POLYNOMIAL CALCULATOR

DOCUMENTATION - ASSIGNMENT 1

Programming Techniques

STUDENT: DOMȘA-STÎNĂ ANDA
GROUP: 30423

TABLE OF CONTENTS

1. Objectives ..... **Error! Bookmark not defined.**
2. Analizing the requirements 4
3. Design 7
4. Implementation 11
5. Results 14
6. Conclusions 14
7. Bibliography 15

1. Objectives

A. Main objective

Project specification:

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 main objective of this homework was to develop a Java application capable of implementing and performing operations on polynomials having integer coefficients. Therefore, the user should be able to input the polynomials and to choose the operation that should be performed on them: addition, subtraction, multiplication or division. Some additional operations such as derivation and integration of a polynomial should also be presented. The program should have a user friendly interface implemented using Java Swing or JavaFX.

B. Sub-objectives:

Analyze the problem and identify requirements

A polynomial calculator may be useful for a wide range of users, from computer science experts who need to compute complex polynomial operations, to students who just started learning about polynomials and need an application to help them do basic polynomial processing. An intuitive GUI will result in better productivity in working with the program in comparison to a console-based application, which will probably be difficult to use by non-expert users.

The first step in implementing the program is to understand the problem, to identify the tasks it should perform and to model its functionalities using an object-oriented approach. This will be forward detailed in chapter 2.

Design the polynomial calculator

The next step in solving this project's requirements is to design the polynomial calculator by dividing it into smaller components using structural diagrams. This will be further discussed in chapter 3.

Implement the polynomial calculator

Now that the requirements are understood and modeled, the next step is to implement the actual java program by writing code for the needed classes in Intellij. This step will be covered in chapter 4.

Test the polynomial calculator

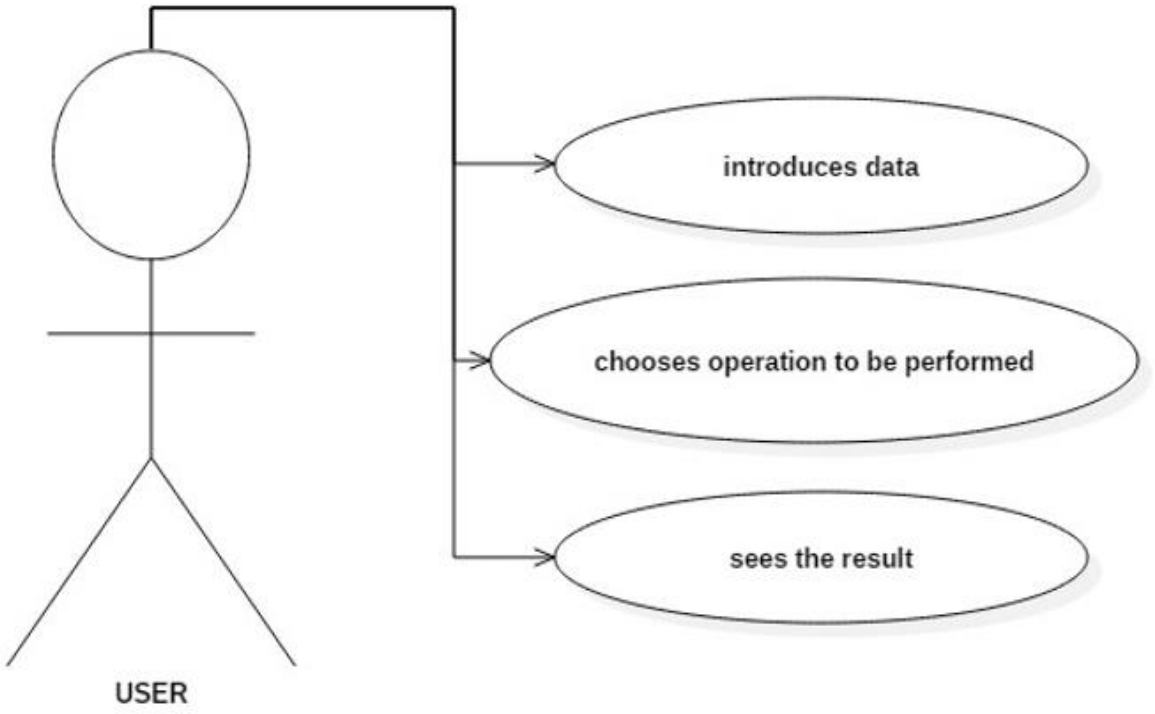
The last step of the project development is to test the calculator's operations using Junit, a process which I have described in chapter 5 of the documentation.

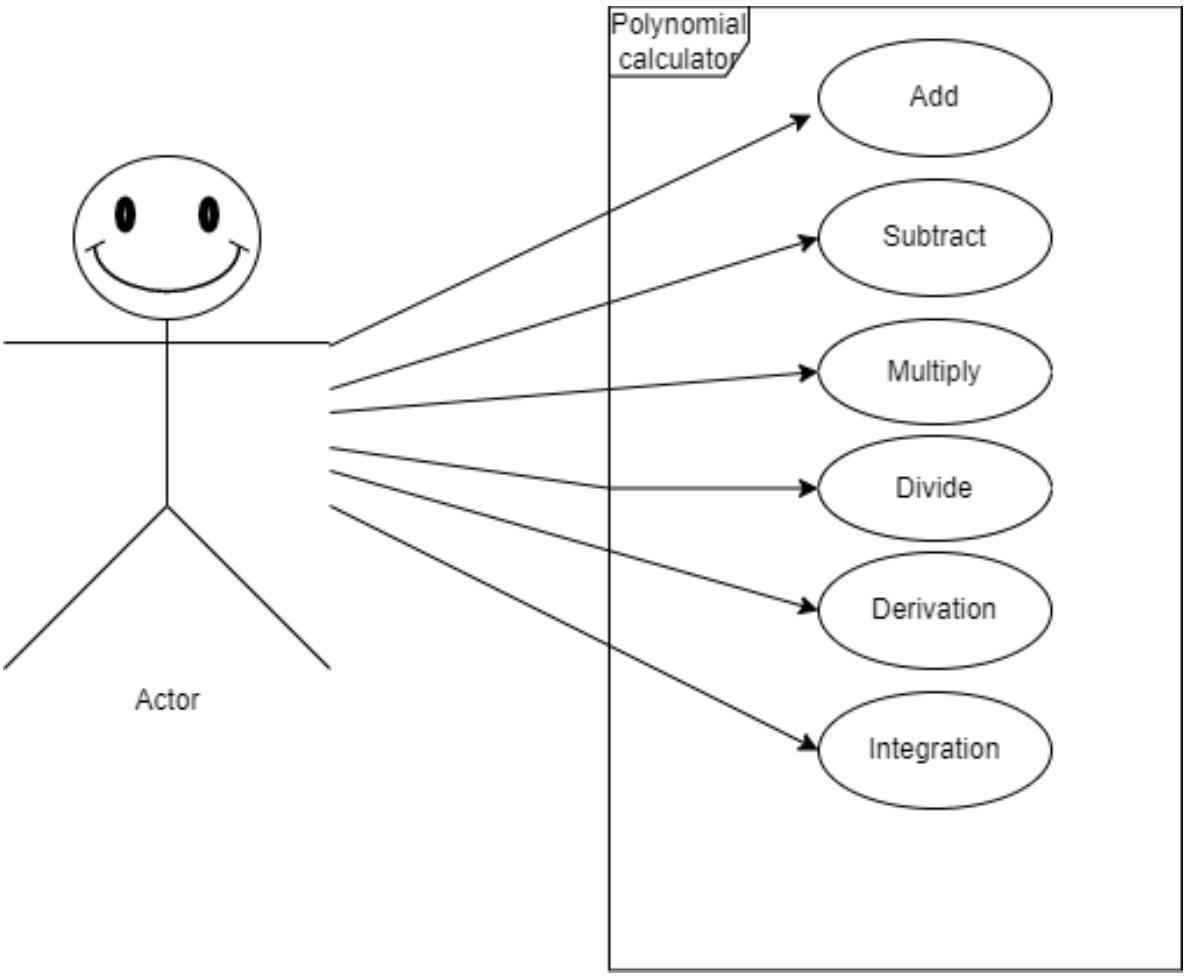
2. Analyzing the requirements

The first step in implementing the program is to understand the problem, to identify the tasks it should perform and to model its functionalities using an object-oriented approach.

So basically, the polynomial calculator should take two polynomials from the user and implement correctly the most common operations for polynomials. The program will be able to compute the following operations: addition, subtraction, multiplication, derivation, division and integration.

Use case diagrams:





The use case diagram presents the actor, which in our case is the user that interacts with the application. He can choose to perform several actions on the two inserted polynomials, such as addition, subtraction, multiplication, division, derivation and integration.

Requirements:

1. Functional requirements of the project:
    - The polynomial calculator should allow the user to input to polynomials in a string format
    - The polynomial calculator should allow users to select the mathematical operation
    - The polynomial calculator should be able to add two polynomials
    - The polynomial calculator should be able to subtract two polynomials
    - The polynomial calculator should be able to multiply two polynomials
    - The polynomial calculator should be able to divide two polynomials
    - The polynomial calculator should be able to derivate a polynomial
    - The polynomial calculator should be able integrate a polynomial
    - The polynomial calculator should be able to display the result on the graphical user interface in a string format

2. Non-functional requirements:
    - The polynomial should be intuitive and easy to use by the user

3. UI Requirements:
   The I/O mechanism for the Polynomial Processing Application must contain:
    - One Java compatible device which can compile and run Java standalone applications
    - A keyboard from which the user can input a polynomial.
    - A screen onto which he can see the input / output for the application.

Modelling the problem:

The most important object that had to be modeled was of course the Polynomial. I chose to model it as an array of monomials, so I will need two classes : “Polynomial” and “Monomial” for the implementation.

Possible scenarios:

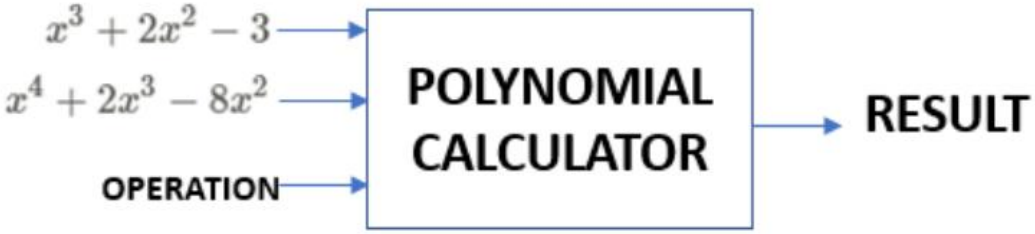
Main success scenario: (Use case : operation, actor: user)

- Step 1. The user inserts the two polynomials in the graphical user interface.
- Step 2. The user selects the operation by pressing the button corresponding to it (“Addition”, “Subtraction”, “Multiplication”, “Division”, “Derivation”, “Integration”).
- Step 3. The polynomial calculator performs the operation and displays the result.

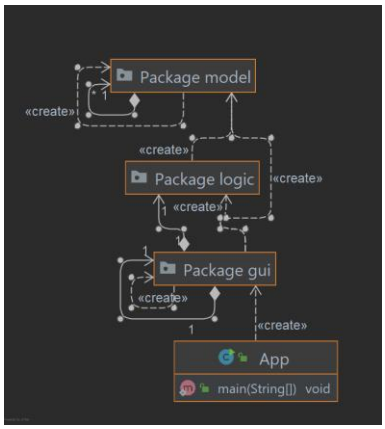
Alternative scenario:

- The user inserts incorrect polynomials
- The program displays a warning pop-up
- The program returns to step 1

3. Design



The next step in solving this project's requirements is to design the polynomial calculator by dividing it into smaller components using structural diagrams.



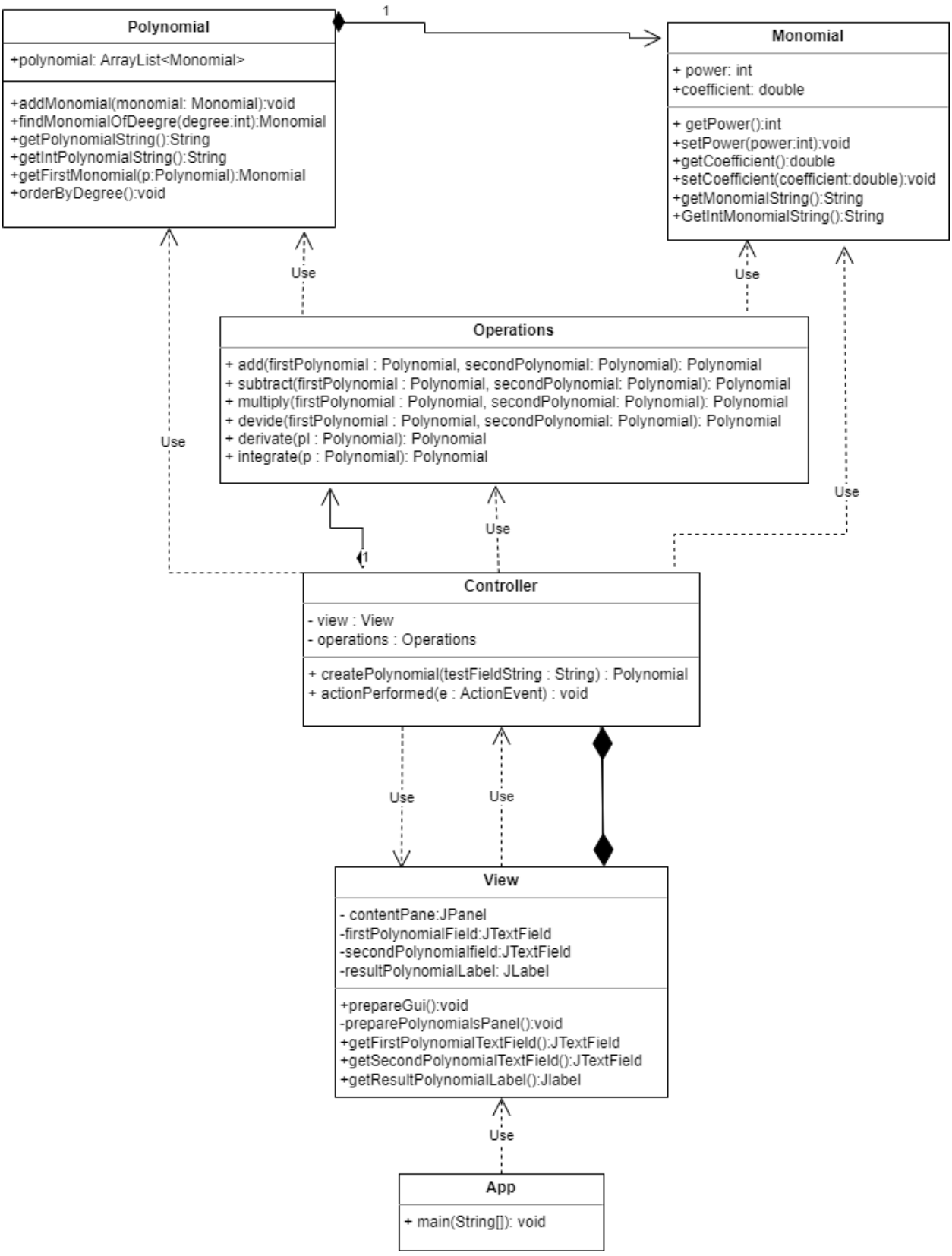
For the design of the project I used the Model View Controller architectural pattern, using the "model" package for the implementation of the classes "Polynomial" and "Monomial" and the "gui" package for classes "Controller" and "View". I also included the package "logic" which contains the "Operations" class and the separate "App" class for running the application.

The polynomials are constructed from terms named monomials, so I used for polynomial an ArrayList of monomials. The "Monomial" class and the "Polynomial" class have an aggregation relationship.

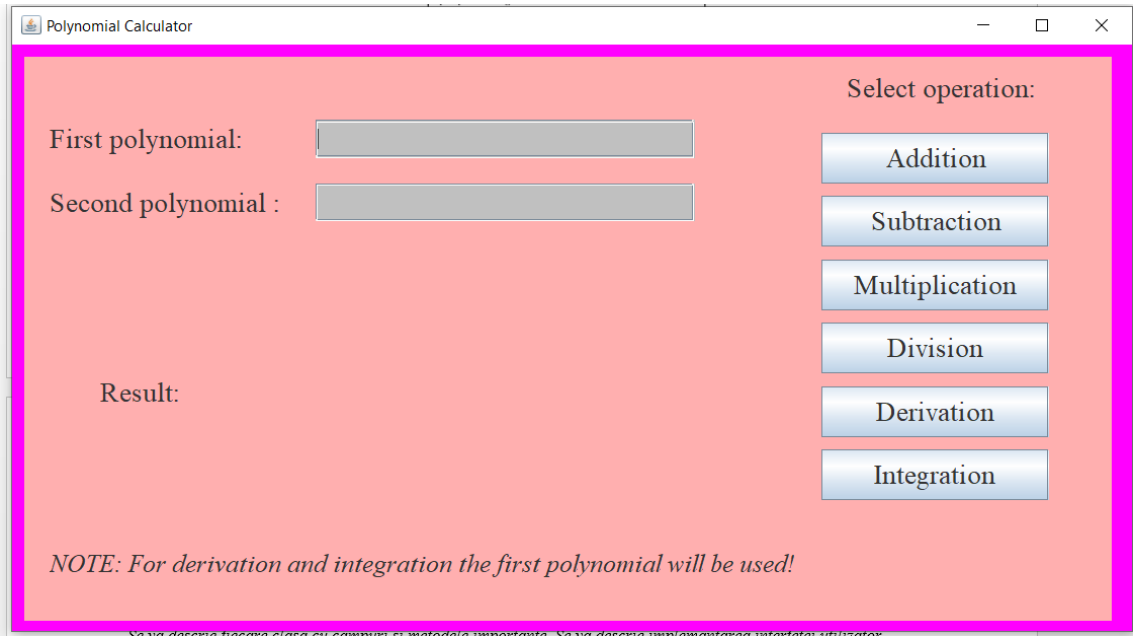
Data Structures

The data structures that I have been working with in this problem are primarily primitive data types (especially integers and doubles) and a more complex one, such as ArrayList type object or new created object such as Monomial and Polynomial.

So I have decided to use ArrayList instead of the classic arrays because I think that they are more efficient from the point of view of memory management, performance and provide a faster access to their content. Also, the size of an ArrayList is not fixed and adding new elements to the list is very easy because we do not have to worry about exceeding the length of the list (array).

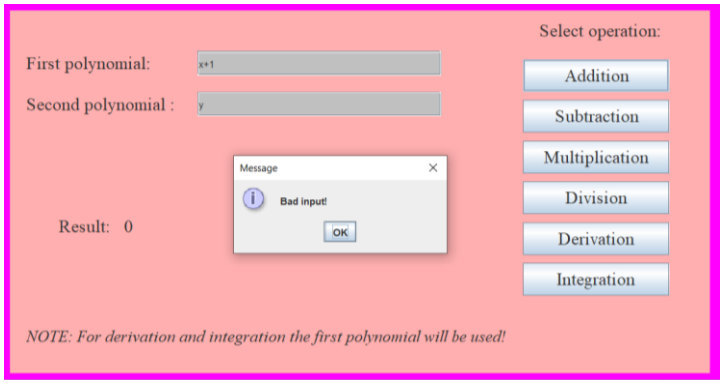

Polynomial
+polynomial: ArrayList<Monomial>
+addMonomial(monomial: Monomial):void
+findMonomialOfDeegre(degree:int):Monomial
+getPolynomialString():String
+getIntPolynomialString():String
+getFirstMonomial(p:Polynomial):Monomial
+orderByDegree():void
1
Monomial
+ power: int
+coefficient: double
+ getPower():int
+setPower(power:int):void
+getCoefficient():double
+setCoefficient(coefficient:double):void
+getMonomialString():String
+GetIntMonomialString():String
Use
Use
Use
Operations
+ add(firstPolynomial : Polynomial, secondPolynomial: Polynomial): Polynomial
+ subtract(firstPolynomial : Polynomial, secondPolynomial: Polynomial): Polynomial
+ multiply(firstPolynomial : Polynomial, secondPolynomial: Polynomial): Polynomial
+ devide(firstPolynomial : Polynomial, secondPolynomial: Polynomial): Polynomial
+ derivate(pl : Polynomial): Polynomial
+ integrate(p : Polynomial): Polynomial
Use
Use
Use
1
Controller
- view : View
- operations : Operations
+ createPolynomial(testFieldString : String) : Polynomial
+ actionPerformed(e : ActionEvent) : void
Use
Use
View
- contentPane:JPanel
-firstPolynomialField:JTextField
-secondPolynomialfield:JTextField
-resultPolynomialLabel: JLabel
+prepareGui():void
-preparePolynomialsPanel():void
+getFirstPolynomialTextField():JTextField
+getSecondPolynomialTextField():JTextField
+getResultPolynomialLabel():Jlabel
Use
App
+ main(String[]): void

GUI – Graphic User Interface



The graphical user interface is implemented using Java Swing. It can be used to compute the addition, subtraction, multiplication, division, derivation and integration of polynomials with integer coefficients. The user must input the polynomials so the first polynomial goes into the "First polynomial" text field and the second one goes into the "Second polynomial" text field. The polynomials can be introduced either in this format: "4x^5+3x^6+x+4" or in this format : "4*x^4+3*x^6+1*x^1+4*x^0".

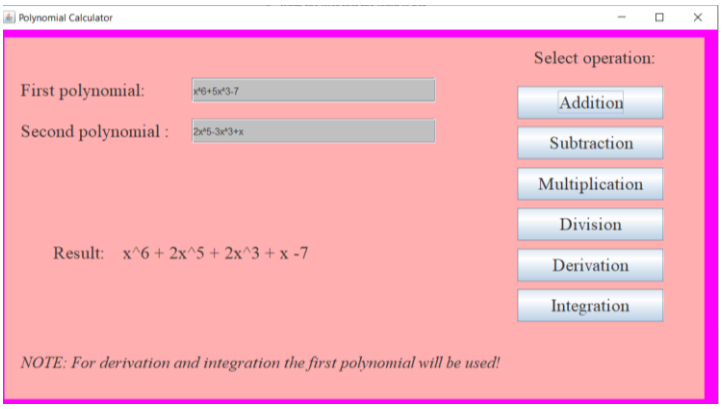
After inserting the polynomials the user must choose an operation to be performed by pressing the button corresponding to the desired operation.



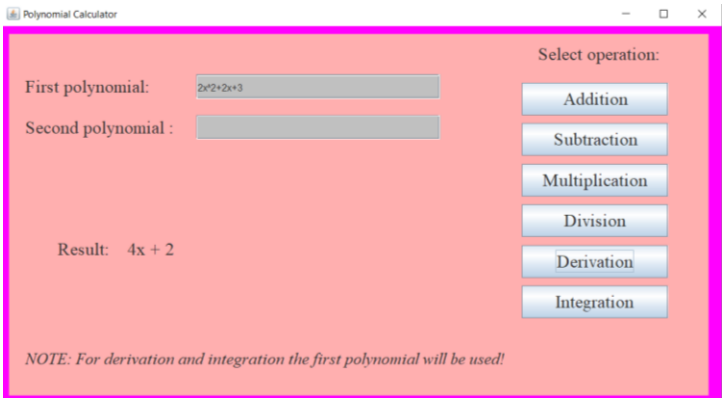
If the user does not provide a valid polynomial, the program will open a pop-up window which will warn the user that the input is not correct. For example, the input will be incorrect is there is more than one variable or if the symbol '^' is not followed by any number. The user must then provide a valid input for the calculator to work.

If the input is correctly inserted, the result will be displayed near the “Result” label.

For example, if the user inserts the following polynomials: First polynomial: “x^6+5x^3-7” and Second polynomial: “ 2x^5 -3x^3 + x ” and they choose the operation “Addition” , the result will show the following String: “ x^6 + 2x^5 + 2x^3 + x -7 ”.



The same goes for the subtraction, multiplication and division.



For derivation and integration, only the first polynomial is taken into account so only that one must be provided.
For example, if the user inserts the polynomial: “ 2x^2 + 2x + 3 ” and presses the “Derivation” button, the result will be “ 4x + 2”.

The running of the application will stop if the user presses the “x” (close) button.

The GUI also contains a note field with an instruction for the user, so the user know which polynomial is used for the operations of derivation and integration.

4. Implementation

I started the implementation by working on the "Polynomial" and "Monomial" classes, both of them being found in the "model" package.

Polynomial:

The "Polynomial" class has a single variable, an ArrayList of Monomials.

Methods:

- The methods implemented in this class will help with the operations methods.
- *public Monomial findMonomialOfDegree(int degree)* is a method that receives as parameter an integer degree and returns the monomial (from a polynomial) that has the power equal to that degree. It is used in computing the addition, subtraction and multiplication.
- *public String getPolynomialString( ) :* returns the String form of a polynomial with double coefficients
- *public String getIntPolynomialString( ):* returns the String form of a polynomial with integer coefficients
- *public Monomial getFirstMonomial(Polynomial p):* returns the first monomial (the one with the greatest power) from a polynomial
- *public void orderByDegree :* orders the monomials in the "natural" order, from highest to lowest

Monomial:

A polynomial is composed by one or more terms, which are called monomials.

This class has two instance variables, an integer power and a double coefficient.

Methods:

- *public String getMonomialString():* returns the String form of a monomial withdouble coefficient; used in *getPolynomialString( )*

- *public String getIntMonomialString():*returns the String form of a monomial with integer coefficient; used in *getIntPolynomialString( )*

Next, I implemented the "Operations" class from the "logic" package and I tested the operations using JUnit to verify them before working with a GUI.

Operations:

This class groups together all the operation methods .

Methods:

- *public Polynomial add(Polynomial firstPolynomial, Polynomial secondPolynomial):* returns the sum of two polynomials by searching for monomials with the same power and adding their coefficients
- *public Polynomial subtract(Polynomial firstPolynomial, Polynomial secondPolynomial):* returns the difference of the two polynomials by searching for monomials with the same power and subtracting their coefficients
- *public Polynomial multiply(Polynomial firstPolynomial, Polynomial secondPolynomial):* returns the product of the polynomials
- *public Polynomial divide(Polynomial firstPolynomial, Polynomial secondPolynomial):* returns the division of the polynomials
- *public Polynomial derivate(Polynomial p):* returns the derivated polynomial
- *public Polynomial integrate(Polynomial p):* returns the integrated polynomial

Implementation of the GUI:

The package "gui" contains the classes "View" and "Controller" which are responsible for creating the User Interface and providing the user with the ability to interact with the application.

I implemented the graphical user interface using Java Swing.

View:

Generates the Interface, the panels, buttons and labels.

I designed the panels, labels and buttons using the ".setBounds()" function. I also set a new font and new colors for the interface's components.

```
private JPanel contentPane;
private JTextField firstPolynomialTextField;
private JTextField secondPolynomialTextField;
private JLabel resultPolynomialLabel;

Controller controller = new Controller(this);

Font myFont = new Font("Times New Roman", Font.PLAIN, 22);
```

Methods:

- *public void prepareGui()*
- *private void preparePolynomialsPanel() :* generates almost all the elements of the interface

Example of adding a button:

```
//addition button
JButton additionButton = new JButton("Addition");
additionButton.setFont(myFont);
additionButton.setBounds(630,60 , 180, 40);
additionButton.setActionCommand("Addition");
additionButton.addActionListener(this.controller); //if button pushed
polynomialPanel.add(additionButton);
```

Controller:

It is the class that contains the most important methods for the program. It makes the link between the view and the data because it acts on both model and view. It controls the data flow into model object and updates the view whenever data changes.

Methods:

- *public Polynomial createPolynomial(String textFieldString):* it takes the string form of the input polynomial and converts it into a list of monomials using regular expressions like “replaceAll” or ”split”
- *public void actionPerformed(ActionEvent e):* it manages the actions that take place when the user gives a command

App:

It is the class that runs the start() method from the MainController in order to “turn on” the application.

5. Results

JUnit:

For testing the correctness of the polynomial operations I implemented I used JUnit testing. After adding the needed dependencies, I created the class "OperationsTest" in the test package and Iimplemented some methods for testing each operation.

```
public void addTest(){

   Monomial m2 = new Monomial(2,2);
   Polynomial p1 = new Polynomial();
   p1.getPolynomial().add(m2);

   Monomial mm1 = new Monomial(1, 1);
   Monomial mm2 = new Monomial(0, 2);
   Polynomial p2 = new Polynomial();
   p2.getPolynomial().add(mm1);
   p2.getPolynomial().add(mm2);

   Polynomial sum;
   sum=operations.add(p1,p2);

   assertEquals(" 2x^2 + x + 2 ", sum.getIntPolynomialString(), "OK!");
}
```

The test method for the addition is presented above. I used the method *getIntPolynomialString()* to convert the result into a String so that it could be compare to the actual result String.

The tests for the other operations are done in a similar manner, in the end all of them were successful.

6. Conclusion

Even though the start was a little hard because I never imagined I would be able to implement this kind of an application, after diving in it all started to make sense and towards the end working on it actually became kind of fun and relaxing and made me feel quite accomplished.

Besides renewing my knowledge about polynomials, the project helped me learn how to work better with OOP in Java. I also learned how to work with Java Swing to create a GUI and I had fun playing with dimensions and colors. I learned how to test with JUnit and renewed my knowledge about working with git as well.

Possibilities of further improvement:

To improve the polynomial calculator in the future I could implement a method that gets the roots of a polynomial or one that computes the result of the polynomial for a certain value of x.

7. Bibliography:

- https://app.diagrams.net/
- https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf
- PT/Lectures/Lecture_2_UML_Diagrams/Lecture_2_UML_Diagrams.pdf
- https://google.github.io/styleguide/javaguide.html
- https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_domains/packagediagram.html
- www.stackoverflow.com
- www.wikipedia.org
- https://www.draw.io/ (for diagrams)