DOCUMENTATION

ASSIGNMENT 1 - POLYNOMIAL CALCULATOR

NUME STUDENT: Bumbuc Ioana

GRUPA: 30421

# CUPRINS

[1. Assignment’s objective 3](#_Toc95297885)

[*2.* Problem analysis, modelling, scenarios, use-case*s 3*](#_Toc95297886)

[3. Design 3](#_Toc95297887)

[4. Implementation 3](#_Toc95297888)

[5. Results 3](#_Toc95297889)

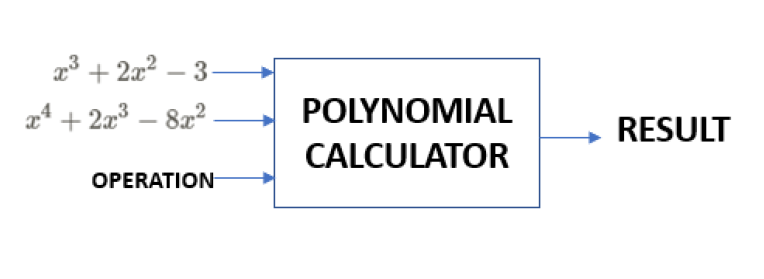
[6. Conclusions 3](#_Toc95297890)

[7. Bibliography 3](#_Toc95297891)

# Assignment’s Objective

The main objective of this assignment is to design and implement a polynomial calculator that performs operations on polynomials introduced by the user, through the graphical interface. The application should take one or two polynomials as input and display the result of a chosen operation.

# Problem analysis, modelling, scenarios, use-cases



* 1. **Functional Requirements**

The application should be able to read and process input provided by the user. The input must respect the mathematical syntax of type “ax^n”, where “x” is the variable, “a” is the coefficient and “n” is the degree of the monomial. If the input is given incorrectly, the application must display an error and not perform any computations. The user can select the operation they want to perform on the introduced input.

The operations that the calculator can perform are addition, subtraction, multiplication, division, which take two polynomials as input, and differentiation and integration, which apply on one polynomial. After computing a result, the application must display it.

* 1. **Theoretical Considerations**
     1. **Polynomial definition**

In mathematics, a polynomial is a mathematical expression consisting of variables and coefficients. Variables appear at a non-negative integer exponent and coefficients considered for this project have integer values. An example of a polynomial with one variable is *x*2 − 3*x* + 4.

A monomial is one term of a polynomial. It has a coefficient and a variable at some exponent. The algebraic sum of one or more monomials is a polynomial.

* + 1. **Polynomial addition and subtraction**

Polynomials are added (and subtracted) using associative law of addition, and they are grouped according to their power. For example:

* + 1. **Polynomial division**

The division operation is the only one that provides two polynomials as output – one quotient and one remainder. If A is the dividend and B is the divisor, which must be nonzero, then A/B = Q and R, where Q is the quotient and R is the remainder. Thus, A can be written as A=BQ+R, where the degree of R must be lower than the degree of B, and Q and R are unique results. If R=0, that means that B is a *factor* of A. The algorithm for polynomial division A/B is the following:

1. Divide first term of A to the first term of B. First term is the term that appears at the highest power in the polynomial, defining its degree. Result of the division is one term of the quotient Q.
2. Multiply B with the result previously obtained (Q).
3. Subtract from A the result obtained in step two.
4. Repeat the first three steps but change A to the last result obtained after each repetition.
5. Repeat until you cannot longer divide by B, that is until the first three steps produce a polynomial with the degree smaller than the degree of B.
6. The quotient is Q (obtained by doing the algebraic sum of all results obtained at step 1), and the remainder R is the polynomial resulted after step 5.

Pseudocode for polynomial division is the following:

**Divide (A, B)**

Q ← 0

R ← A

**while** R ≠ 0 **and** degree(R) ≥ degree(B) **do**)

Q ← Q + lead(R) / lead (B) // Divide the leading terms

R ← R – (lead(R)/lead(B)) × B

**return** (Q, R)

* + 1. **Polynomial multiplication**

Multiplying two polynomials is the algebraic sum of the products of each element of the first polynomial with each element of the second polynomial. After that, the result might need to be simplified, that is computing the final coefficient of each power that appears, in case it appears more than once in the previous step. For example:

* + 1. **Polynomial differentiation and integration**

Differentiating a polynomial works by differentiating each monomial with the formula. The result of each monomial differentiation is added together to form the result. For example:

Same process applies for integrating a polynomial.

* 1. **Use Cases**

The use-cases of the application have to do with computing the result of a chosen operation. They can be divided into two categories, which work the same for each operation in that category.

* + 1. **Operations that take 2 polynomials as parameters and display one polynomial as result – Addition, Subtraction, Multiplication**

Example of use-case for addition:

Primary actor: the user

Main success scenario:

* User inserts two polynomials – one in each field
* Selects “Add” from drop down menu
* Clicks “Calculate”
* Correct result of addition appears after the “Result:” field

Alternative Sequence: Invalid input

* An exception appears
* Issue is resolved when user corrects the given input, in which case the correct scenario will happen.
  + 1. **Operations that take one polynomial as input and display one polynomial as result – Differentiation, Integration**

Example for Differentiation:

Primary actor: the user

Main success scenario:

* User inserts one polynomial in the first field
* Selects “Differentiate” from drop down menu
* Clicks “Calculate”
* Correct result of differentiating the input appears after the “Result:” field

Alternative Sequence: Invalid input

* An exception appears
* Issue is resolved when user corrects the given input, in which case the correct scenario will happen.
  + 1. **Operations that take two polynomials as input and display two polynomials as result – Division**

Primary actor: the user

Main success scenario:

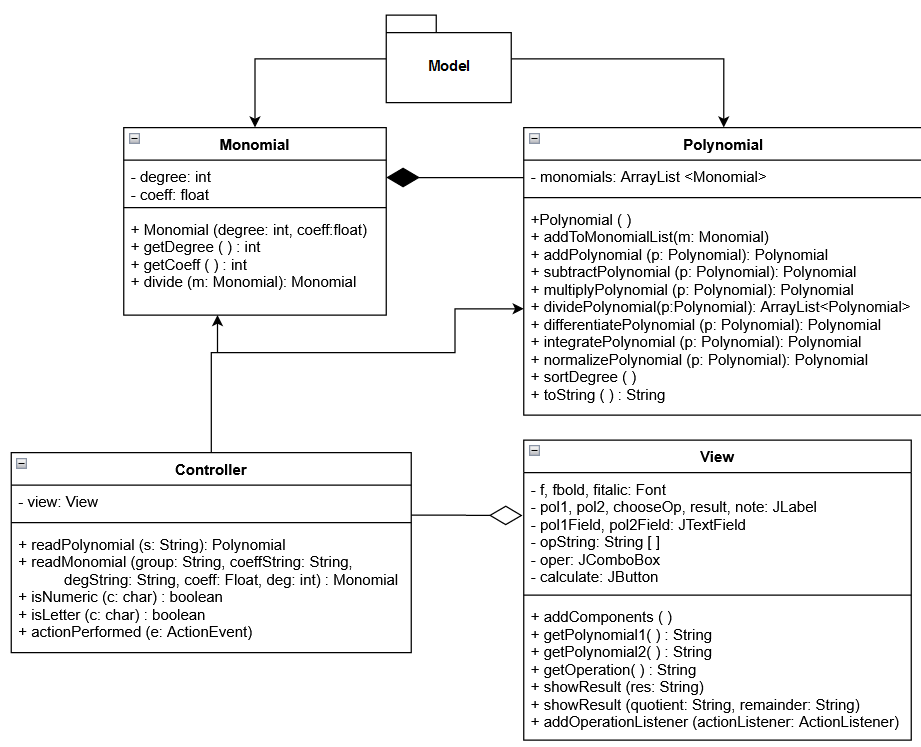
* User inserts two polynomials – one in each field
* Selects “Divide” from drop down menu
* Clicks “Calculate”
* Correct result of dividing first polynomial to the second appears on the visual interface, after the “Quotient” and “Remainder“ labels.

Alternative Sequence: Invalid input

* An exception appears
* Issue is resolved when user corrects the given input, in which case the correct scenario will happen.

# Problem Design

The design of this problem follows the Model View Controller Architecture. Each element of MVC makes up a package.



* 1. **The Model Package**

The Model is composed of two classes, Monomial and Polynomial. Monomial class contains as fields a coefficient and a degree. The coefficient is represented as a float, while the degree is an integer value (int). The classes interact with each other in the sense that a Polynomial contains a list of Monomials.

All operations on polynomials are defined inside the Polynomial class, as a function that is called on an object of class Polynomial and can take one (or no) Polynomial as parameter. The purpose of the model is to provide the lowest level of maintaining data relevant to the application. It is responsible for computing results, retrieving parameters from the controller, and passing results back to the controller.

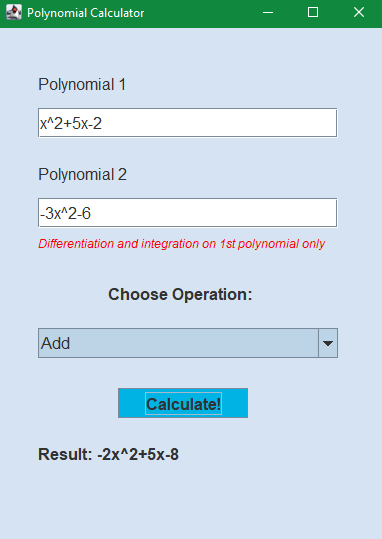
When designing the model, I had to take into account the theoretical considerations of the wanted result, that is polynomial computations and division algorithm defined in section 2.2.

* 1. **The Controller**

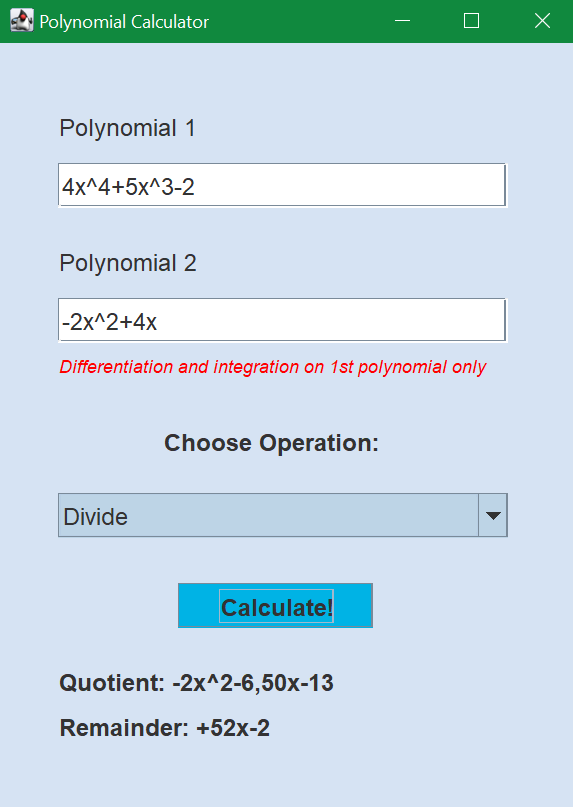
The controller is the means of interaction between the user and the model. It retrieves data that the user introduces through the graphical interface and sends commands to the model to execute in order to provide with the wanted results. The controller deals with transforming the user input from a string of characters to objects of classes define in the model that can be used to make computations. It provides functionalities for the “Calculate” Button on the user interface, taking a command and passing the output to the view to be displayed.

* 1. **The view**

The view represents the graphical interface of the application. It provides data representation, generating a user-friendly environment where the user doesn’t have to see the works behind the result obtained. The view is comprised of graphical components, implemented using the swing GUI widget toolkit. The components used are labels for visualizing polynomials, text fields for the user to enter polynomials, a button to trigger the computation of results and a drop-down box for choosing the wanted computation. View receives information to display from the controller and sends strings of characters representing polynomials to the controller for it decode.

O imagine care conține text

Descriere generată automat



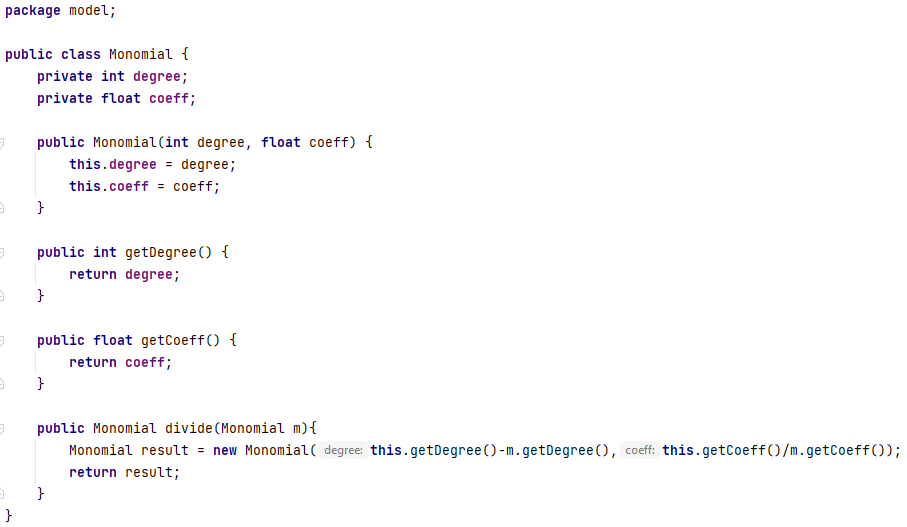
# Implementation

The implementation of the project is represented by four main classes: Monomial, Polynomial, Controller and View.

* 1. **Monomial**

Monomial Class represents the core of the problem. A monomial is represented using two fields: coefficient of type float and degree of type int. A list of one or more monomials represents a polynomial. The Monomial class contains the constructor which takes the degree and the coefficient as parameters, as well as getters for those fields.

The “divide” method implements the division of two monomials, where the degree of the result takes the value of the difference between the two degrees, while the coefficient takes the value of the division.



* 1. **Polynomial**

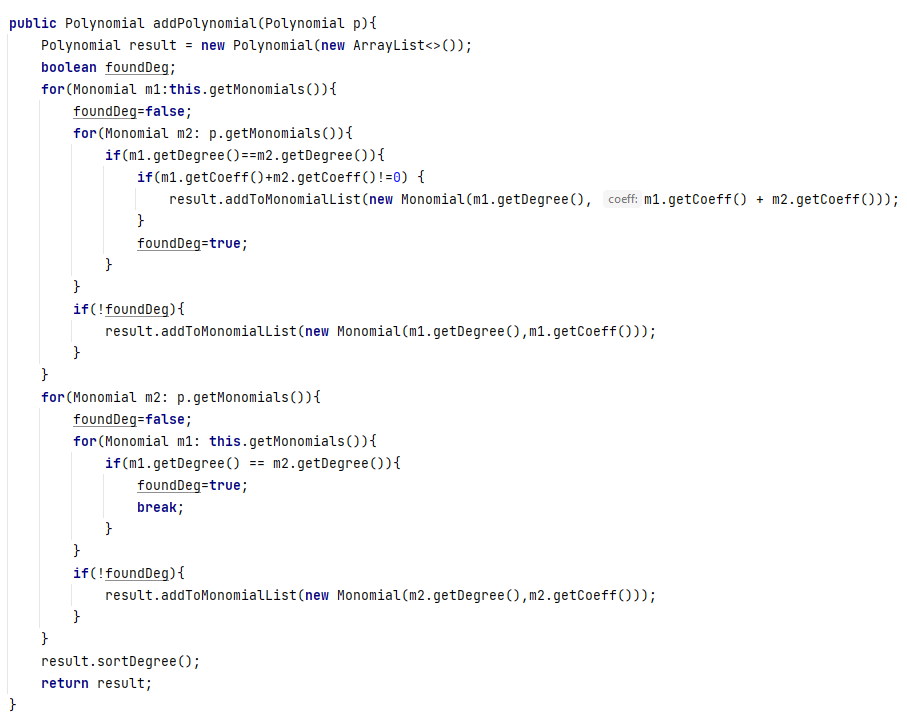
The Polynomial class has a list of monomials on which all the operations are implemented. The class consists of a constructor, a getter and a setter, and methods for implementing the needed operations.

* + 1. addToMonomialList (Monomial m)

The method takes a monomial as parameters and adds it to the list of monomials held by the polynomial on which it is called. This method is used for simplicity of adding an element to the list without calling the getMonomials ( ) method.

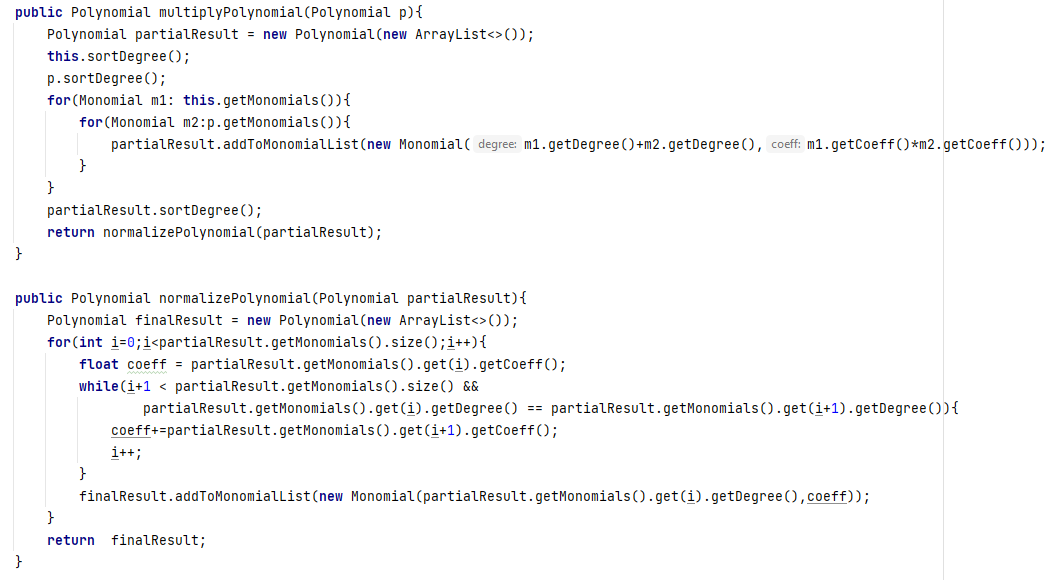
* + 1. addPolynomial (Polynomial p), subtractPolynomial (Polynomial p)

The idea of these methods is performing operations on two polynomials, taking them term by term. First, it goes through all elements of polynomial p1. For each monomial p1, it compares it with all monomials of polynomial p2. If a degree match is found, the coefficients are added (or subtracted) and the result is added to the result’s monomial list. At the end of the for-each loops, it goes one more time through all monomials of the second polynomial, in case some monomials were missed. This case can happen if there are monomials in the second polynomial whose degree do not appear in the first, and this is a verification step.



* + 1. multiplyPolynomial (Polynomial p)

The method multiplies each monomial of the first polynomial to each monomial of the second, using two for-each loops, ensuing a partial result. When multiplying two monomials, the degree of the result is the sum of degrees, while the coefficient is the product of coefficients. Afterwards, it calls the normalizePolynomial method on the partial result, which adds together the coefficients of monomials which appear multiple times with the same degree.

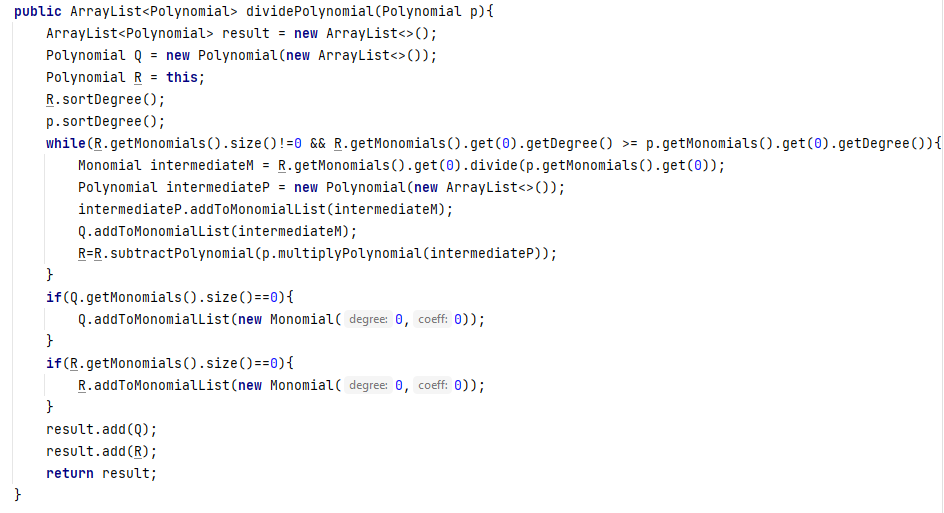


* + 1. differentiatePolynomial ( ), integratePolynomial ( )

These methods apply the rules for differentiating and integrating a polynomial. They do not take parameters, as they perform computations on the object on which the method is called.

* + 1. dividePolynomial (Polynomial p)

This method performs division between two polynomials and returns an array list of polynomials, containing the quotient and the remainder after dividing. The method follows the pseudocode described in section 2.2.3.



* + 1. toString ( )

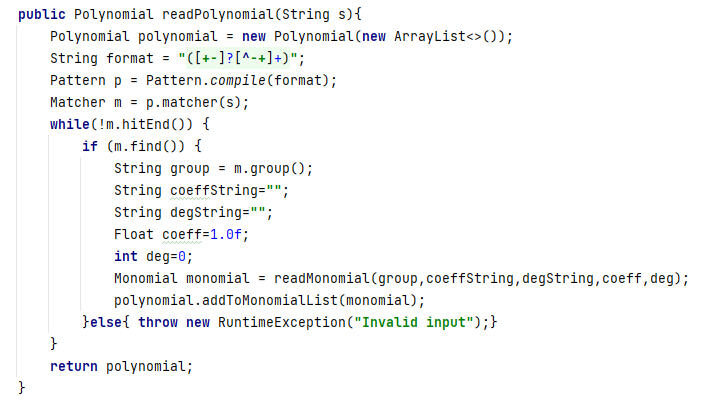
The method overrides the default toString and it is called on an object of class Polynomial, and it returns a string of characters that represent a polynomial. This method is necessary because a polynomial is stored as a list of ints and floats, and it is needed to visualize in natural language the result of computations.

* 1. **Controller**

Controller class has one field of type View, and methods necessary for creating the connection between the user interface and the model. Those methods refer to translating input from string to object of type Polynomial, as well as providing functionality to the button and the selected item from the drop-down menu.

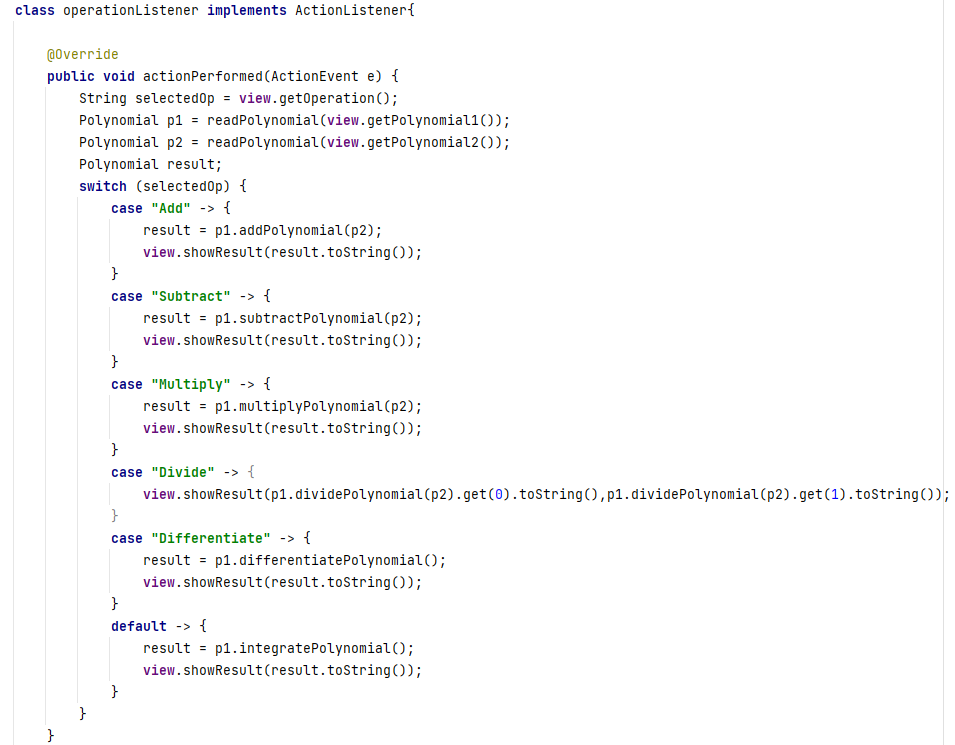
* + 1. readPolynomial (String s), readMonomial (String group, …)

This method uses regular expressions to separate the input string s into groups, each representing a monomial. The input is written in the form “ax^n”, where “a” is the coefficient, “x” the variable and “n” the power. Using Pattern and Matcher, the input is divided into groups on which the readMonomial method is called.



* + 1. actionPerformed (ActionEvent e)

The Controller class has an inner operationListener class implementing the ActionListener interface. The purpose of this class is to define the functionality of the “Calculate” button. The controller takes from the View the operation selected by the user, using a getOperation ( ) method. The controller chooses what method to call from the Polynomial class by comparing the string it receives from the user (“Add”, “Subtract” …) and provides the output polynomial as a string to the View. Depending on the scenario, the output could be either two polynomials for division, or one polynomial for the other operations.



* 1. **View**

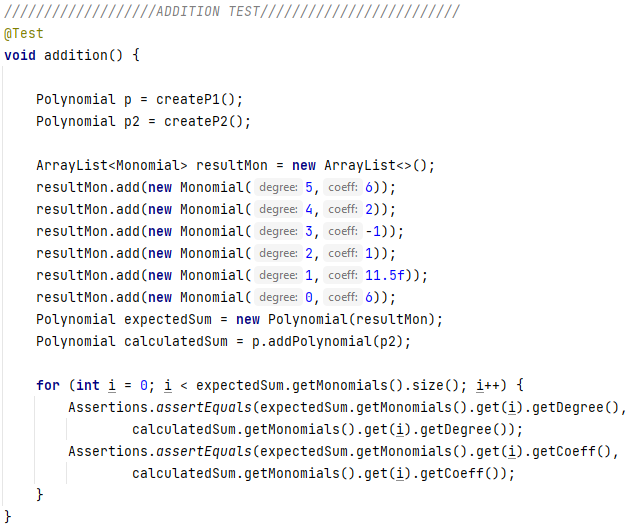
The View class is the one that generates the user interface of the application. It is implemented using Java Swing. Its fields are components that make up the user-friendly graphical image of the polynomial calculator, being composed of labels, text fields, buttons, and a combo-box for choosing the wanted computation. The view has a constructor which generates the window of the application, as well as methods for displaying those components on the screen.

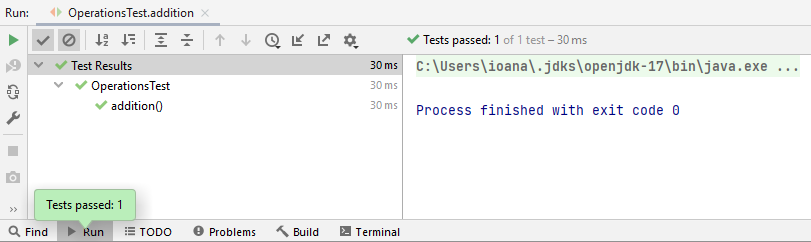
Other methods implemented are getPolynomial1 ( ) and getPolynomial2 ( ) which take data inserted by the user in text fields and return it as a string. Those methods are then called by the controller in order to transform it from strings to objects of class Polynomial.

The methods showResult (String result) and showResult (String quotient, String remainder) are overloading. Depending on the chosen operation that outputs one or two results, the view will display one polynomial or two – quotient and remainder.

# Results

Testing of the program was realized with JUnit framework. For this I designed the OperationsTest class that has several methods, one for each operation to test. The idea for testing a method is to compare the result with an already defined result that is known and correct, using “Assertions.assertEquals ( … )”.





# Conclusions

The purpose of the application was creating and developing a polynomial calculator that is capable of performing computations on polynomials and interact with the user using the graphical interface. Developing this application required skills regarding object-oriented programming, as well as mathematics. Knowledge of classes, implementation and method writing are crucial for creating this project.

# Bibliography

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

<https://towardsdatascience.com/everything-you-need-to-know-about-mvc-architecture-3c827930b4c1>

<https://www.vogella.com/tutorials/JUnit/article.html>