**SWE 510 Project**

**RPN Calculator using Stack in Java**

**Student**: Osman Selcuk Sarioglu

**Date**: 20/12/2020

**INTRODUCTION**

This project report contains basic information about RPN Calculator coded in Java. RPN Calculator gets input from user in two ways: i) Graphical User Interface ii) Console / command line (shown in Figure 1 and 2)

User enters numbers and arithmetic operators at either GUI or console, and gets calculated result in GUI display or console.

Input can be integers or float. User can enter numbers in clear format (like 2.13 , 5000, 12345) or with thousand or decimal formats like 12,345.67. Calculator understands whether user puts the expression in allowed format or not. If there is an error identifed for input, user gets relevant error code as output.

Output of calculations is always shown to user with thousands separated format (i.e. 12,345.67)

|  |  |  |
| --- | --- | --- |
| **Figure 1**: RPN Calculator GUI |  | **Figure 2**: Console input |

**INSTALLATION MANUAL**

RPN project is written in Java Programming Language having JRE System Libraries with JDK 14.0.2. Coding is done in Eclipse IDE version 4.17.0

Program code will be provided with a zip code uploaded to Moodle. Zip code will contain 3 different file folders (zip files) having content below:

* Java source code and executable JAR file
* Java test code
* Javadoc files and Project report.

Java source code should be imported in order to run the code in Eclipse or other editor applications.

Source code is packed in “tr.edu.boun.osmansarioglu” package

Test Suite is created in Junit test environment, similar to main code, it should be imported.

Since the program has GUI, executable JAR file can be run to use RPA calculator.

**USER MANUAL**

RPN Calculator starts running when “Engine” class is executed.

RPN Calculator gets input from user in two ways: i) Graphical User Interface ii) Console / command line

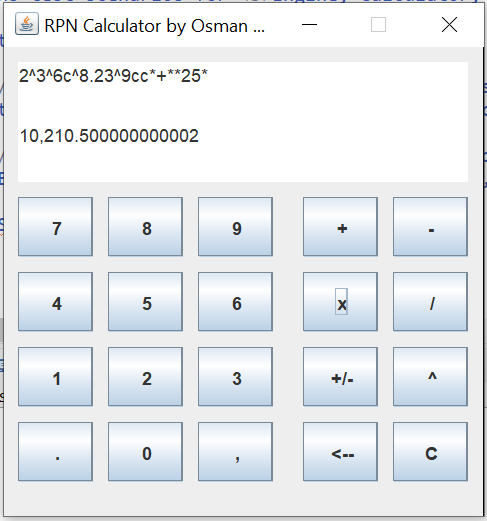
User enters inputs either as integers or floats. User can enter numbers in clear format (like 2.13 , 5000, 12345) or with thousand or decimal formats like 12,345.67. Calculator understands whether user puts the expression in allowed format or not. If there is an error identifed for input, user gets relevant error code as output.

Output of calculations is always shown to user with thousands separated format (i.e. 12,345.67)

**User Manual for GUI**

Graphical User Interface (GUI) has 4 main parts

* Input and Output Displays
* Numpad including numbers, decimal and thousand separator, sign change, and enter button
* Aritmetic Calculations buttons
* Input correction buttons: Backspace (<--) and Clear (C)



Input Correction buttons

Arithmetic Operators

Change sign and Enter

Output Display

Input Display

Number Pad

Whenever user clicks on buttons (other than arithmetic operators), input display is updated. Any input is added at the end of input display until clear button“C” is clicked.

Backspace button “<--“ deletes last character of string at input display, or last error message received at output display.

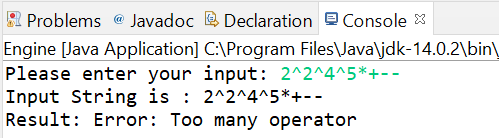
Change sign button “+/-“ puts a “c” character at input display. User can add multiple “c” , and final sign of number is considered by changing number N times ( N: # of c ).

There is no “Enter” button for triggering calculation. “^” refers input number for calculation

Whenever, an “Arithmetic operator” is clicked, input display is updated with adding operator at the end, calculation is triggered for string showed at input display. Result of calculation is shown at output display.

**User Manual for Console Input**

User can also type an input at Console of Eclipse IDE (shown below)



When program is executed, console and GUI starts simultenously, so user can input expressions at console as well.

Console gives a message to user “Please enter your input”, and waits for user to write express and click on “Enter” at keyboard.

As soon as input is entered, calculator shows entered expression by user, and shows result of calculation at a new line.

**Error Handling at Calculator.**

If user enter numbers in wrong format, or clicks or enter arithmetic operators excessively (maximum number of operator cannot exceed N-1, where N is equal to # of number inputs), user gets an error message at output display or console as “result”.

Examples for inputs and error messages are below:

|  |  |
| --- | --- |
| *Expression* | *Error Message* |
| 12osman^3^5\*\* | Error: Non Numeric Value |
| 1^2-+ | Error: Can't calculate: Too many operator or Division to 0 |
| 1.000,2^1\* | Error: Wrong Usage of Thousand Seperator |
| 1.000.000^1\* | Error: Wrong Usage of Decimal Seperator |
| 1,00,000.00^1\* | Error: Wrong Usage of Thousand Seperator |

**MAINTENANCE MANUAL**

RPN Calculator program is coded in Java.

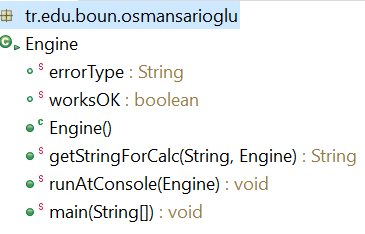
Code has 4 main Classes, and they are all at the same package “tr.edu.boun.osmansarioglu”

* **Engine** : The Engine class has the code for triggering calculations. This has “main” method to run the program
* **Separator** : The Separator class has the code for dividing given String input into proper pieces and storing them.
* **Calculator** : The Calculator class has the code for making calculations
* **CalculatorGUI**: The CalculatorGUI class implements an application that creates a GUI for RPN Calculator

Below table shows the content of each class.

|  |  |
| --- | --- |
|  |  |
|  |
|  |

**Engine Class:**

The Engine class has the code for triggering calculations. It uses **Calculator**, **CalculatorGUI**, **Separator** classes for collecting data inputs, and call methods for calculations.

It has 3 different methods and 1 constructor to make these operations:

Engine(): This constucter is used to initialize Error state for class. worksOK is true, and errorType is blank.

getStringForCalc(String strInput, Engine runningEngine) : Makes calculation for given "String" **strInput** and returns output as "String". "Engine" runningEngine is used to track "Error state" during all operations.

This method gets strInput String as parameter. **strInput** is representing calculation input that is sent to calculator. **runningEngine** is used to handle calculation state and error message handling.

It returns result for calculation as String so that GUI display shows calculation result or error code.

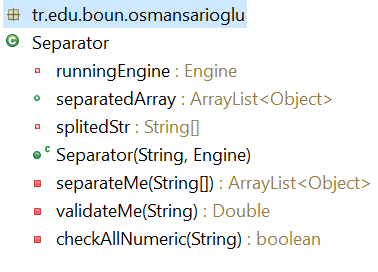
runAtConsole(Engine runningCalc) : Enables inputs from user by using "System.in" and show results with "System.out"

This method is used to make calculation by using "Console " enabling user to input as a "String" and gives result at "Console".

"Engine" object **runningEngine** is used to track "Error state" and handle calculation state and error message handling during all operations.

main(String[] args) : Main method to run code. This triggers GUI enabling, console running waiting for inputs. Engine object **newCalculator** and CalculatorGUI object **rPNCalculator** are initiated.

Method of runAtConsole is run to enable user to give inputs, and get results.

**Separator Class:**

The Separator class has the code for dividing given String input into proper pieces and storing them.

It uses **ArrayList** class containing **Object** class for storing data after validating their format.

It has 3 different methods and 1 constructor to make these operations:

Separator(String inStr , Engine engine) : This is used a constructor for new Separators. It gets parameter as String **inStr** and Engine object **engine** which is used to track "Error state" during all operations. Parameter **inStr** is representing the input to be split and validated.

separateMe(String[] valStr) : Divides given String arry **valStr** into pieces, and keep them at ArrayList of Object type after *validation* whether they are *numbers* or *operators*. Object type is used to have flexibility at “Engine” object which sends parameters to "Calculator" object.

This method is used to transform the given String array by merging into meaningful format. Output of method is stored in an ArrayList having different types of Objects at the same time: *Double for numbers* and *String for operators*.

Method gets **valStr** String array as parameter. **valStr** represents the set of characters to be validated and merged in proper format. Method calls **validateMe()** method for number validation, and returns validated output or calculation operators into an ArrayList having different types of Objects at the same time: Double for numbers, and String for operators.

validateMe(String checkMe) : Makes validation of given String **checkMe** whether is a numerical expression in expected format.

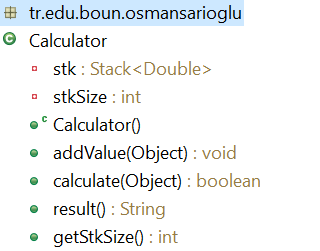
Output of method is returned as Double. If the validated String has no proper format, i.e. non-numeric, wrong usage of decimal or thousand separators ".," assign **runningEngine** working state as "false" and relevant error message is input to **errorType**.

Method gets **checkMe** String as parameter. **checkMe** represents the String to be validated whether it can be a number or not to be validated and merged in proper format.

Method returns a Double value by converting String into double format. If there is an error identified at format, method return 0.0 and updates error handling fields.

checkAllNumeric(String checkStr) : Makes validation of given String **checkStr** whether there is *no non-numerical* expression at the string.

This method is used to validate the given String **checkStr**has only numbers or allowed signs ".,-". Output of method is returned as true or false. If the validated String has proper format, i.e. numeric, method returns *true* and if there is any non-numeric value in it, it returns *false*.

**Calculator Class:**

The Calculator class has the code for making calculations. It uses **Stack** object having **Double** numbers for collecting data for calculations. It has 3 main methods, 1 constructor and 1 getter method to make these operations:

"Object" type is used to have flexibility at Engine which sends parameters to Calculator.

Calculator() : This constructor is used to initialize Stake **stk** for class.

getStkSize() : This getter method is used to get private valiable stkSize’s value, which represents the number of items stored at stk.

addValue(Object dblNo) : Adds given object "double number" on top of Stack

This method is used to add values on top of Stack **stk.** Since Stack **stk** contains Double objects, given Object **dblNo** is transformed to Double first. **dblNo** is sent as a parameter to this method, and it's a Double object sent from ArrayList of **Separator** object.

calculate(Object operator) : Makes mathematical calculation according to given object "mathematical operator" by popping top 2 numbers at Stack and putting the result on top of Stack

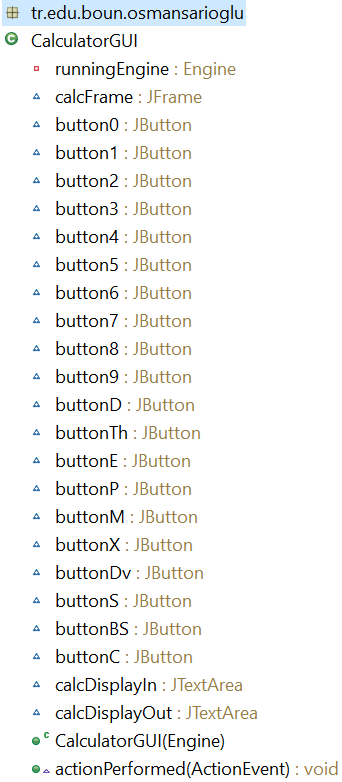
Method gets **operator** Object as parameter. **operator** is a String object sent from an ArrayList of Separator object having different types of Objects at the same time: *Double for numbers* and *String for operators*. Since calculation sign is a String value, given parameter Object **operator** is transformed to String class while using.

Method returns true or false:

* *true:* There is enough value to make calculation, and calculation is done.
* false: There is not enough value to make calculation, and calculation cannot be done.

result() : Returns value of number on top of Stack

This method is used to send *result* by sending top value at Stack. Value is formated according to *n,nnn,nnn.n* pattern. "," stands for *thousand* separator, and "." represents *decimal* separator.

**CalculatorGUI Class:**

The CalculatorGUI class implements an graphical user interface that visualize RPN Calculator. This calculator creates an input string for RPN Calculator and whenever a calculation identifier is clicked on.

GUI sends latest string and calls result for calculation

GUI has input and output displays. "Input display" shows latest string created, and when a calculation is triggered the result is shown at "Output Display"

It has 1 method and 1 constructor to make these operations:

CalculatorGUI(Engine engine) : This constructer is used to setup components of GUI.

Component layout has:

* Number Buttons: 0,1,2...9
* Decimal and Thousand Separators: . and ,
* Arithmetic Operators: +,-,x,/
* Input Operator: ^
* Sign Changer: +/-
* Correction Buttons: Backspace (<--), Clear (C)

GUI has size of 335ppx width, and 350ppx height. Constructor gets Engine object **engine** as parameter to manage "Error state".

actionPerformed(ActionEvent e) : This method is used listen click events and trigger action when any buttons is clicked. There are 2 main actions when a button is clicked.

1. Add value of button at input display, and update stringat display.
2. Trigger calculation when an operation button is clicked and display result

Method gets ActionEvent **e** as parameter, and this parameter listens click actions.

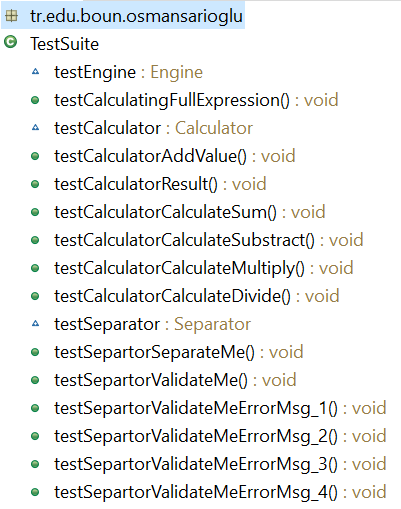
**Codes for Program and JAVADOCs**

Code of program has 4 class files for objects mentions above.

Code has “comments” in details mentioning how operations are done at each method including explanation of declarations as well.

Each method has its JavaDoc comments, and javadoc files are provided in zip file shared with project paper.

Code set contains TestSuite, which is mentioned at next section, as well.

**Test Scenarios and Test Suite:**

Test Suite for RPN Calculator contains test scenarios for Engine, Calculator, Separator classes and their methods .

Below tests are considered :

* **Engine tests:** testCalculatingFullExpression
* **Calculator tests**: testCalculatorAddValue, testCalculatorResult, testCalculatorCalculateSum, testCalculatorSubstract, testCalculatorMultiply, testCalculatorCalculorDivide
* **Separator tests** : testSepartorSeparateMe, testSepartorValidateMe, testSepartorValidateMeErrorMsg\_1, testSepartorValidateMeErrorMsg\_2, testSepartorValidateMeErrorMsg\_3, testSepartorValidateMeErrorMsg\_4

**testCalculatingFullExpression** :

This test contains 19 different scenarios, and compare expected output of RPN calculator, and actual output. If all scenarios give "True", test passes. Any failure at tests results as "Fail"

Test scenarios covers all cases mentioned at project paper, and includes cases expecting "Errors"

Scenarios:

|  |  |  |
| --- | --- | --- |
| **Expression** | **Expected Result** | **Test Result** |
| 2^3+ | 5.0 | TRUE |
| 2^3- | -1.0 | TRUE |
| 2^3^4++ | 9.0 | TRUE |
| 2^3^4+- | -5.0 | TRUE |
| 2^3^5+\* | 16.0 | TRUE |
| 2^3^5\*+ | 17.0 | TRUE |
| 1^2+3+4+ | 10.0 | TRUE |
| 2c^3c- | 1.0 | TRUE |
| + | Error: Can't calculate: Too many operator or Division to 0 | TRUE |
| 2+ | Error: Can't calculate: Too many operator or Division to 0 | TRUE |
| 1^2-+ | Error: Can't calculate: Too many operator or Division to 0 | TRUE |
| 2.1^3.05+ | 5.15 | TRUE |
| 1ccc^1\* | -1.0 | TRUE |
| 1.2c^1\* | -1.2 | TRUE |
| 12,345.67c^1+ | -12,344.67 | TRUE |
| 1.000,2^1\* | Error: Wrong Usage of Thousand Seperator | TRUE |
| 1.000.000^1\* | Error: Wrong Usage of Decimal Seperator | TRUE |
| 1,00,000.00^1\* | Error: Wrong Usage of Thousand Seperator | TRUE |
| 12osman^3^5\*\* | Error: Non Numeric Value | TRUE |

**testCalculatorAddValue**:

This test checks whether Calculator class **addValue()** method works properly. If all scenarios give "True", test passes. Any failure at tests results as "Fail".

Test scenarios covers 2 cases to add a value to Stack of Calculator

Both values are added, and countStackSize was same as the actual size.

**testCalculatorResult**

This test checks whether Calculator class **result()** method works properly. This method shows the value at top of stack. 2 values are added to stack and test gets expected output as last number added to stack.

**testCalculatorCalculateSum, testCalculatorSubstract, testCalculatorMultiply, testCalculatorCalculorDivide**

These test checks whether Calculator class calculate() method works properly according to given operator :

* Scenarios checks whether "Sum", "Substraction", "Multiplication", "Division" of 2 numbers are correct or not

All scenarios get expected result, when test code is run.

**testSepartorSeparateMe**

This test checks whether Separator class **separateMe()** method works properly. This method divide a given String into pieces, and keep them in an ArrayList with Object container. ArrayList contains Double numbers, and String operators.

Scenarios checks whether ArrayList contains right value after split. This is tested by finding size of ArrayList when an expression “2^3^4^5+++” is prosessed. Moreover, 4th element in arraylist is checked. Both gets expected results.

**testSepartorValidateMe**

This test checks whether Separator class **validateMe()** method works properly. This method validates a given String is numeric with expected format. It results Double value of given String into *seperatedArray* . Method returns "0.0" and triggers Engine.worksOK as "False" when format is not in suitable for converting a number. It returns proper "Error Message" with updating Engine.errorType

Tests given right outputs when expression "2.23^5.23.4^" is processed.

**testSepartorValidateMeErrorMsg\_1,2,3,4:**

This test checks whether Separator class **validateMe()** method works properly. This method validates a given String is numeric with expected format. It results Double value of given String into *seperatedArray* . Method returns "0.0" and triggers Engine.worksOK as "False" when format is not in suitable for converting a number. It returns proper "Error Message" with updating Engine.errorType

* Msg\_1 Error is triggered when someone enters more than 1 decimal separator
* Msg\_2 Error is triggered when someone uses thousands separator in wrong place
* Msg\_3 Error is triggered when someone uses thousands separator at decimal place
* Msg\_4 Error is triggered when someone enters a non-numeric expression at console

Test suite gets correct error messages for each scenarios.