DOCUMENTATION

ASSIGNMENT 1

STUDENT NAME: Sîrca Florentina Raluca
GROUP: 30423

CONTENTS

1. Assignment Objective

The primary goal of this laboratory project was to propose, design and implement a system capable of processing polynomials of a single variable with real coefficients.

In order to achieve this primary objective, it was necessary to divide it into several sub-objectives, including:

- Collection of input data and produces output data once the calculations are complete;
- Creates an interface which was developed with careful attention to user experience, ensuring that it can be navigated with minimal difficulty and offers a wide range of useful features;
- allows users to manipulate two separate polynomials by addition;
- allows users to manipulate two separate polynomials by subtraction;
- allows users to manipulate two separate polynomials by multiplication;
- allows users to manipulate two separate polynomials by division;
- allows users to manipulate one polynomial by derivation;
- allows users to manipulate one polynomial by integration;
- incorporates a user-friendly graphical interface, designed to facilitate ease of use and comfort for all users;

2. Problem Analysis, Modeling, Scenarios, Use Cases

Problem Analysis

In Mathematics, a polynomial is a mathematical expression consisting of variables, powers and coefficients, combined by arithmetic operations such as addition, subtraction, multiplication, division and non-negative integer exponents of variables. Typically, the variables used in a polynomial are denoted by letters such as x or y. The coefficients, on the other hand, are constants that may be real numbers, complex numbers, or even elements of more general algebraic structures. The powers are, in general, integer numbers.

Polynomials can be expressed as a sum of monomials. A monomial is a term in a polynomial that consists of a single variable raised to a non-negative integer power multiplied by a coefficient. In general, a polynomial of degree n (the highest power of the variable in the polynomial) can be expressed as the sum of n+1 monomials, each with a different coefficient and a variable raised to a power ranging from 0 to n.

The general form is $a_n\, x^n + a_{n-1}\, x^{n-1} + \cdots + a_1\, x^1 + a_0\, x^0$, where n is the degree of the polynomial, and a_0 , a_1 , … , a_n are the coefficients.

Modelling the problem

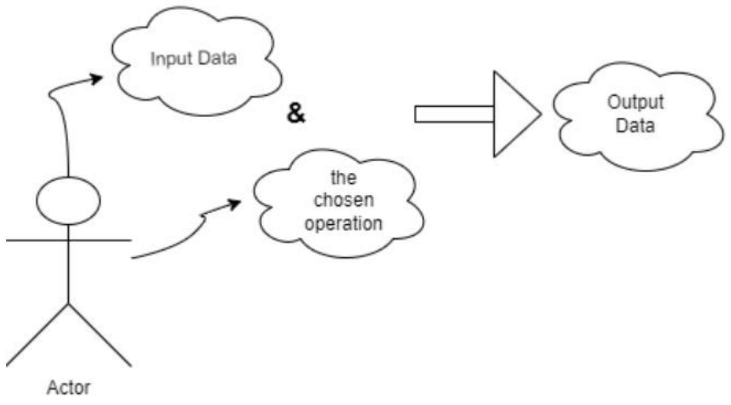
A first step of the modeling process is to look at the polynomials as a list of monomials, because with them it is much easier to operate, being characterized only by degree and coefficient.

Thus, in order to implement the desired operations at the polynomial level, they would have to be functional at the monomial level and expanded later.

These polynomials will then be fetched by the application and stored as polynomial model for further use. Once the polynomials are stored, the user can select a specific operation to be performed on them, such as:
addition of two polynomials, subtraction of two polynomials, multiplication of two polynomials, division of two polynomials, differentiation of a polynomial, integration of a polynomial.

The result of the chosen operation will be displayed in the interface. If the input is not a proper polynomial, the application will throw an error box.

Scenarios and use-case



The use case presents the actor, which in our case is the user that interacts with the application. She/ He can perform several actions on the two chosen polynomials, such as addition, subtraction, multiplication, division, integration and differentiation.

3. Design

Packages

Java packages facilitate the systematic arrangement of several modules by grouping interconnected classes and interfaces.

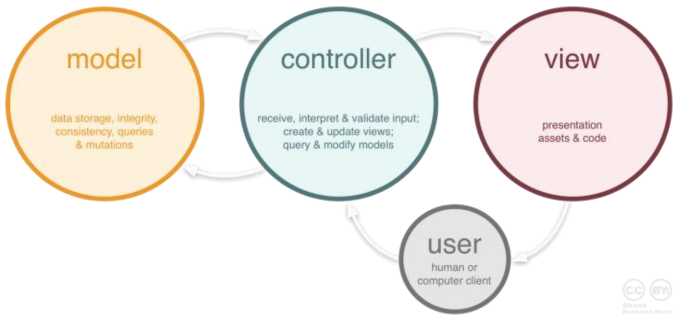
In the object-oriented programming development,, the **model-view-controller** (MVC) approach or design pattern is implemented to establish an effective and optimized connection between the user interface and the fundamental data models.

The model defines what data the app should contain. If the state of this data changes, then the model will usually notify the view (so the display can change as needed) and sometimes the controller (if different logic is needed to control the updated view).

The view defines how the app's data should be displayed, (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth).

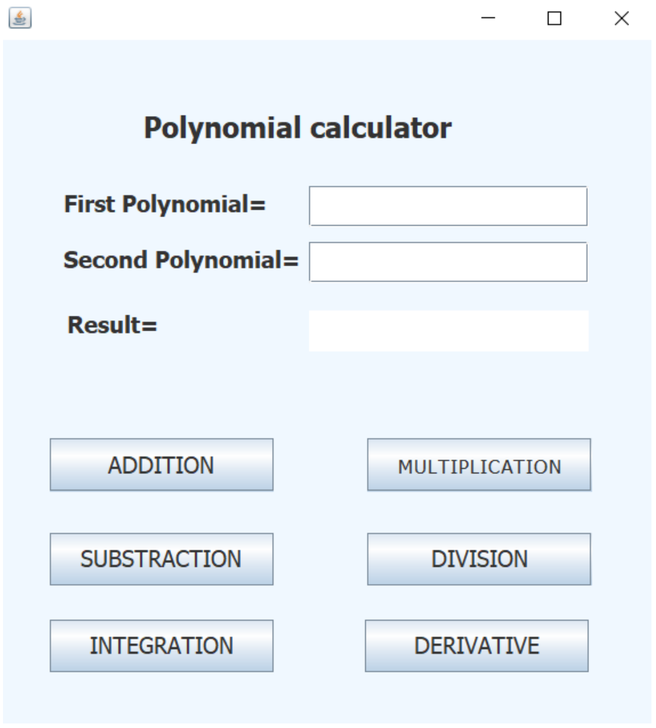
The controller contains logic that updates the model and/or view in response to input from the users of the app. It represents the classes connecting the model and the view.

- Models – contains the "brain" of the application, the classes which model the problem
    - Monomial
    - Polynomial
- Views – contains a single class which represents the GUI
- Controllers – it interconnects the model and the view


model
data storage, integrity, consistency, queries & mutations
controller
receive, interpret & validate input; create & update views; query & modify models
view
presentation assets & code
user
human or computer client

To utilize the calculator, the user must input two polynomials into their respective TextFields in the interface. The application will then retrieve the polynomials and generate them internally. When the user wishes to perform an operation, they simply need to press the corresponding button and the outcome will be displayed in the result TextField.

For some operations such as addition, subtraction, division or multiplication, the user has to choose which operation to be executed and both polynomials will be used, but when the user uses one of the operations such as integration or derivation, only the first polynomial will be used.



Used algorithms

The user will add the two polynomials, then depending on the button chosen, the following will be done:

- Addition: the sum of the two polynomials will be done by going through each polynomial to see the common powers to which the coefficients can be added, and if there are no common elements, the monomials will be added to the final result
- Substraction: the subtraction of the two polynomials will be done by going through each polynomial to see the common powers to which the coefficients can be subtracted, and if there are no common elements, the monomials from the first polynomial will be added to the final result, and from the second one will be subtracted
- Multiplication: the two polynomials will be multiplied term by term
- Division: we typically use long division method, which involves dividing the leading term of the dividend polynomial by the leading term of the divisor polynomial to obtain the first term of the quotient polynomial. Then, the resulting term is multiplied by the divisor polynomial, and the resulting polynomial is subtracted from the dividend polynomial to obtain a remainder polynomial. This process is repeated with the remainder polynomial, and the quotient polynomial is built by concatenating the terms obtained in each division step. The final result is a quotient polynomial and a remainder polynomial.

- Integration: only the first polynomial will be integrated; the integration of a polynomial is done by going through the monomials where the coefficient is divided by the power, and the power will increase by one unit;
- Derivation: only the first polynomial will be derived; the derivation of a polynomial is done by going through the monomials where the coefficient is multiplied by the power, and the power will decrease by one unit.

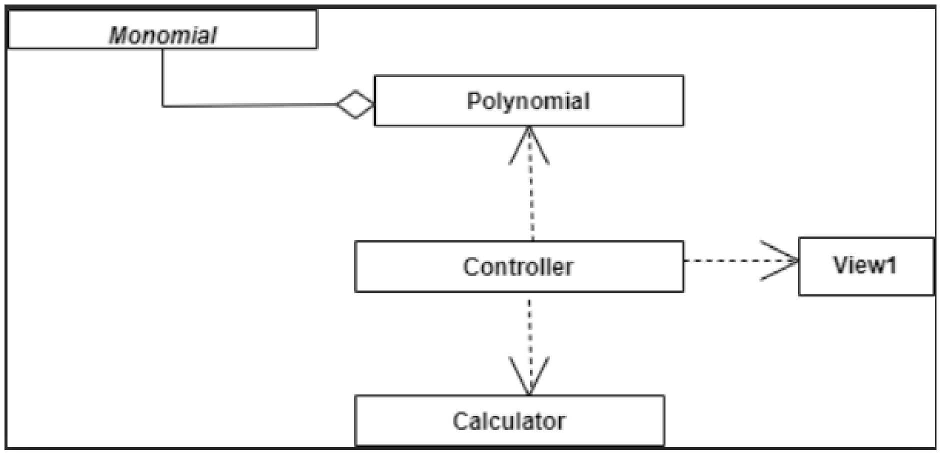
Data structures

The data structures with which I have been working in this problem are either primitive data types, especially integers and floats, and a more complex one, such as HashMap type object or new created object such as Monomial and Polynomial. Regarding this, I have decided to use HashMap because it is a data structure that offers several advantages over other data structures in certain contexts.

One of the primary benefits of using a HashMap is that it provides fast access to elements. This is because HashMaps are implemented using a hash table, which uses a hashing function to quickly retrieve the value associated with a given key.

In addition to their fast access times, HashMaps also allow for dynamic resizing, meaning that they can expand or contract in size as needed to accommodate additional elements or conserve memory. This is particularly useful when the number of elements in a collection is unpredictable or subject to change.

Another advantage of HashMaps is that they do not require that their elements be stored in a specific order. This is in contrast to other data structures like arrays, which must maintain the order of their elements. Because the order of elements is not important for certain problems, HashMaps can be a more efficient solution.



4. Implementation

Because I followed the MVC architecture, my program consists of 3 parts:

i) Models: contains the logic of the application

Monomial Class:

- A polynomial is composed by one or more terms, which in Mathematics are called monomials.
- This class has two instance variables, an int power and a float coefficient

Constructors:
+ *Monom (int power, float coefficint)* : the constructor that initializes the monomials with the transmitted coefficient and power
Methods:
+ int getPower() : returns the degree of the monomial
+ void setPower(int power) : allows us to set the degree of a monomial
+ float getCoefficient() : returns the coefficient of the monomial
+ void setCoefficient(float coefficient) : allows us to set the coefficient of the monomial

Polynomial Class:

- This class has an instance variable which consists of a HashMap<> class that is used to keep the monomials of this polynomial.

Constructors:
+ *Polinom()* : default constructor
Methods:
+ Map<Integer, Monomial> getPolynomial(): returns the map of the polynomial
+ void setPolynomial(Map<Integer, Monomial> polynomial): allows us to set the polynomial
+ void implementing(Monomial monomial): this method put the monomial in polynomial only if the coefficient is different from zero.
+ String toString(): this method transforms the polynomial result in string.

Calculator Class:

- This is the class where I implemented all the operations.

Methods:
+ Polynomial oppAdder(Polynomial pol1, Polynomial pol2)
+Polynomial oppSubtraction(Polynomial pol1, Polynomial pol2)
+Polynomial oppDerivation(Polynomial pol1)
+Polynomial oppIntegration(Polynomial pol1)
+Polynomial oppMultiplication(Polynomial pol1, Polynomial pol2)
+String oppDivision(Polynomial pol1, Polynomial pol2)
+int maxPower(Polynomial pol)
+float maxCoeff(Polynomial pol)

ii) Controllers: This is a very important class because it acts on both model and view. It controls the data flow into model object and updates the view whenever data changes.

Controller Class:

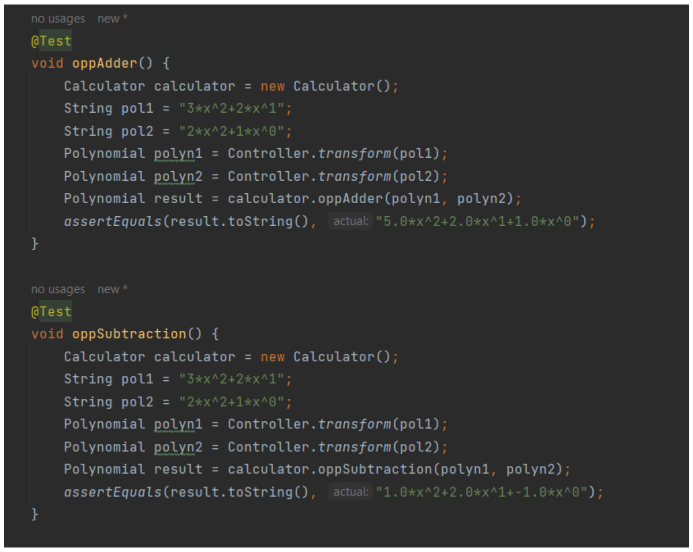
- this contains the linking between the model and the view;

Constructors:
+Controller(View1 view, Polynomial polynomial, Calculator calculator)
Methods:
+ *SubstractionButtonListener implements ActionListener*
+ *AdditionButtonListener implements ActionListener*
+ *IntegrationButtonListener implements ActionListener*
+ *DerivationButtonListener implements ActionListener*
+ *MultiplicationButtonListener implements ActionListener*
+ *DivisionButtonListener implements ActionListener*
+ *public Polynomial transform(String pol):* this method transforms the string in polynomial.

iii) <u>Views</u>– contains the graphical interface, the user interface. I decided to extend the JFrame class for this purpose because it simplifies the process of setting certain properties such as DefaultCloseOperation, size, visibility, and layout, as well as adding graphical components. The AbsoluteLayout was my preferred choice because it grants me the freedom to position my elements (such as labels, text fields, and buttons) wherever I desire, by specifying their individual bounds.

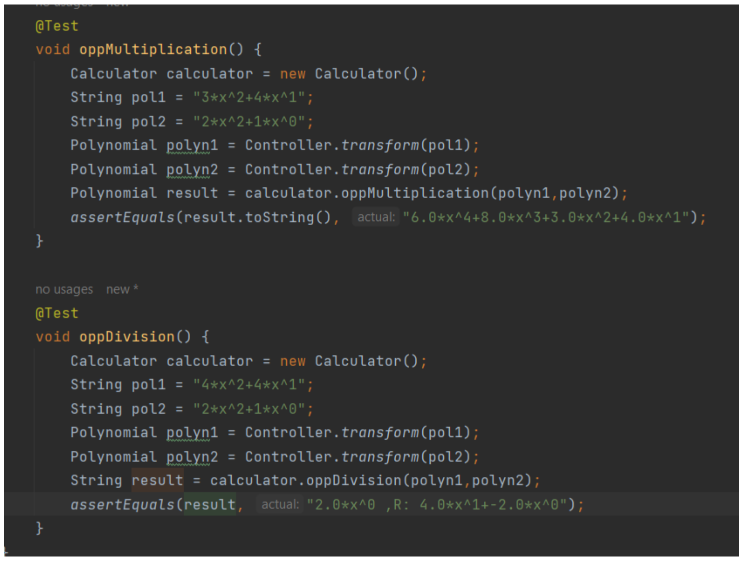
View1 Class
Constructors:
+View1()
Methods:
+void showErorrMessage(String message)
+void addAdditionButtonListener(ActionListener ActionListener)
+void addSubstractionButtonListener(ActionListener ActionListener)
+void addMultiplicationButtonListener(ActionListener ActionListener)
+void addDerivationButtonListener(ActionListener ActionListener)
+void addDivisionButtonListener(ActionListener ActionListener)
+void addIntegrationButtonListener(ActionListener ActionListener)
+gets and sets for buttons

5. Results



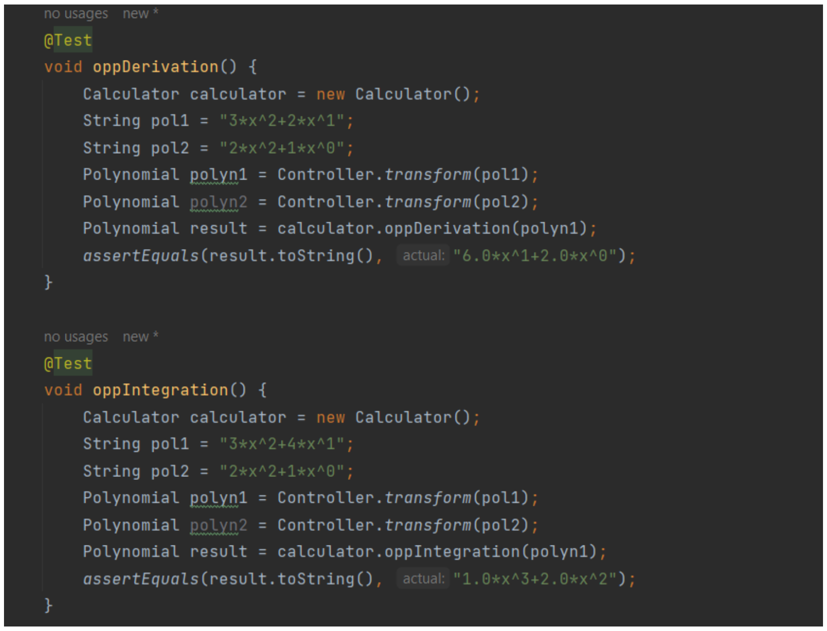
```
no usages   new *
@Test
void oppAdder() {
    Calculator calculator = new Calculator();
    String pol1 = "3*x^2+2*x^1";
    String pol2 = "2*x^2+1*x^0";
    Polynomial polyn1 = Controller.transform(pol1);
    Polynomial polyn2 = Controller.transform(pol2);
    Polynomial result = calculator.oppAdder(polyn1, polyn2);
    assertEquals(result.toString(),  actual: "5.0*x^2+2.0*x^1+1.0*x^0");
}

no usages   new *
@Test
void oppSubtraction() {
    Calculator calculator = new Calculator();
    String pol1 = "3*x^2+2*x^1";
    String pol2 = "2*x^2+1*x^0";
    Polynomial polyn1 = Controller.transform(pol1);
    Polynomial polyn2 = Controller.transform(pol2);
    Polynomial result = calculator.oppSubtraction(polyn1, polyn2);
    assertEquals(result.toString(),  actual: "1.0*x^2+2.0*x^1+-1.0*x^0");
}
```



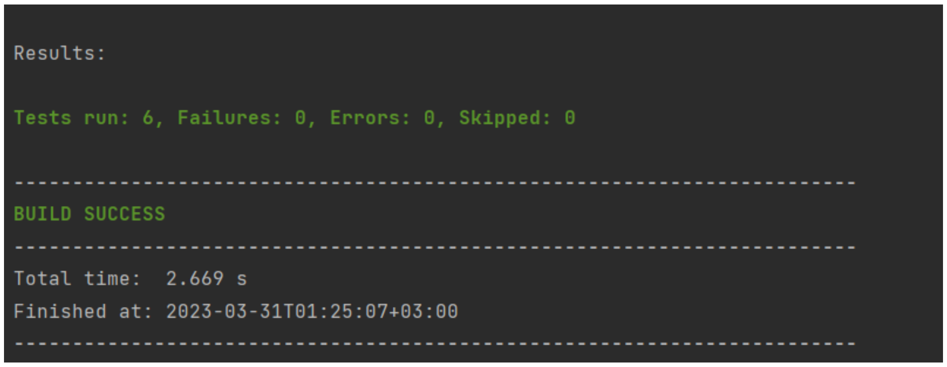
```
@Test
void oppMultiplication() {
    Calculator calculator = new Calculator();
    String pol1 = "3*x^2+4*x^1";
    String pol2 = "2*x^2+1*x^0";
    Polynomial polyn1 = Controller.transform(pol1);
    Polynomial polyn2 = Controller.transform(pol2);
    Polynomial result = calculator.oppMultiplication(polyn1,polyn2);
    assertEquals(result.toString(),  actual: "6.0*x^4+8.0*x^3+3.0*x^2+4.0*x^1");
}

no usages   new *
@Test
void oppDivision() {
    Calculator calculator = new Calculator();
    String pol1 = "4*x^2+4*x^1";
    String pol2 = "2*x^2+1*x^0";
    Polynomial polyn1 = Controller.transform(pol1);
    Polynomial polyn2 = Controller.transform(pol2);
    String result = calculator.oppDivision(polyn1,polyn2);
    assertEquals(result,  actual: "2.0*x^0 ,R: 4.0*x^1+-2.0*x^0");
}
```



```
no usages   new *
@Test
void oppDerivation() {
    Calculator calculator = new Calculator();
    String pol1 = "3*x^2+2*x^1";
    String pol2 = "2*x^2+1*x^0";
    Polynomial polyn1 = Controller.transform(pol1);
    Polynomial polyn2 = Controller.transform(pol2);
    Polynomial result = calculator.oppDerivation(polyn1);
    assertEquals(result.toString(),  actual: "6.0*x^1+2.0*x^0");
}

no usages   new *
@Test
void oppIntegration() {
    Calculator calculator = new Calculator();
    String pol1 = "3*x^2+4*x^1";
    String pol2 = "2*x^2+1*x^0";
    Polynomial polyn1 = Controller.transform(pol1);
    Polynomial polyn2 = Controller.transform(pol2);
    Polynomial result = calculator.oppIntegration(polyn1);
    assertEquals(result.toString(),  actual: "1.0*x^3+2.0*x^2");
}
```



```
Results:

Tests run: 6, Failures: 0, Errors: 0, Skipped: 0

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.669 s
Finished at: 2023-03-31T01:25:07+03:00
------------------------------------------------------------------------
```

6. Conclusions

Undertaking this project provided a valuable opportunity to recall and consolidate the object-oriented programming concepts previously learned in the first semester, while also acquiring new ones. At first, it was a challenging yet useful experience. Several key takeaways from this project come to mind.

First and foremost, time management plays a crucial role in project success. Maintaining an organized approach helps to break down tasks into manageable steps, allowing progress to be made steadily and effectively.

Secondly, proper problem modeling from the outset significantly accelerates the implementation process. It is essential to approach problem-solving with a well-defined methodology and a clear understanding of the problem domain.

Thirdly, when facing coding issues, it is beneficial to undertake independent research and experimentation to identify and resolve problems. This approach facilitates learning and a deeper understanding of both new and existing programming concepts.

Lastly, I discovered the importance of building interfaces from code, rather than relying on tools such as WindowBuilder. While WindowBuilder can appear simple and easy to use at first, it often creates more problems than it solves. This project enabled me to enhance my interface-building skills and develop greater confidence in this area.

7. Bibliography

1. *What are Java classes? -* *www.tutorialspoint.com*
2. *https://www.maa.org/external_archive/joma/Volume8/Kalman/Poly1.html*
3. *https://stackoverflow.com/questions/55427988/how-to-right-model-a-mvc-class-diagram-in-uml*
4. *https://en.wikipedia.org/wiki/Class_diagram*