Math Olympiads

Arithmetic Expression Evaluator
Software Architecture Document

Version <2.0>

Revision History

Date	Version	Description	Author
<25/10/24>	<1.0>	<Create document and assign roles>	<Entire Team>
<11/8/24>	<2.0>	<Finish project and submit>	<Entire Team>

Table of Contents

Software Architecture Document

1. Introduction

This document contains specifications regarding the architecture of the Arithmetic Expression Evaluator (henceforth AEE) as it relates to the previous document, “Software Requirements Specifications” and the use case model within.

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

This document is about code architecture, the step between design and production, and as such does not include information about design or any code used in production.

1.3 Definitions, Acronyms, and Abbreviations

AEE: Arithmetic Expression Evaluator

SRS: Software Requirements Specifications

1.4 References

SRS: Software Requirements Specifications, located in the same folder on GitHub as this document.

Use Case Model: Located with the SRS

1.5 Overview

Section 2: Architectural Representation describes what software architecture is for the current system, and how it is represented. It enumerates the views that are necessary, and for each view, explains what types of model elements it contains.

Section 3: Architectural Goals and Constraints describes the software requirements and objectives that have some significant impact on the architecture; for example, safety, security, privacy, use of an off-the-shelf product, portability, distribution, and reuse. It also captures the special constraints that may apply: design and implementation strategy, development tools, team structure, schedule, legacy code, and so on.

Section 4: Use Case View: N/A

Section 5: Logical View describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities. You should introduce architecturally significant classes and describe their responsibilities, as well as a few very important relationships, operations, and attributes.

Section 6: Interface Description describes the major entity interfaces, including screen formats, valid inputs, and resulting outputs.

Section 7: Size and Performance: N/A

Section 8: Quality describes how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on.

2. Architectural Representation

The program will utilize object-oriented architecture. The architecture is represented by multiple views that

provide a comprehensive description of the program:

- Logical/Structural View: Fundamental building blocks and their relationships
  - Classes define and separate components of the program and determine their responsibilities. These contain specialized functions which perform specific tasks
  - Relationships between classes are defined by inheritance hierarchies, as well as composition relationships and associations between components
  - Interfaces ensure consistent interactions between function types and other components
  - Data structures store and organize data within the program
- Process View: Dynamic and runtime behavior
  - Processing sequences for handling expressions
  - Error handling
  - Data flow between components
  - Execution paths for individual expressions; simple or complex depending on input
- Module View: Organization and dependencies
  - Calculator module which contains primary logic and expression management
  - Parser module handles user input and syntax.
  - Function module contains mathematical operations
  - Utility module contains shared functionality and error handling

3. Architectural Goals and Constraints

- Goals
  - Security
    - Ensure each input is properly sanitized, preventing crashes or other undefined behavior.
    - Prevent unknown behavior by implementing unit tests, particularly on any form of user input.
  - Portability
    - Verify the software works on the target architecture. Portability is not a current focus.
  - Extensibility
    - Ensure the program is able to be extended later with further functionality. The design should allow for further development of the software without requiring major refactoring.
    - New operator functionality should be able to be loaded into the software. Any form of input validation or checking needs to ensure it is using the most up to date list.
  - Distribution
    - Compiled binaries will be created for the target platform and published as a Github release.
  - Team Structure
    - Due to the overall smaller team size, each team member will also be required to have a good grasp of others' roles.

Use-Case View

N/A

4. Logical View

The functionality of the program can be broken into terms of subsystems of a package. This section describes the job of each component and how they interact with each other.

4.1 Overview

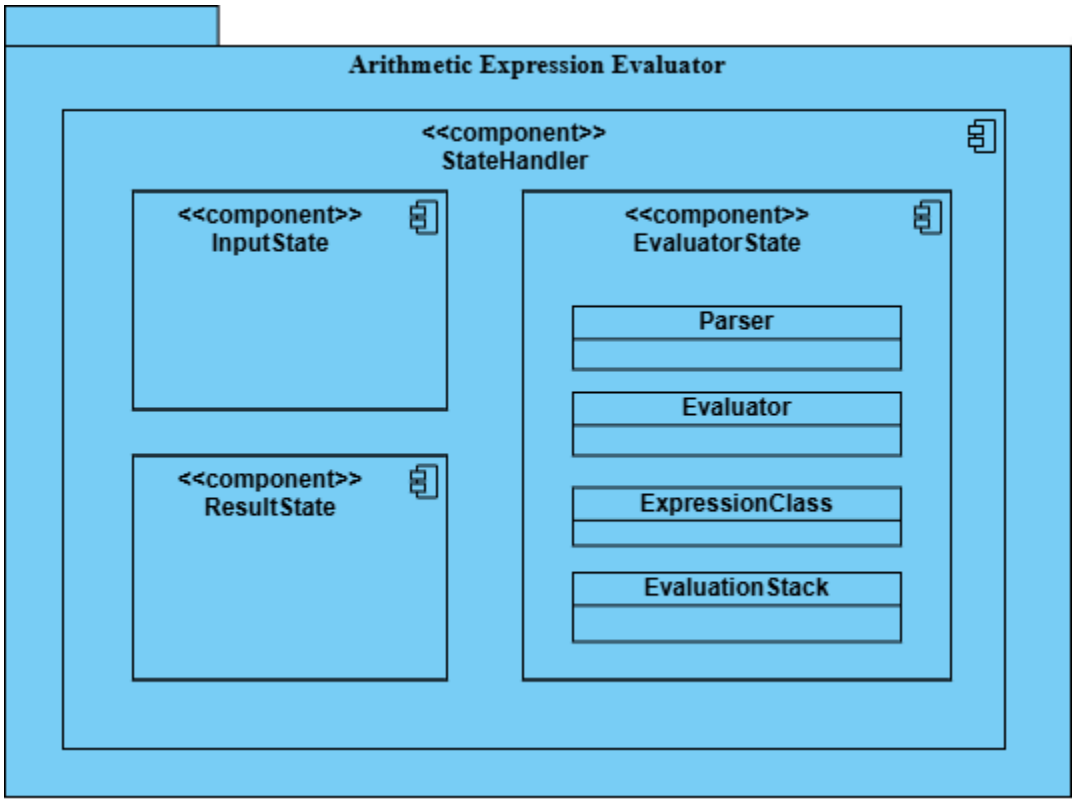
The design model of the program consists of a single main package with multiple components. The package, Arithmetic Expression Evaluator, is composed of smaller components which complete the functionality of the program together.

The hierarchy of the architecture follows standard object oriented programming principles. As such, the

components are made up of classes and subclasses. The three layers can be described as the following:

- **Arithmetic Expression Evaluator**
  - StateHandler
    - InputState
    - EvaluatorState
      - Parser
      - Evaluator
      - ExpressionClass
      - EvaluationStack
    - ResultState

4.2 Architecturally Significant Design Modules or Packages



StateHandler: This component controls the current state of the program. It is used for switching between the ‘input’ mode of the program and showing the ‘output’ of the calculation.

InputState: Handles prompting and receiving the user’s desired expression. Should show a real time display as the user is typing.

EvaluatorState: Handles evaluating the expression received by the InputState. It parses expressions using Parser and evaluates them using the EvaluationStack.

Parser: Receives the user input into the EvaluationStack so the expression is ready to be evaluated by the Evaluator.

Evaluator: Uses the EvaluationStack to work through the expression by going through each operation contained in the stack. Can either return a numeric result value or an error message.

ExpressionClass: Contains one of the multiple components to the inputted expression. Has the left operand, the operator, and the right operand.

EvaluationStack: A sequential stack of operations to be performed in order to calculate the result of the user's input expression.

ResultState: Used to either display the resulting value or an error message. Allows user input to determine if it should go back to the InputState, allowing the user to enter another expression.

5. Interface Description

User Interface Layout

- Main Screen Layout
    - The main interface consists of a numeric keypad, function buttons, and a display screen. Key components include:
        - Display Screen: A large, clear area at the top to show entered values, operations, and the result of calculations.
        - Numeric Keypad: Buttons labeled 0-9 for entering numbers.
        - Operation Buttons:
            - Basic operations: +, -, *, /
            - Additional functions: sqrt, ^ (exponentiation), % (percentage)
            - Other controls: C (clear), = (equals), and . (decimal point)
        - Navigation: Buttons are arranged in a grid layout with the numeric keypad at the bottom, operational buttons above the keypad, and the display screen at the top.
        - Note: The device keyboard can be used to enter numbers and operations.

User Input Requirements

- Valid Inputs:
    - Numbers: Users can enter integers or decimal values using the numeric keypad.
    - Operations: Supported operations include basic arithmetic (+, -, *, /), and special functions (e.g., square root sqrt, exponentiation ^).
    - Clear: Pressing C resets the calculator, clearing all input and results.
    - Equals: Pressing = triggers the calculation based on the current input.
- Input Restrictions:
    - Order of Operations: Users should enter operations in a sequence; multiple operations without an intervening number will trigger an error.
    - Decimal Point: Only one decimal point is allowed per number.
    - Divide by Zero: Division by zero will trigger an error message.
    - Unsupported Input: If the user attempts to input unsupported characters using the device keyboard, they will not appear in the input display.

Output and Error Handling

- Display Outputs:
    - Results: After pressing =, the result of the calculation appears on the display screen.
    - Real-Time Input Display: As users press buttons, the corresponding number or operation is immediately shown on the display screen.
    - Error Messages: Invalid inputs result in error messages such as "Invalid Operation" or "Cannot Divide by Zero." These messages are shown until the user starts a new input sequence.
- Error Handling:
    - If an unsupported sequence (e.g., **, //, sqrt+) is entered, the system clears the last entry and shows an "Invalid Operation" message.
    - If division by zero occurs, the screen will display "Cannot Divide by Zero" until the user

clears the entry.

6. Size and Performance (N/A)

N/A

7. Quality

Extensibility

- Though it is being designed primarily to only expect the operators we provide now, the software's object-oriented structure should still be able to support the ability to add new functionality such as new operators / operations.

Reliability

- The software will be sufficiently coded and tested in order to maximize reliability, there will be minimal or (hopefully) no errors with the program's function and output.

Portability

- The software is not currently being developed with portability in mind.

Privacy and Safety

- The program will not require or retrieve any input from the user other than the mathematical equations and operations they would like the program to compute, so safety / privacy is of no concern.