**Polynomial Calculator**

**Student: Flueran Robert-Denis**

**Group: 30423**

**Semigroup: 1**

**1.Homework Objectives**

The main objective of the homework is designing a calculator for polynomials, capable of performing all the basic operations between two polynomials.

Steps (objectives) in designing the polynomial calculator:

* Creating Java classes that describe monomials and polynomials

Monomials are going to have a coefficient and power. Polynomials are composed of many monomials of different coefficients and powers. More details can be found at [Monomial](#Monomial)

* Creating a class in which basic operations between polynomials are implemented

The respective class is called Logic. It contains the implementation for operations, such as Addition, Subtraction, Multiplication, Division, Derivation and Integration. More details can be found at [Operations](#Operations)

* Designing a GUI by firstly defining the View for the calculator

The class meant to take care of this is View class. There, I have defined how the buttons and the text is arranged in the Graphical User Interface. More details can be found at [View](#View)

* Linking the operations between polynomials with the User Interface such that we can visualize the output and enter the input in an easy manner

The class that makes all of this happen is called Controller. Its sole purpose is to “listen” to the presses of the buttons on the GUI and call the appropriate operations. More details can be found at [Controller](#Controller)

* Making sure that the inputs are always correct and sending the appropriate notifications to the user in case of errors

RegexMatches class contains methods for checking if the input polynomials are correct and outputs appropriate messages in case of errors. The class also contains methods for converting a polynomial to string and viceversa in order the output everything in a readable manner. Even the Logic class has some methods that may correct some wrong input cases. More details can be found at [RegexMatches](#RegexMatches)

* Putting the application to the test with a host of test cases that cover possible mistakes in the input

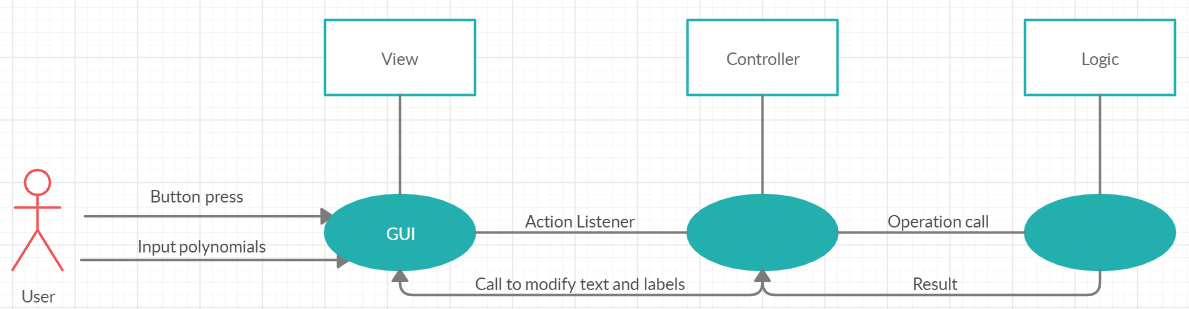
For this, I have made use of the JUnit Library and created a class called TestUnit that checks for all sorts of cases to make sure that the operations have been performed correctly. More details can be found at [Test](#Test)

**2.Problem Analysis**

The problem consists mainly in finding the best method for implementing the operations required.

**“Calculate Polynomial” use-case**

1. User inputs polynomials to take part in an operation
2. User presses a button on the calculator (GUI View)
3. Controller (that implements ActionListener) checks which button has been pressed
4. Assuming that the input isn’t empty, Controller calls the function from Logic that performs the operation called by the button
5. Logic class performs additional checking to make sure that the input is correct and uses methods from RegexMatches to check if the input is indeed a polynomial
6. Logic performs the required operation and returns the result to the Controller
7. Controller returns the result to the calculator (View)



**3.Design**

I have created a host of packages that contain all the classes needed to perform the calculations. They are organized as follows

* Controller

It contains the Controller class which checks for the button pressed and calls the corresponding function in the Logic class

* Datatype (Model)

It defines the data structures that the calculator works it. Such data structures are Monomial and Polynomial

* Operations

It contains the classes that check for correct input (RegexMatches) and perform operations on polynomials (Logic)

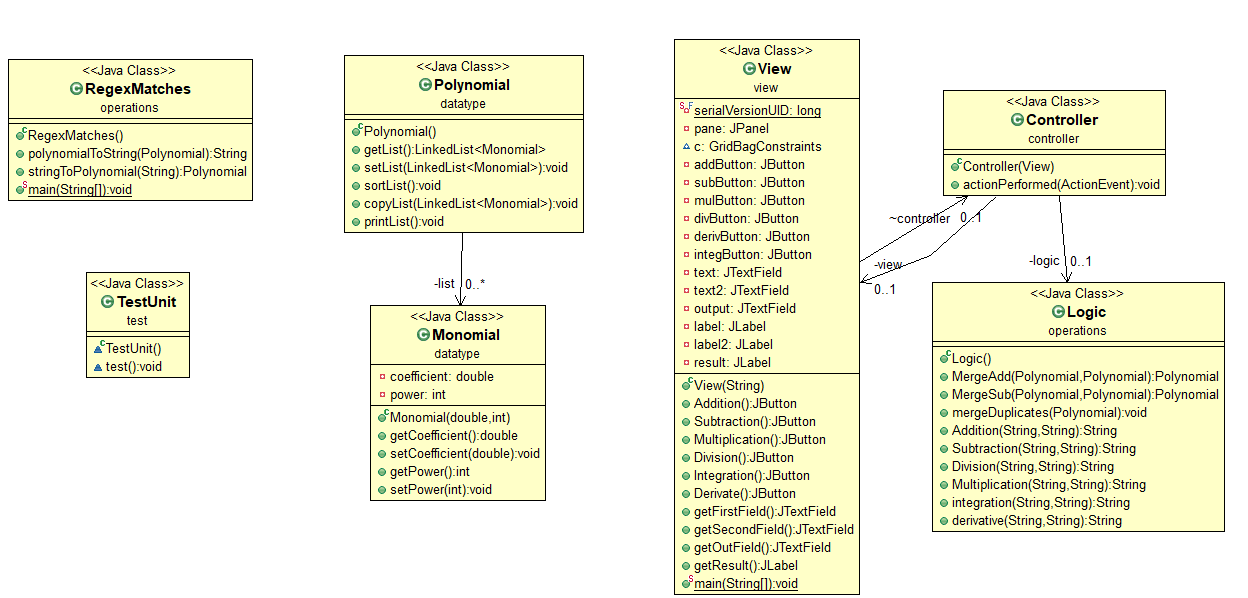
* View

It contains the class in which the GUI is implemented. It has the same name as the package

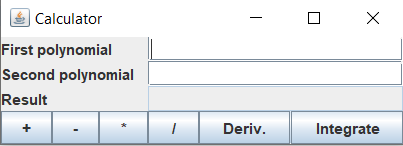
* Test

It contains the JUnit class in which the app goes through all sorts of tests to make sure everything is alright

**UML Class Diagram**



**Graphical User Interface**



6

5

4

3

2

1

1.Addition

Once it’s pressed, it firstly checks if two polynomials have been entered. Afterwards, it proceeds by calling the Addition method in Logic

2.Subtraction

Once it’s pressed, it firstly checks if two polynomials have been entered. Afterwards, it proceeds by calling the Subtraction method in Logic

3.Multiplication

Once it’s pressed, it firstly checks if two polynomials have been entered. Afterwards, it proceeds by calling the Multiplication method in Logic

4.Division

Once it’s pressed, it firstly checks if two polynomials have been entered. Afterwards, it proceeds by calling the Division method in Logic. If the second polynomial is 0, then the application will output and appropriate message

5.Derivation

Once it’s pressed, it firstly checks if a polynomial has been entered in the first text box. Afterwards, it proceeds by calling the Derivation method in Logic

6.Integration

Once it’s pressed, it firstly checks if a polynomial has been entered in the first text box. Afterwards, it proceeds by calling the Integration method in Logic

**4.Implementation**

Let’s start with the datatype (Model) package. Here, we’ve got two classes:

* Monomial

Monomial class simply defines how a monomial looks like. Obviously, a monomial must have a coefficient (which is gonna be of double type to allow real numbers) and a power (which must be an int).

Methods:

This class has no methods except for getters, setters (since we are working with private fiels, we are gonna need these) and a constructor with parameters.

* Polynomial

A Polynomial can be comprised of one or more monomials with different coefficients and powers. So, in the Polynomial class, the one and only private field is going to be a list of Monomial type (LinkedList<Monomial> list).

Methods:

Here we have as well getters, setters and a constructor, but we also have methods that helped with the implementation of the calculator operations (sortList, that sorts the monomials in a polynomial in an increasing order by their power; copyList, which simply copies a polynomial’s list to another’s) and debugging (printList, which simply prints the monomials in a polynomial)

VERY IMPORTANT NOTE

In order for the polynomial calculator to work properly, you have to write the polynomial in this fashion: number\*x(or X)^number. The only monomials that don’t need ^ are zero and first degree ones (you can write x without adding ^1 at the end). If coefficient is 1, then it’s not necessary to add it (you can write x^5 without having to add 1\* at the beginning). Oh, and any other letter besides x isn’t accepted.

Let’s get back to our business by describing the controller package:

* Controller

The Controller Class implements ActionListener interface. We are using this class solely in order to check which button has been pressed on the GUI and call the method corresponding to that button and also checking if the fields where polynomials should have been inputted by the user aren’t empty. If everything goes alright, Controller should call methods from Logic class

Methods:

Apart from a constructor, we have the actionPerformed method, which simply checks which button has been pressed (it has an if clause for each operation) and also sets the result of the operations on the Result text field on the GUI

Let’s describe the operations package. Here, we have two classes:

* RegexMatches

RegexMatches’ main purpose is to check if the input polynomials are correct.

Methods:

Here, we have two very important methods. The first function, stringToPolynomial(string s) takes the input from the GUI, parses the string and turns it into a polynomial. In order for it to check if the polynomial is correct, it makes use of the classes from Regex library, it gets rid of all unnecessary spaces (x + 1 turns into x+1), replaces “-“ with “+-“ and then splits the input string by “+”. Afterwards, it checks (using Regex) if each of the new smaller strings (which are supposed to be monomials from the polynomial) are properly written. If anything goes wrong, the method returns null, which will result in getting the text message “Wrong input values”. There is some other code after that makes sure that zero and first degree polynomials don’t actually need ^ in their expression and that monomials with coefficient 1 don’t actually have to have 1\* at the beginning.

The second function, polynomialToString(string s) simply converts the resulting polynomial that we get from an operation into a string, such that we can output it to the GUI in a readable manner. The result of this function in returned to the Controller class, which sets the Result text field on the GUI. The function does the convertion by creating a string and concatenating in increasing order the contents of the polynomial’s list and makes sure that the double coefficient only has two digits after the dot to make stuff more readable (without this, if we divided 2 by 3, we would have got 0.6666666667; instead, we get 0.67). If the polynomial’s list turns out to be empty (output remains “”, as it was defined in the beginning of the function), it means the polynomial is “0.0”.

* Logic

Logic class contains methods for all the calculator’s operations and for merging monomials with equal powers. This class doesn’t have any field variables, since we declare local variables in each method accordingly.

Methods:

Obviously, all methods are very important. We have our standard operations (Addition, Subtraction, Multiplication, Division, Derivation, Integration) and we also have a method for merging monomials with equal powers (MergeDuplicates). If we call the latter, from a polynomial like x^2+1+x+1+x, we’re gonna get x^2+2\*x+2. The class also contains two other methods: MergeSub and MergeAdd. Their purpose is merging the lists of the two input polynomials into a single list of the Result polynomial (the polynomial that we are going to output to the GUI), one is for the addition operation, and the other is for the subtraction. I could have merged these two methods into a single one, but it may have taken a little more lines of code.

All operation methods are firstly declaring two polynomials and a RegexMatches object. Afterwards, they try parsing the strings received as parameters to the functions to see if they are polynomials. If everything goes right, the two declared polynomials get their values from the previously parsed strings. If any of the two polynomials is null, it means at least one of the inputs has a mistake in it and cannot be converted to a polynomial (this results in the output message “Wrong input values”). The two polynomials are then sorted in increasing order and their monomials with equal powers are merged. Only after all of these, the corresponding calculator operation is performed and the result is returned to the result polynomial. The result polynomial is then converted to a string and it’s returned to the Controller.

There are a few operation methods that are a little different. For example, Division also checks if the 2nd polynomial is “0.0” (in this case, it’s going to return “Division by zero not allowed”). Derivative and Integration only care about the first polynomial.

Let’s now talk about view package:

* View

Here, we define the Graphical User Interface. It contains three labels (First polynomial, Second polynomial, Result), a text field for each label (text fields have size 20), and a button for each calculator operation (six buttons total). The GUI’s panel has GridBagLayout. The buttons are placed one after the other, and the text fields have a large gridwidth such that the buttons fit underneath them.

Methods:

All the components of the GUI have getters and setters such that they can be accessed and modified by the Controller class after operations are performed. Obviously, buttons have action listeners such that we can tell which button has been pressed by the user.

**6.Results**

To make sure the application is working as intended, I’ve made use of the JUnit library and created a class (called TestUnit) in which I put the application through all sorts of test cases. There, I have added comments to highlight all the operations I did and tested such that you can clearly see from the code what has been done.

Firstly, I tested adding simple monomials to some locally declared polynomials. I’ve even tested what happens if we try adding a monomial with coefficient 0 (and nothing changed in the polynomial, as it was intended).

Afterwards, I started testing all the operations. I’ve started with Addition. We can observe that the result polynomials after performing operations are returned in increasing order (starting from the lowest degree monomials). I’ve went through all other operations after that. The only “special” test case was trying to divide by zero, which resulted in the message “Division by zero not allowed”.

I’ve also went through a few wrong inputs to showcase the testing of the input polynomials. If you run TestUnit, you can also see the error messages in the console. An unparsable string that has to be assigned to a polynomial will actually return null.

**7.Conclusions**

I have managed to perform all of the operations and testing required for the assignment. It has been a really interesting homework for me, since I didn’t get to experience much working with GUI before (I did almost nothing GUI-related before, besides a little GUI for the OOP project) and I didn’t program anything related to polynomials before.

The calculator could be further developed to be more visually appealing and to show the history of recent operations.

**8.Bibliography**

* Regex

<https://www.tutorialspoint.com/java/java_regular_expressions.htm>

* Polynomial division

<https://en.wikipedia.org/wiki/Polynomial_long_division>