Desktop Style Calculator Documentation

> A Java Swing-based calculator with graphical user interface for basic arithmetic operations.

1. Overview

This is a **Java Swing-based calculator** that provides a graphical user interface (GUI) for performing basic arithmetic operations. It supports:

- **Standard operations**: Addition (`+`), Subtraction (`-`), Multiplication (`*`), Division (`/`)
- **Parentheses**: `(` and `)` for grouping operations
- **Decimal numbers**: Supports floating-point calculations
- **Editing**: Backspace (`←`) and Clear (`C`) functionality

2. Features

2.1. User Interface

Display: A text field at the top shows the input and results.

Buttons:

- **Digits**: `0-9`
- **Operators**: `+`, `-`, `*`, `/`
- **Special Functions**:
  - `(` and `)` for grouping
  - `.` for decimal input
  - `=` to evaluate the expression
  - `C` to clear the display

    - `←` to delete the last character

2.2. Mathematical Operations

- Follows **standard operator precedence** (PEMDAS/BODMAS rules):
    - Parentheses first
    - Multiplication & Division (left to right)
    - Addition & Subtraction (left to right)
- Handles **floating-point numbers** (e.g., `3.5 + 2.1 = 5.6`)
- **Error Handling**:
    - Displays `"Error"` for invalid expressions
    - Prevents division by zero

3. How It Works

3.1. Expression Evaluation

The calculator uses a **stack-based algorithm** to parse and evaluate mathematical expressions:

1. **Tokenization**: Processes numbers and operators.
2. **Operator Precedence**: Ensures `*` and `/` are evaluated before `+` and `-`.
3. **Parentheses Handling**: Evaluates nested expressions first.

3.2. Key Methods

Method	Description
`evaluate(String expr)`	Parses and computes the result of an arithmetic expression
`isOperator(char c)`	Checks if a character is `+`, `-`, `*`, or `/`
`hasPrecedence(op1, op2)`	Determines if `op1` has higher precedence than `op2`
`applyOperation(op, a, b)`	Performs the arithmetic operation (`+`, `-`, `*`, `/`)

4. Usage

4.1. Running the Calculator

Compile & Run:

```
javac DesktopStyleCalculator.java
java DesktopStyleCalculator
```

Interact with the GUI:

- Enter numbers and operators using buttons.
- Press `=` to compute the result.
- Use `C` to clear or `←` to correct mistakes.

4.2. Example Calculations

Input	Output
`3 + 5 * 2`	`13`
`(3 + 5) * 2`	`16`
`10 / 3`	`3.333...`
`5 / 0`	`Error`

5. Limitations

- Does not support advanced functions (e.g., `sin`, `log`, `^`).
- No memory (M+, M-, MR) features.
- Limited error messages (generic `"Error"` display).

6. Future Improvements

- Add **scientific functions** (e.g., square root, exponents).
- Implement **memory storage** (store/recall values).
- Improve error messages (e.g., `"Syntax Error"`, `"Division by Zero"`).

7. Conclusion

This calculator provides a **simple, functional GUI** for basic arithmetic. It is built with **Java Swing** and avoids external dependencies by using a **custom expression evaluator**.

Source Code

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Stack;

public class DesktopStyleCalculator {
    private JTextField display;

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new DesktopStyleCalculator().c
    }

    public void createUI() {
        JFrame frame = new JFrame("Calculator");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(350, 450);
        frame.setResizable(false);

        display = new JTextField();
        display.setFont(new Font("Segoe UI", Font.BOLD, 28));
        display.setHorizontalAlignment(JTextField.RIGHT);
        display.setEditable(false);
        display.setPreferredSize(new Dimension(350, 60));

        JPanel buttonPanel = new JPanel(new GridLayout(5, 4, 5, 5));
        String[] buttons = {
            "7", "8", "9", "/",
            "4", "5", "6", "*",
            "1", "2", "3", "-",
            "0", ".", "=", "+",
            "C", "(", ")", "←"
        };

        for (String text : buttons) {
            JButton btn = createButton(text);
            buttonPanel.add(btn);
        }

        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS))
        mainPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10,
        mainPanel.add(display);
        mainPanel.add(Box.createRigidArea(new Dimension(0, 10)));
        mainPanel.add(buttonPanel);

        frame.setContentPane(mainPanel);
        frame.setVisible(true);
    }

    private JButton createButton(String text) {
        JButton button = new JButton(text);
        button.setFont(new Font("Segoe UI", Font.BOLD, 20));
        button.setFocusPainted(false);
        button.setBackground(Color.WHITE);
        button.setForeground(Color.BLACK);
        button.setPreferredSize(new Dimension(80, 60));

        button.addActionListener(e -> handleClick(text));
        return button;
    }

    private void handleClick(String text) {
        if (text.equals("C")) {
            display.setText("");
        } else if (text.equals("←")) {
            String current = display.getText();
            if (!current.isEmpty()) {
                display.setText(current.substring(0, current.length() -
            }
        } else if (text.equals("=")) {
            evaluateExpression();
        } else {
            display.setText(display.getText() + text);
        }
    }

    private void evaluateExpression() {
        String expr = display.getText();
        try {
            if (expr.isEmpty()) {
                display.setText("Empty");
                return;
            }
            
            double result = evaluate(expr);
            display.setText(String.valueOf(result));
        } catch (Exception e) {
            display.setText("Error");
        }
    }
    
    // Custom expression evaluator for basic arithmetic
    private double evaluate(String expression) {
        // Remove all whitespace
        expression = expression.replaceAll("\\s+", "");
        
        Stack numbers = new Stack<>();
        Stack operators = new Stack<>();
        
        for (int i = 0; i < expression.length(); i++) {
            char c = expression.charAt(i);
            
            if (c == ' ') {
                continue;
            }
            
            if (c == '(') {
                operators.push(c);
            } else if (c == ')') {
                while (operators.peek() != '(') {
                    numbers.push(applyOperation(operators.pop(), number
                }
                operators.pop();
            } else if (isOperator(c)) {
                while (!operators.empty() && hasPrecedence(c, operators
                    numbers.push(applyOperation(operators.pop(), number
                }
                operators.push(c);
            } else {
                StringBuilder sb = new StringBuilder();
                while (i < expression.length() && (Character.isDigit(ex
                    sb.append(expression.charAt(i++));
                }
                i--;
                numbers.push(Double.parseDouble(sb.toString()));
            }
        }
        
        while (!operators.empty()) {
            numbers.push(applyOperation(operators.pop(), numbers.pop(),
        }
        
        return numbers.pop();
    }
    
    private boolean isOperator(char c) {
        return c == '+' || c == '-' || c == '*' || c == '/';
    }

    private boolean hasPrecedence(char op1, char op2) {
        if (op2 == '(' || op2 == ')') {
            return false;
        }
        if ((op1 == '*' || op1 == '/') && (op2 == '+' || op2 == '-')) {
            return false;
        }
        return true;
    }

    private double applyOperation(char op, double b, double a) {
        switch (op) {
            case '+': return a + b;
            case '-': return a - b;
            case '*': return a * b;
            case '/':
                if (b == 0) throw new ArithmeticException("Division by
                return a / b;
        }
        return 0;
    }
}
```