Technical University of Cluj-Napoca

Programming Techniques

Laboratory – Assignment 1

Polynomial Calculator

Logo, company name

Description automatically generated

Teacher: prof. Ioan Salomie

Teacher Assistant: Dr. Cristina Pop

Student: Rusu David

Group: E\_30424

Contents

[1.Assignment objective 3](#_Toc66728166)

[2.Problem analysis, modeling, scenarios, use cases 3](#_Toc66728167)

[I. Problem analysis 3](#_Toc66728168)

[II. Modeling 3](#_Toc66728169)

[III. Use cases and scenarios 4](#_Toc66728170)

[3.Design 5](#_Toc66728171)

[I. Class design decisions 5](#_Toc66728172)

[II. UML diagrams 6](#_Toc66728173)

[III. Data structures 7](#_Toc66728174)

[IV. Packages 7](#_Toc66728175)

[V. Algorithms 7](#_Toc66728176)

[VI. User interface 8](#_Toc66728177)

[4.Implementation 10](#_Toc66728178)

[Monomial 10](#_Toc66728179)

[Polinomial 10](#_Toc66728180)

[Controller 10](#_Toc66728181)

[Views Package 11](#_Toc66728182)

[5.Results 11](#_Toc66728183)

[6.Conclusions 11](#_Toc66728184)

[7.Bibliography 11](#_Toc66728185)

# 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 (i.e. addition, subtraction, multiplication, division, derivative, integration) to be performed and view the result. Note: Consider the polynomials of one variable and integer coefficients.

# 2.Problem analysis, modeling, scenarios, use cases

## Problem analysis

The first step in our analysis is to define what is a polynomial, according to Wikipedia, a polynomial is an expression consisting of variables and coefficients that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponentiation of variables. An example of a polynomial is given by the following generic formula:

.

where **a** {\displaystyle a\_{0},\ldots ,a\_{n}}are constants and **x** is the indeterminate.

As we can see, a polynomial is a sum of terms, a term is given by the following form: . The official name for a term is **Monomial,** thus we can say that a polynomial is formed by a sum of monomials. A monomial can be broken down even further, into **coefficient** and **power.** The coefficient is represented as and it is a **real** number. The power is represented as **n** and can be found here: , it is a **natural** number.

Now that we know what a polynomial is, we can move on to the second step in the analysis. This step is dedicated to the fact that our application will need a UI. We can observe that all the operations will be done through it whether it be introducing the polynomial or viewing the result.

## Modeling

Now that we analyzed the problem, we can start to model it. Our app will be made up of two main parts:

1. The user interface

This will be the part that the user interacts with, so it must be intuitive, good looking and **complete**. The UI is complete only if the user can perform all the required operations through it, such as:

- Addition

- Subtraction

- Multiplication

- Division

- Differentiation

- Integration

- Polynomial introduction

The results of all the above operations should also be shown to the user through the UI, for example in a text box.

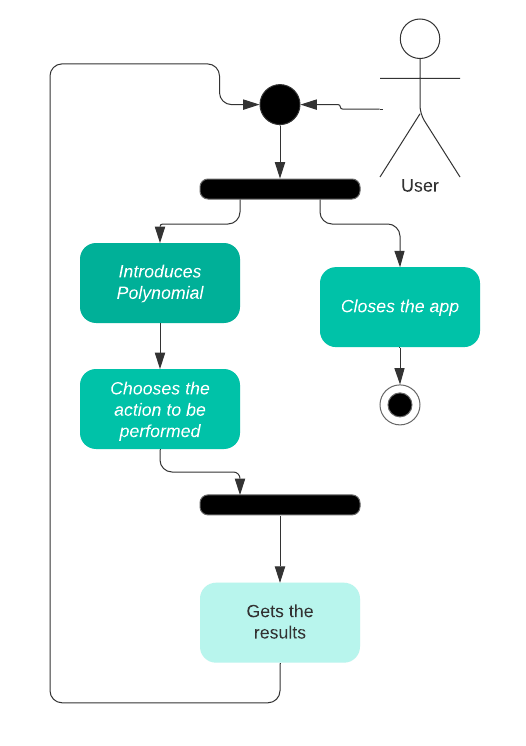
1. The backend functionality

The backend will host all the algorithms for the operations mentioned above, as well as the overall program flow, such as what happens when a button is pressed. This part will not be directly visible to the user, however it is crucially important for it to be stable and correct.

## Use cases and scenarios

This app has a very exact set of use cases, seeing that it is a calculator, its primary use case is to compute a result for the user, thus the app will do operations on polynomials that are introduced by the user.

Use case and scenario diagram:



Following the above diagram, we will describe each step.

1. Choose whether to close the app or continue using it.
2. If the user does not close the app, he will then have to input the desired polynomials in text fields.
3. Choose the desired operation.
4. Based on the previous inputs, the user will either get a result, or an error message. For example, if the user tries to divide by 0, the result will be a warning message saying that we cannot divide by 0. If the inputs are correct, then the result will be shown to the user in the UI.

# 3.Design

## I. Class design decisions

One of the most important parts in designing our app is to choose a structure. We have 2 options here, either a custom structure or an already existing one, better known as a design pattern. In this case we will go with a preexisting design pattern called MVC (Model View Controller). The aim of the MVC is to divide the program into 3 interconnected elements.

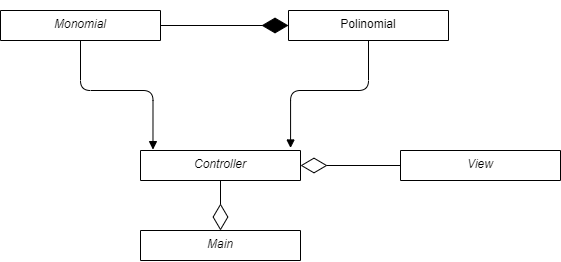
* **Model:** It is the application's dynamic data structure, independent of the user interface.
* **View:** Renders the user interface. Any representation of information such as a chart, diagram, or table.
* **Controller:** responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it, and then passes the input to the model.

Graphical user interface, application, timeline

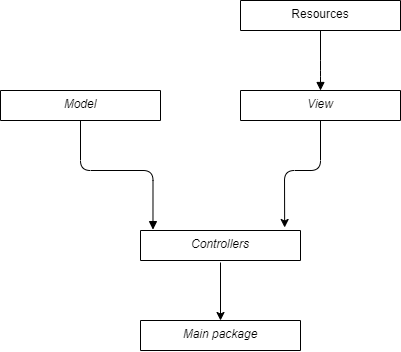
Description automatically generated

In our case, the model will be made up of the monomial and polynomial classes, the view will be the JavaFX app and the controller will hold the algorithms as well as the program flow logic.

## II. UML diagrams

Class wise dependency diagram

Package wise UML diagram:



## III. Data structures

All the application specific data structures can be found in the Model package. In that package we have 2 classes that represent a data type:

* Monomial, data type that holds a monomial
* Polinomial, holds multiple monomial’s thus forming a polynomial

Besides the above-mentioned data types LinkedList’s were also used to implement the polynomial data type.

## IV. Packages

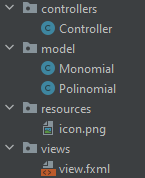
The application is split into multiple packages in order to obey the MVC design pattern. The application has 4 main packages:

- controllers: Holds the Controller class

- model: Holds all our application's dynamic data structures

- resources: In this package we find images that are used in the view component

- view: In this package, the user interface is found



## V. Algorithms

Seeing as this application is a polynomial calculator, there are quite a few algorithms used. The following are the most important:

1. Addition

This operation is performed on two polynomials, the result is obtained by adding the coefficients of monomials that have the same power.

b) Subtraction

This operation is performed on two polynomials, the result is obtained by subtracting the coefficients of monomials that have the same power.

c) Multiplication

This operation is performed on two polynomials, the result is obtained by multiplying each monomial of the first polynomial with each monomial of the second polynomial. When multiplying the monomials, the powers are added, and the coefficients are multiplied.

d) Integration

This operation is performed on one polynomial, the result is obtained by adding 1 to the power of each monomial, and multiplying the coefficient of each monomial with

1/ (the power of the monomial).

e) Derivation

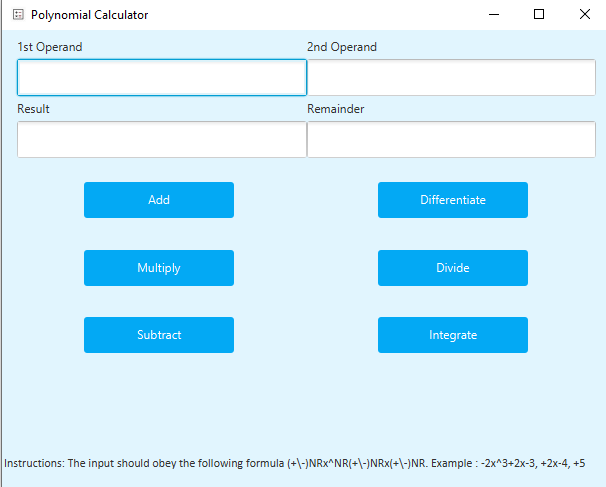
This operation is performed on one polynomial, the result is obtained by multiplying the coefficient with the power, after that the power is subtracted by 1. If the initial power is 0, that monomial is deleted from the polynomial.

f) Division

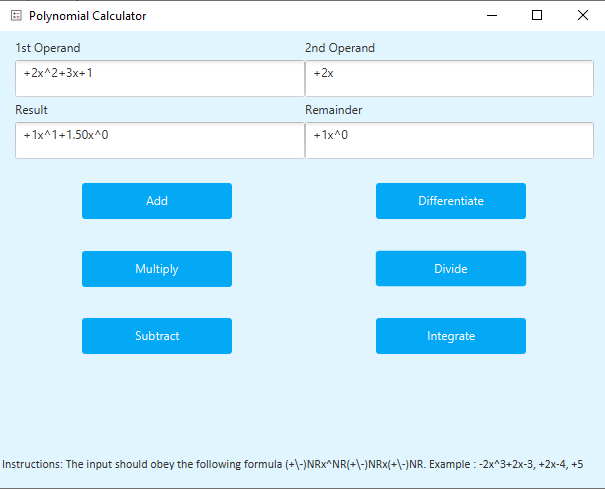
This operation is performed on two polynomials, the result is obtained by using the long division polynomial algorithm.

## VI. User interface

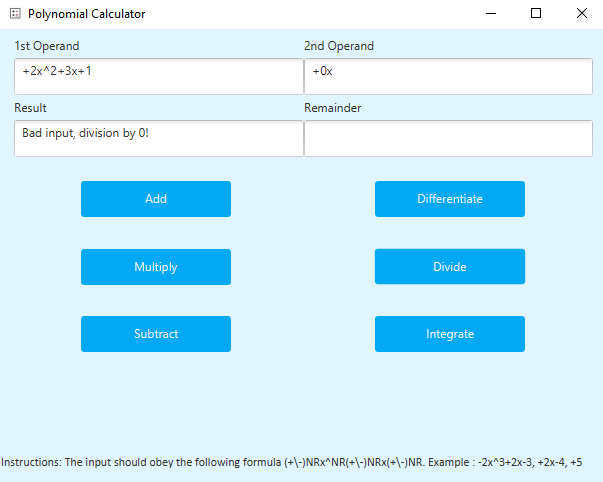
The user interface was written in JavaFX this was chosen over Swing because it is more modern. Through the interface the user can interact with the algorithms in a much friendlier manner. Below we will describe each element from the user interface.

* 1st Operand: The user inputs the desired polynomial here (the input must follow the correct format).
* 2nd Operand: The user inputs the desired polynomial here (the input must follow the correct format).
* Result Field: The result will be displayed in this field, or in case of an error, a message will be shown.
* Remainder Field: This field is only used for the Divide operation; it displays the remainder.
* Add button: This button adds the 1st polynomial to the 2nd polynomial
* Multiply button: This button multiplies the 1st polynomial to the 2nd polynomial
* Subtract button: This button subtracts the polynomials in this order **1st – 2nd**
* Differentiate button: This button differentiates the **1st** polynomial
* Divide button: This button divides the polynomials like so: **1st / 2nd**. The results are shown in the Result and Remainder field.
* Integrate button: This button integrates the **1st** polynomial. 

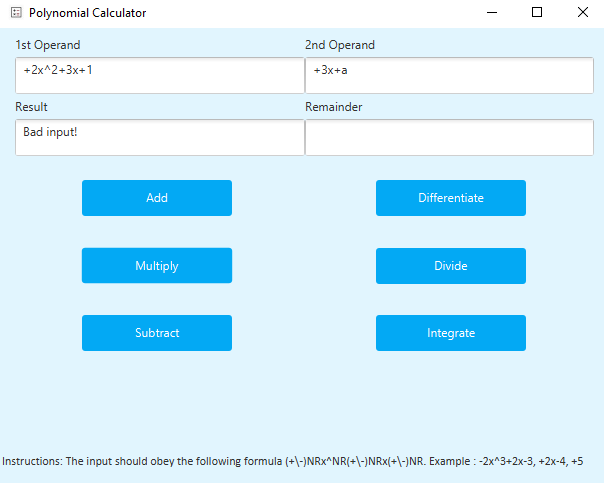
1. In the screenshot below, there is a valid division example.



1. In the below screenshot, there is a division by 0 example. Note the error message.



1. In the below screenshot, there is a bad input example. The issue is at the second polynomial **“+3x+a”**, note that **a** should not be there, as only **x** is acceptable as the indeterminate variable.



# 4.Implementation

The implementation of this project is done in 3 classes and one package. Their description is found below:

## Monomial

This class is in charge for story a Monomial, it has two private fields **power** and **coefficient.** Thosefieldsare changed with getters and setters. A very important method is the “toString()” method, it handles the way the monomial is represented. For example, if the monomial has a floating coefficient, then it will be represented with a floating number, however if the coefficient is an integer, then it will be represented with an integer, thus it will not have a decimal point.

## Polinomial

This class oversees storing a polynomial, it has one private field, a LinkedList of Monomials.

Using getters and setters we can change that list of monomials. In this class we have multiple important methods. The most important method is “*regexParser(String poli)*”, this is used in the constructor to convert a string to a polynomial. Another important method is “toString()”, this returns the polynomial under a string.

## Controller

This class handles the algorithms, the data models as well as the interface functionality.

In the class we have two types of methods:

1. Methods that are called from the view

These are used so that some code will execute when a certain button is pressed. The following methods provide this functionality: AddBtn(), SubtractBtn(), MultiplyBtn(), DivideBtn(), DifferentiateBtn(), IntegrateBtn(). In general, all these methods call the appropriate algorithm to compute some data, and then set the fields with the results.

1. Helper Methods

These methods are mainly the algorithms that were covered in section V. Algorithms. Besides the algorithms, there are two other important methods, “isGoodInput(String input)”, this checks if the introduced polynomial obeys the required string format, “normalizePolinomial(polinomial pol)”, this method formats the polynomial to be in non-increasing order of the powers, it also deletes null elements form the polynomial, example : “+0x^0”.

## Views Package

In this package we have the user interface. It is written in fxml, and the main structure used is a FlowPane one. Inside the flowpane, we use a gridPane, this way we can organize all the children (in this case the buttons and input/output fields) to be in a grid format.

# 5.Results

To test the program JUnit was used to write the unit tests. For each operation, a few cases were wrote such that most scenarios are covered. The tests themselves can be found in the **test/java** folder under the class “ControllerTest”. To run them, the ControllerTest class must by run. As an example we will take the poliDifferentiate() test method. The first test, “+2x^2+2x+1” is a general way to see that the main functionality is there. The next test is an edge case “+1”, this should yield “0” as the result, which it does. The tests for the other operations follow a similar structure.

# 6.Conclusions

In conclusion, this assignment helped cover the MVC design pattern, which helps divide a project in such a way that multiple people can work on it at the same time as well as making the project easier to understand and modify later down the road. Another important skill learned is Regex expressions, this helps a lot in parsing strings or verifying if a string obeys a certain pattern. JavaFX was also learned in this assignment, this offers a much more modern way to write a user interface and it is much easier to make a visually pleasing one. The last new skill acquired is, writing tests using Junit, even though it is not necessarily hard it is a must know.

# 7.Bibliography

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

<https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm#:~:text=MVC%20Pattern%20stands%20for%20Model,or%20JAVA%20POJO%20carrying%20data.&text=It%20controls%20the%20data%20flow,the%20view%20whenever%20data%20changes>.

<https://regex101.com/>

<https://openjfx.io/openjfx-docs/#gradle>

<https://www.tutorialspoint.com/junit/junit_test_framework.htm>

<https://en.wikipedia.org/wiki/Polynomial>