DOCUMENTATION

HOMEWORK *1*

SUDENT NAME: SANDU MIHAI-ALEXANDRU

GROUP: 30423

# CUPRINS

[1. Homework objective 3](#_Toc95297885)

[2. Problem analysis, modeling, scenarios, use cases 3](#_Toc95297886)

[3. Design 3](#_Toc95297887)

[4. Implementation 3](#_Toc95297888)

[5. Results 3](#_Toc95297889)

[6. Conclusion 3](#_Toc95297890)

[7. Bibliography 3](#_Toc95297891)

# Homework objective

# The main objective of this homework is to implement a Polynomial Calculator along with 6 operations:

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Derivation
6. Integration

The secondary objectives of this homework are:

* Creating a graphical user interface (GUI) in Swing or JavaFX, I chose JavaFX. Will be further described in the Design category.
* Read the input from the user using regular expressions and pattern matching. Will be further described in the Design category.
* Usage of a MVC architecture of the project. Will be further described in the Design category.
* Test the application. Will be further described in the Implementation category.

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

To use the Polynomial Calculator, the user must introduce the input in a specific format. Because I am using regular expressions and pattern matching, so that the polynomial can be correctly read and interpreted by the application, it must have the following format:

* First polynomial: (+-)x^n1(+-)x^n2(+-)x^n3….(+-)C
* Second polynomial: (+-)x^n1(+-)x^n2(+-)x^n3….(+-)C

Where (+-) represent the sign of the monomial. Even the starting monomial, although it has a positive coefficient needs to have a sign. For example, the following input is wrong: x^2-x+5.

The numbers n1, n2, … represent the powers of the monomials

The C represent the coefficient for the monomial which has the power equal to 0.

The user also needs to introduce a polynomial where the powers of each monomial are different. For example, the following input is wrong: +x^2-x^2+x-3.

How to use the Polynomial Calculator:

* Introduce the polynomials that you wish to apply one of the six operations in their specific fields: first polynomial, second polynomial
* Press one of the six buttons corresponding to the six operations in order to get the result in the “Result” field.
* For the “Integrate” and “Derive” operations, the user only needs to introduce the polynomial in the “First polynomial” field. Introducing the second polynomial, however will not result in an error. After the “Integrate” or “Derive” buttons were pressed, the user input in the field “Second polynomial” will simply be deleted
* If the user wishes to reset the result field and the polynomials fields, he must press the “Reset Polynomials” button

O imagine care conține masă

Descriere generată automat

1. **Design**

The application consists of the following Java classes:

* App – used to run the application, it simply runs the application
* HelloApplication – it loads the .fxml file (simply put the View) and it launches the application
* Controller – handles request flow. Used to describe what each button does when pressed
* Operations – used to implement each operation performed by the Calculator
* Monomial – it is part of the model package. The project uses a MVC architecture. It contains information like the coefficient and the power. It implements the Comparable Interface, to use Collections.sort to sort the Polynomial in the descending order of the powers for each monomial. It also has the equals method and hashCode.
* Polynomial – part of the model package. It consists of an ArrayList of Monomials and methods.
* PolynomialException – it extends the Exception class, used to throw an error when the Input is wrongly introduced.
* PolyUtil – it defines many methods:
  + monoFormatter - used to format the Monomial to be displayed
  + polyFormatter – used to format the Polynomial to be displayed
  + deriveMonomial
  + integrateMonomial
  + combinePoly – used to combine Monomials with identical powers
* hello-view.fxml – the View, fxml code generated by the Scene Builder Software, practically the implementation of the graphical user interface

In the Operations class and PolyUtil are described the algorithms used.

In Operations:

* addPoly(Polynomial firstPolynomial, Polynomial secondPolynomial): Polynomial

This method returns a Polynomial which consists of the sum of the 2 introduced Polynomials. First, a resulting Polynomial is declared, and the first Polynomial is copied into it. Then it iterates through each Monomial of the second Polynomial and adds the coefficients with the same power, of the Monomial from the second Polynomial. If the coefficient is 0, we remove that Monomial from the resulting Polynomial. Else, we modify its coefficient. If it doesn’t find a Monomial with the same power, we add that Monomial in the resulting Polynomial.

* subPolly(Polynomial firstPolynomial, Polynomial secondPolynomial): Polynomial

This method first inverts the signs of the Monomials from the secondPolynomial. Then it uses the addPoly method to add them.

* derivatePolynomial(Polynomial polynomial): void

It uses the deriveMonomial method from the PolyUtil class. It derives each Monomial. Then it iterates through the new Monomials ArrayList and add each Monomial with the coefficient equal to 0 in the polyToBeRemoved Polynomial. Then it deletes all the monomials in the polyToBeRemoved from the polynomial.

* integratePolynomial(Polynomial polynomial): void

It uses the integrateMonomial method from the PolyUtil class. It iterates through each Monomial and applies the integrateMonomial method to it.

* multiplyPolynomials(Polynomial firstPolynomial, Polynomial secondPolynomial): Polynomial

It uses the combinePoly method the the PolyUtils class. It multiplies each Monomial from the first Polynomial with all the Monomials from the second Polynomial and adds the resulting Monomial in the result Polynomial. Then it applies the polyCombine method to the result Polynomial.

* dividePolynomials(Polynomial firstPolynomial, Polynomial secondPolynomial): Polynomial[]

First, we assing a quotient and remined empty Polynomials. We find the Polynomial with the highest degree and assign it to pPolynomial and the other one to qPolynomial. Then we assign pPolynomial with the first Monomial from the pPolynomial, and qPolynomial with the first Monomial from the qPolynomial. We repeat the following algorithm until pMonomial power is lower than qMonomial power. We divide those Monomials and create a new Monomial with the power and the coefficient that we got earlier. We add the monomial to the intermediate Polynomial and quotient Polynomial. Then we multiply intermediate Polynomial with the qPolynomial and substract the result from the pPolynomial. We check if the pPolynomial is empty, if it is, we break the while loop. Else pMonomial becomes the new first monomial from the new pPolynomial. We repeat the loop if necessary. Return the quotient and the reminder.

In PolyUtil:

* monoFormatter(Monomial monomial, char type): String

It converts the monomial in a string to be displayed. First it checks if the type is ‘d’ for float or ‘a’ for integer. If float, String coefficient is equal to the float value converted in a string of the coefficient. Int in the other case. If the coefficient is 0, it returns and empty string. Then if the coefficient is equal to -1, it concatenates a “-“ to the resMono string. If the Monomial is 1, it concatenates the resMono the coefficient. I the coefficient is different from 1, it simply concatenates it to the resMono string. If the power is 0 and the coefficient is -1 it concatenates to the resMono string “1”. Else if the power is 1, it concatenates “x” to the resMono string. If its higher than 1, it concatenates “x^” + the power to the resMono string.

* polyFormatter(Polynomial polynomial, char type): String

it applies the monoFormatter method on every Monomial of the polynomial and concatenates the string returned by the method to resPolly string. It returns resPolly

* deriveMonomial(Monomial monomial): void

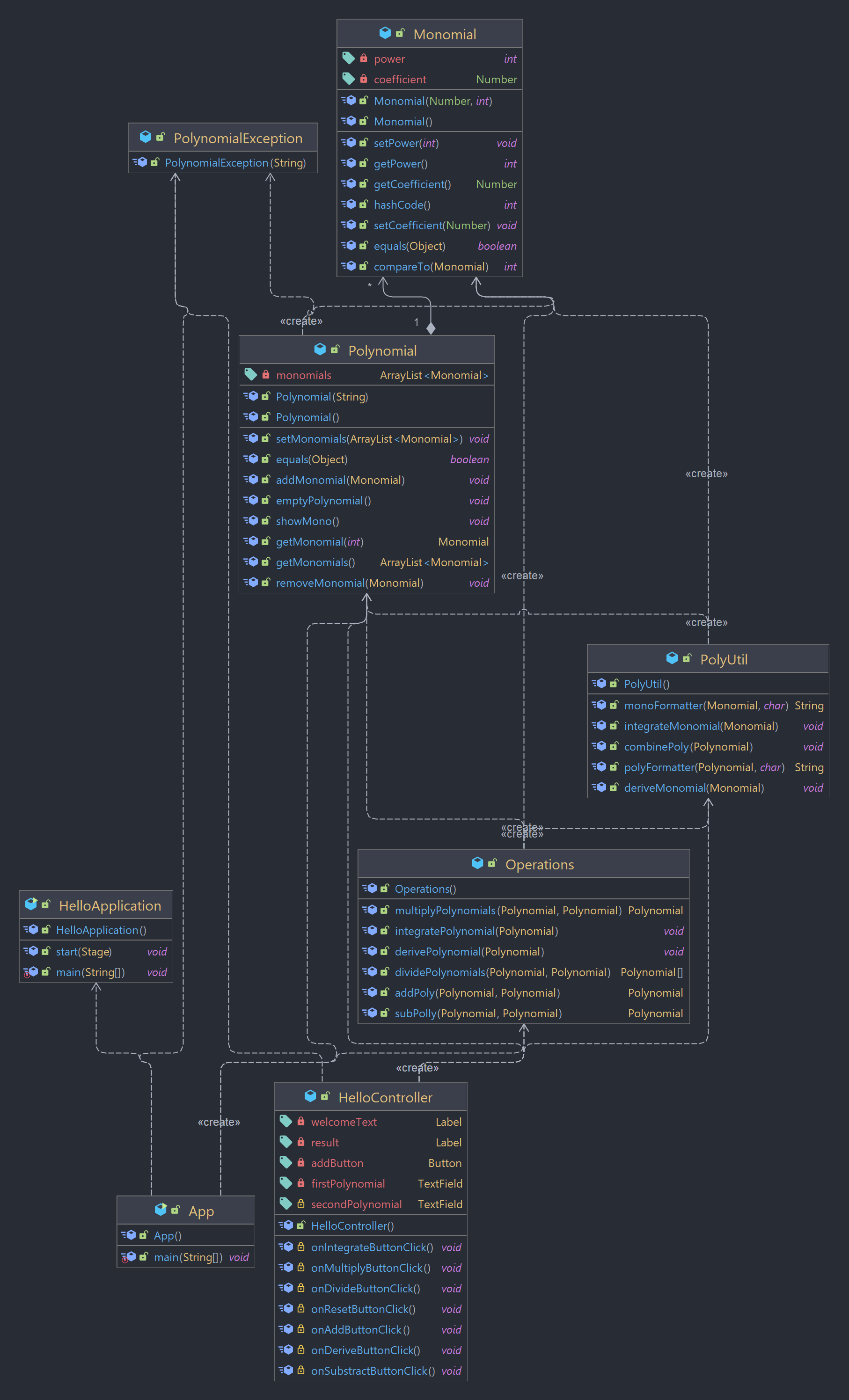
it applies the derivation formula (x^n)’ = (n+1)\*x^n-1

* integrateMonomial(Monomial monomial): void

it applies the integration formula

* combinePoly(Polynomial polynomial): void

it combines the Monomials with the same power into one Monomial. If the resulted Monomial is not in the toRemove Poynomial, it adds it to it. Then it empties the polynomial and adds the monomials from toRemove Polynomial to polynomial.



The application uses an object-oriented programming design bu using encapsulation, it defines appropriate classes as a result of problem decomposition such as Polynomial and Monomial.

It also makes use of decomposition, by breaking the Operations that need to be implemented into two classes. Operations and PolyUtil. Operations makes use of methods from PolyUtil, and it that way, the methods do not exceed the specified number of lines.

As for data structures, the only significant data structured used is the ArrayList<Monomials> as an argument of the Polynomial class, that is used to store the Monomials.

1. **Implementation**

**Implementation of the GUI**

For implementing the graphical user interface I used JavaFX combined with Scene Builder. First, I had to create a new JavaFX project. Then I had to install the Open JavaFX 17.0.2 and Java Scene Builder 17. To make use of the Scene Builder, I had to go to Setting -> Language & Frameworks -> JavaFX. There is a field “Path to SceneBuilder” in which I had to put the path of the Scene Builder install directory, more precise to SceneBuilder.exe. Then I had to add a new run configuration in which I had to choose the HelloApplication to Run. Then open Project Structure -> Libraries and add a new Library. There I had to add the javafx-sdk-17.0.2 lib folder. Then, modify the Run configuration, add VM options and write the following: --module-path "C:\Program Files\Java\javafx-sdk-17.0.2\lib" --add-modules javafx.controls,javafx.fxml . To finish it, go to Project Structure -> Artifacts -> Add -> JAR -> From modules with dependencies ->choose App -> Add -> File then add everything form the javafx-sdk-17.0.2 bin folder, all the files ending with .dll.

After this, I used the SceneBuilder software to create a graphical interface. The application generated fxml code into the hello-view.fxml file. In the HelloController class, I had to implement methods for each operation button. Then, in the SceneBuilder application, click on each button, then click on Code then in the On Action field, choose the specific method for each button. For the result labe, I had to assign it an ID to access it, from the SceneBuilder.

**Classes**

1. App – the main application class, it calls the HelloApplication.main method. Used to run the application
2. HelloApplication – loads the application graphical interface. It also sets the dimensions, the scene and the tittle. .show() method is used to display the application.
3. HelloController – it handles the user’s requests. This class has several important methods, 7 to be more specific, one for each button click. First, we need to declare the result Label, the firstPolynomial TextField, secondPolynomial TextField in order to access them. Methods:
   * 1. onAddButtonClick(): void, It initializes the Polynomial firstPolynomial, Polynomial SecondPolynomial with the values from the TextFields. Then it also uses the Operations class and PolyUtil class to display the resulted Polynomial. It simply calls the .addPoly method from Operations class and sets the Label result text to the resulted Polynomial converted into a string using PolyUtil .polyFormatter method.
     2. onSubstractButtonClick(): void , It initializes the Polynomial firstPolynomial, Polynomial SecondPolynomial with the values from the TextFields. Then it also uses the Operations class and PolyUtil class to display the resulted Polynomial. It simply calls the .subPoly method from Operations class and sets the Label result text to the resulted Polynomial converted into a string using PolyUtil .polyFormatter method.
     3. onDeriveButtonClick(): void , It initializes the Polynomial firstPolynomial, with the value from the TextField. Then it also uses the Operations class and PolyUtil class to display the resulted Polynomial. It simply calls the .derivePolynomial method from Operations class and sets the Label result text to the resulted Polynomial converted into a string using PolyUtil .polyFormatter method.
     4. onIntegrateButtonClick(): void , It initializes the Polynomial firstPolynomial, with the value from the TextField. Then it also uses the Operations class and PolyUtil class to display the resulted Polynomial. It simply calls the .integratePolynomial method from Operations class and sets the Label result text to the resulted Polynomial converted into a string using PolyUtil .polyFormatter method.
     5. onMultiplyButtonClick(): void , It initializes the Polynomial firstPolynomial, Polynomial SecondPolynomial with the values from the TextFields. Then it also uses the Operations class and PolyUtil class to display the resulted Polynomial. It simply calls the .multiplyPolynomials method from Operations class and sets the Label result text to the resulted Polynomial converted into a string using PolyUtil .polyFormatter method.
     6. onDivideButtonClick (): void , It initializes the Polynomial firstPolynomial, Polynomial SecondPolynomial with the values from the TextFields. Then it also uses the Operations class and PolyUtil class to display the resulted Polynomial. It simply calls the .dividePolynomials method from Operations class and sets the Label result text to the resulted Polynomial converted into a string using PolyUtil .polyFormatter method. It also checks if the Polynomials quotient and reminder are empty, in that case, it initializes the string with “0” so it can display them.
4. Operations – it implements all the operations for the Polynomial Calculator. All the methods in this class were described in the previous section, when explaining the algorithms.
5. Monomial class – it has two private arguments. Number coefficient and int power. It is used to store the data for each monomial.
6. Polynomial class – it has a single private argument. ArrayList<Monomials> monomials. It has the following methods:
   * 1. addMonomial(Monomial monomial): void - it adds the monomial to the monomials array list
     2. removeMonomial(Monomial monomial): void – it removes the monomial from the monomilas array list
     3. constructor: Polynomial(String inputPoly) – this is the most important method of the class. It takes a string and it converts it into data that is stored in the monomials array list. It uses two regexs expressions. First, it uses this: [+-](\d+)?(x(\^\d+)?)? to get each monomial. Then on each monomial it applies ([+-]\d\*)(x?\^?)(\d\*) to break it into 3 groups so it can easily access the string and transform them into integers.
     4. showMono() – displays in the console the polynomial
     5. @Override equals(Object o) – used for the testing part
7. PolynomialEception – it extend the Exception class, used to throw Exceptions when the input is not correctly introduced.
8. PolyUtil – all the methods from this class were explained in the previous section, when explaining the algorithms
9. **Results**

For testing the application, in the early stages of development I used the console and the input was hardcoded. Later, I used Junit for testing the application. Each operation has its own testing class, and 4-6 test cases. For the subtraction, addition, derivation, multiplication operations, input string for the polynomials were gives, then the corresponding operation was applied to it. The result was given as a string and transformed into a polynomial. Then, it checks if the polynomial after the operation was applied on it is equal to the result polynomial. For the integration and division cases, it takes the strings as input, transforms them into Polynomials, applies the operation then transforms it back into a string to be displayed. Those cases check if strings are equals, because in those cases, the coefficients can have float values.

1. **Conclusions**

I learned from this homework how to use JavaFX and SceneBuilder, how the Maven architecture works, how to work on a bigger project than most of the projects I worked before. My conclusion is that I am very pleased how it turned out, with only three days of work. Later on, the app can be developed so that it can take floats for coefficients for the input polynomials and not to need the + sign for the first monomial. The code can also be made a little bit easier to understand, mostly the formatter part, it was in some cases trial and error.

1. **Bibliography**

<https://dsrl.eu/courses/pt/materials/PT2021-2022_Assignment_1.pdf>

<https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf>

<https://www.youtube.com/watch?v=_j11AsUGpc8>

<https://www.baeldung.com/parameterized-tests-junit-5>

<https://www.baeldung.com/junit-5>

<https://www.youtube.com/watch?v=zvgWgpGZVKc>

https://www.youtube.com/watch?v=iik25wqIuFo