Calculator Project Report

Project Name:

Calculator (JAVA)

Project By:

Aditya Kumawat

Table of Contents

Abstract:

This project introduces a Java-based calculator application, providing users with a comprehensive tool for arithmetic computation. Key features of the calculator include:

Responsive User Interface:

The calculator's graphical user interface (GUI) is designed to be highly responsive, ensuring optimal usability across a wide range of devices and screen sizes. Leveraging Swing components and flexible layout managers, the UI dynamically adjusts to accommodate varying display configurations, offering a seamless user experience.

Modular Architecture:

The project adopts a modular architecture, dividing the application into distinct components for UI management, mathematical logic, and event handling. This modular design enhances maintainability and scalability, facilitating future updates and modifications.

Advanced Mathematical Operations:

The calculator supports a wide range of arithmetic operations, including addition, subtraction, multiplication, and division. Moreover, it can handle complex expressions with multiple operators and operands, utilizing advanced parsing techniques and efficient computation algorithms.

Flexible Input Handling:

Users can input mathematical expressions using both keyboard and mouse inputs, with support for common functionalities such as clear, delete, and decimal point insertion. The calculator's input handling mechanism ensures intuitive interaction and seamless expression evaluation.

Error Handling and Validation:

The application incorporates robust error handling and input validation mechanisms to ensure accurate computation and prevent runtime errors. Invalid expressions are detected and appropriately handled, providing users with informative feedback to aid in error correction.

User-Friendly Design:

Emphasizing usability and accessibility, the calculator features a visually appealing design with intuitive controls and informative feedback. Users can easily navigate the interface, perform calculations, and interpret results, enhancing overall user satisfaction.

Extensibility and Customization:

The modular architecture of the calculator facilitates extensibility and customization, allowing developers to integrate additional features or modify existing functionality as per specific requirements. This flexibility enables the adaptation of the calculator to diverse use cases and user preferences.

By intertwining the principles of responsive UI design, modular architecture, and advanced computation capabilities, this project culminates in the development of a multifaceted and intuitive calculator application in Java. This application caters to a diverse user base, encompassing both casual users seeking simplicity and professional users demanding accuracy and efficiency in their calculations. Its responsive UI ensures seamless interaction across various devices and screen sizes, while its modular architecture facilitates future scalability and customization. Moreover, the incorporation of advanced computation algorithms enhances the calculator's capability to handle complex mathematical expressions with precision. Thus, this Java-based calculator emerges as a versatile tool that not only meets the needs of everyday users but also serves the requirements of professionals in diverse fields.

Objective:

The primary aim of this endeavor is to develop a Java-based calculator application that embodies a blend of innovative design concepts, modular structural elements, and sophisticated computational functionalities. The overarching objectives guiding this project are outlined below:

Responsive Interface Enhancement:

The primary focus lies in refining the user interface to ensure optimal responsiveness across a spectrum of devices and screen sizes. Leveraging dynamic layout management and adaptable component design, the UI will seamlessly adjust to varying display configurations, thereby enhancing user accessibility and engagement.

Modular Architecture Refinement:

The project seeks to further refine the modular architecture underpinning the calculator application. This entails the segregation of components responsible for UI management, mathematical computation, and event handling into discrete modules. By enhancing the modularity and encapsulation of these components, the architecture will be primed for enhanced scalability, maintainability, and extensibility.

Complex Computational Capability Augmentation:

A key focus area involves augmenting the calculator's computational capabilities to support complex mathematical operations with enhanced precision and efficiency. This involves refining parsing algorithms and computation methodologies to handle intricate mathematical expressions encompassing multiple operators and operands seamlessly.

Adaptive Input Handling Optimization:

The project endeavors to optimize input handling mechanisms to ensure flexibility and adaptability to diverse user inputs. This includes refining input validation routines to detect and handle erroneous expressions effectively while providing informative feedback to users to aid in error correction and enhance overall usability.

Error Resilience Strengthening:

Robust error handling mechanisms will be implemented to fortify the application against runtime errors and inaccuracies. By incorporating comprehensive error detection and mitigation strategies, the calculator will provide users with a seamless and error-resilient computing experience, thereby instilling confidence in its reliability and accuracy.

Intuitive Design and Navigation Enhancement:

Emphasis will be placed on enhancing the intuitiveness of the calculator's design and navigation. This involves refining visual elements, streamlining user interactions, and providing informative cues and prompts to guide users through the calculation process, thereby fostering a user-friendly computing environment.

Customization and Extension Facilitation:

The project aims to facilitate customization and extension of the calculator application to cater to diverse user requirements and preferences. By refining the underlying architecture and design principles, developers will be empowered to integrate additional features, modify existing functionality, and tailor the application to specific use cases and user preferences seamlessly.

By aligning with these multifaceted objectives, the project endeavors to deliver a comprehensive and adaptable Java-based calculator application that not only meets the diverse needs of users but also serves as a benchmark for excellence in software design and development.

Introduction:

Embarking on a journey of software innovation and excellence, this project endeavors to craft a Java-based calculator application that transcends conventional boundaries through a convergence of cutting-edge design paradigms, modular architectural prowess, and computational sophistication. Guided by a relentless pursuit of perfection, the introduction of this calculator seeks to redefine user experience and computational efficiency, setting new benchmarks in the realm of software engineering. The foundational pillars driving this endeavor encompass a spectrum of transformative objectives, each meticulously curated to elevate the calculator's capabilities to unprecedented heights:

Innovative Design Ideation:

Central to the project's ethos is the cultivation of an innovative design ethos that transcends traditional boundaries, fostering an environment where user interaction converges seamlessly with aesthetic appeal and ergonomic efficiency. Through the integration of intuitive graphical interfaces and responsive design elements, the calculator aims to revolutionize the user computing experience, empowering users with unparalleled flexibility and usability.

Modular Architectural Brilliance:

At the core of the calculator's development lies a commitment to modular architectural brilliance, characterized by the meticulous segmentation of application components into cohesive modules. By embracing modularity, the project seeks to enhance scalability, maintainability, and extensibility, laying the foundation for seamless integration of future enhancements and feature expansions.

Computational Eminence:

The calculator aspires to transcend the boundaries of conventional computation, ushering in an era of computational eminence marked by unparalleled precision, efficiency, and versatility. Through the integration of advanced parsing algorithms and computational methodologies, the calculator endeavors to unravel the complexities of mathematical expressions with unrivaled prowess, catering to the diverse computational needs of users across domains.

Usability Optimization:

A cornerstone of the project's vision is the relentless pursuit of usability optimization, aimed at streamlining user interactions and enhancing overall accessibility. By refining input handling mechanisms and error resilience strategies, the calculator seeks to instill confidence in users, empowering them to engage with mathematical computations with unparalleled ease and efficiency.

Error Resilience Reinforcement:

Robust error handling mechanisms are poised to serve as the bulwark against computational inaccuracies and runtime anomalies, ensuring the calculator's resilience in the face of adverse scenarios. Through comprehensive error detection and mitigation strategies, the project endeavors to foster a computing environment characterized by reliability, accuracy, and trustworthiness.

Visionary Design Philosophy:

The project embraces a visionary design philosophy that transcends conventional boundaries, fostering an environment where design innovation converges seamlessly with user-centric principles. By prioritizing intuitive navigation, informative feedback, and aesthetic appeal, the calculator aims to set new standards in design excellence, captivating users with its visual allure and functional sophistication.

Driven by an unwavering commitment to excellence and innovation, this project aspires to chart new frontiers in the realm of software engineering, delivering a Java-based calculator application that transcends the boundaries of conventional computation, redefining user experience and computational efficiency in the process.

Methodology Report for Calculator Project:

Implementation:

The calculator application consists of three main components:

Calculator Logic:

Responsible for evaluating mathematical expressions entered by the user and performing calculations.

Calculator UI:

Provides the graphical user interface for the calculator, including buttons for input, display area for showing the expression and result, and event handling for user interactions.

Main Class (Calculator):

Acts as the entry point of the application and integrates the calculator logic with the UI.

Introduction to Methodology:

The methodology employed in the development of the calculator project encompasses a systematic approach aimed at achieving the desired functionality, usability, and reliability of the application. This report outlines the key phases, techniques, and tools utilized throughout the project lifecycle.

Requirement Analysis:

The methodology begins with a comprehensive analysis of user requirements and expectations for the calculator application. Stakeholder consultations, user surveys, and market research were conducted to identify essential features, user interface preferences, and performance benchmarks.

Design Phase:

Following requirement analysis, the design phase focuses on translating user requirements into a coherent architectural design and user interface layout. Object-oriented design principles were employed to create modular, scalable, and maintainable code structures. The Model-View-Controller (MVC) architectural pattern was adopted to separate the application's logic, presentation, and user interaction components, promoting code reusability and flexibility.

Implementation Strategy:

The implementation strategy involved the development of the calculator application using Java programming language and Swing GUI toolkit. Object-oriented programming (OOP) concepts such as encapsulation, inheritance, and polymorphism were leveraged to encapsulate behavior within objects and promote code modularity. Version control systems such as Git were used to manage code revisions and facilitate collaborative development among team members.

Testing and Quality Assurance:

Rigorous testing procedures were integrated into the development process to ensure the correctness, robustness, and reliability of the calculator application. Unit testing, integration testing, and system testing were conducted to validate individual components, interactions between modules, and overall system behavior. Test-driven development (TDD) principles were applied to write test cases before implementing corresponding functionality, facilitating early bug detection and code refactoring.

User Feedback and Iterative Development:

User feedback played a crucial role in refining the calculator application's features, usability, and performance. Alpha and beta testing phases were conducted to gather feedback from a diverse user base and incorporate necessary improvements iteratively. Agile development methodologies, such as Scrum or

Kanban, were adopted to adapt quickly to changing requirements and prioritize feature implementation based on user needs.

Performance Optimization:

Performance optimization techniques were employed to enhance the calculator application's responsiveness and efficiency. Algorithmic optimizations, data structure selection, and resource management strategies were implemented to minimize computational overhead and memory footprint. Profiling tools and performance monitoring metrics were used to identify bottlenecks and fine-tune critical code paths for optimal execution speed.

Documentation and Knowledge Sharing:

Comprehensive documentation was created to facilitate knowledge sharing, onboarding of new team members, and future maintenance of the calculator application. Documentation artifacts, including design documents, API references, user manuals, and troubleshooting guides, were prepared to provide comprehensive guidance on the application's architecture, functionality, and usage.

Deployment and Release Management:

The deployment phase involved packaging the calculator application for distribution across different platforms and environments. Cross-platform compatibility considerations were addressed to ensure seamless deployment on Windows, macOS, and Linux operating systems. Continuous integration (CI) and continuous delivery (CD) pipelines were established to automate the build, testing, and deployment processes, enabling frequent releases and rapid iteration cycles.

Feedback Evaluation and Continuous Improvement:

Post-deployment feedback mechanisms were implemented to gather user feedback, monitor application performance, and identify areas for further enhancement. Feedback evaluation sessions were conducted periodically to analyze user satisfaction metrics, prioritize feature requests, and plan future iterations of the calculator application. Continuous improvement initiatives were initiated to address user feedback iteratively and maintain the application's relevance in a dynamic market landscape.

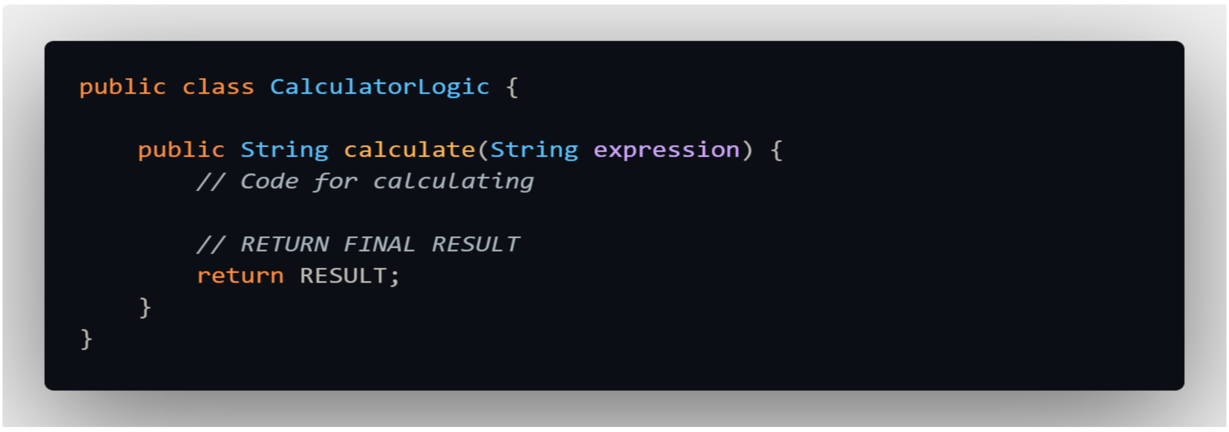
Conclusion:

The methodology employed in the development of the calculator project reflects a systematic and iterative approach aimed at delivering a high-quality, user-centric application. By combining rigorous requirement analysis, robust design principles, agile development practices, and continuous feedback loops, the project achieved its objectives of creating a versatile, efficient, and user-friendly calculator application tailored to diverse user needs. Ongoing support and maintenance efforts ensure the longevity and sustainability of the application in meeting evolving user requirements and technological advancements.

Workflow:

1. **Calculator Logic**:

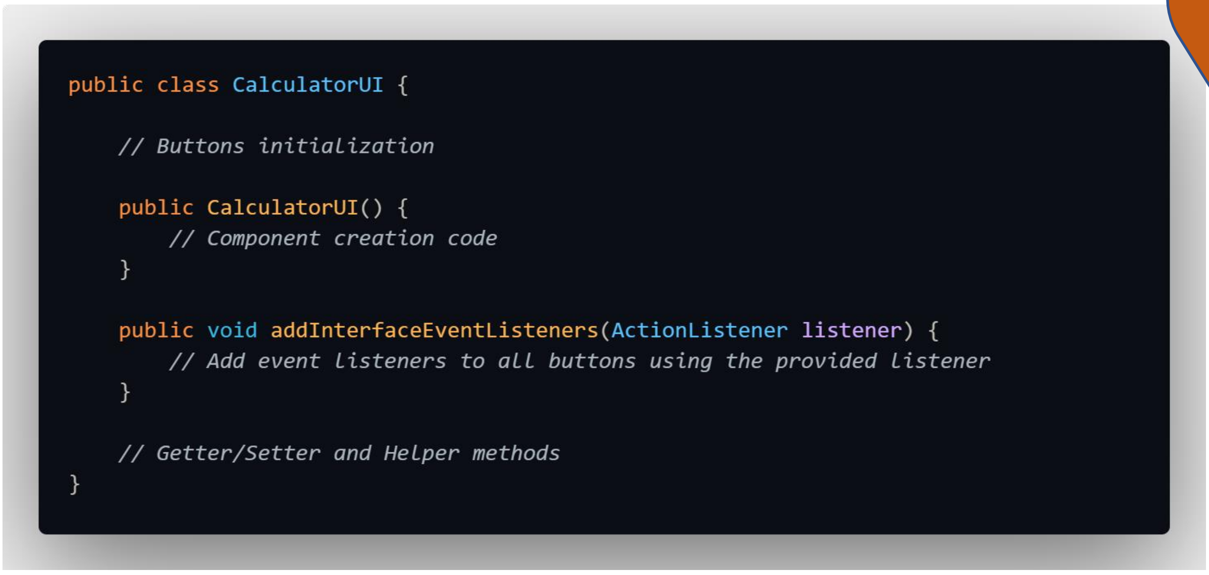
   - Parses the input expression to extract operands and operators.
   - Performs arithmetic calculations based on the operator precedence (multiplication/division before addition/subtraction).
   - Handles negative numbers and updates operands' signs accordingly.
   - Returns the final result of the expression.



```java
public class CalculatorLogic {

    public String calculate(String expression) {
        // Code for calculating

        // RETURN FINAL RESULT
        return RESULT;
    }
}
```

2. **Calculator UI**:

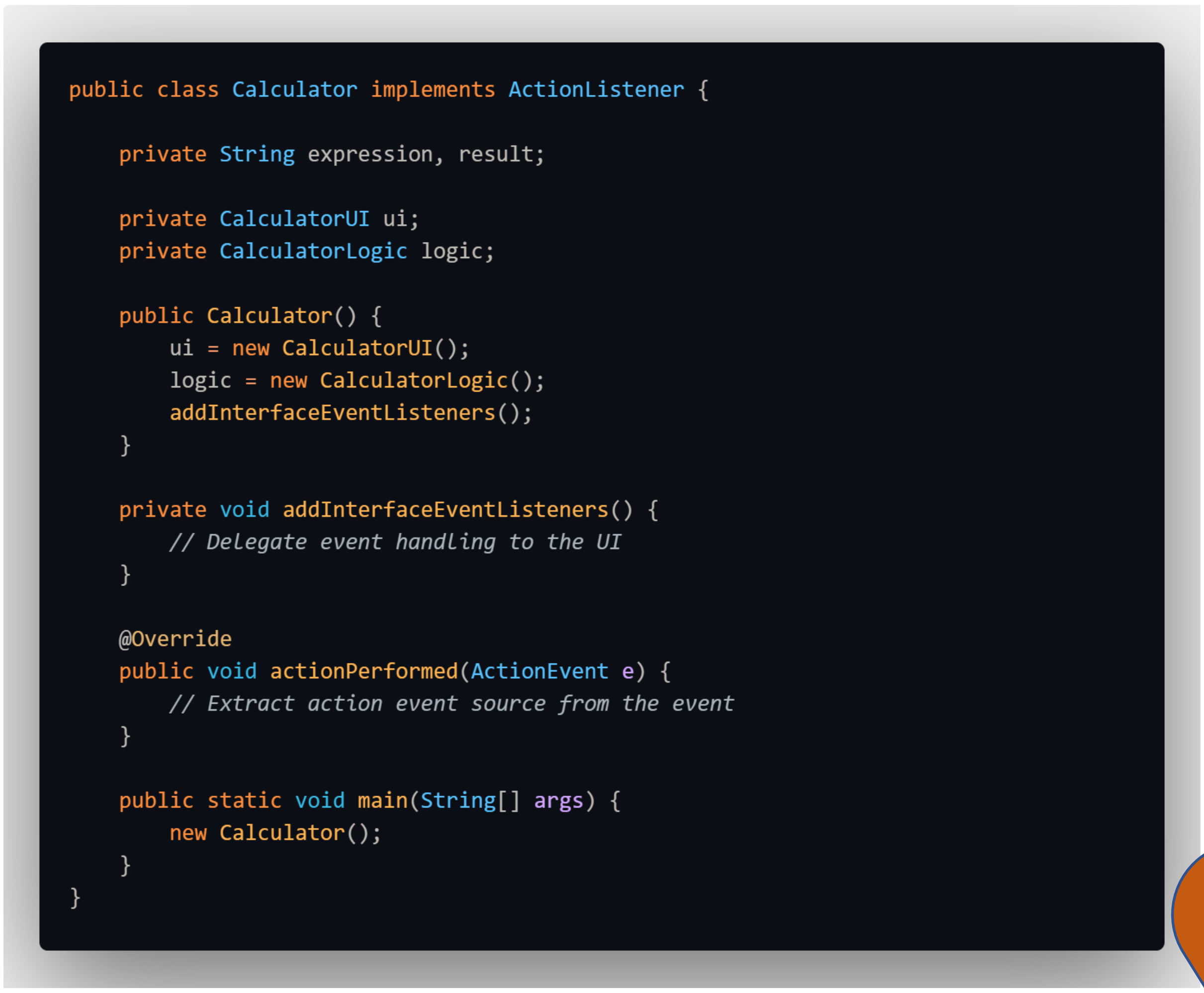
   - Constructs the graphical interface using Java SWING components such as JFrame, JLabel, JTextField, and JButtons.
   - Defines event listeners to handle user interactions with the calculator buttons.
   - Displays the input expression and result in the designated areas.



```java
public class CalculatorUI {

    // Buttons initialization

    public CalculatorUI() {
        // Component creation code
    }

    public void addInterfaceEventListeners(ActionListener listener) {
        // Add event listeners to all buttons using the provided listener
    }

    // Getter/Setter and Helper methods
}
```

3. **Main Class (Calculator)**:

- Initializes the Calculator UI and Calculator Logic components.
- Sets up event listeners to delegate user actions to the appropriate methods.
- Manages the interaction between the UI and logic components to perform calculations and update the display.



```java
public class Calculator implements ActionListener {

    private String expression, result;

    private CalculatorUI ui;
    private CalculatorLogic logic;

    public Calculator() {
        ui = new CalculatorUI();
        logic = new CalculatorLogic();
        addInterfaceEventListeners();
    }

    private void addInterfaceEventListeners() {
        // Delegate event handling to the UI
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        // Extract action event source from the event
    }

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

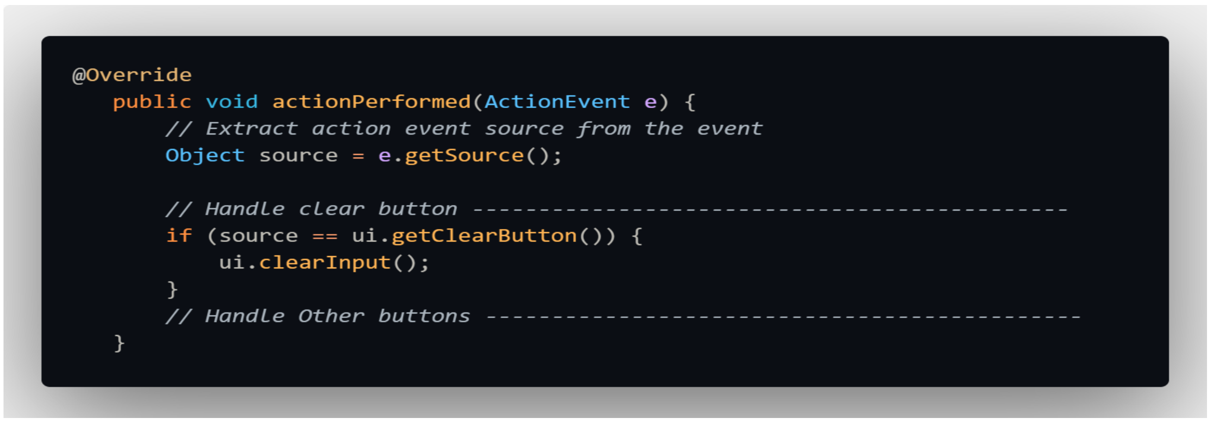
Code:

So, as you have already seen the workflow of the Calculator program Here in code section we will discuss the minor workings of the program and see which code does what.

Lets discuss the main Calculator class first:

1. Here first we Initializes the Calculator UI and Calculator Logic components.
2. Sets up event listeners to delegate user actions to the appropriate methods.
3. Manages the interaction between the UI and logic components to perform calculations and update the display

   In third part the decision making was done, what change should be done in UI is calculated and Transfer control to Logic is to be done. Shown in the below code.



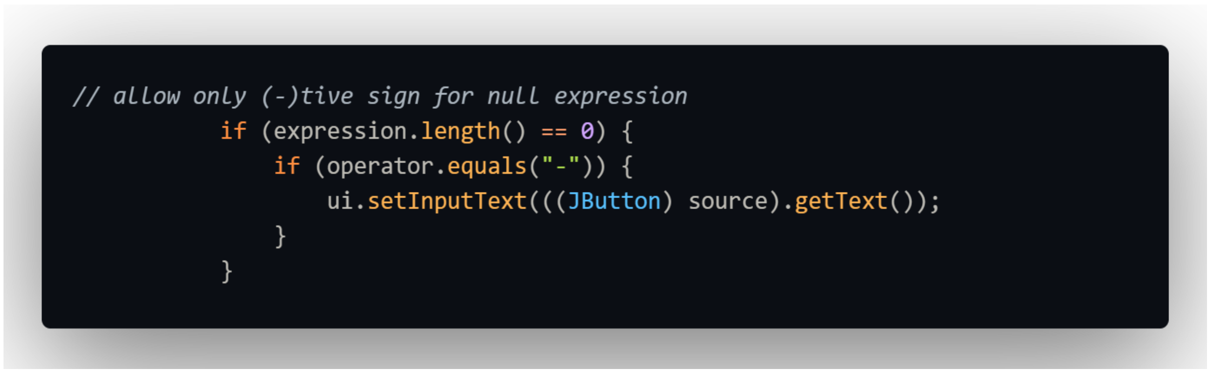
```java
@Override
    public void actionPerformed(ActionEvent e) {
        // Extract action event source from the event
        Object source = e.getSource();

        // Handle clear button ---------------------------------------------
        if (source == ui.getClearButton()) {
            ui.clearInput();
        }
        // Handle Other buttons ---------------------------------------------
    }
```

So, as we saw setting actions on user interraction buttons are easy except for one category of buttons and they are operator buttons

e.g. Addition, Substraction, Multiplication, Division.

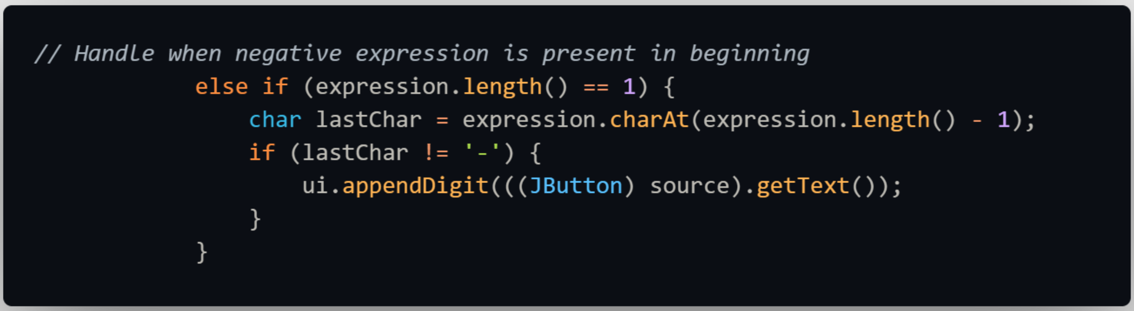
These buttons need to handle some edge cases, Lets discuss them below:

1. **Allow only negative operator if it is in the beginning of expression.**



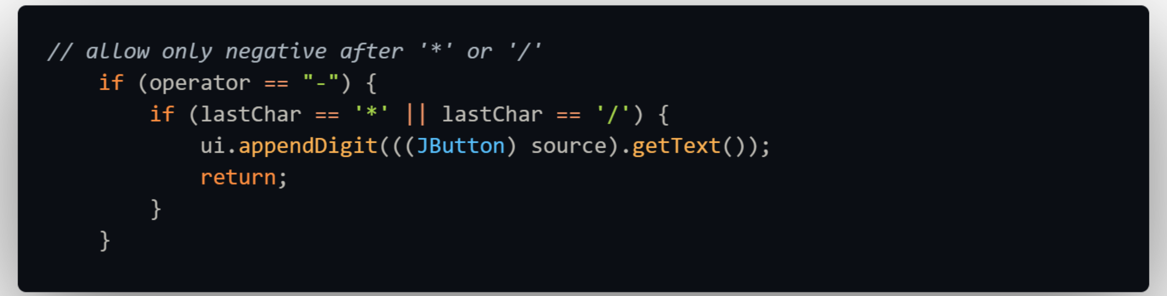
```java
// allow only (-)tive sign for null expression
            if (expression.length() == 0) {
                if (operator.equals("-")) {
                    ui.setInputText(((JButton) source).getText());
                }
            }
```

2. **Handle the first character negative operator in the expression.**



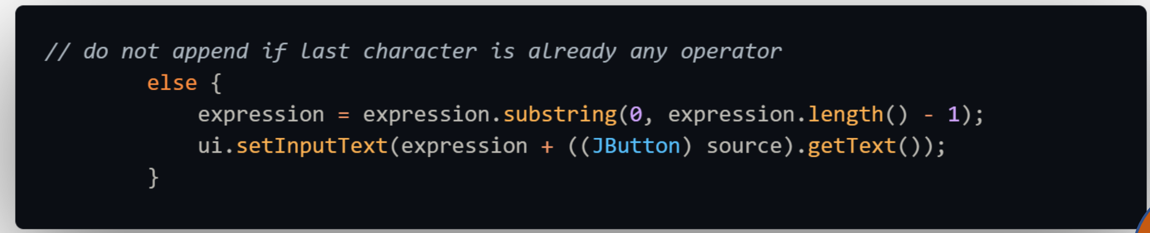
```java
// Handle when negative expression is present in beginning
            else if (expression.length() == 1) {
                char lastChar = expression.charAt(expression.length() - 1);
                if (lastChar != '-') {
                    ui.appendDigit(((JButton) source).getText());
                }
            }
```

3. **Allow only negative operator after '*' or '/' case.**



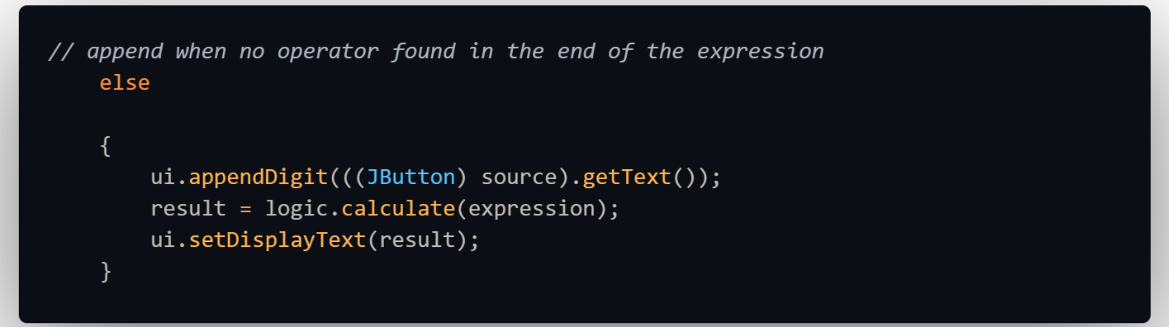
```java
// allow only negative after '*' or '/'
    if (operator == "-") {
        if (lastChar == '*' || lastChar == '/') {
            ui.appendDigit(((JButton) source).getText());
            return;
        }
    }
```

4. **Do not apend if last character is already operator case.**



```java
// do not append if last character is already any operator
        else {
            expression = expression.substring(0, expression.length() - 1);
            ui.setInputText(expression + ((JButton) source).getText());
        }
```

5. **At last when ther is no operator at the end of expression case.**

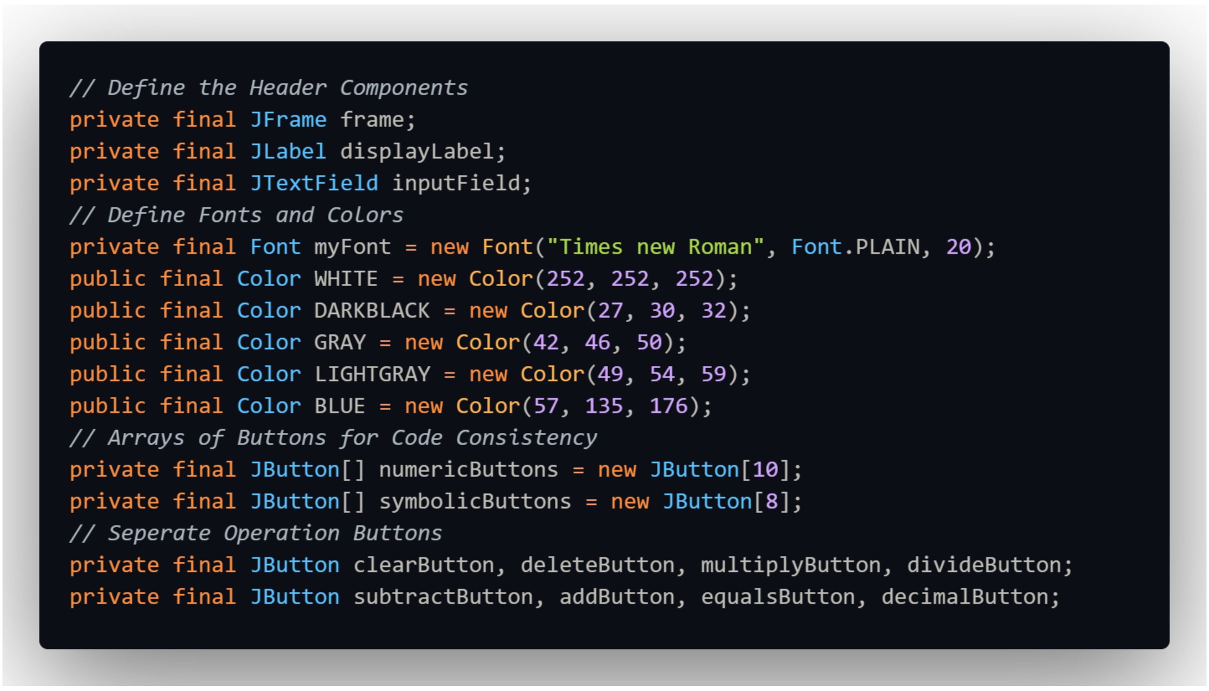


```java
// append when no operator found in the end of the expression
    else

    {
        ui.appendDigit(((JButton) source).getText());
        result = logic.calculate(expression);
        ui.setDisplayText(result);
    }
```

With the above cases handled there is nothing important left in the main Calculator class.

So now lets dive into CalculatorUI class:

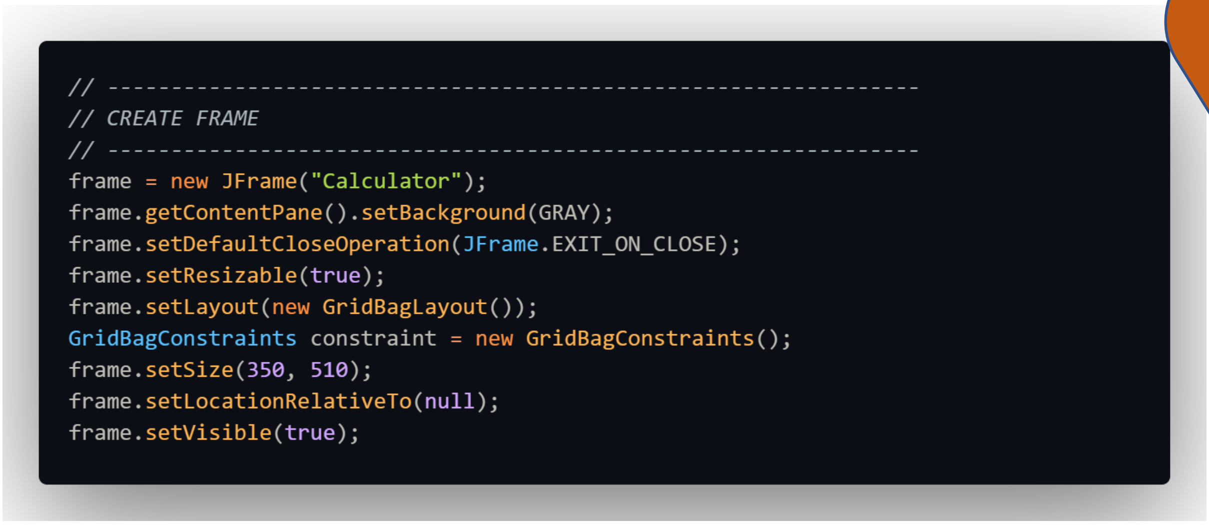
1. Defining all the component variables and constraints for building UI.



```java
// Define the Header Components
private final JFrame frame;
private final JLabel displayLabel;
private final JTextField inputField;
// Define Fonts and Colors
private final Font myFont = new Font("Times new Roman", Font.PLAIN, 20);
public final Color WHITE = new Color(252, 252, 252);
public final Color DARKBLACK = new Color(27, 30, 32);
public final Color GRAY = new Color(42, 46, 50);
public final Color LIGHTGRAY = new Color(49, 54, 59);
public final Color BLUE = new Color(57, 135, 176);
// Arrays of Buttons for Code Consistency
private final JButton[] numericButtons = new JButton[10];
private final JButton[] symbolicButtons = new JButton[8];
// Seperate Operation Buttons
private final JButton clearButton, deleteButton, multiplyButton, divideButton;
private final JButton subtractButton, addButton, equalsButton, decimalButton;
```

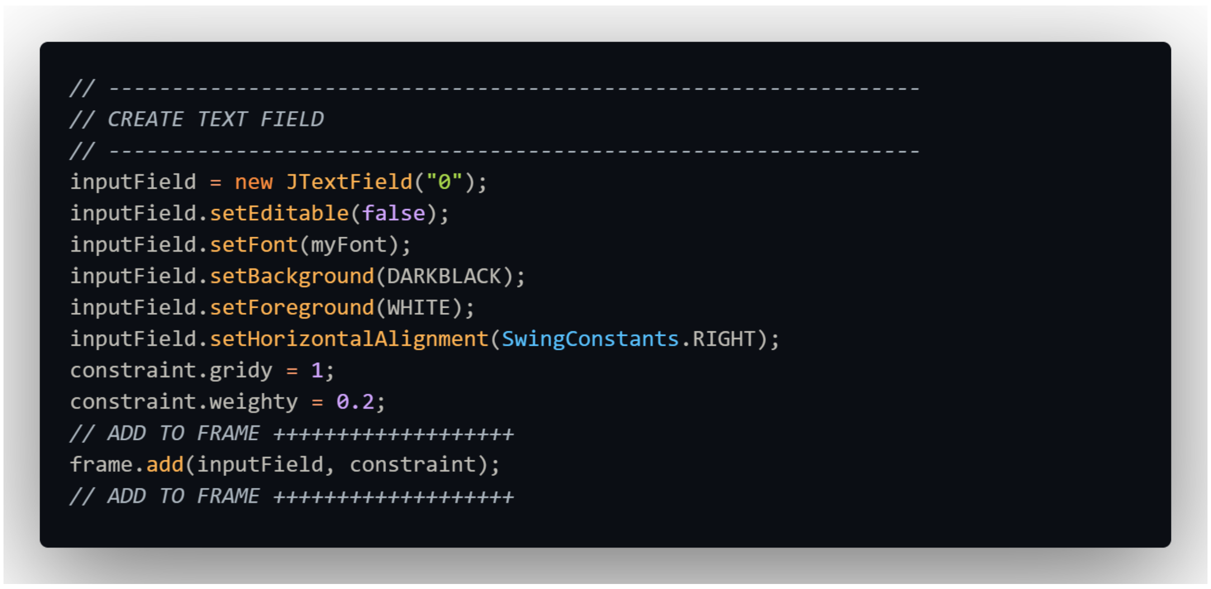
2. Constructs the graphical interface using Java SWING components such as JFrame, JLabel, JTextField, and JButtons.

JFrame:



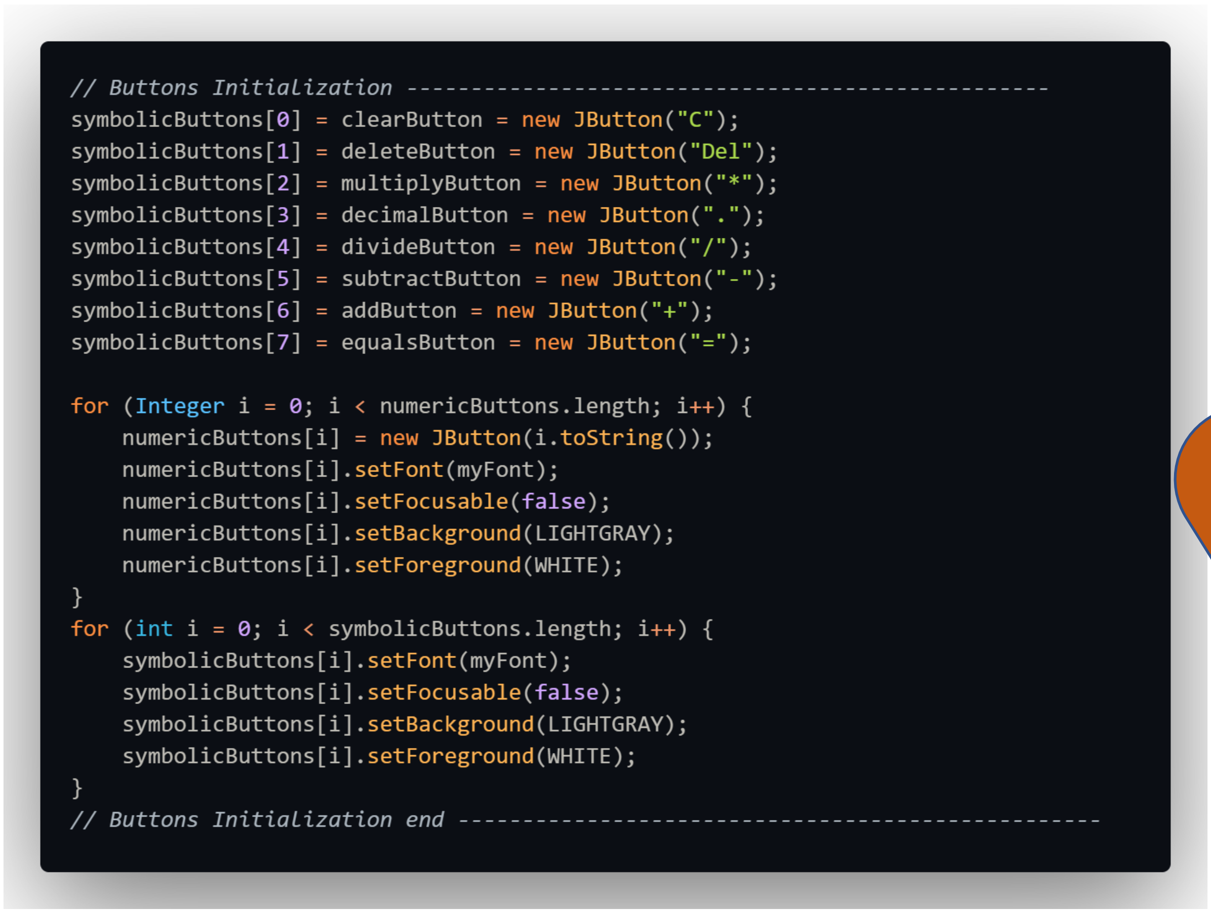
```java
// ---------------------------------------------------------------
// CREATE FRAME
// ---------------------------------------------------------------
frame = new JFrame("Calculator");
frame.getContentPane().setBackground(GRAY);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setResizable(true);
frame.setLayout(new GridBagLayout());
GridBagConstraints constraint = new GridBagConstraints();
frame.setSize(350, 510);
frame.setLocationRelativeTo(null);
frame.setVisible(true);
```

JTextField:



```java
// ----------------------------------------------------------------
// CREATE TEXT FIELD
// ----------------------------------------------------------------
inputField = new JTextField("0");
inputField.setEditable(false);
inputField.setFont(myFont);
inputField.setBackground(DARKBLACK);
inputField.setForeground(WHITE);
inputField.setHorizontalAlignment(SwingConstants.RIGHT);
constraint.gridy = 1;
constraint.weighty = 0.2;
// ADD TO FRAME +++++++++++++++++++
frame.add(inputField, constraint);
// ADD TO FRAME +++++++++++++++++++
```

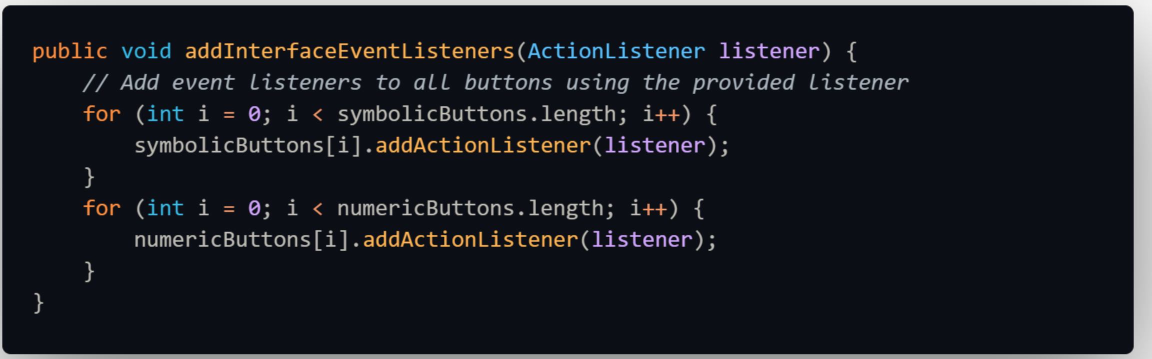
JButtons:



```java
// Buttons Initialization --------------------------------------------------
symbolicButtons[0] = clearButton = new JButton("C");
symbolicButtons[1] = deleteButton = new JButton("Del");
symbolicButtons[2] = multiplyButton = new JButton("*");
symbolicButtons[3] = decimalButton = new JButton(".");
symbolicButtons[4] = divideButton = new JButton("/");
symbolicButtons[5] = subtractButton = new JButton("-");
symbolicButtons[6] = addButton = new JButton("+");
symbolicButtons[7] = equalsButton = new JButton("=");

for (Integer i = 0; i < numericButtons.length; i++) {
    numericButtons[i] = new JButton(i.toString());
    numericButtons[i].setFont(myFont);
    numericButtons[i].setFocusable(false);
    numericButtons[i].setBackground(LIGHTGRAY);
    numericButtons[i].setForeground(WHITE);
}
for (int i = 0; i < symbolicButtons.length; i++) {
    symbolicButtons[i].setFont(myFont);
    symbolicButtons[i].setFocusable(false);
    symbolicButtons[i].setBackground(LIGHTGRAY);
    symbolicButtons[i].setForeground(WHITE);
}
// Buttons Initialization end --------------------------------------------------
```

After buttons Initialization we need to define their respective location in UI….

3. Defines event listeners to handle user interactions with the calculator buttons.



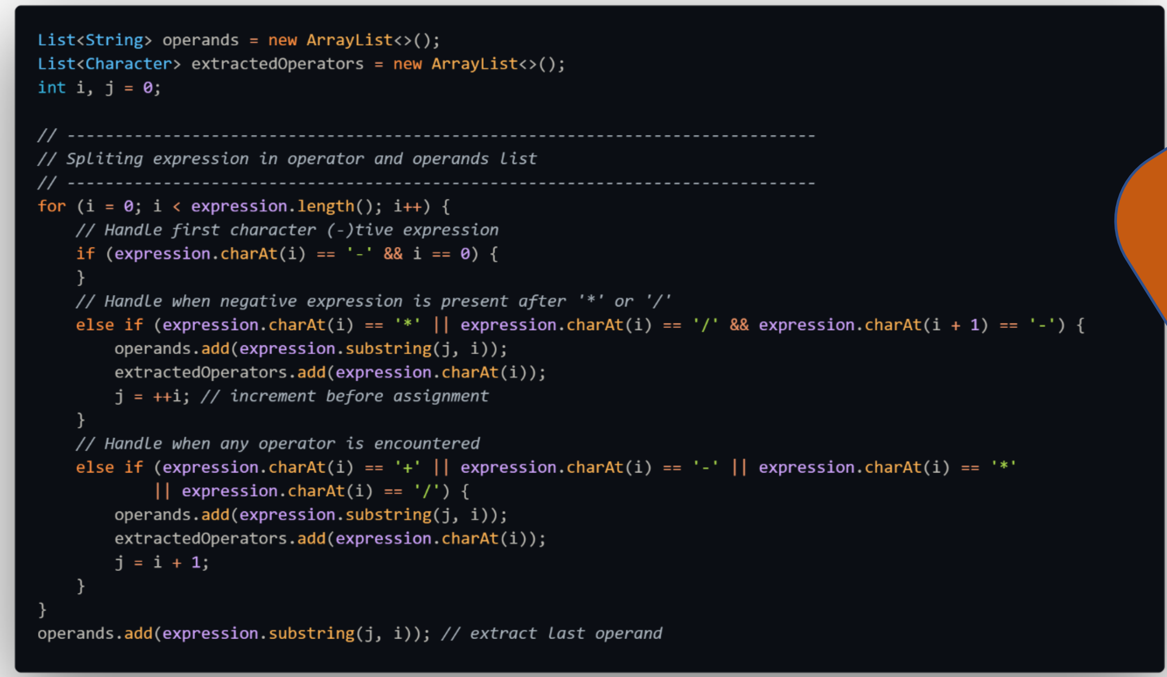
```java
public void addInterfaceEventListeners(ActionListener listener) {
    // Add event listeners to all buttons using the provided listener
    for (int i = 0; i < symbolicButtons.length; i++) {
        symbolicButtons[i].addActionListener(listener);
    }
    for (int i = 0; i < numericButtons.length; i++) {
        numericButtons[i].addActionListener(listener);
    }
}
```

That's all about the UI now the only thing left to handle is definitely the most Important thing which is Logic handling.

Let See the working of CalculatorLogic class:

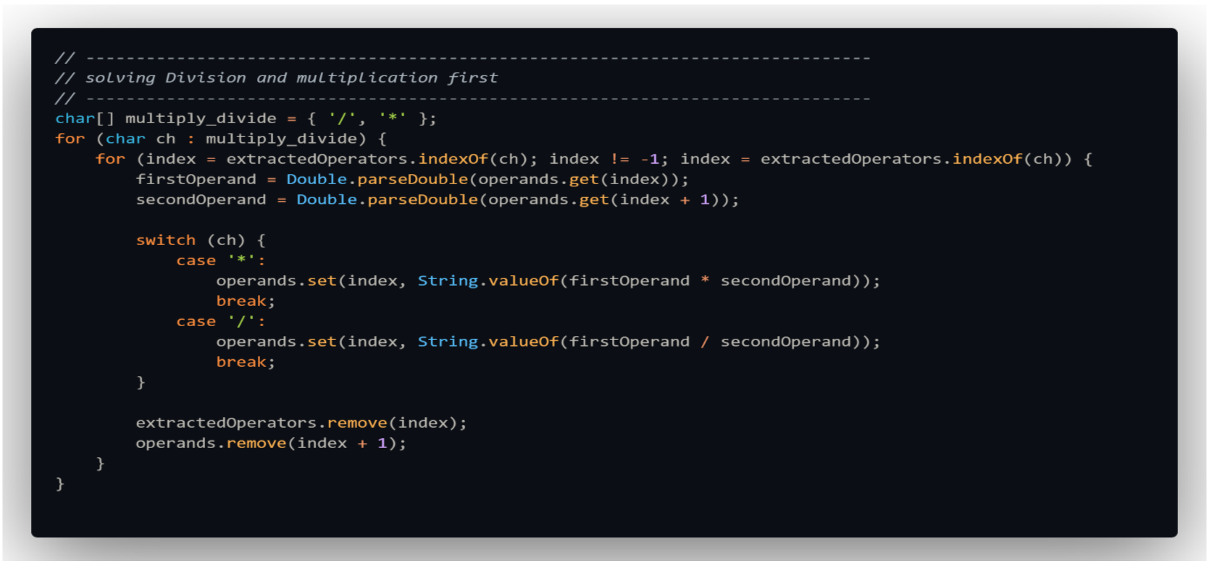
Here the idea is to take the Arithmetic expression as string and divide it into **Operand list** and **Operator list** and than perform calculations and return the result.

1. **Divide the arithmetic expression in list of operands and operator.**



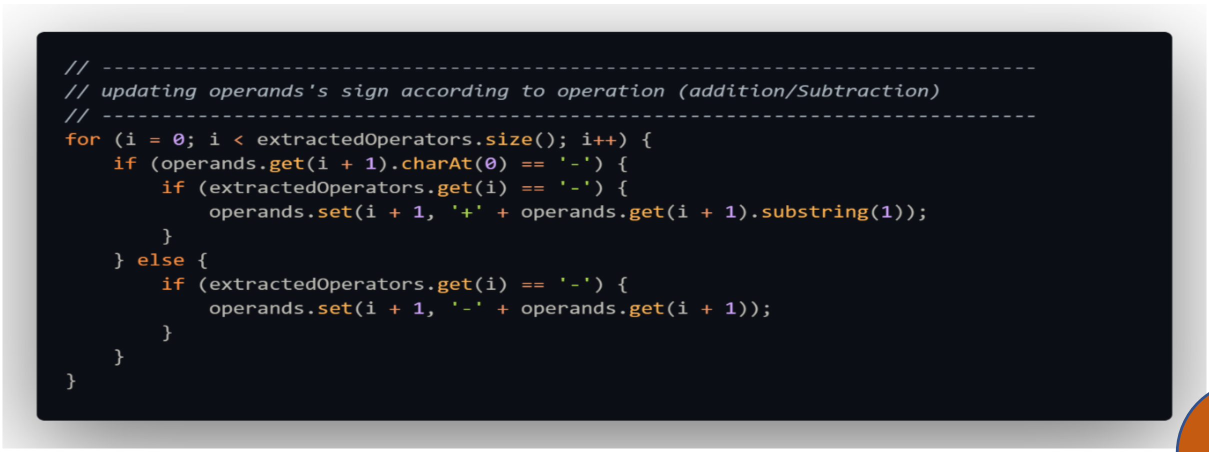
```java
List<String> operands = new ArrayList<>();
List<Character> extractedOperators = new ArrayList<>();
int i, j = 0;

// ----------------------------------------------------------------------------
// Spliting expression in operator and operands list
// ----------------------------------------------------------------------------
for (i = 0; i < expression.length(); i++) {
    // Handle first character (-)tive expression
    if (expression.charAt(i) == '-' && i == 0) {
    }
    // Handle when negative expression is present after '*' or '/'
    else if (expression.charAt(i) == '*' || expression.charAt(i) == '/' && expression.charAt(i + 1) == '-') {
        operands.add(expression.substring(j, i));
        extractedOperators.add(expression.charAt(i));
        j = ++i; // increment before assignment
    }
    // Handle when any operator is encountered
    else if (expression.charAt(i) == '+' || expression.charAt(i) == '-' || expression.charAt(i) == '*'
            || expression.charAt(i) == '/') {
        operands.add(expression.substring(j, i));
        extractedOperators.add(expression.charAt(i));
        j = i + 1;
    }
}
operands.add(expression.substring(j, i)); // extract last operand
```

2. **Solve the Division and Multiplication operations first.**



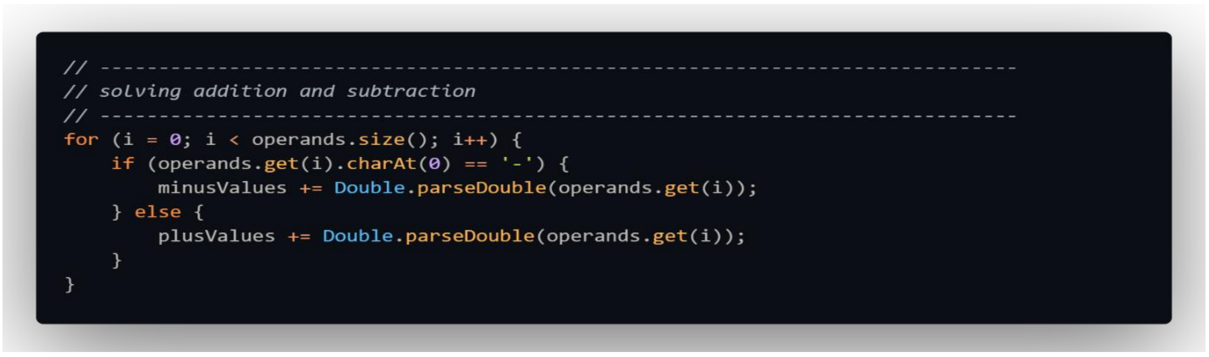
```java
// ---------------------------------------------------------------------------------
// solving Division and multiplication first
// ---------------------------------------------------------------------------------
char[] multiply_divide = { '/', '*' };
for (char ch : multiply_divide) {
    for (index = extractedOperators.indexOf(ch); index != -1; index = extractedOperators.indexOf(ch)) {
        firstOperand = Double.parseDouble(operands.get(index));
        secondOperand = Double.parseDouble(operands.get(index + 1));

        switch (ch) {
            case '*':
                operands.set(index, String.valueOf(firstOperand * secondOperand));
                break;
            case '/':
                operands.set(index, String.valueOf(firstOperand / secondOperand));
                break;
        }

        extractedOperators.remove(index);
        operands.remove(index + 1);
    }
}
```

3. **For solving the addition and subtraction we need to correct the sign of operands first. Below is the code to do that.**



```java
// ---------------------------------------------------------------------------------
// updating operands's sign according to operation (addition/Subtraction)
// ---------------------------------------------------------------------------------
for (i = 0; i < extractedOperators.size(); i++) {
    if (operands.get(i + 1).charAt(0) == '-') {
        if (extractedOperators.get(i) == '-') {
            operands.set(i + 1, '+' + operands.get(i + 1).substring(1));
        }
    } else {
        if (extractedOperators.get(i) == '-') {
            operands.set(i + 1, '-' + operands.get(i + 1));
        }
    }
}
```

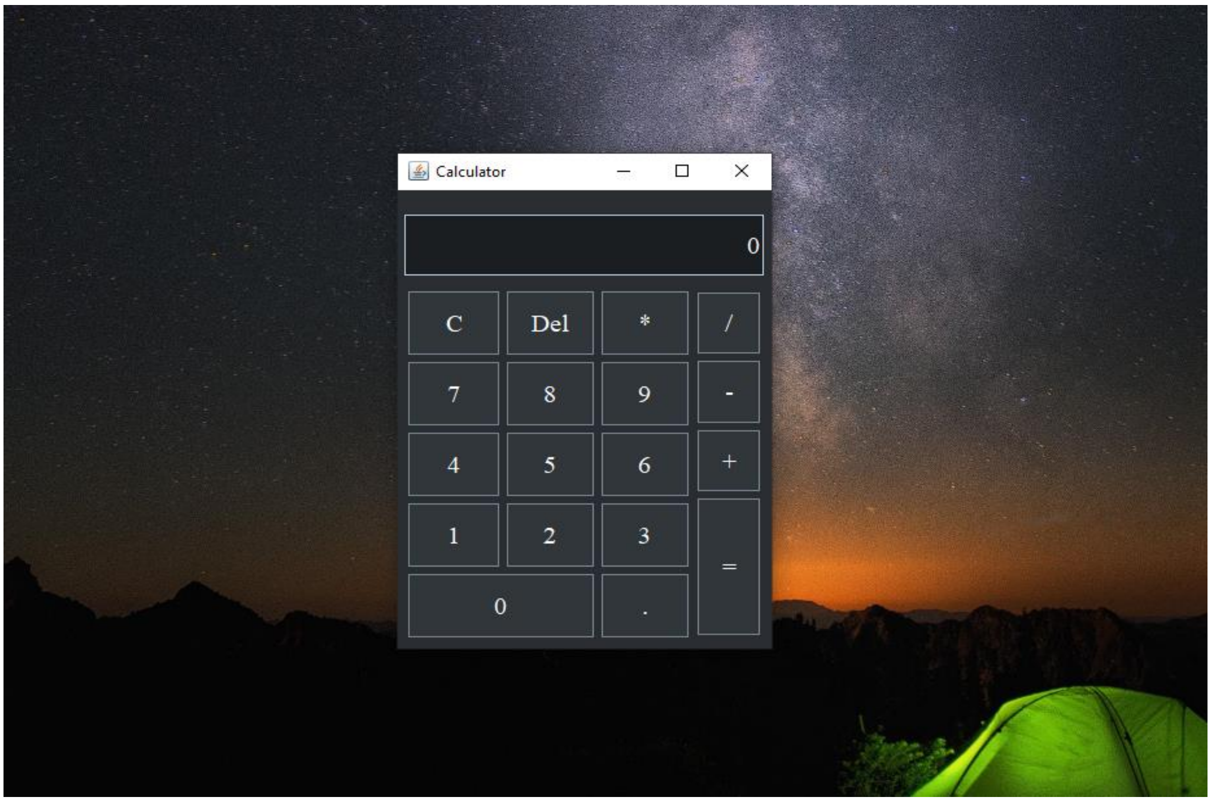
4. **At last we need to gather the positive operands at one side and negative operands at other side, calculate the output and than return the RESULT.**

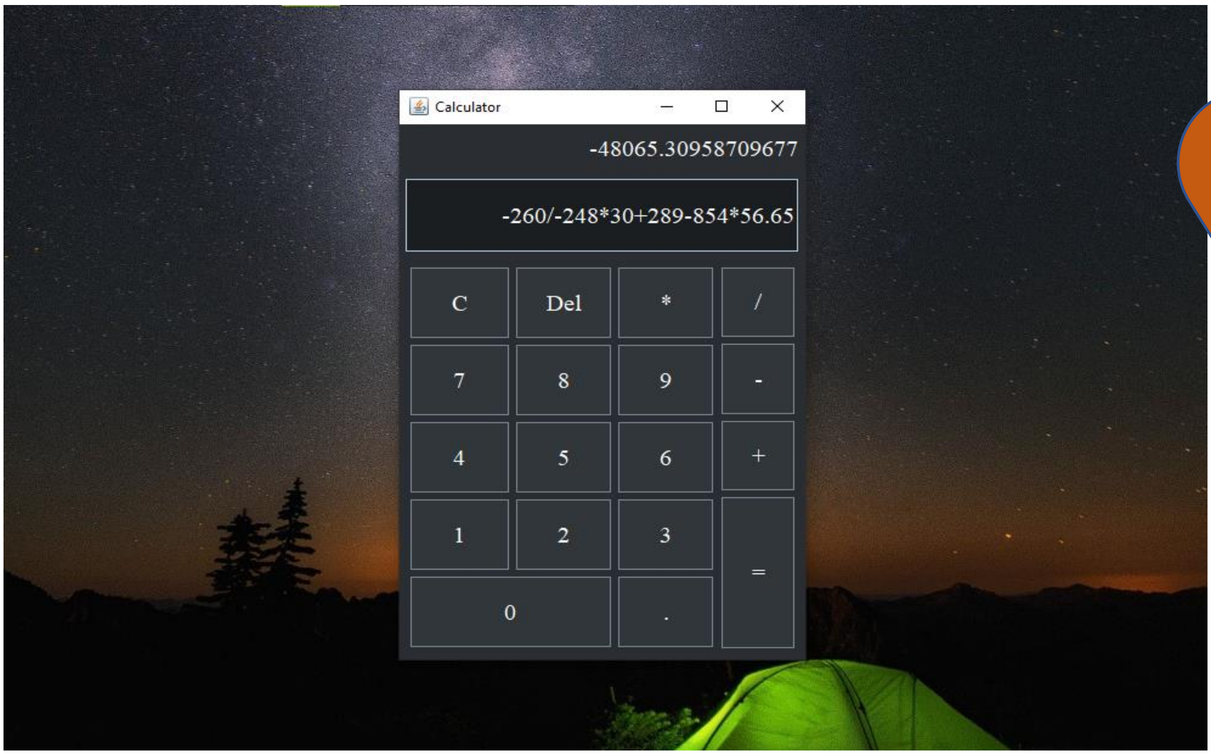


```java
// ---------------------------------------------------------------------------------
// solving addition and subtraction
// ---------------------------------------------------------------------------------
for (i = 0; i < operands.size(); i++) {
    if (operands.get(i).charAt(0) == '-') {
        minusValues += Double.parseDouble(operands.get(i));
    } else {
        plusValues += Double.parseDouble(operands.get(i));
    }
}
```

UI Look:

After doing all the research, learning, experimenting, and hours of work lets look at the final ui of the calculator.





-48065.30958709677

-260/-248*30+289-854*56.65

Conclusion:

In conclusion, the development of the calculator project has been both challenging and rewarding. Through the process of designing and implementing the calculator application, several key insights were gained into the fields of user interface design, event handling, and mathematical logic implementation.

The project began with the identification of objectives and requirements, leading to the creation of a clear plan and methodology for development. The user interface was carefully crafted to provide a seamless and intuitive experience for users, with attention to detail given to layout, color scheme, and font selection.

The implementation of the calculator's functionality involved the creation of logical operations to handle arithmetic calculations, including addition, subtraction, multiplication, and division. Error handling and edge cases were considered to ensure the robustness and reliability of the application.

Throughout the development process, teamwork and collaboration played a crucial role in overcoming challenges and achieving project goals. Communication between team members facilitated problem-solving and decision-making, leading to a successful outcome.

In summary, the calculator project demonstrates the importance of effective planning, design, and implementation in software development. By applying fundamental principles of programming and design, we were able to create a functional and user-friendly application that meets the needs of its intended audience.

Moving forward, there is potential for further refinement and enhancement of the calculator application, including the addition of advanced features and optimizations. Overall, the project serves as a valuable learning experience and lays the foundation for future endeavors in software development.