RPN Calculator

Software Design Document

Sze Yick

Date: 13/06/2017

Introduction:

This software design document describes the architecture and system design of the RPN (Reverse Polish Notation) calculator, presented as a coding exercise by Airwallex.

System Overview:

The system is responsible for performing several basic calculator operations provided by a user. Input is provided as a string through the command line, and the system processes each string request and outputs the current state of the calculator along with any warnings raised during processing.

Input is expected to be a string comprised of space separated numeric values and operators. Numeric values are pushed onto the internal stack and maintained, and operators perform their respective operations on the values in the stack.

The system has the capability to perform the following operations:

1. **Add** – Input represented as a +, adds two values on the stack together.
2. **Multiply** – Input represented as a *, multiplies two values on the stack together.
3. **Divide** – Input represented as a /, divides two values on the stack.
4. **Subtract** – Input represented as a -, subtracts a value from another value.
5. **Square Root** – Input represented as “sqrt”, performs the square root on a value on the stack.
6. **Undo** – Input represented as “undo”, undoes the previous operation.
7. **Clear** – Input represented as “clear”, removes all elements from the stack.

System Architecture:

Attached to **Appendix A** is a class diagram that represents the system architecture. The following section provides an overview of the design.

RPN Calculator:

This is the main component of the system. Represented in the design as the **RPNCalculator**, it is responsible for processing the user input to perform the requested operations for the calculator and to output the current stack state when input has been processed.

This component maintains the current stack that the calculator uses to perform operations, along with a map that defines the operations that are available to the calculator. Execution of the requested operation is delegated to an external class to reduce the complexity of this component.

Internally, this component also maintains the history of the previous stack states. This is done to allow for previous states to be restored when the undo command is executed.

Operations:

Each operation (i.e. +, -, *, sqrt, clear, undo) is managed by an individual class. The **RPNCalculator** creates and maintains a reference of each and utilizes the strategy pattern to evaluate during runtime which operation to invoke based off the user input.

To do this, all operations are required to implement the **IOperatorStrategy** interface. Doing so allows for the **RPNCalculator** to generically refer and call the required operation based off the input string.

The operation from the input string is mapped to an **OperatorEnum** that allows the calculator to know which strategy interface to invoke.

Additionally, a **BasicOperation** abstract class is also created that is extended by any operation that requires two stack elements to perform an operation.

A special case to take note of is for the division operation. There is a case, where it is possible have a stack that contains 0, and a division operation is executed. If the denominator is the 0 value, then it will cause a divide by zero error. To handle this case, if the denominator is 0, the division is not invoked.

Number Utilities:

A utilities component that is represented in the design as the **NumberUtils** class. It is responsible for providing functionality to format a number for output and to determine if a given string represents a numeric value.

Data Design:

The specification for the RPN Calculator defines that a stack is to be used to maintain the values it performs operations on. For this system, the stack implementation selected to be used is the **ArrayDeque**. The system uses this data structure to represent the stack due to the performance optimizations that it provides over the standard stack implementation provided by Java.

Also, the **RPNCalculator** maintains a map of available operator strategies keyed by **OperatorEnum**. This is to be maintained in an **EnumMap** again for the performance optimizations and suitability since all keys in the map are enums.

Requirements:

System requirements are provided in the attachment titled **Airwallex - RPN_Calculator.pdf**

Appendix:

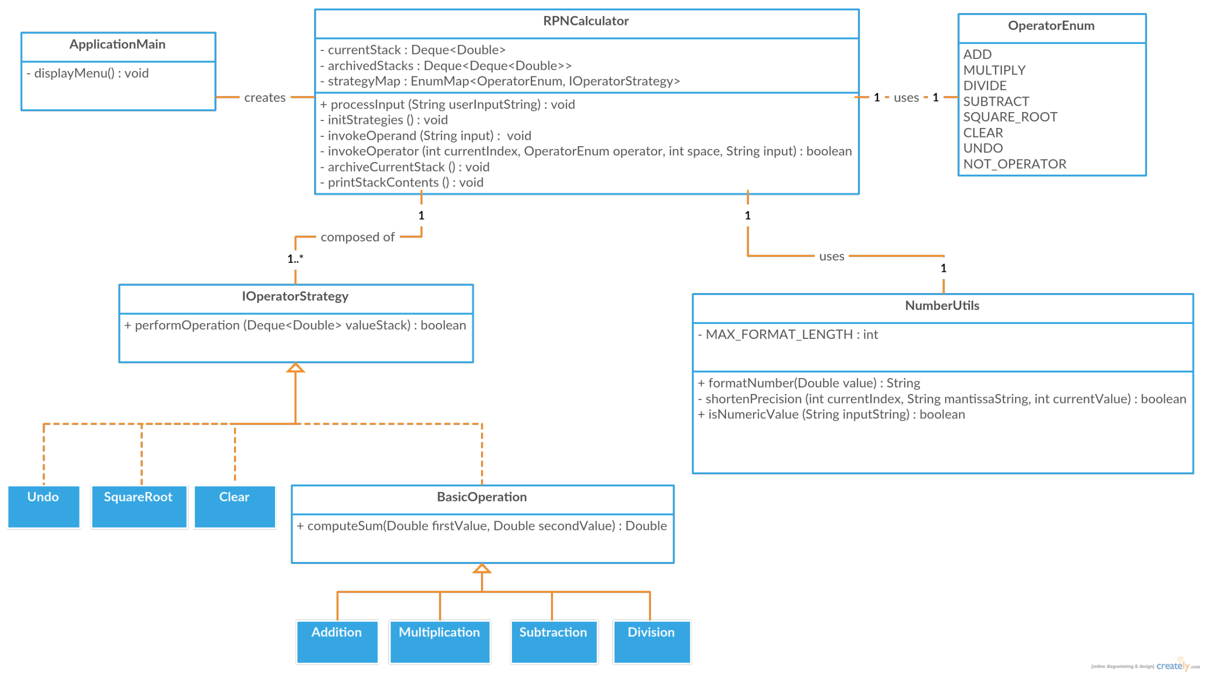
ApplicationMain
- displayMenu() : void
creates
RPNCalculator
- currentStack : Deque<Double>
- archivedStacks : Deque<Deque<Double>>
- strategyMap : EnumMap<OperatorEnum, IOperatorStrategy>
+ processInput (String userInputString) : void
- initStrategies () : void
- invokeOperand (String input) : void
- invokeOperator (int currentIndex, OperatorEnum operator, int space, String input) : boolean
- archiveCurrentStack () : void
- printStackContents () : void
1 - uses - 1
OperatorEnum
ADD
MULTIPLY
DIVIDE
SUBTRACT
SQUARE_ROOT
CLEAR
UNDO
NOT_OPERATOR
1
composed of
1..*
1
uses
1
IOperatorStrategy
+ performOperation (Deque<Double> valueStack) : boolean
NumberUtils
- MAX_FORMAT_LENGTH : int
+ formatNumber(Double value) : String
- shortenPrecision (int currentIndex, String mantissaString, int currentValue) : boolean
+ isNumericValue (String inputString) : boolean
Undo
SquareRoot
Clear
BasicOperation
+ computeSum(Double firstValue, Double secondValue) : Double
Addition
Multiplication
Subtraction
Division
creately

Figure 1. System Architecture Class Diagram.