**Polynomial Calculator**



Teacher: Ioan Salomie

Teacher Assistant: Ciprian Stan

Student: Bodea Razvan-Marius

Group: English\_30422

Year: 2021

1. **Assignment Objective**

Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation to be performed and view the result.

Sub-Objectives:

* + Analyze the problem and identify requirements
  + Design the polynomial calculator
  + Implement the polynomial calculator
  + Test the polynomial calculator

Why create a polynomial calculator?

Because solving polynomial operations on paper with polynomials of higher degrees is difficult and time consuming.

Because it solves a problem that is often encountered in the domain we are studying.

Because it is a good start in designing even more complex designs and applications.

1. **Problem analysis, modeling, scenarios, use cases**
   1. Analysis

A polynomial *P* in an indeterminate *X* is formally defined as:

* where:
  + - *a1*, *a2*, …, *an* represent the polynomial’s coefficients
    - *n* represents the polynomial degree

A monomial is a special type of polynomial with only one term.

A polynomial is a list of monomials (this is important for the way I implement the operations on polynomials).

A monomial is defined by a coefficient and an exponent (aX^b)

a->coefficient

b->exponent

By knowing / finding the coefficient and exponent, we can recreate the monomial or use them for operations because they are the terms that impact the results and the way operations are done.

A way to find monomials and isolate the coefficient and exponent is using regex (pattern building/ recognition). I will discuss this later.

2.2 Modeling

The UI will present the user with 2 fields for the polynomials he wants to perform an operation with.

The user will have access to buttons necessary for the input of a polynomial, or the user can select the text field and use the keyboard for typing a polynomial.

The input will be verified to check if it is in a correct form.

The result is displayed in a separate result field.

Field1 -> Polynomial1

Field2-> Polynomial2

Some operations like integration and differentiation are performed on only 1 polynomial. In this case, the polynomial used is the one in the 1st field.

The result is computed and displayed after the user enters a polynomial and selects the operation they want to perform on it.

Operations:

* Addition (Polynomial1, Polynomial2)
* Subtraction (Polynomial1, Polynomial2)
* Multiplication (Polynomial1, Polynomial2)
* Division (Polynomial1, Polynomial2)
* Differentiation (Polynomial1)
* Integration (Polynomial1)

2.3 Scenarios

The best-case scenario and, also the case the app needs in order to compute a correct result is:

The user provides correct polynomials on the text fields (of correct form)

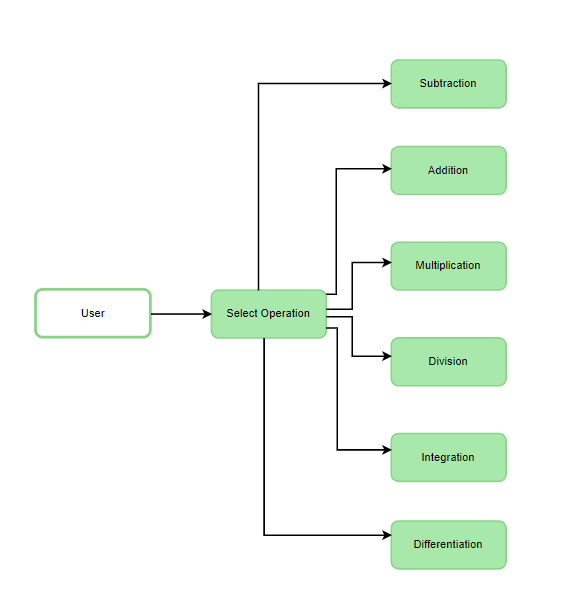
A correct polynomial should look like:

aX^b + cX^(b-1) +… - dX^(b-n)

!! The program will order the polynomials given by the user from the one with the highest power to the one with the lowest one, and, also, will add together the monomials with the same power.

In the case in which the input is incorrect, when the user selects an operation to be performed on it, the Result field will display an error message that looks like “Wrong input, please try again!”.

2.4 Use Cases



The use case and reason to use the polynomial calculator is fairly obvious:

The user wants an easy and reliable way to perform operations on polynomials, that is also faster than he could perform the operations on paper. The polynomial calculator does all these jobs well.

1. **Design (design decisions, UML Diagrams, data structures, class design, interfaces, relationship packages, algorithms, user interface)**

3.1 Design decisions

The idea I started with: Have a polynomial calculator that is easy to use and performs the operations correctly.

The User Interface should not feel janky and should be user friendly.

The User Interface is resizable and, also has some sliders to hide/show/increase the area of different buttons or fields. The UI includes buttons for all the needed numbers/characters for writing a correct polynomial and a button for clearing the input. The UI also supports entering the polynomials with the keyboard.

Designs related to the code itself, not required by the user

The design of the App is based on the classical Model-View-Controller structure

This model helps dividing the App into smaller packages/classes that are easy to work with and have a well-designed and clear purpose. This also makes searching for problems/improving on some aspects of the code a lot easier.

(I will discuss these aspects in the following points of the document)

3.2 UML Diagrams

The polynomial class cannot exist without the existence of the monomial.

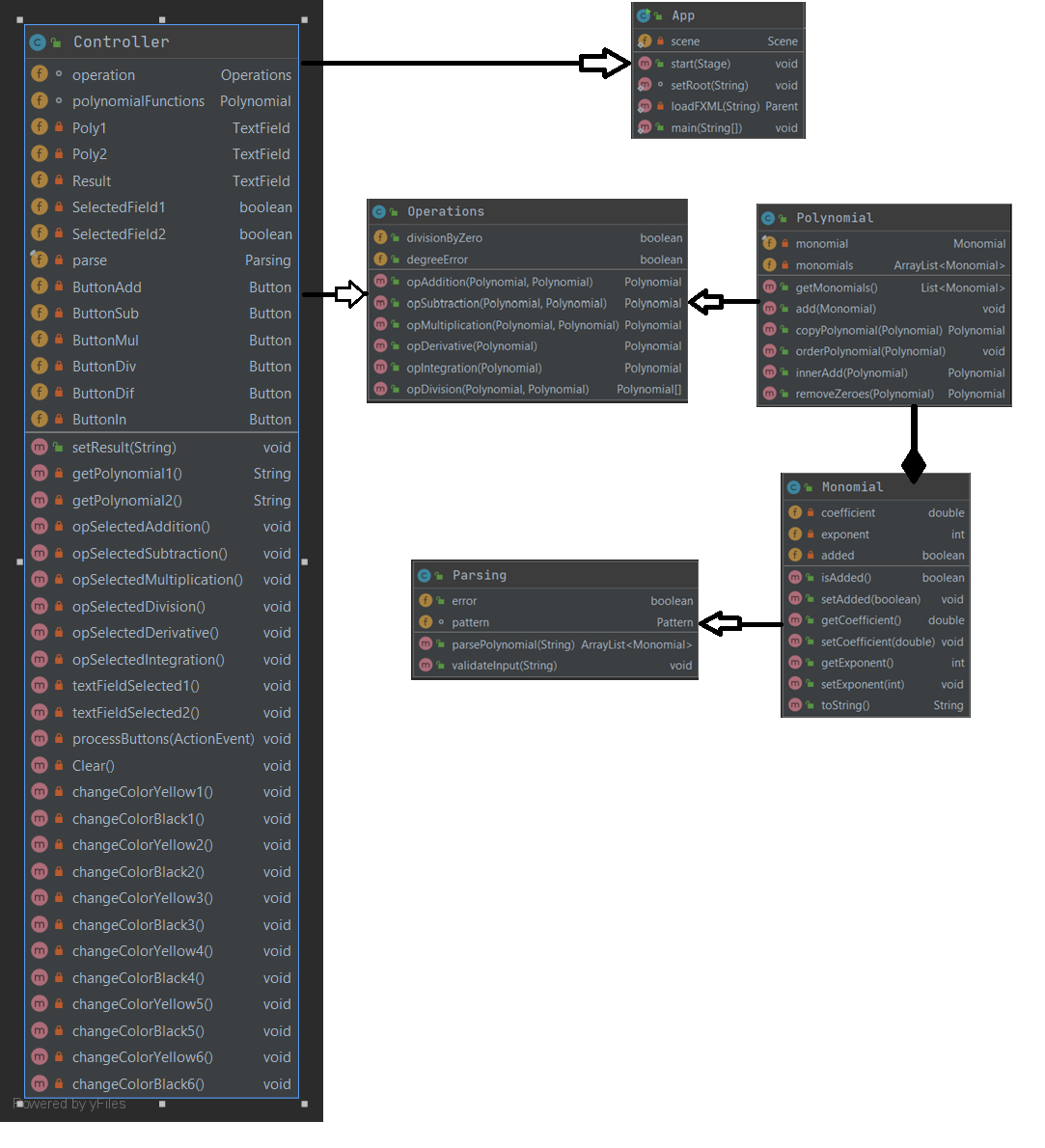
(The polynomial is defined as a list of monomials).

The operations from the Operations class are performed on Polynomials.

The UI is initialized in the App class (the main class).

The parsing method is used to parse the input from the user into monomials (defined by a coefficient and an exponent).

The Controller has the task of receiving the events from the User Interface and interpret them into what the program should do next.



3.3 Data Structures/data types

Important concepts for this program:

!! The polynomial is seen as an ArrayList of monomials. Using forEach, we can iterate through all the monomials of the polynomial and perform operations on them.

Some problems I encountered with this ArrayList implementation and I fall back to the classical iteration method: Deleting a monomial from the polynomial, at a position that is not well known (This would result in a memory error because the ForEach would try to iterate through monomials that are not existent (I had a forEach in a forEach)).

The degree of a polynomial is the exponent of the highest value of the polynomial.

Each monomial will have a coefficient and an exponent.

Coefficients are of type double (ex: 3.33). This type is required for operations like division and integration.

Exponents are considered to be integers.

Some extra data types:

The text fields from the user interface are interpreted in the code as some variables that memorize the input as a string.

A Boolean data type is used to store information about the state of the monomial (if it was used for an operation or not).

3.4 Class Design, Interfaces, Packages

The idea I started with here:

Have Classes that are compact, well organized, that have methods that perform specific operations and are named accordingly.

The packages for the App are the following:

The main package that includes all the source files of the project.

The Controller package that includes the class Controller.

The Model package that includes classes that perform specific tasks like operations/parsing data etc.

The View package that includes the User Interface.

The final design ended up with these final classes: Controller, Monomial, Polynomial, Operations, Parsing, App and the UI file in .fxml format that contains the characteristics for the user interface.

The main ideas of the classes are the following (I will delve deeper into the code, methods and algorithms in the next chapters)

In the Controller class, I make the link between the back-end and the front-end. The methods from the controller class are called when the user acts on the User Interface (Ex: The user presses the addition button - > the method opAdditionSelected is called in the Controller, and the logic of the addition is called from another class named Operations).

The Controller also has methods for writing and reading data on/from the text fields in the User Interface.

The UI.fxml file contains information about the design of the user interface and the name of the methods that are linked to the methods in the Controller class.

The Operations class contains the methods for the required operations: Addition, Subtraction, Multiplication, Division, Differentiation and Integration.

The Parsing class contains the methods needed for parsing and interpreting the data that is read from the text fields. This class uses regex (pattern matching/creating) in order to separate the input into groups that contain information needed to create the polynomials inside the program’s memory. (we store the polynomial as a list of monomials, each having a coefficient and an exponent)

The Monomial class contains the characteristics of a monomial and a method that converts the monomial back into a String that can be displayed and read by the user.

The Polynomial class contains the characteristics of a polynomial (A polynomial is an ArrayList of monomials) and also some methods that order the monomials in the right order (from highest exponent to lower exponent) and adding the monomials that have the same exponent.

The App class contains the initialization of the App. When the program is run, tis is the first code that is executed.

3.5 Algorithms

For this chapter I will describe some of the algorithms that I consider important, in my own words and a bit of pseudo-code because it is a lot clearer and easier to follow. (The actual code can be seen in the source files).

Addition:

Each monomial of the 1st polynomial is compared with each monomial of the 2nd polynomial.

If they have the same exponent, their coefficients are added together, and the exponent of the result remains the same. (after we do this operation, we set the value of isAdded to TRUE)

With the new coefficient and exponent, I create a new monomial that is added to the result polynomial.

All that is left to do now is add to the result all the monomials from the 1st and 2nd polynomial if they haven’t been added already in the previous step (if the isAdded of the monomial is still FALSE)

Subtraction:

This algorithm is basically just an addition with the + and – signs of the 2nd polynomial inversed.

The changes made to the addition algorithm are:

Instead of adding the terms with the same exponent, I subtract the 2nd one from the 1st one.

The monomials from the 2nd polynomial that are not used at the previous step, will be added to the result with opposite sign.

These algorithms were easy to implement because I also store the sign of each monomial and that makes it easy to inverse them for the subtraction.

Multiplication:

For the multiplication operation we have to multiply every monomial of the first polynomial with every monomial of the second polynomial.

The coefficient of the result monomial will be the result of the multiplication between the coefficient of the 1st monomial and the coefficient of the 2nd monomial.

The exponent of the result will be the result of the sum between the exponent of the 1st monomial and the exponent of the 2nd monomial.

After we compute the monomials of the result, we iterate through all of them and add together the ones with the same exponent, in order to have a result that looks like a correct polynomial.

Derivation:

For the derivation, we iterate through all the monomials of the given polynomial and multiply the coefficient with the current exponent and then the new exponent of the monomial will be (current exponent -1).

Integration:

For the integration, we iterate through all the monomials of the given polynomial and divide the current coefficient with (current exponent +1) and the new exponent will be increased by 1 to obtain the new exponent.

The result of the division may not be an integer.

Division:

The division is a more complex operation. The procedure is specified in the documentation given for this app.

To divide two polynomials *P* and *Q*, the following steps should be performed:

Step 1 - Order the monomials of the two polynomials *P* and *Q* in descending order according to their degree.

Step 2 - Divide the polynomial with the highest degree to the other polynomial having a lower degree (let’s consider that *P* has the highest degree)

Step 3 – Divide the first monomial of *P* to the first monomial of *Q* and obtain the first term of the quotient

Step 4 – Multiply the quotient with *Q* and subtract the result of the multiplication from *P* obtaining the remainder of the division

Step 5 – Repeat the procedure from step 2 considering the remainder as the new dividend of the division, until the degree of the remainder is lower than *Q*.

Some other useful algorithms/methods I used:

toString:

Takes a monomial stored in the program memory and transforms it into a string form that can be printed in order to display a correct polynomial for the user.

Special cases for this method: coefficient 0, coefficient 1, exponent 0, exponent 1, coefficient <0

parsePolynomial:

This method uses a pattern to create groups and isolate parts of the input in order to parse the data into the information we need to create correct monomials into our program memory.

The regex pattern used was inspired by a forum (link in the Bibliography)

(?:\\h\*)([-+]?\\d\*)(?:\\h\*)[x](\\^(\\d+))?|(?:\\h\*)([-+]?\\d+)

orderPolynomial:

This method will take a polynomial and order it’s monomials depending on their exponent.

If (monomial1.getExponent() > monomial2.getExponent()) -> swap

This method is extremely useful for displaying the result in the classical way a polynomial is written (from highest exponent to lowest exponent)

innerAdd:

This method will take a polynomial and will add together it’s monomials if they have the same exponent.

I call this method on all the inputs just to make sure the polynomial is stored correctly in the memory, and also on the result of the multiplication because it could give as a result more monomials with the same exponent.

This method is extremely important because without it, my code wouldn’t work on inputs like (5x + 5x +5x +5x) \* (5x + 5x), but after the call of the method, the operation would be performed correctly.

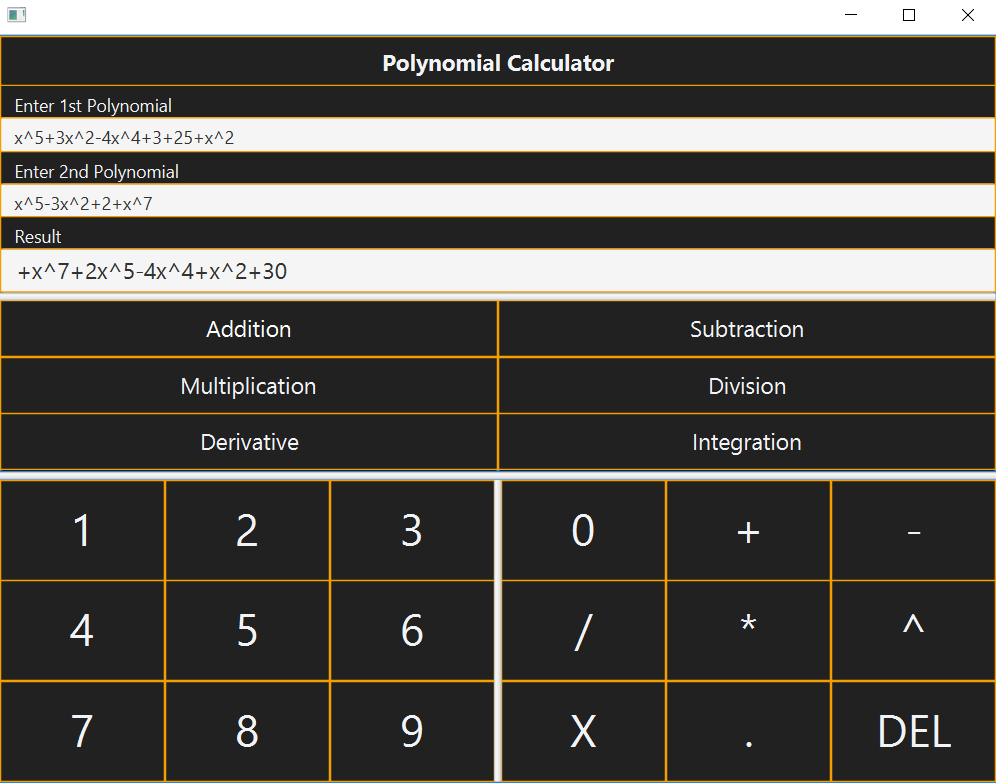
After innerAdd -> 20x \* 10x.

removeZeroes:

Iterate through the monomials of a polynomial and remove the ones with coefficient 0.

If the only monomial in the list is 0, it should not be deleted.

3.6 User Interface

For creating the user interface, I used Scene Builder (link in Bibliography) and also some details I added with code (like the change of color when pressing a button).

The UI is implemented with javafx.

The User Interface should be simple and accessible for all users.

The UI includes buttons the user could use to input data in the text fields, but also accepts input directly from the keyboard.

The UI calls the operation to be executed when the user presses the button for the operation (dynamical).

The input polynomials are given in a text field and the result or the possible error messages are displayed on another field: Result field.

1. **Implementation**

The Implementation and decision I made are already presented in the previous chapters (the description of the classes and methods and User Interface), so I will summarize the process of creating the app here.

The first part of the App that I created was the User interface. For this I used Scene Builder, a very useful tool for creating interfaces in javafx. (I detailed this process at the user interface chapter).

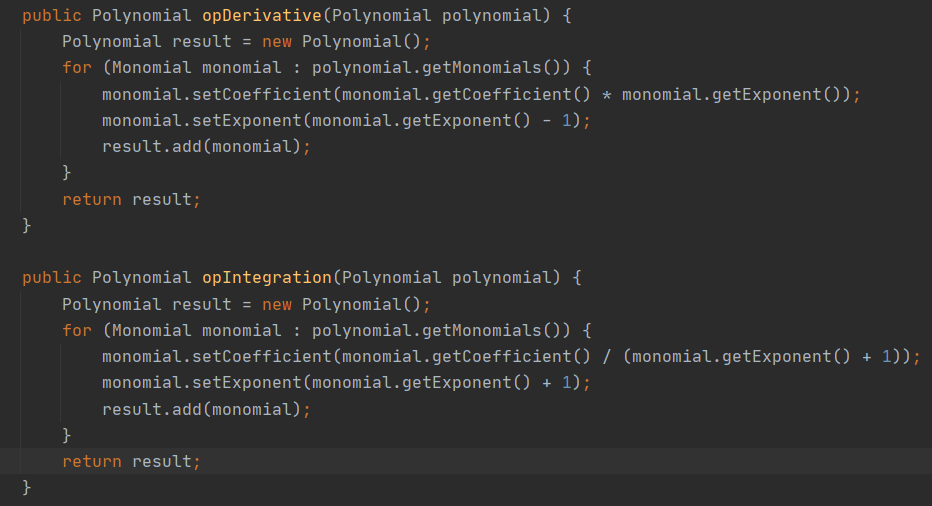
The next part of the project was the controller that contains what happens when the UI is interacted with. I defined some methods and some temporary bodies for them that included the flow of the operations. (Ex: opAdditionSelected-> read polynomial1&2 ->parse the data in order to create monomials and polynomials inside the code-> order and add together the monomials depending on the exponents -> perform the actual addition of the polynomials -> display the result)

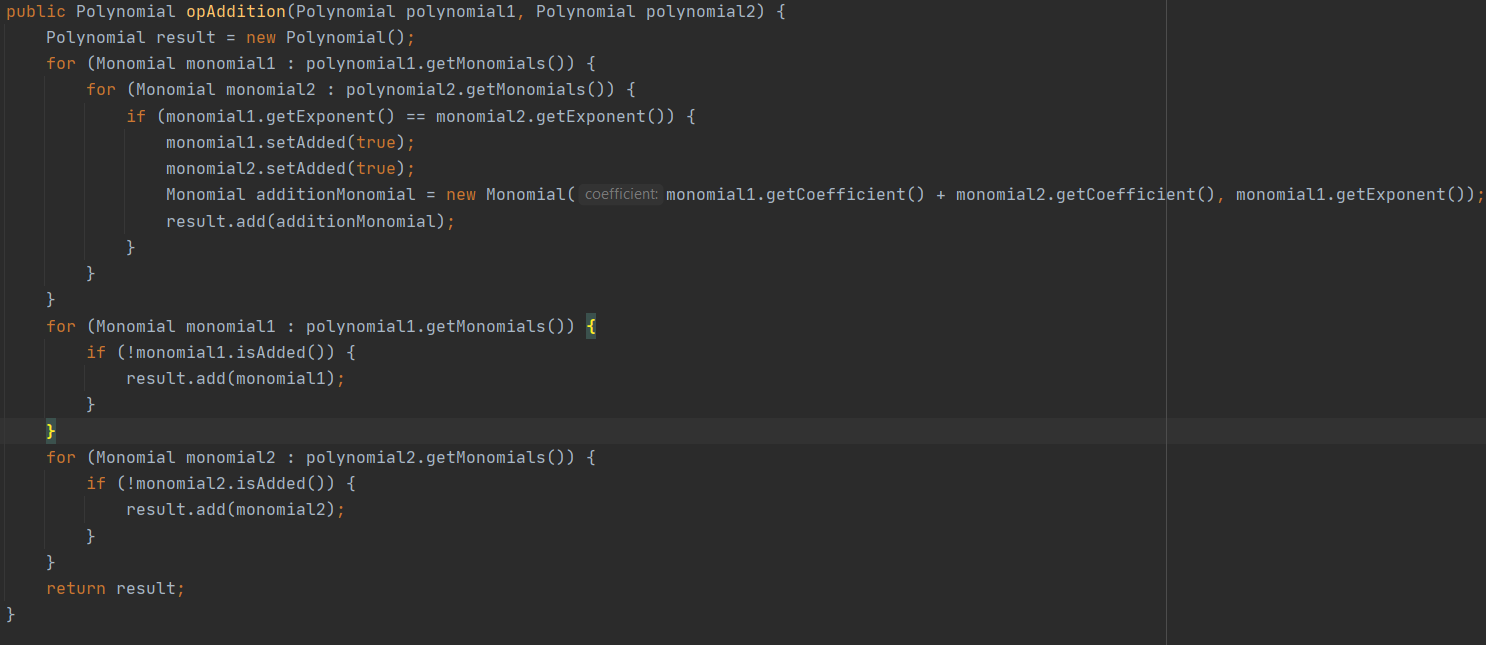
Based on this flow, I started creating the methods that I needed.

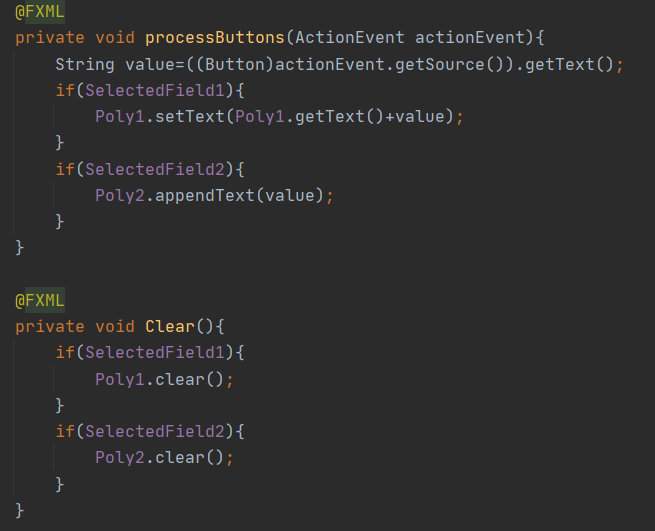
I created the Monomial and Polynomial classes, then I created a class containing the code for parsing the input from the user, then I started implementing the algorithms for the operations.

The final step is testing the whole App. (Unit testing and using the app like a normal user).

Some of the main methods are (the code is explained in one of the previous sections)



****

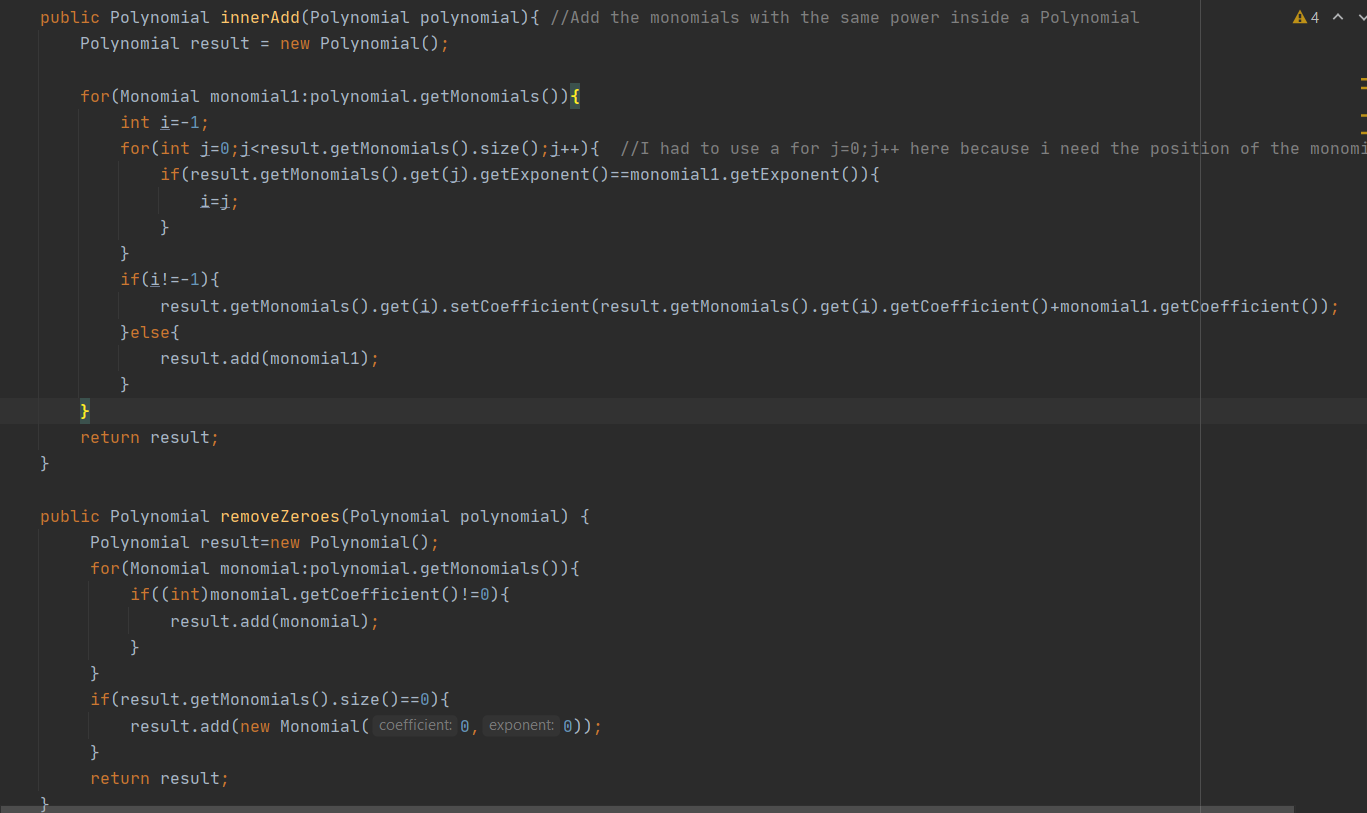
****

Here I added a function to use for all the buttons that input data on the textFields.

When A button is pressed, I check which text field is selected and I write the text on the button on top of the text field.



This method is used to change the colors of a button when it is pressed and revert it back to normal when it is released. This gives the feeling of pressing the button.



These 2 methods are important for organizing the polynomials in such a manner that the operations can be performed.

innerAdd, removeZeroes and orderPolynomial allow the program to give a correct output even if the polynomials are given in a wrong order or the same exponent is repeating in different polynomials.

1. **Results**

The testing of the App was done in 2 ways: Trying to use the app like a normal user and see the results for different inputs and operations.

By using Junit for testing (we test different methods by giving them an input and check if the output is the same as the expected one (also declared by us)).

I added 2 Junit tests, for some random cases, for each of the operations.

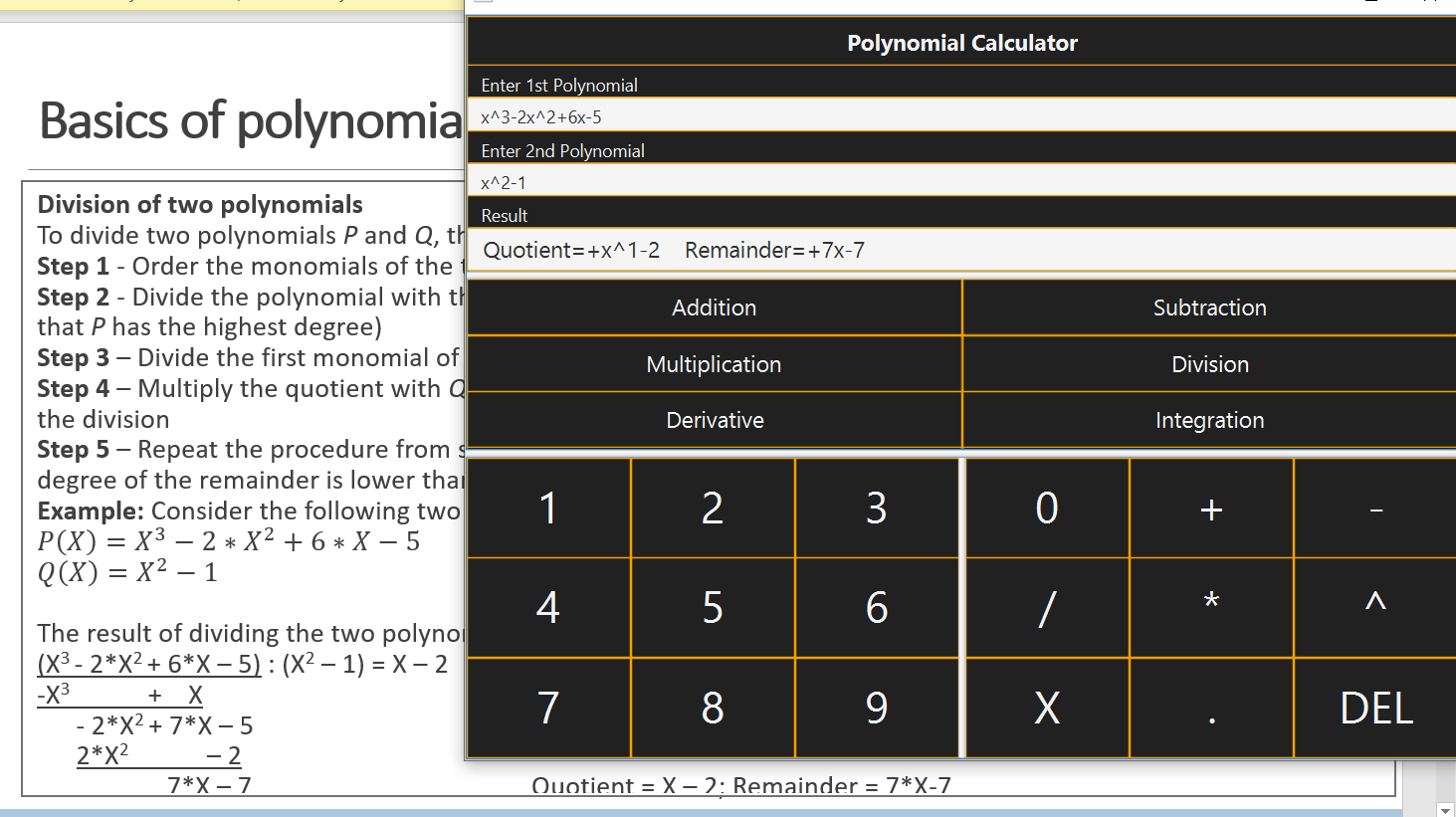
The main method I used was assertEquals(expected, actual) which returns true if the operation result is the same as the one we expect (given from the keyboard).

I also tested my other methods with 1 test to see if they give the expected result.

The tests for division showed that not all cases gave the expected result. ☹







1. **Conclusions**

This project was a great learning experience for me, because I worked with a lot of new information and features.

This was the first time I used: Maven for creating a project

Junit for testing

Javafx for creating the user interface

Regex for pattern matching/finding/parsing input

I would try to add some extra operations to the calculator like rising to a power, square root etc.

I also found out that I enjoy designing and working with User interfaces.

1. **Bibliography**

--The documentation for the project, and the course

--Stack Overflow for Regex Pattern, also tips for operations on polynomials

[Polynomial Java Array add subtract - Stack Overflow](https://stackoverflow.com/questions/42198612/polynomial-java-array-add-subtract)

[Java regex, separate degree & coeff of polynomial - Stack Overflow](https://stackoverflow.com/questions/28859919/java-regex-separate-degree-coeff-of-polynomial/44188848)

--Scene Builder for creating UI

[Scene Builder - Gluon (gluonhq.com)](https://gluonhq.com/products/scene-builder/)

--Regex 101 for testing, editing regex patterns

[regex101: build, test, and debug regex](https://regex101.com/)

--Youtube video about polynomial division

[Java for Scientific Computing: Polynomial Division - YouTube](https://www.youtube.com/watch?v=i_Qn-d6WExs&ab_channel=JavaforEngineers)

--For maven information

[Maven – Welcome to Apache Maven](https://maven.apache.org/)

--javaFX

[JavaFX (openjfx.io)](https://openjfx.io/)

--Junit guide

[JUnit 5 User Guide](https://junit.org/junit5/docs/current/user-guide/)