# Introduction

## Project Overview

This project served as an introduction to Object Oriented Design and its concepts to provide structure and scalability to the code. The goal was to implement an expression evaluator class with specific and correct operators and operands and the rules. The expressions would be read by String Tokenizer class in Java and are stored as “tokens”. Each token will be pushed into two different stacks i.e. Operator and Operand class for each token. They will then be operated upon based on simple priority rules for +, -, /, \*, ^, (,). It will take single tokens or variables and then compute them based on its priority in the whole equation and also based on simple mathematics rules. Then outputs the calculated answer. Invalid equations or expressions won’t work because they are invalid so one has to erase them and write one that can be computed. The code is connected to an interface which displays a mini calculator for you to perform actions.

## Technical Overview

Operator is an abstract class that provides abstract methods that returns the priority and executes the operation depending on the type of the Operator. The abstract methods in the Operator class are implemented by the different Operator classes that extend the Operator abstract class, as the priority and execution of the different Operators are different. The Operator class has a static Hash Map that is used in the Evaluator class to map operator string and Operator object to it. The object of type operator is initialized in the Evaluator class using an interface in the Operator class that returns the type of Operator object (addition, division...). The Evaluator class had a skeleton of the algorithm, that reads all the tokens through a while loop, and makes checks for whether the token is an operator or operand. If the token is an operand, then it is pushed to the operand Stack in the Evaluator class. We were assigned to implement the algorithm for the Operator Stack. I added checks for the token to be a right parenthesis first, and if the token is a right parenthesis, then I solve the expression inside the () until I reach the left parenthesis. If left parenthesis is encountered, exit the loop and pop it from the Operator Stack. If the token is not a right parenthesis, and the operator Stack still has operators and the operator on the top of the operator stack has higher precedence that the token recently read, and the token does not equals left parenthesis, perform execution on top two values of the operand stack popping the top of the operator stack. When all the tokens are read and pushed into their respective stacks, I finally check if there are any operators remaining in the stack, and if so, I pop the operand and the operator stack and solve the expression. The final value of the execution is pushed into the operand stack. Finally, I pop the operand stack for the final result and return it.

## Summary of Work Completed

Fixed and added code in Operand, Operator, Evaluator java files.These are the classes used to operate on classes that computes values such as AddOperator, SubtractOperator, MultiplyOperator, DivideOperator, PowerOperator, OpenParanthesis, ClosedParanthesis. It is simultaneously checked with EvaluatorDriver for test conditions and after everything works the code is connected to GUI i.e. EvaluatorUI.

# Development Environment

IntelliJ IDEA Version 2018.3

# How to Build/Import your Project

Import the “calculator” folder into the IDE and build the Evaluator and EvaluatorUI java files.

# How to Run your Project

Run the EvaluatorUI file and a mini calculator GUI will pop up.

# Assumption Made

The project skeleton was provided, so I thought we needed to sub class Operators and not much of work would go on the Evaluator class. But as I worked on, my algorithm for evaluating expressions only solved non-braces expressions. I also had concerns about handling Arithmetic Exception and InvalidTokenException at the very end of connecting GUI, so I put a try catch block and the calculator will not perform invalid expressions.

# Implementation Discussion

The Evaluator class uses abstract class Operator, and it’s subclasses, which is the type of operator for each class, for defining it’s algorithm. The evaluator class has two different operator and operand stacks that store the type of the operator objects. Java’s String Tokenizer library is utilized to get tokenize the expression, and each token is checked if it’s an operator or operand and pushed into its respective stack. The interfaces in the operator and operand class check whether the token is an operand or operator. Upon recognizing an operand, the operand class gets the token gets pushed in the Operand Stack. If the token is an operator then static checkOperatorType() is called in the Operator class to initialize the Operator object depending on its type because Operator cannot be instantiated as it is an abstract class.

## Class Diagram

A screenshot of a cell phone

Description automatically generated

# Project Reflection

With this project I was able to brush up on my JAVA skills and Data Structures knowledge to implement the expression evaluator with certain methods and classes and how the whole navigation works. Debugging and finding errors was quite tedious and frustrating. It was turns out to be a silly mistake or error which might have been missed initially. GUI was also fun to play with and use the different buttons.

# Project Conclusion/Results

I was successfully able to pass all the test cases provided , and also I test my work with some

expressions that I created, and it successfully passed those as well. The most challenging part

for me was to figure out how to solve the embedded parenthesis as initially my algorithm only

evaluated SimpleParanthesis, I used the debugger to optimize my algorithm to pass some tests and to find errors throughout the program. Also connecting GUI was a bit tricky but eventually worked out well.