard for faster interaction.

Event Handling:

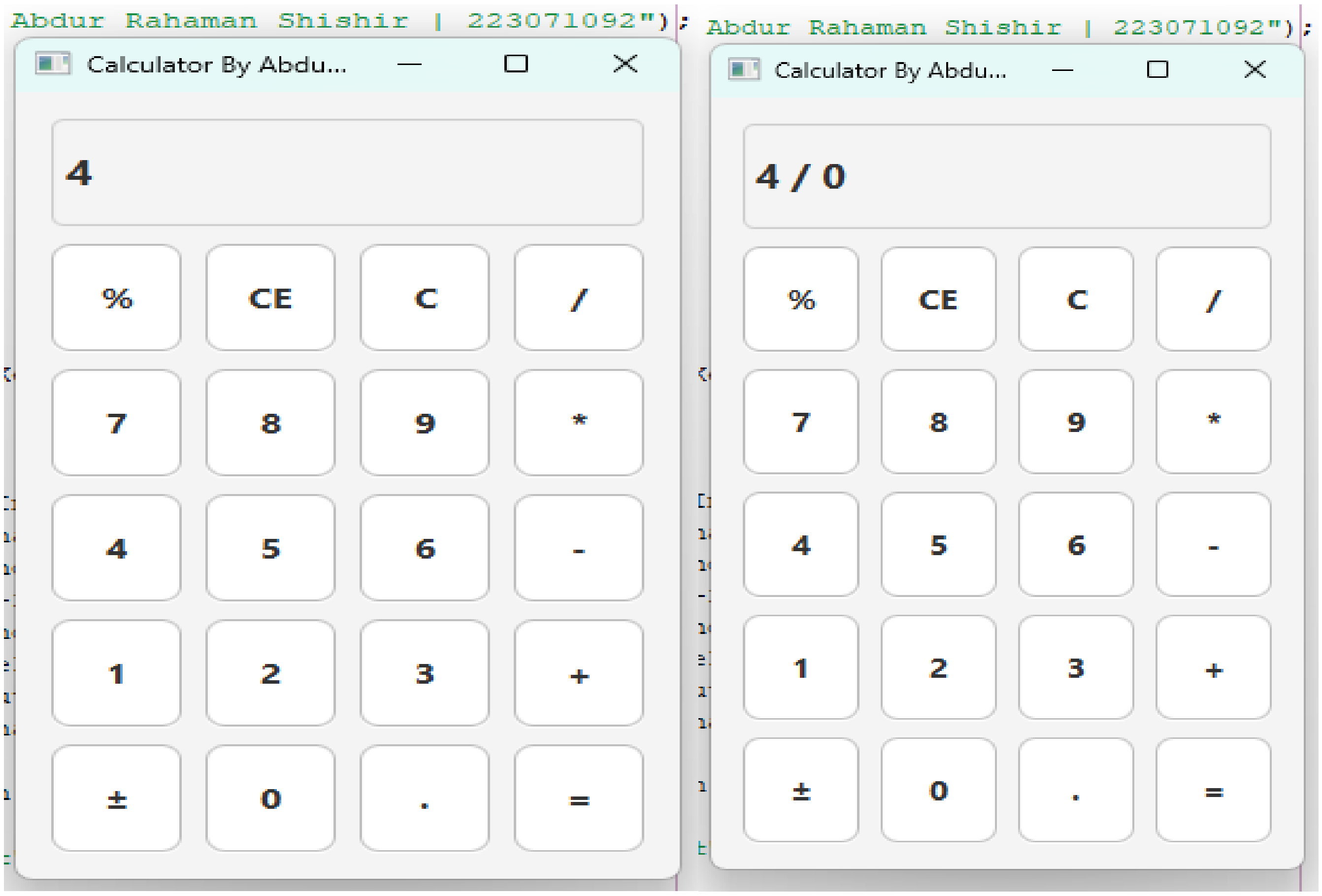
- Each button is assigned an action event handler using **setOnAction**, which calls a centralized method **handleInput()** to process logic based on the input.
- For keyboard input, **setOnKeyPressed** is used on the scene to detect and map key events to corresponding calculator functions.
- This centralized approach simplifies input processing and ensures consistent behavior across both input methods.

UI Layout (With Visual Represent Pictures):



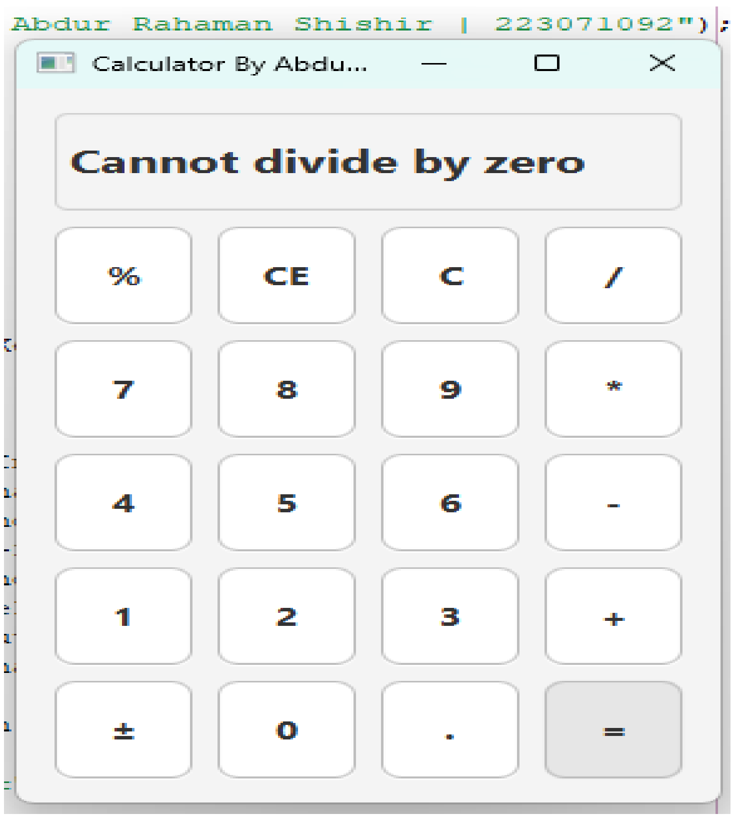
```
Abdur Rahaman Shishir | 223071092");
```

```
Abdur Rahaman Shishir | 223071092");
```

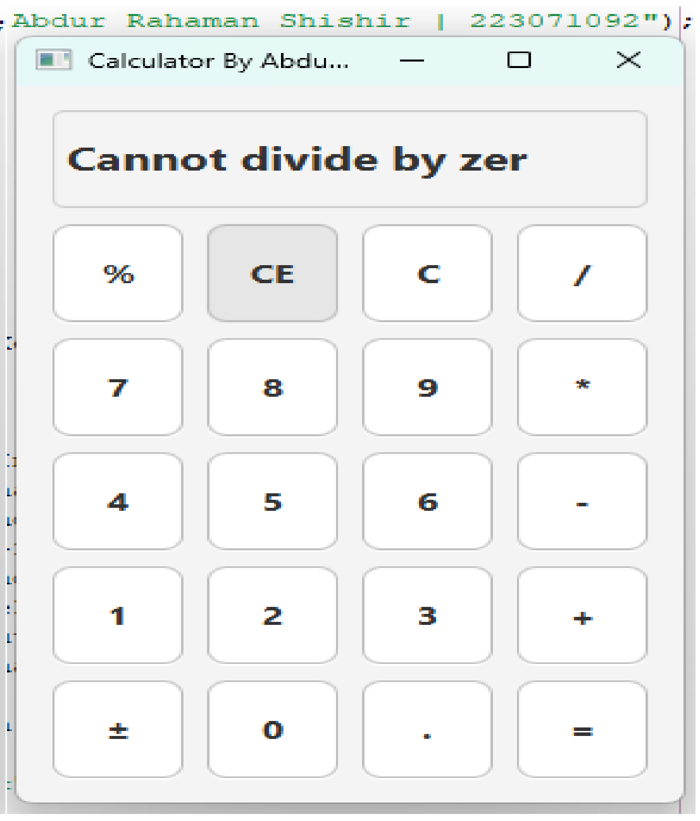


```
Abdur Rahaman Shishir | 223071092");
```

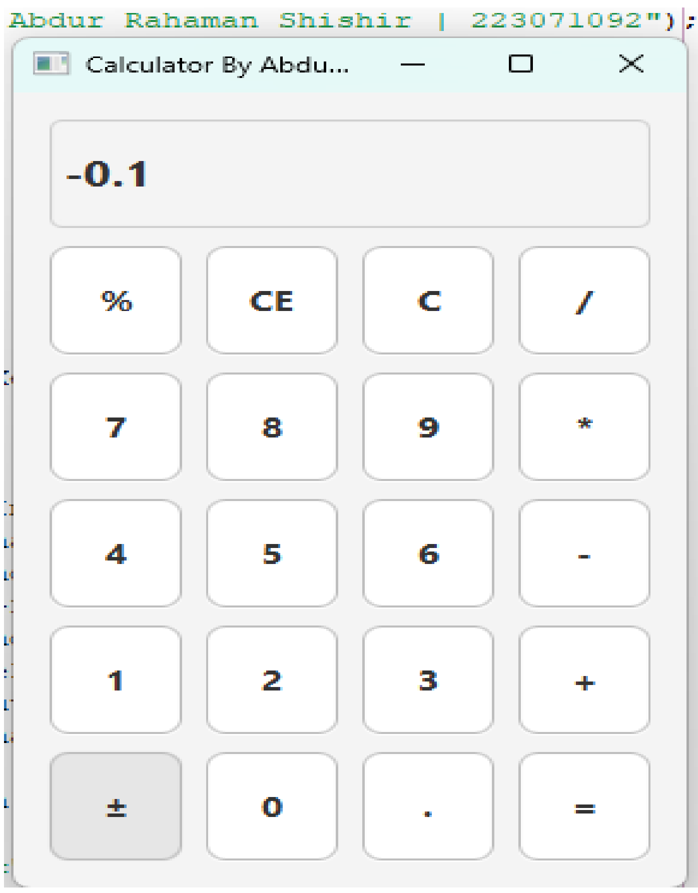
```
Abdur Rahaman Shishir | 223071092");
```


```
Abdur Rahaman Shishir | 223071092");
```
Calculator By Abdu...

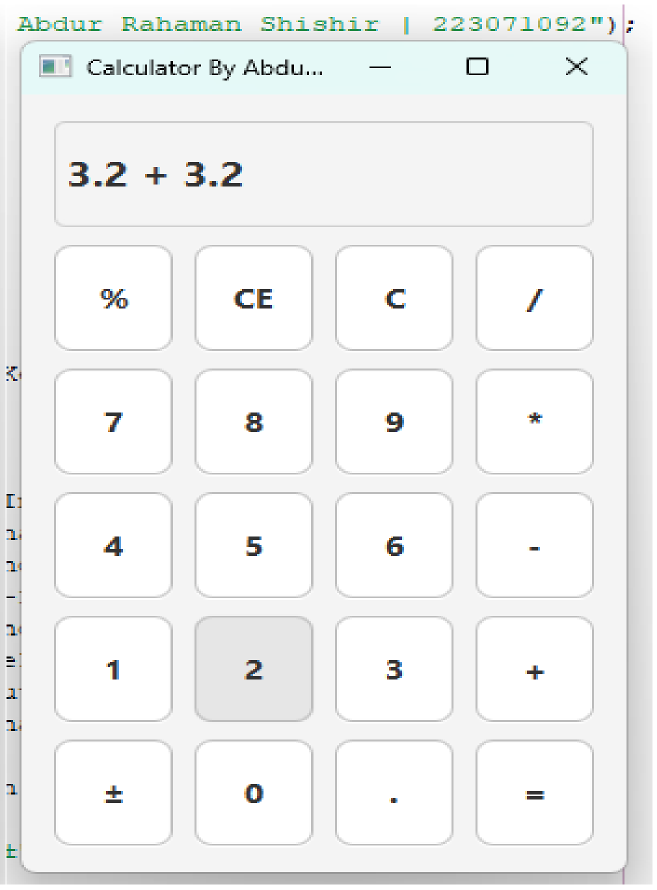
```
Cannot divide by zero
```


```
Abdur Rahaman Shishir | 223071092");
```
Calculator By Abdu...

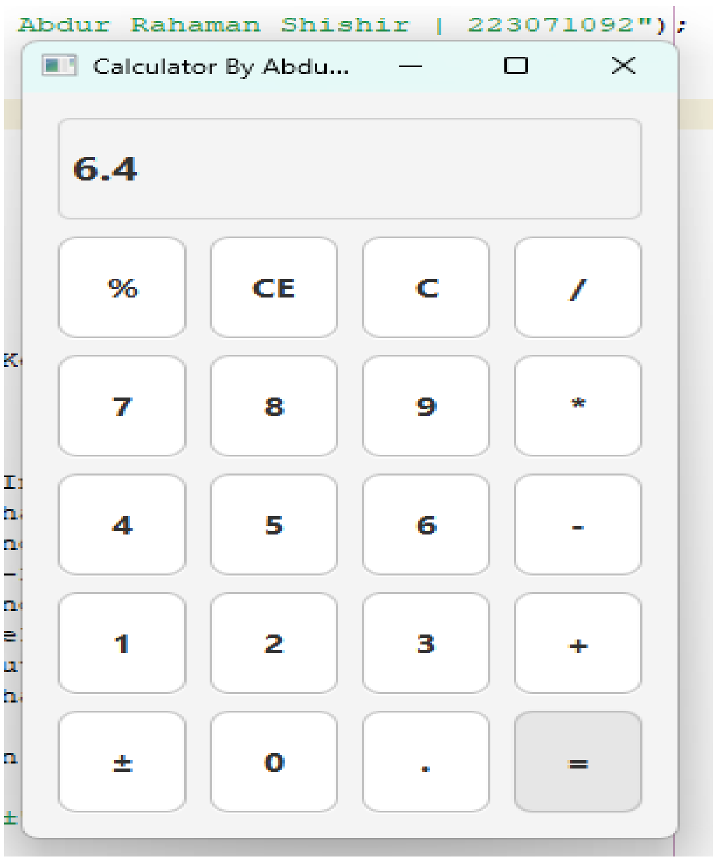
```
Cannot divide by zer
```


```
Abdur Rahaman Shishir | 223071092");
```
Calculator By Abdu...

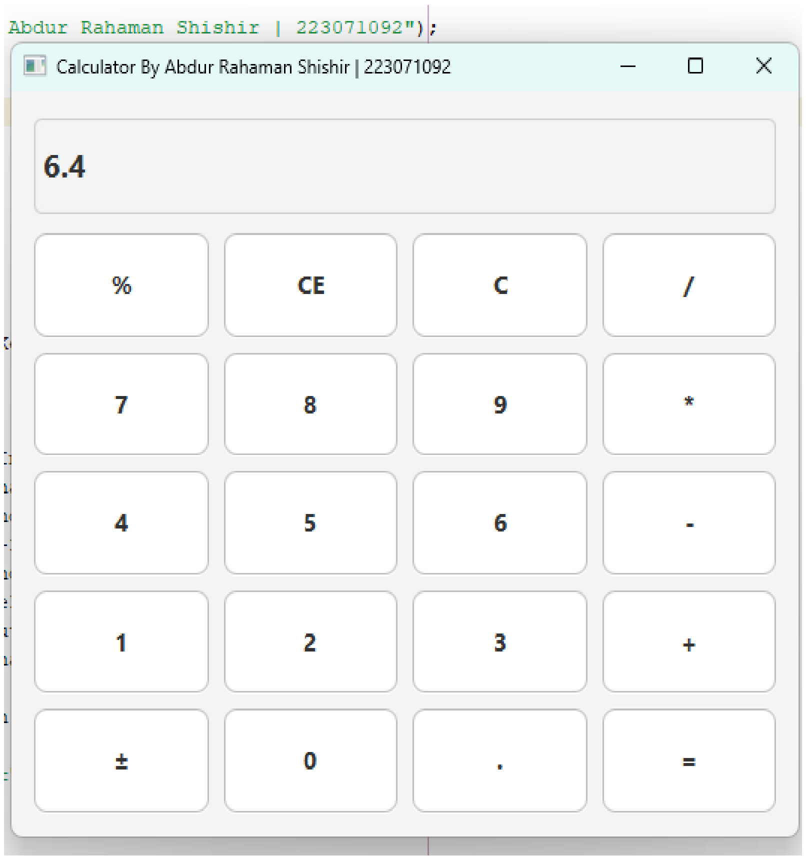
```
-0.1
```


```
Abdur Rahaman Shishir | 223071092");
```
Calculator By Abdu...

```
3.2 + 3.2
```



```
Abdur Rahaman Shishir | 223071092");
```



```
Abdur Rahaman Shishir | 223071092");
```

Classes:

public void start(Stage primaryStage) { //...104 lines }

private RowConstraints createRowConstraints() { //...6 lines }

private ColumnConstraints createColumnConstraints() { //...6 lines }

private void handleInput(String input) { //...129 lines }

private String formatNumber(double value) { //...3 lines }

public static void main(String[] args) { //...3 lines }

6. Implementation Details:

This calculator application focuses on simplicity, responsiveness, and accuracy. Below are the key implementation aspects of the code:

Mathematical Operations:

- ✓ Basic operations **(+, -, *, /)** are implemented within the **handleInput()** method.
- ✓ The application stores the first operand **(num1)** and the selected operator when the user presses an operation button.
- ✓ On pressing =, it parses the second operand from the display, performs the calculation using a **switch** expression, and updates the display with the result.

Input / Output Handling:

Input:

- ❖ Mouse: Each button is linked to the **handleInput(String input)** method using **setOnAction**.
- ❖ Keyboard: The Scene listens for key presses using **setOnKeyPressed**, mapping keys like **(digits, +, -, Enter, etc.)**, to their corresponding actions.

Output:

- ❖ The **TextField** named display shows current input, ongoing operations, and results. It is styled, non-editable, and updated programmatically based on user actions.

Error Handling:

Handles division by zero explicitly with:

```
if (operator.equals("/") && num2 == 0)
    {
        display.setText("Cannot divide by zero");
    }
```

Also catches invalid inputs and number formatting errors using try-catch blocks to display "Error" if an exception occurs.

Styling and CSS:

- ✓ Buttons and display are styled directly via inline CSS in **setStyle():**
- ✓ Font size and weight for better visibility.
- ✓ Background and border colors for a modern, minimal look.
- ✓ Rounded corners and hover effects using **setOnMouseEntered / setOnMouseExited**.
- ✓ Layout is made responsive with **Priority.ALWAYS** settings for columns and rows, ensuring the UI scales properly.

7. Testing and Evaluation:

The calculator application was tested extensively to ensure correct functionality, responsiveness, and error handling. Both typical and edge-case scenarios were considered to validate its reliability.

Input Testing:

- A wide range of numerical and operational inputs were manually entered to confirm accurate calculations.
- Chained operations **(e.g., 5 + 3 =, then * 2 =)** were tested to verify correct result retention and continuation.
- The C and CE buttons were tested to ensure they clear the display or remove the last character, respectively.

Keyboard & Mouse Input Testing:

- Every button was clicked using the mouse to ensure each triggers the correct action.
- Keyboard inputs were tested using number keys, operation keys **(+, -, *, /),** Enter, Backspace, and Delete.
- Special keys like **Shift + 5 (for %) and F1 (for ±)** were also validated.

Edge Case Testing:

- **Multiple decimal points**: Input such as **5.3.2** is prevented; only one decimal is allowed per number segment.
- **Division by zero**: Proper error message **"Cannot divide by zero"** is displayed without crashing.
- **Empty input operations**: Pressing operations on an empty or invalid display doesn't cause unexpected behavior.
- **Toggle sign (±)**: Confirmed that it negates the correct operand, whether before or after selecting an operator.
- **Backspace (CE)**: Validated that it only removes one character at a time without affecting calculation logic.

8. Challenges and Solutions:

Logic Issues (Bugs & Fixes):

Problems:

- Pressing = did not reset state, allowing new input to append to the result.
- Multiple consecutive operators **(e.g., 5 + - *)** caused invalid expressions.
- Users could input multiple decimal points in a single number, leading to **NumberFormatException**.
- Division by zero resulted in crashes or unintended behavior.
- **CE** and **C** behaved inconsistently during active operations or after errors.

Solutions:

- Introduced **startNewNumber** and **justCalculated** flags to manage display clearing after results.
- Replaced existing operators when a new one is entered instead of allowing multiple.
- Checked the number segment for existing decimals before allowing a new ..
- Implemented specific handling for division by zero with a clear message: **"Cannot divide by zero".**
- Clearly separated **C (clear all)** and **CE (clear last entry)** logic for expected behavior.

Layout / UI Issues (Problems & Fixes)

Problems:

- Button layout was not scaling properly on different screen sizes.
- Lack of hover effects made buttons feel unresponsive or static.
- Keyboard input occasionally behaved differently from mouse input.
- Layout became misaligned during resizing or switching screens.

Solutions:

- Used **GridPane** with **RowConstraints** and **ColumnConstraints**, applying **Priority.ALWAYS** for flexible resizing.
- Added hover effects using **setOnMouseEntered** and **setOnMouseExited** for better visual feedback.
- Unified keyboard and mouse input logic by routing both through the same handleInput() method.
- Fixed UI alignment using consistent padding, spacing, and alignment properties **(Pos.CENTER).**

9. Testing and Evaluation:

Developing this JavaFX calculator project provided valuable hands-on experience with building GUI applications using Java and object-oriented programming principles. Through this process, I gained deeper understanding of event handling, user interface design, and managing application state for interactive tools.

The calculator is functionally complete, supporting basic arithmetic operations, percentage calculations, sign toggling, error handling (e.g., division by zero), and both mouse and keyboard input. The application is user-friendly and styled with simple, clean UI enhancements.

Future enhancements could include:

- ❖ Adding support for parentheses and operation precedence.
- ❖ Implementing a history or memory function.
- ❖ Enhancing UI responsiveness for different screen sizes or resolutions.

10. References:

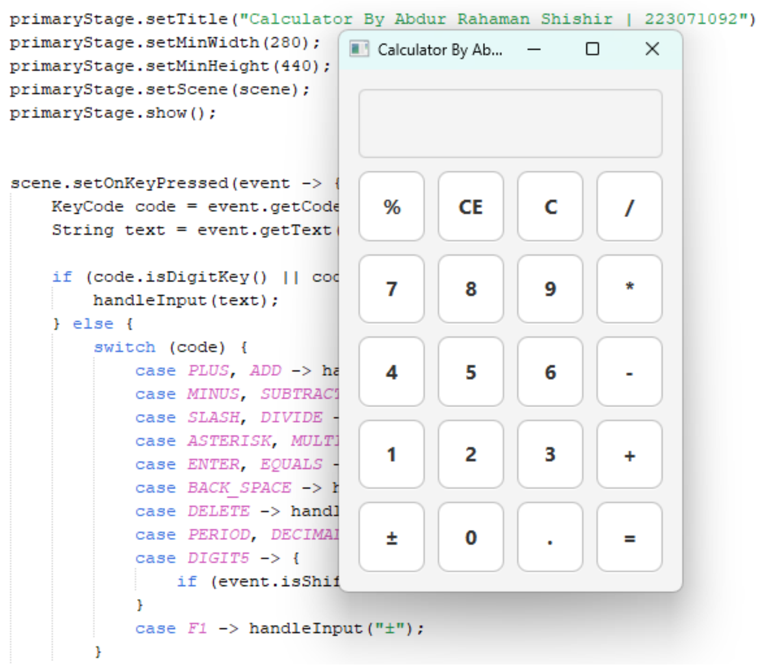
- Oracle JavaFX Documentation
  https://openjfx.io/
  Official documentation and API reference for JavaFX.
- JavaFX Tutorial by CodeGym
  https://codegym.cc/groups/posts/javafx-tutorial
  Helpful for understanding layout panes and event handling.
- Stack Overflow
  https://stackoverflow.com/
  Used for resolving specific issues like keyboard event mapping and operator formatting.
- GeeksforGeeks – JavaFX Basics
  https://www.geeksforgeeks.org/javafx/
  Provided foundational knowledge on JavaFX components and application structure.
- CSS Styling in JavaFX
  https://docs.oracle.com/javafx/2/api/javafx/scene/doc-files/cssref.html
  Used for enhancing the look and feel of the calculator UI.

11. Appendix:
Full Code:

```
package idontknow;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import static javafx.scene.input.KeyCode.DECIMAL;
import static javafx.scene.input.KeyCode.PERIOD;
import javafx.scene.layout.ColumnConstraints;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.RowConstraints;
import javafx.stage.Stage;

public class Idontknow extends Application {

    private final TextField display = new TextField();
    private double num1 = 0;
    private String operator = "";
    private boolean startNewNumber = false;
    private boolean justCalculated = false;

    @Override
    public void start(Stage primaryStage) {

        display.setEditable(false);
        display.setStyle("""
            -fx-font-size: 20px;
            -fx-font-weight: bold;
            -fx-background-color: #f4f4f4;
            -fx-border-color: #cccccc;
            -fx-border-radius: 5px;
            -fx-padding: 5px;
        """);
        display.setPrefHeight(60);
        display.setMaxWidth(Double.MAX_VALUE);
        GridPane.setHgrow(display, Priority.ALWAYS);

        GridPane grid = new GridPane();
        grid.setPadding(new Insets(15));
        grid.setHgap(10);
        grid.setVgap(10);
        grid.setAlignment(Pos.CENTER);
        grid.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);

        String[] buttons = {
            "%", "CE", "C", "/",
            "7", "8", "9", "*",
            "4", "5", "6", "-",
            "1", "2", "3", "+",
            "±", "0", ".", "="
         };

        int row = 1;
        int col = 0;

        for (String text : buttons) {
            Button btn = new Button(text);
            btn.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
            btn.setStyle("""
                -fx-font-size: 16px;
                -fx-font-weight: bold;
                -fx-background-color: #ffffff;
                -fx-border-radius: 8px;
                -fx-background-radius: 8px;
                -fx-border-color: #bbbbbb;
                -fx-cursor: hand;
            """);

            btn.setOnMouseEntered(e -> btn.setStyle(btn.getStyle().replace("#ffffff",
"#e6e6e6")));
            btn.setOnMouseExited(e -> btn.setStyle(btn.getStyle().replace("#e6e6e6",
"#ffffff")));

            btn.setOnAction(e -> handleInput(text));

            GridPane.setHgrow(btn, Priority.ALWAYS);
            GridPane.setVgrow(btn, Priority.ALWAYS);
            grid.add(btn, col, row);
            col++;
            if (col > 3) {
                col = 0;
                row++;
            }
        }
        grid.add(display, 0, 0, 4, 1);

        for (int i = 0; i < 4; i++) {
            grid.getColumnConstraints().add(createColumnConstraints());
        }
        for (int i = 0; i < 6; i++) {  // 1 for display, 5 for button rows
            grid.getRowConstraints().add(createRowConstraints());
        }

        Scene scene = new Scene(grid, 280, 440);


        primaryStage.setTitle("Calculator By Abdur Rahaman Shishir | 223071092");
        primaryStage.setMinWidth(280);
        primaryStage.setMinHeight(440);
        primaryStage.setScene(scene);
        primaryStage.show();


        scene.setOnKeyPressed(event -> {
            KeyCode code = event.getCode();
            String text = event.getText();

            if (code.isDigitKey() || code.isKeypadKey()) {
                handleInput(text);
            } else {
                switch (code) {
                    case PLUS, ADD -> handleInput("+");
                    case MINUS, SUBTRACT -> handleInput("-");
                    case SLASH, DIVIDE -> handleInput("/");
                    case ASTERISK, MULTIPLY -> handleInput("*");
                    case ENTER, EQUALS -> handleInput("=");
                    case BACK_SPACE -> handleInput("CE");
                    case DELETE -> handleInput("C");
                    case PERIOD, DECIMAL -> handleInput(".");
                    case DIGIT5 -> {
                        if (event.isShiftDown()) handleInput("%");
                    }
                    case F1 -> handleInput("±");
                }
            }
        });
    }

    private RowConstraints createRowConstraints() {
        RowConstraints rowConstraints = new RowConstraints();
        rowConstraints.setPercentHeight(100.0 / 6); // 6 rows: 1 for display, 5 for buttons
        rowConstraints.setVgrow(Priority.ALWAYS);
        return rowConstraints;
    }

    private ColumnConstraints createColumnConstraints() {
        ColumnConstraints columnConstraints = new ColumnConstraints();
        columnConstraints.setPercentWidth(25); // 4 columns
        columnConstraints.setHgrow(Priority.ALWAYS);
        return columnConstraints;
    }

    private void handleInput(String input) {
        switch (input) {
            case "C" -> {
                display.clear();
                num1 = 0;
                operator = "";
                startNewNumber = false;
                justCalculated = false;
            }
            case "CE" -> {
                String currentText = display.getText();
                if (!currentText.isEmpty()) {
                    display.setText(currentText.substring(0, currentText.length() - 1));
                }
            }
            case "%" -> {
                try {
                    String text = display.getText();
                    if (operator.isEmpty()) {
                        double value = Double.parseDouble(text);
                        value = value / 100;
                        display.setText(formatNumber(value));
                    } else {
                        String[] parts = text.split(" \\" + operator + " ");
                        if (parts.length == 2) {
                            double value = Double.parseDouble(parts[1]);
                            value = value / 100;
                            display.setText(parts[0] + " " + operator + " " + formatNumber(value));
                        }
                    }
                } catch (NumberFormatException e) {
                    display.setText("Error");
                }
            }
```

```
            case "±" -> {
                String text = display.getText();
                if (text.equals("Cannot divide by zero") || text.equals("Error") || text.isEmpty())
{
                    return;
                }

                try {
                    if (operator.isEmpty()) {
                        double value = Double.parseDouble(text);
                        value *= -1;
                        display.setText(formatNumber(value));
                    } else {
                        String[] parts = text.split(" \\" + operator + " ");
                        if (parts.length == 2) {
                            double value = Double.parseDouble(parts[1]);
                            value *= -1;
                            display.setText(parts[0] + " " + operator + " " + formatNumber(value));
                        }
                    }
                } catch (NumberFormatException e) {
                    display.setText("Error");
                }
            }
            case "=" -> {
                if (!operator.isEmpty()) {
                    String[] parts = display.getText().split(" \\" + operator + " ");
                    if (parts.length == 2) {
                        try {
                            double num2 = Double.parseDouble(parts[1]);
                            if (operator.equals("/") && num2 == 0) {
                                display.setText("Cannot divide by zero");
                            } else {
                                double result = switch (operator) {
                                    case "+" -> num1 + num2;
                                    case "-" -> num1 - num2;
                                    case "*" -> num1 * num2;
                                    case "/" -> num1 / num2;
                                    default -> 0;
                                };
                                String resultText = (result % 1 == 0) ? String.valueOf((int) result) :
String.valueOf(result);
                                display.setText(resultText);
                                num1 = result;
                                justCalculated = true;
                            }
                            operator = "";
                            startNewNumber = true;
                        } catch (NumberFormatException e) {
                            display.setText("Error");
                        }
                    }
                }
            }
            case "+", "-", "*", "/" -> {
                try {
                    String text = display.getText();

                    if (justCalculated) {
                        operator = input;
                        display.setText(formatNumber(num1) + " " + operator + " ");
                        justCalculated = false;
                        startNewNumber = false;
                    } else if (!text.isEmpty()) {
                        if (!operator.isEmpty() && text.matches(".* [\\+\\-\\*/] $")) {
                            display.setText(text.substring(0, text.length() - 3) + " " + input + " ");
                            operator = input;
                        } else if (operator.isEmpty()) {
                            num1 = Double.parseDouble(text);
                            operator = input;
                            display.setText(formatNumber(num1) + " " + operator + " ");
                        }
                    }
                } catch (NumberFormatException e) {
                    display.setText("Error");
                }
            }
            default -> {
                if (startNewNumber || display.getText().equals("Cannot divide by zero") ||
display.getText().equals("Error")) {
                    display.clear();
                    startNewNumber = false;
                }

                if (input.equals(".")) {
                    String text = display.getText();
                    if (operator.isEmpty()) {
                        if (text.contains(".")) return;
                    } else {
                        String[] parts = text.split(" \\" + operator + " ");
                        if (parts.length == 2 && parts[1].contains(".")) return;
                    }
                }

                display.appendText(input);
            }
        }
    }

    private String formatNumber(double value) {
        return (value % 1 == 0) ? String.valueOf((int) value) : String.valueOf(value);
    }

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

Application Overview:



```
primaryStage.setTitle("Calculator By Abdur Rahaman Shishir | 223071092")
primaryStage.setMinWidth(280);
primaryStage.setMinHeight(440);
primaryStage.setScene(scene);
primaryStage.show();


scene.setOnKeyPressed(event -> {
    KeyCode code = event.getCode
    String text = event.getText(

    if (code.isDigitKey() || coc
        handleInput(text);
    } else {
        switch (code) {
            case PLUS, ADD -> ha
            case MINUS, SUBTRACT
            case SLASH, DIVIDE -
            case ASTERISK, MULTI
            case ENTER, EQUALS -
            case BACK_SPACE -> h
            case DELETE -> handl
            case PERIOD, DECIMAL
            case DIGIT5 -> {
                if (event.isShif
            }
            case F1 -> handleInput("±");
        }
```