**DOCUMENTATIE**

***TEMA 1***

NUME STUDENT: NEAGOI MIHAI

GRUPA: 30421

**CONTENT**

1. Objectives …....................................................................................................... 3

2. Problem analysis, modeling, scenarios, use cases ….......................................... 3

3. Design …............................................................................................................. 3

4. Implementation ................................................................................................... 3

5. Results ................................................................................................................ 3

6. Conclusions ........................................................................................................ 3

7. Bibliography ....................................................................................................... 3

**1.Objectives**

The main objective 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.

The secondary objectives are:

• Analyze the problem and identify requirements

• Design the polynomial calculator

• Implement the polynomial calculator

• Test the polynomial calculator

The problem analysis will be described in the second chapter along with the use cases and the modeling.

The analysis of the problem will be done in more steps: identifying and defining the problem, generating possible solutions, evaluating alternatives, deciding on the implementation followed by a final evaluation.

The modeling will be done using the UML diagrams.

Each use case will branch into two categories the success scenario and the alternative scenarios.

To design the polynomial calculator, we will make use of the class diagrams from previous chapter in the modeling part.

For the implementation of the calculator, we will use Java 8.

The Results chapter will consist of a quick overview of the progress made so far.

The Conclusions chapter will contain a recap of the concepts learned and the challenges encountered during the whole process.

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

The problem statement is to design and implement a polynomial calculator that can perform the following six operations: addition, subtraction, multiplication, polynomial long division, integration and to compute the first derivative.

**2.0 Theoretical considerations**

A polynomial is in fact a sum of monomials. Each monomial is of the form c\*x^n where c is any real number and n is a nonnegative integer.

When adding two polynomials we factor each x at the power n and add the coefficients.

Example: (x^3-x^2) + (2x^2-x^1) = x^3 + x^2 – x

Same thing goes for subtracting the polynomials.

When multiplying two polynomials we must multiply each monomial from the first polynomial to every monomial from the second polynomial and then normalize the result. To normalize the result, we factor the x’s with the same power.

Example: (x^2+1) \* (x+1) = x^3 + x^2 +x + 1

To divide 2 polynomials, we can use Horner’s algorithm or Blomqvist's method.

When dividing 2 polynomials, A and B, we get a quotient Q and a rest R such that A = B\*Q + R.

Computing the first derivative of a polynomial is done by applying the first derivative to each monomial contained in the polynomial, using the formula:

(c\*x^n)’ = n\*c\*x^(n-1)

To compute the integral of a polynomial we apply the formula (c/(n+1)) \* x^(n+1) term by term.

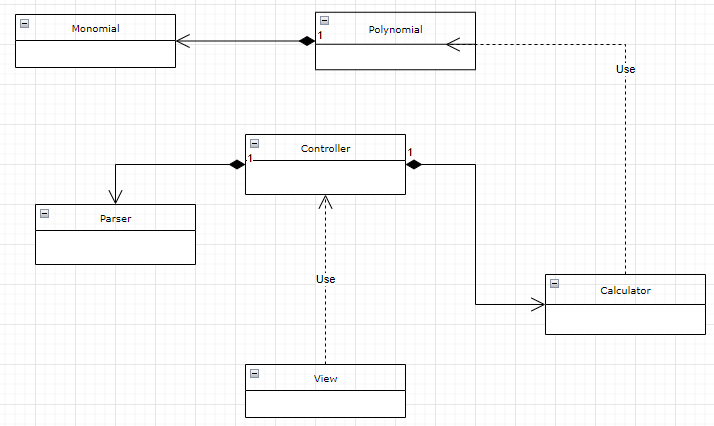
**2.1 Modeling the problem**

The user must be able to input two polynomials and perform one of the six operations. If the expression for the polynomial is invalid the application should inform the user how to write the polynomial expression. After the two polynomials are introduced the end user should press on one of the six operations. The result will be displayed in a text box.

The six operations that the user can choose from are:

* + Polynomial long division
  + Multiplication
  + Addition
  + Subtraction
  + Integration
  + First Derivative

Knowing that each polynomial is composed of multiple monomials we can design the Polynomial class such that it contains a list of Monomials. We can use a Calculator class to perform operations on these Polynomials. And a parser class to check if the input from the end user is valid and to convert the data types. Knowing this the following class diagram comes naturally:



**2.2 Use cases**

Use case: add polynomials

Main success scenario:

1. The user inserts the first polynomial in the first text field

2. The user inserts the second polynomial in the second text field

3. The user clicks on the ‘+’ button

4. Calculator performs the addition and displays the result in the third field

Alternative sequence: incorrect format

-The polynomials inserted are incorrect

-Second polynomial is 0 and the user performs division

-User is prompted with an error and we return to step 1

Functional Requirements:

- The polynomial calculator should allow users to insert polynomials

- The polynomial calculator should allow users to select the mathematical operation

- The polynomial calculator should add two polynomials

- The polynomial calculator should subtract two polynomials

- The polynomial calculator should multiply two polynomials

- The polynomial calculator should perform the long division of two polynomials

- The polynomial calculator should integrate the first polynomial

- The polynomial calculator should compute the first derivative of the first polynomial

Non-Functional requirements:

- The polynomial calculator should be intuitive and easy to use by the user

- The graphical user interface should be as simple as possible and contain analogous colors

**3.Design**

The division into sub-systems was done by using the model view controller architectural pattern.

Knowing that a polynomial is a collection of monomials we can deduce that between the monomial class and the polynomial class we have a composition relationship. The controller uses the parser in order to feed the calculator the corresponding data. Then fetches the result from the calculator and the view collects it.

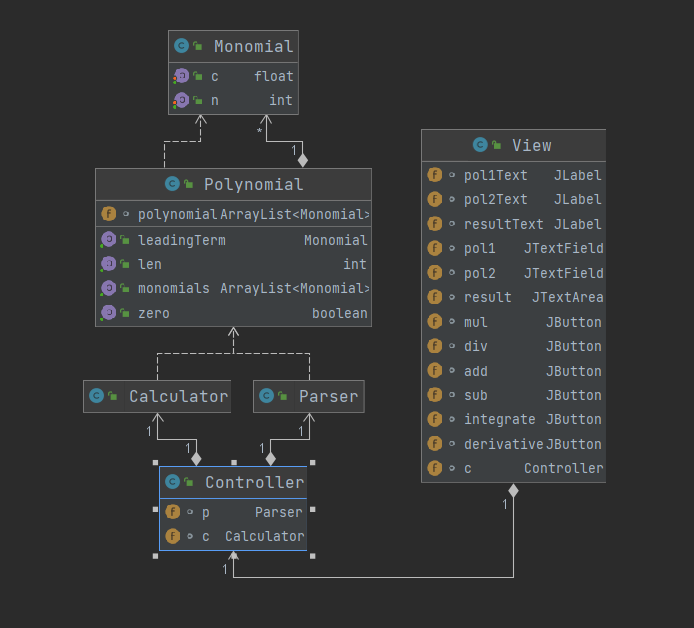
The calculator uses polynomials as input and returns polynomials as

output.

The view class takes care of the graphical user interface.

As a result, the Monomial class should contain 2 private variables, a float for the coefficient and an integer for the power, along with mutators and accessors. Moreover, it should implement the Comparable interface for quicker sorting. The polynomial class will consist of a collection of polynomials, a method that adds a monomial to the collection, a method to get the degree of the polynomial and another method to sort the polynomials in decreasing order based on the power of the monomials. The methods deleteRedundantTerms (terms with coefficient 0) isZero and getLeadingTerm will come in handy when performing the division.

The calculator class contains methods that take as parameters polynomials and return polynomials as output. In order to add two polynomials an adapted version of the common “Merge two sorted arrays” problem.



**4. Implementation**

**4.0 Data Structures**

The data structures used in this project are either primitive types, predefined data structures like ArrayList or objects such as Monomial and Polynomial.

The advantage of using ArrayList is that it manages memory more efficiently and is easier to perform addition and deletion on. Another advantage is that you can use several methods on them that are already implemented.

The algorithms used are Blomqvist's method for dividing polynomials and an algorithm similar to merging two sorted arrays.

**4.1 Packages**

Packages are used to group related classes. The three main packages used are the Model, the View, and the Controller.

This application is based on the Model View Controller architectural pattern. The View package handles the Graphical User Interface (everything that is shown onto the screen). The Model deals with the computational part of the application and transforming the user readable data into abstract data types (using regular expressions). The Controller acts as an intermediary between the View and the Model. It takes the data from the View and passes it to the Model. The Model then passes it back to the Controller and the Controller back to the View.

Classes:

Model:

-> Monomial

-> Polynomial

-> Calculator

-> Parser

View:

-> View

Controller:

-> Controller

**4.2 Class Implementation**

One of the main OOP principles is encapsulation. By definition, encapsulation describes the idea of bundling data and methods that work on that data within one unit, in Java we call it a class. This concept is also often used to hide the internal representation, or state of an object from the outside.

This project consists of 4 classes:

A) Monomial

- contains 2 private fields along with the corresponding getters and setters

- since it implements the comparable interface, it must also override the compareTo function

- it also overrides the toString() function

B) Polynomial

- addMonomial() - adds a Monomial to the ArrayList polynomial

- sortByPower() - sorts the list of Monomials based on their power

- getLen() - getter for the size of the list

- getMonomials() - returns the list of Monomials

- getLeadingTerm() - returns the Monomial with the maximum power

- isZero() - returns true if every coefficient is zero or the list is empty

- degree() - returns the degree of the polynomial (maximum power)

- toString() - makes use of the toString() function from the monomials

- deleteRedundantTerms() - removes monomial from the list if it’s coefficient is zero

C) Calculator

- solve() - first it sorts the monomials of the input polynomial by their power. Computes the sum of all monomial coefficients if the power of the current monomial matches the power of the previous monomial, otherwise it adds the sum so far and the power of the previous monomial to the result polynomial.

- addPolynomials() - first it normalizes the polynomials (using solve) and sorts them by power, it goes through every monomial as long as there are remaining monomials in both lists. If the powers of both monomials match, then add their coefficients and place them in a result polynomial otherwise add the monomial with the lesser power. If there are remaining monomials in any of the two lists add them to the result.

- subtractPolynomials() - same concept as addPoynomials

- derivative() - goes through each monomial and applies the mathematical formula for the first derivative if the power is nonzero

- integrate() - goes through each monomial and applies the mathematical formula to integrate a monomial if the power is nonzero

- multiply() - multiplies each monomial with each monomial then sorts them by power and normalizes the result

- divide() - based on the pseudocode from Blomqvist's method of dividing two polynomials it makes use of the previously implemented methods

D) Parser

- parsePolynomial() - returns null if the input String does not match the pattern for a polynomial. After that it extracts each monomial from the input String and adds them to the result polynomial

E) View

- functions to add the UI elements

- action listeners for the buttons

F) Controller

- functions of the type get<Operation> which uses the calculator and the parser to create new polynomials to return the resulting String to the view

**5. Results**

The application uses simple concepts from mathematics implemented using java. The testing was done using jUnit and does checks the following:

Addition:

Input: 2x^3+2x^2 ; 2x^3+2x^2

Output: +2.0\*x^3+3.0\*x^2+1.0\*x^1

Expected output: +2.0\*x^3+3.0\*x^2+1.0\*x^1

Subtraction:

Input: 2x^3+2x^2 ; 2x^3+2x^2

Output: -2.0\*x^3-1.0\*x^2+1.0\*x^1

Expected output: -2.0\*x^3-1.0\*x^2+1.0\*x^1

Multiplication:

Input: 2x^3+2x^2 ; 2x^3+2x^2

Output: +2.0\*x^5+4.0\*x^4+2.0\*x^3

Expected output: +2.0\*x^5+4.0\*x^4+2.0\*x^3

First Derivative:

Input: 2x^3+2x^2

Output: +6.0\*x^2+4.0\*x^1

Expected output: +6.0\*x^2+4.0\*x^1

Integration:

Input: 21x^6+12x^2+1\*x^1

Output: +3.0\*x^7+4.0\*x^3+0.5\*x^2

Expected output: +3.0\*x^7+4.0\*x^3+0.5\*x^2

Division:

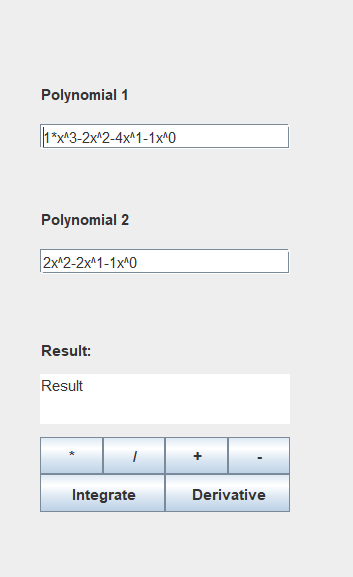
Input: 1x^3-2x^2-4x^0 ; 1x^1-3x^0

Output: +1.0\*x^2+1.0\*x^1+3.0\*x^0\n+5.0\*x^0

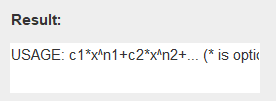
Expected output: +1.0\*x^2+1.0\*x^1+3.0\*x^0\n+5.0\*x^0

All tests passed successfully.

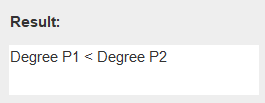
The GUI is the following.



If an invalid polynomial is passed into one of the two fields the end user is prompted with the message.



If the end user attempts division by zero or the degree of the polynomial denominator is greater than the degree of polynomial in the numerator, the user is prompted with the message.



**6. Conclusions**

This project was a great exercise in remembering the core OOP principles and enforcing existing ones such as object immutability (used for monomials).

As for the challenges I’ve faced throughout the project I strongly believe that the hardest one was implementing a division algorithm with the already existent methods of the calculator and the polynomial classes.

Another key aspect that I had to take into consideration was in the implementation part for the GUI. The choice between JavaFX and Swing was a difficult one but I decided to stick with Swing since I’ve used it in the past.

One last important lesson that I’ve learned during this project is that taking a break is absolutely necessary when in doubt.

**7. Bibliography**

* <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>
* <https://app.diagrams.net/>
* <https://en.wikipedia.org/wiki/Polynomial_long_division>
* <https://www.jetbrains.com/help/idea/work-with-maven-dependencies.html#generate_maven_dependency>
* <https://www.symbolab.com/solver/polynomial-equation-calculator>