Tyler Hsieh
https://github.com/sfsu-csc-413-fall-2022-roberts/assignment-1---calculator-Tyler9648.git

## Project Introduction

The goal of this project was to create a calculator in Java that calculates expressions. We're also required to give the calculator a GUI. To put it simply, it is a basic and fully functional calculator coded purely in Java, using object oriented programming.

## Execution and IDE

I used IntelliJ to write the code for my calculator. Conceptually, the logic behind how the calculator processes operands and operators as objects made sense to me. My approach was to first code the smaller parts of the program, so I could design the superclass (Operator class) and core class (Evaluator class) around them. After I finished coding all parts of the calculator, I spent about ~ 12-15 hours debugging, almost solely due to syntax errors.

## Debugging

When I first tried to compile the EvaluatorTester, I ran into 26 errors. Nearly all of them were due to me not properly understanding or using object syntax. I grinded the errors down to just 6 after 12 hours of debugging and Googling. Here is what I learned afterwards:

- To not initialize the HashTable inside the Operator constructor
- The Operator superclass doesn't even need a constructor
- HashMap and its keys/values have to be static
- Abstract classes and methods are very similar in the sense that the body of their functions are empty
- Abstract classes/methods serve as placeholders/references and for organization
- Remember to change subclass name every time I copy and paste it
- Math.pow returns double, not int
- How to use try catch, and learning it's much simpler than using a for loop on every Operator

After not being able to fix the last 6 errors, which mainly revolved around "Operator newOperator = new Operator(token)", I asked a friend for advice on how to create a subclass through an abstract superclass after Google turned up dry. He told me even though abstract classes cannot be instantiated, static methods inside them can still be called. With my new found knowledge, I created a static method that directly returns the subclass object by accessing the hash table with a token.

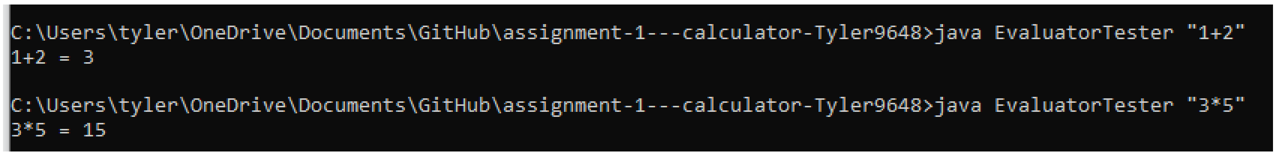
From that step forward, it was just a few errors caused by typos and careless mistakes which were easily fixed. Everything else after (just UI) went smoothly.

## Scope of work

- [x] ~~Subclasses for each operator~~
- [x] ~~Implementing Eval algorithm~~
- [x] ~~Processing operator by popping two operands and an operator, then execute all three together and push the result as operand into operandStack~~
- [x] ~~Running program using TestEvaluator~~
- [x] ~~Running program using EvaluatorUI~~
- [x] ~~Creating hashmap inside Operator superclass with token as keys, and subclasses as values~~
- [x] ~~Interface for Operator class~~
- [x] ~~After all tokens are read, process operators until stack is empty~~
- [x] ~~Reusability in UI~~

## Command Line

java EvaluatorTester "1+2"
1+2 = 3
java EvaluatorTester "3*5"
3*5 = 15



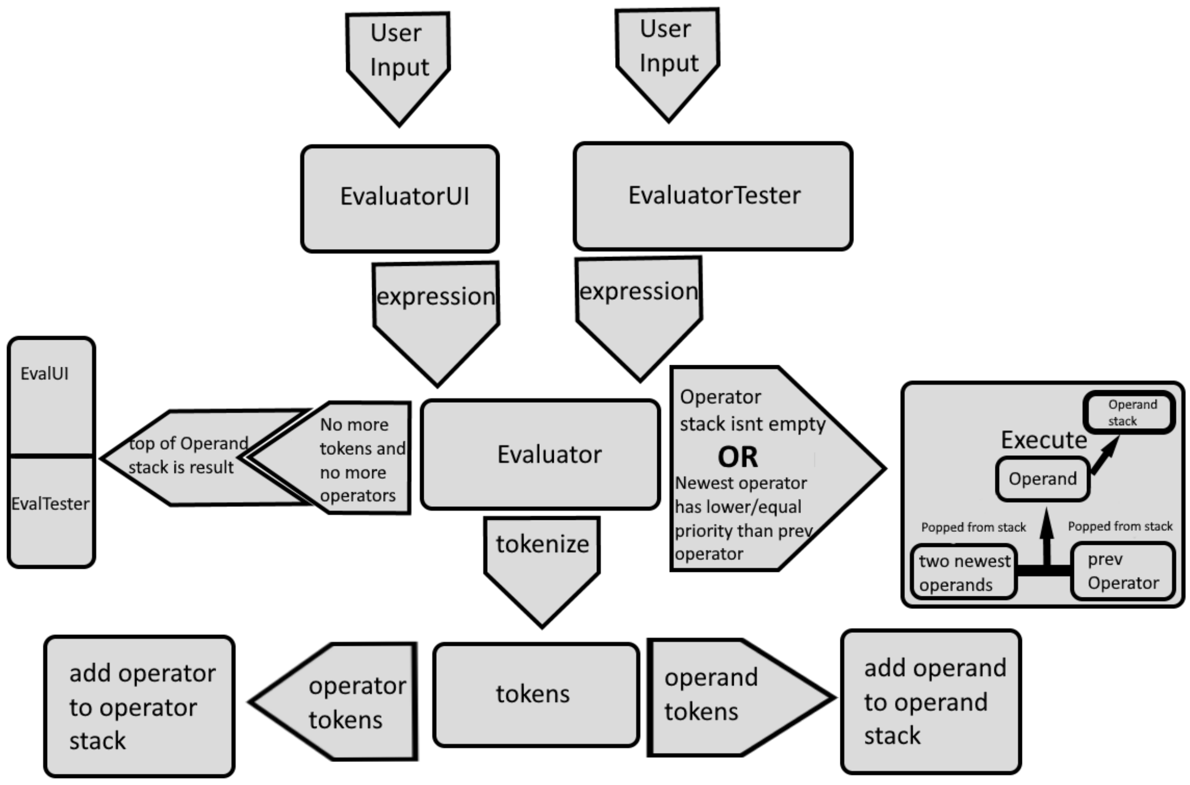
```
C:\Users\tyler\OneDrive\Documents\GitHub\assignment-1---calculator-Tyler9648>java EvaluatorTester "1+2"
1+2 = 3

C:\Users\tyler\OneDrive\Documents\GitHub\assignment-1---calculator-Tyler9648>java EvaluatorTester "3*5"
3*5 = 15
```

## Assumptions

- The interface doesn't serve a significant purpose because Operator is abstract
- EvaluatorTester doesn't need to be changed at all
- Execute method must be written in each subclass
- Operator class methods are used before an Operator object is created -> Used static methods that don't require an instance of Operator
- Result of Execute is always an int -> Math.pow typecasted to int
- Parenthesis operator works differently in that two different operators can encapsulate multiple operators and more than two operands -> While loop to process all operators within two parenthesis until no more operators between them

## Class Diagram



## Implementation

I implemented all of the Operator types as subclasses, and an unique execute method for each class besides the parenthesis operators. Parentheses operators have a subclass for the opening bracket, and the closing bracket, each having a different priority.

For the EvaluatorUI, the expression String and textField are updated every time that something is inputted. When the equal sign is pressed, the evaluator processes the expression and returns the answer back into the textField and expression. The CE and C button clear the expression. The textField is always updated with the expression.

After the tokenizer runs out of tokens, Operators are processed until there are none left using a while loop to check if stack is empty.

Execute method created to make code less redundant and cleaner. Each time Execute method is called, the Operator at the top of the stack is processed.

## Code Organization

Evaluator class handles tokenizing of expression, and creating operand/operator objects from tokens. This class also handles operand and operator stacks, processes operators, and returns the final answer.

Operand and Operator class both check if a token belongs to their respective class. Individual types of operators are implemented as subclasses of Operator so they can all be called from Operator. Calculations of processing operators are called from Operator subclasses.

## Conclusion

My finished project is a working expression evaluator that can also be implemented into a calculator.

While all the things I learned from this project are listed in the debuggion section of this PDF, there are definitely a few that standout to me. I was amazed when I learned that I can call different types of objects from a single line of code by using superclasses. I also learned that static properties and methods inside an object class don't need an instance to be called.

I encountered many syntax errors that plagued my program and took me hours of Googling why this or that error is happening. I also had to ask an experienced friend about how to call subclasses from an abstract superclass.

My final conclusion from this project is that I struggled a lot, but for that reason I also learned a lot. It helped improve my understanding of java Objects and abstract/subclasses.