CSC 413 Project Documentation

Spring 2019

Brian Lai

916818167

CSC413.03

[*https://github.com/csc415-03-spring2019/csc413-p1-blai30*](https://github.com/csc415-03-spring2019/csc413-p1-blai30)

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

The project is a calculator that reads math expressions and calculates a result using the elementary rules of basic operations, applying the traditional order of operations.

## Technical Overview

This Expression Evaluator calculates a result based on a mathematical expression given as a string input. The string is tokenized into two stacks: an operand stack for storing numbers and an operator stack for storing mathematical operators. When evaluating the expression, operands and operators are popped from the top of the stack while following the traditional order of operations to obtain the correct result.

## Summary of Work Completed

Created new subclasses from the Operator class to serve as different operators that execute different calculations. Initialized a static HashMap in the Operator class that stores each operator subclass. Filled out the eval function in the Evaluator class to have it tokenize the expression into operand and operator stacks and calculate the expression based on order of operations including parentheses. Added a check for if the operator stack is not empty in order to process the stacks. By the time the evaluation completes, the operand stack will contain the result which will be returned by the eval function.

# Development Environment

Java version: JDK 11.0.2

IDE: JetBrains IntelliJ IDEA Ultimate 2018.3.3

# How to Build/Import your Project

Importing and building project

1. Open IntelliJ IDEA
2. Click import project
3. Navigate to the “calculator” directory
4. Click OK
5. Make sure “Create project from existing sources” is selected and click NEXT
6. Proceed to click NEXT until import is finished, making sure all JUnit and JDK files are detected
7. Build project using IntelliJ IDEA IDE by navigating to Build > Build Project

Building executable .jar application

1. Open project in IntelliJ IDEA and navigate to File > Project Structure
2. Choose “Artifacts” from the side panel
3. Click the blue + button to add a JAR > From modules with dependencies…
4. Select the Module dropdown and choose “main”
5. Ensure “extract to the target JAR” is selected under “JAR files from libraries” and click OK
6. Click OK to exit Project Structure and navigate to Build > Build Artifacts… and the executable .jar application should be located in calculator\out\artifacts\main\_jar

# How to Run your Project

After building the executable .jar application of the project, navigate to its directory at calculator\out\artifacts\main\_jar and double click main.jar.

If running from the command-line, run this command while in the executable .jar application directory: “java -jar main.jar”. On Windows, ensure JDK 11 is installed and at the top of %PATH%.

# Assumption Made

* If all the evaluator tests pass, my evaluator function and all associated classes are complete.
* Must use HashMap for operators.
* Must use Stack for operands and operators when evaluating expression.
* Parentheses count as operators.
* Parentheses do not calculate multiplication.
* Expression is fully evaluated when operator stack is empty.
* Spaces cannot be in DELIMITERS because the tokenizer while loop already takes care of spaces.

# Implementation Discussion

I decided to make a separate class for each operator all in the same directory, including the parentheses operators. This puts all operators into one spot and makes it clear what operators are available. Creating a new operator is made easier by simply creating a new class that extends Operator, filling out the abstract functions, and adding a new entry to the HashMap in the Operator file. For operators that only affect one operand, such as the factorial operator, the priority will need to be 4 or higher. The Evaluator class will pick up all new operators without the need to update the code.

## Class Diagram

# Project Reflection

# Project Conclusion/Results