

Duccio Rocca
ID = 922254031
February 14th, 2022

Assignment 1 Documentation

<i>Github Repository</i>	2
<i>Project Introduction and Overview</i>	2
<i>Scope of Work</i>	2
<i>Execution and Development Environment</i>	3
<i>Compilation Result</i>	3
<i>Assumptions</i>	4
<i>Implementation</i>	4
Evaluator	4
Operator and the Strategy Software Design Pattern.....	5
Operand	5
EvaluatorUI.....	5
Code Organization	6
Class Diagram	7
<i>Results and Conclusion</i>	8
Challenges	8
Future Work	8
Summary of Technical Work.....	9

Github Repository

[Assignment one repository.](#)

or

<https://github.com/sfsu-csc-413-fall-2022-roberts/assignment-1---calculator-RINO-GAELICO>

Project Introduction and Overview

This project required me to implement a calculator that would perform integer-only expressions starting from a code skeleton provided by the instructor. The algorithm for the calculator was provided in natural language form and the instructions included specific requirements that had to be met.

As a second step, it required me to integrate the implemented calculator into a GUI, which was also based on a partially implemented class.

Scope of Work

Task	Completed
Implement the eval method of the Evaluator class	X
Test the implementation with expressions that test all possible cases. The following expressions were used:	X
1+(10-9)*2^2-4 expected = 1; result = 1	X
2*3-6+2^2*(10-9) expected = 4; result = 4	X
2^(3-1*3) expected = 1; result = 1	X
Implement methods in the abstract Operator class	X
boolean check(String token)	X
abstract int priority()	X
abstract Operand execute(Operand operandOne, Operand operandTwo)	X
static retrieveOperator : Lookup mechanism for operators to prevent instantiation of the same operator more than once	X
Implement Operator subclasses	X
Properly organize subclasses (I used a package named Operators)	X
Implement subclasses for the required operators: multiplication, division, addition, subtraction, exponentiation, open parentheses, and close parenthesis)	X
Implement Operand class	X
Properly organize package (I used a package named Operand)	X
Constructors (String and int parameters)	X
boolean check(String token)	X
int getValue()	X
toString()	X
Complete implementation of the Calculator UI in the EvaluatorUI class	X
Use the previously implemented Evaluator	X

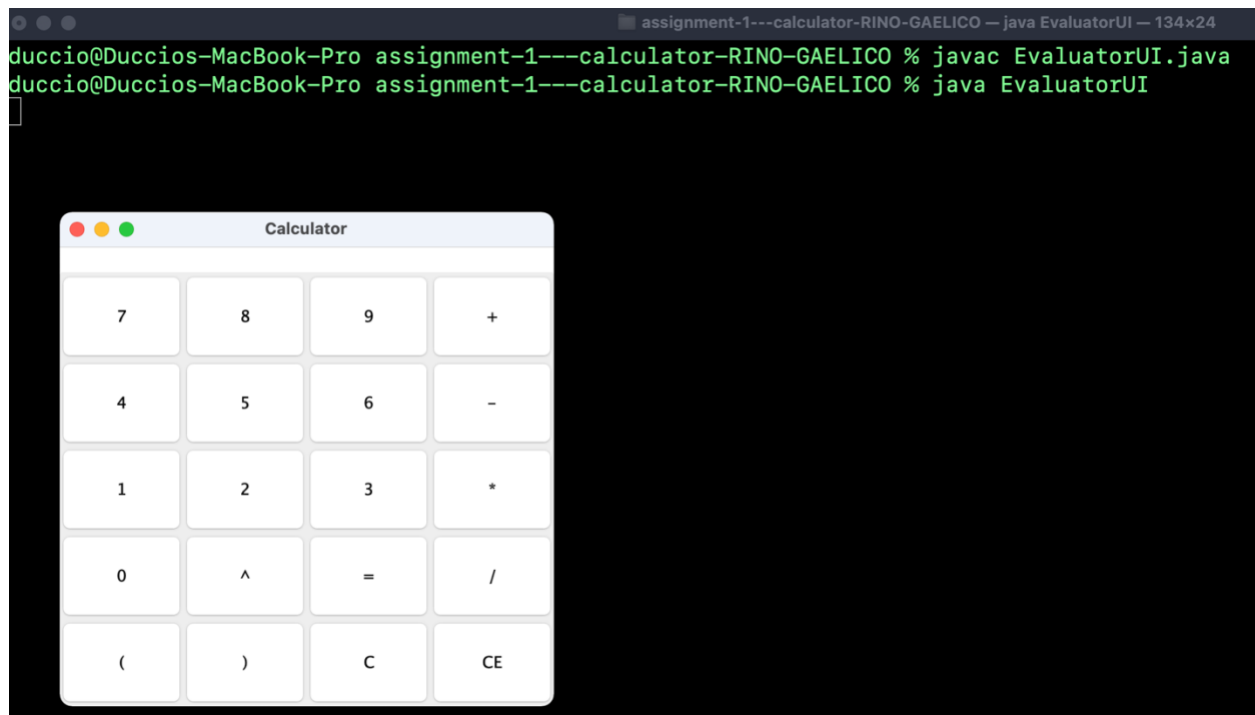
	Implement the actionPerformed method to handle button presses	X
--	---	---

Execution and Development Environment

As IDE I used IntelliJ on my MacBook Pro and I tested in both IDE and terminal through the command line.

Compilation Result

Using the instructions provided in the assignment 1 specification I compiled and ran EvaluatorUI:



The screenshot shows a terminal window with the following commands and output:

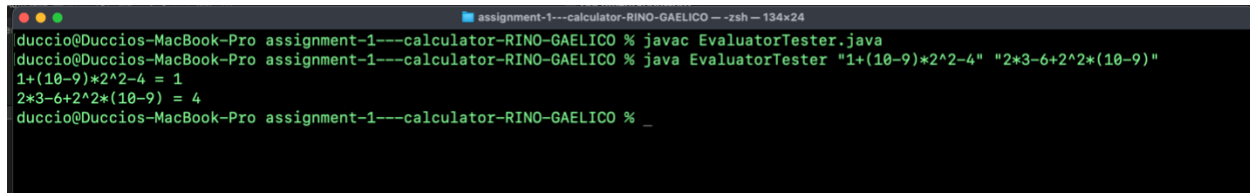
```
duccio@Duccios-MacBook-Pro assignment-1---calculator-RINO-GAELICO % javac EvaluatorUI.java
duccio@Duccios-MacBook-Pro assignment-1---calculator-RINO-GAELICO % java EvaluatorUI
```

Below the terminal window, a Java Swing window titled "Calculator" is displayed. It features a standard calculator interface with a display area at the top and a grid of buttons below. The buttons are arranged in five rows and four columns:

7	8	9	+
4	5	6	-
1	2	3	*
0	^	=	/
()	C	CE

No error messages or warnings were displayed, and the application ran as expected.

I also compiled and ran EvaluatorTester.java:

A terminal window titled "assignment-1---calculator-RINO-GAELICO -- zsh -- 134x24" is shown. The prompt is "duccio@Duccios-MacBook-Pro". The first command is "javac EvaluatorTester.java". The second command is "java EvaluatorTester \"1+(10-9)*2^2-4\" \"2*3-6+2^2*(10-9)\"". The output shows two lines of calculations: "1+(10-9)*2^2-4 = 1" and "2*3-6+2^2*(10-9) = 4". The prompt returns to "duccio@Duccios-MacBook-Pro assignment-1---calculator-RINO-GAELICO %".

```
duccio@Duccios-MacBook-Pro assignment-1---calculator-RINO-GAELICO % javac EvaluatorTester.java
duccio@Duccios-MacBook-Pro assignment-1---calculator-RINO-GAELICO % java EvaluatorTester "1+(10-9)*2^2-4" "2*3-6+2^2*(10-9)"
1+(10-9)*2^2-4 = 1
2*3-6+2^2*(10-9) = 4
duccio@Duccios-MacBook-Pro assignment-1---calculator-RINO-GAELICO % _
```

No error messages or warnings were displayed, and the application ran as expected.

Assumptions

I assumed that all tokens would be valid and all expressions evaluated would be correct. Therefore, I didn't implement any error handling for invalid tokens, except for the one already present in the skeleton code provided; also, even though I implemented an exception handling for Arithmetic errors in the GUI part, I expect it not to be used by the program since I assumed all expressions would be a valid expression (no division by zero).

Implementation

Evaluator

To implement the evaluator I followed the instructions provided in class and the pdf. The algorithm was clear and I followed the logic pretty easily.

The main challenge was represented by the parenthesis and how to deal with them since this part of the program's logic wasn't included in the algorithm provided. I added a few lines of code that included some conditionals to check if either close parenthesis or open parenthesis were encountered. If close parenthesis were found the operators' stack had to be emptied and every operation performed until an open parenthesis was found. On the other hand, if an open parenthesis was encountered, it had to be pushed onto the stack regardless of the priority level of the first operator on top of the stack.

To perform each operation I always used the same lines of code, which included popping one operator and two operands from the respective stack and using the class-specific execute-method for each operator. This repetitive code seemed appropriate for a specific helper method, which I called processingTokens.

The final part of eval required emptying the stack to obtain the final result. Since this action seemed like an independent functionality, I preferred to write a separate helper method following the Single Responsibility Principle.

Operator and the Strategy Software Design Pattern

The implementation of the operators' classes required us to fully understand the Strategy design pattern as explained in class. Following this design pattern, a programmer can define different classes that encapsulate different algorithms implemented as a class hierarchy of algorithms. Each of these algorithms represents a different behavior that can be used according to the need of the client at run time.

In this specific case, the Operator class is the abstract parent class that provides an abstract method (execute), which is implemented in different ways (different strategies) by the subclasses representing the various operators. In particular I implemented 7 different subclasses: AdditionOperator, SubtractionOperator, MultiplicationOperator, DivisionOperator, PowerOperator, OpenParenthesisOperator, and CloseParenthesisOperator.

Each of the classes can provide a different behavior according to the runtime need. In particular, a HashMap is statically created including an instance of every subclass. This allows instantiating each operator only once and retrieving them through a key, which is the corresponding String representation of each operator, at the moment of need.

I chose to include CloseParenthesis in this set of subclasses even though it does not provide any specific strategy (different behavior) because it seemed consistent with the overall implementation, which required us to create an object from each token provided.

I also included an overwritten toString method for each Operator class, because I found it useful for debugging and it seemed an appropriate implementation for this kind of class.

Operand

The Operand class seemed simpler to implement with respect to the Operator class. The methods to be implemented were similar to Operator but with some differences. I implemented the check method using a try and catch block since the static method of the Integer wrapper class throws a NumberFormatException if the String passed is not a parsable integer. The constructor method with String parameter uses a similar try and catch block but stores the value returned by the Integer's static method into the instance variable OperandValue.

The implementations of the other methods were pretty straightforward.

I also included a toString method for the same reason mentioned above for the operators' classes.

EvaluatorUI

The code provided was partially implemented, therefore it only required me to add a specific implementation of the `actionPerformed` method. I decided to use a series of conditions to determine what behavior to implement. For each condition statement, I used the method `getActionCommand` from the `ActionEvent` class.

In case of C and CE, the `displayedExpression` instance variable is manipulated to be able to display the corrected expression.

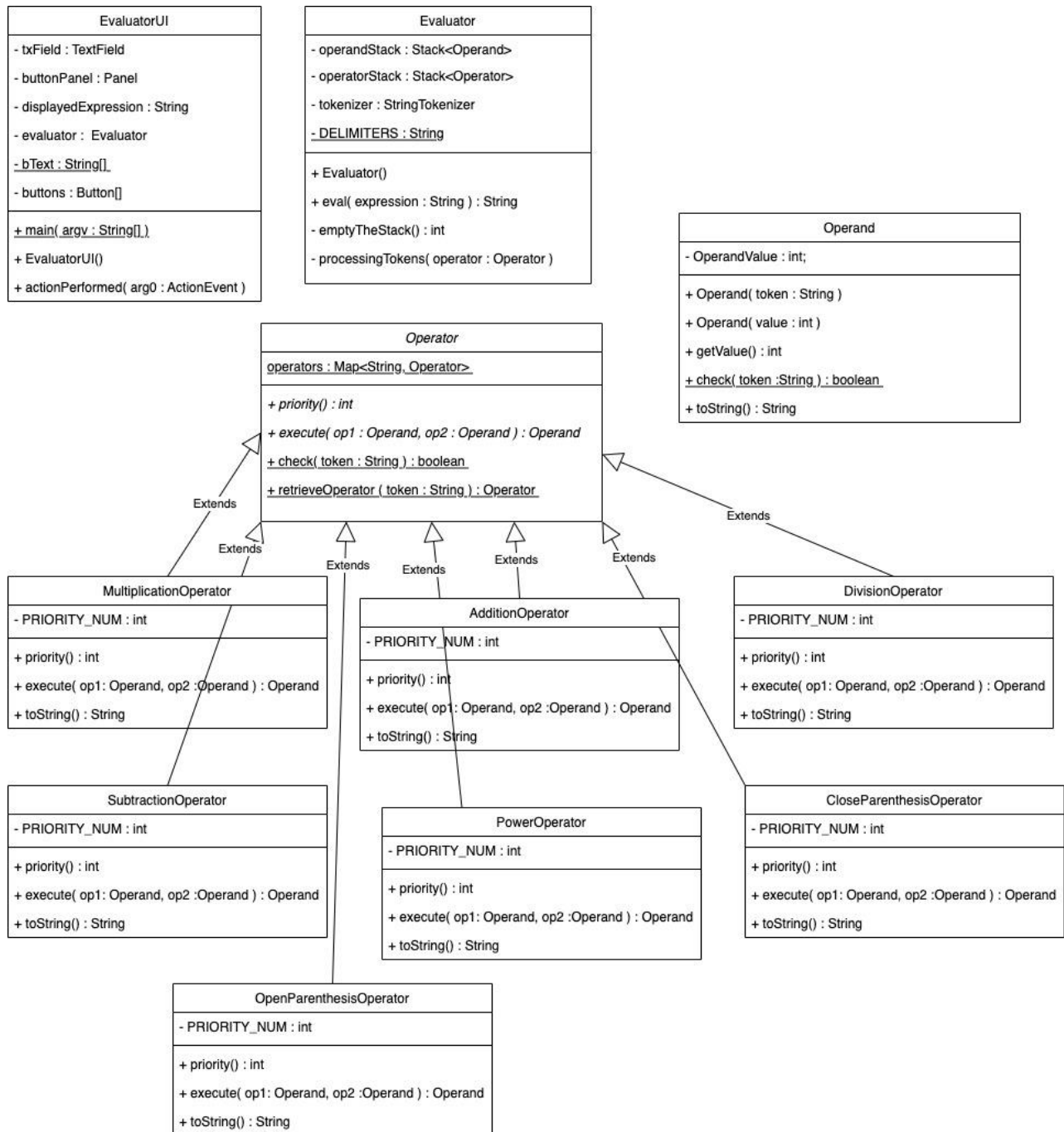
The case of “=” was the one that required to call the `eval` method of the evaluator object instantiated as an instance variable of `EvaluatorUI`. Out of extra caution, I added a try and catch block to deal with a possible divide by zero error, even though this eventuality is guaranteed to not occur because only valid expression would be tested. After performing the evaluation, the result was stored back into the `displayedExpression` variable, and placed back in the text field so that the user could continue appending the expression starting from the previous result.

Finally, the case of Operators or Operands required a concatenation of Strings to the expression already present in the text field.

Code Organization

I organized the classes into packages. Two distinct packages: one for Operators and one for Operands.

The root folder only contains the `Evaluator`, `EvaluatorUI`, and `EvaluatorTester` classes.



Class Diagram

The above class diagram shows the various classes included in the project (I did not include EvaluatorTester because I did not modify anything in that particular class).

Results and Conclusion

I worked with design patterns before and this project was a good opportunity to refresh the Strategy pattern, which is very useful and easy to apply. Also, the GUI part **allowed me** to recall some of the basic functions of Javafx.swing and Java.awt.

It took me about a week of work to finish all the required features and I am satisfied with the realized implementations.

Challenges

I made extensive use of debugging when it came to implementing the eval method in the Evaluator class. The eval method was probably the most challenging part of the assignment. I was familiar with the implementation of an evaluator through stack but I had to deal with the specific class organization provided which made it a little bit more difficult. The other challenging part was related to the parenthesis since they had to be treated differently than the other operators. I initially thought that it was sufficient to find the right priority value, but then I gave up on that approach and opted for conditionals that would check for parenthesis objects using the instanceof operator.

Future Work

I would like to transform this evaluator into something similar to a real calculator, perhaps starting to deal with invalid expressions and arithmetic errors. Then an interesting feature to add would be the possibility to input also negative numbers. To that, we could add a specific button or use the minus already present but that would require a more convoluted mechanism to recognize its position within the expression.

Summary of Technical Work

Category	Description		Notes
Code Quality (15)		✓ Code is clean, well formatted (appropriate white space and indentation)	5
		✓ Classes, methods, and variables are meaningfully named (no comments exist to explain functionality - the identifiers serve that purpose)	5
		✓ Methods are small and serve a single purpose	3
		✓ Code is well organized into a meaningful file structure	2
Documentation (25)		✓ A PDF is submitted that contains:	3
		✓ Full name/Student ID	1
		✓ A link to the github repository	1
	Overview	✓ Project introduction	2
	Overview	✓ Summary of technical work	2
	Overview	✓ Execution and development environment described	1
	Overview	✓ Scope of work described (including what work was completed)	2
		✓ Command line instructions to compile and execute	2
		✓ Assumptions made	1
	Implementation discussion	✓ Class diagram with hierarchy	3
	Implementation discussion	✓ Implementation decisions	3
	Implementation discussion	✓ Code organization	1
		✓ Results/Conclusions	1
		✓ Formatting	2
Requirement 1 (15)			15
Requirement 2 (15)			15
Requirement 3 (25)	Operator		25
Requirement 4 (5)			5