DOCUMENTATION

POLYNOMIAL CALCULATOR

NAME: Mihai Anca

CONTENTS

1. Assignment Objective

Main objective: design and implement a polynomial calculator with a dedicated graphical user interface through which a person can enter polynomials, select operations to apply on them (e.g.: addition, subtraction, multiplication, division, derivative, integral) and view the results of their actions on the screen.

Sub-objectives:

1. understand the problem and its domain

2. analyze the problem and identify the requirements

3. find solutions to individual problems, from which the overall solution is synthesized, sub-objective which can be divided into another two:

3.1 design the polynomial calculator

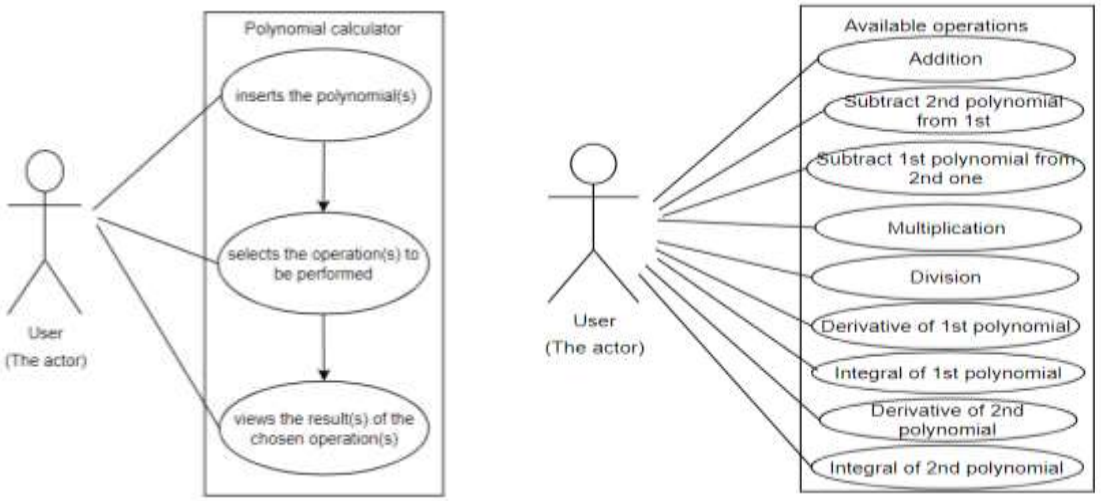
3.2 implement the polynomial calculator

4. test the polynomial calculator

2. Problem Analysis, Modeling, Scenarios, Use Cases

.

Use case diagrams:



Functional requirements:

1. The calculator should allow users to insert polynomials

2. The calculator should allow users to reset the fields of the input polynomials (and of the result one)

3. The calculator should allow users to select the mathematical operation

4. The calculator should add two polynomials

5. Regarding the subtraction, the calculator should allow users to select which polynomial is subtracted from the other one

6. The calculator should multiply two polynomials

7. The calculator should divide two polynomials

8. The calculator should be able to find out the derivative of the chosen polynomial

9. The calculator should be able to find out the integral of the chosen polynomial

Non-functional requirements*:*

1. The calculator should be intuitive and easy to use by the user

2. The calculator should perform the operations fast

3. The calculator should be reliable and consistent (should be able to do operations correctly for ANY polynomial introduced)

Note: A polynomial is considered to be correctly introduced if its composed of monomials, where a monomial has the following structure: [+ for positive numbers (excluding the first monomial)][integer][opt: X[opt: ^natural number]].
Any other structure, any other characters or empty fields for the considered polynomials will generate a pop-up saying that the input has a bad format.

Use cases:

1. *add polynomials*

Primary Actor: user

Main Success Scenario:

I. The user inserts the 2 polynomials in the graphical user interface.

II. The user selects the "Addition" operation.

III. The polynomial calculator performs the addition of the two polynomials and displays the result.

Alternative Sequence:

Incorrect polynomials - The user inserts incorrect polynomials (e.g. with 2 or more variables)- The scenario returns to step **I**

2. *subtract first polynomial from second one or vice versa*

Primary Actor: user

Main Success Scenario:

I. The user inserts the 2 polynomials in the graphical user interface.

II. The user selects the "Subtract P2 from P1" (or "Subtract P1 from P2") operation.

III. The polynomial calculator performs the subtraction of the two polynomials and displays the result.

Alternative Sequence:

Incorrect polynomials - The user inserts incorrect polynomials (e.g. with 2 or more variables)- The scenario returns to step **I**

3. *multiply polynomials*
Primary Actor: user
Main Success Scenario:
I. The user inserts the 2 polynomials in the graphical user interface.
II. The user selects the "Multiplication" operation.
III. The polynomial calculator performs the multiplication of the two polynomials and displays the result.
Alternative Sequence:
Incorrect polynomials - The user inserts incorrect polynomials (e.g. with 2 or more variables)- The scenario returns to step **I**

4. *divide polynomials*
Primary Actor: user
Main Success Scenario:
I. The user inserts the 2 polynomials in the graphical user interface.
II. The user selects the "Division" operation.
III. The polynomial calculator performs the division of the polynomial with higher degree to the one with the lower degree and displays the result.
Alternative Sequence:
Incorrect polynomials - The user inserts incorrect polynomials (e.g. with 2 or more variables)- The scenario returns to step **I**
Divide by 0 – At least one of the inserted polynomials is equal to 0 – The scenario returns to step **I**

5. *differentiate polynomial*
Primary Actor: user
Main Success Scenario:
I. The user inserts the considered polynomial in the graphical user interface.
II. The user selects the "Derivate of P1" (or "Derivate of P2") operation.
III. The polynomial calculator performs the differentiation of the considered polynomial and displays the result.
Alternative Sequence:
Incorrect polynomials - The user inserts incorrect polynomials (e.g. with 2 or more variables)- The scenario returns to step **I**

6. *integrate polynomial*
Primary Actor: user
Main Success Scenario:
I. The user inserts the considered polynomial in the graphical user interface.
II. The user selects the "Integral of P1" (or "Integral of P2") operation.

III. The polynomial calculator performs the integration of the considered polynomial and displays the result.

Alternative Sequence:

Incorrect polynomials - The user inserts incorrect polynomials (e.g. with 2 or more variables)- The scenario returns to step **I**

7. *reset fields*

Primary Actor: user

Scenario:

I. The user selects "Reset fields".

II. The fields of the input polynomials and the field of the result are cleared.

3. Design

OOP design of the application:

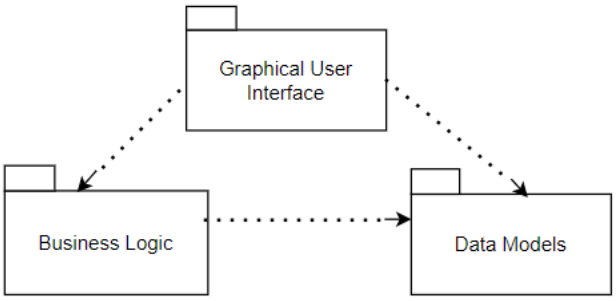
The project is based on a layered architecture, which has the following advantages:

1. **Code Maintenance is easy**: we can easily determine any kind of change in the code

2. **Security:** the data-providing package isn't affected by the other packages

3. **Ease of development:** building time taken by the application will be small as all the layers can work together at the same time

UML package diagram:



The project has 3 packages + the Main class (its purpose is to run the application):

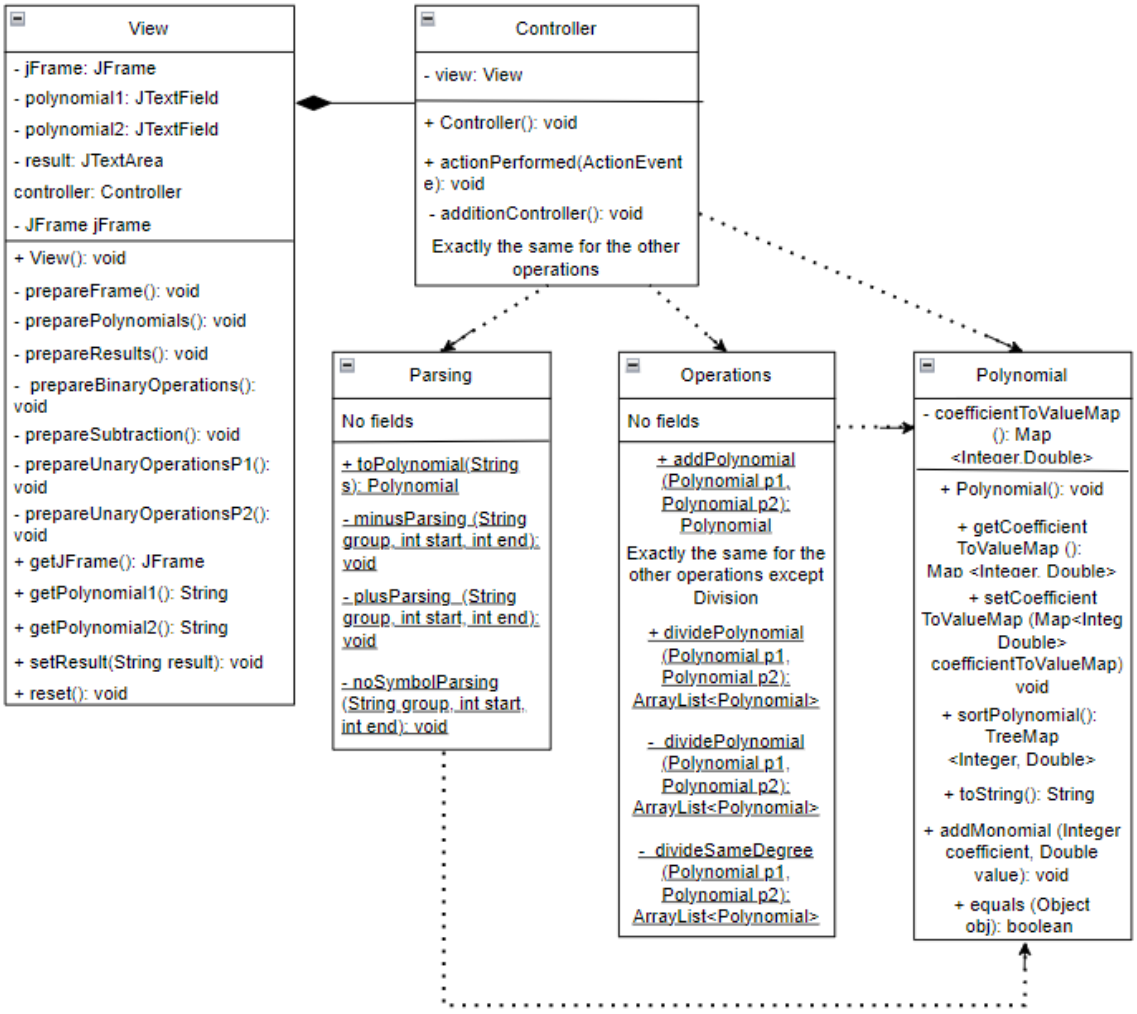
1. *Data Models* – contains the classes modeling the application data

- in this app, it contains only a class: the Polynomial class

2. *Graphical User Interface* – contains the classes implementing the graphical user interface

3. *Business Logic* – contains the class implementing the mathematical functionality (*Operations* class) and also the one transforming the inputs of the graphical user interface into the required data (*Parsing* class)

Class Diagram:



Used data structures:

- *primitive data types* (ex: int)
- their *Object- equivalent classes* (ex: Integer, Double)
- *Maps* (TreeMap, HashMap): I used *Java Collections Framework* because it reduces programming effort and increases program speed, quality respectively; I used the *Map* interface in order to connect degree (key) -> number (value) of each monomial of the

polynomial; in addition, I used *HashMap* to handle collisions and *TreeMap* to sort the polynomial based on the degree of the each monomial

- *newly created objects* (ex: Polynomial)

Used algorithms – *Polynomial arithmetic*:

1. *Multiplication of two polynomials*:

To multiply two polynomials, multiply each monomial in one polynomial by each monomial in the other polynomial, add the results and simplify if necessary.

2. *Division of two polynomials*:

To divide two polynomials P and Q, the following steps should be performed:

- **Step 1** - Order the monomials of the two polynomials P and Q in descending order according to their degree.
- **Step 2** - Divide the polynomial with the highest degree to the other polynomial having a lower degree (let's consider that P has the highest degree)
- **Step 3** – Divide the first monomial of P to the first monomial of Q and obtain the first term of the quotient
- **Step 4** – Multiply the quotient with Q and subtract the result of the multiplication from P obtaining the remainder of the division
- **Step 5** – Repeat the procedure from step 2 considering the remainder as the new dividend of the division, until the degree of the remainder is lower than Q.

4. Implementation

Implementation of classes:

I. data_models Package**:**

Polynomial Class

- fields:
  - *private Map<Integer, Double> coefficientToValueMap (coefficient* represents the degree and *value* the number of each monomial)
- important methods*:*
  - *public Polynomial()* : its constructor
  - *public Map<Integer, Double> getCoefficientToValueMap()* : getter for the monomials of the polynomial
  - *public void setCoefficientToValueMap(Map<Integer, Double> coefficientToValueMap)* : setter for the monomials of the polynomial
  - *public TreeMap<Integer, Double> sortPolynomial()* : sort the monomials of the polynomials in desceding order with respect to their degree
  - *public void addMonomial (Integer coefficient, Double value)* : method for adding new monomials to the polynomials (it doesn't add monomials with value equal to 0

and it takes into consideration the fact that a monomial with same degree can already exist in the polynomial's structure)

- *public String toString()*: shows the polynomial's monomials as a string
- *public boolean equals (Object obj):* override of the Objects.equals() method, used in the Junit tests

II. business_logic Package:

Operations Class

-important methods (all return polynomials):

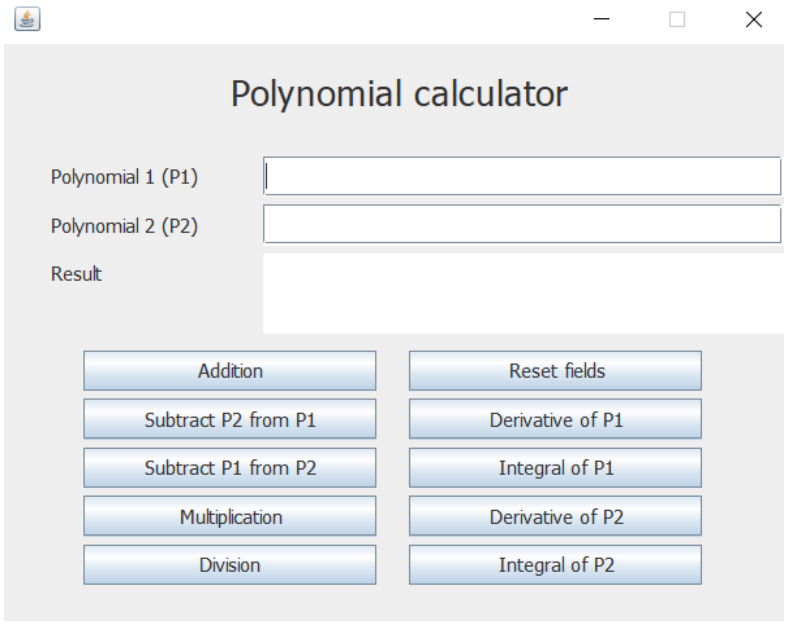
- *public static Polynomial addPolynomial (Polynomial p1, Polynomial p2)*
- *public static Polynomial subtractPolynomial (Polynomial p1, Polynomial p2)*
- *public static Polynomial multiplyPolynomial (Polynomial p1, Polynomial p2)*
- *public static ArrayList<Polynomial> divisionPolynomial (Polynomial p1, Polynomial p2)* – this method returns an *ArrayList* because contains both the *quotient* (element 0) and the *remainder* (element 1) of the division; it establishes the polynomial with the higher degree and the division itself is done with the additional methods: *private static ArrayList<Polynomial> dividePolynomial (Polynomial p1, Polynomial p2)* and *private static ArrayList<Polynomial> divideSameDegree (Polynomial p1, Polynomial p2)*
- *public static Polynomial derivativePolynomial (Polynomial p1)*
- *public static Polynomial integralPolynomial (Polynomial p1)*

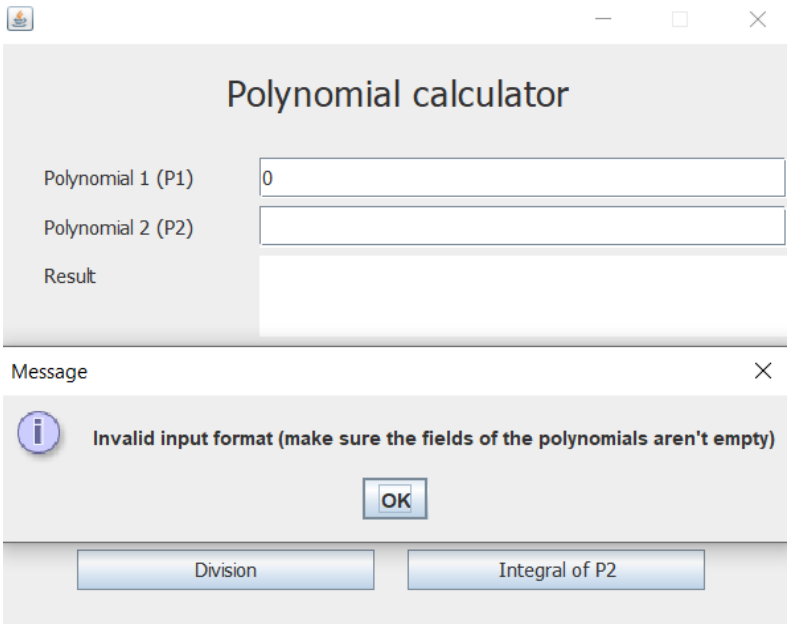
Parsing Class

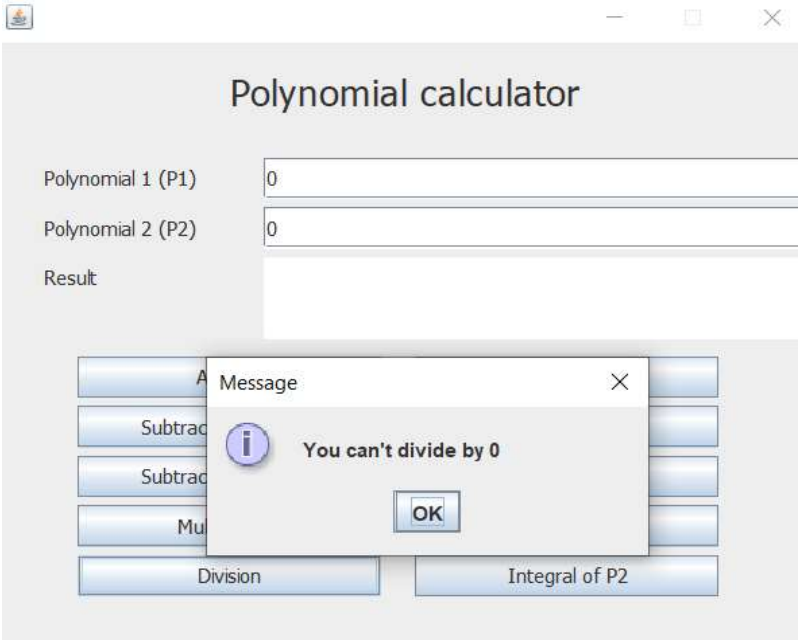
-important methods:

- *public static Polynomial toPolynomial (String s)*: tests if a string respects the structure of a polynomials (a set of monomials connected through "+" or "-"); it's "supported" by 3 additional methods for 3 possible cases: a monomial is introduced by "-", "+" or neither*: private static Polynomial minusParsing (String group, int start, int end), private static Polynomial plusParsing (String group, int start, int end), private static Polynomial noSymbolParsing (String group, int start, int end)* respectively

III. gui Package (**Implementation of the Graphical User Interface**)







View class – this class presents the graphic aspect of the calculator

- has as a <u>field</u> JFrame because, in this way, I can freely choose its size, visibility and layout
- I chose the AbsoluteLayout because I can put my elements (labels, TextFields, buttons) however I want, by specifying the bounds of each element
- <u>another fields:</u>
  - *private JTextFields polynomial1* and *polynomial2*
  - *private JTextArea result*
  - *Controller controller*
- <u>important methods:</u>
  - *public View()*: its constructor, which uses the following methods:

1. *private void prepareFrame ();*

2. *private void preparePolynomials ();*

3. *private void prepareResult ();*

4. *private void prepareBinaryOperations ();*

5. *private void prepareSubtraction ();*

6. *private void prepareUnaryOperationsP1 ();*

7. *private void prepareUnaryOperationsP2 ();*

  - getters: *public JFrame getJFrame (), public String getPolynomial1 (), public String getPolynomial2 ()*
  - *public void setResult (String result)* : for displaying the result in its dedicated area
  - *public void reset()* : used when resetting the fields of the calculator

Controller class

– this class controls both the data flow and the view of the app (whenever data changes; that's why it's only <u>field</u> is *private View view*)

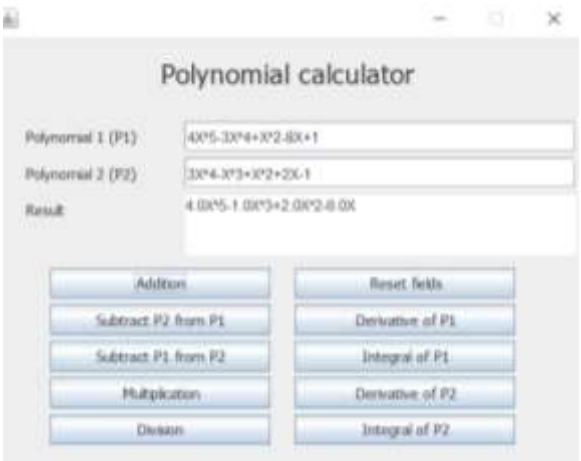
- <u>important methods</u> (all of them control buttons of the view, except its constructor: *public Controller (View view)*):

  - *public void actionPerformed (ActionEvent e)* aided by:
    1. *private void additionController ()*
    2. *private void divisionController ()*
    3. *private void multiplicationController ()*
    4. *private void subtract1Controller ()*
    5. *private void subtract2Controller ()*
    6. *private void derivative1Controller ()*
    7. *private void derivative2Controller ()*
    8. *private void integral1Controller ()*
    9. *private void integral2Controller ()*

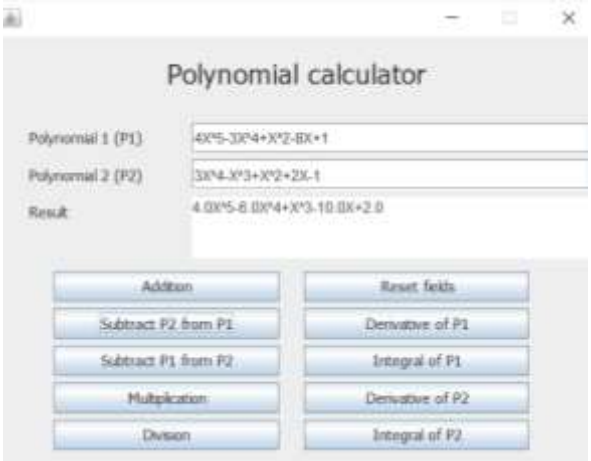
5. Results

Displayed by the Graphical User Interface

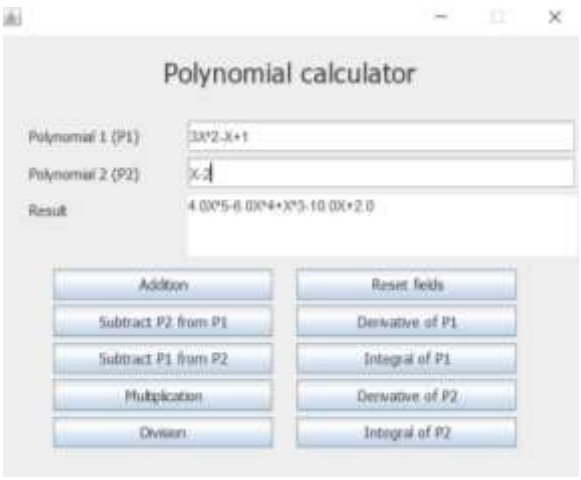
1. *Addition*



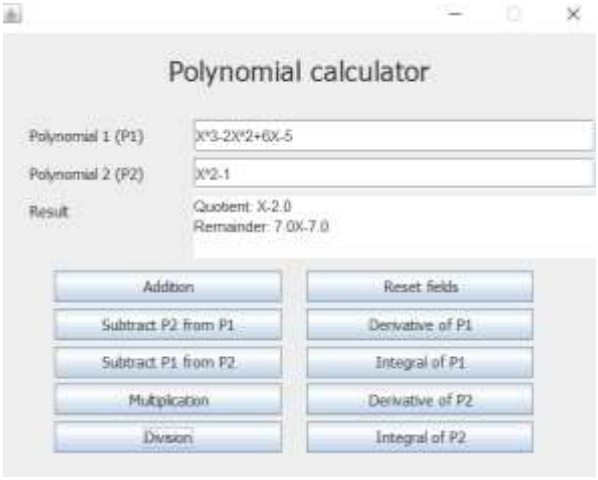
2. *Subtraction (<u>Subtract P2 from P1</u>)*



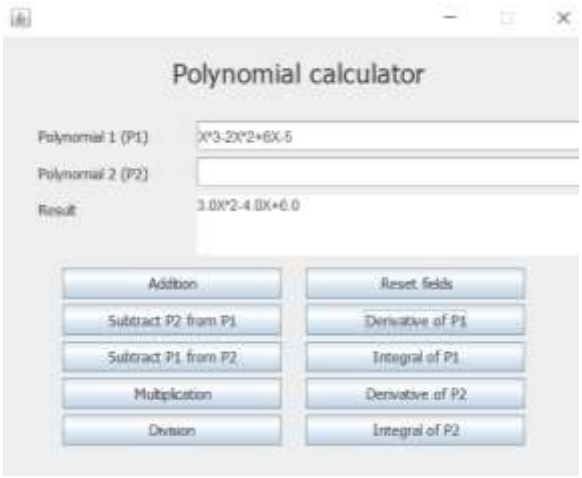
3. *Multiplication*



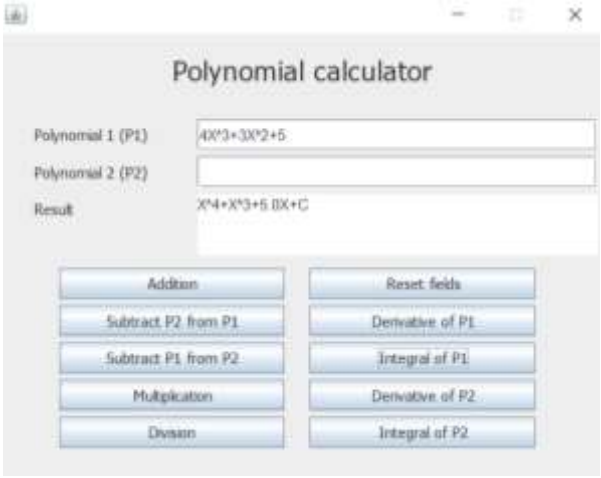
4. *Division*



5. *Derivative (Derivative of P1)*



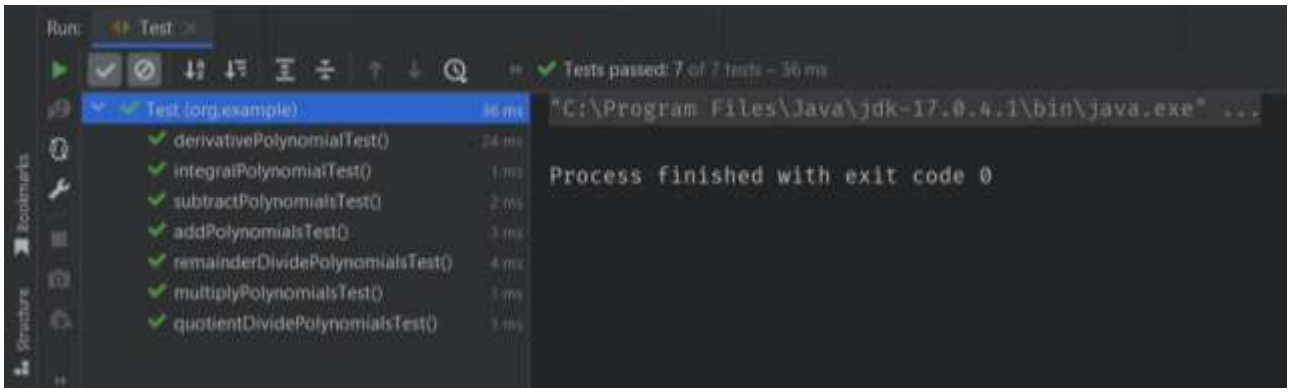
6. *Integral (Integral of P1)*



Junit Tests

What's tested	Input	Expected result	Actual result	Pass/Fail
Addition	P1: 4X^5 -3X^4 + X^2 -8X + 1 P2: 3X^4 -X^3 + X^2 + 2X - 1	4X^5 -X^3 + 2X^2 -6X	4X^5 -X^3 + 2X^2 -6X	Pass

Subtraction	P1: 4X^5 -3X^4 + X^2 -8X + 1 P2: 3X^4 -X^3 + X^2 + 2X - 1	4X^5 -6X^4 + X^3 - 10X + 2	4X^5 -6X^4 + X^3 - 10X + 2	Pass
Multiplication	P1: 3X^2 – X + 1 P2: X - 2	3X^3 -7X^2 + 3X - 2	3X^3 -7X^2 + 3X - 2	Pass
Quotient of division	P1: X^3 -2X^2 + 6X - 5 P2: X^2 - 1	X - 2	X - 2	Pass
Remainder of division	P1: X^3 -2X^2 + 6X - 5 P2: X^2 - 1	7X - 7	7X - 7	Pass
Derivate	X^3 - 2X^2 + 6X - 5	3X^2 - 4X + 6	3X^2 - 4X + 6	Pass
Integral	4X^3 + 3X^2 + 5	X^4 + X^3 + 5X	X^4 + X^3 + 5X	Pass



6. Conclusions

Conclusions

Firstly, with the help of this assignment, I have remembered the OOP notions learned last semester and also improved them. In addition, I have "relearned" the basics of polynomial arithmetic.

Secondly, I had to manage my time properly in order to be able to complete until deadline this assignment which represents quite a complex project.

Lastly, I faced and learned from multiple scenarios when the code I wrote wasn't doing exactly what I was expecting it to do and, therefore, I had to debug it, tool which can be seen as crucial when programming.

What I have learned from this assignment

- testing with JUnit and, because of that, working with dependencies, too
- working and checking that certain patterns are respected in a String Object with Regex and Matcher
- more about the usage of static methods
- more about the usage of Map Interface with the help of *TreeMap* and *HashMap* Classes

Future developments

- finding the roots of a polynomial (polynomial has the degree 1,2,3 or 4)
- computing the value of a polynomial when X = a certain number
- operations on polynomials with multiple variables
- operations on polynomials with coefficients being floating point numbers

7. Bibliography

1. Lectures of Fundamental Programming Techniques - S.L. dr. ing. Cristina POP
2. https://www.geeksforgeeks.org/
3. https://stackoverflow.com/questions
4. https://www.javatpoint.com/java-regex