CSC 413 Project 1 Documentation

Summer 2019

Jonathon Knack

917629133

CSC413-01

https://github.com/csc413-01-summer2019/csc413-p1-jknack0.git

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

In this project we were required to make a program that took in an input of a math equation processed that equation and output the correct answer. We were also required to connect this program to a nice GUI.

## Technical Overview

In the starter file we were given the abstract class Operator, Operand, a broken evaluator class and a UI that wasn’t connected to any actions. In the given Operator class there was an abstract method called priority that returned the priority of the operator that was given in the project pdf, execute that would perform the correct operation based on a given string (+, /, \*, -, ^), check that would check if a given string was an operator and get operator that would return the correct operator object based on a given string. In order to keep from having create new operator objects every time we needed one a static hash-map was also put in the operator class containing instances of each type of operator so that all of the operator references were pointing to the same object in order to save memory.

Each individual operator class was an abstraction of Operator that contained the priority method and execute method. The priority method just returned an int that was given in the project pdf. The execute method took in 2 operands and using the .getValue() method used the values to perform the correct operation and returned a new operand containing the result.

The operand class needed an instance variable for the value it was holding. Then there were 2 constructors one that took in a string and was converted to an int using parseInt and the other took in an int and just assigned the value to operandValue. There was a getter that just returned the value of operandValue. Then there was the check method that would check if a given string was an int. There were 2 ways I saw on going about checking if the string was an int and the first was to try parseInt and if it threw an exception then it wasn’t a number but I opted to loop through each character in the string and check if it was a number using the Character.isDigit() method I did this because I didn’t want an exception being thrown every time an operator was found in the evaluator class so this was what I chose.

The evaluator class contains 2 stacks one for operators and one for operands and uses the eval method and conditional logic to solve the given expression. I added a process method that pops the top 2 operands and the top operator and executes and returns the new operand value to the top of the operand stack. The eval method takes in a string that is the expression to be solved and creates a tokenizer using the operators as delimiters to split up the expression correctly in case of two digit numbers. The method then checks if the current token is a blank space and if it isn’t it proceeds. The first conditional is to check if it is an operand and if it is push it to the operand stack, it then checks if it is an operator and if it isn’t throws an exception. If the token is a valid operator and the operator stack is empty it will be pushed to the operator stack. If the token is an operator, the operator stack is not empty and the priority of the current operator is higher than the one on top of the operator stack then we push this operator to the stack. If the token is the opening parenthesis we push it to the operator stack and if it is a closing parenthesis we process the stack until we find the opening one then pop the opening one off the stack. If none of these conditions are met we process. Once there are no more tokens we process until there are no more operators and return the value that’s in the operand stack.

The GUI was basically completed we just had to connect a few actions to certain buttons specifically the =, C and CE buttons. When the = is pushed we create a new evaluator object the call the eval method using the GUI text box as our expression parameter. I then updated the GUI text box so it has the original expression followed by a = and then the answer. The C button clears the entire express so I just set the text field to an empty string. The CE button clears the last input so I created a substring of the text box starting at the first character and ending at the string length – 1 and assigned that to the text box.

## Summary of Work Completed

I created all the additional operator classes and implemented their functions. I Implemented all of the functions in operand. I implemented the eval function in the evaluator class. I implemented everything in actionPerformed in the EvaluatorUI and the GUI works well. All of my tests pass except for test #12 in evaluator test. It had something to do with evaluating the expression from right to left because of the stack I saw what was happening but couldn’t figure out how to fix it without all of my other tests failing.

# Development Environment

The IDE used was the latest version of IntelliJ using java version “12.0.1”.

# How to Build/Import your Project

To import the project just set the root to the calculator folder.

# How to Run your Project

To run the evaluator class just click run in IntelliJ and if you want to run the EvaluatorUI just click the class file in IntelliJ and hit run.

# Assumption Made

The assumptions made in this project are that we will always get a valid expression, there will be no negative operands when eval is run, we will never divide by zero and no negative exponents.

# Implementation Discussion

The biggest implementation choice I think we had was how to handle the parenthesis because everything was already laid out for us. I chose to make them extend operator which is kind of weird because their execute function wouldn’t do anything and their priority wasn’t given in the project pdf. I made an opening and closing parenthesis class that extended Operator and made their priority 0. I made their priority zero so that no matter what operator came after the opening parenthesis it would always get pushed to the stack I also had a conditional for the parenthesis so that eval wouldn’t try to execute a parenthesis operator.

## Class Diagram

# Project Reflection

I tried taking this class last semester and learned my lesson to start these projects early so this was honestly a breeze. There were still parts where I was beating my head against the keyboard but nothing taking a break didn’t help or sleeping on it didn’t help. I think if anything I should try to code longer in my homework sessions because always around the 4 hour mark I kind of hit a mental wall where my brain just doesn’t work anymore. I also really liked that we got to have a GUI because I was getting sick of looking at the console window.

# Project Conclusion/Results

I successfully