CSC 413 Project 1 Documentation

Name: Vimean Sam
ID: 915819611
Link to GitHub repository: https://github.com/csc413-03-sp18/csc413-p1-Vsam326

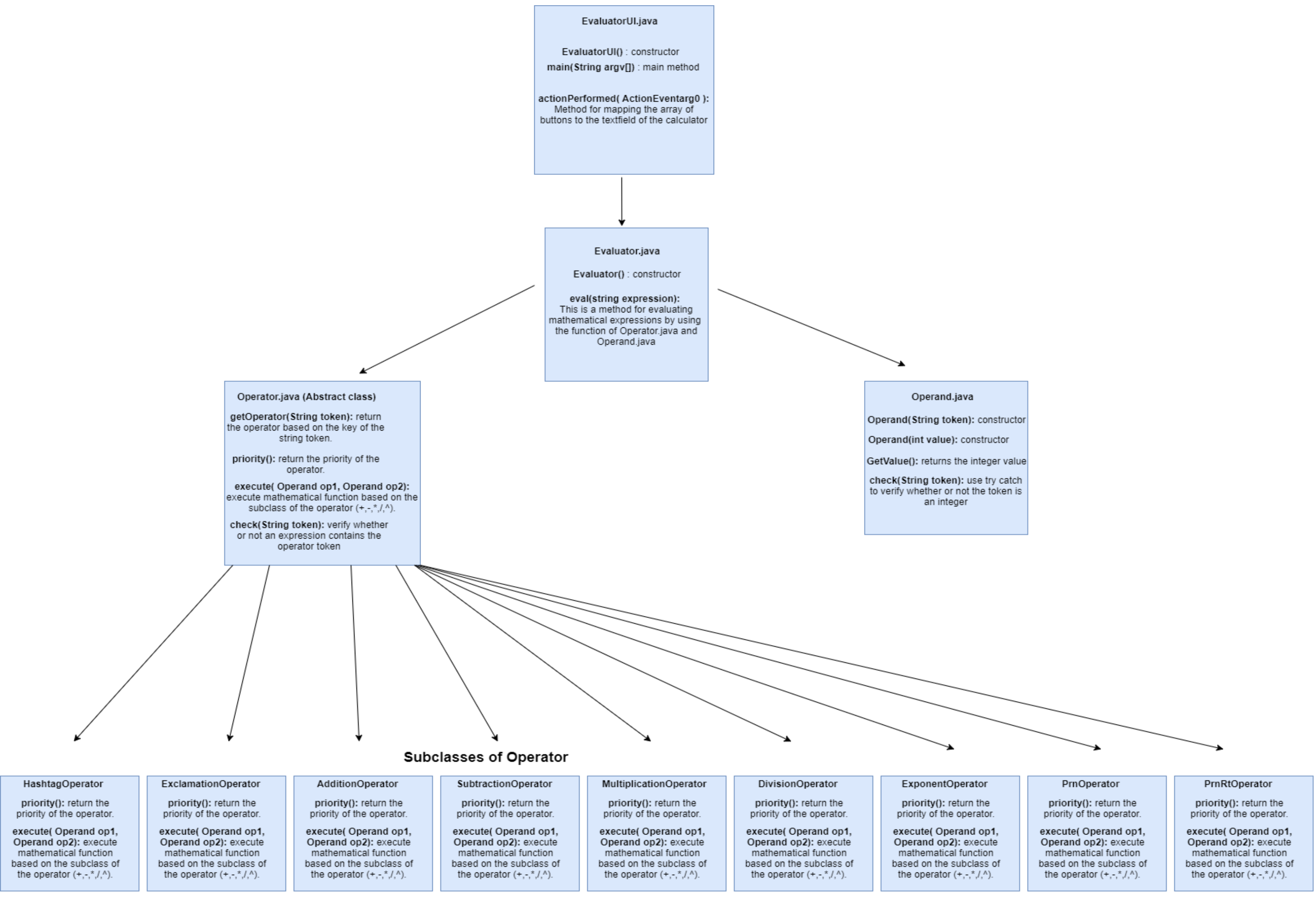
Project Introduction: In this first project of CSC 413, I was given a skeleton code of a calculator as I have to implement its function in order for the calculator to follow mathematical order of operations (PEMDAS) using Netbeans IDE 8.2 in java language.

Assumption: While implementing the code, I made assumption that my codebase will work for all valid expressions as I didn't have to worry about character inputs because the mapping of the array of buttons in EvaluatorUI.java consists of only numbers and mathematical operations.

Command line Instructions: I did not use any instructions. I just manually downloaded the files from Github and import the source codes in my Netbeans project folder and work on it.

Implementation Discussion: I completed the task of mapping the arrays of buttons to the text field in EvaluatorUI.java and implementing the addition, subtraction, multiplication, division, exponent, the parentheses and the extra credit operator of "#" and "!" in Operator.java that will be used in Evaluator.java. There are two stacks in Evaluator.java. OperandStack for the numbers and OperatorStack for the mathematical operators. I pushed the "#" into operatorStack to indicate the beginning of an expression as it prevents the "Empty Stack Exception" error and save time. Instead of scanning for operators and pushing it into operatorStack consistently, the "#" operator will prevent the operatorStack from being empty and it will alert the program that there will be expression inputs every time the calculator starts. On the other hand, I used the "!" operator to indicate the end of an expression as I added a code "expression.concat("!")" to add "!" to the end of the expression as "2+2" will be displayed as "2+2!". I found that adding "!" to the end of expression and adding "OperandStack.size() > 1" the parameter in the while (operatorStack.peek().priority() >= newOperator.priority()) loop prevents me from adding an additional method that evaluates the expression until the operatorStack contains the initial "#" operator as my calculator is fully functional when I ran the GUI and test cases in EvaluatorTester.java. Adding the parameter "OperandStack.size() > 1" would only execute the operations when there are more than one operands in the OperandStack which prevents Empty Stack Exception also. Therefore, using "#" and "!" shortened my code and workload to implement the calculator. The class diagram is given in the next page.

Class Diagram


EvaluatorUI.java
EvaluatorUI() : constructor
main(String argv[]) : main method
actionPerformed(ActionEventarg0): Method for mapping the array of buttons to the textfield of the calculator
Evaluator.java
Evaluator() : constructor
eval(string expression): This is a method for evaluating mathematical expressions by using the function of Operator.java and Operand.java
Operator.java (Abstract class)
getOperator(String token): return the operator based on the key of the string token.
priority(): return the priority of the operator.
execute(Operand op1, Operand op2): execute mathematical function based on the subclass of the operator (+,-,*,/,^).
check(String token): verify whether or not an expression contains the operator token
Operand.java
Operand(String token): constructor
Operand(int value): constructor
GetValue(): returns the integer value
check(String token): use try catch to verify whether or not the token is an integer
Subclasses of Operator
HashtagOperator
ExclamationOperator
AdditionOperator
SubtractionOperator
MultiplicationOperator
DivisionOperator
ExponentOperator
PrnOperator
PrnRtOperator
priority(): return the priority of the operator.
execute(Operand op1, Operand op2): execute mathematical function based on the subclass of the operator (+,-,*,/,^).

Results and Conclusions: This project exploits one of my weaknesses in CS: the skeleton code. Throughout the difficulty, I learned about the importance of debugging my code using System.out.println(); to see the functionality of the code clearly. In addition, I learned that building a calculator can be extremely difficult when it comes down to order of operations and the parentheses. I overcame the challenges by debugging my code and print out variables on the console (debug println is still in my code but commented out) and I'm able to fix the bugs/problems through various trials and errors. But at the end, I learned a lot about GUI programming and stacks. This is a refresh of java to my mind as I haven't used java in almost 2 years.