CSC 413 Project Documentation

Fall 2018

Aung Hein

917649101

CSC413-02

https://github.com/csc413-02-fa18/csc413-p1-VagueClarity

Project Overview

This project is a calculator that can do simple math operations.

Technical Overview

This project creates two programs:

1. An object that evaluates mathematical expressions
2. A GUI for object in (1)

Summary of work completed

I created a static HashMap in the abstract class, Operator for all required operators then created the operator classes that extends off of Operator. Referencing the operator classes in the HashMap, I was able to calculate tokenized expressions given to the Evaluator class using two stacks, one for Operand and Operator the other. Finally, when everything worked out in the Evaluator class, I used if statements in EvaluatorUI to do some basic checks and print out results to the GUI calling Evaluator for the calculations.

Development Environment

a. jdk 1.8.0_121
b. IntelliJ IDEA

How was it built

I started off with cloning the given repo off GitHub then imported the calculator folder as a Gradle project using IntelliJ. The IDE automatically set up the pathing and I had to manually gave it the path to my jdk. The project got built and everything went smoothly.

How to run your project

I run the EvalutatorUI class using Intellij and pressing Ctrl-Shift-F10. The project has Gradle integration, so a Gradle File might be needed to compile. This project can be ran without using the UI file by executing the EvaluatorDriver class instead. However there will not be a Calculator formatted GUI.

Assumptions made when designing and implementing your project

I assume that the user will put correctly formatted input without trying to break the code.

In additional designed the project in a way that any class should be interchangeable without breaking the code.

Implementation Discussion

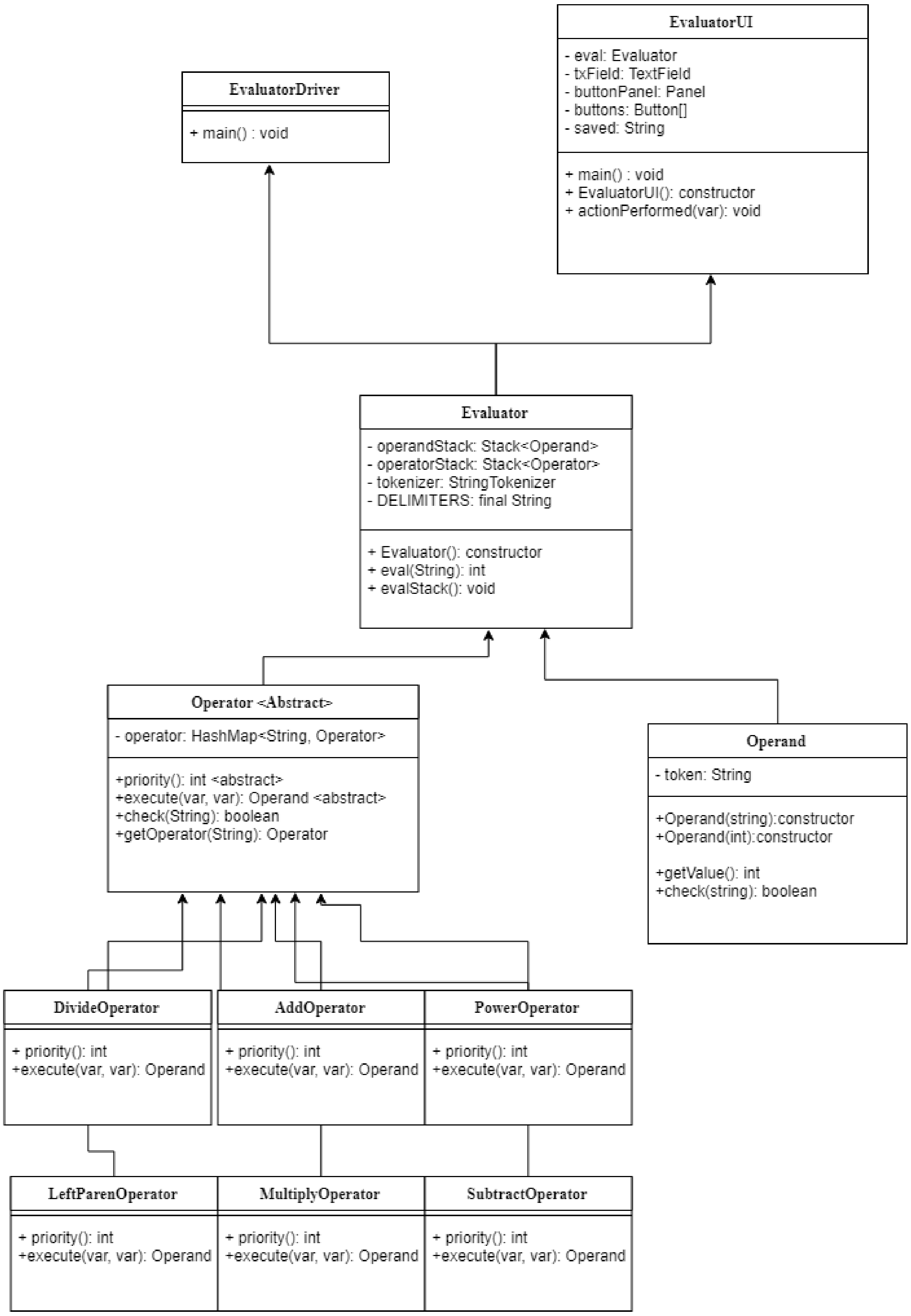
The expressions given are evaluated using a stack for storage for an operator and an operand. I have added a parentheses as part of the operators so that the user can use them. The algorithm looks for the right parentheses to evaluate the expression between the parentheses then it does the rest normally checking for priority. I used private scope for variables that the user should not be able to access such as the HashMap. The evaluator UI is independent of the evaluator class such that a new working evaluator class can be replaced and the code would still work.

Project Reflection

It took me a while to understand how the stack was implemented to evaluate the given expression. But once I understood it, the rest was easy for me to figure out and solve. I also learned how to initialize a group of static data variables in an abstract class.

Project conclusion and results

This program can do simple math operations using stack as an effective data structure to efficiently evaluate expression. It also leaves the back end for the calculations while the GUI give inputs to the evaluator class.


EvaluatorDriver
+ main() : void
EvaluatorUI
- eval: Evaluator
- txField: TextField
- buttonPanel: Panel
- buttons: Button[]
- saved: String
+ main() : void
+ EvaluatorUI(): constructor
+ actionPerformed(var): void
Evaluator
- operandStack: Stack<Operand>
- operatorStack: Stack<Operator>
- tokenizer: StringTokenizer
- DELIMITERS: final String
+ Evaluator(): constructor
+ eval(String): int
+ evalStack(): void
Operator <Abstract>
- operator: HashMap<String, Operator>
+priority(): int <abstract>
+execute(var, var): Operand <abstract>
+check(String): boolean
+getOperator(String): Operator
Operand
- token: String
+Operand(string):constructor
+Operand(int):constructor
+getValue(): int
+check(string): boolean
DivideOperator
+ priority(): int
+execute(var, var): Operand
AddOperator
+ priority(): int
+execute(var, var): Operand
PowerOperator
+ priority(): int
+execute(var, var): Operand
LeftParenOperator
+ priority(): int
+execute(var, var): Operand
MultiplyOperator
+ priority(): int
+execute(var, var): Operand
SubtractOperator
+ priority(): int
+execute(var, var): Operand