CSC 413 Project Documentation

Fall 2018

Cory Lewis

ID: 917359162

CSC413.01

https://github.com/csc413-01-fa18/csc413-p1-mecharmor

Table of Contents

1 Introduction

1.1 Project Overview

This project is to create a basic calculator utilizing our knowledge of Java, Data Structures, and Algorithms. Attaching a GUI to our app will create a simple user interface for users to interact with the software by other means than the command line.

1.2 Technical Overview

Because of how this program was designed, if the command line is going to be used to enter arguments the following operators are acceptable for calculation: "-+/*^". Using the calculator GUI, the user will be restricted to the type of input they can enter.

1.3 Summary of Work Completed

In this project I created subclasses that inherit an abstract class of type Operator. Once this was completed I implemented certain methods within the Operator class and Operand class. I implemented the display for the GUI and the string that was generated by the button presses was then passed to the eval function in Evaluate.java.

2 Development Environment

This project utilized the JAVA jdk (version: jdk-10.0.2). IntelliJ (2018.2.3) was used to develop this application. Gradle was used as the wrapper for the project. This project was developed using test driven development. The tests can be ran individually using intelliJ by simply selecting the test and clicking the green arrow on the left hand side.

3 How to Build/Import your Project

Importing your Project

- Go to File ➔ new ➔ import project from existing sources
- Navigate to your directory and click ok
- Make sure you select 'use gradle wrapper'
- Click ok until you are into the project
- Build the project by selecting build at the top and choose 'Build Project' (Ctrl+F9)
- Gradle will be added to your computer if it is not already installed

4 How to Run your Project

Running the project within intelliJ:

1. Command Line usage
    a. Select the EvaluatorDriver class and click the green arrow at the top left hand side where the class declaration is.
    b. The program will run and the console output will be displayed below
2. GUI Usage
    a. Select the EvaluatorUI
    b. Click the green arrow on the top left hand side to launch the program
    c. Click the 'x' on the top right to close the GUI/application

5 Assumption Made

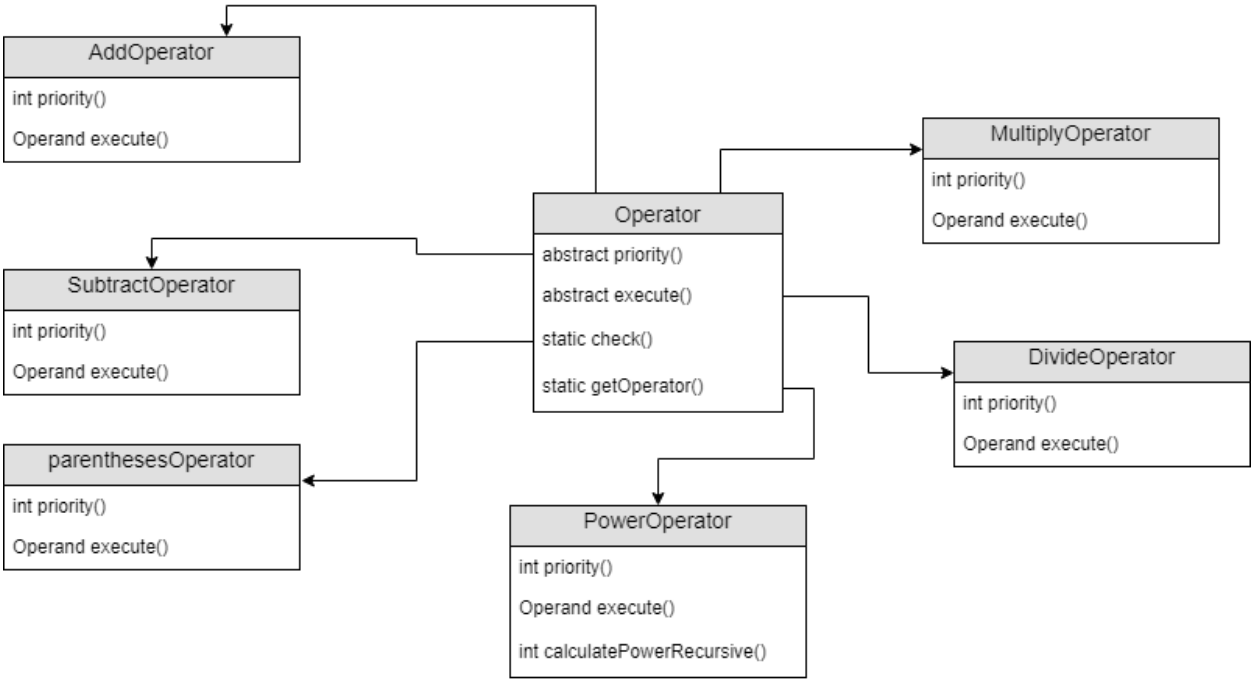
This project was designed to meet the requirements of the client/grader which implies that it's use cases are limited to correct input that will not exceed the scope of this software's capabilities. Good input would be given so the backend development would not require as much validation (ex| "1+++++5").

6 Implementation Discussion

Implementing this program, I needed to develop the structure of the classes through implementation of the Operator abstract class. The Add Operator, DivideOperator, MultiplyOperator, PowerOperator, SubtractOperator, and ParenthesesOperator all extend the Operator abstract class. Some issues I encountered with using the Operator class is in the Evaluator class I needed a way to dynamically downcast the Operator class to one of its child classes based on the given operator type. I solved this solution by referencing the static HashMap with the .get method to retrieve the down casted object type which executed after I validated the key existed in the HashMap. I created a parenthesesOperator with a priority of 4 so I can implement the parentheses using the same logic the other operators used. For the GUI I created helper methods to better direct the program flow with the greatest modularity for possible improvement or modification in the future.

6.1 Class Diagram

Below is a class diagram displaying he inheritance of the child classes from the parent Operator class:



7 Project Reflection

This project helped me reflect on previous knowledge about data structures, polymorphism, and class inheritance. I learned a great deal in terms of proper class hierarchy for real world

development and efficient practices for utilizing polymorphism as a technique for efficient and clean code. Additionally, this program is extremely modular so if changes needed to be made then they could easily be modified for add-ons in future versions. The hardest part for me in this project was implementing the Evaluator class, due to the infix expression algorithm being confusing in general.

8 Project Conclusion/Results

I ran all the Test classes, refactored my code, and cleaned my comments for the maximum efficiency and readability. Originally, I hard coded my checks for the Operator class, but I realized the proper implementation of the HashMap and utilized that properly for checks and object creation. The GUI implementation used many helper methods to increase modularity and decreased copying and pasting unnecessary code. I used a Deque for keeping track of the user input on the GUI and created a merge helper method to combine strings if the user enters a # and the follows a # instead of an operator. Overall this program is extremely modular and offers many clear helper methods for future developers to improve upon the design easily without rewriting the entire program.