CSC 413 Project Documentation

Fall 2018

Keith Rich

915799591

CSC 413.01

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

Table of Contents

[1 Introduction 3](#_Toc522827688)

[1.1 Project Overview 3](#_Toc522827689)

[1.2 Technical Overview 3](#_Toc522827690)

[1.3 Summary of Work Completed 3](#_Toc522827691)

[2 Development Environment 3](#_Toc522827692)

[3 How to Build/Import your Project 3](#_Toc522827693)

[4 How to Run your Project 3](#_Toc522827694)

[5 Assumption Made 3](#_Toc522827695)

[6 Implementation Discussion 3](#_Toc522827696)

[6.1 Class Diagram 3](#_Toc522827697)

[7 Project Reflection 3](#_Toc522827698)

[8 Project Conclusion/Results 3](#_Toc522827699)

# Introduction

## Project Overview

Implementation and construction of calculator that follows the rules P.E.M.D.A.S. as is Parenthesis, Exponents, Multiplication, Division, Addition, and Subtraction.

## Technical Overview

* Mapping a HashMap with operators with constructors that perform said operation.
* Implementing two separate stacks. One for operators and one for operands.
* Following the algorithm found here: http://csis.pace.edu/~murthy/ProgrammingProblems/16\_Evaluation\_of\_infix\_expression  
  s) That was provided in the assignment descriptions.
* Pushing, pooping (heh), and performing the operations in the specified order of the algorithm.
* Lastly returning the value of the expression which is inputted.

## Summary of Work Completed

* Evaluator Class

The evaluator class was the bulk of the work. Created a function that runs outside of the main eval function that performs the operation once a “)” is found.

Towards the middle of the eval function a few checks were implemented. Checks include a check for the function call once “)” is found. Check if the operator stack is empty. If one operator priority is higher then the new operator. Checking to push or pop depending on what operator is added ex “(” and “)”.

Running through one more check on weather or not the operator stack is empty and operator stack priority. Then finally returning the top of the operator stack.

* EvaluatorUI Class

Created a switch statement for the following cases. “=”, “C”, “CE”, and default to other buttons pushed.

“=” evaluate the expression.

“C” and “CE” are implemented the same way and are done by setting the text field to “”.

Default enters the text field and action command and displays it in the calculator view.

* Operand Class

Constructor that takes in a string token and converts it to an integer as well as a integer constructor that returns said integer. A get value functions that returns the values of said token. And lastly a check to find if the token is a int.

* PEMDAS Classes

Constructors that build objects that contain a priority and a operation of the operands that are passed to said constructor. These objects include add, subtract, multiply, divide, and powers.

* Operator Class

This is where the implementation of a HashMap with the corresponding operators.

Because the HashMap is an abstract class we must access it through a public function called “functionCall”.

A function that returns the operator from said HashMap.

Lastly a check for if the token is an operator.

* Open & Close Class

Constructor that sets a unique priority and returns a Operator that contains nothing of value.

# Development Environment

* 1.8 (java version “1.8.0\_171”)
* IntelliJ IDEA Ultimate

# How to Build/Import your Project

* If you have GitHub and a computer and fingers.
* Cd to a directory of your choosing
* Git clone the URL of the repository
* In your favorite compiler select “open from exiting sources”
* Find “calculator”, select it, specify that you want to wrap the program in Gradle
* Select one of the buildable classes ie. EvaluatorDriver, EvaluatorUI, or any of the tests under “test” folder under src.
* Right click on the chosen class and click “Build” under the class chooses.

# How to Run your Project

* After building the said class press the green sideways play button located at the upper right side of your IDE

# Assumption Made

I remember getting about halfway through the implementation of the algorithm from the website that was provided and thinking. “Well this wont work if you have a large string with lots of different priorities.” Turns out it works super well. So long as you don’t have “))”.

When first working within the project I didn’t realize that other classes where needed (such as AddOperator, OpenParenthesis ect) and spend a large amount of time working all of these classes in the Operator class. It wasent till class the following week that I was told we should be keeping all of these functions in separate classes.

Another challenge was because the HashMap was inacessable due to abstract having to determing weather to use an interface or a function came into mind. Oviously function was the best choice but determiening that only came after a class discution as well.

# Implementation Discussion

## Class Diagram

# Project Reflection

# Project Conclusion/Results