Project 1 Project Page
CISC 191-86930

Alexandra Steiner

March 11, 2022

Contents

1 Project Pitch

For this project, I plan to create a command line utility that evaluates equations provided as text, and modifies then outputs the provided text to reflect the evaluation. This command line utility will be called from the command line as follows.

```
evaluate [options...] [-w -write [filepath]]
       [-f -file [filepath]] | [-t -text [textinput]]
```

1.1 Options:

- **-w -write** [<**filepath**>] Writes the evaluated output to a file at the specified path. If a file at the path already exists, overwrites the file. If no file exists, tries to create a file. If this option is selected, the results of the evaluation will not be written to the output stream
- **-f -file** [<**filepath**>] Attempts to read text from the provided file path and evaluate it as an equation [1]
- **-t -text** [<**textinput**>] Attempts to evaluate the provided text input as an equation [2]
- **-h -help** Displays the help message
- **-v -version** Displays version and program information
- **-q -quiet** Does not write the results of the evaluation to the output stream. Results are written to the output stream by default

1.2 Scope of Evaluation

Since coding in every operator and mathematical quirk would be beyond the scope of this project, only the below specified operators are valid. Additionally, dealing with variables is beyond the scope of this project, so all values separated by operators must be numeric.

1.2.1 Valid Operators

- **x+y** Adds x and y (Add, Binary)
- **x-y** Subtracts y from x (Subtract, Binary)
- **x*y** Multiplies x by y (Multiply, Binary)
- **x/y** Divides x by y (Divide, Binary)

[1] If executed with the **-t -text** option, will evaluate provided text as seperate equation(s)
[2] If executed with the **-f -file** option, will evaluate provided text as seperate equation(s)

- `-x` Negates x (Prefix Operator Only) (Negate, Unary)
- `x=y` Checks x and y for equality (Equate, Binary)

1.3 Equation Processing

Equations are always provided to the evaluate utility as a text. Before processing, the provided text is checked for illegal characters and illegal syntax. Provided text may contain multiple equation separated by new lines.

Equation may be provided to the evaluate utility as a file path, a command argument, or through Java's InputStream as detailed in the command line option specifications. Equations may be provided in the following formats. Assume "expression" is a valid expression and "evaluation" is the evaluation of expression.

- "expression" is processed into "expression = evaluation"
- "expression = " is processed into "expression = evaluation"
- "expression = expression" is processed into "expression = expression is true/false/unsolvable"

An equation is considered valid when:

1. The text of the equation only contains valid characters. Valid characters are all numeric characters, the period, and all valid operators (+-*/=). Expressions may not contain new lines
2. The equation is provided in a valid format (detailed above)
3. The equation's syntax is valid

1.4 Program Examples

myEquation.txt

```
(1+2+3)/4
10*15-4 = 15
```

Commands

```
evaluate -v
>> evaluate version 2.45
>> Created by author for CISC 191

evaluate -t "1+2+4/2"
>> 1+2+4/2 = 5

evaluate -t "18 = 36/2"
```

```
>> 18 = 36/2 is true

evaluate -t "18 = "
>> 18 = 18

evaluate << myEquation.txt
>> (1+2+3)/4 = 1.5
>> 10*15-4 = 15 is false

evaluate -f myEquation.txt
>> (1+2+3)/4 = 1.5
>> 10*15-4 = 15 is false

evaluate -f myEquation.txt -w myEquation.txt
```

myEquation.txt after commands are executed

```
(1+2+3)/4 = 1.5
10*15-4 = 15 is false
```

2 UML Diagram

See figure 1 for this project's uml diagram.

3 Learning Outcomes

3.1 LO1: Employ design principles of object-oriented programming

This program will use the divide and conquer principle of OOP to divide equations into smaller pieces that will interact with each other. While equations can be solved without OOP, dividing equations into many smaller pieces allows for more flexibility and the easy addition of additional operators down the road

3.2 LO2: Construct programs utilizing single and multi-dimensional arrays

This project will use arrays to store equations and equation components. Multi-dimensional arrays will be used to preform a syntax check on equations before processing to improve efficiency.

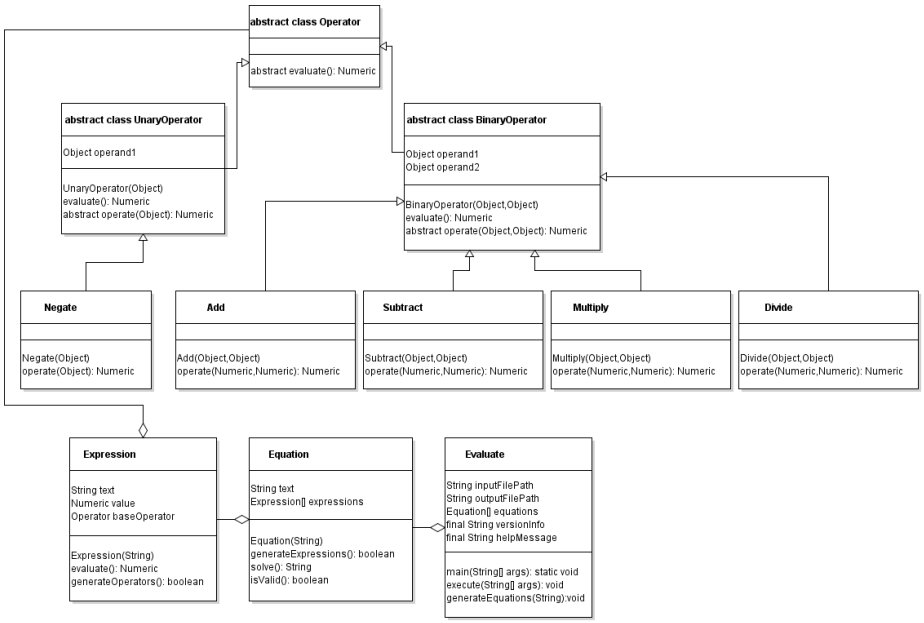

Figure 1: The UML Diagram for the evaluate utility

3.3 LO3: Construct programs utilizing object and classes in object-oriented programming, including aggregation

This program will store equations as classes, aggregating a number of different numeric and operator classes that represent the equation. The equation class will also contain a String, which is another example of class aggregation.

3.4 LO4: Construct programs utilizing inheritance and polymorphism, including abstract classes and interfaces

This program uses inheritance to eliminate redundant code when create operators. Additionally, operator operands use polymorphism to store either Numeric values or Operator values. This gives operators more flexibility. Moreover, numeric values benefit from polymorphism since some values are better stored as integer values and other values are better stored as decimal values.

3.5 LO5: Construct programs utilizing exception handling

Since this program will attempt to read and write to file paths, utilizing exception handling is a must to ensure that users trying to read equations from thisfiledoesnotexist.docx will not get an ugly face-full of Java error.

3.6 LO6: Construct programs utilizing text file I/O

File IO will be used to handle file input / output as specified in the project pitch

4 Work Schedule

I will be working on this project at approximately 1-4 pm on Saturdays and likely sporadically throughout the week whenever I can make time.

5 Timeline

5.1 Week 1

- Write the project proposal
- Plan the code. Determine classes (with fields and methods) and interfaces and their responsibilities
- Create project page

5.2 Week 2

- Create framework for all classes (define methods, fields, and classes as specified in figure 1).
- Flesh out Evaluate class so project has a working user interface going forwards
- Develop test cases and create JUnit tests
- Determine where exception handling is needed to ensure the program fails gracefully
- Update project page with progress details
- Submit code written so far

5.3 Week 3

- Finish writing classes
- Test and debug code
- Work out any edge cases and add to JUnit tests
- Update project page with progress details
- Submit code written so far

5.4 Week 4

- Debug any remaining problems
- Create project demonstration supporting files and figures
- Create project demonstration video, including information about how each LO is used as part of the project
- Submit final code on Canvas and add videos to project page