

Computer Science Department  
San Francisco State University  
CSC 413  
Fall 2018

**Assignment 1 - Expression Evaluator and Calculator GUI**

Note that the due date applies to the *last commit timestamp into the main branch of your repository*.

DO NOT change the file structure of the project. Modifying the structure could cause points to be lost.

CHANGING FILE/PACKAGE STRUCTURE WILL CAUSE A 20 POINT PENALTY TO BE APPLIED TO YOUR GRADE.

When importing the project into an IDE use the calculator folder as root of the source files.

**Overview**

The purpose of this assignment is to practice object oriented design to create two programs:

1. An object that evaluates mathematical expressions
2. A GUI around the artifact from (1)

**Submission**

You are required to submit a documentation **PDF** which is to be stored in the documentation folder in your repository.

Source code should be submitted to the GitHub repository as well in the correct folders.

**PDFS ONLY**

The documentation must include the following sections:

Your documentation **MUST** contain the following sections:

1. Title page containing
  - a. Name
  - b. Student ID
  - c. Class, Semester
  - d. A Link to the repositories.
2. Introduction
  - a. Project Overview
  - b. Technical Overview
  - c. Summary of work completed

3. Development environment.
  - a. Version of Java Used
  - b. IDE Used
4. How to build or import your game in the IDE you used.
  - a. Note saying things like hit the play button and/or click import project is not sufficient. You need to explain how to import and/or build the game.
5. How to run your project
6. Assumptions Made when designing and implementing your project
7. Implementation Discussion
  - a. I strongly recommend the use of graphical artifacts to help describe your system and its implementation: class diagrams, hierarchy, etc. Implementation decisions, code organization
8. Project reflection
9. Project Conclusion and Results.

Organization and appearance of this document is critical. Please use spelling and grammar checkers - your ability to communicate about software and technology is almost as important as your ability to write software!

## Requirements

You will be provided with an *almost complete* version of the `Evaluator` class (`Evaluator.java`). You should program the utility classes it uses - `Operand` and `Operator` - and then follow the suggestions in the code to complete the implementation of the `Evaluator` class. The `Evaluator` implements a single public method, `eval`, that takes a single `String` parameter that represents an infix mathematical expression, parses and evaluates the expression, and returns the integer result. An example expression is `2 + 3 * 4`, which would be evaluated to 14.

The expressions are composed of integer operands and operators drawn from the set `+`, `-`, `*`, `/`, `^`, `(`, and `)`. These operators have the following.

Operator	Priority
<code>+</code> , <code>-</code>	1
<code>*</code> , <code>/</code>	2
<code>^</code>	3

The algorithm that is partially implemented in `eval` processes the tokens in the expression string using two `Stacks`; one for operators and one for operands (algorithm reproduced here from [http://csis.pace.edu/~murthy/ProgrammingProblems/16\\_Evaluation\\_of\\_infix\\_expressions](http://csis.pace.edu/~murthy/ProgrammingProblems/16_Evaluation_of_infix_expressions)):

- \* If an operand token is scanned, an `Operand` object is created from the token, and pushed to the operand `Stack`
- \* If an operator token is scanned, and the operator `Stack` is empty, then an `Operator` object is created from the token, and pushed to the operator `Stack`
- \* If an operator token is scanned, and the operator `Stack` is not empty, and the operator's precedence is greater than the precedence of the `Operator` at the top of the `Stack`, then an `Operator` object is created from the token, and pushed to the operator `Stack`
- \* If the token is `(`, and `Operator` object is created from the token, and pushed to the operator `Stack`
- \* If the token is `)`, the process `Operators` until the corresponding `(` is encountered. Pop the `( Operator`.
- \* If none of the above cases apply, process an `Operator`.

Processing an `Operator` means to:

- \* Pop the operand `Stack` twice (for each operand - note the order!!)
- \* Pop the operator `Stack`
- \* Execute the `Operator` with the two `Operands`
- \* Push the result onto the operand `Stack`

When all tokens are read, process `Operators` until the operator `Stack` is empty.

***Requirement 1: Implement the above algorithm within the `Evaluator` class (this implementation need not be submitted, but it is strongly recommended that you begin with this version).***

***Requirement 2: Test this implementation with expressions that test all possible cases (you may use the included `Unit Tests` and `Driver` class to do this).***

***Requirement 3: Implement the following class hierarchy***

- \* `Operator` must be an abstract superclass.
  - \* `boolean check( String token )` - returns true if the specified token is an operator
  - \* `abstract int priority()` - returns the precedence of the operator
  - \* `abstract Operand execute( Operand operandOne, Operand operandTwo )` - performs a mathematical calculation dependent on its type
  - \* This class should contain a `HashMap` with all of the `Operators` stored as values, keyed by their token. An interface/public method should be created in `Operator` to allow the `Evaluator` (or other software components in our system) to look up `Operators` by token.
- \* Individual `Operator` classes must be subclassed from `Operator` to implement each of the operations allowed in our expressions
- \* `Operand`

- \* boolean check( String token ) - returns true if the specified token is an operand
- \* Operand( String token ) - Constructor
- \* Operand( double value ) - Constructor
- \* int getValue() - returns the integer value of this operand

***Requirement 4: Reuse your Evaluator implementation in the provided GUI Calculator (EvaluatorUI.java).***

***Additional Notes.***

***Please use the following names for the operator classes***

***operatornameOperator***

***For example:***

***AddOperator***

***DivideOperator***

***MultipleOperator***

***PowerOperator***

***SubtractOperator***