The University of Kansas

<Project Name>
Software Architecture Document

Version <1.0>

Revision History

Date	Version	Description	Author
06/11/23	1.0	Initial version of the Software Architecture Document	Hannah Smith

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document aims to establish a shared understanding of the system's architectural design, serving as a reference for programmers throughout the project's progression. It covers architectural views, project goals and constraints, logical system review, package breakdown, user interface, and product quality.

1.2 Scope

This document provides an architectural overview of the Arithmetic Expression Evaluator. It affects the overall design of the project and is a good resource to identify the general structure of the final product.

1.3 Definitions, Acronyms, and Abbreviations

Software Architecture – refers to the structure and design of a software system and involves decisions about the organization of components and how they interact; provides a blueprint for building and maintaining a software application, ensuring that it meets its functional and non-functional requirements

Logical View – representation of a software system's structure and organization from a high-level, abstract perspective; provides a conceptual understanding of how a software system works

Package – mechanism for organizing and grouping related code and resources together as well as managing the complexity of large software systems by providing a hierarchical structure for organizing files and classes

Subsystems - self-contained, modular components or smaller systems within a larger, more complex system

User Interface – point of interaction between a user and the software application

Constraints – limitations, rules, or restrictions that must be adhered to during the design, development, or operation of a system

Interface – programming construct or contract that defines a set of methods, properties, and/or events that a class or object must implement; defines a common set of functionalities that multiple classes can adhere to

Object-oriented Programming (OOP) – programming paradigm or methodology that uses objects, which are instances of classes, to design and structure computer programs

Tokenization – process of breaking down a sequence of text into individual units, known as tokens; divides text into smaller, meaningful units to facilitate the analysis

1.4 References

Project Management Plan, 09/22/2023, University of Kansas

This Software Development Plan serves as a document aimed at collecting the essential details needed for the Arithmetic Expression Evaluator project’s control and management. It outlines the strategy being used for software development in this project and stands as the primary plan that team members rely on to guide the project's progress.

Software Requirements Specifications, 10/09/2023, University of Kansas

The Software Requirements Specifications document is a document that describes what and how the software will perform. It also takes stakeholders' needs into account and describes the functionality needed to fulfill these needs. Finally, the Software Requirements Specifications document outlines nonfunctional requirements, design constraints, and anything else needed to give a complete description of the software requirements.

1.5 Overview

The rest of the Software Architecture Document contains the following sections in the given order: Architectural Representation, Architectural Goals and Constraints, Logical View, Interface Description, and Quality. For this Arithmetic Expression Evaluator, neither Use-Case View nor Size and Performance will be discussed.

2. Architectural Representation

There is one main way to represent the architecture for the Arithmetic Expression Evaluator. This is the logical view – this is the manner in which the architecture is decomposed into its individual parts. There are two primary packages; one is end-user facing (containing one primary class) and the other is back-end facing (containing two primary classes).

3. Architectural Goals and Constraints

The software requirements that significantly impact the architecture include expression parsing, operator support, operator precedence, parenthesis handling, numeric constants, and the user interface. The main characteristics of the goals and constraints for the architecture are expression handling, tokenization, and calculation.

4. Use-Case View

N/A

4.1 Use-Case Realizations

N/A

5. Logical View

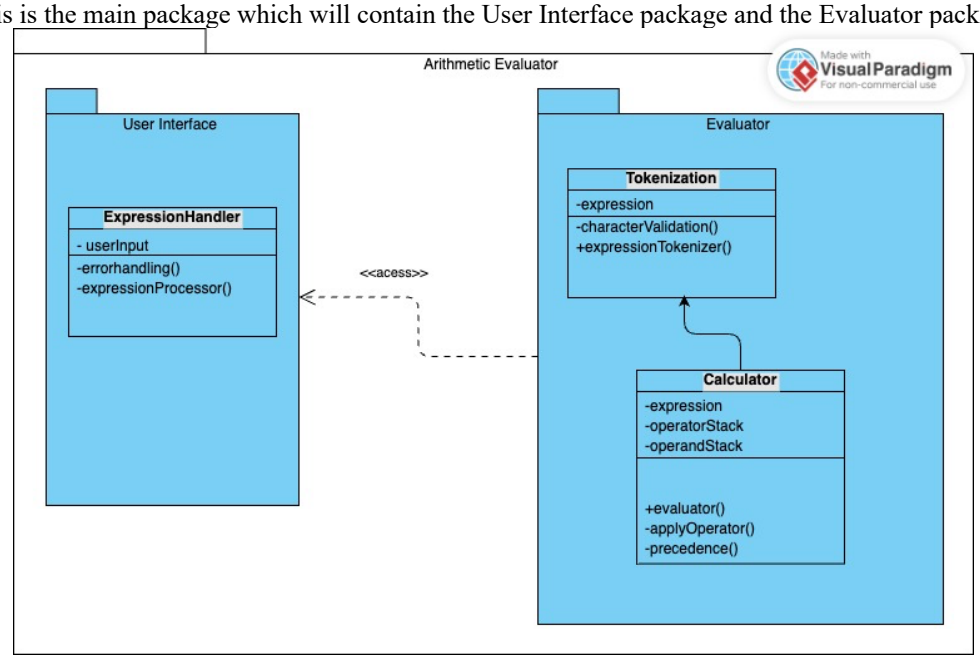
5.1 Overview

This architecture design has a single package that then contains two other packages. One package is focused on interfacing with the user, taking an input and returning the correct response. The other package is focused on tokenizing the expression and then evaluating it.

5.2 Architecturally Significant Design Modules or Packages

5.2.1 Arithmetic Evaluator

This is the main package which will contain the User Interface package and the Evaluator package.



5.2.2 User Interface

This package will handle interactions with the user. It will also display errors thrown by the Evaluator package in easily understandable terminology for the user. The ExpressionHandler class will execute the functionality of this package. The ExpressionHandler, will have a userInput attribute, an errorHandling method, and an expressionProcessor method.

5.2.3 Evaluator

This package will tokenize and evaluate a user-given expression. It will also detect any errors in the expression and throw a corresponding error code. This package will have two classes: Tokenization and Calculator. The Tokenization class will tokenize a user-given expression and it will validate the characters. The Calculator class will evaluate the expression and detect any errors within it.

6. Interface Description

The user interface will be via command line and will ask the user to input an expression to be evaluated. It will output a result to the user with “Result: (answer)” under the input expression. These are some valid inputs and resulting outputs:

Enter an Expression:
3+4
Result: 7

Enter an Expression:
8 - (5 - 2)
Result: 5

Enter an Expression:
10 * 2 / 5
Result: 4

Enter an Expression:
2 ^ 3
Result: 8

Enter an Expression:
4 * (3 + 2) % 7 - 1
Result: 5

Enter an Expression:
(((2 + 3))) + (((1 + 2)))
Result: 8

The Arithmetic Expression Evaluator will be able to evaluate valid expressions including addition, subtraction with parentheses, multiplication & division, and exponentiation and support mixed operators, complex addition with extraneous parentheses, mixed operators with extraneous parentheses, nested parentheses with exponents, combination of extraneous and necessary parentheses, extraneous parentheses with division, combining unary operators with arithmetic operations, unary negation and addition in parentheses, negation and addition with negated parentheses, unary negation and exponentiation, and combining unary operators with parentheses.

7. Size and Performance

N/A

8. Quality

All defect metrics will be gathered and recorded as Change Requests.

All deliverables will be reviewed as needed to ensure that each is of acceptable quality.

All defects found after release/integration will be recorded and tracked as post-integration Change Requests.

Security will be managed via public and private methods among classes.

The architecture is structured such that additional operators can be implemented to improve the quality of the final product.