UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

PROGRAMMING TECHNIQUES
ASSIGNMENT 1

POLYNOMIAL CALCULATOR

STUDENT: GYARMATHY CSABA
UNIVERSITY: TECHNICAL UNIVERSITY OF CLUJ-NAPOCA
FACULTY: COMPUTER SCIENCE
TEACHER: CRISTINA BIANCA POP
TEACHER ASSISTANT: ANDREEA VALERIA VESA
YEAR: 2
ACADEMIC YEAR: 2021/2022
GROUP: 30424

Contents

Assignment Objective

Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation (i.e., addition, subtraction, multiplication, division, derivative, integration) to be performed and view the result. Consider the polynomials to be of one variable and integer coefficients.

Project analysis, scenario, approach and use cases

Analysis

A polynomial is an expression that can be built from constants and symbols called variables by means of addition, multiplication and exponentiation to a non-negative integer power.

A polynomial depending on a single variable x can always be written in the following form:

$$\sum_{i=0}^{n}(a_i x^i) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

where $a_0, a_1, a_2, \ldots, a_n$ are constants and x is the variable. The variable x is indeterminate, meaning that it represents no particular value, although any value may be substituted for it. A single term from a polynomial in the form ax^b is called a monomial. Therefore, we can say that a polynomial composed of one or more monomials.

Alternatively, a polynomial of one variable can be represented as a vector of length *p*, where the first element represents the coefficient of the variable at power *p-1*, the second element represents the coefficient of the variable at power *p-2*, and so on. For example:

[-2, 3, 0, -4, 1], where $p = 5$,

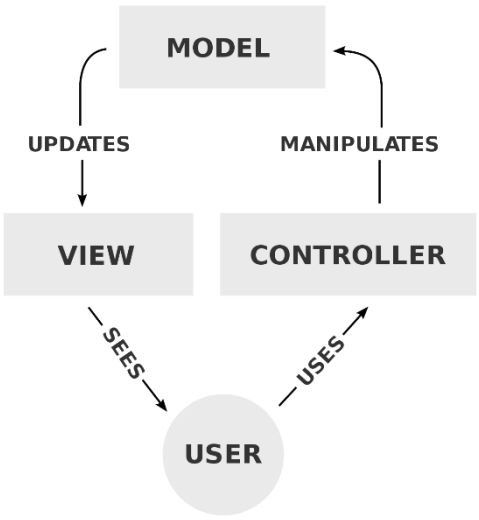
can be written as:

$$-2x^4 + 3x^3 - 4x + 1.$$

Representing a polynomial as a list of monomials can be used to perform the most common operations: addition, subtraction, multiplication, division, differentiation, and integration. In our scenario, this representation will be used to process the polynomials.

Scenario and modelling

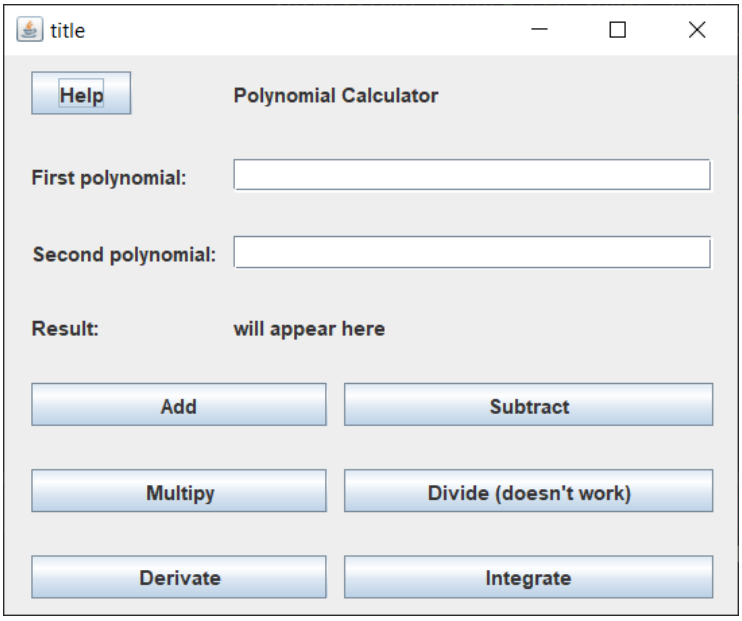
The user is required to enter two polynomials in the user interface he/she wishes to perform the operations on. Once introduced according to the formatting requirements that can be seen by pressing the HELP button, the user can choose which operation to perform by pressing the respective button. The result will appear on the same window, as it would on a normal calculator. Note that for the derivation and integration operation, only the first polynomial will be processed.



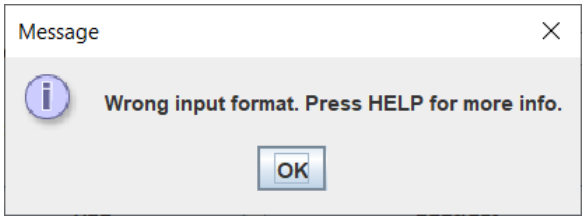
The project is following a classic MVC (model, view, controller) architecture design, which functions as the diagram shows. The user gives a command to the controller, which invokes the model, and as a consequence the model updates the view which the user sees. More technically, the Model has to manage the data and the logic of the application, the View describes the way the Model is displayed in the UI and the Controller takes user input and converts it into a command that is then forwarded to the Model. This creates a circular motion of logic between the packages: the user interfaces with the View, which then passes the call to the Controller, which manipulates the Model, which in turn creates events to pass back to the View. This approach is really efficient for code re-use and parallel development.

Therefore, I created the necessary model, view and controller packages, and outside of these in the main directory I created the Main.java class, which initializes the controller.

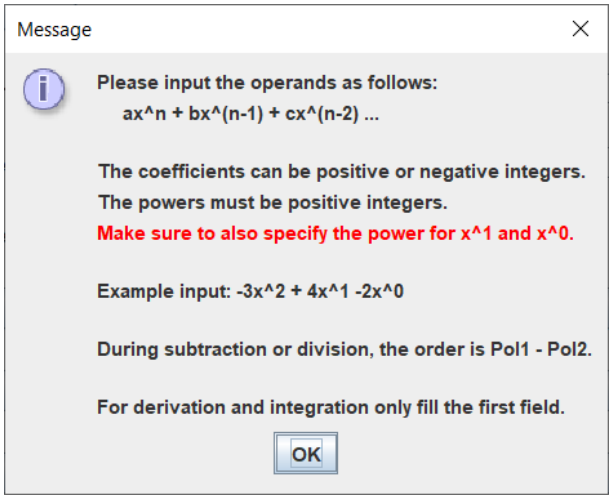
The GUI (Graphical User Interface) is user friendly and easy to navigate. On application start-up, the user is met with the following window:



It really doesn't have much going on for it. As mentioned before, the user inputs two polynomials he/she wishes to perform the operations on, and the result will be displayed on the area where we can see the "will appear here" label. Upon first input, the user may be met with this warning window:



But not to worry, everything is explained in the Help window, which the user can bring up by pressing the HELP button, located in the top left corner of the main window. Here we can see information about the input, formatting of the input, how some operations are performed and even an example input is given.



The result of whichever operation will display the polynomial according to its powers in descending order.

Use case scenarios

A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal. The use cases are strongly connected with the user steps.

As mentioned before, the user must input the polynomials according to the given format and choose which operation to perform. If the warning window pops up, alerting the user about the wrong input format, that means he/she did not follow the instructions given in the Help window.

In case of subtraction, the result will be obtained by subtracting the second polynomial from the first one. The division would follow the same principle, if I would have it. For derivation and integration, only the first polynomial will be processed, anything in the text field of the second polynomial will be ignored, and won't affect the result.

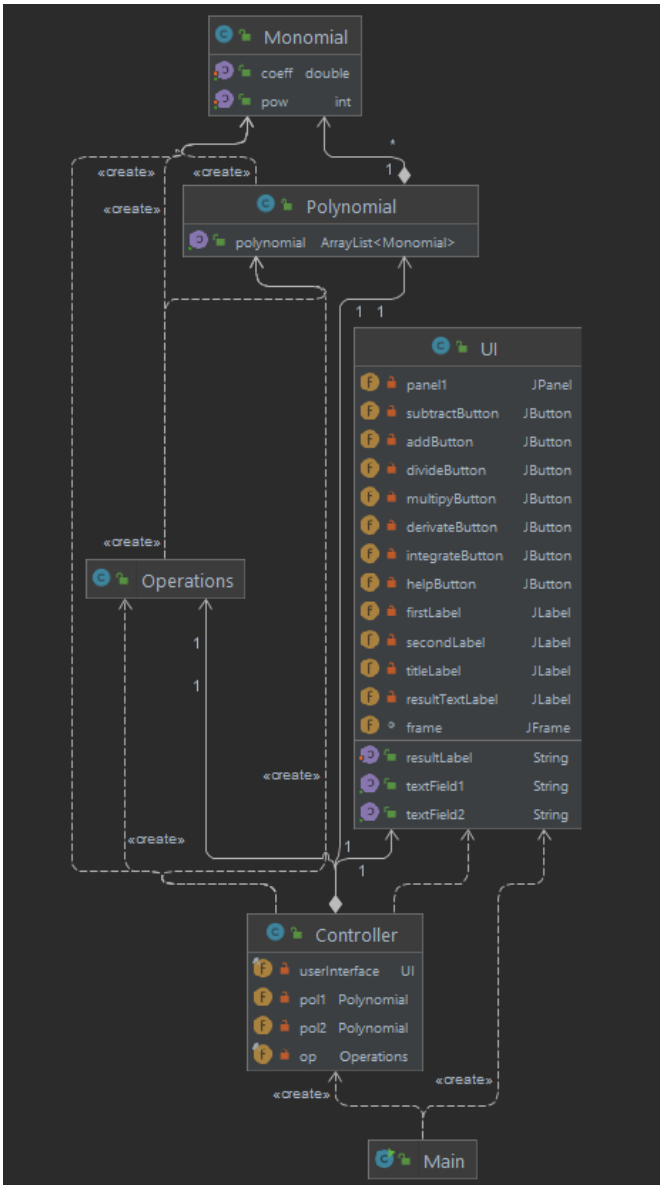
Implementation

Diagrams

Class diagram

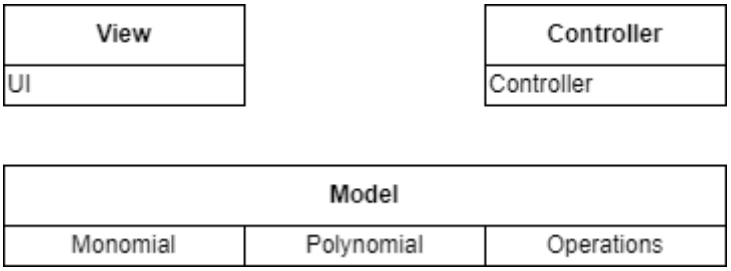
In the course of creating the project, several diagrams were made to help with the development.

Below you will see a UML (Unified Modelling Language) diagram, used to visualize, specify, and document the software system. This diagram contains all the Java classes used in the implementation of this project, as well as their dependencies and relations with each other. This diagram is known as the class diagram.



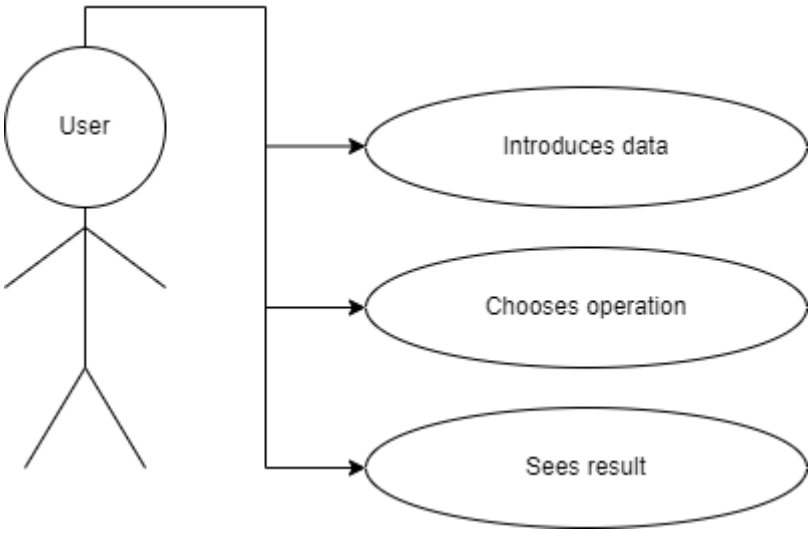
Package diagram

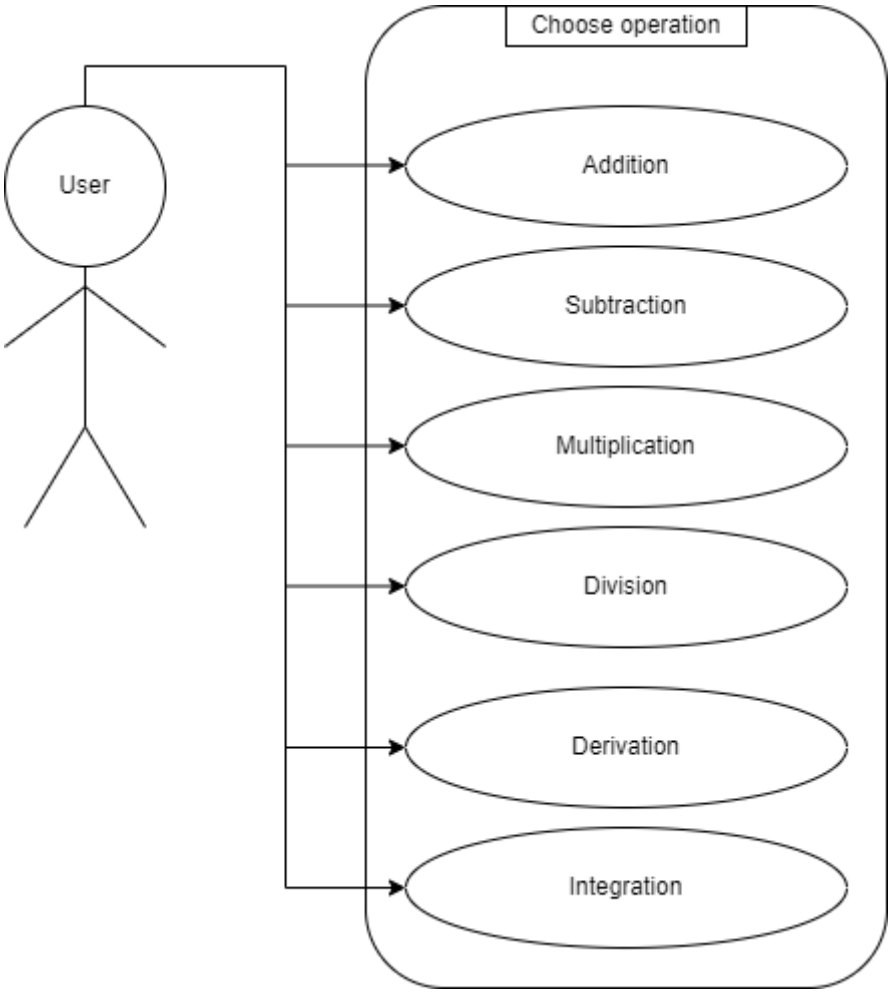
The next diagram shows the 3 main packages and the classes inside of them. The view package contains all the GUI elements, and it is what the user sees. The model package contains the Monomial, Polynomial and Operations classes, and lastly, the controller package contains the Controller class, which is the “brains” of the whole application.



Use case diagram

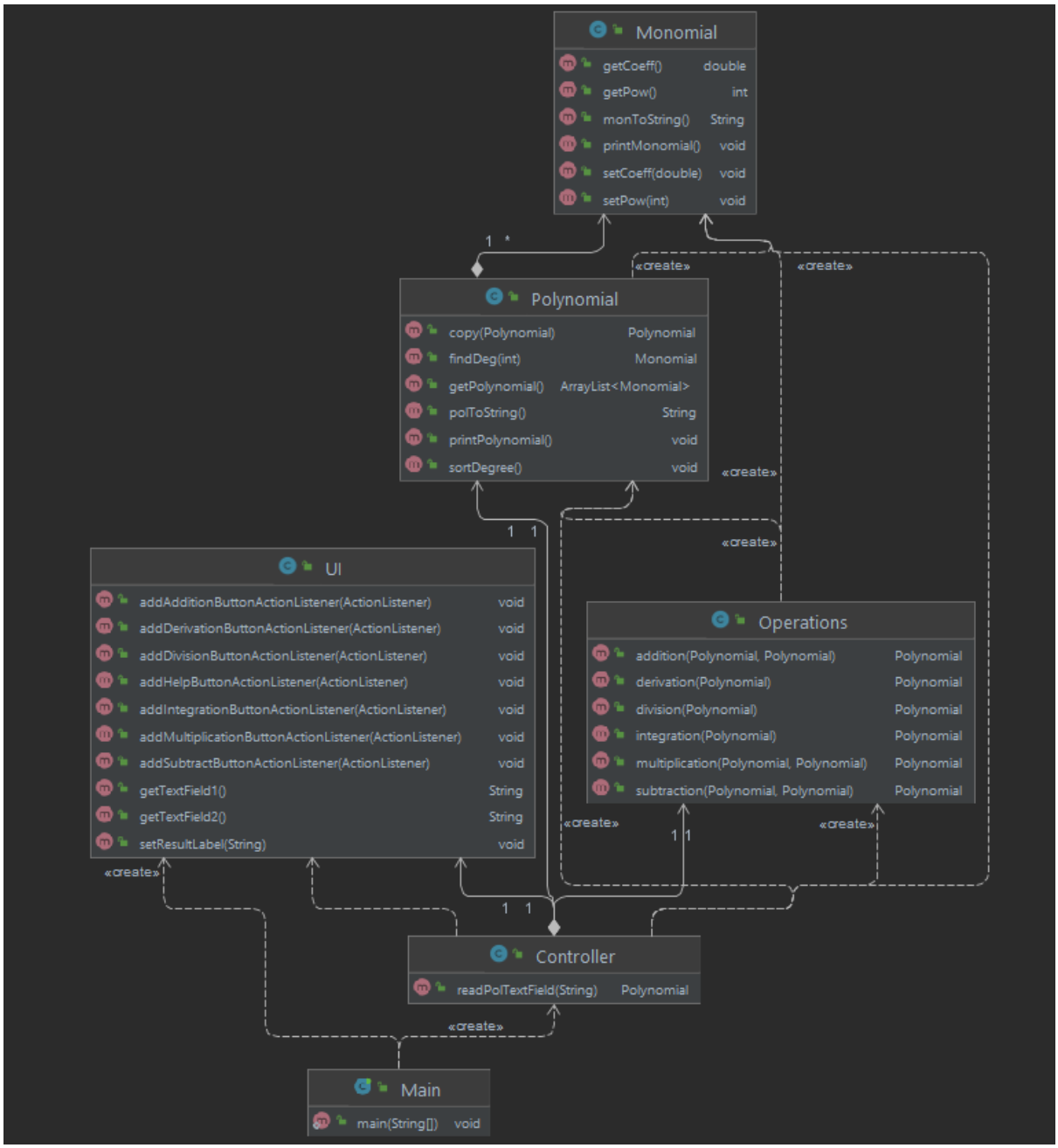
The following diagrams are the use case diagrams, depicting the different cases a user finds itself in.





Class diagram (with methods)

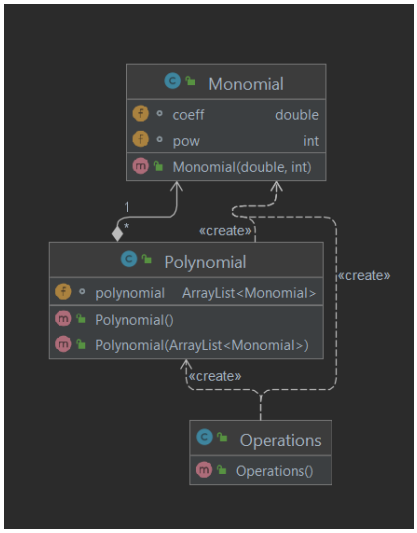
Since showing the methods on the main class diagram would make it way too clustered and impossible to read, the next diagram shows just the methods in all the classes.



Several getters, setters, and constructors were used.

Packages

Model



The model package contains 3 classes, depending on each other. The polynomial is a list of monomials, and the operations are performed on polynomials.

Further you will see code segments for all the classes in this package.

The Monomial class has two attributes: coefficient and power. Along with the getters, setters and constructor, you will see a printMonomial method that was used for testing throughout the development, and also a monToString method, which transfroms a Monomial type object into a String, for easier printing.

```
package model;

public class Monomial {
    double coeff;
    int pow;

    public double getCoeff() {
        return coeff;
    }

    public void setCoeff(double coeff) {
        this.coeff = coeff;
    }

    public int getPow() {
        return pow;
    }

    public void setPow(int pow) {
        this.pow = pow;
    }

    public Monomial(double coeff, int pow) {
        this.coeff = coeff;
        this.pow = pow;
    }

    //prints the monomial
    public void printMonomial() {
        if (coeff > 0)
            System.out.println("+" + coeff + "x^" + pow + " ");
        else if (coeff < 0)
            System.out.print(coeff + "x^" + pow + " ");
        else
            System.out.print("+" + 0 + " ");

    }
```

```
    //transforms monomial into printable string
    public String monToString() {
        StringBuilder sBuilder = new StringBuilder();

        if (coeff > 0)
            return sBuilder.append("+
").append(coeff).append("x^").append(pow).append(" ").toString();
        else if (coeff < 0)
            return sBuilder.append("
").append(coeff).append("x^").append(pow).append(" ").toString();
        else
            return "";
    }
}
```

The next class is the Polynomial class which contains an array list of Monomial type objects, 2 similar methods to print the polynomial and transform it into a string, and additionally, methods to sort the polynomial by its degree, copy one polynomial into another, and find its degree.

```
package model;

import java.util.ArrayList;

public class Polynomial {
    ArrayList<Monomial> polynomial = new ArrayList<>();

    public Polynomial() {
    }

    public Polynomial(ArrayList<Monomial> polynomial) {
        this.polynomial = polynomial;
    }

    public ArrayList<Monomial> getPolynomial() {
        return polynomial;
    }

    //sorts the polynomial by degree
    public void sortDegree() {
        polynomial.sort((m1, m2) -> Integer.compare(m2.getPow(),
m1.getPow()));
    }

    //just prints the polynomial in the console
    //used for initial testing
    public void printPolynomial() {
        for (Monomial currentMonom : this.getPolynomial()) {
            currentMonom.printMonomial();
        }
        System.out.println(" ");
    }

    //finds the degree
    public Monomial findDeg(int deg) {
        Monomial result = null;

        for (Monomial current : this.getPolynomial()) {
            if (current.getPow() == deg)
                result = current;
```

```
        }
        return result;
    }

    //copies the given polynomial into another
    public Polynomial copy(Polynomial p) {
        Polynomial copy = new Polynomial();

        for (Monomial current : p.getPolynomial()) {
            int pow = current.getPow();
            double coeff = current.getCoeff();

            Monomial mon = new Monomial(coeff, pow);
            copy.getPolynomial().add(mon);
        }
        return copy;
    }

    //creates a string from the polynomial type object for easy printing
    public String polToString() {
        StringBuilder builder = new StringBuilder();

        for (Monomial current : this.getPolynomial()) {
            builder.append(current.monToString());
        }
        return builder.toString();
    }
}
```

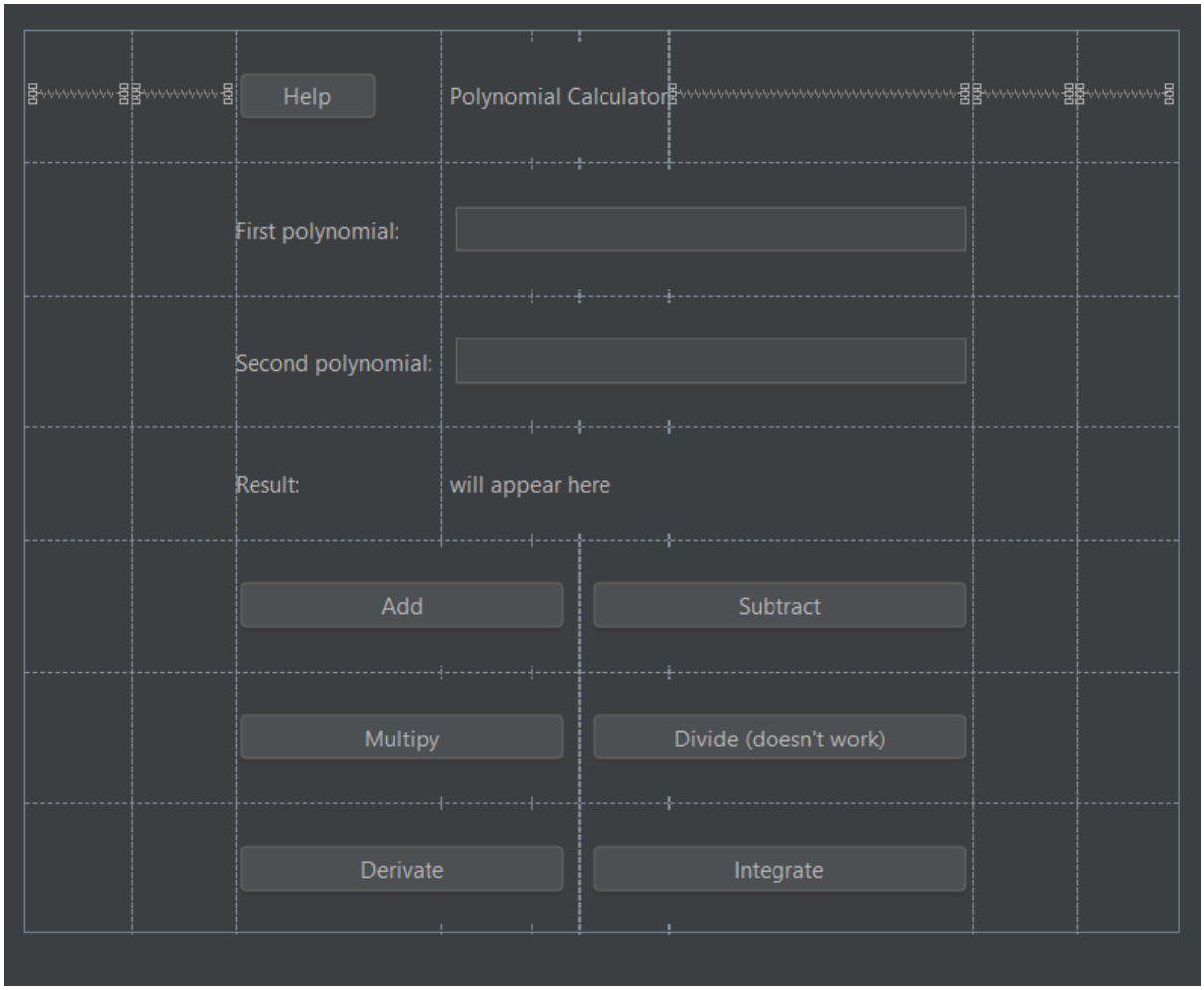
The Operations class contains all the operations between polynomials the user can choose from to perform, except the division. Only one such operation will be presented here, the rest you may find on the Gitlab repository.

```
package model;
import java.util.Iterator;

public class Operations {

    //ADDITION
    public Polynomial addition(Polynomial p1, Polynomial p2) {
        Polynomial sum = p1.copy(p1);

        for (Monomial current : p2.getPolynomial()) {
            int deg = current.getPow();
            double coeff = current.getCoeff();

            Monomial searched = sum.findDeg(deg);
            if (searched == null) {
                sum.getPolynomial().add(current);
            } else {
                double oldCoeff = searched.getCoeff();
                searched.setCoeff(coeff + oldCoeff);
            }
        }

        sum.sortDegree();
        return sum;
    }
```

View

This class contains all the user interface elements that were used to create the visible GUI of the application, such as buttons, labels and text fields. It's class diagram is the same as before, so it won't be shown again.

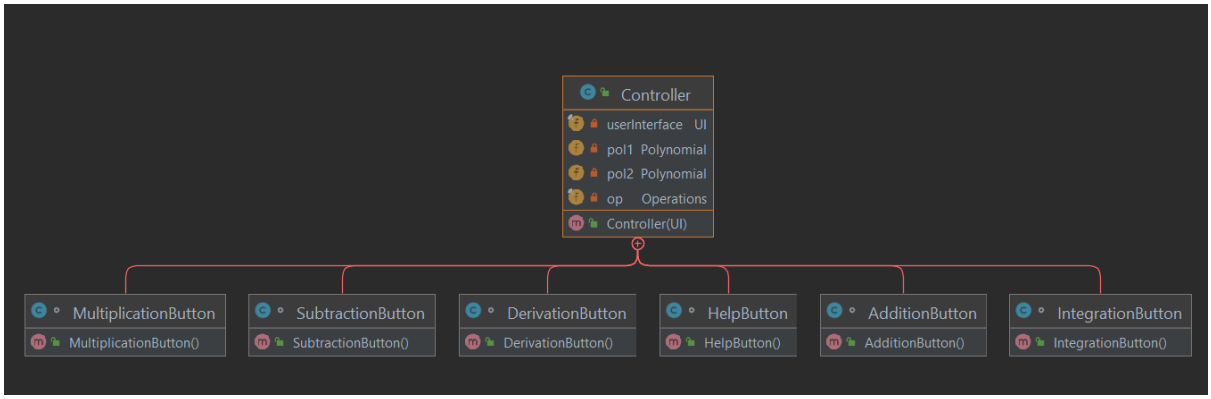
The UI was created in the Java Swing editor, which lets you quickly position any component on the window.



Controller

This class is the "brains" of the application. It fetches the input polynomials from the UI, processes them with the chosen operation, and then outputs the result for the user. This class contains inner classes for each button that performs any action This way the view package is kept clear of any polynomial processing methods.

Furthermore, this class contains the RegEx, the method which transforms the polynomial input string into a Polynomial type object, which is by far the most important method of the whole application.



The diagram shows all the inner classes for the functional buttons.

Out of the 6 classes that implement the ActionListener class, only the one that performs the addition will be shown, because it's easy to understand what happens, and the other classes follow this same pattern.

```
public class Controller {

    private final UI userInterface;
    private Polynomial pol1;
    private Polynomial pol2;
    private final Operations op;


    //reads the polynomial from the textfield
    public Polynomial readPolTextField(String textField) {
        Polynomial current = new Polynomial();
        String polString = textField;

        try {
            polString = polString.replaceAll("\\s", ""); // \\s means
whitespace \\s = " "
            if (polString.charAt(0) == '+') {
                polString = polString.substring(1);
            }

                for (String val : polString.split("((?=\\+)|(?=-))")) {
                double coeff;
                int pow;
                int positionX = val.indexOf("x");
                int positionPow = val.indexOf("^");

                try {
                    coeff = Double.parseDouble(val.substring(0,
positionX));
                    pow = Integer.parseInt(val.substring(positionPow + 1));
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(null, "Wrong input
format. Press HELP for more info.");
                    return null;
                }

                if (coeff != 0) {
                    Monomial mon = new Monomial(coeff, pow);
                    current.getPolynomial().add(mon);
                }
            }
```

```
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, "Wrong input format. Press
HELP for more info.");
            return null;
        }
        return current;
    }

    public Controller(UI view) {
        userInterface = view;
        op = new Operations();

        view.addAdditionButtonActionListener(new AdditionButton());
    }

    //classes to be used in the actionlistener methods
    //ADDITION
    class AdditionButton implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            pol1 = readPolTextField(userInterface.getTextField1());
            pol2 = readPolTextField(userInterface.getTextField2());

            Polynomial sum;

            try {
                sum = op.addition(pol1, pol2);
                userInterface.setResultLabel(sum.polToString());
            } catch (Exception e1) {
                System.out.println(e1.getMessage());
            }
        }
    }
```

The readPolTextField method is the RegEx – creates a Polynomial from a string. This method is called at the beginning of each ActionListener class to process the input polynomials, and after that the respective operation is performed on the two processed polynomials.

In the Controller constructor, the methods created in the UI are called and they are given as parameter the respective ActionListener class, this way making the connection between the view and the controller packages.

Unit Testing and results

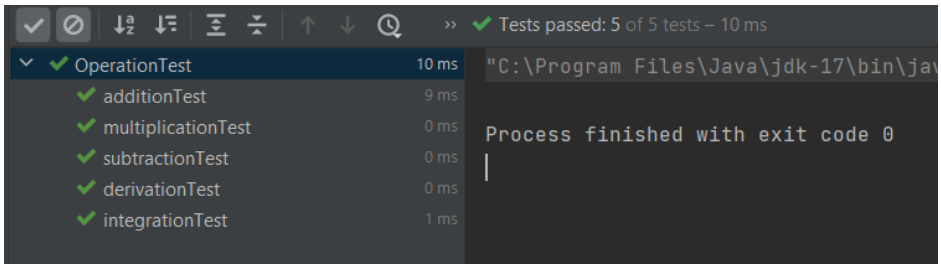
To test the correctness of the application, an additional OperationTest class was created, to whom are given both the input and the output polynomials. If the test runs successfully, the application functions properly. The unit test also highlights any errors, if there are any.

This class is found in a separate test directory, outside of the rest of the application. Once again, only the test for the addition operation will be shown, because the other operations follow the same pattern.

```
public class OperationTest {
    UI view = new UI();
    Operations op = new Operations();
    Polynomial pol1 = new Polynomial();
    Polynomial pol2 = new Polynomial();
    Controller co = new Controller(view);
```

```
    @Test
    public void additionTest(){
        pol1 = co.readPolTextField("4x^5-4x^3+3x^1");
        pol2 = co.readPolTextField("-3x^3-3x^0");
        assertEquals("+ 4.0x^5  -7.0x^3 + 3.0x^1  -3.0x^0 ",
op.addition(pol1, pol2).polToString());
    }
}
```

Once the test is run, the check marks and the 0 exit code show that all the tests were successfully conducted.



Further development

As stated, several times in the documentation before, the application is missing the division operation as a possible further development of the project.

Conclusion

This polynomial calculator can take any two polynomials and add them, subtract them, multiply them. If we want to derivate or integrate a polynomial it is also possible. The GUI is easy to understand and the way we have to provide the polynomials is self-explanatory. The project uses many object-oriented paradigms and is easy to develop and work on in the future.

This assignment proved to be useful in helping me to grasp the idea of Object-Oriented programming, encapsulation, and the MVC architecture.

Unit testing with JUnit was a new thing to me, and I think this was the perfect way to get introduced to it.

Bibliography

https://dsrl.eu/courses/pt/
https://stackoverflow.com/
https://www.youtube.com/
https://www.geeksforgeeks.org/

Several topics were used from these links.