<Calculator>
Software Architecture Document

Version <1.1>

Revision History

Date	Version	Description	Author
<30/10/2023>	<1.0>	First additions	Xavier Ruyle
<08/11/2023>	<1.1>	Finishing the document	Xavier Ruyle

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

The purpose of this document is to convey the design of the calculator project in regards to the implementation. It will specify the details and design of each class and how they relate to meeting the requirements needed for the calculator.

1.2 Scope

The Software Architecture Document is related to the design and structure of the calculator project as a whole.

1.3 Definitions, Acronyms, and Abbreviations

UI: User Interface

IFN: Infix Notation

PN: Postfix Notation

EXPR: Expression

1.4 References

N/A – no references at this moment

1.5 Overview

The rest of the Software Architecture document will detail the architectural representation as well as its goals and constraints. It will then describe the use cases and the subsystems that will be used.

2. Architectural Representation

3. Architectural Goals and Constraints

4. Logical View

4.1 Overview

The calculator will utilize a tokenizer to divide user inputted mathematical expressions into individual token strings which will then be sent to a queue. It will then send the tokenizer result to a parser which will use an algorithm to convert the tokenizer queue to PN. The PN will be stored in a queue. The PN will then be calculated using a basic algorithm using a stack using the calculator class. Finally, a display manager class will output the final result obtained from the calculator class.

4.2 Architecturally Significant Design Modules or Packages

Calculator Project (Package)

- Tokenizer
    - **Description**: The tokenizer will tokenize a given input string from the user into the **expressionQueue**. Given an input from the user, it will translate each part of that input into meaningful **ints**, floating points, operators, represented by a string. It will also smartly strip whitespace.

- Decimals will not signal a new token must be made. Instead, the tokenizer will know that it is a floating point
- EX: "2+ 3.0004 * 2" - > ["2","+","3.0004","*","2"]

- Parser
  - **Description**: Parses a given expression queue that has been tokenized by the tokenizer. Uses an algorithm to convert the infix notation (expressed passed by user input) to postfix notation (a notation which is easier to calculate with). The **infixToPostfix** algorithm uses a queue and a stack to temporarily place operators in the **operatorStack** and place them back in the **outputQueue** when needed to facilitate PEMDAS requirements. The **outputQueue** is used for future use for the calculator class. For precedence, the parser uses a map which corresponds to PEMDAS rules
- Calculator
  - **Description:** Calculates a given postfix expression which is obtained from the parser. Result is stored in a final result attribute.
  - Uses a stack to calculate the PN EXPR.
- Display Manager
  - **Description:** The display manager class will provide an interface for the user to input data to the project and expect an output corresponding to that input. It will work with the calculator class to output a final answer for any input given by the user.

5. Interface Description

The interface that the user will interact with is the command line. The user will be able to input mathematical expressions which can handle the following operators, +, -, *, /, %, ^ or **. Expressions must be mathematically valid or else there will be errors.

The command line will show the following

Input Expression: 3+4 *(user input)*

Answer: 7

Input Expression: (((((5 - 3))) * (((2 + 1))) + ((2 *

3)))

Answer: 12

6. Quality

The calculator is designed to ensure that the classes address various quality attributes.

In regards to extensibility each class is built to have its own purpose. For instance, the calculator can add new operators if the project were to be expanded. The parser could be changed to address a change in operator precedence. And the tokenizer could change to address other forms of multiplication (e.g 5(2+3) = 5*(2+3)). The calculator is reliable as the project accounts for bad user input and has multiple different types of errors which will inform the user what went wrong. The project also accounts for portability as the command line is hardware agnostic.