FACULTY OF AUTOMATION AND COMPUTER SCIENCE

DOCUMENTATION

ASSIGNMENT 1

POLYNOMIAL CALCULATOR

CURAC MIHAI – IONUT

30421

CONTENTS

1. Assignment Objective....3
2. Problem analysis, modelling, scenarios and use cases....3
3. Design....5
4. Implementation....6
5. Results. Testing....8
6. Conclusions....8
7. Bibliography....9

1. Assignment Objective

The main objective of this assignment is to design and implement a polynomial calculator with a GUI through which users can insert two polynomials, select the mathematical operation to be performed and view the result. We consider polynomials of one variable and integer coefficients. To achieve this, we followed the following sub-objectives:

- Analyse the problem and its requirements
- Design the polynomial calculator
- Implement the calculator
- Test the entire functionality of the polynomial calculator

The sub-objectives will be thoroughly described in the following chapters, in that order.

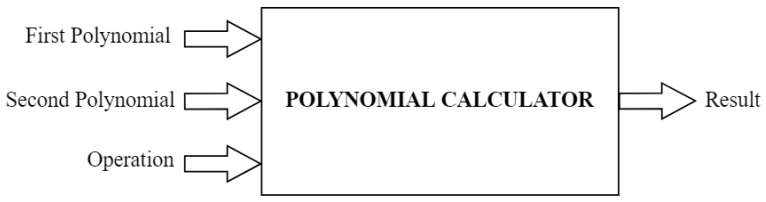
2. Problem analysis, modelling, scenarios and use cases

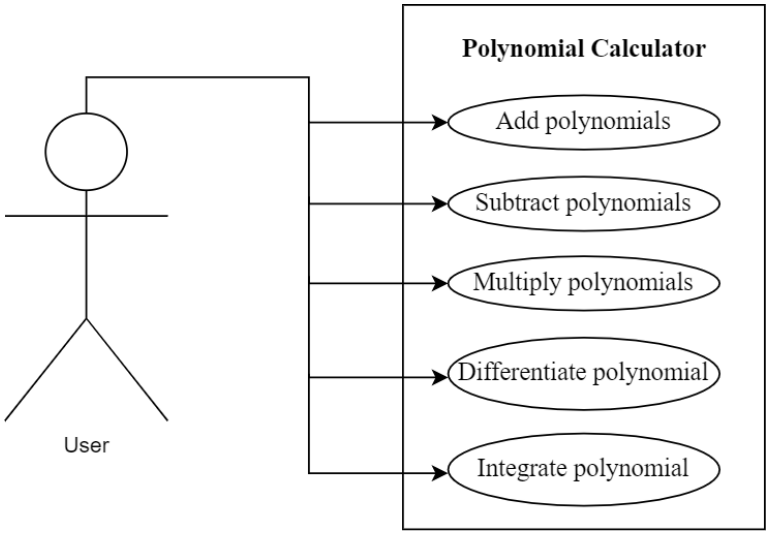
Polynomials are mathematical expressions involving a sum of powers in one or more variables multiplied by coefficients. The individual summands with the coefficients included are called monomials. We will work with polynomials in one variable and integer coefficients, in the following form:

$$a_n x^n + \cdots + a_2 x^2 + a_1 x + a_0$$

Polynomials appear in many areas of science. They are used to form polynomial equations, to define polynomial functions or, in calculus and numerical analysis, to approximate other functions. Thus, a calculator with a friendly graphical user interface capable of doing operations on polynomials can be of great use to many people.

After starting the program, the user will be able to introduce two polynomials in the GUI's text fields. They can be written in the standard form presented above, but the calculator is designed to also support other input formats, like typing the monomials in a random order (without worrying about ordering them by their degrees). This will be thoroughly presented in the next sections of the documentation. Next, the user can select one of the five operations that the calculator is capable of performing and see the result at the bottom of the window. These can be considered the functional requirements of the project – the polynomial calculator should allow users to insert the two polynomials, select the mathematical operation from the set of 5 available (addition, subtraction, multiplication, differentiation, integration), then perform the operation and display the result. Here is the "black box" of the system and then the use-case diagram:





We will further describe the calculator's use-cases that are available to the user. They are the addition, subtraction, multiplication, differentiation, and integration use-cases and have mostly the same success scenarios and alternative sequences.

For the *addition, subtraction* and *multiplication* use-cases, the main success scenario is:

1. The user inserts two valid polynomials in the calculator's text fields.
2. The user selects the desired operation to be performed on the two polynomials.
3. The polynomial calculator performs the operation and displays the result.

The alternative sequence is:

- The user inserts incorrect polynomials, that do not respect the developed algorithm's pattern recognition.
- The scenario returns to step 1. The user is still invited to insert two valid polynomials as long as the window remains open.

For the *differentiation* and *integration* use-cases, the main success scenario is:

1. The user inserts a valid polynomial in the first text field of the calculator. The second text field can be left blank or can be filled with another polynomial.
2. The user selects the differentiation or integration operation.
3. The polynomial calculator performs the operation on the first polynomial and displays the result.

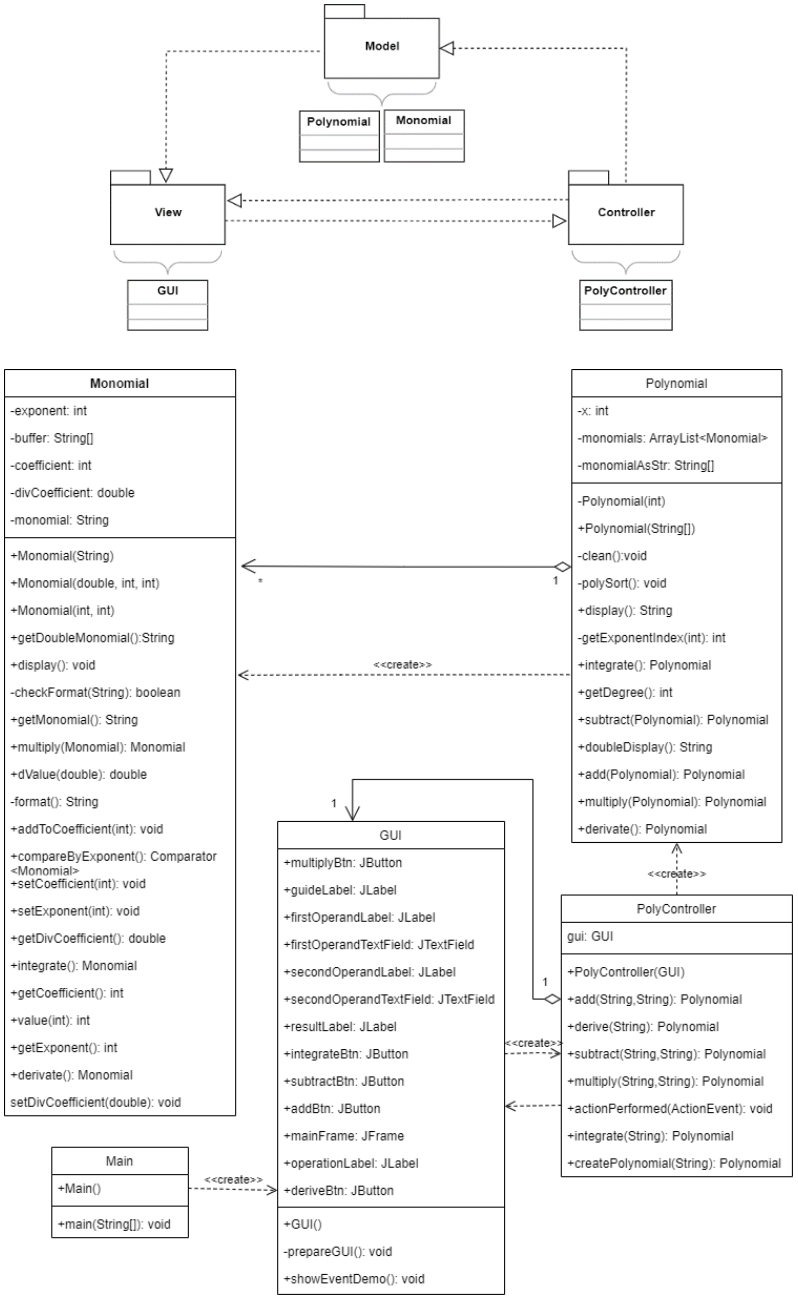
The alternative sequence is:

- The user inserts an incorrect polynomial in the first field, that does not respect the developed algorithm's pattern recognition.
- The scenario returns to step 1. The user is still invited to insert a valid polynomial as long as the window remains open.

Furthermore, we can mention as non-functional requirements the following: the calculator should be easy to use and intuitive, the program should exit only when closed by the user.

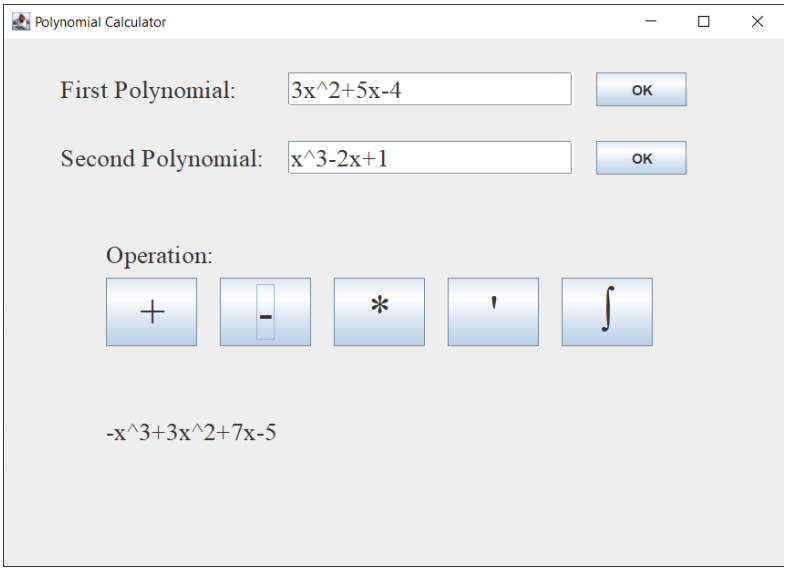
3. Design

The project uses 4 main classes organized according to the Model View Controller (MVC) Architectural Pattern. Thus, the application logic is separated from the user interface while designing the software using model designs. The Model represents the business layer of the application. Its components encapsulate core data and functionality. The View represents the presentation layer of the project and is used to visualize the data that the Model contains. Finally, the Controller works on both the Model and the View and is used to manage the flow of the application, more exactly the data flow in the model object and to update the view whenever data is changed. It receives input from the keyboard and mouse buttons as events that are translated to service requests, which are sent either to the Model or to the View. Below is the division into packages and classes and the UML Class Diagram.



The data structures we used were mostly primitive data types, strings and ArrayLists. Specifically, the class Monomials contains the *coefficient*, *exponent* int fields that are part of any monomial, the *divCoefficient* double field for non-integer coefficients, like the ones resulted after the integration of a polynomial, and a *buffer* String array used for splitting and formatting the string representing a monomial. The Polynomial class has an ArrayList of monomials as its main field, that will be used to add, remove and process each monomial of the given polynomials.

The GUI class uses Java Swing to provide an easy to use and simple graphical interface for the calculator. We used components like JFrame, JButton, JLabel, JTextField, and set Action Commands for each of the buttons. The GUI is shown in the screenshot below.



4. Implementation

The main goal of the implementation was to give the final user as much flexibility as possible when entering the two polynomials. Therefore, the class **Monomial** has a constructor *public Monomial(String mon)* that takes the string containing the monomial and calls the *checkFormat* method. In this method, we used *regex* multiple times and pattern matching in order to check the degree of the monomial and whether its coefficient is different from 0. To find the exact values of the coefficient and the exponent, we had two big possibilities. The first one was given by the absence of a "^" character introduced by the user, in which case we just split the string by the variable character or the character "*" and found our values. In case the user explicitly states the exponent, we had to split the string twice, once by the "^" character and once like in the previous case. We made sure the users can introduce polynomials in many valid ways, like:

- 3x^2+5x-4
- 5x^1-3x^0
- 5*x^4+3*x+2
- 8x^3-x+1

The last example demanded additional lines of code, since we needed to tell the program that just a "-" near the variable means that its coefficient is -1. We also gave users the possibility to write the polynomials' components in any order they want, since the program knows how to sort the monomials in descending order by their degree.
The method *public String getMonomial()* handles the return of the monomials as strings, after all the arithmetic operations and transformations have been done on them. We made sure that the monomials are displayed as friendly as possible, so we only used the "^" character when the exponent is different from 0 or 1. Similarly, the coefficient is displayed only if it's different from 1 or -1. *public String getDoubleMonomial()* is the complementary method that handles the case in which we need to display a monomial with real coefficients i.e. when the integration operation is done. Here, we check whether the coefficient is an integer number or not, so as to add decimal points only when necessary. Thus, we avoid returning monomials as "1.0x^3+2.5x^2-3.0x", for example. Our program returns "x^3+2.5x^2-3x" for readability and ease of use.
The *addToCoefficient, multiply, derivate, integrate and compareByExponent* methods in the Monomial class perform operations on the coefficients and exponents of the monomials and are adjacent to the similarly named methods in the Polynomial class.
In the **Polynomial** class, the *public Polynomial add(Polynomial poly2)* method handles the addition of two polynomials. We add everything in a new polynomial called *result*. This is done by firstly taking each monomial from the first polynomial and checking whether its exponent appears in one of the monomials of the second polynomial. If so, then we add the two coefficients and keep the exponent intact, while also not forgetting to remove the monomial from the second polynomial afterwards. If not, then we just add the monomial from the first polynomial to *result*. Finally, we add the monomials from the second polynomial that remained unchanged, since they had different degrees to the ones in the first polynomial. We also need to remember to sort and clean the result polynomial. More details will follow on this subject.
The subtraction operation is very similar, we only needed to multiply the coefficients of the second polynomial by -1 and then perform the addition operations on the two.
The *public Polynomial multiply(Polynomial poly2)* method handles the multiplication of two polynomials, by taking each of the monomials of the first polynomial and each of the monomials of the second one and calling the *multiply* method from the Monomial class on each pair of them. This ensures that their coefficients get multiplied by one another and that their exponents are added, with the result being sent back to the method in the Polynomial class. A sort of the result's monomials is necessary here as well.
The *public Polynomial derivate()* and the *public Polynomial integrate()* methods' roles are self-explanatory – the first one takes each monomial and updates its coefficient (it gets multiplied by the exponent) and exponent (it's decremented by 1), while the second one takes each monomial, increments its exponent and updates its coefficient (it gets divided by the exponent). Here, we must pay attention to use the "divCoefficient" field of the monomial, since its coefficients won't always be integers anymore and can be double.
The *private void polySort()* is extremely important, since it searches for different monomials with the same exponent and adds them together, while also reversely sorting the monomials by their degree. This ensures that the polynomials will be displayed in a friendly and easy to read manner.
In the **PolyController** class, the *public Polynomial createPolynomial(String buffer)* method is of upmost importance as well, since it takes the string introduced from the keyboard in the GUI

and prepares it for further processing in the Model classes. Precisely, it replaces every occurrence of the minus sign with the "+-" characters, in order to then split the input string by "+" and take in correctly all the monomials.
To make the connection between the GUI and the controller, we added Action Commands and Action Listeners for each of the operation buttons, got the polynomials from the Text Fields based on the commands and then added the result of the operations to the result label in the GUI.

5. Results. Testing

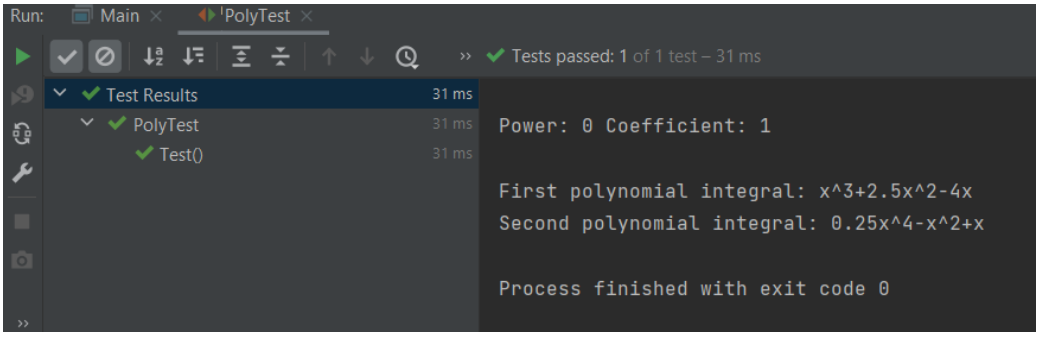
We used JUnit to test all of the operations that the calculator is capable of doing. To prove the correctness of the program, we considered the following two polynomials given as strings:

- 3x^2+5x-4
- x^3-2x+1

with the expected results:

- Sum: x^3+3x^2+3x-3
- Subtraction: -x^3+3x^2+7x-5
- Product: 3x^5+5x^4-10x^3-7x^2+13x-4
- First polynomial derived: 6x+5
- Second polynomial derived: 3x^2-2
- First polynomial integral: x^3+2.5x^2-4x
- Second polynomial integral: 0.25x^4-x^2+x

All of the tests were passed successfully.



6. Conclusions

I learnt many things related to OOP while working on this project, like how all of the different UML diagrams work, all the special relationships between classes and their UML notations, how using the MVC design pattern provides a better way to organize my code and how data can flow between the Model classes and the View that contains the GUI.

Future developments for the polynomial calculator may include a more sophisticated graphical user interface that contains a Clear button and a JComboBox that lets us select on which polynomial to apply the differentiation or integration operations. We could also add error message dialogs that would tell users they introduced incorrect polynomials. Furthermore, we could add support for polynomials with more than one variable and real coefficients.

7. Bibliography

- https://www.javatpoint.com/mvc-architecture-in-java
- https://en.wikipedia.org/wiki/Monomial
- https://java-programming.mooc.fi/part-11/1-class-diagrams
- The Assignment 1 Support Presentation