<Company Name>

Calculator project
Software Architecture Document

Version <1.1>

Revision History

Date	Version	Description	Author
11/9/23	<1.0>	Created document	Sean Brady, Chris Harvey, Naran Bat, Peter Walsdorf, William Morris, Kristin Boeckmann
12/3/23	1.1	Edit based on changes in implementation	Sean Brady, Chris Harvey, Kristin Boeckmann, Naran Bat, Peter Walsdorf, William Morris

Table of Contents

Software Architecture Document

1. Introduction

The purpose of this document is to capture the architecture design of our project. It will cover the use cases, logical view, and interface of our program.

1.1 Purpose

This document's role is to outline an architecture that keeps the project efficient and easy to use. It is a blueprint for the software construction and evolution. The audience is the people building the system, and they will use the document to look over the different pieces of the software that are supposed to be built, the order they are to go about things, and how they will attack building software. Another audience is a supervisor who uses the document to see how different pieces of the system are supposed to be built and in what way software is supposed to be made.

1.2 Scope

This document details how the program is implemented and helps the programmer plan function cohesion and coupling.

1.3 Definitions, Acronyms, and Abbreviations

N/A

1.4 References

N/A

1.5 Overview

The document contains detailed descriptions about the Calculator Project's architecture. The document is organized in such a format that we begin by describing our goals, an overview of the architecture and how we decomposed the design model. Next we describe the specific design modules and packages used in our project, followed by a description of our user interface. Lastly, we describe how the software architecture is of fundamental importance to all parts of our application.

2. Architectural Representation

User Interface (UI) View:

- Model Elements
    - Display for showing input and results
    - Button for control

Calculator Logic View:

- Model Elements:
    - Arithmetic functions (addition, subtraction, etc. More on it below).
    - Variables to store operandi and results.
    - Functions for performing calculations.

Controller View:

- Model Elements:
    - Event handlers to manage user input
    - Logic to coordinate communication between UI and Calculator Logic
    - Error handling mechanisms

3. Architectural Goals and Constraints

This program will be run directly by the user and has no access to the user's system or any server so there aren't any security issues to consider for this product. We will not be using any off-the-shelf product or code. It will be distributed via an executable file that the user will download and run with a command line interface. We will develop this program using a modular coding paradigm with each function being in the same file which all runs together from the main function. We will use github to work together on the project and each file should be able to be changed without affecting the performance of the program.

4. Logical View

The following section will provide a detailed breakdown of the modules, with their services and features, that will make up the final program.

4.1 Overview

Decomposition of this application was done in two steps. First, all of the mathematical operations required by the program were separated out into modules that can be tested and used by other modules. Second, all of the utility functions required by the program to read, parse, perform, and return the results of the input were identified and added to the appropriate modules. This breakdown of functionalities will be useful when implementing our software product as all important operations will be able to be extensively tested to ensure correct results.

4.2 Architecturally Significant Design Modules or Packages

We have one file that includes all of these functions to make the software work:

isOperator:

- service: checks if a character is an operator or not

getPrecedence:

- service: assigns a value to each operation that determines the order

applyOperator:

- service: applies the operators that are being used to the entered values

removeWhitespace:

- service: removes whitespace from an expression and returns the expression string back

isValidVariableName:

- service: cleans var_name and checks if there are any invalid characters in the variable name
- features: removeWhitespace(var_name)

isValidExpression:

- service: makes sure that all parentheses are matched correctly

evaluateExpression:

- service: works as a parser for the program, checks expression for unmatched parentheses, makes sure that if a variable, var_name, is used, that it has a value, evaluates the expression using a list of op and operand values, look through expressions to see if the expression[i] character is being used for a unary sign, and perform order of operations in an expression by checking each character, expression[i]
- features: isValidExpression(expression), removeWhitespace(var_name), applyOperator(op, operand1, operand2, error), isOperator(expression[i]), getPrecendence(expression[i])

main:

- service: prints the forward facing lines, remove whitespace from var_name, checks cleaned_var_name for a valid variable name, evaluates expressions that have variables as well as expressions that do not, and print the output
- features: removeWhitespace(var_name), isValidVariableName(cleaned_var_name), evaluateExpression(expression, variables)

5. Interface Description

Main window:

- input box for user to input equations
- button to run script to check input for errors and evaluate expression
  - the user pressing the ‘enter’ key should also log this event
- window to log outputs
  - outputs are either an error message indicating invalid formatting/characters in the equation or the result of the expression
- button to close window
- button to minimize window

6. Quality

Maintainability:

The clear division of responsibilities among classes simplifies maintenance. Modification or updates in one component are less likely to affect others, promoting ease of debugging and enhancement

Scalability:

While a simple calculator may not demand extensive scalability, the modular architecture lays the groundwork for scalability. If the application were to evolve into a more complex mathematical tool, the existing structure would support such growth.

Usability:

The logical and UI separation enhances usability. The UI package can be redesigned without affecting the underlying logic, allowing for improvements in user experience and interface design without compromising the calculator‘s functionality.

Extensibility:

The modular structure allows for addition of new features or mathematical functions. Additional operations can be integrated without significantly impacting existing code.

Reliability:

Error handling mechanisms enhance reliability by capturing and managing errors gracefully. This prevents unexpected crashes and provides a better user experience.