DOCUMENTATION

ASSIGNMENT NUMBER 1:

**— POLYNOMIAL CALCULATOR —**

NAME: Chețe Doru-Gabriel

GROUP NUMBER: 30423

1. Objective of the assignment

The main objective of this assignment is to build a calculator for operations between two polynomials or the integration/differentiation of one of them, while providing a graphical user interface that can be used to select the preferred operation as well as to introduce the values of the polynomials in the form of a string that follows the standard mathematical notation. Finally, the user should be able to view the result and compute another operation if needed.

The secondary objectives of the assignment are the exact steps which need to be taken in order to accomplish the main objective, namely:

1. Analysis of the problem and identification of the requirements

2. Development of the design of the polynomial calculator

3. Implementation of the design

4. Testing of the implementation

With these objectives clearly set, the start of the analysis of the problem will be presented:

2. Analysis of the problem, modeling, scenarios and use cases

Analysis:

The first important aspect to consider is the mathematical proprieties of a polynomial. Formally, a polynomial is an expression consisting in variables and coefficients. As for operations, these expressions involve only addition, subtraction, multiplication i.e., the result is also a polynomial. A division between two polynomials is also possible, except it may produce a result that is not a polynomial e.g. 1/(*x*2 + 1). There is however the notion of Euclidean division of polynomials and this notion, A(x)/B(x) results in two polynomials: a quotient Q(x) and a remainder R(x) such that A(x) = B(x)Q(x) + R(x) and degree(R) < degree(B). The quotient can be computer with the use of the polynomial long division algorithm, which will be used for the purpose of this calculator. The operations of integration and differentiation for a polynomial can also be computed relatively easy. For the purpose of the calculator, only the first polynomial will have the features of integration and differentiation.

To sum up, the standard form polynomial of degree 'n' is:



and the result of the addition, subtraction and multiplication can be computed as another polynomial. In the case of division, the long division algorithm will be used, and for the primitives and derivatives the following formulas will be used (the antiderivative and the derivative respectively):

Diagram

Description automatically generated Text, letter

Description automatically generated

Modeling:

In order to model these mathematical constructs in Java, the following propriety will be used: a polynomial can be expressed as a sum of monomials. A monomial is just a polynomial with one term. When the operation of multiplication is performed for example, the terms of one polynomial get multiplied by the terms of the other polynomial, one by one. This can be modeled as a multiplication of monomials.

With this in mind, the graphical user interface model will be considered next. The interface is simple, with just two text fields for two polynomials and a “Result” text field where the result of the operation between the two polynomials or the antiderivative/derivative of the first polynomial. The input should be written in the form cx^e+…, where c and p are the coefficient and the exponent respectively. In the case that c = 1, 1 may not be written, but the exponent is always needed, even for the case x^0 = 1.

The available buttons on the interface will be used to select the operation that is to be performed, after which a new operation may be set.

Scenarios:

In the case that the user inserts an input string that doesn’t correspond to the expected input, such as other characters than digits or ‘^’, capital X or spaces they will be ignored as much as possible, however for certain scenarios the result text field will display an “Invalid input” message e.g. in the case that the user selected polynomial division and the first polynomial (the dividend) has a lower degree than the second polynomial (the divisor).

In the case that the user inserts inputs that are correct, the result will be calculated in the moment that one of the buttons is pressed, and the result will be displayed. Then the program waits for a new input and/or for a button to be pressed again.

Use cases:

Diagram

Description automatically generated with medium confidence

The use case for this program is very straightforward: in order to quickly get the answer to a polynomial operation or to get the antiderivative/derivative of a polynomial, the user provides the polynomials as strings, and then presses one of the buttons that correspond to one of the 6 operations. The division is a special case, since the first polynomial has to be greater than the second. So if this isn’t the case, the user must provide another input. Otherwise, the result will be displayed on the screen, in the result text field.

3. Design of the calculator

For the design of the calculator, the Model-View-Controller architectural pattern is used. Within the model, I initially decided to add 3 classes: Polynomial, Monomial and Operation. The Polynomial object was to have a field with an array of monomials, in order to represent the polynomial. However I quickly realized there was the need for a sort of “translator” that can convert the string given as input to an array of monomial objects. Therefore I added another Translator class, used for the regex expression transformation.

In the Controller, there is just a class with the same name, that will handle the interaction between the model and the View.

For the purpose of this project, I decided to use JavaFX, and therefore I used the resources provided by this framework, namely the .fxml file. This is in fact the View. Here the description of the GUI as well as the action events such as “On Mouse Clicked” can be found. When the requirements of the problem were considered, it was clear to me that there was only the need for a simple interface. There should be text fields for the 2 polynomials that can be operated on, as well as simple buttons that can be clicked on to perform the respective operation. When I thought of how this GUI might look, I considered a real-life calculator. In my opinion, the user will feel familiar with an interface that resembles a physical object that is commonly used, and shouldn’t be needlessly confused with intricate interfaces. After all, the purpose of this application is to quickly compute something that would take a long time to do with pen on paper. So the conclusion was that the GUI should be very simple.

Tying all these elements together is of course the Main class where the main() method resides.

Data structures that are used:

Coming back to the mathematical aspect of the problem, polynomials can be written as a sum of monomials, so within the Polynomial class, there can be a simple ArrayList of Monomial objects and this would suffice for the fields of the class.

Within the Monomial class, there are two fields: coeff, which is a double and the degree which is an integer. I chose the double type for coeff because despite the fact that the requirements of the problems require an input with integer coefficients, some operations such as division, differentiation and integration can result in polynomials with real coefficients. Therefore, this type was a requirement for the program to accurately represent the monomial. The degree is just the exponent of the monomial and will always be an integer.

Given that the ArrayList data structure is used, whenever a method iterates through a polynomial, namely its array of monomials, a for each loop is used and therefore I avoided situations where I remove objects from the array while iterating through it, like in the case of the standardForm() method, that aims to sum up all monomials with the same degree within a polynomial. In this case I instead used another auxiliary array that got all the objects that were to be removed and then I used the removeAll() methods from Collections. This ensures that all monomials that have the same degree will end up being just one monomial object.

UML diagram:

Graphical user interface, application

Description automatically generated

For the UML diagram, the Polynomial class cannot exist without the Monomial class, since it is the type of the array list inside it. Therefore, between these two classes there is a relationship of composition. The Translator class uses Polynomial for building the polynomial from the given string so there is a dependency relation. The same can be said for the Controller class and the Operation class. In the case of the Controller, there are instances of the Polynomial class in its methods, however this is not enough for an association, so it is also a dependency relationship. The Operation class uses Polynomial both as the parameter type and the return type, so this is also a dependency. Finally, the Controller has an instance of Operation and one of Translator, so these are two association relationships.

Algorithms:

The first algorithms that were needed were to rearrange the monomials of a newly created Polynomial object. When an input string is given, a Polynomial object may correctly be created, but it may contain many monomials with the same degree, that perhaps even sum up to 0, so should be eliminated completely from the monomials array list. In order to reduce the monomials array, I thought of an algorithm. First an auxiliary array list is created, and it is at first empty. Then the monomials array of the object is iterated through. If a monomial with the same degree can be found in the auxiliary array, then its coefficient is added to the one already in the auxiliary, and if not, a new Monomial object is added to the auxiliary. Finally, all monomials with the same degree from the original array list will be merged into one. One last step is to again iterate through the reduced array and add the monomials which have a zero coefficient to a “to be deleted” array.

Ordering the monomials from highest degree to lowest in the array list of a Polynomial object was another problem that had to be solved. This can be done using a relatively simple sort with a nested for loop.

Translating a polynomial to a string also needed a solution. This is done with an algorithm that also iterates through the monomials in the array list and builds a string step by step by appending first the coefficient, then “x^” and then the degree. This algorithm also checks for 1 coefficients which aren’t written in the final string for simplicity.

The addition and subtraction algorithms are very straightforward: monomials from both polynomials are added to the same monomials array and then reduced and ordered using the previously explained algorithms. Division, as previously discussed is done with the help of the long division algorithm.

4. Implementation

The Monomial class is very simple, yet the building block of the Model of the application. It has just the fields of coefficient and degree, and some getters and setters.

The Polynomial class has just one field, the Array List of Monomial that represents the sum of monomials that make up the polynomial. Besides getters and setters, there is the public method standardForm() which makes use of the algorithm previously discussed in chapter 3. It makes use of a boolean variable to check if the monomial is already in the reduced list and adds it if it isn’t already there. To remove zero coefficient monomials another for loop is used, in which all such monomials are added to a zeroCoefficientMonomials array. Once this is done they are all removed using the removeAll() method. The orderMonomials() method uses a simple sort in a nested for each loop to order the monomials from greatest degree to lowest.

Operation class contains all operation methods employed in the Controller. Addition, subtraction, multiplication and division methods build polynomials that represent the result and return them. In the case of division there is an array of Polynomials as the return type, where Polynomial[0] is the quotient and Polynomial[1] is the remainder. Integration and differentiation methods return just one Polynomial, that represents the antiderivative or the derivative respectively. This returned object is also created within the method.

The Translator class just contains the regex pattern and the method that transforms a string into a Polynomial object. This is done using the regex library, using Matcher and Pattern objects. Matcher iterates through the found matches and transforms the groups into Monomial objects, i.e. first group would be the coeff field in the Monomial field.

The Controller class has instances of the Operation and Translator objects, which it uses to handle the GUI. For every button there is a method corresponding to a mouse clicked event which calls on the operation an translator to first translate the inputs and then make the appropriate operation. There is also the private utility method print() that translates a polynomial back into a string in order to be printed in the result text field.

Finally GUI.fxml contains the user interface.

Graphical user interface, application

Description automatically generated

This was made using JavaFX library. As aimed in the beginning chapter, it is very simple and straightforward. User input comes from the keyboard in the form of a string in the first 2 text fields. The slider in the middle can be used to adjust the proportion of the 2 panels. The result text field will only update after an operation is performed, i.e. a button is pressed, when the polynomial operations will be computed in the model and a result will then be translated back into a string and set to the result text field.

5. Results

For testing the implementation of the program I used the JUnit library. For the model classes that had relevant methods (Monomial class only had getters and setters) I created a test class. All operations work as expected, however they need subsequent calls to the standardForm() and orderMonomials() methods, since the result is not reduced. This is done within the controller class before converting the monomials array back into a string and displaying it in the result text field. For the integration and derivation operations, the result is also as expected, and in the case of the integration of the x^-1 monomial, which actually results in ln|x| and doesn’t stick to the formula used. This is handled when the monomials array is built back as a string, with a special printAntiderivative function. However, within the Polynomial object the result of this integration is actually x^0, even that isn’t really correct.

The tested scenarios were unordered polynomials, both with coefficients or without (coefficient 1). In the case of division, the example from the support presentation of the project was used, and in this case the result polynomials, quotient and remainder as also as expected. The operations of integration/differentiation on the coefficients also work, however for fractions that have a real result, this will get represented as a double, so a result can be tested only to a certain decimal.

Moving to the polynomial, I tested perhaps the most important method of the project: standardForm(). I checked an example with multiple monomials that have the same degree, and end up summing to 0, so should be eliminated from the final, reduced monomials list. This works as expected and only the monomial with a non-zero coefficient ends up in the final array that represents the polynomial: x^3. For the orderMonomials() test I used a string that indicated a polynomial in a form where monomials aren’t sorted in order of their degree. The method indeed orders them and the result is as expected: highest degree monomial is first: 4x^8, then 7x^6, and so on. A test that the given input string will be converted into a polynomial object and then back into a correct string is also performed. This works, with the added mention that if a monomial is given with a 1 as coefficient “1x^7” when the translateIntoString() method is called the 1 is omitted since it is unnecessary.

Finally, for the translator a test to check that the polynomial object is correctly built is performed.

6. Conclusions

During the analysis and development of this project I learned some lessons. I made some mistakes when handling the monomials array lists and when fixing them I concluded that I have to be more careful when considering adding references of an object to another array. This also made me consider the way the garbage collector feature of Java works. As for the UI part, it was the first time I made a GUI using JavaFX and it was an experience that taught me a lot about interfaces and the way they work: events and handling them, as well as designing the actual visual aspect of an application. Using JUnit for unit testing was also something new for me and it is definitely is very useful and convenient. Modeling a mathematical problem with an OOP approach was another useful experience that I got with this project. All in all, I made some mistakes, got to work with some libraries that were new to me and learned quite a couple of things.

7. Bibliography

<https://regex101.com/r/zH8mE4/4>

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

<https://en.wikipedia.org/wiki/Polynomial#Composition>

<https://www.cuemath.com/algebra/dividing-polynomials/>

<https://www.baeldung.com/javafx>

this article helped a lot in seeing a practical example of how a regex pattern can be processed with the help of the Matcher and Pattern classes: <https://stackoverflow.com/questions/28859919/java-regex-separate-degree-coeff-of-polynomial/44188848>

this provided me with some ideas for the iterations through the for each loops: https://stackoverflow.com/questions/10431981/remove-elements-from-collection-while-iterating