Technical University of Cluj-Napoca

Programming techniques

Laboratory assignment 1



**Polynomial Calculator**

Student: Sicobean Alexandra Maria

Group: 30422

**Index**

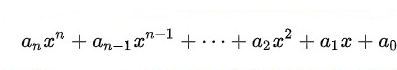
1. Objective
2. Analysis, scenarios, use cases
3. Design
4. Implementation
5. Results
6. Conclusions
7. Bibliography
8. **Objective**

The objective of the project is to design 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.

1. **Analysis, scenarios, use cases**

**Analysis of the problem:**

A polynomial is an expression consisting of variables and coefficients. They may appear in many areas of mathematics and science from word problems to complicated scientific problems. A polynomial with a single indeterminate x can be written as:



or

a0, a1, a2, …, an – constants

The following polynomial can be also perceived by a list of monomials: formed by a coefficient, an indeterminate and the power of the indeterminate.

Polynomials can be added and subtracted using the associative law of addition. They can also be multiplied, where the distributive law is repeatedly applied and every term is multiplied by every term of the order. The division of one polynomial is not typically a polynomial, therefore the result of the division of two polynomials will usually be a rational fraction. In addition to these basic operations, the Polynomial Calculator will include an option for integration and derivation.

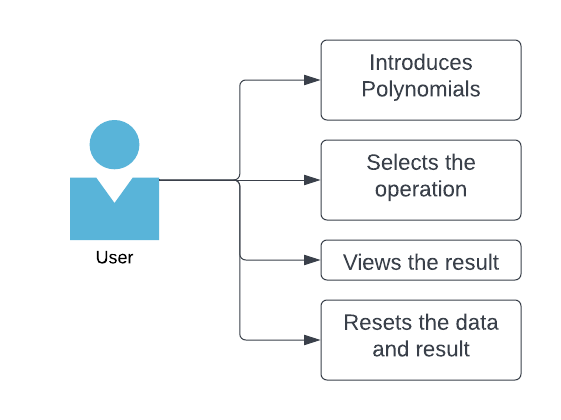
**Planning of the problem:**

The user will be asked to introduce two polynomials in the graphical interface and select the operation to be performed. Until the moment of the laboratory presentation the following operations have been implemented:

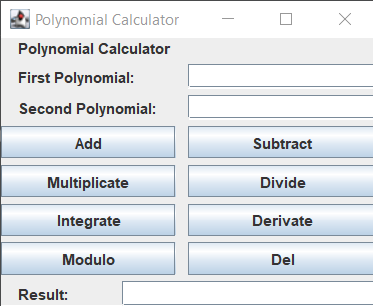
* Addition of two polynomials
* Subtraction of two polynomials
* Multiplication of two polynomials
* Differentiation of a polynomial
* Integration of a polynomial

The result of the selected operation will be displayed on a textbox on the bottom of the interface.

**Scenarios and use cases:**

****

The user is required to introduce two polynomials to be computed. After choosing the operation to be performed, two polynomials will be internally generated and computed according to the operation. The result will be immediately displayed on the GUI. For operations such as integration and derivative, only the first polynomial will be used.

****

**Use case 1:** addition

**Primary actor:** user

**Main success scenario:**

1. The user inserts two polynomials in the graphical user interface
2. The user selects the “addition” operation
3. The calculator performs the operation and displays it on the screen

**Alternative sequence:** error if not both polynomials are introduced

**Use case 2:** subtraction

**Primary actor:** user

**Main success scenario:**

1. The user inserts two polynomials in the graphical user interface
2. The user selects the “subtraction” operation
3. The calculator performs the operation and displays it on the screen

**Alternative sequence:** error if not both polynomials are introduced

**Use case 3:** multiplication

**Primary actor:** user

**Main success scenario:**

1. The user inserts two polynomials in the graphical user interface
2. The user selects the “multiplication” operation
3. The calculator performs the operation and displays it on the screen

**Alternative sequence:** error if not both polynomials are introduced

**Use case 4:** integration

**Primary actor:** user

**Main success scenario:**

1. The user inserts two polynomials in the graphical user interface
2. The user selects the “integration” operation
3. The calculator performs the operation and displays it on the screen

**Alternative sequence:** error if the first polynomial is not introduced

**Use case 5:** derivative

**Primary actor:** user

**Main success scenario:**

1. The user inserts two polynomials in the graphical user interface
2. The user selects the “derivative” operation
3. The calculator performs the operation and displays it on the screen

**Alternative sequence:** error if the first polynomial is not introduced

**Use case 6:** division

**Primary actor:** user

**Main success scenario:**

1. The user inserts two polynomials in the graphical user interface
2. The user selects the “division” operation
3. The calculator performs the operation and displays it on the screen without the modulo

**Alternative sequence:** error if not both polynomials are introduced

**Use case 7:** modulo

**Primary actor:** user

**Main success scenario:**

1. The user inserts two polynomials in the graphical user interface
2. The user selects the “addition” operation
3. The calculator performs the operation and displays the modulo of the division of polynomials

**Alternative sequence:** error if not both polynomials are introduced

**Use case 6:** del

**Primary actor:** user

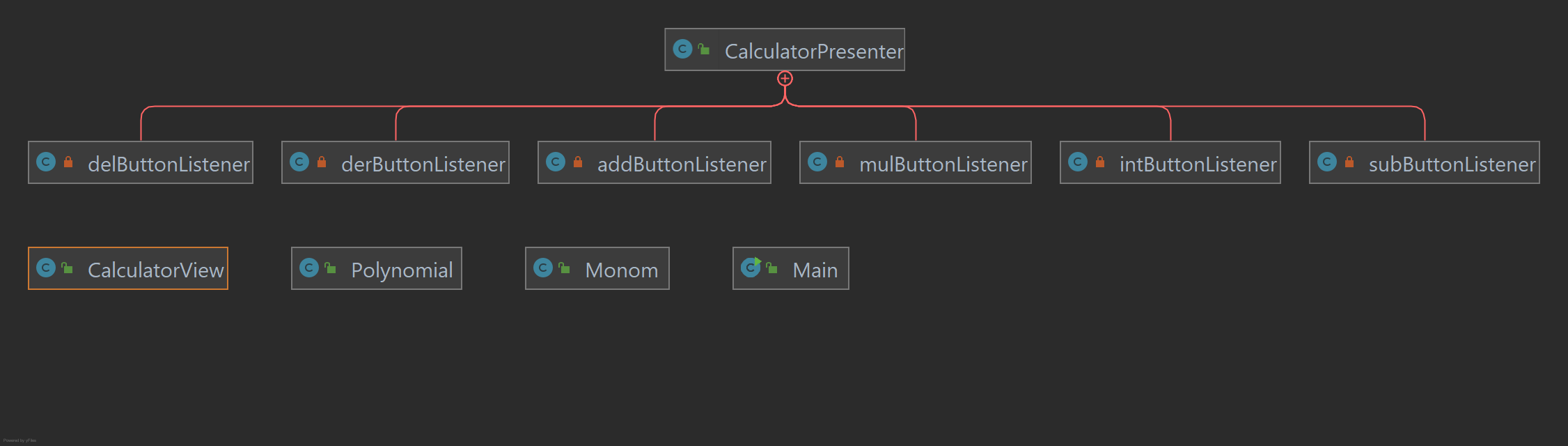
**Main success scenario:**

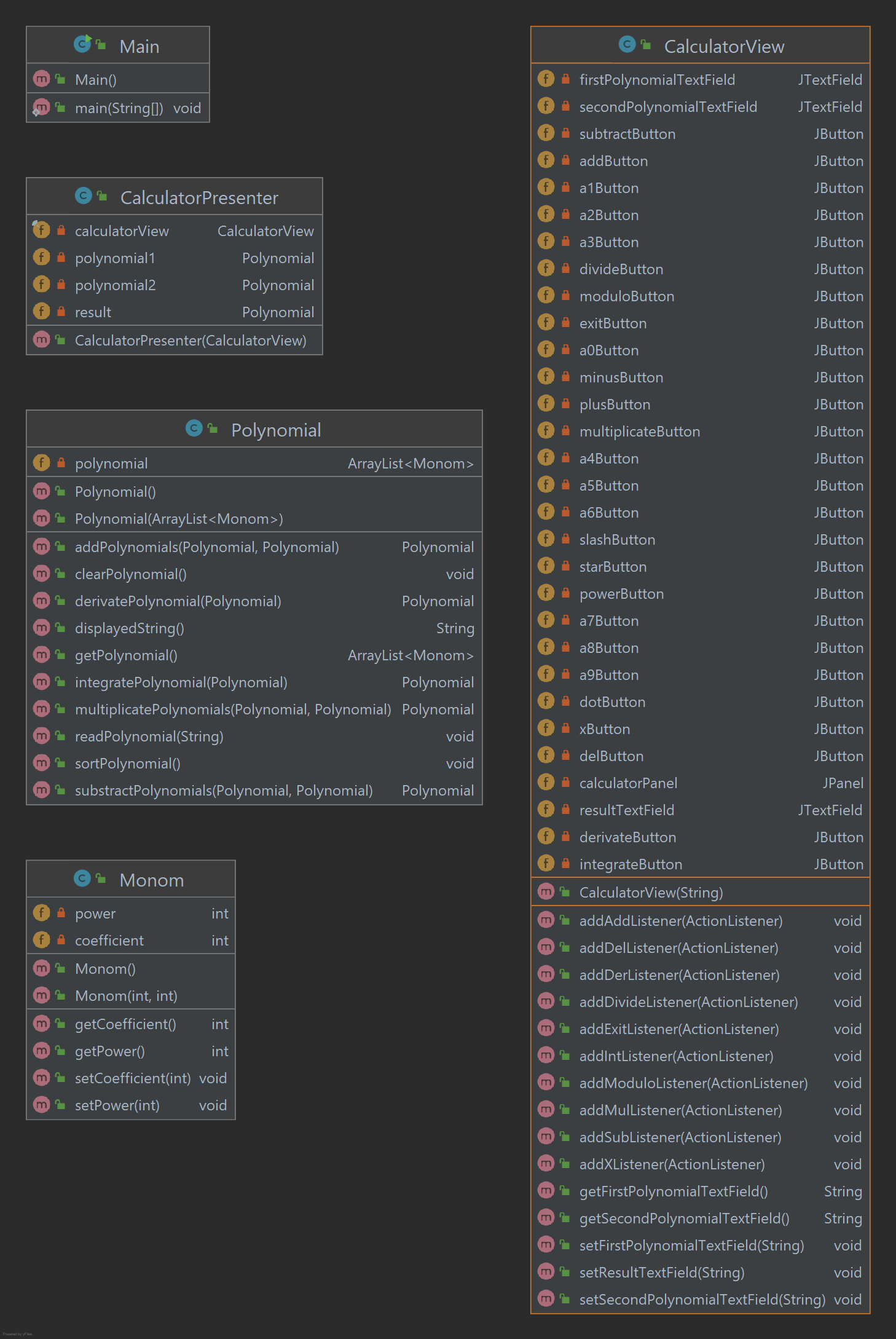
1. The content of polynomial 1, polynomial 2 and result is discarded

**Alternative sequence:** no alternative sequence

1. **Design**

**UML diagram:**

For the UML diagrams I chose two representations: one that displays inner classes and one that displays classes with their attributes and methods. The CalculatorPresenter class has 8 inner classes implemented through ActionListener.

****

**Data structures:**

The data structures used while implementing the project are either primitive types like integers and doubles, or more complex structures like ArrayLists of objects. Monom and Polynomial are objects created to serve as data structures as well.

Monom as an entity stores the coefficient and power of an unknown variable – in our case x. Using ArrayLists of generic type Monom, we can implement the structure defined as Polynomial and easily compute operatioins.

**Packages:**

I chose to implement the project after an MVP model; therefore, the packages were organized accordingly. The *model* is used for defining data such as the Monom and Polynomial. The *view* has only a passive role. It displays the data from the model and routes the events to the presenter to act upon the data. The *presenter* retrieves data from the model and formats it for the display.

Based on the above-described model, my classes were divided between these 3 main packages:

* Model: Monom, Polynomial
* Presenter: CalculatorPresenter
* Control: CalculatorView – CalculatorViewGUI and CalculatorView as a class where methods were implemented

1. **Implementation**

In the following section we will examinate in depth each of the implemented classes:

**Model package:**

**Monom Class:**

A monom is a structure that describes the attributes of one unknown variable.

Attributes:

* Power – integer
* Coefficient – double – although an integer when introduced, the integration and division operations may need a double type of structure

Constructors:

* Monom(int power, int coefficient) – initializes a monomial with the provided power and coefficient
* Monom() – initializes an empty monomial

Methods:

* int getPower() – returns the power of the monomial
* void setPower(int power) – changes the power of the monomial
* int getCoefficient() – returns the coefficient of the monomial
* void serCoefficient(double coefficient) – changes the power of the coefficient

**Polynomial Class:**

A polynomial is a sequence of monoms.

Attributes:

* polynomial – ArrayList<Monom> contains the sequence the monomials that define a polynomial

Constructors:

* Polynomial(ArrayList<Monom> polynomial) – initializes a polynomial with the provided list of monomials
* Polynomial() – initializes an empty polynomial

Methods:

* Arraylist<Monom> getPolynomial – returns the polynomial
* void readPolynomial(String textFieldString) – interprets the introduced polynomial String and converts it intro an ArrayList of monoms using pattern matching
* void sortPolynomial() – sorts the array list of monomials
* void clearPolynomial() – deletes the content of the current polynomial so that it can be reused
* Polynomial addPolynomials(Polynomial polynomial1, Polynomial polynomial2) – returns the addition of two polynomials
* Polynomial subtractPolynomials(Polynomial polynomial1, Polynomial polynomial2) – returns the subtraction of two polynomials
* Polynomial multiplicatePolynomials(Polynomial polynomial1, Polynomial polynomial2) – returns the multiplication of two polynomials
* Polynomial dividePolynomials(Polynomial polynomial1, Polynomial polynomial2) – returns the division of two polynomials
* Polynomial moduloPolynomials(Polynomial polynomial1, Polynomial polynomial2) – returns the modulo of two polynomials
* Polynomial integratePolynomial(Polynomial polynomial) – returns the integration of the first polynomial
* String displayesString() – transforms the polynomial in a String to be displayed on the graphical interface

**Presenter package:**

**CalculatorPresenter Class:**

This class links the view to the models provided in the model package.

Attributes:

* CalculatorView calculatorView – manages the graphical interface
* Polynomial polynomial1 – the first polynomial to be computed
* Polynomial polynomial2 – the second polynomial to be computed
* Polynomial polynomial – result

Constructors:

* CalculatorPresenter(CalculatorView calculatorView) – initializes the attributes and adds the button listeners

Inner classes:

Inside the CalculatorPresenter class the following classes are nested:

* addButtonListener
* delButtonListener
* subButtonListener
* derButtonListener
* intButtonListener
* mulButtonListener
* divButtonListener
* modButtonListener

**View package:**

**CalculatorView Class:**

This class manages the view. For implementing the graphical interface of the application, I chose a GUI form that comes with the CalculatorView class and the CalculatorView form. I used the form as to not have to precisely calculate the place of each element on the screen and because it keeps the graphical interface clean and organized. I won’t get into attributes because it involves a long list of elements that have only an aesthetic role, but I will list some of the methods implemented.

Constructors:

* CalculatorView(String title) – sets the title, extends the JPanel with all its components and closes the program when the “X” is pressed

Methods:

* void setFirstPolynomialTextField(String firstPolynomialTextField) – sets the content of the first polynomial text field
* void setSecondPolynomialTextField(String secondPolynomialTextField) – sets the content of the second polynomial text field
* void setResultTextField(String resultTextField) – sets the content of the result text field
* String getFirstPolynomialTextField() – gets the content of the first polynomial text field
* String getSecondPolynomialTextField() – gets the content of the second polynomial text field
* void add…Listener(ActionListener listener) – adds an action listener for each button on the screen in order to be used in the presenter

1. **Results**

Here I am going to include the input and the expected output for operations that have been tested through Junit testing:

|  |  |  |  |
| --- | --- | --- | --- |
| First polynomial | Second polynomial | Operation | Result |
| x^2+2x+1 | x+1 | + | x^2+3.0x^1+2.0x^0 |
| - | x^2+x^1+0.0x^0 |
| \* | x^3+3.0x^2+3.0x^1+x^0 |
| / | x^1+x^0 |
| % | x^1+x^0 |
|  | Integr. | 0.3333333333333333x^3+x^2+x^1 |
| Deriv. | 2.0x^1+2.0x^0 |

1. **Conclusion**

The project resembled the OOP project developed last year. However, there were many aspects that were implemented much more carefully and organized due to the steps of preparation and design before starting the actual implementation.

Although I started my project even before it was presented to us, time proved to not be enough. Therefore, in addition to the organizational skills I managed to acquire, I also learnt how crucial time management is. However long the planning part took, it saved me a lot of time and is a good use I will count on from now on.

1. **Bibliography**

https://en.wikipedia.org/wiki/Polynomial#:~:text=In%20mathematics%2C%20a%20polynomial%20is,x2%20%E2%88%92%204x%20%2B%207.