Fundamental Programming Techniques

Assignment 1

Polynomial Calculator

Student: Filip Denisa-Mariana
Group: 30422
Semigroup: 1
Lecturer: Dr. Eng. Prof. Salomie Ioan
Teaching Assistant: Stan Ciprian
Faculty of Automation and Computer Science, TUCN
2020-2021

Table of Contents

1. Assignment Objective

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.

The secondary objectives of this assignment are:

Secondary Objective	Chapter in which it is detailed
Analyze the problem and identify requirements	2
Design the polynomial calculator	3
Implement the polynomial calculator	4
Test the polynomial calculator	5

1.1. Description of secondary objectives

- **Analyze the problem and identify requirements** – in the beginning, a problem is formulated into natural language. Prior to the design process, research into the theoretical background of the problem is needed. Afterwards, the problem needs to be broken down into functional and non-functional requirements, alongside with the possible scenarios and use cases that arise.
- **Design the polynomial calculator –** the solution needs to firstly be designed so that its implementation is one of the best ones possible. There are many steps to be completed before starting to implement a problem, such as choosing the optimal architectural patterns and design patterns, establishing the relationships between classes, the algorithms and data structures to be used and so on.
- **Implement the polynomial calculator** – during this step, the classes with their respective fields and methods are created according to the previously chosen design.
- **Test the polynomial calculator** – the testing of an application is just as important as designing and implementing it. Testing assures the absence of problems in logic and code and it helps to provide the user a fully functional application.

2. Problem analysis, modeling, scenarios, use cases

2.1. Problem Analysis

The problem identified is the construction of a polynomial calculator that supports arithmetic between polynomials, such as addition, subtraction, multiplication, division, differentiation and integration. These operations require either one or two polynomials as parameters. Moreover, for simplicity, it was specified that the input polynomials will contain:

- only integer coefficients;
- only one indeterminate.

To understand the specifics of the given problem, a theoretical background of the required terms is given.

In mathematics, a polynomial is an expression consisting of variables (also called indeterminates) and coefficients, that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponentiation of variables. An example of a polynomial of a single indeterminate x is x^2-4x+7. (Wikipedia)

Generally, a polynomial is of the form $P(X) = a_n * X^n + a_{n-1} * X^{n-1} + \cdots + a_1 * X + a_0$, where a_n, ..., a_1 are the coefficients, X is the indeterminate and n is the degree of the polynomial.

A monomial is an expression in algebra that contains only one term, for example 2x. It can contain one or multiple indeterminates. It is, roughly speaking, a polynomial with only one term. In the same manner, a polynomial is constructed from multiple monomials.

2.2. Modeling

The polynomial calculator is meant to be used by an external user, since computing these operations on paper can prove to be difficult and time consuming. The user will input the desired polynomials, choose the required operation and view the result, all on the same interface. The polynomials will be introduced by the means of the personal keyboard of the user.

Regarding the operations that require two parameters (addition, subtraction, division and multiplication), the user will input two polynomials. On the other hand, for the operations that need just one parameter (differentiation and integration), the user can choose whether to insert one or two polynomials. If two polynomials are entered, then the chosen operation will be performed *individually* on each of them.

Additionally, for division, two slots will be generated for the solution – one for the quotient of the division, and the other for the remainder.

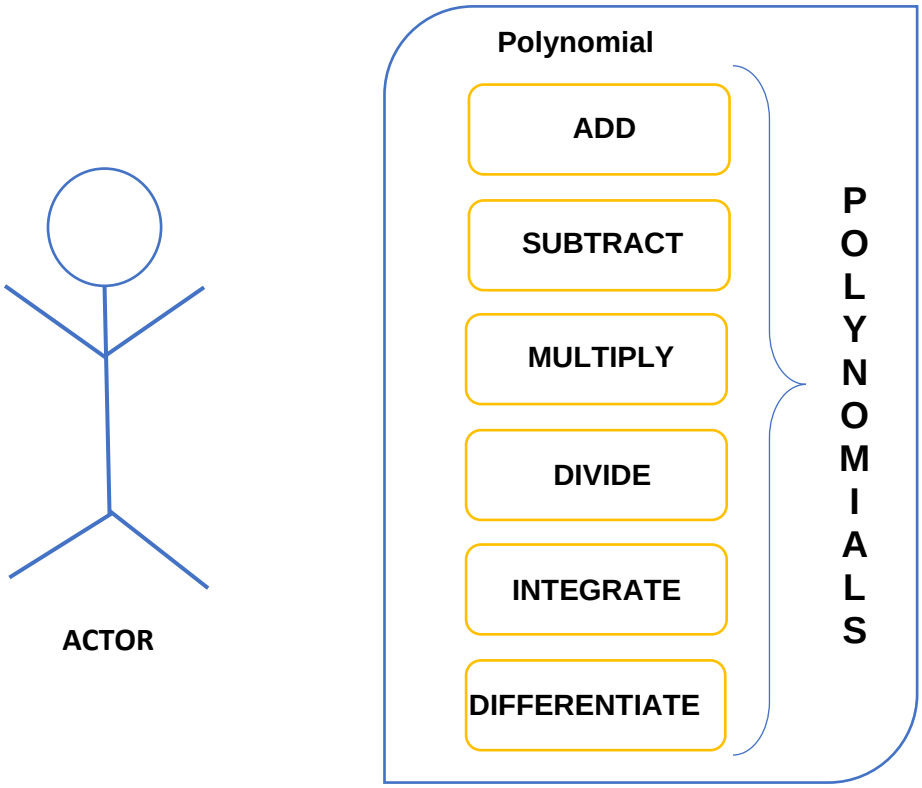
If the user wishes to perform operations on other polynomials, they have the possibility to clear the polynomial slots and restart the entire process.

2.3. SCENARIOS

When utilizing the polynomial calculator, a few different scenarios can arise. The user can input valid polynomials, press the button of the desired operation and the result will be automatically generated on the reserved text field. When starting another operation, the user will press the clear button for resetting the calculator.

However, a user can also introduce incorrect polynomials which demands the verification and validation of the user input. In another scenario, the user opens the polynomial calculator and closes it immediately.

2.4. USE CASES



Use case: add/subtract/multiply/divide polynomials
Primary actor: user
Main Success Scenario:

- the user inputs two polynomials from their personal keyboard;
- the user presses the button Add/Subtract/Multiply/Divide;
- the polynomial calculator performs the desired operation between the two inserted polynomials;
- the result is generated and displayed under the Result label, when the button is pressed.

Alternative scenario:

- the user inserts incorrect polynomials;
- an alert dialog window pops up, informing the user of their mistake;
- the user has to insert the polynomials once again, thus repeating the process.

Use case: integrate/differentiate polynomial
Primary actor: user
Main Success Scenario:

- the user inputs one or two polynomials from their personal keyboard;
- the user presses the button Integrate/Differentiate;
- the polynomial calculator performs the desired operation – on either one or two polynomials, individually;
- the result/results is/are generated and displayed under the Result label, when the button is pressed. If two polynomials were inserted, two Result text fields will appear.

Alternative scenario:

- the same as in the previous use case.

After defining some use cases, the following functional and non-functional requirements were identified:

Functional Requirements	Non-functional requirements
insertion of polynomials by user	an intuitive and easy-to-use calculator
selection of arithmetic by user	a Clear button that resets the calculator
computing the selected mathematical operation	a pleasant to look at interface
displaying the result/results	
notifying the user if incorrect polynomials were inserted	
handling exceptions, such as division by 0	

3. Design

3.1. Design Decisions

To divide my project into intuitive packages, I have chosen to implement the MVC (Model View Controller) architectural pattern. MVC is a interactive system pattern that divides a project into three categories: input, output and process. I will describe each of them individually:

- Under the view package resides a single .fxml file, corresponding to the GUI of the polynomial calculator (output);
- Under the controller package lies a single Controller java class, since each view is associated with only one controller;
- Under the model package I have created my own classes that model the problem given in natural language into code. This package is further divided into Operations, Exceptions and Comparators, alongside with the Polynomial and Monomial classes.

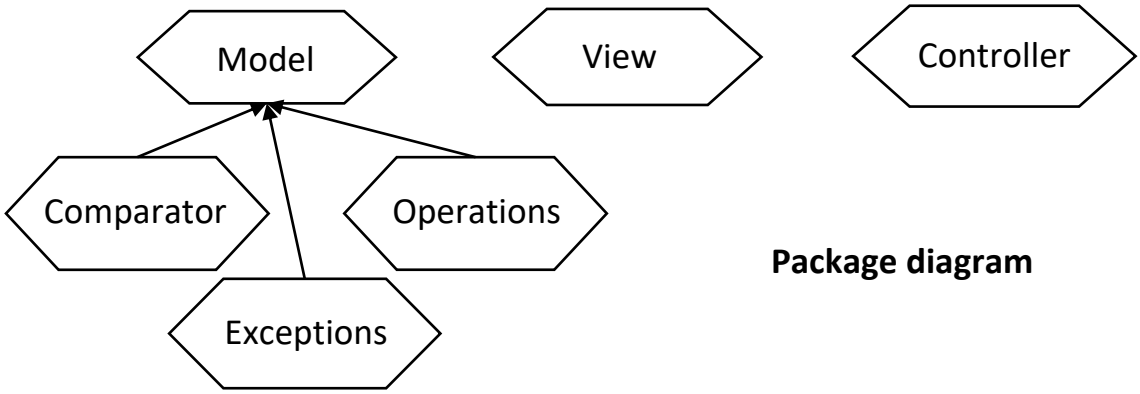
The main classes of the project are Monomial and Polynomial. Their fields and methods describe the real-life behaviour of these mathematical concepts. These classes will be further described under the Data Structured and Implementation sections.

The required mathematical operations that I have implemented have been structured into classes according to the **Strategy behavioural design pattern**. This pattern allows the behaviour or algorithm of a class to be changed at runtime. To integrate this pattern into my project, I have created two interfaces corresponding to Operations with One Parameter (for integration and differentiation) and Operations with Two Parameters (for addition, subtraction, multiplication).

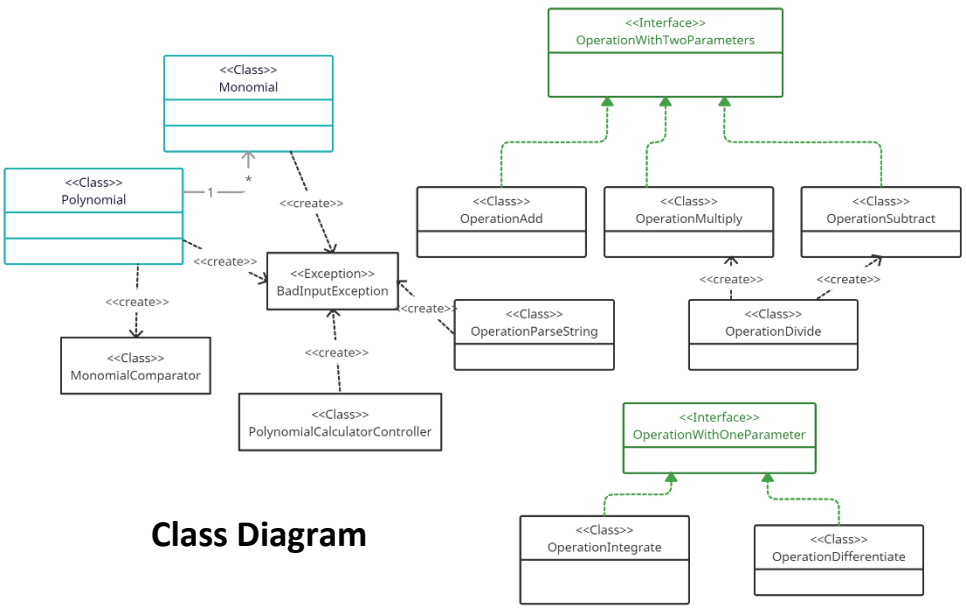
I also have created separate classes for the Division operation and for parsing a string into its corresponding Monomial/Polynomial object. This is due to the fact that they have different return values from the aforementioned operations, so they couldn't have implemented the two interfaces.

Splitting each operation into its own class helped with the unit testing. It also avoids clustering classes with too many tasks and it improves the over-all understandment and readability of the project.

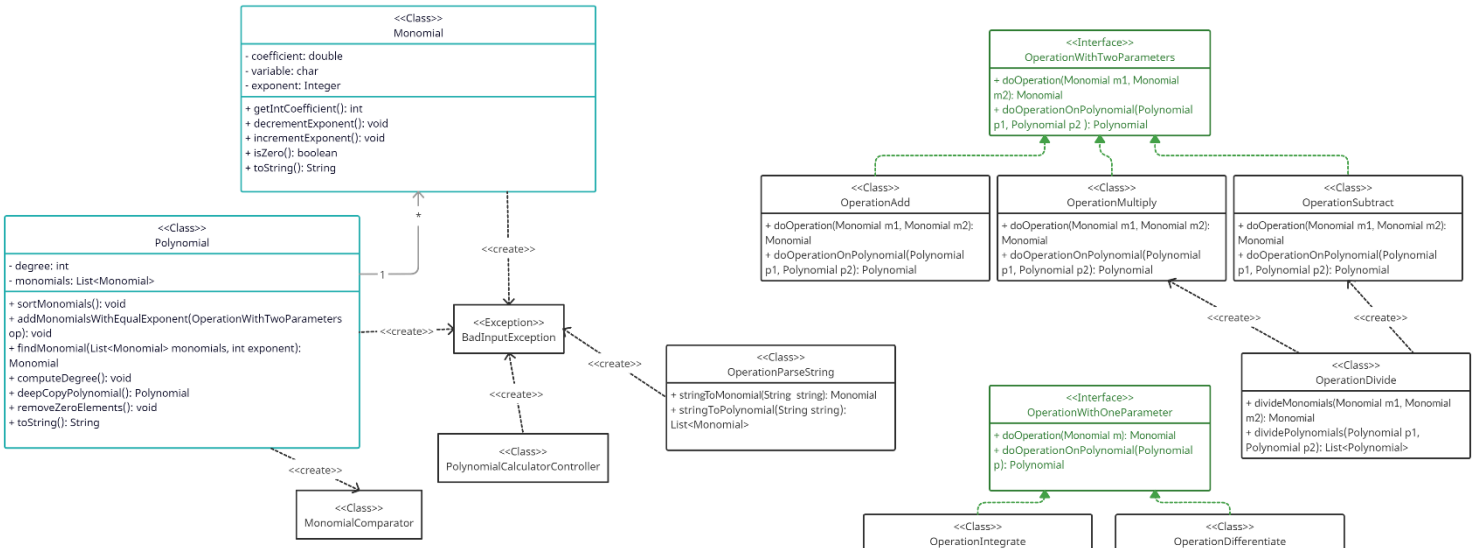
3.2. UML DIAGRAMS



Package diagram



Class Diagram



Class Diagram with Fields and Methods

3.3. Data Structures

All throughout the project, I have used the List and ArrayList Collections, replacing the conventional arrays. I have chosen to do so because an ArrayList is a variable length, resizing each them an element is removed or added to the Collection. At the same time, the removal of objects is faster for the ArrayList.

Regarding my own data structures, I have created Monomial and Polynomial. These objects correspond to the mathematical notations and are the center of the project. They are the data types on each every operation is performed.

3.4. Class and Interface Design

Each class contains some of the following:

- private fields;
- constructors (with and without arguments);
- methods;
- the overriden toString, equals and hashcode methods, where it was needed;
- the getters and setters were generated through the Project Lombok library.

The interfaces OperationWithOneParameter and OperationWithTwoParameters contain the methods that will be overriden by the classes that implement them.

3.5. Relationships

There exist Is-A relationships between the OperationWithOneParameter and OperationWithTwoParameters interfaces and the classes that implements them.

Between the classes Polynomial and Monomial there is a Has-A relationship (aggregation). Since Polynomial uses a List of Monomials, the relationship is one to many (one polynomial has

multiple monomials). There also are some dependency relationships between classes, especially when it comes to the Operations classes and those who instantiate objects of their type.

3.6. ALGORITHMS

For each operation performed on polynomials, an algorithm is used. Whereas addition, subtraction, multiplication, integration and differentiation are quite trivial, the division operation is more complicated.

$$P(X) = a_n * X^n + a_{n-1} * X^{n-1} + \cdots + a_1 * X + a_0$$

$$Q(X) = b_n * X^n + b_{n-1} * X^{n-1} + \cdots + b_1 * X + b_0$$

Addition: $P(X) + Q(X) = (a_n + b_n) * X^n + (a_{n-1} + b_{n-1}) * X^{n-1} + \cdots + (a_0 + b_0)$

Subtraction: $P(X) - Q(X) = (a_n - b_n) * X^n + (a_{n-1} - b_{n-1}) * X^{n-1} + \cdots + (a_0 - b_0)$

Multiplication: $P(X) \cdot Q(X) = X^{2n} \cdot a_n \cdot b_n + X^{2n-1} \cdot a_n \cdot b_{n-1} + \cdots + a_0 \cdot b_0$

Differentiation: $(cx^n)' = c \cdot n \cdot x^{n-1}$

Integration: $\int cx^n dx = c \cdot \frac{x^{n+1}}{n+1} + C$

For implementing the division operation, I have used the following pseudocode:

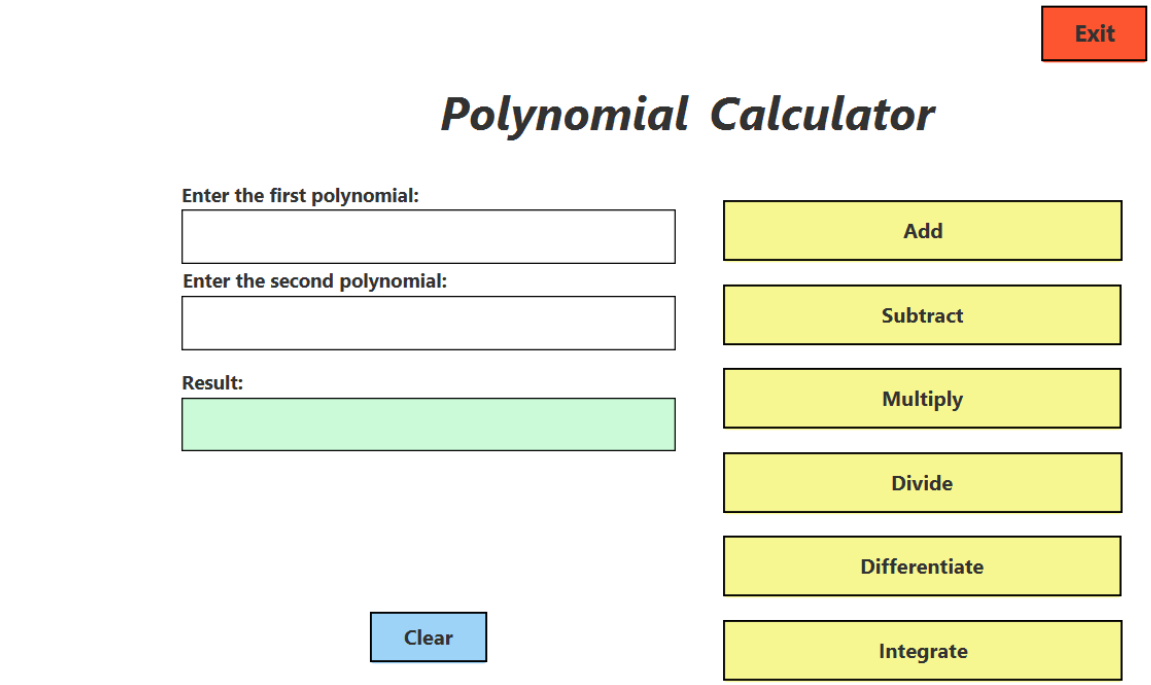
```
function n / d:
    require d ≠ 0
    (q, r) ← (0, n)                          //At each step n = d × q + r
    while r ≠ 0 AND degree(r) ≥ degree(d):
        t ← lead(r)/lead(d)                  //Divide the leading terms
        (q, r) ← (q + t, r - (t * d))
    return (q, r)
```

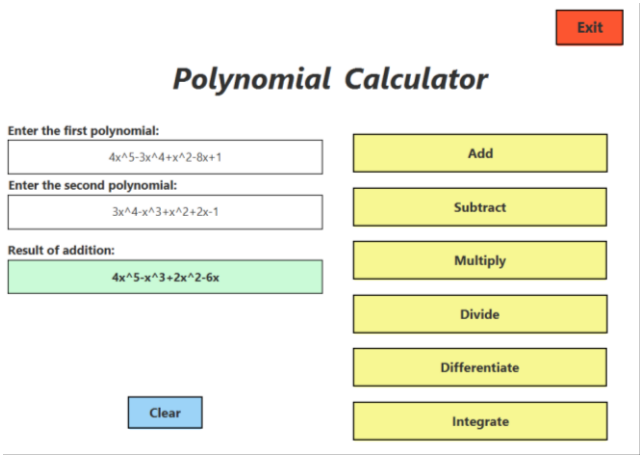
- where: n – numerator, d – denominator, q – quotient, r – remainder (these are polynomials). If the denominator is 0, then an ArithmeticException is thrown. Otherwise, the quotient and remainder polynomials are initialized and the division operation is performed.

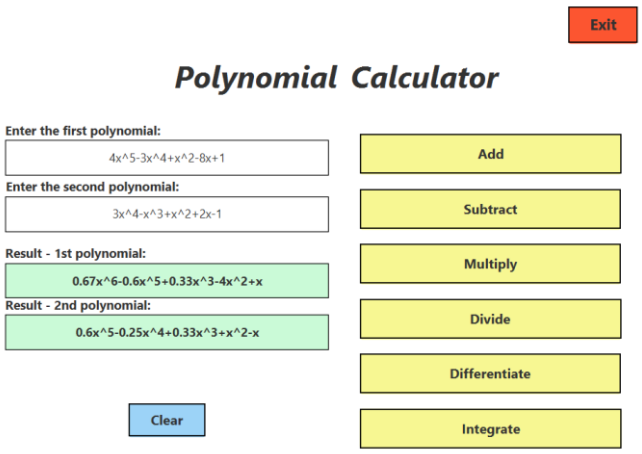
3.7. USER INTERFACES

The Graphical User Interface was implemented using JavaFX and its tool, SceneBuilder. I have chosen the JavaFX software platform for creating and delivering desktop applications because it is constantly improved and it will replace the standard GUI library for Java SE in the forseeable future.

I have opted for a simple and intuitive design, with just one window for the entire application. The Result text field and label are changed based on the operation that was performed. In the case of an addition, it will display „*Result of addition*". In the case of integration, if both polynomials were inserted, two results will appear – „*Result – 1st polynomial*" and „*Result – 2nd polynomial*".



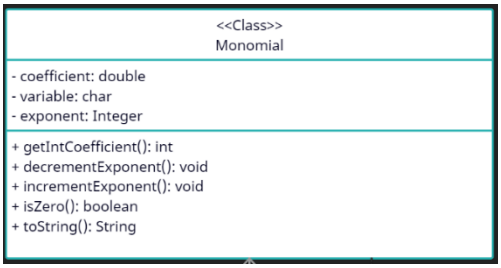




4. Implementation

I will present each class sequentially, describing its most important fields and methods:

1) Monomial



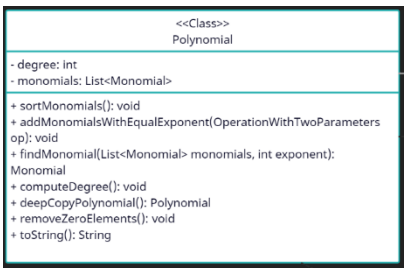
Fields:

- **coefficient** – the coefficient of the monomial. Despite having as input only polynomials with integer coefficients, during operations such as division and integration, the resulting coefficient can have a real value. This is why it is of type double.
- **exponent** – exponent of the monomial;
- **variable –** the indeterminant of the monomial.

Methods:

- **getIntCoefficient()** – returns the integer value of the double coefficient, through a cast;
- **toString()** – takes a Monomial and returns its string equivalent.

2) Polynomial



Fields:

- **degree** – the degree of the polynomial (the maximum exponent present in the polynomial);
- **monomials** – a list of monomials that form the final polynomial.

Methods:

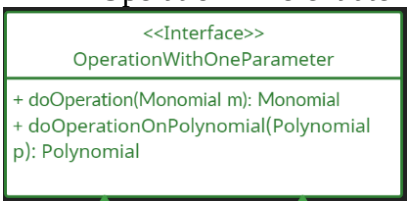
- **sortMonomials()** – sorts the monomials of the polynomial in descending order by their exponent;
- **addMonomialsWithEqualExponent()** – if two or more monomials with the same exponent are found in a polynomial, then they are added together in a single polynomial (e.g. $2X^3 + 4X^2 + X^3 + 1 = 3X^3 + 4X^2 + 1$);
- **findMonomial()** – finds a monomial by its exponent in a polynomial. If no such monomial exists, it returns null;
- **toString()** – takes a Polynomial and returns its string equivalent.

3) MonomialComparator

Methods: compare() – takes 2 monomials and returns the result of the comparison between their exponents. It is used to sort the monomials, in the method sortMonomials().

For the Operation classes, I have defined the same operation first on Monomials and then on Polynomials. Since a Polynomial is a List of Monomials, I believed it would be better to iterate through the list and call the operation on each Monomial.

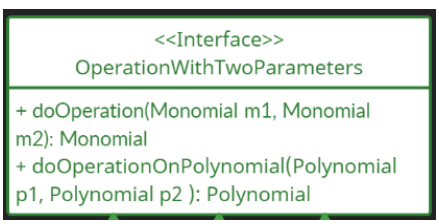
4) **OperationWithOneParameter** – Interface (classes OperationIntegrate, OperationDifferentiate implement it)



Methods:

- **doOperation(Monomial m)** – performs a specific operation on one monomial and returns the result;
- **doOperationOnPolynomial(Polynomial p)** – performs a specific operation on one polynomial and returns the result.

5) **OperationWithTwoParameters** – Interface (classes OperationAdd, OperationMultiply, OperationSubtract implement it)



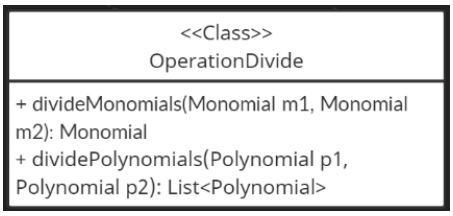
Methods:

- **doOperation(Monomial m1, Monomial m2)** – performs a specific operation between the two input monomials and returns the result;

- **doOperationOnPolynomial(Polynomial p1, Polynomial p2)** – performs a specific operation between the two input polynomials and returns the result.

These methods are not implemented in the interface. They are overriden in each class that implements the interface, the compiler deciding at runtime which version to perform (Strategy pattern).

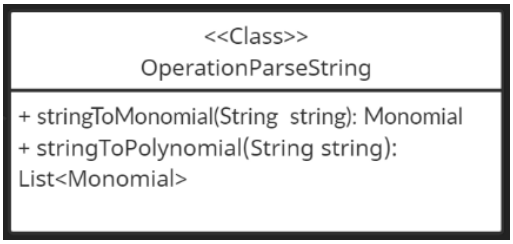
6) OperationDivide



Methods:

- **divideMonomials(Monomial m1, Monomial m2)** – divides the two input monomials and returns the result;
- **dividePolynomials(Polynomial p1, Polynomial p2)** – divides the two input polynomials and returns a list of Polynomials, containing the quotient and remainder of the division.

7) OperationParseString



Methods:

- **parseStringToMonomial(String string)** – takes a string and parses it into a Monomial object.

This is done by the means of a regular expression (regex):

```
(?!$)([+-]?+(?=\\w)\\d*)?+\\*?+((?<=\\d|[+-]|\\*)[a-zA-Z]?+|^[a-zA-Z])(
(?:(?<=[A-Za-z])\\^)?+((?<=\\^)\\d*)?+'
```

I will break it up and explain it in a table:

Regular Expression	Explanation
(?!$)	This negative lookahead checks to see if the end of the line was reached – since each group of a monomial is optional, this stops empty strings from being matched.
([+-]?+(?=\w)\d*)?+	The first group of the regex – responsible for matching the coefficient: the +/- sign is matched once or not at all (possessive quantifier), only if it is followed by a word character (through the positive lookahead (?=\w) – to stop +/- from being matched on their own. Afterwards, a digit is expected zero or more times (greedy quantifier). This entire group can appear once or not at all.
*?=	The multiplication sign can be found once or not at all. It is optional, since, for example, 2x is the same as 2*x.

((?<=\d|[+-]|*)[a-zA-Z]?+|^[a-zA-Z])	The second group of the regex – responsible for matching the indeterminate: through the negative lookbehind, a letter is matched only if it is preceded by a digit, a +/- sign or the * sign. Otherwise, it will match the indeterminate only if it is at the beginning of the string.
(?:(?<=[a-zA-Z])\^)?+	This is a non-capturing group. It checks for the power ^ sign, only if it is preceded by an indeterminate. This group can be matched once or not at all, since it is not mandatory for an exponent to exist.
((?<=\^)\d*)?+	The third group of the regex – responsible for matching the exponent. The exponent is matched only if it is preceded by the ^ sign, through a positive lookbehind. A digit is expected zero or more times, but the entire group can be matched once or not at all.

- **parseStringToPolynomial(String string)** – takes a string and parses it into a Polynomial object. At the beginning, each space from the string is taken out. Afterwards, the string is parsed into monomials and the method parseStringToMonomial() is called for each of them. The splitting is done by the means of a regular expression.

```
(?=[^A-Za-z0-9^*]|^[A-Za-z0-9])
```

The regex splits the string using a positive lookahead. It searches for other characters beside letters, digits and the ^/* signs (specifically, in the case of correct polynomials, it looks for the +/- signs). If no such signs were found, it tries to look at the beginning of the string, to detect a letter or a digit (e.g. x^2+x or 3x^3+5). The found matches are split into an array of strings.

8) BadInputException – extends Exception

This class is a custom exception, that gets thrown whenever a bad input was detected (either by the previously explained regular expressions, or from manually added data to constructors). The exception is handled in the PolynomialCalculatorController, where a custom alert pops up in a dialog window if the user inserts invalid characters.

The GUI

For the GUI I have used an .fxml file containing the view (the aspect) of the Graphical User Interface and the PolynomialCalculatorController class, the combines the model and the view of the project.

For containers, I have used GridBoxes. I have found that it is easier to arrange controls on the canvas with GridBoxes and they get better resized if the dimensions of the windows are manually changed by the user. As for controls, I have Text Fields, Labels and Buttons. The buttons are responsible for every action that can be performed by the user. When a button is pressed, its associated function starts running and the code is executed.

5. Results

Regarding testing, I have used the Junit framework, which I added to the project through Maven. A separate test class was generated for each class, where every method is tested.

The convention used for naming test methods was test_methodName_nameOfTestedScenario.

For the operations performed on monomials and polynomials, I have tested cases such as positive/negative monomials/polynomials, performing an operation when a monomial/polynomial is 0, testing if an operation is commutative or not, performing an operation between a monomial/polynomial and a constant, performing an operation when the exponent was the same/different.

For the Polynomial and Monomial classes, I have tested each method by providing the correct input and comparing it to the output of the method. For example, for the addMonomialsWithEqualExponent() method, I have tested the following cases: when monomials with equal exponent were present in the same polynomial, and when they were absent. The successful tests proved that the method was indeed functioning as envisioned.

6. Conclusions

After completing the Polynomial Calculator, I have managed to gain new knowledge regarding design patterns, architectural patterns and over-all OOP concepts. I learned the importance of unit testing and how it should be implemented, and I found new Java libraries, such as Lombok, that can simplify a few tasks (writing getters/setter, constructors and so on). I improved my JavaFX skills, even if the GUI was a simple one for this calculator. I have also gained an insight in working with GitLab, since I have only used GitHub before.

Polynomial arithmetic is time-consuming when done on paper and I have enjoyed automating it for future uses.

Further developments:

- displaying each step of an operation, making it easier for the user to understand the mathematics behind the arithmetic;
- performing operation on polynomials with double coefficients and multiple indeterminates;
- allowing for multiplications to be performed inside a polynomial (e.g. 3*5x+5*6 = 15x+30).

7. Bibliography

Fundamental Programming Techiniques – *Lecture Slides*: Dr. Eng. Prof. Salomie Ioan

Fundamental Programming Techiniques – *Assignment One Support Presentation*: Dr. Eng. Prof. Salomie Ioan

Wikipedia – *Polynomial*: https://en.wikipedia.org/wiki/Polynomial

YourDictionary – *Example of Monomials*: https://examples.yourdictionary.com/examples-of-monomials.html

TutorialsPoint – *Design Patterns* – *Strategy Pattern*: https://www.tutorialspoint.com/design_pattern/strategy_pattern.html

Guru99 – *UML Relationships*: https://www.guru99.com/uml-relationships-with-example.html

Refactoring Guru – *Strategy in Java*: https://refactoring.guru/design-patterns/strategy/java/example