Reverse Polish Notation Calculator

Ben Bonavia 2017

Introduction

This document intends to describe the design of the Reverse Polish Notation Calculator.

Design Considerations

Input Methods

The requirements provided explicitly state that a graphical interface is not required, as such it was decided that the calculator be implemented as a API which is used by the RpnCalculator application.

The application can then have multiple input strategies which have been considered to include command line (stdin) and file inputs. This allows for a GUI input strategy to be considered at a later time.

The Calculator

The calculator needs to be able to perform simple arithmetic operations as well as control operations. These include:

- Addition +
- Subtraction -
- Multiplication *
- Division /
- Square Root sqrt
- Undo Last Operation undo
- Clear Current Stack clear

It has been decided that in the performing of these operations, the calculator itself should not store any data but instead be a service which only acts on its inputs. Here the inputs are the input string and a memory object which should hold all the data.

Therefore a facade is required to hold the memory and pass it and the received inputs to the calculation service.

Calculator Memory

The calculator memory needs to hold the current stack as well a history of previous operations.

The stack has been decided to be implemented as a Deque<> as opposed the obvious Stack<>. This is mainly due to the safety of the Deque<> which only gives visibility of the top and bottom of the collection. A Deque<> is a child of Vector<> which allows access to any element.

The implementation of a Deque<> chosen is an ArrayDeque<> as it provides the fastest access add and remove elements when adding/removing the last item.

- Removing elements: O(1)
- Adding elements: O(1) to O(n) (if the array needs to be expanded, n is the array length)

The memory also needs to hold a history of all operations. It has been decided to implement this as a "stack of stacks" and on every undo simply replace the stack with a previous version. When there is no history recorded, the stack is cleared as this is accurately the previous state.

Handling Of Inputs

The inputs cannot be processed as strings and need to be converted appropriately. To keep precision up to 15 decimal places, it has been decided that numbers should be stored as a Double.

It has been decided that on-the-fly parsing and validation of each input be performed before being handled. This means internal input structures are not required and input items are only ever processed once. Validation is required to determine if an operation can be

performed before handling it.

Operators are stored in an enum which also holds its string representation and the number of operands required to be performed.

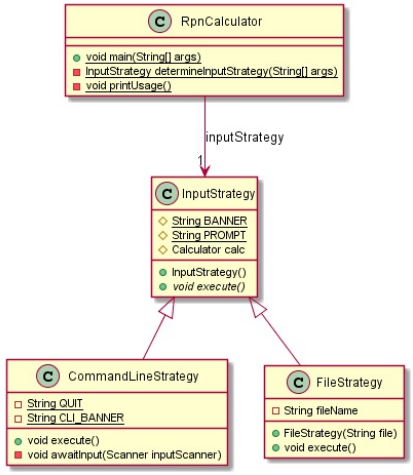
Where parsing or validation of the inputs fail, an exception should be raised and caught by the facade to display the appropriate message.

- Parsing failure - UnknownOperatorException
- Validation failure - InsufficientParametersException

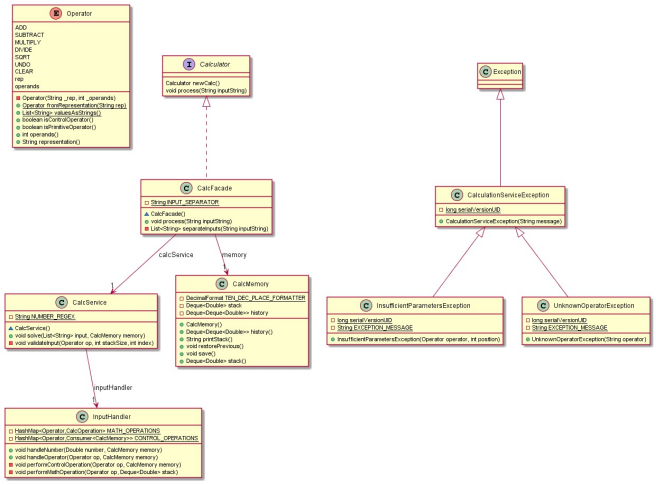
To save the file count and the complexity involved, functional interfaces are utilised to perform the operations. This allows the operations to be stored in a map which puts all the operations together which makes it easier to understand.

Architecture

This section contains the class diagrams which describe the architecture of the system.



The Application Input



The Calculator API