DOCUMENTATION

Assignment number 1:

Polynomial Calculator

Student name: Lungoci Toma

Group: 30423

Table of contents

1. Objectives

The main objective is to design and implement a polynomial calculator with a dedicated graphical interface. The user can insert the polynomials, select the mathematical operation from the following: addition, subtraction, multiplication, division, derivation and integration, and finally view the result of the operation on the screen. The GUI has been implemented with the JavaFX framework.

Along with the main objective, there are several sub-objectives that must be taken into account. Analysis of the problem along with identifying the requirements, exhibited with the help of use case diagrams and the classification between functional and non-functional requirements. Designing the system into multiple levels of abstraction which resulted in the division in several subsystems such as: Graphical User Interface, Business Logic, and Data Models, and also in structuring the project with the Model View Controller Architectural Pattern in mind. The packages compose the corresponding to provide the necessary functionality. The design of the system is also presented with package and class diagrams. The implementation is done in Java code following the OOP principles. Some implementation details and code snippets are presented. Finally, the testing of the polynomial calculator is done using the Junit framework, to ensure the optimal functionality.

2. Problem analysis, modeling, use cases

Functional requirements:

-The polynomial calculator should allow users to insert polynomials using the keyboard

-The polynomial calculator should allow users to select the mathematical operation

-The polynomial calculator should add two polynomials

-The polynomial calculator should subtract two polynomials

-The polynomial calculator should multiply two polynomials

-The polynomial calculator should divide two polynomials

-The polynomial calculator should differentiate one polynomial with respect to the variable

-The polynomial calculator should integrate one polynomial with respect to the variable

Non-Functional requirements:

-The polynomial calculator should be intuitive and easy to use by the user

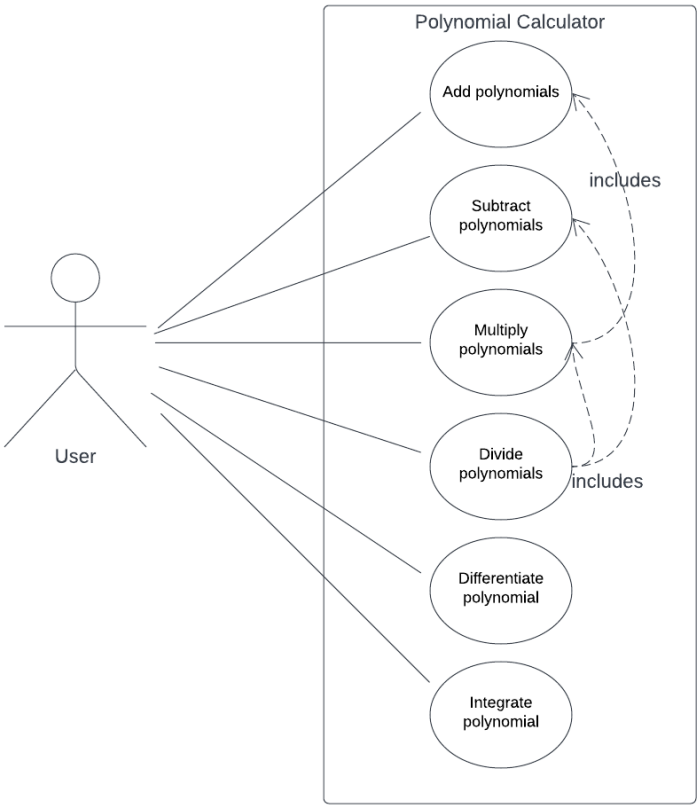
-The polynomial calculator should allow the user to write the polynomials in any way that is mathematically correct also including redundant terms or without combining like terms.

-The polynomial calculator should display the result with the monomials ordered by degree in decreasing order, fully simplified(combine like terms) and with no redundant coefficients or exponents

-The polynomial calculator should display on the screen errors induced by the user input, with a description and type of error that occurred

-The polynomial calculator should perform the operations in a fast and efficient manner

The following page illustrates the Use Case Diagram of the system.



1. Use case: **add polynomials**

Primary Actor: user

Main success scenario:

1) The user inserts the 2 polynomials in the graphical user interface.
2) The user selects the “addition” operation from the choice box.
3) The user clicks on the “compute” button
4) The polynomial calculator performs the addition and the result is displayed on the screen.

Alternative Sequence:

- Incorrect polynomials: The user inserts incorrect polynomials, the scenario returns to step 1) and the error message is displayed on the screen
- Just one polynomial provided: The user inserts just one operand, the scenario returns to step 1) and the error message is displayed on the screen

3. Use case: **subtract polynomials**

Primary Actor: user

Main success scenario:

1) The user inserts the 2 polynomials in the graphical user interface.
2) The user selects the “subtraction” operation from the choice box.
3) The user clicks on the “compute” button
4) The polynomial calculator performs the subtraction and the result is displayed on the screen.

Alternative Sequence:

- Incorrect polynomials: The user inserts incorrect polynomials, the scenario returns to step 1) and the error message is displayed on the screen
- Just one polynomial provided: The user inserts just one operand, the scenario returns to step 1) and the error message is displayed on the screen

2. Use case: **multiply polynomials**

Primary Actor: user

Main success scenario:

1) The user inserts the 2 polynomials in the graphical user interface.
2) The user selects the “multiplication” operation from the choice box.
3) The user clicks on the “compute” button
4) The polynomial calculator performs the multiplication and the result is displayed on the screen.

Alternative Sequence:

- Incorrect polynomials: The user inserts incorrect polynomials, the scenario returns to step 1) and the error message is displayed on the screen
- Just one polynomial provided: The user inserts just one operand, the scenario returns to step 1) and the error message is displayed on the screen

5. Use case: **divide polynomials**

Primary Actor: user

Main success scenario:

1) The user inserts the 2 polynomials in the graphical user interface.
2) The user selects the “division” operation from the choice box.
3) The user clicks on the “compute” button
4) The polynomial calculator performs the division and the result is displayed on the screen(the quotient signified by ‘Q: ’ and the remainder by ‘R: ‘).

Alternative Sequence:

- Incorrect polynomials: The user inserts incorrect polynomials, the scenario returns to step 1) and the error message is displayed on the screen
- Just one polynomial provided: The user inserts just one operand, the scenario returns to step 1) and the error message is displayed on the screen

4. Use case: **differentiate polynomial**

Primary Actor: user

Main success scenario:

1) The user inserts the polynomial in the graphical user interface.
2) The user selects the “derivation” operation from the choice box.
3) The user clicks on the “compute” button
4) The polynomial calculator performs the differentiation and the result is displayed on the screen.

Alternative Sequence:

- Incorrect polynomials: The user inserts incorrect polynomials, the scenario returns to step 1) and the error message is displayed on the screen
- Two polynomials provided: The user inserts two operands instead of one, the scenario returns to step 1) and the error message is displayed on the screen

6. Use case: **integrate polynomials**

Primary Actor: user

Main success scenario:

1) The user inserts the polynomial in the graphical user interface.
2) The user selects the “integration” operation from the choice box.
3) The user clicks on the “compute” button
4) The polynomial calculator performs the integration and the result is displayed on the screen.

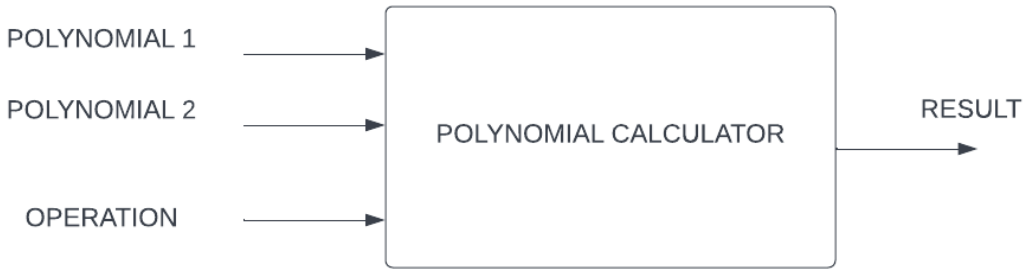
Alternative Sequence:

- Incorrect polynomials: The user inserts incorrect polynomials, the scenario returns to step 1) and the error message is displayed on the screen
- Two polynomials provided: The user inserts two operands instead of one, the scenario returns to step 1) and the error message is displayed on the screen

The multiplication operation uses the functionality of the addition operation, while the division algorithm uses the functionality of both subtraction and multiplication, represented with the dotted line on the use case diagram.

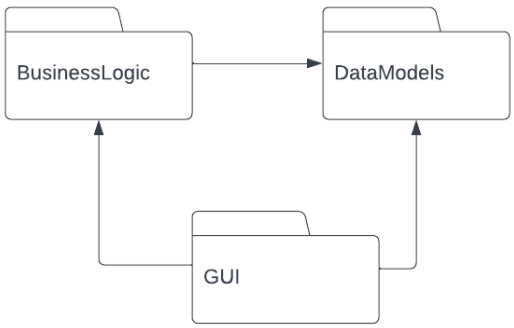
3. Design

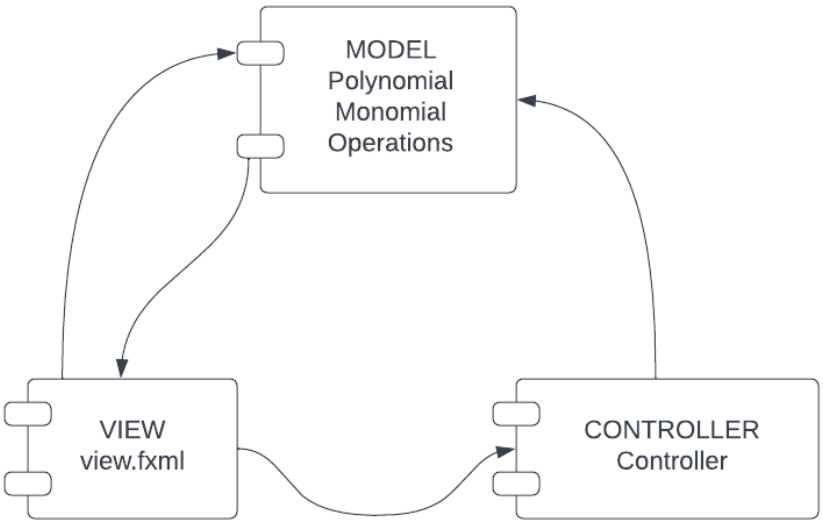
- Overall system design



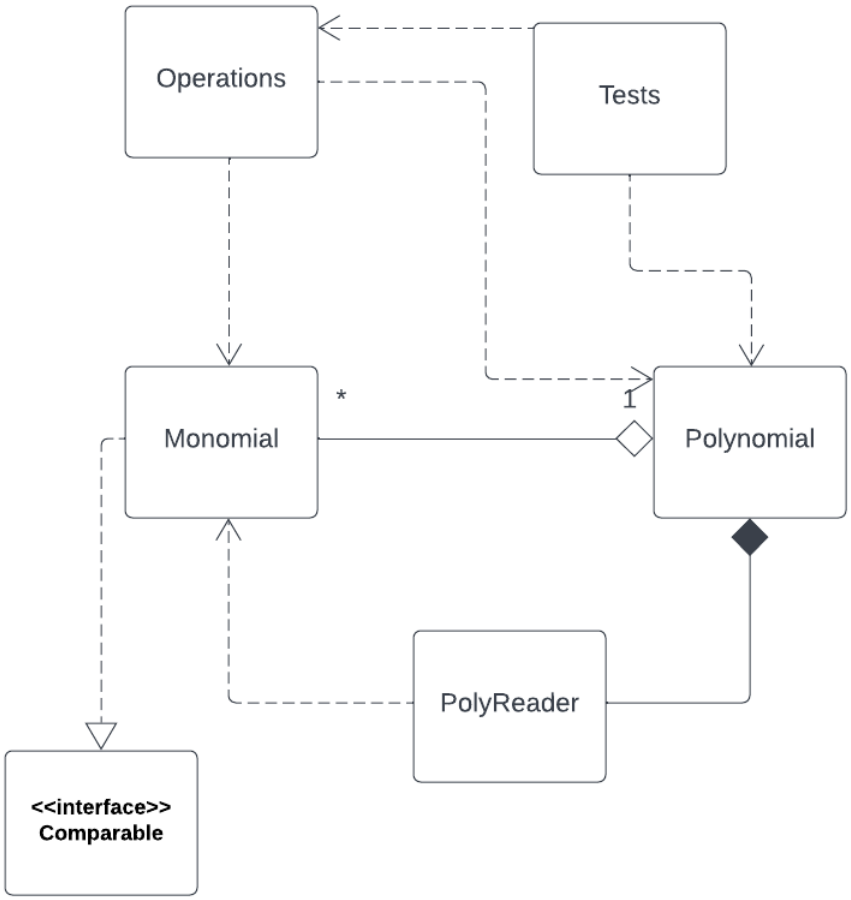
- Division into sub-systems/packages

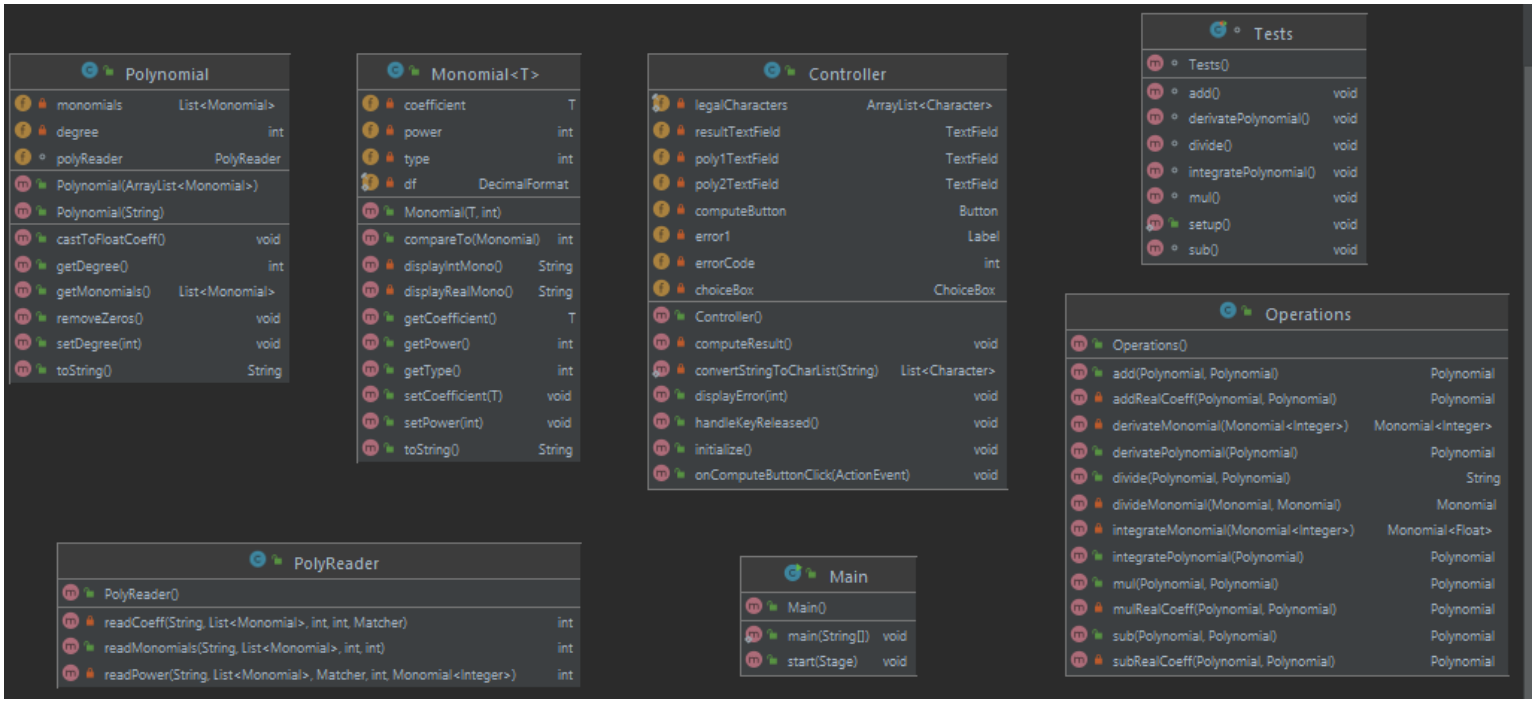
The DataModels represents the “Monomial” and “Polynomial” classes, which store all the necessary data for operating with the mathematical concepts. BussinesLogic holds the “Operations” class, which performs all the operations required by the system while the GUI is described using the FXML markup language, along with a CSS stylesheet. Following the MVC architecture, along with the View(.fxml) we have the Controller(class “Controller”) which maps the View events to changes in the Model, calling the functions associated with the buttons, labels, etc.. The relationships between the Model and the View represent the request of data.





- Division into classes





4. Implementation

The GUI is described in the .fxml file and it contains a GridPane, along with a Button, a ChoiceBox and Labels. The input TextFields will provide the strings from which the polynomials are build.

The Controller is the one responsible with mapping events in the view changes in the model and logic. Before that, it checks if all the input parameters are correct. It displays several error messages in the contrary case. For example, if both fields are empty, if the user enters illegal characters or if we provide two polynomials for operations which require just one, such as Integration. The application will not proceed if the errorCode is 0, that is there are no errors.

```
case 4:
    error1.setText("ERROR: Illegal characters!");
```

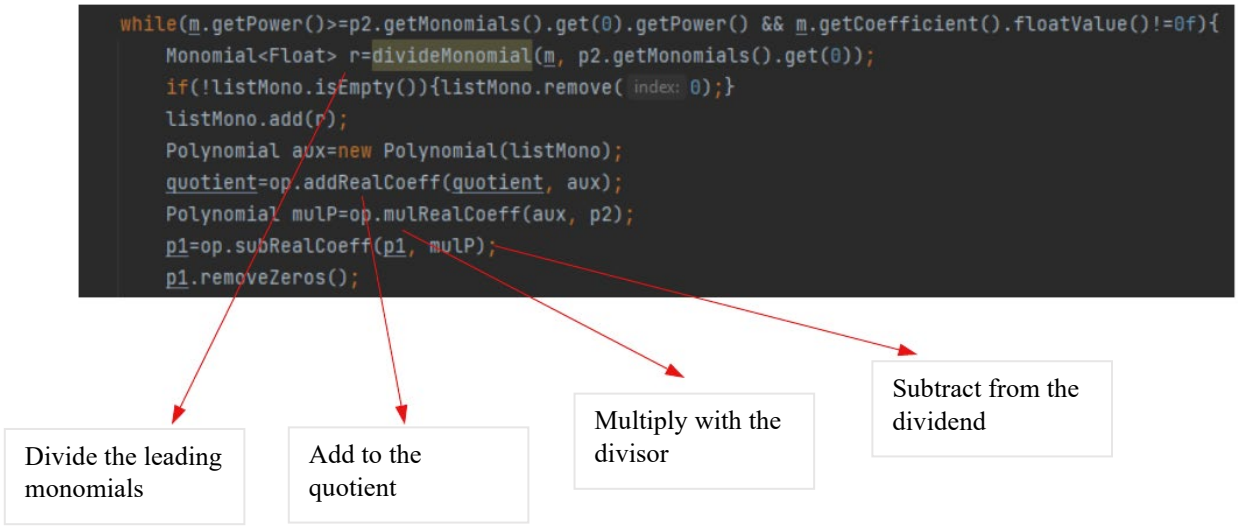
The Monomial object is the “building block” for constructing polynomials. Its most important fields are: the coefficient and the power. The coefficient must be an object that extends the Number class, and more specifically, in our case, either an integer or a real number. Aside getters and setters, there is overridden a ToString method, so that the monomial can be displayed in the form: “coefficient*x^power”. The comparison between monomials is made upon their exponent, so that a collection of monomials can be sorted with Collections.sort.

The Polynomial class is the most important class of the system. It is composed of a list of monomials, a degree and a PolyReader object to provide the utility needed to transform strings into a polynomial object. The polyReader is used in the constructor of the Polynomial, and uses regular expressions and pattern matching to decrypt the input string. The regex that I used groups the string into 4 groups, from which the most important are group 2(the coefficient) and group 4(the power), while the last group, group 4 will truncate the whole monomial. The substrings

```
String POLY_PATTERN = "(\\s*([+-]?\\s*[0-9]*+)\\*?(x\\^?)?([0-9]+)*)";
Pattern pattern = Pattern.compile(POLY_PATTERN);
Matcher matcher = pattern.matcher(s);
```

Are processed and a monomial is constructed and then added in the polynomial’s list of monomials. The approach offers the user some flexibility in typing his input. All like terms are discovered upon first sight and added to the previous coefficient value. Terms such as 0* or x^0 are also evaluated on the spot. The toString method displays the polynomial in descending monomial power, and some additional changes are made: not displaying the ‘+’ sign on the first monomial, not displaying ‘1*’, ‘x^1’ or ‘x^0’. Other than that, the polynomial is displayed in the most formal form.

The Operations class is the one which performs the computation of the result. It contains only the methods for each operation. Generally, it creates a new Polynomial result, which will be returned by the functions. Addition and subtraction are straightforward, just working with the coefficients of same exponent monomials. Integration and differentiation follow the basic rules, modifying the coefficient and increasing respectively decreasing the exponent. The division operation was more challenging to implement, since it requires more work and it involves in itself multiplication and subtraction. It is also different since it returns directly the String with the result, combining the toString of quotient polynomial and remainder polynomial. Here it is also made use of the Polynomial constructor which receives as arguments a list of monomials. Before dividing however, there is verified that it always divide the higher degree polynomial to the lower degree one. The steps used in the division algorithm are the following: Divide the leading monomials, add the result to the quotient, multiply this result with the divisor, and subtract the multiplication result from the dividend. Continue the steps until the only terms left from the dividend are the remainder. I have used a while loop wo make sure that I avoid concurrent modification of a monomial list in a polynomial. Following is a code snippet with part of the algorithm.



5. Results

For testing the functionality of the polynomial calculator I have used the Junit framework. In the Test class there are methods which check the proper working of all the operations. Each test is responsible for testing not only the result of the operation between polynomials, nut also the processing of an input string into a Polynomial object, and the correct display of the result. The general approach was to construct the two polynomials from a string, as in the real world use of the application, instantiate also a Operations object, perform the operation and check the result and the display of the resulted polynomial. For example:

```
Polynomial result=operation.mul(poly1, poly2);
assertEquals("3*x^3-7*x^2+3*x-2",result.toString(),"multiplication is not
correct!");
```

The result is a polynomial object, not a string, so this verification ensures that both the result is correct and also that the displaying of it is the one expected by the user.

6. Conclusions

The project was very useful for acquiring a more in depth knowledge of the different levels of system design and structure, identifying and classifying the requirements, developing useful UML diagrams to describe the models. Implementing the GUI and working with Junit was fun and I have also practiced my OOP programming skills. The project is not perfect but it encapsulates all operations and functionalities. Some further improvements can be made in memory management, efficiency and code reusability, providing the same results with less code.

7. Bibliography

https://www.youtube.com/watch?v=_FSXJmESFmQ

https://www.geeksforgeeks.org/

https://stackoverflow.com/