CSC 413 Project Documentation

Fall 2018

Moataz Affan

916231373

CSC413.03

[***https://github.com/csc413-03-fall2019/csc413-p1-Moataz-A***](https://github.com/csc413-03-fall2019/csc413-p1-Moataz-A)

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

We are creating a calculator that can evaluate expressions and outputs the results for you in the command line or as a GUI.

## Technical Overview

Given a string input, the operands go in a stack and operators go in another stack then they get popped as needed for the operation following the rules of PEMDAS.

## Summary of Work Completed

Using a HashMap to store basic operation operators and making subclasses for every one of those operators. Implemented check, eval and proccessOperator methods. Also, key mapped the GUI calculator using getSource().

# Development Environment

Intellij Community 2018.3.2

JDK 11.0.2

# How to Build/Import your Project

BUILD:

1. File
2. Project Structure
3. Add jar files
4. Module menu choose main
5. “Extract to target JAR”
6. Build Artifacts

IMPORT:

1. Open IDE
2. Import project
3. Locate directory (calculator)
4. “Create project from existing sources”
5. Next until done
6. Inside IDE find Build then Build Project

# How to Run your Project

Find and run: Calculator\out\artifacts\main\_jar

# Assumption Made

Operators: +,-,\*,/,^

Must be binary

Parenthesis ()

Only integers for operands

# Implementation Discussion

Most design and OOP methods have been used except Singleton and a few others.  
Inheritance and composition are used in operator classes in that way we have the class that does a specific operation such as addition but inherits all the other operator values from the mother class Operator. In the sense of Encapsulation we have made some fields private and methods private such as the HashMap. Polymorphism was implemented in the Operator class that has priority() and execute() such as all other children classes can use those methods and it changes functionality depending on the class.

## Class Diagram

A close up of text on a white background

Description automatically generated

# Project Reflection

This project is a good warm up since it’s been a while since I actually worked with Java. It’s on a tight schedule but considering that it’s not much to code, it’s not bad. To be honest it took me a long time to comprehend what’s going on in the Evaluator class and I didn’t know where to start. Everything snowballs after making the sub operator classes and you go faster from there.  
I’m not experienced with Java GUI and I had some issue mapping the keys to the calculator because most of the functions were new to me such as getSource() and buttons []. I used switches in the beginning but that turned ugly and messy quickly so I had to redesign it with several if statements.

# Project Conclusion/Results

All the unit tests are passed and tried some complex custom inputs and it works like a charm. The only problem is that in the GUI calculator every time I click a button I receive tons of error messages in the IDE terminal without crashing or giving wrong results. I tried to fix some of them but they are too many and when I tried to change the method of implementing the DriverUI class it crashed so I went back to the errors as long as it works.