DOCUMENTATION

ASSIGNMENT 1

STUDENT NAME: **Mureşan Mara**
GROUP: **30423**

CONTENTS

1. Assignment Objective

- Main Objective
    - Create (design and implement) a polynomial calculator with a user-friendly graphical interface through which the user can insert 2 polynomials, select the mathematical operation to be performed (addition, subtraction, multiplication, division, derivative and integration) and view the result.
- Sub-objectives
    - Analyze the problem and identify the requirements (Section 2)
        - Examining the problem statement to understand the goals and constraints of the project
        - Identifying the specific requirements of the polynomial calculator, such as the supported mathematical operations, the input format for the polynomials and the desired user interface features
    - Design the polynomial calculator (Section 3)
        - Designing the architecture of the project and sketching the user interface
        - Creating a plan for how the calculator will function, including the data structures and algorithms needed to handle polynomial operations
    - Implement the polynomial calculator (Section 4)
        - Writing the code to implement the polynomial calculator based on the design structured at the previous step
        - Implementing the necessary classes, methods and GUI features
    - Test the polynomial calculator (Section 5)
        - Conducting different tests to ensure that the polynomial calculator meets the specified requirements and functions correctly
        - Testing each individual component/method and then testing the project as a whole

2. Problem Analysis, Modeling, Scenarios, Use Cases

- Functional requirements
    - The polynomial calculator should allow users to insert 2 polynomials P1 and P2 by typing their contents
    - The polynomial calculator should allow users to select the mathematical operation (addition, subtraction, multiplication, division, derivative, integration)
    - The polynomial calculator should allow users to delete the previous entered polynomials and the previous calculated result, in order to insert new polynomials
    - The polynomial calculator should perform the selected operation and display the result as a new polynomial called “Result”
    - The polynomial calculator should display an error/warning message if the user tries to perform an operation on some input polynomials that are not written in the required format (each polynomial term must have a sign, and before and after each sign there is a “space”)
    - The polynomial calculator should efficiently manage memory usage, ensuring that it can handle large polynomials without crashing or becoming unresponsive

- Non-functional requirements
    - The graphical interface should be visually appealing and user-friendly, with intuitive controls and clear feedback to the user (example: when performing the division operation, the polynomial calculator clearly specifies which polynomial is the quotient and which one is the reminder)
    - The polynomial calculator should respond to user inputs quickly and efficiently, with minimal delay in performing mathematical operations on polynomials
    - The polynomial calculator should be reliable and dependable, functioning correctly without crashing or producing errors under normal usage conditions

Use Cases

Description

Use Case: Divide polynomials

Primary Actor: User

Success Scenario Steps:

1. The user introduces 2 polynomials in the provided text boxes from the GUI
2. The user presses the button corresponding to the division operation
3. The polynomial calculator checks if the polynomials respect the correct structure (each polynomial term must have a sign, and before and after each sign there is a “space”)
4. The polynomial calculator performs the division of the 2 polynomials and displays the result in the form of another 2 polynomials, the quotient and the reminder

Alternative Sequence: Incorrect Inputs

- The user inserts polynomials with wrong structure
- The polynomial calculator displays an error message and provides an example of a correct polynomial
- The scenario returns to step 1

The descriptions of the use cases for the other operations are very similar to the description of the division operation.

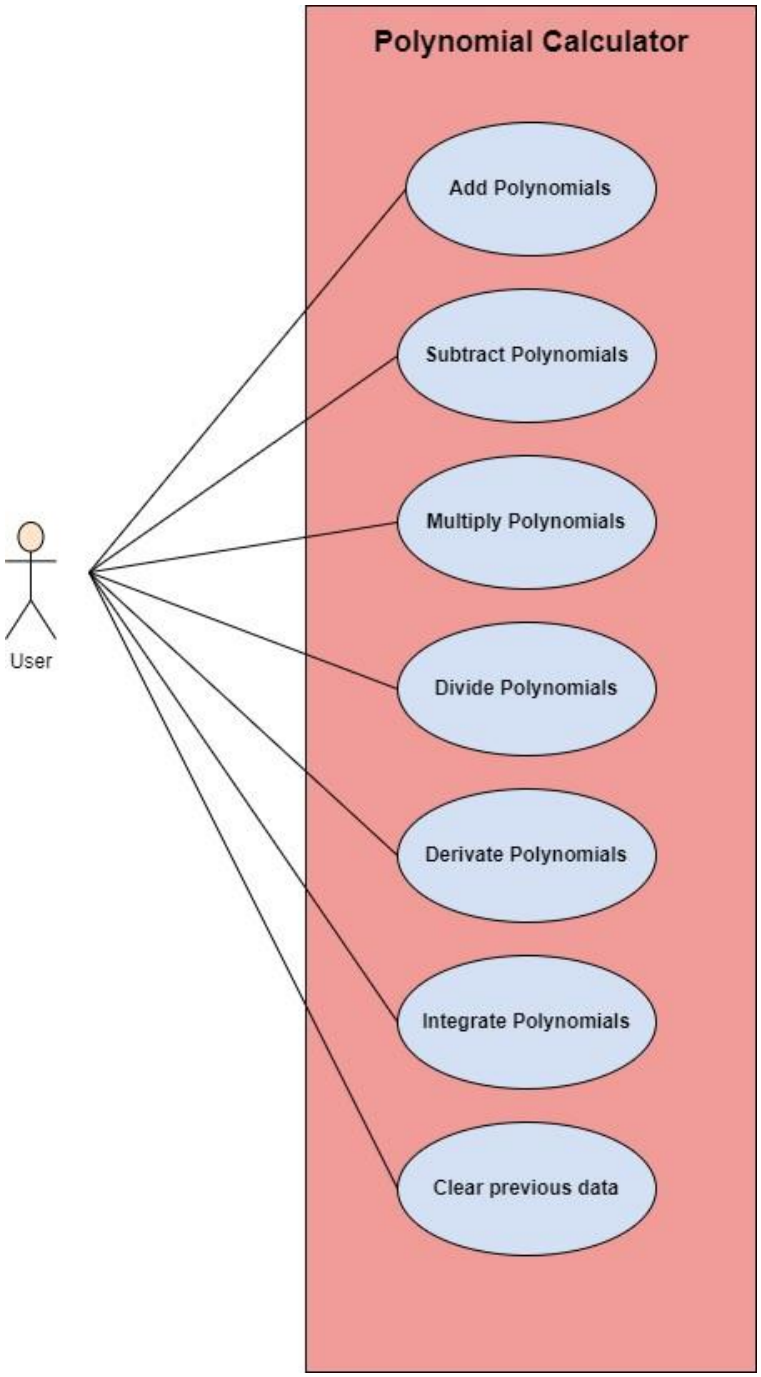
Use Case: Clear previous data

Primary Actor: User

Success Scenario Steps: (it doesn’t exist an “Alternative Sequence”)

1. The user clicks the “Clear” button
2. The polynomial calculator erases the 2 previously inserted polynomials together with the last result, in order to make space for new data to be inserted

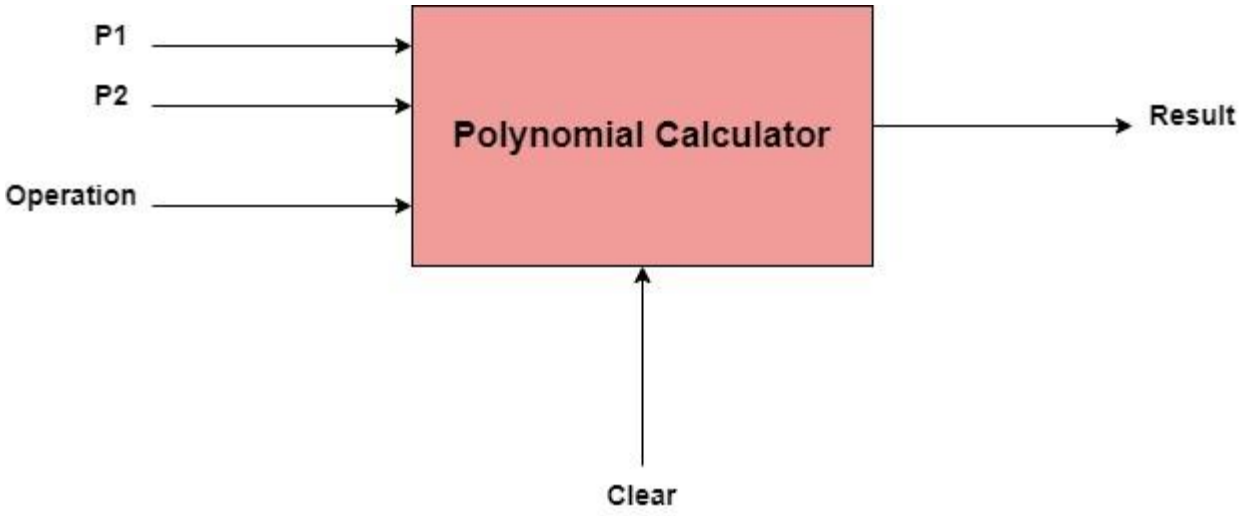
Use Case Diagram



Reference: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

3. Design

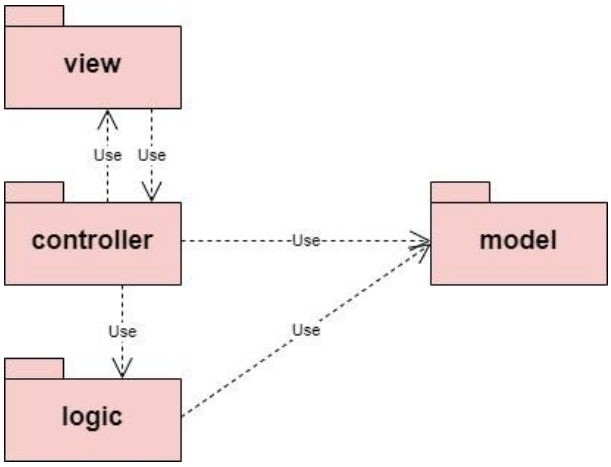
The System's "Black Box"



I organized the application into 4 packages:

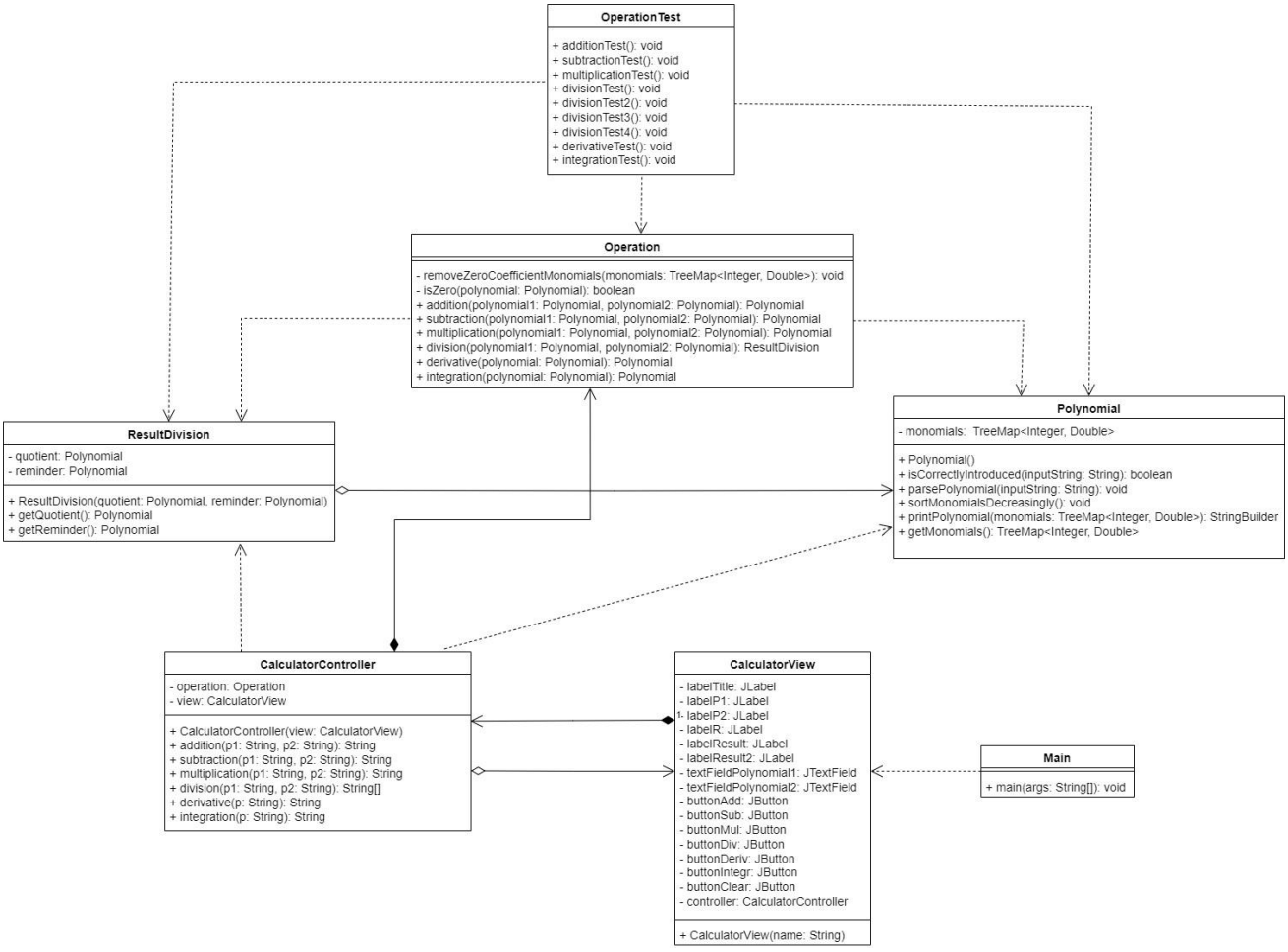
- model
    - contains the class that models the application data: "***Polynomial***"
- view
    - contains the class that builds the user interface, responds to user interactions and triggers appropriate actions within the calculator functionalities: "***CalculatorView***"
- controller
    - contains the class that is responsible for connecting all the components of the application: "**CalculatorController**"
    - takes user inputs as strings, coverts them into polynomials, performs operations on them and returns the results as strings in order to be used in the view
- logic
    - contains the classes implementing the mathematical operations functionality: "***Operation***" & "***ResultDivision***"

The UML Package Diagram



The UML Class Diagram

Reference: https://www.gleek.io/blog/uml-relationships



Used Data Structures

My first thought was that a polynomial is composed by many terms called monomials, so, in "Java language" this would translate as a polynomial is a collection of monomials.

I started the application using a HashMap for storing the terms/monomials of the polynomials.

- KEYS → degrees of the monomials: *Integer*
- VALUES → coefficients of the monomials: *Double*

Ongoing, I realized that I need to sort the monomials in decreasing order of the keys for a clean display of the result. So, I changed the HashMap into a TreeMap, because the TreeMap is sorted according to natural ordering of the keys, and then I performed a sorting decreasingly method.

Using a TreeMap was a really good idea also because, for performing the division, I needed to find the term with the maximum degree at each step. For this, I used the firstKey() feature of the TreeMap, instead of going through all the monomials each time, comparing their degrees, and finding the one with the maximum degree. It was straightforward that the monomial with the maximum degree is the first one in the monomials collection sorted in decreasing order.

Operations Complexity for TreeMap: (https://www.baeldung.com/java-collections-complexity)

- get: O(log(n))
- put: O(log(n))
- containsKey: O(log(n))
- remove: O(log(n))

Used Algorithms

I used the well-known mathematical algorithms for operating on polynomials. (Reference: https://dsrl.eu/courses/pt/materials/PT_2024_A1_S1.pdf)

The form of the polynomials:

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

$a_0, a_1, \ldots, a_n$ – the coefficients and 0, 1, …, n – the degrees

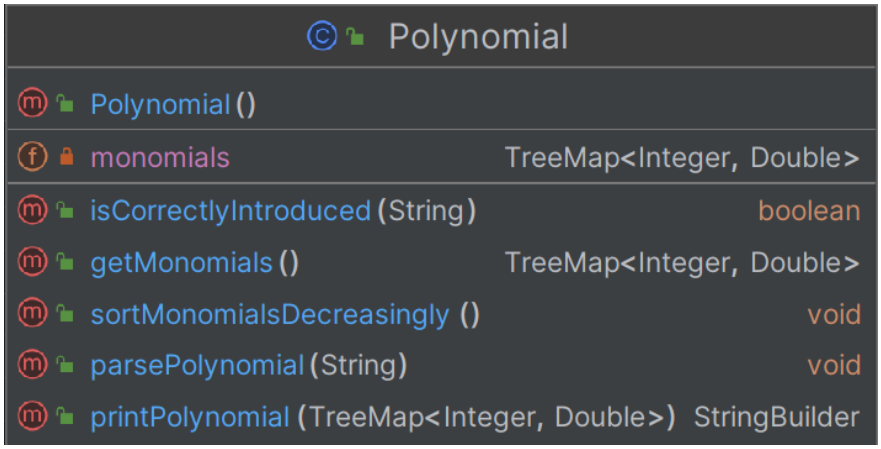
The most challenging algorithm was the division. For implementing this operation I followed the next pseudocode.

1. Order the monomials of the two polynomials P and Q in decreasing order according to their degree.
2. Find the polynomial with the highest degree and swap them (if it is necessary), such that the polynomial P has the highest degree.
3. Divide the first monomial of P to the first monomial of Q and obtain the first term of the quotient.
4. Multiply the quotient with Q and subtract the result of the multiplication from P obtaining the reminder of the division.
5. Repeat the procedure from step 3 considering the reminder as the new dividend of the division, until the degree of the reminder is lower than the degree of Q.

4. Implementation

The application contains 7 classes.

1. **Polynomial**



Each polynomial is in fact a **TreeMap of monomials**, with integer degrees (stored as keys) and double coefficients (stored as values).

The **constructor** initializes a TreeMap of monomials when a Polynomial object is created.

```
10 usages  new *
public boolean isCorrectlyIntroduced(String inputString) {
    String regex = "([+-])\s(\d+)?(x(\^(\d+))?)?";
    if(!inputString.matches( regex: regex + "(\s*" + regex + ")*")) return false;
    return true;
}
```

isCorrectlyIntroduced(String inputString) method checks if the input string is correctly formatted to represent a polynomial. It uses a regular expression to validate the polynomial's pattern: each monomial has a sign, and before and after each sign there is a "space".

The **parsing** method traverses all the matcher groups from the regex string and extracts the degrees and the coefficients with which it populates the monomials TreeMap.

```
26 usages  Mara
public void parsePolynomial(String inputString) {
    String regex = "([+-])\s(\d+)?(x(\^(\d+))?)?";
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(inputString);

    while (matcher.find()) {
        int sign;
        if(matcher.group(1).startsWith("-")) sign = -1;
        else sign = 1;

        double coefficient;
        if (matcher.group(2) == null) coefficient = sign;
        else coefficient = sign * Double.parseDouble(matcher.group(2));

        int degree;
        if (matcher.group(3) == null) degree = 0;
        else if (matcher.group(4) == null) degree = 1;
        else degree = Integer.parseInt(matcher.group(5));

        if(monomials.containsKey(degree)){
            coefficient = monomials.get(degree) + coefficient;
        }

        monomials.put(degree, coefficient);
    }
}
```

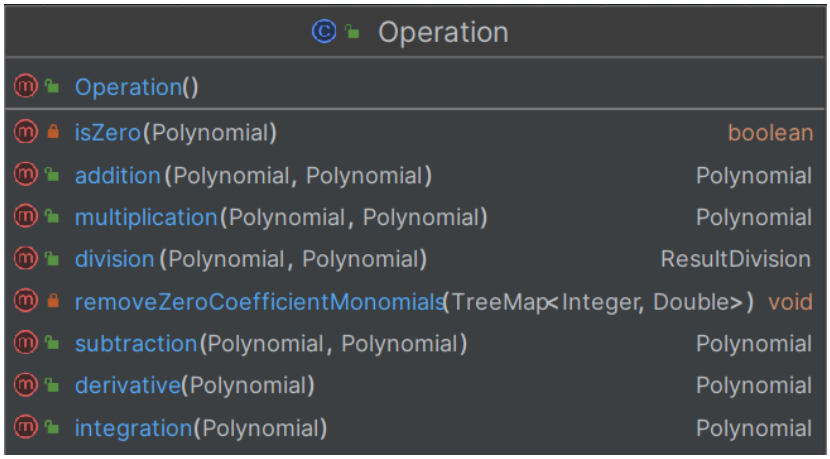
sortMonomialsDecreasingly() sorts the monomials in decreasing order of degrees.

printPolynomial(TreeMap<Integer, Double>) constructs a string representation of the polynomial. An important feature is that the double coefficients are displayed with 2 decimals

after the decimal point. (the inputs can have only integer coefficients, because the regex string is build in this way, but however, the results can have double coefficients)

The **getter** method retireves the monomials TreeMap.

2. Operation



The methods in this class manipulate polynomial data structures and perform arithmetic operations based on mathematical rules and algorithms.

removeZeroCoefficientMonomials(TreeMap<Integer, Double>) is a private method that removes monomials with zero coefficients from the given TreeMap. This method is very useful because is redundant to display the monomials with the 0 coefficient. Moreover, this method solves the problem of infinity loop, which I encountered during implementing the division operation (even if the coefficient was 0, the maximum degree remained always the same and the algorithm never went further).

```
2 usages  new *
private void removeZeroCoefficientMonomials(TreeMap<Integer, Double> monomials) {
    monomials.entrySet().removeIf(entry -> entry.getValue() == 0);
}
```

isZero(Polynomial polynomial) checks if the given polynomial is a zero polynomial.

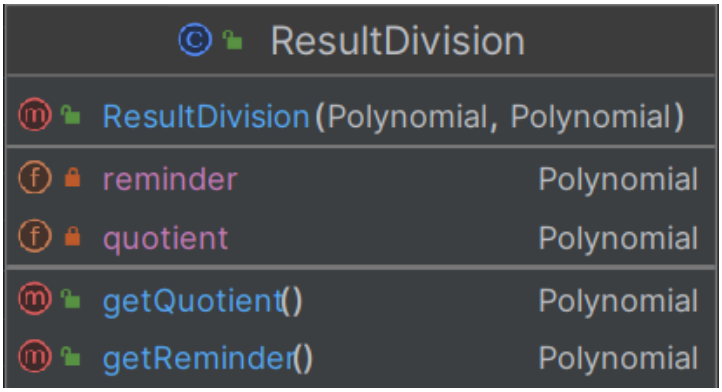
The **addition**, **subtraction**, **multiplication**, **derivative** and **integration** are very similar, following the mathematical algorithms on polynomials.

```
2 usages   Mara *
public Polynomial addition(Polynomial polynomial1, Polynomial polynomial2) {

    Polynomial result = new Polynomial();
    result.getMonomials().putAll(polynomial1.getMonomials());

    for(int degree: polynomial2.getMonomials().keySet()) {
        double coefficient2 = polynomial2.getMonomials().get(degree);
        if(result.getMonomials().containsKey(degree)) {
            double coefficient1 = result.getMonomials().get(degree);
            result.getMonomials().put(degree, coefficient1 + coefficient2);
        }
        else result.getMonomials().put(degree, coefficient2);
    }
    removeZeroCoefficientMonomials(result.getMonomials());
    return result;
}
```

The **division** operation was harder to implement. I followed the algorithm stated in the previous section. I implemented the division in order to work every time, because it always divides the polynomial with a bigger maximum degree to the polynomial with a smaller maximum degree (ex: if P2 > P1, it will compute P2/P1). Also, if one of the polynomials is 0, the result would be 0 (the division by 0 case is treated by displaying 0 as result).

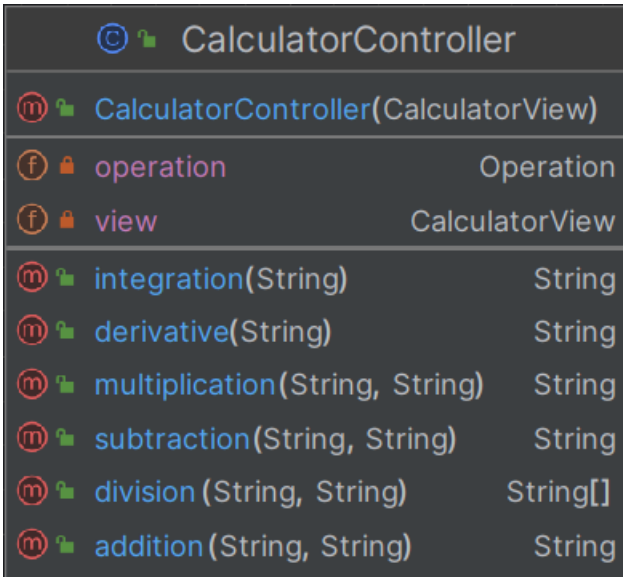
```
5 usages   Mara *
public ResultDivision division(Polynomial polynomial1, Polynomial polynomial2) {
    Polynomial P, Q = new Polynomial();
    Polynomial quotient = new Polynomial();
    Polynomial reminder = new Polynomial();
    int max1, max2, aux;
    if(isZero(polynomial1) || isZero(polynomial2)) { quotient.getMonomials().put(0,0.00); reminder.getMonomials().put(0,0.00); }
    else {
        polynomial1.sortMonomialsDecreasingly();
        polynomial2.sortMonomialsDecreasingly();
        max1 = polynomial1.getMonomials().firstKey();
        max2 = polynomial2.getMonomials().firstKey();

        if (max1 >= max2) { P = polynomial1; Q = polynomial2; }
        else { P = polynomial2; Q = polynomial1; aux = max1; max1 = max2; max2 = aux;}

        while (max1 >= max2) {
            int degree = max1 - max2;
            double coefficient = P.getMonomials().get(max1) / Q.getMonomials().get(max2);
            quotient.getMonomials().put(degree, coefficient);
            Polynomial step = new Polynomial();
            step.getMonomials().put(degree, coefficient);
            P = subtraction(P, multiplication(Q, step));
            if(P.getMonomials().isEmpty())  P.getMonomials().put(0,0.00);
            P.sortMonomialsDecreasingly();
            max1 = P.getMonomials().firstKey();
        }
        reminder = P;
    }
    return new ResultDivision(quotient, reminder);
}
```

3. **ResultDivision**



The division method should return 2 polynomials: the **quotient** and the **reminder**. For this, I created the ResultDivision class, with 2 private fields, a **constructor**, and **getter** methods to access the quotient and reminder polynomials separately.

4. **CalculatorController**



This class provides the interaction between the user interface (view), the logic (operation), and the data representation (polynomial model).

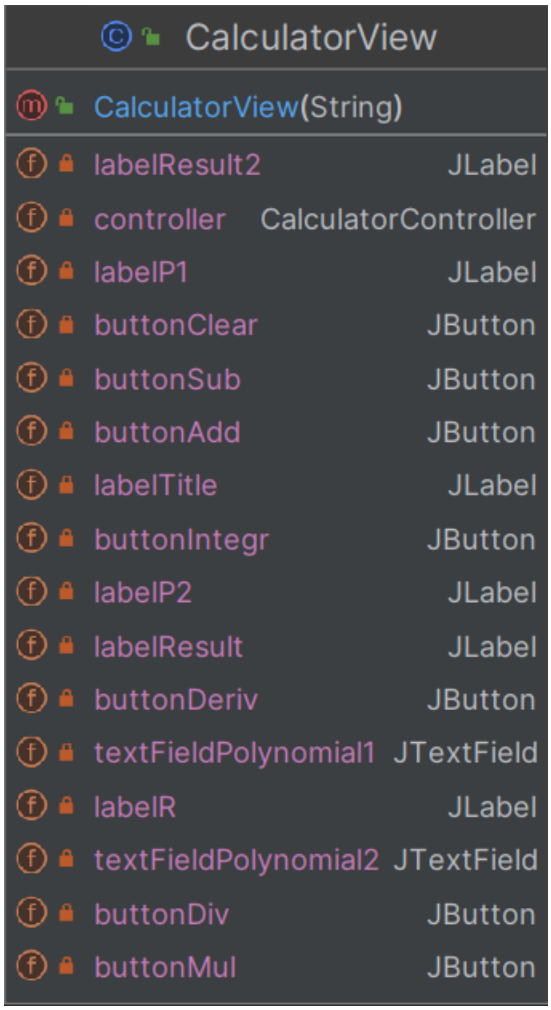
It creates **instances** of the "**Operation**" and the "**CalculatorView**" classes. It has a **constructor**.

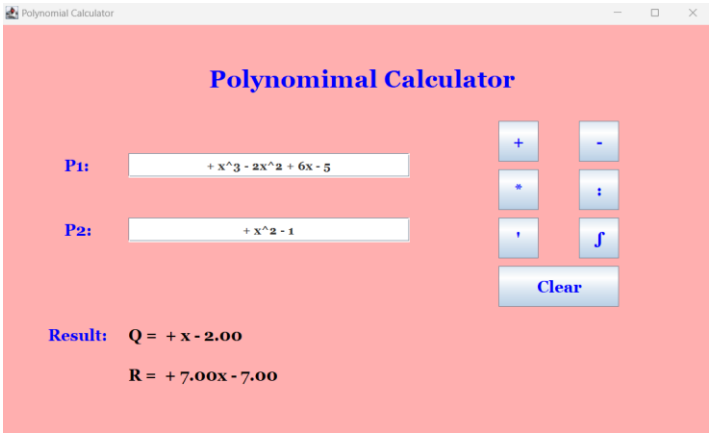
It contains methods for the **mathematical operations** similar to those from the "**Operation**" class. However, here, the methods' parameters are strings, so, the methods take the user inputs given as strings, they convert them into polynomials, call the operations from the "Operation"

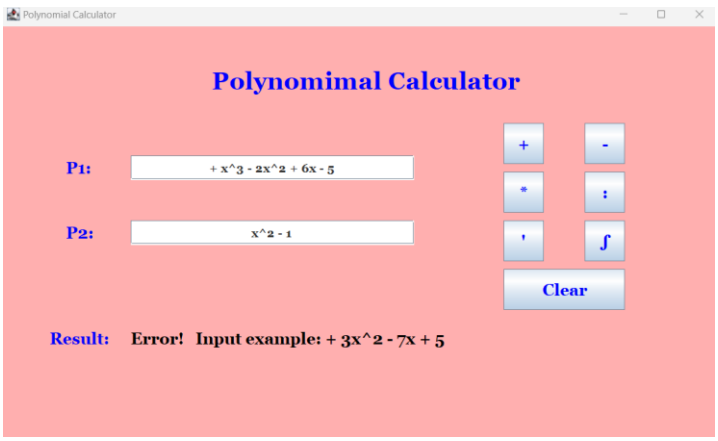
class and return the results/error messages also as strings (make the link between the user input strings and the operations on polynomials).

```
2 usages   Mara *
public String integration(String p) {
    Polynomial polynomial = new Polynomial();

    if(polynomial.isCorrectlyIntroduced(p)) {
        polynomial.parsePolynomial(p);

        Polynomial result = operation.integration(polynomial);
        result.sortMonomialsDecreasingly();
        StringBuilder resultString = result.printPolynomial(result.getMonomials());
        return resultString.toString() + " + C";
    }
    else return "Error!   Input example: + 3x^2 - 7x + 5";
}
```

5. CalculatorView







This class represents the **graphical user interface (GUI)** of the polynomial calculator application. I utilized **Swing** components to create a window with input textFields, buttons for mathematical operations and “Clear” (erases everything that was written before), and labels to display the results. (Reference: https://dsrl.eu/courses/pt/materials/PT_2024_A1_S2.pdf)

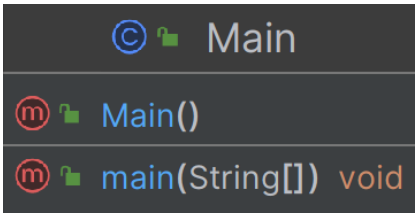
I started by following this tutorial and then I customized the calculator as I wanted: https://www.youtube.com/watch?v=GG950kz9p5Y

It contains a **constructor** with the title of the window as parameter. This constructor initializes the GUI components, sets their properties (font, color, bounds), and adds action listeners to the buttons to handle the user interactions.

When a button is clicked, corresponding actions are performed, such as invoking methods from the “CalculatorController” class, retrieving inputs from text fields, and uptdating the result labels accordingly.

```
buttonDeriv = new JButton( text: "'");
buttonDeriv.setFont(new Font( name: "Georgia", Font.BOLD,  size: 20));
buttonDeriv.setForeground(Color.BLUE);
buttonDeriv.setBounds( x: 620,  y: 240,  width: 50,  height: 50);
 Mara *
buttonDeriv.addActionListener(new ActionListener() {
    Mara *
    @Override
    public void actionPerformed(ActionEvent e) {
        labelResult.setText("");
        labelResult2.setText("");
        String polynomial1 = textFieldPolynomial1.getText();
        String polynomial2 = textFieldPolynomial2.getText();
        String result1 = "D1 = " + controller.derivative(polynomial1);
        String result2 = "D2 = " + controller.derivative(polynomial2);
        labelResult.setText(result1);
        labelResult2.setText(result2);
    }
});
add(buttonDeriv);
```

6. **Main**



This class serves as an entry point of the plynomial calculator application. It creates an instance of the “CalculatorView” class, sets up the main JFrame window, and displays it to the user.

The application will exit when the user closes the window.

```
Mara *
public class Main {
    Mara *
    public static void main(String[] args) {

        JFrame frame = new CalculatorView( name: "Polynomial Calculator");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(null);
        frame.pack();
        frame.setBounds( x: 400,  y: 200,  width: 900,  height: 550);
        frame.getContentPane().setBackground(Color.PINK);
        frame.setVisible(true);
    }
}
```

7. **OperationTest**

This class contains the test cases to verify the functionality of the operations. (Section 5)

5. Results

I performed unit testing with **JUnit**.

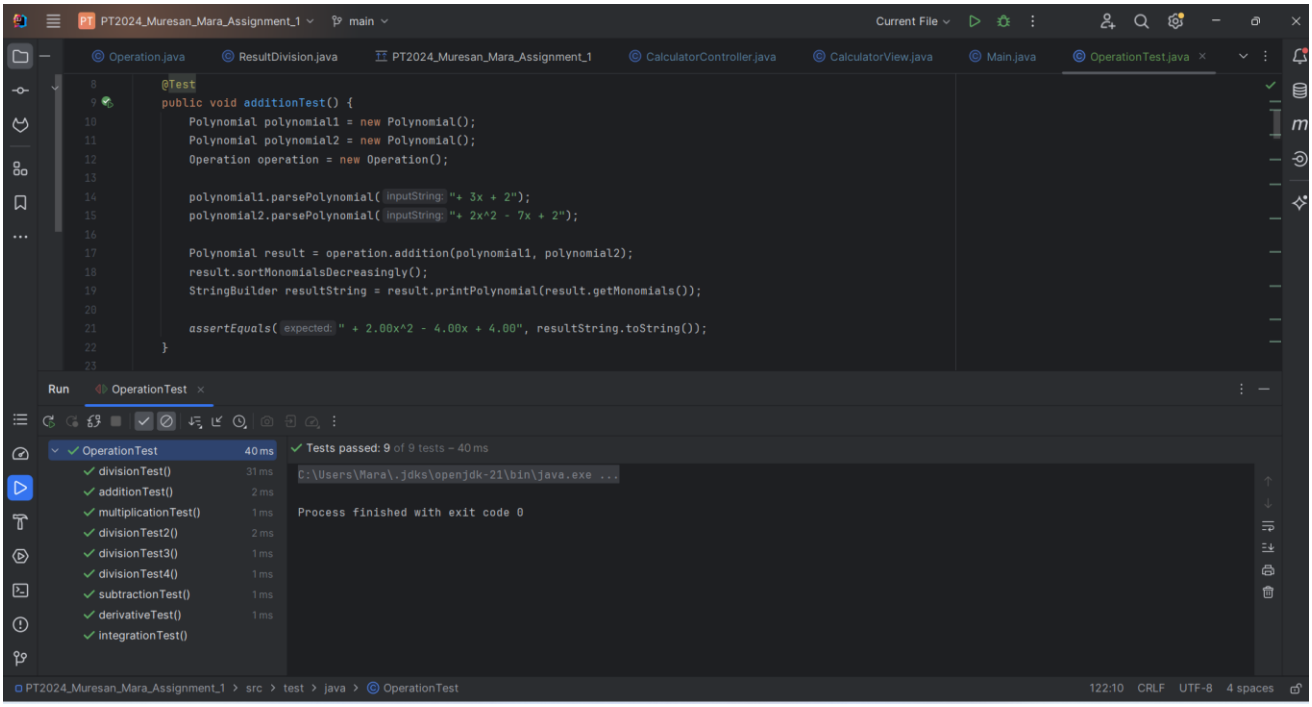
(Reference: https://dsrl.eu/courses/pt/materials/PT_2024_A1_S3.pdf)

Each test method is annotated with "**@Test**", indicating that it is a test case.

The "**assertEquals()**" method is used to compare the expected result with the actual result obtained from the tested method.

Special cases, such as division by 0 or division by opposite terms, are also tested to ensure robustness and correctness.

Overall, the "OperationTest" class ensures the correctness and reliability of the polynomial operation methods implemented in the "Operation" class.



```
new *
@Test
public void divisionTest3() {
    Polynomial polynomial1 = new Polynomial();
    Polynomial polynomial2 = new Polynomial();
    Operation operation = new Operation();

    polynomial1.parsePolynomial( inputString: "+ x^3 - 2x^2 + 6x - 5");
    polynomial2.parsePolynomial( inputString: "+ 0");

    ResultDivision result = operation.division(polynomial1, polynomial2);
    Polynomial quotient = result.getQuotient();
    quotient.sortMonomialsDecreasingly();
    Polynomial reminder = result.getReminder();
    reminder.sortMonomialsDecreasingly();

    StringBuilder quotientString = quotient.printPolynomial(quotient.getMonomials());
    StringBuilder reminderString = reminder.printPolynomial(reminder.getMonomials());

    assertEquals( expected: " + 0.00", quotientString.toString());
    assertEquals( expected: " + 0.00", reminderString.toString());
}
```

```
new *
@Test
public void divisionTest4() {
    Polynomial polynomial1 = new Polynomial();
    Polynomial polynomial2 = new Polynomial();
    Operation operation = new Operation();

    polynomial1.parsePolynomial( inputString: "+ x^2 - 1");
    polynomial2.parsePolynomial( inputString: "- x^2 + 1");

    ResultDivision result = operation.division(polynomial1, polynomial2);
    Polynomial quotient = result.getQuotient();
    quotient.sortMonomialsDecreasingly();
    Polynomial reminder = result.getReminder();
    reminder.sortMonomialsDecreasingly();

    StringBuilder quotientString = quotient.printPolynomial(quotient.getMonomials());
    StringBuilder reminderString = reminder.printPolynomial(reminder.getMonomials());

    assertEquals( expected: " - 1.00", quotientString.toString());
    assertEquals( expected: " + 0.00", reminderString.toString());
}
```

6. Conclusions

In conclusion, this project was a success, because I solved all the requirements for the polynomial calculator.

Through this assignment, I understood better how to represent, manipulate and perform mathematical operations on polynomials, all by using Java programming techniques. Also, I clarified the OOP principles such as aggregation, composition, encapsulation, inheritance and polymorphism. It was my first time writing tests with Junit and I can say that I really liked this method of testing. It was an easy and straightforward way to check if I implemented correctly all the operations. I also used Swing for the first time for creating the graphical user interface and it was a little bit challenging, because writing the code and only after running the program the interface is displayed is hard to follow and check. I think that it would have been easier to use Scene Builder, where we could place the GUI components with drag and drop on a panel to make an idea about how the GUI will look (also the JavaFX code would have been generated automatically).

As future developments, I could modify the regex string to be able to write input polynomials with double (real) coefficients. I could also extend the functionalities to support additional mathematical operations, such as finding roots. I could also improve the GUI to look better. I could add graphical visualization capabilities to plot the polynomials on a graph, providing users a visual representation of the polynomials they are working with. Also, a good idea would be to implement memory management features to allow users to store and recall previously entered polynomials.

7. Bibliography

In addition to the links added during the documentation in the specific section, I studied the Lectures and the Assignment Guides posted on the *Fundamental Programming Techniques* website: https://dsrl.eu/courses/pt/

I drew the diagrams using this website: https://app.diagrams.net/