**Documentation Homework 1**

**Polynomial Calculator**

**Name: Popa Adriana**

**Group: 30423**

**Laboratory assistant: Antal Marcel**

1. **Objectives**

* Main 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

1. **Dimensions of the problem**
2. **Problem analysis**

In mathematics, a polynomial is an expression consisting of variables (also called indeterminates) and coefficients, that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponentiation of variables. An example of a polynomial of a single indeterminate x is x2 − 4x + 7.

A representation of the polynomial can be done by creation pairs (ai,i), called monoms, where ai is the coefficient of x and I is the power.For example:

x2 − 4x + 7={(0,2),(-4,1).(7,0)}

This way of representing the polynomials will allow us to permorm the most common operations on the polynomials:  addition, subtraction, multiplication, division, differentiation, and integration.

1. **Modelling the problem**

The user will be able to see the result of the operations by introducing on the interface two polynomials. The polynomials will be introduces in a valid form, readable by the system and then, the user will select the operation to be performed:

* 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 will be displayed on the interface and the user can continue with the operations or he/she cand exit the program.

1. **Scenarios and use cases**

The scenario and the use cases are strongly connected to the user’s actions. All use cases and scenarios are similar due to the similitude of the process, I will shortly present one use case that includes all scenarios:

Use Case Name: Addition Operation

Actors:

* User (does not need registration, introduces the polynomials and selects the operation)
* Operations (executed by the system, not visible to the user)
* Result (written in the result text field in the user interface)

Triggers:

* The user indicates that he wants to execute the selected operation on the polynomials entered.

Preconditions:

* User has introduced the polynomials.
* User has selected the operation.

Post-conditions:

* The polynomials will be read by the system every time an operation is selected.
* The user has to introduce the polynomials in the correct form in order to obtain a correct result.
* The result will be shown to the user.

Normal Flow:

1. The user will introduce the polynomials he needs to do the operations on.
2. The system will read the polynomials.
3. The user will select the Addition operation.
4. The selected operation will be executed.
5. The result will be shown on the interface.
6. The user will introduce another set of polynomials in order to continue with this type of operation or
7. The user will select another operation or
8. The user will exit the system.

Alternate Flows:

1A1. The user introduces the polynomials in a wrong/unaccepted way.

1. He will need to reintroduce the polynpmials in the accepted/correct form
2. The system will continue with the operation.

2.A1 The user selects another operation.

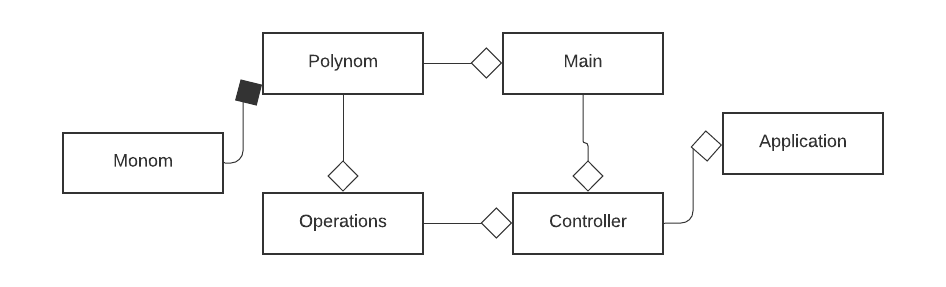
1. The selected operation will be executed instead.
2. The user can select the Addition operation after the result was displayed in order to obtain the needed result.

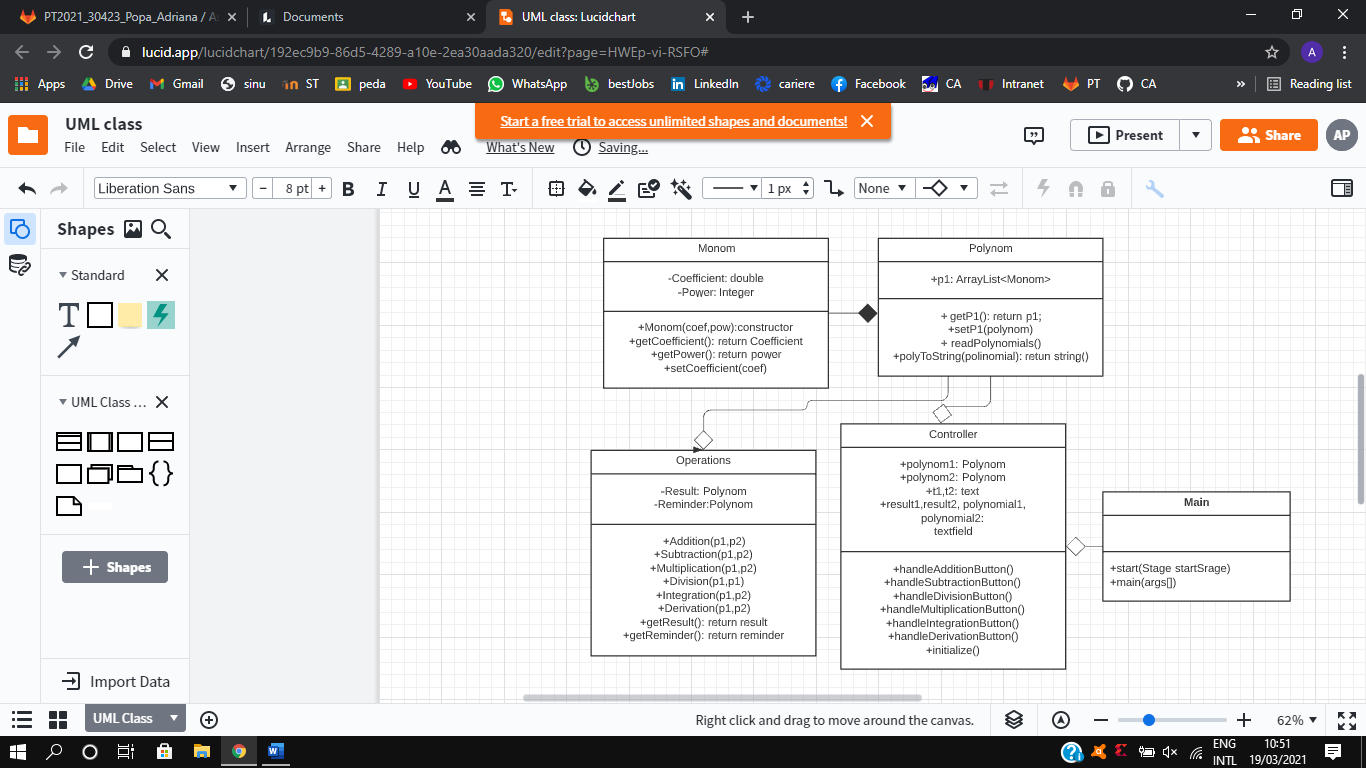
**3. Project design**

1. **Design decisions:**

I decided to implement the polynomial calculator in a simple manner, with only 4 needed classes and an easy-to-use interface, as presented in the assignment presentation.

1. **UML Diagram**





1. **Packages**

Java packages help in organizing multiple modules and group together related classes and interfaces.

In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently relating the user interface to underlying data models. The MVC pattern is widely used in program development with programming languages such as Java, Smalltalk, C, and C++.

The MVC pattern has been heralded by many developers as a useful pattern for the reuse of object code and a pattern that allows them to significantly reduce the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components or objects to be used in software development:

- *Model*, which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.

- *View*, which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)

- *Controller,* which represents the classes connecting the model and the view and is used to communicate between classes in the model and view.

My project is based on Model – View – Controller Pattern, so I splitted my classes into four

* Application – contains a single class, which contains the customary main() method
* Model –  contains the “brain” of the application, the classes which model the problem
  + Monom
  + Operations
  + Test
* View – contains a single class which represents the GUI
* Control – it interconnects the model and the view

1. **Algorithms**

* **Addition:**

The addition of two ordered polynomials is done by starting from the first position in each polynomials and

* + 1. Compare the power of the first monomial
    2. If a has the grather power, it will be added to the result array list

If b has the greater power, b will be added to the result

If a and b have the same power their coefficients will be added and introduced in the result.

* + 1. Repeat until a or b have no monomials
    2. If there are any remaining monomials in a or b, they will be added to the result
* **Subtraction:**

Uses the same algorithm as the addition but instead of adding the coefficients when the powers are equal, it subtracts them . Also, if b has remaining monomials their coefficients will became negative.

* **Multiplication:**

The multiplication is done by multiplying each monomial in the first polynomial with each monomial in the second polynomial. The hardest part in the multiplication is simplifying the monomials with the same power:

* + 1. Sort the array list
    2. Insert the first value of partial result in the final result.
    3. Delete the monomial from the partial result
    4. If there is any other monomial in the partial result with the same power add it and delete it
    5. Repeat until all the monomials in the partial result are deleted
* **Division:**

For polynomials, division is the most complex operation and I used the polynomial long division algorithm for this.

Polynomial long division is an algorithm that implements the [Euclidean division of polynomials](https://en.wikipedia.org/wiki/Euclidean_division_of_polynomials), which starting from two polynomials *A* (the *dividend*) and *B* (the *divisor*) produces, if *B* is not zero, a *quotient* *Q* and a *remainder* *R* such that

*A* = *BQ* + *R*,

and either *R* = 0 or the degree of *R* is lower than the degree of *B*. These conditions uniquely define *Q* and *R*, which means that *Q* and *R* do not depend on the method used to compute them.

The result *R* = 0 occurs [if and only if](https://en.wikipedia.org/wiki/If_and_only_if) the polynomial *A* has *B* as a [factor](https://en.wikipedia.org/wiki/Polynomial_factorization). Thus, long division is a means for testing whether one polynomial has another as a factor, and, if it does, for factoring it out. For example, if a [root](https://en.wikipedia.org/wiki/Root_of_a_polynomial) *r* of *A* is known, it can be factored out by dividing *A* by (*x* – *r*).

* + 1. Check is power a[0] is greater or equal to power b[0]
    2. If so, create a new monomial with coefficient equal to the division between their coefficients and power equal to the difference between their powers
    3. Create a new polynomial by multiplying b with the monomial
    4. Subtract it from a
    5. Keep the new resulted a in the reminder
    6. Repeat until power of b becomes greater than a’s power
* **Integration:**

For each monomial add 1 to the power and then divide the coefficient by the power and then.

* **Derivation:**

For each monomial multiply the coefficient with the power and then subtract 1 from the power.

If the power is 0, we drop that monomial.

**4. Implementation**

The problem is divided into smaller problems and then you solve these little, simple and well-known problems.

1. The model – contains the logic of the application

* Monom Class

A polynomial is composed by one or more terms, which in Mathematics are called monomials

This class has two instance variables, an integer power and a double coefficient.

Constructors:

*-public Monom (Double Coefficient, Integer Power)-*  the constructor that initializes the monomials with the transmitted coefficient and exponent

Getters and setters:

*-public int getPower() :* returns the degree of the monomial

*-public void setPower(int power) :* allows us to set the degree of a monomial

*-public double getCoefficient()* : returns the coefficient of the monomial

*-public void setCoefficient(double coefficient)* : allows us to set the coefficient of the monomial

* Polynom Class

This class contains the ArrayList<Monom> which is our polynom.

Constructors:

-*public Polynom()-*initializes the arraylist

Getters and setters:

*-public void setP1(Polynom p)*

-*public Polynom getP1():* returns our polynomial

Methods:

*-public void readPolynomial* (String s) – reads the polynomial using pattern matching into the pol.

*-public String polyToString*(*)* – transform the polynomial into a string and return it.

* Operations Class

This class contains Polynoms : result and reminder , storing the results of the operations.

The monomials are ordered in this list, from the highest to the lowest exponent of the polynomial, in the exact same order as they were introduced.

Constructors:

*-public Operations()* -initializes the array lists, they are empty when created.

Getters and setters:

-*public Polynom getResult()* - returns the array list containing the resulted monomials

*-public Polynom > getReminder()* – returns the array list containing the reminder from the division operation

Methods:

*-public void addition (Polynom a, Polynom b):* addition function

*-public void subtraction (Polynom a, Polynom b):* subtraction function -the order matters because it subtracts b from a.

*-public void multiplication (Polynom a, Polynom b):* multiplication function

*-public void division (Polynom a, Polynom b):* division function -the order matters because it divides a to b

*-public void integration (Polynom a):* integration function

*-public void derivation (Polynom a):* derivation function

* Test class

-void testAddition()

-void testSubtraction()

-void testMultiplication()

-void testDivision()

-void testIntegration()

-void testDerivation()

Each of this methods contains 3 cases for each operation.

1. The Control

* Controller class

This class controls initializes the interface and controls the actions of the buttons and text fields: it contains four text fields, two for the read polynomials and 2 for the results and texts for printing together with the result.

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

Methods:

*-public void initialize()* – initializes the interface hiding the unneeded elements

-public void handleAdditionButton() – when the addition button is pressed the method is executed and prints the result on the interface

*-public void handleSubtractionButton()*

*-public void handleMultiplicationButton()*

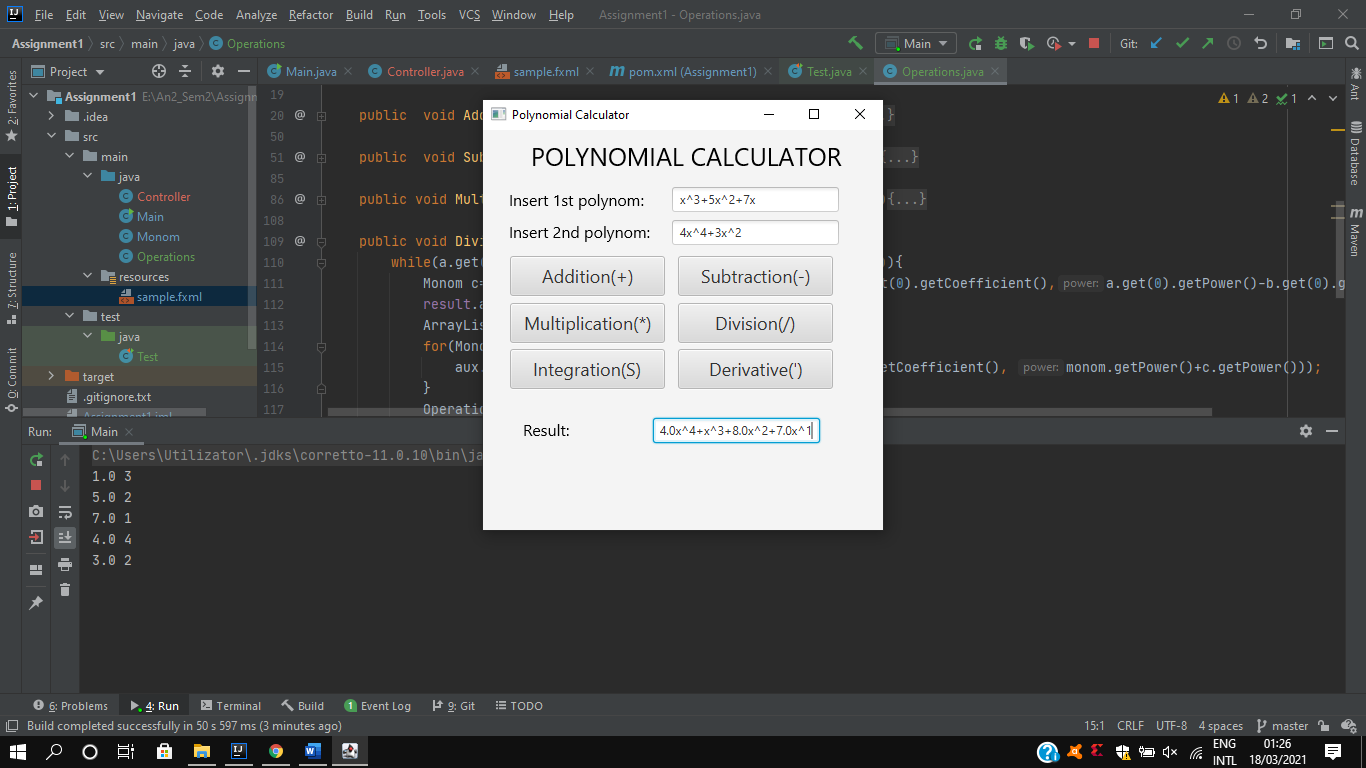
*-public void handleDerivationButton()*

*-public void handleIntegrationButton()*

*-public void handleDivisionButton()*

1. The view / User interface

The view was created using scene builder due to the easiness of designing and assigning the functions to the elements of the interface.

The user interface has a very simple design so that anyone can use it. The use just inserts the polynomial and then selects the operation he wants to perform, and the result will be displayed.

1. The Application - contains the Main class which runs the start() method from the Controller in order to “turn on” the application.

**5 .Results**

I did three tests for each operation and all of them were correct, but I will present only one for each operation:

-Addition: x^3+4x^2+5x^0 + x^3+2x^2+3x^1 = 2.0x^3+6.0x^2+3.0x^1+5.0

-Subtraction : x^3+4x^2+5x^0 = x^3+2x^2+3x^1 = 2.0x^2+-3.0x^1+5.0

-Multiplication: x^3+4x^2+5x^0 \* x^3+2x^2+3x^1 = x^6+6.0x^5+11.0x^4+17.0x^3+10.0x^2+15.0x^1

-Division: x^3+4x^2+5x^0 / x^3+2x^2+3x^1 = x^2+-1.0x^1+2.0 reminder : -3.0x^2+x^1+3.0

-Integration: x^5+3x^3+5x^0 => 0.17x^6+0.75x^4+5.0x^1

-Derivation: x^5+3x^3+5x^0 => 5.0x^4+9.0x^2

The scenarios were presented in the use cases, and as I already said they depend more on the user decision then the implementation of operations.

**6.Conclusions**

First of all, the project was a good opportunity to remember the OOP concepts learned last semester. I learned how to use maven and how to test my project using Junit. Also, it taught me that time management is very important and even if I had a very bad time management, I tried my best to finish this this project and for it to work properly.

Secondly, there are a few improvements, more like new operations that can be added to the project:

-square of a polynomial

-value at a given point

-finding the roots

-designing the graphic of the polynomial

-executing the operations on more than 2 polynomials at a time since it will be faster and easier for use in real life.

**7.Bibliography**

* <https://www.geeksforgeeks.org/>
* <https://whatis.techtarget.com/definition/model-view-controller-MVC>
* <https://stackoverflow.com/>
* <https://en.wikipedia.org/wiki/Polynomial_long_division>
* <https://developer.mozilla.org/>
* Object-Oriented Programming - Lecture Slides of prof. Marius JOLDOS
* Programming Techniques – Lectures of prof. Ioan SALOMIE