CSC 413 Project Documentation

Fall 2018

Ibraheem Chaudry

917227459

CSC 413.01

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

Table of Contents

1 Introduction

1.1 Project Overview

This project involved object-oriented programming to build a calculator application that evaluates an infix expression e.g. “1+2(3*6)”. The application once fully implemented can handle mathematical operators “+ , - , * , /, ^ , (,)” while also taking into account the rule of BODMAS while evaluating expressions. The application has a GUI (graphical user interface) which can be used to input expressions. The application can also be run using the evaluator driver which takes command line arguments for the expression and then evaluates it.

1.2 Technical Overview

The calculator application is written in java and follows a complete object objected structure. The program is broken down into three main classes that handle the operational backend:

1. Operand class
    - Handles the construction and characteristics of operand objects
2. Operator class
    - This is an abstract master class that controls all the operator subclasses.
    - Handles the construction of operator subclasses.
3. Evaluator class
    - Implements the main algorithm that creates new operand and operator object using tokens from the parsed infix string expression and evaluates it

The application can be compiled and run using either:

1. Evaluator Driver
    - This class has a main method that creates an instance of the evaluator and allows user to input infix expression to be evaluated using the command line.
2. GUI
    - The graphical user interface can be run using the UI class.
    - It opens a graphic calculator which the user can interact with using their mouse to input infix expressions and get the result.

1.3 Summary of Work Completed

The operator, operator subclasses and operand classes are fully functional and pass the tests designed to check them.

The evaluator class is also unbroken and is successfully evaluating all the 12 test expressions designed to check its functionality.

The UI class has been fixed and the GUI correctly handles the button pressed and performs the correct operation.

In summary, all requirements for the calculator application were completed and checked thoroughly.

2 Development Environment

a) Version of Java used: JDK 12
b) IDE used: IntelliJ IDEA community version by JetBrains

3 How to Build/Import your Project

1. Create a directory/New Folder in a desired location on your computer using a name of your choice i.e. “Calculator”.
2. Copy and paste the github repository url that is listed on the front page of this document.
3. This step can vary slightly depending on whether you are using a windows computer or linux/Mac
    a. If you are **using windows:**
        i. First make sure that you have **Git Bash** installed.
            1. If you don’t have Git Bash install it from this link https://git-scm.com/downloads
            2. Proceed to next step
        ii. Open a **Git Bash Terminal.** You can do this by pressing the widows key on your keyboard and typing Git Bash and clicking on the application. **(if you have an older version of windows you can go to the location you installed it and run it from there.)**
        iii. From your terminal go to the directory you created in step 1. You can do this by typing this command “cd /path/to/directory” for example “cd /users/documents/calculator”
        iv. Once inside the directory type the command “git clone **paste url copied in step 2”** and then hit enter. This will clone the repository to your folder.
    b. If you are **using Linux/Mac**”
        i. Open a terminal on your computer
        ii. From your terminal go to the directory you created in step 1. You can do this by typing this command “cd /path/to/directory” for example “cd /users/documents/calculator”
        iii. Once inside the directory type the command “git clone **paste url copied in step 2”** and then hit enter. This will clone the repository to your folder.
            1. If the terminal gives a prompt saying git is not installed you can install git by using the command your terminal provides with the prompt and then try above step again to clone.
4. After you have cloned the repository you need to open IntelliJ IDEA IDE. If you don’t have this software you can download the free community version from here and run it: https://www.jetbrains.com/idea/download/#section=windows

5. Click import Project

6. Locate the directory where you cloned the repository. Once you are inside the main repository click on the folder called 'Calculator' and then select import.
7. You will be taken through a setup process
    a. Click next to "Create project from existing resources"
    b. Click next after choosing a name and location for project
    c. When asked to choose JDK for project make sure to select the latest one. If you don't have JDK set up do that first and start the import process again.
    d. Hit next on all the prompts that appear next and you should end up with your project successfully opening in IntelliJ.

4 How to Run your Project

There are two ways to run the project:

1. Under /main/java/edu.csc413.calculator/evaluator there is a class called EvaluatorDriver.
    a. Right click on the EvaluatorDriver and click on the button that says "Run EvaluatorDriver.main()"
    b. The command line for IntelliJ should pop up with the prompt saying "Enter an expression:"
    c. You can enter expression you want to calculate there using keyboard and to evaluate it just press enter
2. Under the evaluator package is another class named EvaluatorUI
    a. Right click on the EvaluatorUI and click on the button that says "Run EvaluatorUI.main()"
    b. A calculator should open on your screen with operand and operator buttons you can click using your mouse
    c. Using your mouse input the expression you want to calculate and then press the '=' button to evaluate.

5 Assumption Made

- Only positive numbers are used as operands.
- Expression entered must be in infix format.

6 Implementation Discussion

1. Operand class
    - This class is used to create instances of all operand objects.
    - This class consists of **constructors** that create an integer operand object using a string, int or double value as an argument.
    - There is a **get function** that returns the operand as integer datatype.
    - The class also has a method/function to **check if a given token is an operand** or not. This function uses regular expression to check if a string is a valid number.
2. Operator class
    - This class is an **abstract master class** used to handle subclasses of each operator that our calculator consists of.
    - The operator class contains an instance of a **static hashmap** that contains a string as key and a new operator object as a value. For example an add operator entry in the hashmap will look like <"+",new AddOperator() >
        i. This hashmap allows our main evaluator function to easily create operator objects using string tokens.
    - The operator class **contains an abstract function** called **priority** that returns an int.
        i. The abstract nature of this function means that it must be implemented and overridden by all subclasses.
        ii. The returns of this function are **used to assign priority** to operators based on their precedence. For example an addOperator would have a priority of int 1, whereas a multiplicationOperator will have a higher priority of int 2.
    - Another important function of the operator class is the **execute function**
        i. This function takes **two operands as the function arguments** carries out the specific operation and returns the result.
        ii. This function returns null in the main operator class but is **implemented and overridden in all operator subclasses** (except for open and close parenthesis) to carry out necessary operations. For example in the addOperator class the execute function will return the result of two operands added together.
    - There is also a **check function** that takes a string argument and checks if a token is a valid operator by looking for it in the hashmap. The last function is a **getter** which returns the operator subclass from the hashmap depending on what string token is passed in as the argument.
    - The operator sub classes created are AddOperator, SubOperator, … etc and they all **extend the main Operator class**. All these sub operator classes contain two function except for open and close parenthesis classes which contain only the priority function.
        i. The **priority function** returns an integer marking the priority of the operator. The priorities of all operators are as such:

Operator	Priority
AddOperator	1
SubtractOperator	1
MultiplyOperator	2
DivideOperator	2
PowerOperator	3
OpenParathesis	4
CloseParathesis	5

ii. The subclasses that have the execution function **perform an operation** on the two operands that are passed into it depending on what operator's class it is i.e. addOperator class would add, subtractOperator class would subtract and so on.

3. Evaluator Class
    - The evaluator class is where the **main algorithm** is written which uses the operand and operator classes to carry out the operation on an infix expression.
    - The class consists of **two stack** (Stack data structures follow the rule of first in last out) data structures:
        i. One holds operands
        ii. One holds operators
    - A **StringTokenizer** is used to parse through the string using our operators as delimiters.
    - The evaluation portion happens in a **function called eval** all with the help of a **while loop** that parses through the string expression.
        i. The while loop keeps parsing through the string expression one token at a time until there are no more tokens left to scan.
        ii. Inside the while loop a token is checked to see if it is an operand or operator.
            1. If the token is an operand it is pushed to the operand stack and the loop continues to the next token in the string.
            2. If the token is an operator there are three scenarios which can take place:
                a. If operator is a close parenthesis ')' the function enters a while loop that pops two operands from the operand stack and one operator from the operator stack and executes the operator operation for those two operands. This loop continues to run until an open parenthesis is encountered in the operator stack. When the loop reaches open parenthesis, it pops it from the stack and then breaks to the beginning of our initial while loop that is parsing through the whole expression.
                b. For any operator other than parenthesis the operator may be compared to the operator that exists at the top of the operator stack. If the operator in the stack has a

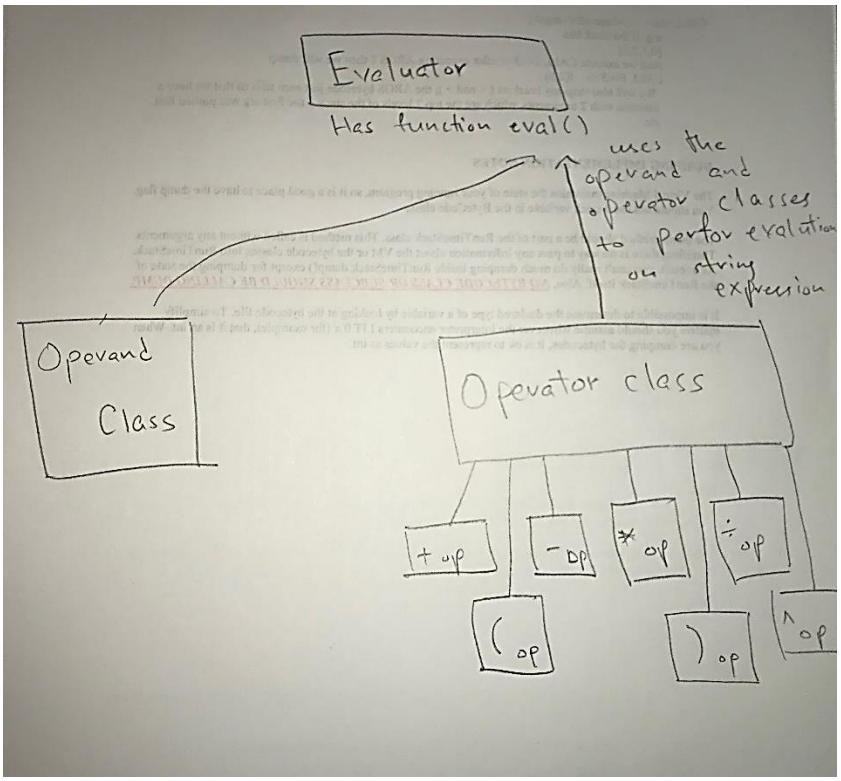
higher priority, then two operands are popped, and one operator is popped from their respective stacks. The operators execute function is used on the two operands and the result is pushed back into the operand stack. This loop keeps on going until the operator on the top of the operator stack has a lower priority than the one that is the current token. After the loop the new operator is pushed to the operator stack.

c. None of the above conditions hold and the operator simply gets pushed to the operator stack without further process.

iii. After the while loop is done parsing through the whole infix expression the final lines of code in the eval function empty out the operator and operand stack to do the final calculations and the result exists as a single operand in the operand stack.

iv. The eval function returns the resultant operand inside the operand stack as the final answer.

6.1 Class Diagram



7 Project Reflection

Working on the project was helpful in brushing up object-oriented programming skills. It was a great learning opportunity as it was my first time experiencing how abstract classes and hash maps can be used in conjunction to construct a beautifully efficient application. Also, the algorithm implemented in the evaluator class and the use of the StringTokenizer taught me a lot and made me feel more comfortable in dealing with string inputs.

8 Project Conclusion/Results

This project was completed successfully, and all tests were passed. **No bugs detected.**