Assignment 2

Team number: 19
Team members

Name	Student Nr.	Email
Hung Hoang Duy	2711030	h.hoangduy@student.vu.nl
Toyesh Chakravorty	2689157	t.chakravorty@student.vu.nl
Dora Kementzey	2714287	d.kementzey@student.vu.nl
Yudai Nakazaki	2707442	y.nakazaki@student.vu.nl

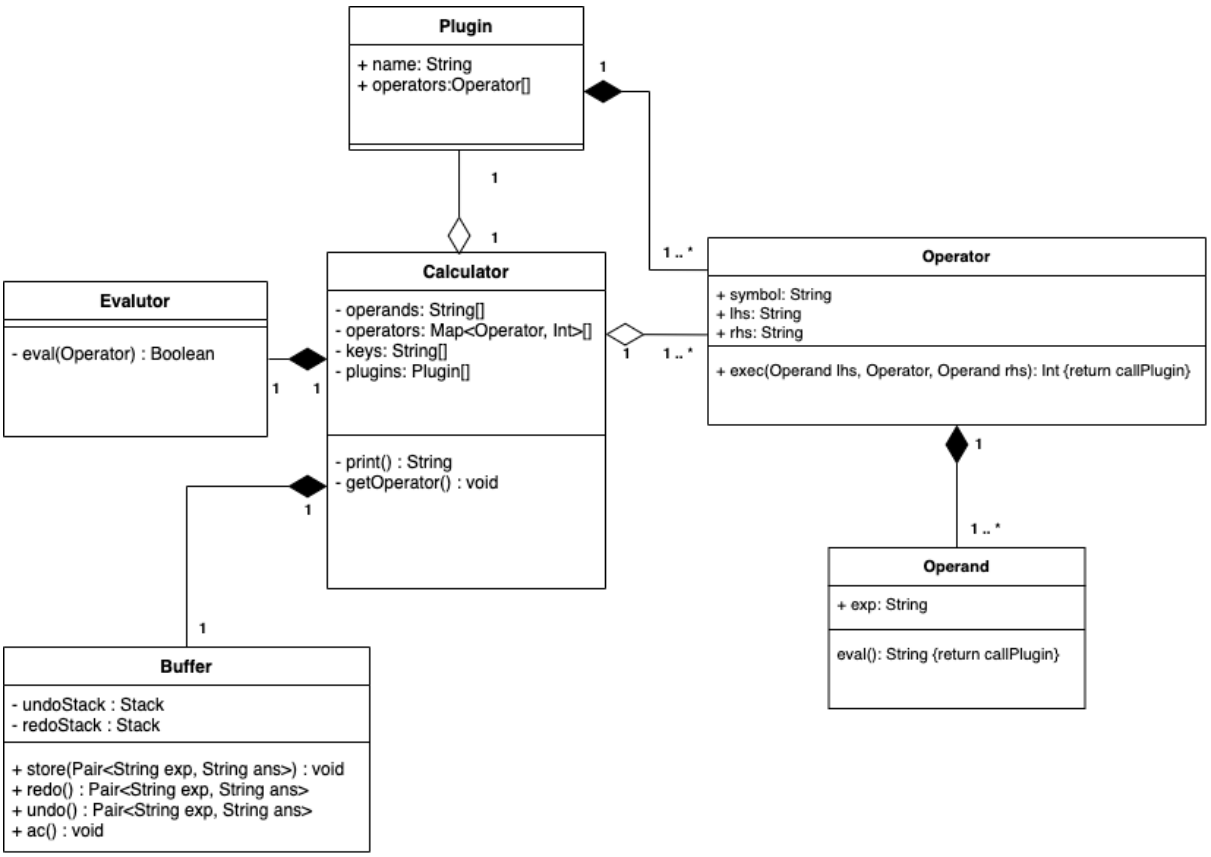
Used modelling tool: https://app.diagrams.net/

Note: All diagrams are prescriptive as we did not implement anything other than the UI, and therefore they are of the same color.

Class diagram

Author(s): Yudai & Hugo

<Diagram>



https://app.diagrams.net/#G18Q1qlswAYXYAmiZzuCVpjDBJzkJVOo1i

<General Idea>
The diagram represents the main components of the pugin-based calculator system, **Calculator**, **Buffer**, **Evaluator**, **Plugin** and **Operator**.

- **Calculator**:
  - **General Idea**
    Calculator class represents the main frame of the calculator, which originally has a list of operands(basically integers) and other keys such as ".", "(", ")" and "=", related to calculation.
  - **Attributes**
    - Operands : String[]
      - List of operands the calculator contains
    - Operators : Map<Operator, int>[]
      - Array of maps to store pairs of Operator object and integer
      - The integer is for tracking the index of the plugin in Plugins, in case we need to remove a plugin from the calculator. It also makes it easy to find a plugin where the operator belongs to.
      - The contents of this array is passed by any plugin classes.
    - Plugins : Plugins[]
      - Array of Plugin Class objects
      - List of plugins which are installed in the calculator
    - Key
      - List of other necessary keys such as ".", "(", ")" and "=".
      - Each element in this list is also shown on the UI.
  - **Operations / Methods**
    - print(string) : String
      - Method to print the result of calculation and error message on the console or the UI.
    - getOperator(Operator[]) : Void
      - Method to obtain the list of operators from the plugin object.
      - It is called when a plugin is plugged.
  - **Association**
    - It has composite relations with the Evaluator class and the Buffer class.

- **Buffer**:
  - **General Idea**
    Buffer class is a part of the Calculator class designed for storing result data and corresponding expression data.
  - **Attributes**
    - undoExpStack
      - A stack of expression data for undoing.
    - redoExpStack
      - A stack of expression data for redoing.
    - undoAnsStack

            - A stack of result data for undoing.
        - redoAnsStack
            - A stack of result data for redoing.
    - **Operations / Methods**
        - store(Pair<>(String exp, String ans)) : void
            - Get calculation result and expression and push them on undoExpStack.
        - undo() : void
            - Pop top of undoExpStack and undoAnsStack, push them on redoExpStack and redoAnsStack, and show them on the UI,
        - redo() : void
            - Pop top of redoExpStack and redoAnsStack, show them on the UI, and push them on undoExpStack and undoAnsStack.
        - ac() : void
            - Empty the content of all the stack.
    - **Association**
        - There is a composition relationship between Buffer and Calculator. Calculator is a composite object of the Buffer object and Calculator can have one single Buffer object in it.

- **<u>Plugin</u>:**
    - **General Idea**
      Each plugin has a data type of Plugin. Plugin objects provide a list of Operators, an evaluator and computer which execute calculations.
    - **Attributes**
        - name : String
            - Name of the plugin
        - Operators : Operator[]
            - Array of Operator class objects
            - This list of operators is passed to the operator list of the calculator as it is plugged.
    - **Operations / Methods**
        - No operation for this class.
    - **Association**
        - A plugin class has a composite relationship with a Calculator class object.

- **<u>Evaluator</u>:**
    - **General Idea**
      Evaluator class is a part of the Calculator class, supporting the calculation process,
    - **Attribute**
        - This class has no attribute.
    - **Operations / Methods**
        - simplify(Operand lhs, Operator, Operand rhs) : Boolean
            - Simplify the expression and deal with operand precedence (* before + etc.)
    - **Association**

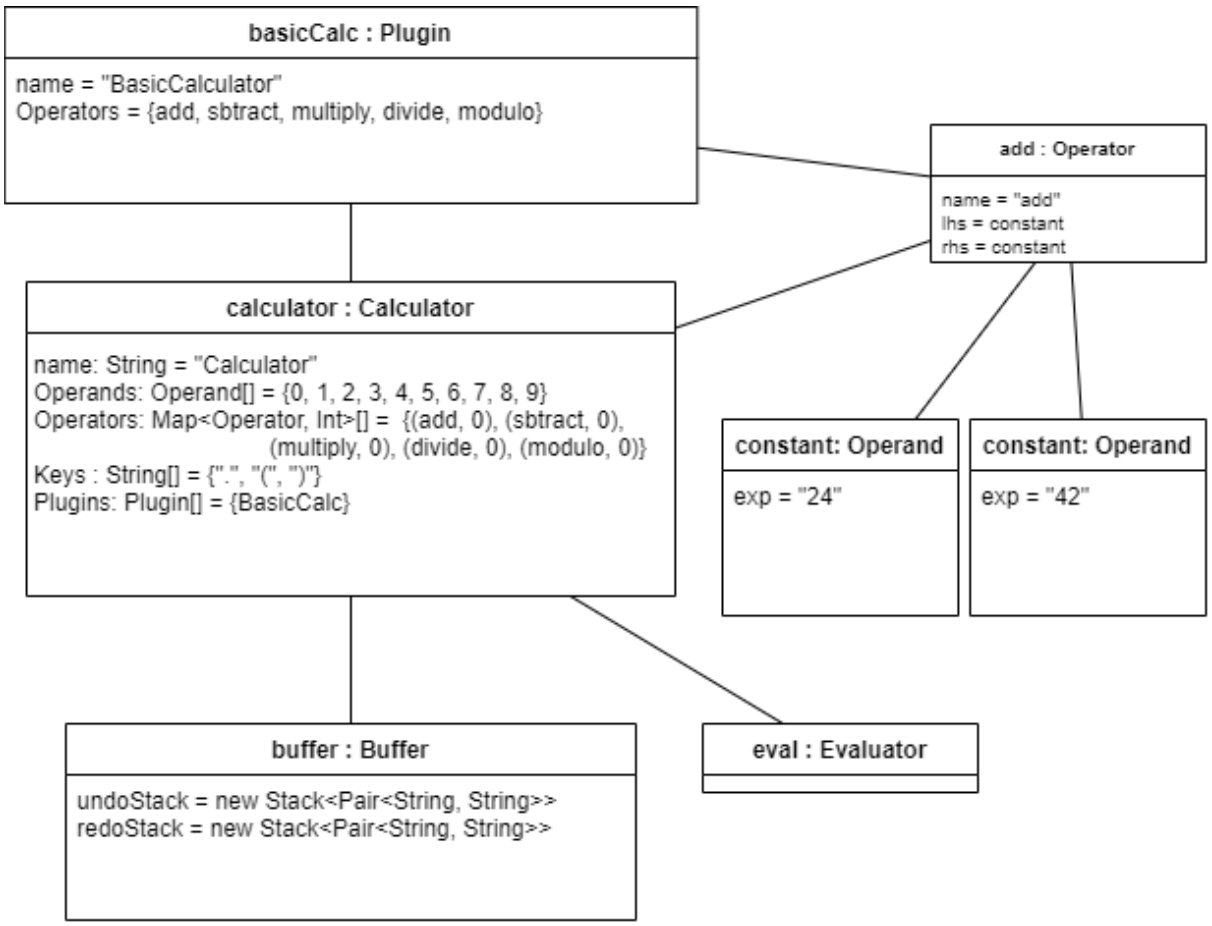
        - The evaluator class has a 1-on-1 composite relationship with the Calculator class.

- **Operator**:
    - **General Idea**
        - Operator class represents an operator which belongs to a plugin.
    - **Attribute**
        - symbol: String
            - It represents the operand as String.
        - lhs: Operand
            - left operand
        - rhs: Operand
            - right operand
    - **Operations / Methods**
        - exec(Operand lhs, Operator, Operand rhs) : String
            - Execute a coresponding calculation.
    - **Association**
        - The Operator class has a relationship with the Plugin class.
- **Operand**
    - **General Idea**
        - Operand class represents an operand (could be a number or an operation that evaluates to a number such as sin(60))
    - **Attribute**
        - exp: String
            - the full operand expression
    - **Operations/Methods**
        - eval(): String
            - Evaluates the expression to a number
    - **Association**
        - Each operand has a relation with one single operation.

Object diagram

Author(s): Yudai & Hugo

<Figure representing the UML object diagram>



<Textual description>
This object diagram describes a use case of the Calculator system with a Basic Calculator plugin.

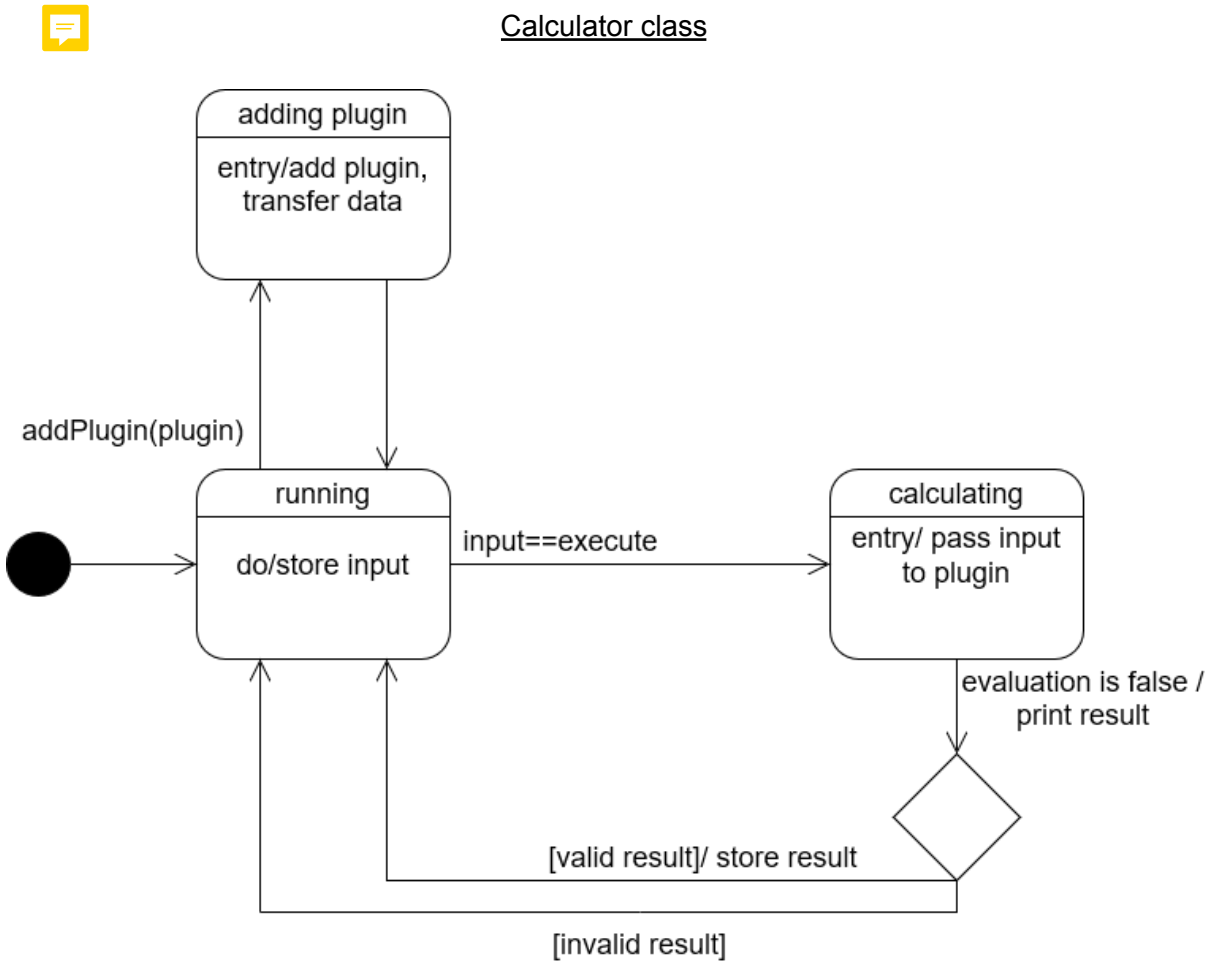
- **calculator : Calculator**
  - Calculator is an object which has a data type of Calculator class. It has a list of operands and basic keys such as “.”, “(” and “)” in a list of Keys. The Calculator object uses some Plugin objects to execute calculations. It obtains the list of operators into the domestic list from the Plugin object. In the diagram, for example, (add, 0) means that add operation is an object which is from a plugin at index 0 (= basicCalc) in Plugins.
- **buffer : Buffer**
  - Buffer is an object which is a part of the Calculator object. The Buffer object stores results and corresponding expressions.
- **basicCalc : Plugin**
  - basicCalc object contains a list of Operators which are provided to execute calculations. basicCalc provides the Calculator with necessary operators to

be used. Some operations on this object are used by the Calculator object to execute calculations.

- **add : Operator**
  - Each object has an operation to execute its own calculation. For example, add takes two operands, lhs(left hand side) and rhs(right hand side), and executes addition for these inputs.
- **constant : Operand**
  - the constant objects contain numbers that are inputted.
- **eval : Evaluator**
  - Check the validity of operators for every calculation step.

State machine diagrams

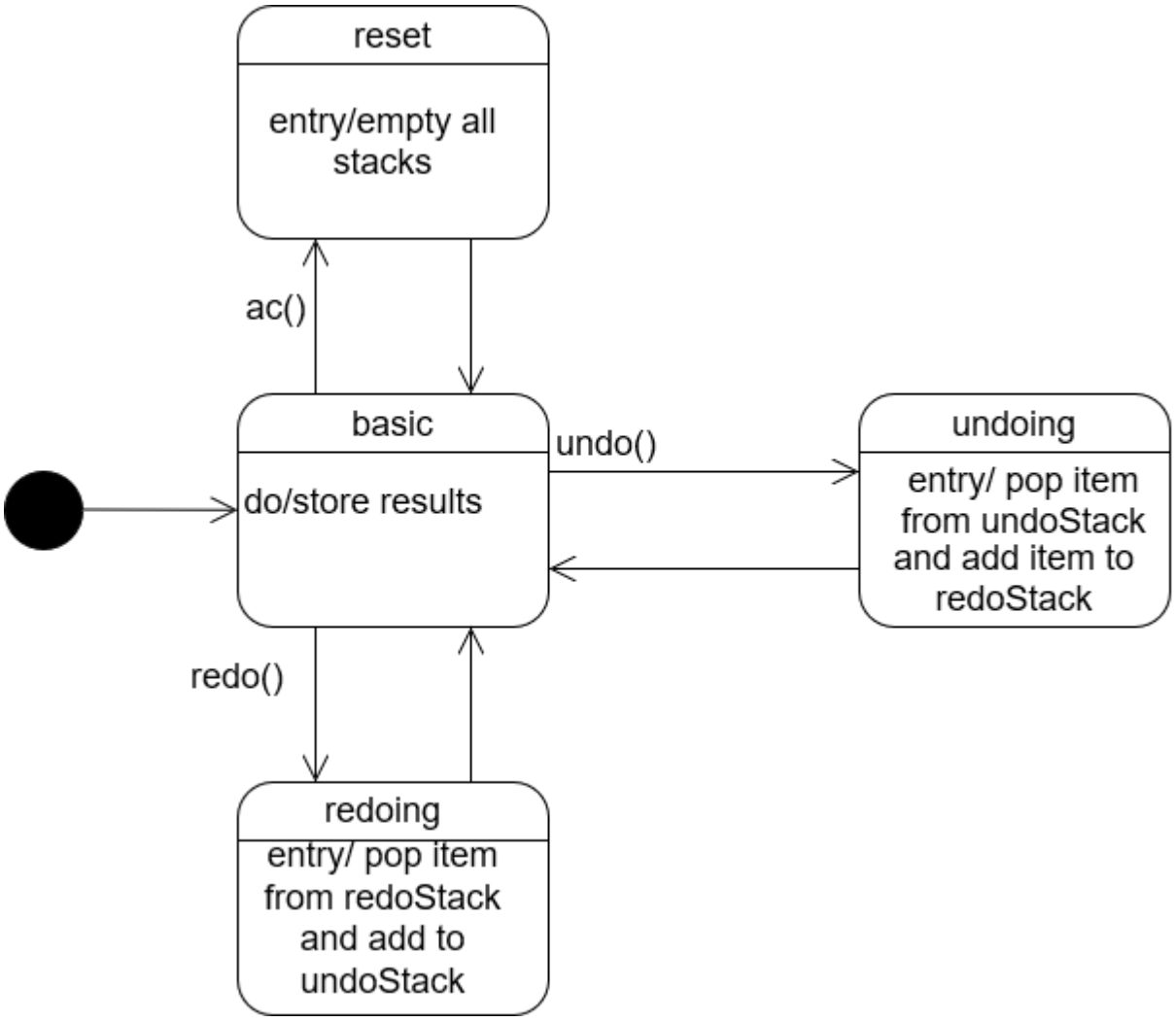
Author(s): Dora & Toyesh



This state diagram is used to describe the main behaviour of the Calculator class and the corresponding results that it needs to wait for. After the initial state, there can be two possible cases. If the user adds a new plugin, the plugin is registered, and the new operators are stored in a map with the corresponding plugin. This way plugins that have overlapping operators can be downloaded, and operators are noted with the first plugin they appear in. The other case is when the user enters a combination of operators/operands, the Calculator class sends it to the Evaluator class to take in the string as an input and consequently

evaluate it. After the evaluation, the evaluated data is sent to the Plugin class which in turn sends a corresponding result back to the Calculator class. This repeats for every subsection of an expression (for ex. 3+2*4) until the final result is received (evaluation==false) and the Calculator. If the result is not an error message this is sent to the Buffer which stores the result. A detailed description of the calculation process can be seen in the sequence diagrams, for clarity this has been omitted here (And also because it does not happen in the actual Calculator). Finally, the Calculator returns to its original state: waiting for the input.

Buffer class

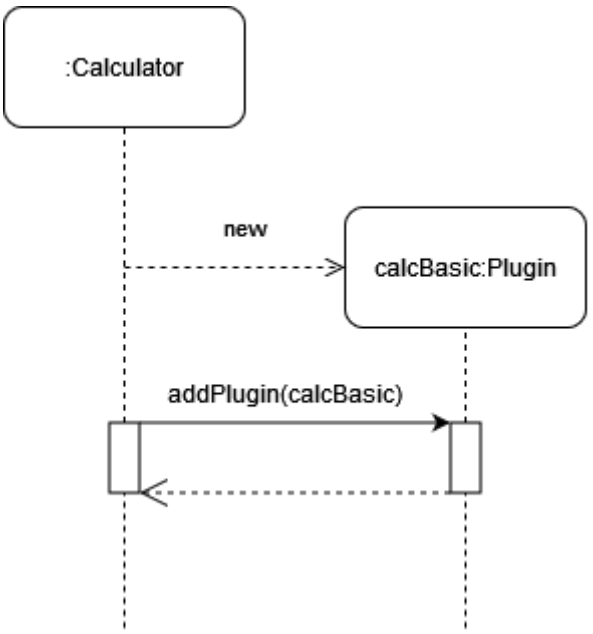


This diagram shows the states of the buffer class. When the calculator is started, the first state called ‘basic’ is activated. Here, the final results of each expression is stored in a stack called undoStack. If the undo button is clicked, the buffer will get the previous result (pop the item from the stack), and move it to the redoStack. After this the buffer returns to waiting for results. If the redo button is clicked, similarly to undo, an item is popped from the redoStack and added back to the undoStack. After this the buffer returns to waiting for results. When ac is clicked, both stacks are emptied, and the buffer returns to its basic state.

Sequence diagrams

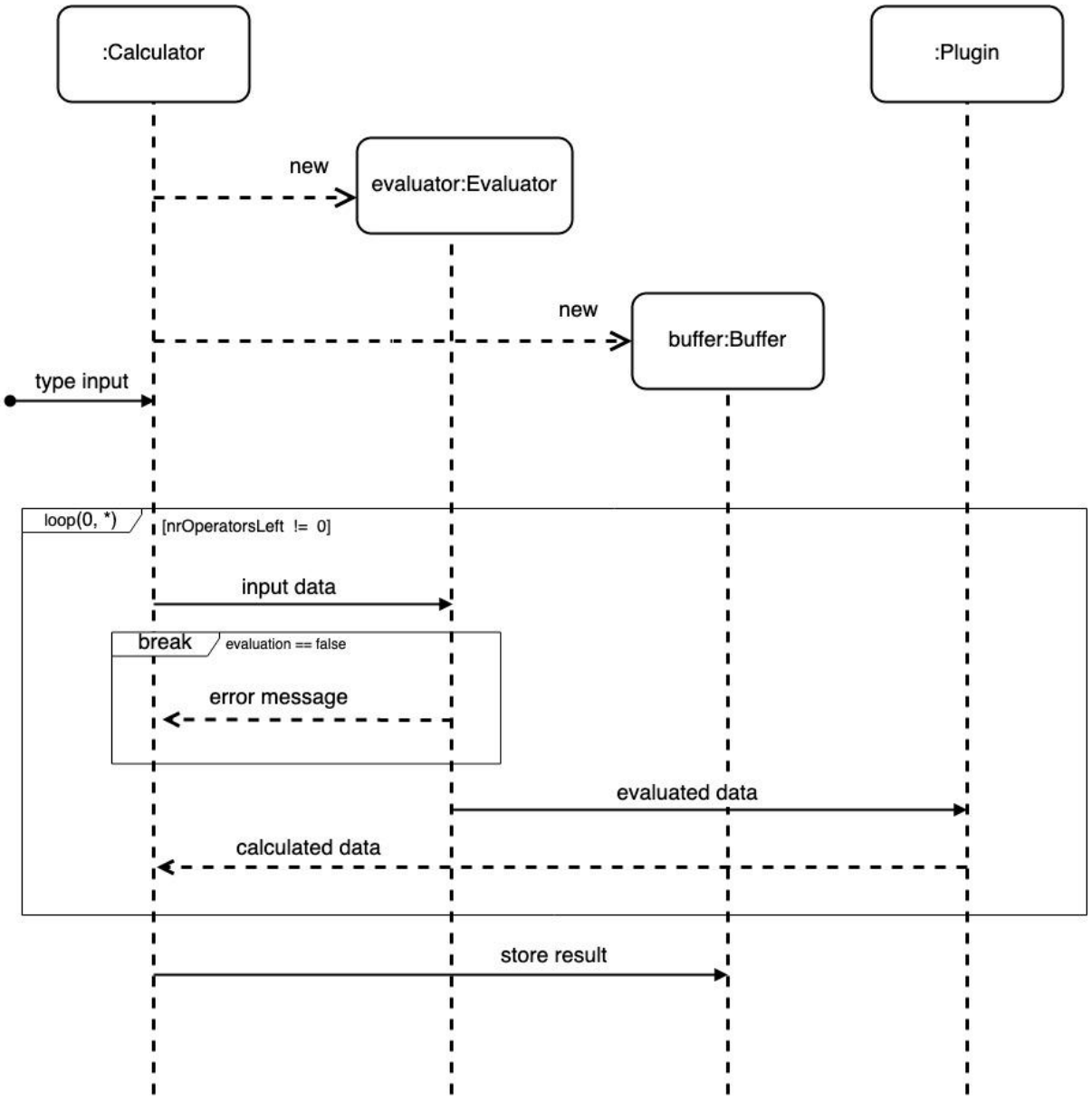
Author(s): Dora & Toyesh

Adding a new plugin



This sequence diagram is used to describe the steps that our system takes when a new plugin is added to the calculator. After the user chooses the desired plugin (from the plugin store for example), the Calculator class creates a new Plugin object to store the corresponding plugin (named *calcBasic* here). Next, this plugin is added to the Plugin class and the Calculator class, correspondingly, is notified of the same.

Calculating an expression



This sequence diagram paints a clear picture of the sequence of steps that our system undergoes when a user inputs a few operands, chooses the desired operation to be performed and then chooses to execute it, that is click on '='.

In the beginning, the class Calculator creates a new object *evaluator* of class Evaluator. Next, it creates a new object *buffer* belonging to the class Buffer. When the user clicks on the buttons (combination of operands and operators), the input is stored and displayed in the form of a string in the Calculator class. After the user finishes typing in the required expression, it is executed by pressing the '=' button.

Now, the program enters the stage of evaluating the expression. A loop is initiated, and it runs until the number of operators in the expression does not equal 0. This means all subsections of the expression are recursively calculated until the final result. The input string is sent to the *evaluator* object of the Evaluator class. If the data is assessed to be True, the evaluated data is sent to the Plugin class. Consequently, a result, that is the calculated data, is sent back to the Calculator class. On the other hand, if the input data was assessed to be False, the break condition would have been executed and an error message would have been sent back to the Calculator class.

Once the loop ends, if there was a successfully computed result, it will be sent from the Calculator class to the Buffer class to be stored for future use (namely undo/redo).

Implementation

Author(s): Hugo & Yudai

<Strategy>
As a strategy to move from UML diagrams to the implementation, we started from the most fundamental and main part of the system. We specifically started by implementing a mockup of the User Interface, because the mainframe of the calculator is the main part of this system and the base of additional features such as plugins (and because we were told to do so).

<Key solutions>
We used the JavaFX framework to implement the GUI part. For this implementation, we mock up the original frame of the calculator. Therefore, the UI shows basic keys such as digits, dot, clear etc.

<Location of the main Java class>
“/src/main/java/backend/Main.java”

<Location of the Jar file to execute>
For Mac = “/out/artifacts/software_design_vu_2020_jar/Calculator-1.0_Mac.jar”
For Windows = “/out/artifacts/software_design_vu_2020_jar/Calculator-1.0_Windows.jar”

<Video for the implementation>
https://youtu.be/IEboSfTRLXA

Time logs

Team number	19		
Member	Activity	Week number	Hours
Group	Meeting	3	0.5
Group	Further outlining system	3	3
Hugo & Yudai	Class diagram	4&5	5
Hugo & Yudai	Object diagram	4&5	3
Toyesh & Dora	Sequence diagrams	4&5	6
Toyesh &Dora	State machine diagrams	4&5	4
Group	Implementation	4&5	8
Group	Meeting	4	0.5
Group	Finishing document& formatting	5	2
Group	Meeting	5	0.25
		TOTAL	32.25