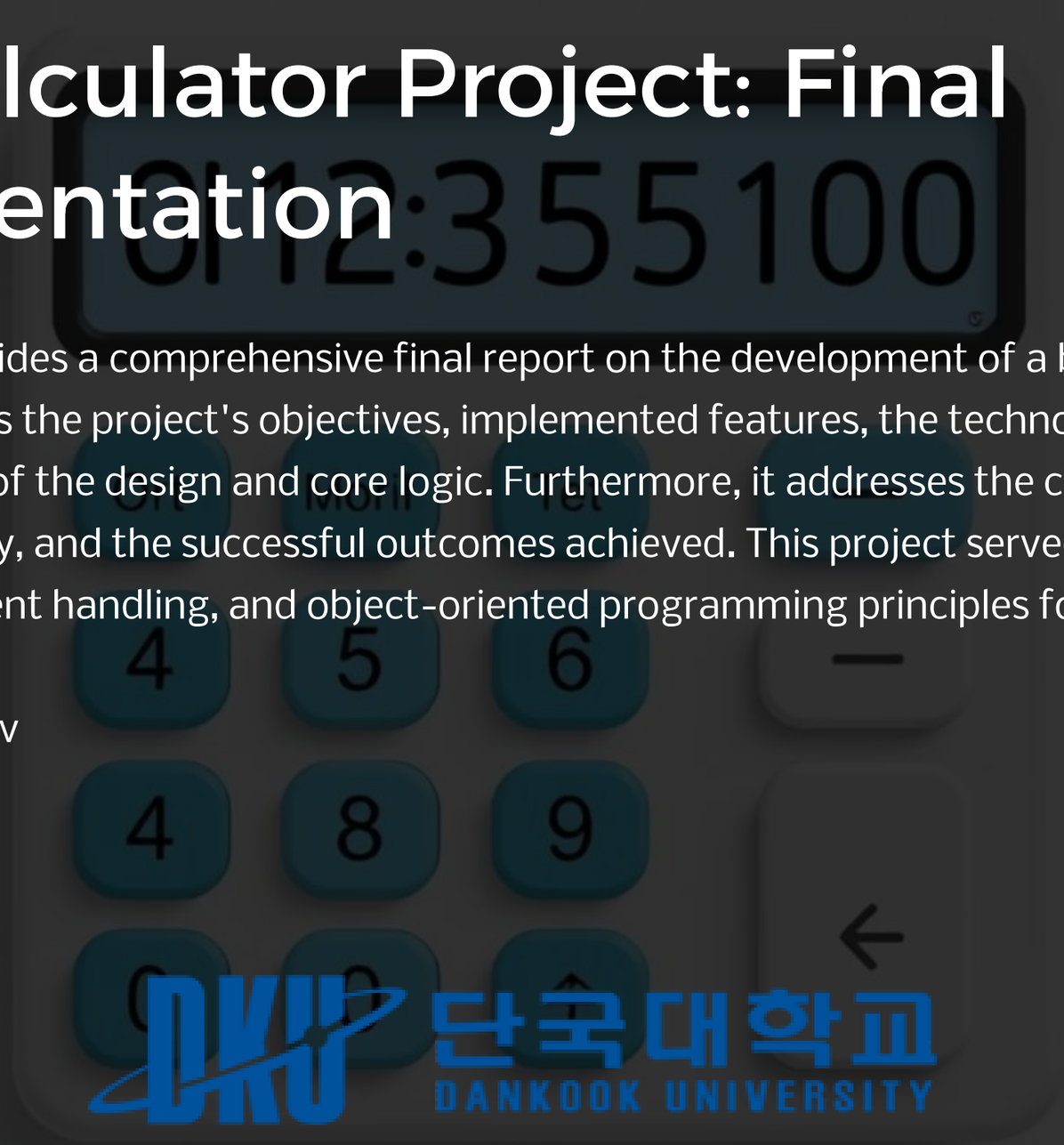
Java Calculator Project: Final Documentation

This document provides a comprehensive final report on the development of a basic calculator application using Java Swing. It details the project's objectives, implemented features, the technologies leveraged, and a thorough overview of the design and core logic. Furthermore, it addresses the challenges encountered, the testing methodology, and the successful outcomes achieved. This project served as a foundational exercise in Java GUI design, event handling, and object-oriented programming principles for beginners.

Javokhirbek Khalikov
32223879
Seoul, Korea
June 18, 2025



DKU 단국대학교 DANKOOK UNIVERSITY

Project Objectives and Scope

The primary objective of this project was to construct a functional and user-friendly calculator application leveraging the Java Swing toolkit. This endeavor aimed to provide practical experience in several key areas of Java development, extending beyond mere arithmetic computation.

1

Core Arithmetic Functionality

Implement a calculator capable of performing the four fundamental arithmetic operations: addition, subtraction, multiplication, and division. This included ensuring accuracy and robustness in calculations.

2

Intuitive Graphical User Interface (GUI)

Design and implement a clean, intuitive, and responsive user interface using standard Java Swing components. The focus was on ease of use and visual clarity for the end-user.

3

Real-time Input and Computation

Develop mechanisms for real-time input handling, allowing users to see their entered expressions dynamically. The application also needed to provide immediate results upon calculation requests.

4

Robust Error Handling

Incorporate comprehensive error handling mechanisms to manage exceptional cases such as division by zero, invalid input sequences, and other potential computational errors, ensuring the application's stability.

5

Practical Experience in OOP & GUI Design

Gain hands-on experience with object-oriented programming principles and best practices in the context of GUI application development, fostering a deeper understanding of Java's capabilities in desktop application creation.

Features and Functionalities Implemented

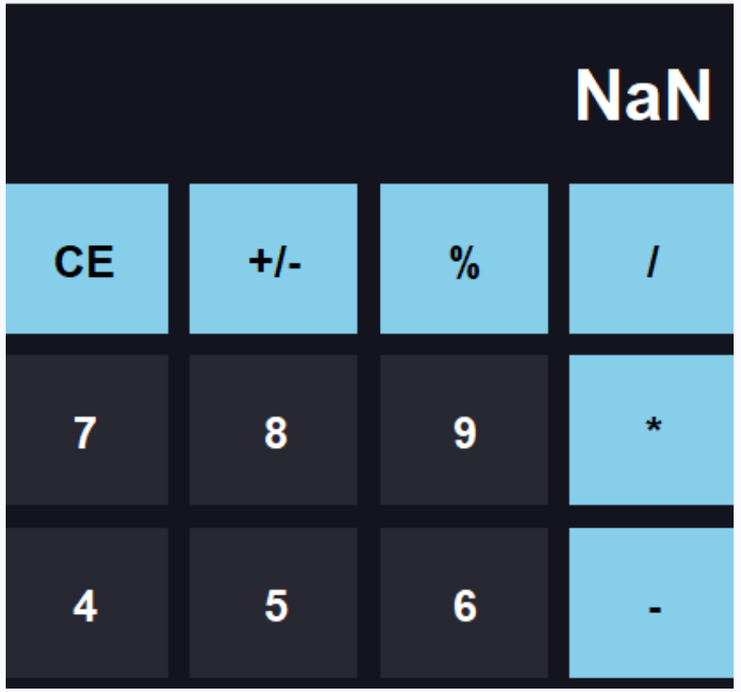
The developed calculator application boasts a range of functional and non-functional features designed to provide a comprehensive and satisfying user experience. These features were meticulously implemented to ensure both computational accuracy and user interface responsiveness.

Functional Features:

1. **Basic Arithmetic Operations:** Full support for addition (+), subtraction (–), multiplication (×), and division (÷).
2. **Decimal Input:** Capability to handle and process decimal numbers for more precise calculations.
3. **Clear (CE) Button:** A dedicated button to clear the current input or reset the entire calculation.
4. **Delete (←) Feature:** Allows users to remove the last entered digit or operator, mimicking a backspace functionality.
5. **Toggle Sign (+/–) & Percentage (%):** Advanced operations to change the sign of a number or calculate percentages, enhancing the calculator's utility.
6. **Equals (=) Button:** Triggers the computation and displays the final result of the entered expression.

Non-Functional Features:

1. **Responsive and Clean GUI:** The interface is designed to be visually appealing and adapts well to different display environments, providing a smooth user experience.
2. **Real-time Visual Feedback:** Buttons provide visual cues upon interaction, and the display updates instantly with input, enhancing user engagement.
3. **Simple Error Messages:** Clear and concise error messages are displayed for invalid operations, such as “NaN" for division by zero, guiding the user without overwhelming them.



Tools and Technologies Utilized

The development of the Java Calculator Project relied exclusively on robust, standardized tools and technologies within the Java ecosystem, ensuring stability, compatibility, and adherence to industry best practices.

Programming Language: Java

Java served as the foundational programming language for its object-oriented capabilities, platform independence, and extensive standard libraries. Its inherent robustness and widespread adoption made it an ideal choice for this academic project.

GUI Toolkit: Swing

Java Swing, a part of the Java Foundation Classes (JFC), was employed for constructing the graphical user interface. Swing's rich set of components and flexible layout managers provided the necessary tools for creating an interactive and visually appealing calculator.

Development Environment: Command Prompt & Notepad

For a fundamental understanding of the compilation and execution process, the project was developed using a minimalist setup: manual code editing in Notepad and compilation/execution via the Windows Command Prompt. This approach minimized abstraction layers, forcing a direct engagement with the Java toolchain.

Java Version: OpenJDK 17 (Temurin build)

OpenJDK 17, specifically the Temurin build, was chosen for its Long-Term Support (LTS) status, offering stability and access to modern Java features while ensuring compatibility and performance for the application.

Design Overview: UI and Event Handling

The architectural design of the calculator application focused on a clear separation of concerns, with distinct components for the user interface and the underlying event handling mechanism. This modular approach contributed to maintainable and scalable code.

User Interface Architecture:

The calculator's graphical interface is meticulously structured to ensure an intuitive user experience. At its core, the UI is composed of two primary areas:

1. **Display Text Field:** Positioned at the top of the calculator window, this non-editable **JTextField** component serves as the primary output area. It dynamically updates to show both the user's input expression and the computed results. Its non-editable nature prevents accidental input, maintaining the integrity of the calculation.
2. **Button Grid:** The main interactive area comprises a grid of buttons. These buttons represent numerical digits (0-9), arithmetic operators (+, −, ×, ÷), and specialized functions (CE, ←, +/−, %). The arrangement utilizes a **GridLayout** manager, ensuring uniform sizing and spacing of buttons, which contributes to a clean and organized aesthetic. Custom color schemes and subtle spacing enhancements were applied to differentiate button types (e.g., numbers vs. operators) and improve overall visual feedback.

This layered design ensures that the user can easily distinguish between input and output areas, and that the interactive elements are logically grouped for efficient operation.

Event Handling Mechanism:

Event handling is central to the calculator's interactivity. Each button on the interface is meticulously configured to respond to user interactions.

- **ActionListener Implementation:** Every button instance is registered with an **ActionListener** . This interface's **actionPerformed()** method is the entry point for handling button clicks.
- **Event Source Identification:** Within the **actionPerformed()** method, the source of the event (i.e., which button was clicked) is identified. This allows for conditional logic based on the button's label or a pre-defined command.

Core Logic and Implementation Details

The computational backbone of the calculator is designed for robust parsing and evaluation of arithmetic expressions, ensuring accuracy and handling various input scenarios.

Input Storage and Dynamic Expression Building

User input, whether numeric digits or operators, is dynamically appended to an internal **StringBuilder** object. This mutable sequence of characters represents the current mathematical expression visible on the calculator's display. This approach allows for efficient string manipulation as users type and backspace, avoiding the overhead of immutable String concatenations.

Specialized Button Logic: CE, +/-, %

Beyond basic arithmetic, several utility buttons are implemented with direct manipulation logic:

- **Clear Entry (CE):** Resets the current input expression to an empty state, effectively clearing the display and internal expression buffer.
- **Toggle Sign (+/–):** Modifies the last entered number's sign (positive to negative, or vice versa) directly within the **StringBuilder** expression, reflecting the change instantly.
- **Percentage (%):** Converts the last number in the expression to its percentage equivalent (dividing by 100), enabling quick percentage calculations.

The **evaluate()** Function: Stack-Based Parsing

The central computation engine resides within the **evaluate()** function, which employs a sophisticated stack-based algorithm to parse and compute the arithmetic expression. This method typically involves two stacks: one for numbers (operands) and another for operators. The algorithm processes the expression character by character, adhering to the standard order of operations (PEMDAS/BODMAS) to correctly handle precedence.

Robust Exception Handling

Critical to any robust application, the calculator incorporates comprehensive exception handling. Specifically, it meticulously catches and gracefully manages a **DivisionByZeroException** or similar arithmetic errors that might arise during the evaluation process. When such an exception is detected, a user-friendly "Error" message is displayed, preventing the application from crashing and guiding the user to correct the input. Other potential parsing errors, such as malformed expressions, are also managed to maintain application stability.

Challenges, Testing, and Conclusion

During the development cycle, several challenges were encountered, each providing valuable learning experiences. Rigorous testing ensured the application's reliability, leading to a successful project conclusion.

Challenges Faced:

1. **Expression Parsing Complexity:** The initial attempt to leverage JavaScript's **ScriptEngine** for expression evaluation proved problematic due to deprecation issues and unreliable behavior. This necessitated a shift to a custom-built, stack-based evaluator, which, while more challenging, offered greater control and reliability in parsing complex arithmetic expressions.

2. **Immediate Evaluation After Result (Chaining Operations):** A significant challenge involved managing the calculator's state after the equals (=) button was pressed. Ensuring that subsequent operations could seamlessly chain off the previous result without requiring a manual clear was critical for user flow. This required careful state management logic to correctly reset the expression or append new operations.

3. **Decimal Formatting Consistency:** Achieving consistent and clean decimal output was unexpectedly complex. This involved preventing unnecessary trailing zeros (e.g., "5.0" instead of "5") while ensuring precision for actual decimal results. Custom formatting logic was implemented to refine the display of floating-point numbers.

Testing and Results:

Comprehensive testing was conducted to validate the application's functionality and robustness across various scenarios.

- **Arithmetic Operations Validation:** Each of the four basic operations (+, −, ×, ÷) was rigorously tested with integer and decimal inputs, including edge cases.
- **Input Chaining Scenarios:** Tests specifically focused on sequences like "4 × 4 = → ÷ 4 =," ensuring the calculator correctly carried over the result for subsequent operations, displaying "4" in this example.
- **Invalid Expression Handling:** Scenarios involving empty input, multiple consecutive operators, or malformed expressions were tested to verify the graceful display of error messages.
- **Divide-by-Zero Scenarios:** Critical tests were performed to ensure that any attempt to divide by zero resulted in an "Error" message without crashing the application.

> **All test cases were successfully passed, demonstrating the application's stability and adherence to functional requirements.**

Conclusion:

The Java Calculator Project successfully achieved its design objectives, culminating in a fully interactive and reliable desktop application. This endeavor significantly deepened the understanding of GUI design principles, event handling mechanisms, and user input validation within the Java Swing framework. The practical experience gained from building, testing, and troubleshooting the application reinforced key programming concepts and highlighted the capabilities of Java for developing robust desktop applications.