**DOCUMENTATION**

ASSIGNMENT 1

STUDENT NAME: LACATUS ARPAD ALEX

GROUP: 30423

# CONTENT

[1. Objective of the assignment 3](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297885)

[2. Problem analysis, modeling, scenarios, use cases 3](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297886)

[3. Design 3](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297887)

[4. Implementation 4](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297888)

[5. Results 6](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297889)

[6. Conclustions 6](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297890)

[7. Bibliography 6](file:///C:\Users\Administrator\Downloads\Documentation.docx#_Toc95297891)

8. UML Diagram 7

1. The Objective of the assignment - The main objective of this assignment is to 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. Some secondary objectives are:

-analyzing the problem and identifying the requirements

-designing a polynomial calculator

-implementing the polynomial calculator in java

-testing the polynomial calculator using Junit

1. Problem Analysis, modeling, scenarios, use-cases

First of all we need to consider all the possible steps our app could go through and what methods we require in order to implement this app as efficient as we could and as user friendly as possible. We will need a lot of methods on polynomials, monomials and also the ones from the Junit testing so we need to analyze well our problem and come up with the best solution.

**Analyzing** the problem, we have the following requirements:

-the polynomial calculator should allow user to insert polynomials

-the polynomial calculator should provide a way for the user to select the operation he wants to execute (add, subtract, multiply etc.)

-the polynomial calculator should provide a correct and comprehensive result

-the polynomial calculator should be easy to use and intuitive(non-functional)

For modeling the data, we chose the classes polynomial, monomial, monomialintcoefficient,

Monomialdoublecoefficient in order to make a clear distinction between the monomials and for a better structure in our program.

**Use cases**: Addition of polynomials, Subtraction of polynomials, Derivative of a polynomial,

Integration of a polynomial, Multiplication of polynomials, Division of polynomials

The primary actor is the user, and the **main scenario** would be:

* + 1. The user inserts the 2 polynomials in the graphical user interface.
    2. The user selects the operation from the selection tab (addition, multiplication etc.)
    3. The user clicks on the “compute” button
    4. The polynomial calculator performs the operation of the two polynomials (or one depending on the case) and displays the result

**Alternative Sequence:** Incorrect polynomials

- The user inserts incorrect polynomials (incorrect format of the polynomial)

- The scenario returns to step 1

To note that for operations such as derivative and integration the process is slightly changed, requiring the user to insert only a single polynomial due to the nature of the operation. For he division we also consider the scenario in which the result also has a rest, for this scenario we need the field rest which provides the additional information that the user seeks.

Another thing to consider is the input of the app. For this app to work the user is supposed to provide a valid polynomial with monomials with power clearly specified.

1. Design

Leve 1: System design:

At the first level we have the 2 polynomial inserted and the operation selected as the inputs in the polynomial calculator and the result as the output

Level 2: Design into sub-systems/packages:

We will need 3 main packages for this assignment:

-Graphical User Interface (GUI) – the package that contains the classes used for implementing the graphical user interface, here we have the classes used for implementing theGUI (view, controller)

-Data Models (data. models) -the package that contains the classes modeling the application data (polynomial, monomial, monomialintcoefficient, monomialdoublecoefficient), here we have the main classes used for implementing the polynomials with polynomial having a list of monomials, each monomial being either monomial with int coefficient or monomial with double coefficient

-Business Logic(logic) – the package that contains the classes used for implementing the mathematical operations functionality, here we have the operations that our app can perform on the polynomials

Level 3: Division into classes

The 2 main classes use are Polynomial and Monomial. Monomial has 2 subclasses which we use MonomialIntCoefficient and MonomialDoubleCoefficient, used in order to clarify the nature of the monomial and ease some of the operations performed on them.

Polynomial is the class that hold monomials, and on which are all the operations performed.

Level 4: Division into routines

All the classes implemented are divided in routines in order to facilitate our work and make the code much easier to understand, the methods don’t have more than 30 lines of code in order to not make things too complicated.

Level 5: Internal routine design

In each routine our code is well delimited in pieces of code with different functionalities, which combined give us a final routine easy to follow and comprehend.

1. Implementation

Data Model classes:

**Monomial Class**: The class used to store a monomial (coefficient and power). It is used to fill the list in the Polynomial. It has 1 private attribute power of type int with its setter and getters, the only constructor takes as the argument the power and sets it for the monomial. This class is the unit we make most of all operations eventually and at almost each line of code it must be casted as one of the following to classes in order to obtain its full functionality.

**MonomialIntCoefficient** and **MonomialDoubleCoefficient** classes: These 2 classes are implementations of the monomial super class, and they are used to better identify the monomial by its coefficient. They both have an extra attribute coefficient of type int for the first class and double for the second one. They also have getters and setter for this attribute and in their constructors, it is taken as an argument. This classes are mainly used with casting a monomial to them so we won’t have many instances of them. Their implementation is needed for operations such as division and integration where there is a clear distinction between the resulted coefficients here using MonomialDoubleCoefficient and for the other operations such as addition, subtraction etc. we use MonomialIntCoefficient. To note that the input polynomials have only Monomials with int coefficients, this requirement being specified in the assignment presentation, so a polynomial with double coefficient is a bad input.

**Polynomial class**: This is the class on which we perform our operations. Each instance of the class has its own list of monomials. The class has only the empty constructor and a method addMonomial in which, using casting we add the monomials of different types changing coefficient when needed, also removing an element if the coefficient becomes 0 after a operation, this is done through casting of the monomials in order to get their coefficient and by traversing the array list we search for a place for the monomial that needs to be added, if a monomial with the same power already exists we just update its coefficient. Other methods implemented here are degree – for getting the degree of a polynomial (used in the implementation of the operation divide), leading – for getting the leading monomial of a polynomial (used in the implementation of the operation divide), toString – for making a string form the polynomial in order to be used at the output, getPolynomialFromString – for getting the polynomial from a string (obtained character by character),method used for the input of our app as well in the testing , equals – for comparing 2 polynomials and checking if they are the same by comparing each of their monomials, copy -returns a copy of the polynomial (this method use is to not change unintentionally the original while working with it)

Logical classes:

**Operations Class**: The class that implements all the operations between polynomials. All the methods are static. The methods implemented here are:

-add- makes the sum of 2 polynomials by traversing the lists of the monomials for both polynomials and applying addMonomial on all of them and returns the resulted polynomial

-subtract – makes the subtraction between 2 polynomials by traversing similarly as the addition but for the second polynomial all its monomials have their coefficient changed prior to applying the addMonomial method on them, returns the resulted polynomial

-multiply – makes the multiplication of 2 polynomials by traversing their monomials and for each pair adding a new monomial with coefficient = coefficient1 \*coefficient2 and power = power1 \*power2 to the resulted polynomial, then returning it

-divide – makes the division between 2 polynomials by repeatedly subtracting and thus obtaining the rest and the result of the operation in a list with 2 elements, eventually returning that said list, here we use the methods degree, leading

-derivative – computes the derivative of a polynomial and returns its value, computation is done by decrementing by one the powers of each monomial and multiplying their coefficient with their power’s old value

-integration – computes the integral of a polynomial and returns its value, computation is done by incrementing by one the powers of each monomial and dividing their coefficient by their power’s old value

GUI classes:

**Controller class**: This class controls the graphical user interface having access to the view class. Has a single method responsible with gathering the input, selecting the right operation needed to be performed based on user’s selection, sends data to the logical unit and then redirects the result to the view, by calling methods implemented specifically for this purpose.

**View class**: This class manages the visual part having all the panels, labels, text fields and buttons. It has getters and setters for its components and methods needed to prepare the GUI part. Method prepareGUI prepares initializes the GUI. Methods prepareNumberPanel and prepareResultPanel are used to initialize the panels that are used with all their fields (buttons, text fields, labels).

Other things to be considered:

For the implementation for the function fromString that returns a polynomial from a string the implementation leaves some restrictions – for each monomial it is mandatory to specify its power i.e., inputs such as “…+x” or “…+4” are not tolerated. A good example for the input would be “x^2+2x^1+x^0”.

For the implementation of the polynomial’s list of monomials I used Array lists in order to easy access each element when needed for removal or addition.

1. Results

For testing Junit was used. The tests consisted of picking random operations and testing if the app gave us the supposed results. A polynomial was created at the beginning of each test (the result which was supposed to be returned for that said operation) then using function assert true we compare the results with equal (). If the result would not be the one, we search for a message “Operation doesn’t work as intended” will be shown. I picked easy values that would give a result with integer coefficient in order to be easy to follow, but for the operations such as integration and division the operation works for the cases in which the result is a double. These tests are essential in order to prove that our app works as intended and the Junit doesn’t fail our expectation in providing a easy to implement AppTest in which with a few lines of code our app proves its functionality.

The app results will be displayed at the result label, and rest respectively as polynomials with the format “+3.00x^2+1.00x^1+0.50x^0 “, this format is provided by the method toString which can be found in the class Polynomial

1. Conclusions

To conclude, this homework proved to be a bit challenging mainly due to implementation of the 2 subclasses MonomialIntCoefficient and MonomialDoubleCoefficient because of the high amount of casting needed, but in the end, I think it helped me gains some experience working with java, Junit, gitlab and many more. Getting the use GUI was also challenging at first but in the end with the help of many examples I managed to create a decent graphic user interface.

I consider that there is still room for improvement in this project and will try to level up my skills for the future ones.

.

Graphical user interface

Description automatically generated