

**Technical University of Cluj-Napoca**

**Faculty of Automation and Computer Science**

Programming Techniques

Assignment 1

Polynomial calculator

Teacher: Prof. Ioan Salomie Student: Alina Mihuț

Teaching assistant: Ing. Ciprian Adrian Stan Group 30422

Table of contents

1. Assignment objective
2. Problem analysis, modeling, scenarios, use cases
3. Design (design decisions, UML diagrams, class design, relationships, packages, algorithms, user interfaces)
4. Implementation
5. Results
6. Conclusions
7. Bibliography
8. Assignment objective

The objective of the assignment is to design and implement a polynomial calculator with a graphical interface with which the user can insert two polynomials, select the mathematical operation (addition, subtraction, multiplication, division, derivation or integration) which he wants to perform and view the result.

1. Problem analysis, modeling, scenarios, use cases
2. Analysis

In mathematics, 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. A polynomial in a single indeterminate x can always be written (or rewritten) in the form *an Xn + an-1 X n-1 + … + a2 X 2 + a1 X + a0*, where a0,…,an are constants and x is the indeterminate. Each term (monomial) consists of the product of a number, called the coefficient of the term[a] – and a finite number of indeterminates, raised to nonnegative integer powers.

The performance of polynomial operations on paper is usually time consuming. So, the solution is the implementation of a polynomial calculator, which is intuitively and easily to be used.

B) Modeling, scenarios and use cases

In the specifications of the assignment, the polynomials to be introduced have integer coefficients, positive, integer powers and a single indeterminant, called X.

The main scenario is the following one:

The user introduces the two polynomials in the corresponding text fields. The application will parse the input and store the values in two objects instances of the Polynomial class. The parsing is done with a regex specific to polynomials, which splits each monomial in 4 groups (the sign, the coefficient, the indeterminant and the power). The user then selects one of the buttons specific to the following operations: addition of the two polynomials, subtraction of the second polynomial from the first one, multiplication of the two polynomials, division of the first polynomial with the second and differentiation and integration of the first polynomial. If the user doesn’t introduce both polynomials for the first four operations or at least one polynomial for the last two operations, an alert box will appear.

The program computes the selected operation and displays the result in a separate text field. The user also has the option to clear all the text fields with a Clear button, if he wishes to introduce another set of polynomials.

The problem has six use cases, each corresponding to one of the six operations to be performed. If we take for example the use case of addition, the flow is the following one:

1. The user introduces the two polynomials, each in a separate text field. If the user introduces only one polynomial, an alert message will appear on screen.
2. The user selects the “Addition” button.
3. The polynomial calculator performs the addition operation and displays the result

The same scenario goes for the multiplication and subtraction operations.

The use case of division flows in the following way:

1. The user introduces the two polynomials, each in a separate text field. If the user introduces only one polynomial, an alert message will appear on screen.
2. The user selects the “Division” button.
3. If the degree of the second polynomial (divisor) is bigger than the degree of the first polynomial (dividend), an error message will appear on screen, saying that the operation can’t be performed
4. If division is possible, the calculator will compute the result and display the quotient and the remainder.

The use cases for integration and differentiation is the following one:

1. The user introduces the polynomial in the first text field. If the user doesn’t introduce at least one polynomial, an alert message will appear on screen.
2. The user selects the “Derivation” or “Integration” button.
3. The calculator will compute the selected operation and display the result.
4. Design

On the top level, the overall system design of the polynomial calculator is represented by three main inputs: the two polynomials and the operation to be performed and the output: the result of the operation.

On a second level, the design is represented by the division of the program in multiple packages with different responsibilities.

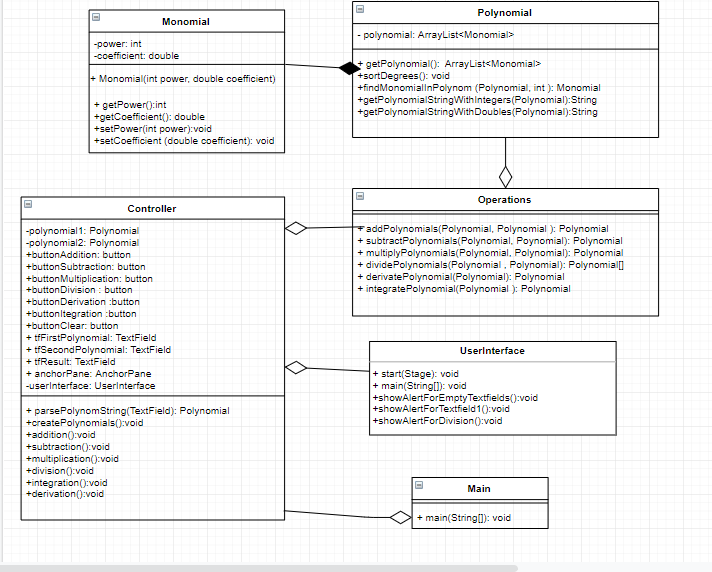
The Model-View Controller pattern is an architectural design pattern which divides the project in three main components: model, view and controller.

* Model – represents the business layer of the application and contains the classes modeling the application data;
* View –represents the visualization of the data the model contains and keeps the classes which implement the graphical user interface;
* Controller - acts on both model and view. It controls the data flow into model object and updates the view whenever data changes.

I designed my project in accordance to the Model-View Controller pattern, so I divided my classes into four main packages:

* Application – the main package which keeps only one class – Main, which runs the whole project.
* Model – the package which holds the objects I worked with in this project (Monomial and Polynomial) as well as the Operations class, with the business logic of the operations I implemented.
* View – the package which contains the User Interface class, with the methods concerning the running of the Graphical User Interface and its’ proper working.
* Controller- the package with the Controller class, as its name suggests, which controls both the GUI classes and the model classes and the data flow between the two. It receives keyboard inputs from the GUI as well as button presses and translates the events into requests, which are sent to the model.

On a third design level, each package is divided into several classes, having their own attributes and containing specific methods. The methods used by each class and the relationships between classes are presented in the following UML diagram.



In the design of the model, I also used several algorithms in order to perform the six operations on the polynomials. For the implementation of each algorithm specific to each operation, I used my mathematical background from high school. Each computation algorithm is done in a specific method in the Operations class, which takes as input one or two polynomials and returns a new polynomial, result of the algorithms.

1. Implementation

Each of the three packages in the Model-View Controller pattern is divided into several classes, containing specific methods.

The model package is divided into three classes: Monomial, Polynomial and Operations.

The Monomial class has the attributes ***int power*** and ***double coefficient*** and holds the information of each monomial in the polynomial. Even though the polynomials are introduced with integer coefficients, the result of the division, integration and differentiation operations can have floating point coefficients, so that’s why I chose to make the coefficient attribute double from the beginning. The Monomial class holds the constructor of the object ***public Monomial(int power, double coefficient)*** as well as the getters and setters of each attribute.

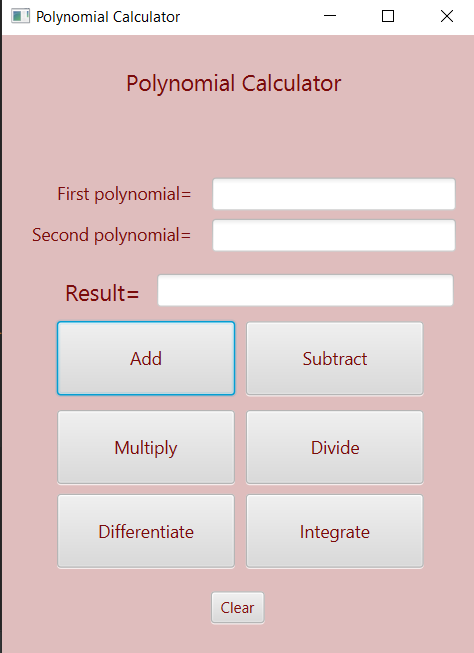
The Polynomial class has only one attribute, an ***ArrayList<Monomial>*** , which keeps the data of all the monomials in the respective polynomial. Apart from the getters and setters, it has several other methods: ***public void sortDegrees() –*** which sorts the monomials in the polynomial in the descending order of their degrees; ***public Monomial findMonomialInPolynom(Polynomial p, int power) –*** which finds and returns (if it exists) the monomial of a respective degree in the polynomial; as well as some helper methods ***public String getPolynomialStringWithDoubles(Polynomial p)*** and ***public String getPolynomialStringWithIntegers(Polynomial p)***, used to return the string created from the resulting polynomial (with double coefficients for the result of division, differentiation and integration and integer coefficients for addition, subtraction and multiplication), in order to display it.

The Operations class has six methods, each for one of the operations (addition, subtraction, multiplication, division, differentiation or integration) which take as parameters either one or two objects of the Polynomial type, perform the respective algorithm of each operation and return a Polynomial object, representing the result of the operation. The division operation is the only one which returns an array of type Polynomial[], of size 2. The first element in the array represents the quotient of the operations and the second one represents the remainder.

The User Interface class in the view package holds the ***start()*** method, which initializes the primary stage and displays it on the running of the application as well as other method which perform actions on buttons or display alerts in the graphical interface for different purposes.

The Controller class in the controller package keeps the methods which set the data flow between the models and the user interface. The method ***public Polynomial parsePolynomString(TextField textField)*** parses the inputs inserted by the user in the text field and creates an object of polynomial type and returns it. The parsing is done using a specific regex for polynomial patterns, which extracts for each monomial in the input string the coefficient and the power. The 6 methods ***public void addition(), public void subtraction(), public void multiplication(), public void division(), public void derivation() and public void integration()*** are each set on the action of a button and they hold the top-level logic of each operation: they create the polynomials, call the method from model.Operations which performs the specific operation, then take the result of the operation and transform it into a string and display the string in the result text field.

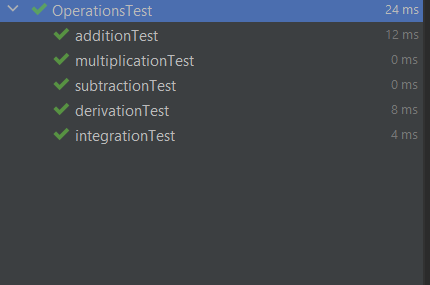
In the implementation of the graphical user interface, I used Scene Builder. The GUI contains only one window, simulating the appearance of a calculator. It has two text fields, where the user inserts the two polynomials, a text field where the result is displayed, six buttons (Add, Subtract, Multiply, Divide, Differentiate and Integrate ) for the computation of each operation and a Clear button which clears the content in each of the text fields. I designed the user interface in order to be easy and inductive for the user. The design of the calculator’s user interface can be seen in the following picture.



1. Results

In order to verify the correctness of the application, I performed Unit Testing, using Junit. Unit Testing is mostly used to verify a small chunk of code by creating a function or a method. I performed Unit Testing on the methods which implement the six operations. I created a java test class called ***OperationsTest.java,*** where I implemented six test methods, one for each operation. In each method, I created one or two polynomials, performed the specific operation, created the result string and then tested if the result string with ***AssertTrue()***. With this occasion, not only did I test the implementation of the six operations, but also the construction of the result string to be displayed.

All six tests passed successfully.



1. Conclusions

Working on this project was a good opportunity for me to gain a deeper understanding of the OOP concepts learned in the first semester as well as working on an application with a graphical user interface. It was a bit challenging at first, until I had some ideas on how to structure and organize the project in order to respect the Model View Controller Pattern, but I realized that a good modelling of the problem from the beginning really helps a lot in the future development.

I am glad that, with the occasion of this project, I learned how to work with Maven and to perform Unit Testing. All in all, I liked working on this project and I feel like I gained a lot of knowledge from it.

As further improvements, a lot could be made. For example, to calculate the value of the expression for a certain X, or to plot the graph corresponding to the polynomial. Both this functionalities would each have separate buttons. As for the user interface, it could also be improved to be more user-friendly. Another implementation would be the option for the user to reintroduce the result of the computation in one of the text fields so as to reuse it in future operations.

1. Bibliography

- ***Assignment\_1\_Support Presentation*** PowerPoint presentation