Report On

Calculator

Submitted in partial fulfillment of the requirements of the Course project in Semester III of Second Year Artificial Intelligence and Data Science

by
Rohan Ganpat Mangaonkar (Roll No. 27)
Tarun Premnarayan Pathak (Roll No. 38)
Aryan Mohan Parab (Roll No. 36)

Supervisor
Prof. Sneha Yadav

University of Mumbai

Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science

(2023-24)

Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science

CERTIFICATE

This is to certify that the project entitled “Eiffil Tower” is a bonafide work of "Rohan G Mangaonkar (Roll No. 27), Tarun Premnarayan Pathak (Roll No. 38), Aryan Mohan Parab (Roll No. 36) " submitted to the University of Mumbai in partial fulfillment of the requirement for the Course project in semester III of Second Year Artificial Intelligence and Data Science engineering.

Supervisor

Prof. Sneha Yadav

Dr. Tatwadarshi P. N.
Head of Department

Table of Contents

Pg. No

ABSTRACT

The provided Java code represents a comprehensive graphical calculator application, designed to offer users a wide range of mathematical capabilities with a user-friendly interface. The code utilizes Java's Swing library to create the graphical elements of the calculator. Here's a more detailed breakdown of the key features and components: Calculator Types and Themes: The calculator has two distinct modes: "Standard" and "Scientific," catering to both basic arithmetic needs and more advanced mathematical functions. Users can switch between these two modes using a dropdown menu.

Additionally, the calculator provides three visual themes for customization: "Simple," "Colored," and "DarkTheme." Users can select their preferred theme, giving the calculator a unique and personalized look.

Numeric Input and Display: The calculator's main input and display area is a JTextField, where users can enter numeric values and view the results of their calculations. The code ensures that the input is properly formatted and that it responds to various numeric inputs. Control Buttons: The calculator includes various control buttons, such as "C" (Clear), "<-" (Backspace), and "%" (Percentage). These buttons allow users to manage their inputs and perform actions like clearing the input, deleting the last character, and calculating percentages.

Arithmetic Operators: The calculator supports standard arithmetic operators, including addition, subtraction, multiplication, division, and the equals sign. Users can input numerical values and perform basic calculations with ease.

Scientific Functions: In "Scientific" mode, the calculator provides additional mathematical functions, including square root (√), exponentiation (pow), and natural logarithm (ln). These functions are accessible to users who require more advanced mathematical operations.

Button Click Handling: The code defines action listeners for each button, ensuring that when a button is clicked, the appropriate action is taken. This includes validating inputs, performing calculations, and updating the display.

Calculator Logic: The calc method is responsible for performing the actual arithmetic calculations based on the operator and operands. It handles addition, subtraction, multiplication, division, percentage calculations, exponentiation, and natural logarithms, depending on the selected operator.

Dynamic GUI Appearance: The code dynamically adjusts the GUI's appearance based on the selected theme. This involves changing background colors, button colors, and text colors to match the chosen theme, providing a visually appealing and customizable user experience. Error Handling: The code ensures that input validation and error handling are in place to prevent unexpected behavior and calculations.

User Interaction: The user interacts with the calculator by clicking on buttons, switching calculator types and themes, and observing the real-time updates on the display. In summary, this calculator application is a versatile and user-friendly tool that caters to both basic and advanced mathematical needs. It offers the flexibility to switch between different calculator types and themes, making it adaptable to users with various preferences and requirements. The code is well-organized, providing a clear and functional interface for performing a wide range of mathematical calculations with ease.

CODE

```
import java.awt.Cursor; import
java.awt.Font; import
java.awt.event.ActionListener; import
java.awt.event.ItemEvent; import
java.util.function.Consumer; import
java.util.regex.Pattern; import
java.awt.Color; import javax.swing.*;
import java.lang.Math;
public class Calculator {

    private static final int WINDOW_WIDTH =
410;     private static final int WINDOW_HEIGHT
= 600;     private static final int
BUTTON_WIDTH = 80;     private static final int
BUTTON_HEIGHT = 70;     private static final int
MARGIN_X = 20;     private static final int
MARGIN_Y = 60;

    private JFrame window; // Main window     private
JComboBox<String> comboCalcType, comboTheme;
    private JTextField inText; // Input
    private JButton btnC, btnBack, btnMod, btnDiv, btnMul, btnSub, btnAdd,
btn0, btn1, btn2, btn3, btn4, btn5, btn6, btn7, btn8, btn9,             btnPoint,
btnEqual, btnRoot, btnPower, btnLog;

    private char opt = ' '; // Save the operator     private boolean go =
true; // For calculate with Opt != (=)     private boolean addWrite =
true; // Connect numbers in display     private double val = 0; // Save
the value typed for calculation

    /*
        Mx Calculator:
X = Row
        Y = Column

        +-------------------+
        |  +-----------+  |  y[0]
        |  |           |  |
        |  +-----------+  |
        |                 |
        |  C  <-  %  /  |  y[1]
        |  7   8   9  *  |  y[2]
        |  4   5   6  -  |  y[3]
        |  1   2   3  +  |  y[4]
        |  .   0     =     |  y[5]
+-------------------+
x[0] x[1] x[2] x[3]     */

    /*
        +-------------------+
        |  +-----------+  |  y[0]
        |  |           |  |
        |  +-----------+  |
        |                 |
        |  0   1   1   3  |  y[1]
        |  4   5   6   7  |  y[2]
        |  8   9   10  11 |  y[3]
        |  12  13  14  15 |  y[4]
        |  16  17    18   |  y[5]
+-------------------+       x[0]
x[1] x[2] x[3]

    */

    public Calculator() {
        window = new JFrame("Calculator");
        window.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
window.setLocationRelativeTo(null); // Move window to center

        comboTheme = initCombo(new String[]{"Simple", "Colored", "DarkTheme"}, 230, 30,
"Theme", themeSwitchEventConsumer);

        comboCalcType = initCombo(new String[]{"Standard", "Scientific"}, 20, 30, "Calculator
type", calcTypeSwitchEventConsumer);

        int[] x = {MARGIN_X, MARGIN_X + 90, 200, 290, 380};
        int[] y = {MARGIN_Y, MARGIN_Y + 100, MARGIN_Y + 180, MARGIN_Y + 260,
MARGIN_Y + 340, MARGIN_Y + 420};

        inText = new JTextField("0");
inText.setBounds(x[0], y[0], 350, 70);
inText.setEditable(false);
inText.setBackground(Color.WHITE);
        inText.setFont(new Font("Comic Sans MS", Font.PLAIN, 33));
window.add(inText);

        btnC = initBtn("C", x[0], y[1], event -> {
            repaintFont();
            inText.setText("0");
opt = ' ';            val = 0;
        });
        btnBack = initBtn("<-", x[1], y[1], event -> {
repaintFont();
            String str = inText.getText();
            StringBuilder str2 = new StringBuilder();
for (int i = 0; i < (str.length() - 1); i++) {
str2.append(str.charAt(i));
            }
            if (str2.toString().equals("")) {
inText.setText("0");
            } else {
                inText.setText(str2.toString());
            }
        });

        btnMod = initBtn("%", x[2], y[1], event -> {
            repaintFont();
            if (Pattern.matches("([-]?\\d+[.]\\d*)|(\\d+)", inText.getText()))
if (go) {
                    val = calc(val, inText.getText(), opt);
                    if (Pattern.matches("[-]?[\\d]+[.][0]*", String.valueOf(val))) {
inText.setText(String.valueOf((int) val));
                    } else {
                        inText.setText(String.valueOf(val));
                    }
opt = '%';
go = false;
                    addWrite = false;
                }
        });

        btnDiv = initBtn("/", x[3], y[1], event -> {
            repaintFont();
            if (Pattern.matches("([-]?\\d+[.]\\d*)|(\\d+)", inText.getText()))
if (go) {
                    val = calc(val, inText.getText(), opt);
                    if (Pattern.matches("[-]?[\\d]+[.][0]*", String.valueOf(val))) {
inText.setText(String.valueOf((int) val));
                    } else {
                        inText.setText(String.valueOf(val));
                    }
opt = '/';                    go =
false;                    addWrite
= false;
                } else {
opt = '/';
                }
        });
        btn7 = initBtn("7", x[0], y[2], event -> {
            repaintFont();
if (addWrite) {
                if (Pattern.matches("[0]*", inText.getText())) {
                    inText.setText("7");
                } else {
                    inText.setText(inText.getText() + "7");
                }
            } else {
inText.setText("7");
                addWrite = true;
            }
            go = true;
        });

        btn8 = initBtn("8", x[1], y[2], event -> {
            repaintFont();
if (addWrite) {
                if (Pattern.matches("[0]*", inText.getText())) {
                    inText.setText("8");
                } else {
                    inText.setText(inText.getText() + "8");
                }
            } else {
inText.setText("8");
                addWrite = true;
            }
            go = true;
        });

        btn9 = initBtn("9", x[2], y[2], event -> {
            repaintFont();
if (addWrite) {
                if (Pattern.matches("[0]*", inText.getText())) {
                    inText.setText("9");
                } else {
                    inText.setText(inText.getText() + "9");
                }
            } else {
inText.setText("9");
                addWrite = true;
            }
            go = true;
        });
        btnMul = initBtn("*", x[3], y[2], event -> {
            repaintFont();
            if (Pattern.matches("([-]?\\d+[.]\\d*)|(\\d+)", inText.getText()))
if (go) {
                    val = calc(val, inText.getText(), opt);
                    if (Pattern.matches("[-]?[\\d]+[.][0]*", String.valueOf(val))) {
inText.setText(String.valueOf((int) val));
                    } else {
                        inText.setText(String.valueOf(val));
                    }
opt = '*';                  go =
false;                  addWrite
= false;
            } else {

        opt = '*';
            }
        });

        btn4 = initBtn("4", x[0], y[3], event -> {
            repaintFont();
if (addWrite) {
            if (Pattern.matches("[0]*", inText.getText())) {
                inText.setText("4");
            } else {
                inText.setText(inText.getText() + "4");
            }
        } else {
            inText.setText("4");
            addWrite = true;
        }
        go = true;
        });

        btn5 = initBtn("5", x[1], y[3], event -> {
            repaintFont();
if (addWrite) {
            if (Pattern.matches("[0]*", inText.getText())) {
                inText.setText("5");
            } else {
                inText.setText(inText.getText() + "5");
            }
        } else {
            inText.setText("5");
            addWrite = true;
        }
        go = true;
        });

        btn6 = initBtn("6", x[2], y[3], event -> {
            repaintFont();
if (addWrite) {
            if (Pattern.matches("[0]*", inText.getText())) {
                inText.setText("6");
            } else {
                inText.setText(inText.getText() + "6");
            }
} else {
            inText.setText("6");
            addWrite = true;
```

```
          }
          go = true;
        });
        btnSub = initBtn("-", x[3], y[3], event -> {
repaintFont();

if (Pattern.matches("([-]?\\d+[.]\\d*)|(\\d+)", inText.getText()))
if (go) {
                val = calc(val, inText.getText(), opt);
                if (Pattern.matches("[-]?[\\d]+[.][0]*", String.valueOf(val))) {
inText.setText(String.valueOf((int) val));
                } else {
                  inText.setText(String.valueOf(val));
                }

                opt = '-';
go = false;
addWrite = false;
              } else {
opt = '-';
              }
        });

        btn1 = initBtn("1", x[0], y[4], event -> {
          repaintFont();
if (addWrite) {
              if (Pattern.matches("[0]*", inText.getText())) {
                inText.setText("1");
              } else {
                inText.setText(inText.getText() + "1");
              }
          } else {
              inText.setText("1");
              addWrite = true;
          }
          go = true;
        });

        btn2 = initBtn("2", x[1], y[4], event -> {
          repaintFont();
if (addWrite) {
              if (Pattern.matches("[0]*", inText.getText())) {
                inText.setText("2");
              } else {
                inText.setText(inText.getText() + "2");
              }
          } else {
              inText.setText("2");
              addWrite = true;
```

```
            }
            go = true;
        });
        btn3 = initBtn("3", x[2], y[4], event -> {
            repaintFont();
if (addWrite) {
                if (Pattern.matches("[0]*", inText.getText())) {

                   inText.setText("3");
                } else {
                    inText.setText(inText.getText() + "3");
                }
            } else {
inText.setText("3");
                addWrite = true;
            }
            go = true;
        });

        btnAdd = initBtn("+", x[3], y[4], event -> {
            repaintFont();
            if (Pattern.matches("([-]?\\d+[.]\\d*)|(\\d+)", inText.getText()))
if (go) {
                    val = calc(val, inText.getText(), opt);
                    if (Pattern.matches("[-]?[\\d]+[.][0]*", String.valueOf(val))) {
inText.setText(String.valueOf((int) val));
                    } else {
                        inText.setText(String.valueOf(val));
                    }
opt = '+';                    go =
false;                  addWrite
= false;
                } else {
opt = '+';
                }
        });

        btnPoint = initBtn(".", x[0], y[5], event -> {
            repaintFont();            if (addWrite) {
if (!inText.getText().contains(".")) {
inText.setText(inText.getText() + ".");
                }
            } else {
                inText.setText("0.");
                addWrite = true;
            }
            go = true;
        });
        btn0 = initBtn("0", x[1], y[5], event -> {
            repaintFont();
if (addWrite) {
                if (Pattern.matches("[0]*", inText.getText())) {
                    inText.setText("0");
                } else {
                    inText.setText(inText.getText() + "0");
                }
            } else {
                inText.setText("0");
                addWrite = true;
            }
            go = true;
        });

        btnEqual = initBtn("=", x[2], y[5], event -> {
            if (Pattern.matches("([-]?\\d+[.]\\d*)|(\\d+)", inText.getText()))
if (go) {
                    val = calc(val, inText.getText(), opt);
                    if (Pattern.matches("[-]?[\\d]+[.][0]*", String.valueOf(val))) {
inText.setText(String.valueOf((int) val));
                    } else {
                        inText.setText(String.valueOf(val));
                    }
opt = '=';
                    addWrite = false;
                }
        });
        btnEqual.setSize(2 * BUTTON_WIDTH + 10,
BUTTON_HEIGHT);

        btnRoot = initBtn("√", x[4], y[1], event -> {
            if (Pattern.matches("([-]?\\d+[.]\\d*)|(\\d+)", inText.getText()))
if (go) {
                    val = Math.sqrt(Double.parseDouble(inText.getText()));
if (Pattern.matches("[-]?[\\d]+[.][0]*", String.valueOf(val))) {
inText.setText(String.valueOf((int) val));
                    } else {
                        inText.setText(String.valueOf(val));
                    }
opt = '√';
                    addWrite = false;
                }
        });
        btnRoot.setVisible(false);
        btnPower = initBtn("pow", x[4], y[2], event -> {
            repaintFont();
            if (Pattern.matches("([-]?\\d+[.]\\d*)|(\\d+)", inText.getText()))
if (go) {
                    val = calc(val, inText.getText(), opt);
                if (Pattern.matches("[-]?[\\d]+[.][0]*", String.valueOf(val))) {
inText.setText(String.valueOf((int) val));
                } else {
                    inText.setText(String.valueOf(val));
                }
opt = '^';                   go =
false;                 addWrite
= false;
            }   else   {
opt = '^';              }
        });
        btnPower.setFont(new Font("Comic Sans MS", Font.PLAIN, 24));
btnPower.setVisible(false);

        btnLog = initBtn("ln", x[4], y[3], event -> {
            if (Pattern.matches("([-]?\\d+[.]\\d*)|(\\d+)", inText.getText()))
if (go) {
                val = Math.log(Double.parseDouble(inText.getText()));
if (Pattern.matches("[-]?[\\d]+[.][0]*", String.valueOf(val))) {
inText.setText(String.valueOf((int) val));
                } else {
                    inText.setText(String.valueOf(val));
                }
opt = 'l';
                addWrite = false;
            }
        });
        btnLog.setVisible(false);

        window.setLayout(null);
window.setResizable(false);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Close button clicked? =
End The process
        window.setVisible(true);
    }

    private JComboBox<String> initCombo(String[] items, int x, int y, String toolTip, Consumer
consumerEvent) {
        JComboBox<String> combo = new JComboBox<>(items);
        combo.setBounds(x, y, 140, 25);
combo.setToolTipText(toolTip);        combo.setCursor(new
Cursor(Cursor.HAND_CURSOR));
combo.addItemListener(consumerEvent::accept);
window.add(combo);

        return combo;
    }
    private JButton initBtn(String label, int x, int y, ActionListener event) {
JButton btn = new JButton(label);
```

```
        btn.setBounds(x, y, BUTTON_WIDTH, BUTTON_HEIGHT);
btn.setFont(new Font("Comic Sans MS", Font.PLAIN, 28));
btn.setCursor(new Cursor(Cursor.HAND_CURSOR));
        btn.addActionListener(event);
btn.setFocusable(false);
        window.add(btn);

        return btn;
    }

    public double calc(double x, String input, char opt) {
inText.setFont(inText.getFont().deriveFont(Font.PLAIN));         double
y = Double.parseDouble(input);
        switch (opt) {
case '+':
                return x + y;
case '-':
                return x - y;
case '*':
                return x * y;
case '/':
                return x / y;
case '%':
                return x % y;
case '^':               return
Math.pow(x, y);             default:
                inText.setFont(inText.getFont().deriveFont(Font.PLAIN));
return y;
        }
    }

    private void repaintFont() {
        inText.setFont(inText.getFont().deriveFont(Font.PLAIN));
    }

    private Consumer<ItemEvent> calcTypeSwitchEventConsumer = event -> {
if (event.getStateChange() != ItemEvent.SELECTED) return;

        String selectedItem = (String) event.getItem();
switch (selectedItem) {             case "Standard":
                window.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
                btnRoot.setVisible(false);
btnPower.setVisible(false);                btnLog.setVisible(false);
                break;
case "Scientific":
                window.setSize(WINDOW_WIDTH + 80,
WINDOW_HEIGHT);                btnRoot.setVisible(true);
btnPower.setVisible(true);                btnLog.setVisible(true);
                break;
        }
    };
    private Consumer<ItemEvent> themeSwitchEventConsumer = event -> {
if (event.getStateChange() != ItemEvent.SELECTED) return;

        String selectedTheme = (String) event.getItem();
switch (selectedTheme) {


  case "Simple":
                window.getContentPane().setBackground(null);
                btnC.setBackground(null);
btnBack.setBackground(null);
btnMod.setBackground(null);
btnDiv.setBackground(null);
btnMul.setBackground(null);
btnSub.setBackground(null);
btnAdd.setBackground(null);
btnRoot.setBackground(null);
btnLog.setBackground(null);
btnPower.setBackground(null);
btnEqual.setBackground(null);
btn0.setBackground(null);
btn1.setBackground(null);
btn2.setBackground(null);
btn3.setBackground(null);
btn4.setBackground(null);
btn5.setBackground(null);
btn6.setBackground(null);
btn7.setBackground(null);
btn8.setBackground(null);
btn9.setBackground(null);
btnPoint.setBackground(null);
btnC.setForeground(Color.BLACK);
btnBack.setForeground(Color.BLACK);
btnMod.setForeground(Color.BLACK);
btnDiv.setForeground(Color.BLACK);
btnMul.setForeground(Color.BLACK);
btnSub.setForeground(Color.BLACK);
btnAdd.setForeground(Color.BLACK);
btnEqual.setForeground(Color.BLACK);
btnLog.setForeground(Color.BLACK);
btnPower.setForeground(Color.BLACK);
btnRoot.setForeground(Color.BLACK);
break;          case "Colored":
                window.getContentPane().setBackground(null);
btnC.setBackground(Color.RED);
btnBack.setBackground(Color.ORANGE);
btnMod.setBackground(Color.GREEN);
btnDiv.setBackground(Color.PINK);
btnMul.setBackground(Color.PINK);
btnSub.setBackground(Color.PINK);
```

```
btnAdd.setBackground(Color.PINK);
btnRoot.setBackground(Color.PINK);
btnLog.setBackground(Color.PINK);
btnPower.setBackground(Color.PINK);
btnEqual.setBackground(Color.BLUE);
btn0.setBackground(Color.WHITE);
btn1.setBackground(Color.WHITE);
btn2.setBackground(Color.WHITE);
btn3.setBackground(Color.WHITE);
btn4.setBackground(Color.WHITE);
btn5.setBackground(Color.WHITE);
btn6.setBackground(Color.WHITE);
btn7.setBackground(Color.WHITE);
btn8.setBackground(Color.WHITE);
btn9.setBackground(Color.WHITE);
            btnPoint.setBackground(Color.WHITE);

            btnC.setForeground(Color.WHITE);
btnBack.setForeground(Color.WHITE);
btnMod.setForeground(Color.WHITE);
btnDiv.setForeground(Color.WHITE);
btnMul.setForeground(Color.WHITE);
btnSub.setForeground(Color.WHITE);
btnAdd.setForeground(Color.WHITE);
btnEqual.setForeground(Color.WHITE);
btnLog.setForeground(Color.WHITE);
btnPower.setForeground(Color.WHITE);
            btnRoot.setForeground(Color.WHITE);               break;
case "DarkTheme":               final Color primaryDarkColor = new
Color(141, 38, 99);               final Color secondaryDarkColor = new
Color(171, 171, 171);
window.getContentPane().setBackground(new Color(68, 68, 68));
btn0.setBackground(secondaryDarkColor);
btn1.setBackground(secondaryDarkColor);
btn2.setBackground(secondaryDarkColor);
btn3.setBackground(secondaryDarkColor);
btn4.setBackground(secondaryDarkColor);
btn5.setBackground(secondaryDarkColor);
btn6.setBackground(secondaryDarkColor);
btn7.setBackground(secondaryDarkColor);
btn8.setBackground(secondaryDarkColor);
btn9.setBackground(secondaryDarkColor);
btnPoint.setBackground(secondaryDarkColor);

            btnC.setForeground(secondaryDarkColor);
btnBack.setForeground(secondaryDarkColor);
btnMod.setForeground(secondaryDarkColor);
btnDiv.setForeground(secondaryDarkColor);
btnMul.setForeground(secondaryDarkColor);
btnSub.setForeground(secondaryDarkColor);
```

```
btnAdd.setForeground(secondaryDarkColor);
btnEqual.setForeground(secondaryDarkColor);
btnLog.setForeground(secondaryDarkColor);
btnPower.setForeground(secondaryDarkColor);
btnRoot.setForeground(secondaryDarkColor);
btnC.setBackground(primaryDarkColor);
btnBack.setBackground(primaryDarkColor);
```

```
            btnMod.setBackground(primaryDarkColor);
btnDiv.setBackground(primaryDarkColor);
btnMul.setBackground(primaryDarkColor);
btnSub.setBackground(primaryDarkColor);
btnAdd.setBackground(primaryDarkColor);
btnRoot.setBackground(primaryDarkColor);
btnLog.setBackground(primaryDarkColor);
btnPower.setBackground(primaryDarkColor);
btnEqual.setBackground(primaryDarkColor);
        }
    };

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

- Adjust the GUI's appearance and functionality based on the selected theme and type.

5. Handle dynamic GUI appearance:
   - Based on the selected theme, change the background colors, button colors, and text colors to match the chosen theme, providing a visually appealing and customizable user experience.

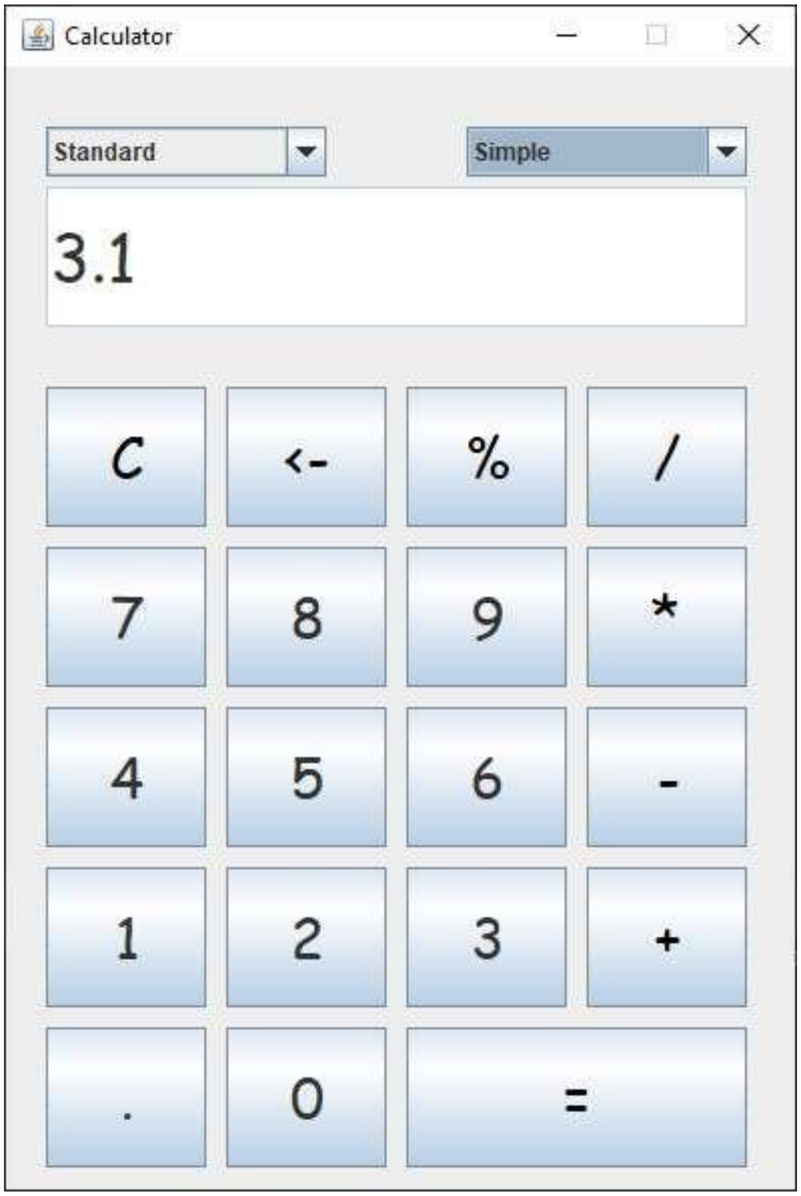
6. Perform input validation and error handling:
   - Ensure that the input is properly formatted and validate user actions to prevent unexpected behavior.

7. Initialize the main window, set its size and properties, and make it visible.

8. The main method creates an instance of the `Calculator` class to run the application.

This algorithm provides a general overview of the calculator's functionality, including user interaction, calculations, and theme customization. The specific implementation details and code structure may vary based on the programming language and libraries used.

RESULT


Calculator
Standard
Simple
3.1
C
<-
%
/
7
8
9
*
4
5
6
-
1
2
3
+
.
0
=

WORKING

The provided Java code is for a graphical calculator application that works as follows:

1. Initialization: When the program is executed, it creates an instance of the `Calculator` class in the `main` method. This instance initializes the graphical user interface (GUI) of the calculator.

2. GUI Setup: The code sets up the GUI with the following components:

   - Text Field (`inText`): This field serves as the input and display area for numeric values and calculation results.
   - Buttons: The calculator includes buttons for digits (0-9), arithmetic operators (+, -, *, /, %), special functions (square root, power, natural logarithm), and control buttons (clear, backspace, equals).
   - Combo Boxes: Combo boxes allow users to select the calculator type (Standard or Scientific) and the visual theme (Simple, Colored, or DarkTheme).

3. User Interaction:
   - Users can enter numbers by clicking on the digit buttons. The entered numbers are displayed in the text field.
   - Users can perform basic arithmetic calculations by clicking on the operator buttons (+, -, *, /, %).
   - For more advanced operations, users can switch to "Scientific" mode, where they can calculate square roots, powers, and natural logarithms.
   - Control buttons allow users to clear the input, delete the last character, and calculate percentages.
   - Users can switch between different calculator types and themes using the combo boxes.

4. Action Listeners:

- Each button has an associated action listener that responds to button clicks. For example, when a digit button is clicked, the corresponding digit is added to the input.

When an operator button is clicked, it triggers the calculation logic.

- The "C" button clears the input and resets the calculator, while the "<-" button removes the last character from the input.
- The equals button performs the calculation based on the operator selected and displays the result.

5. Calculation Logic:
   - The `calc` method is responsible for performing the actual arithmetic calculations. It takes the current value, the input, and the operator as arguments and returns the result of the calculation. The result is then displayed in the text field.

6. Theme and Calculator Type Switching:
   - Users can switch between different themes and calculator types using the combo boxes. The GUI's appearance and functionality adapt to the selected theme and type. For example, in "Scientific" mode, additional mathematical functions become available.

7. Dynamic GUI Appearance:
   - The code dynamically changes the GUI's appearance (colors and fonts) based on the selected theme. This allows users to personalize the calculator's visual style.

8. Error Handling:
   - The code includes basic input validation and error handling to ensure that the calculator functions correctly and to prevent unexpected behavior.

Overall, this calculator application provides a user-friendly interface for performing a wide range of mathematical calculations, from simple arithmetic to more advanced functions, with the flexibility to switch between different calculator types and themes to suit the user's preferences. Users interact with the calculator by clicking on buttons, and the code ensures that the appropriate calculations are performed and results are displayed in real-time on the GUI.