CSC 413 Project Documentation

Spring 2020

Trenton Smith

920056368

CSC413.01, SPR 2020

https://github.com/csc413-01-spring2020/csc413-p1-Trenton-Smith.git

Table of Contents

1 Introduction

1.1 Project Overview

The goal of this assignment is to create a functioning calculator application. It must incorporate a user interface through which the user can input operands and operators and calculate a result from the given information. It should be able to handle all submitted expressions with the correct order of operations being followed. The application should also be able to be terminated upon closing its window.

1.2 Technical Overview

The calculator application is split into four main classes, the Evaluator, the EvaluatorUI, the Operand, and the Operator:

The Evaluator class contains both the operand and operator stacks – our chosen data structure for which we will use to store our user input, as well as the evaluateExpression() method which functions to pop() values from the stacks, perform the given mathematical operation (from Operator subclasses), and push the result back onto the operand stack for access.

The EvaluatorUI class is simply for the user to interact with the program. It functions by utilizing JFrame to create a window resembling a calculator, and creates buttons for each available token (0-9, "+-*/()C^CE="). Once the user clicks on a button, the value is added to a string in the northern text field, which then in turn is sent to the evaluateExpression() after the user clicks "=".

The Operand class functions solely to parse user input into integers to store in the operandStack.

The Operator class is an abstract class which contains two methods passed into all its subclasses (one for each operator i.e. + and - but also for parenthesis), as well as the HashMap containing all our keys (the operators) and their associated values (which are constructor calls to each respective operator). The primary functionality of the Operator subclasses revolve around the priority() and execute() methods. The priority method is used to apply an integer value to each operator representing their standing in the order of operations hierarchy. The execute method takes two given operands and calculates their result when combined by the named operator.

1.3 Summary of Work Completed

Application has complete functionality including working calculator UI as well as correct implementation of the provided algorithm when calculating results.

Evaluator:

- Added parenthesis to String delimiters
- Added getMethods for Operand / Operator stacks (though not implemented)
- Added complete functionality to the evaluateExpression() method
    - Functions by storing operands and operators in their respective stacks based on their hierarchy, then checks for flags [(parenthesis) if any] which are purposely negative in order to trigger calculation blocks.
- Added additional tests in the EvaluatorTest.java

Operand:

- Added functionality to all methods and constructors

Operator:

- Added HashMap with keys for all the operators, and values with their respective subclass constructor calls
- Added all Operator subclasses with overridden abstract methods
    - Treated parenthesis as Operator subclasses as well, but gave negative values for their priority in order for them to function as flags instead of as operators
    - Also made the parenthesis' execute return null

Evaluator UI:

- Added functionality to all buttons in the form of ActionEvents
    - Added value printing for user readability for operands and operators
    - Added calculation by implementing Evaluator class' evaluateExpression() method

2 Development Environment

- ➢ JDK 13
- ➢ Intelli J IDEA 2019.3.2 (Community Edition)
- ➢ Windows 10 Operating System

3 How to Build/Import your Project

1. In terminal type:

```
git clone https://github.com/csc413-01-spring2020/csc413-p1-Trenton-Smith.git
```

2. Import project into IntelliJ IDEA

2.1. Click:	"Import Project"
2.2. Select:	"calculator" folder as source root
2.3. Select:	"Create project from existing resources"
2.4. Click:	"Next" until "Finish" *NOTE* If you do not see JARS or do not have a JDK, please see A1Import.pdf in the repo for instructions
2.5. Click:	"Finish"

4 How to Run your Project

1. Open project in IntelliJ IDEA
2. Navigate to EvaluatorUI.Java
3. Right Click and click "Run EvaluatorUI.main()" or click the green triangle in the IDEA
4. To exit application: close calculator window

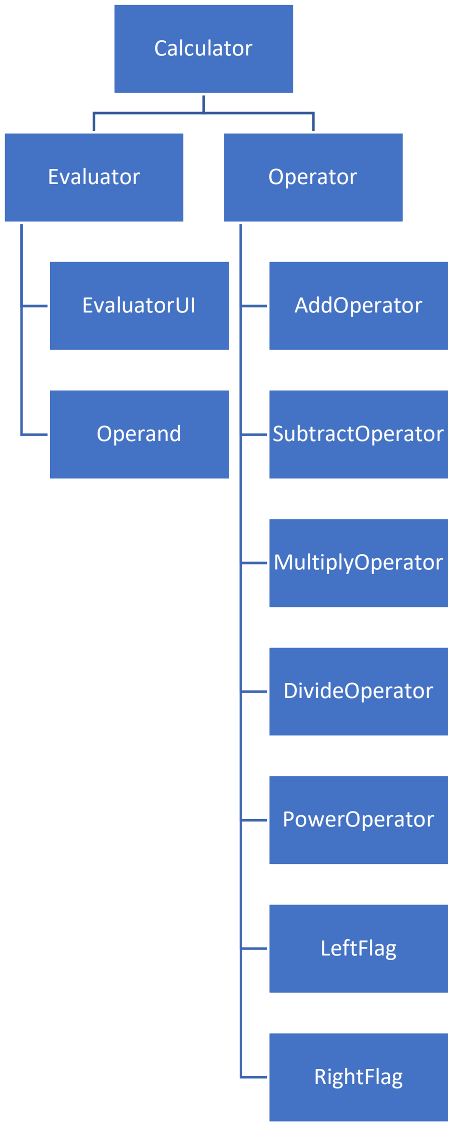
5 Assumption Made

- The EvaluatorDriver, Result, and Test .Java files were all correctly implemented

6 Implementation Discussion

The main design choice was to include the parenthesis as Operator subclasses. I decided that it was easiest (within my degree of knowledge) to go ahead and just include them as I would for any other Operator subclass, but purposefully give their priority() method a negative int for the return value, and returned their execute() method as null; This was to signify them as flaggers for when I navigate through the operator stack and pop() off non parenthesis operators.

6.1 Class Diagram



7 Project Reflection

I learned a ton about Java and overall application structure with this assignment. Although it was daunting due to my lackluster Java expertise, I am very happy with the result. I feel that I could have spent more time with the UI but at this point (1:20am on the day it is due), I need to call it good as it is – which to be clear, is perfectly fine and functioning as intended, but it lacks the flash which would have helped for extra credit. In terms of time management, I know I cut it close (less than 24 hours till turn in), but I essentially had to reteach myself how to code since I hadn't done it in close to a year while taking other classes. Needless to say, the struggle was real, but I feel that I definitely made some headway in getting myself back up to speed. Particularly, this project has helped familiarize myself with the debugger, which I'm sure will be my best friend for assignment II.

8 Project Conclusion/Results

The calculator application runs as intended and passes all the tests provided (as well as some others I hardcoded in for myself). It taught me how to organize basic project structures, refreshed my memory on many Java paradigms, and increased my ability to analyze code (mine and others'), use the debugger tool, and scrutinize algorithms.