

Documentation

Assignment 1

POLYNOMIAL CALCULATOR

Student name: Bojan Darian-Adelin

Group: 30423

CONTENTS

Contents	1
Assignment Objective	2
Problem Analysis, Modeling, Scenarios, Use Cases	3
Design of the application.....	5
Implementation	7
Results	13
Conclusion	14
Bibliography	15

ASSIGNMENT OBJECTIVE

The objective of the assignment is to design and implement a graphical interface for a polynomial calculator, where a user can insert polynomials and perform various operations like addition, subtraction, multiplication, division, integration, or differentiation. The polynomial must have a single variable and integer coefficients.

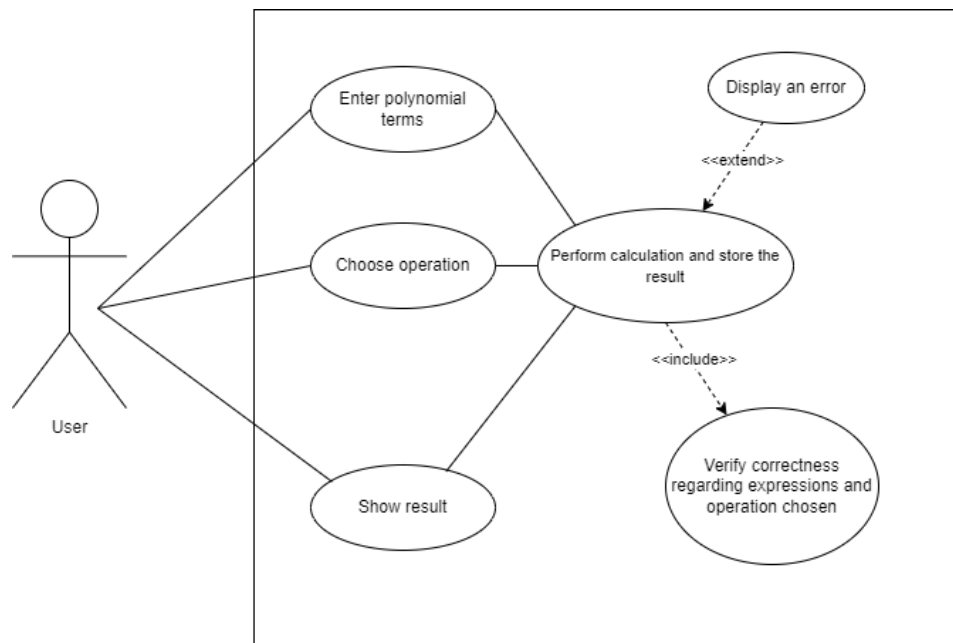
To achieve this task, it is necessary to satisfy the subsequent sub-objectives:

- Use object-oriented programming style (encapsulation, appropriate classes etc.).
- Implement a graphical user interface in Java Swing.
- Implement the correct operation methods.
- Usage of data structures
- Good organization of the code
- Creation of a JUnit class for testing and verification.

PROBLEM ANALYSIS, MODELING, SCENARIOS, USE CASES

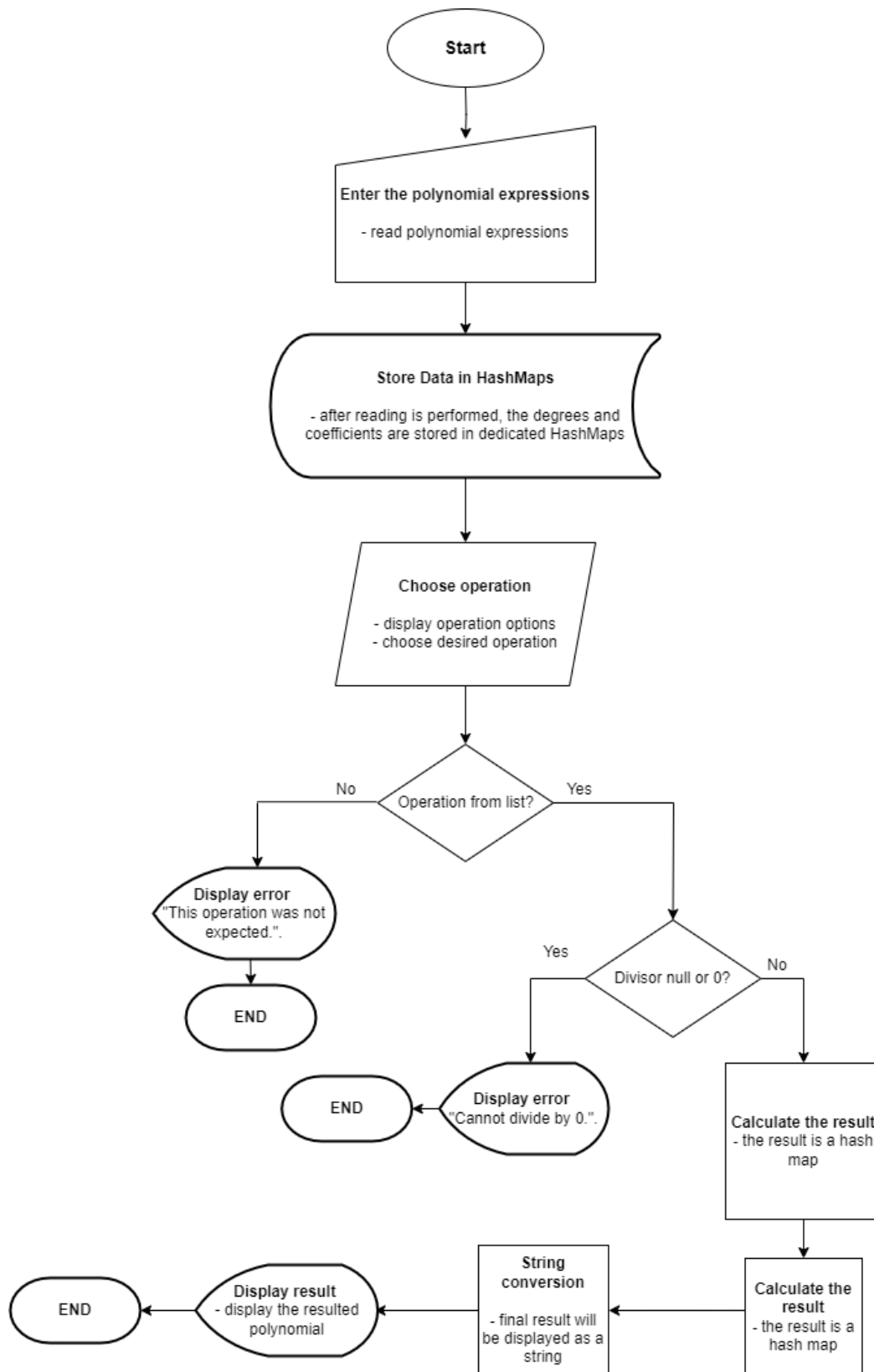
In order to solve this task, it is important to take into consideration the class model and the use cases such that we can have a clear view of the future development.

By starting with the use case diagram, we can gain a clearer understanding of the application we are developing. The user has 3 main interactions with the graphical interface: they need to enter the polynomial expressions, choose the operation, which is to be performed, and, finally, display the result by pressing a button.



Firstly, the polynomial terms are read, then stored in data structures to facilitate calculation and result storage. Subsequently, the program checks for errors (division by 0 or the operation name was not an accepted one). In case of division by 0, a result with a polynomial will not be displayed, but an error message instead ("Cannot divide by 0"). If the operation is not one of the desired ones (addition, subtraction, multiplication, division, differentiation, or integration) the GUI will display the message "This operation was not expected."

The following **flowchart** outlines the actions within the application:

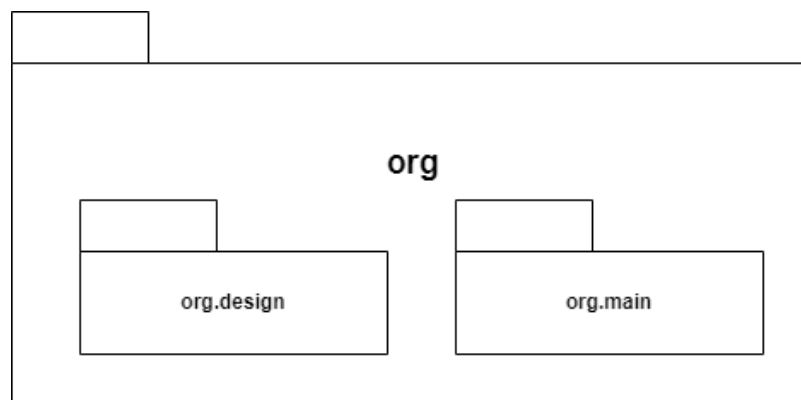


DESIGN OF THE APPLICATION

Classes and their responsibility:

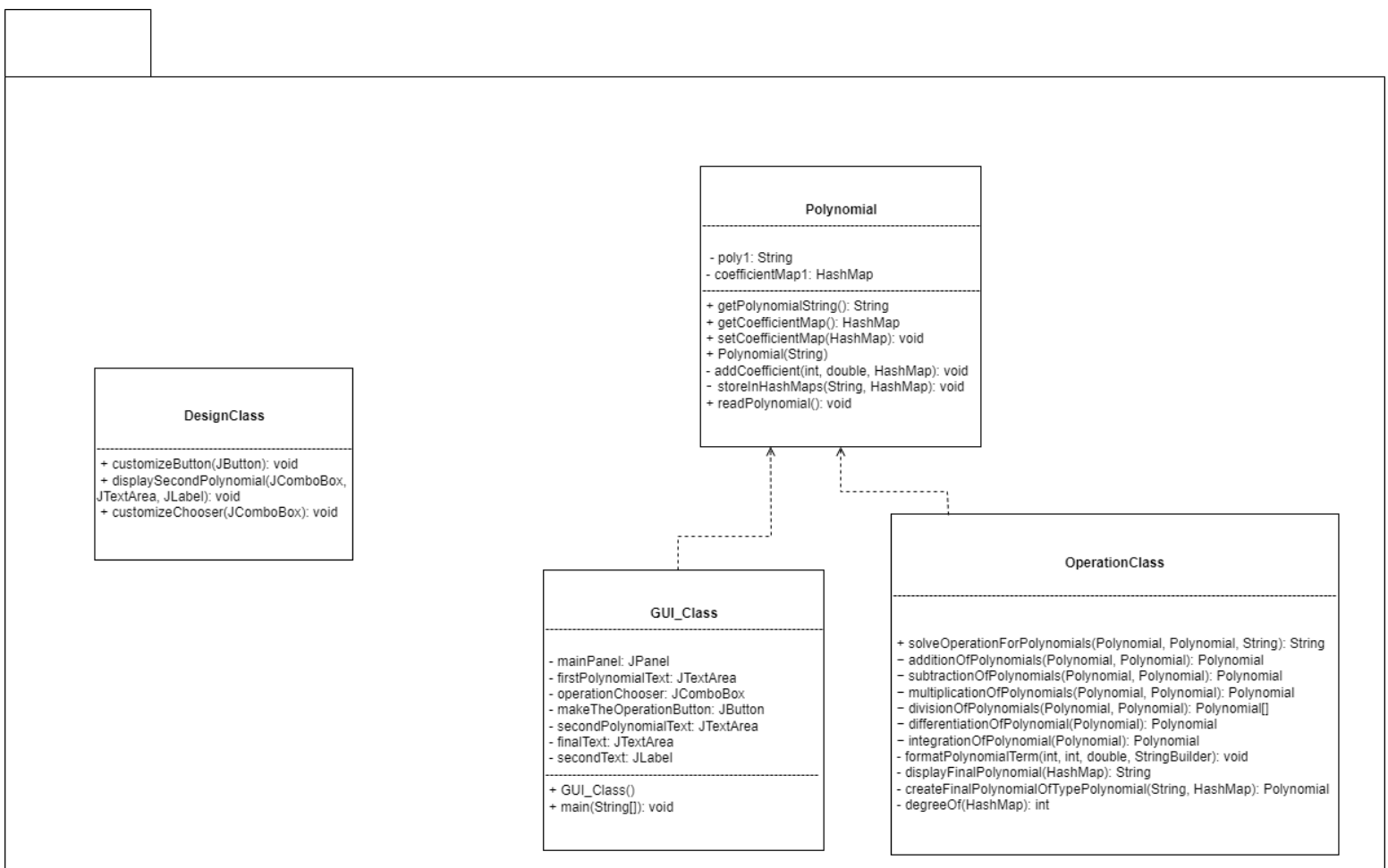
- GUI_Class - This class handles user interactions. The inputs are saved, due to the GUI_Class, in strings which are then parsed to other classes which perform the operations.
- DesignClass – contains static methods for customizing the design of the GUI_Class.
- Polynomial - This class encapsulates the polynomial string and its corresponding coefficient map. It provides methods for reading and manipulating polynomial terms, ensuring data integrity.
- OperationClass - Provides static methods for performing various operations on polynomials, such as addition, subtraction, multiplication, division, differentiation, and integration. The logic of polynomial operations is kept private, ensuring data integrity.

There are two packages in the application: main and design. The DesignClass is located in the “org.design” package, while the remaining classes, including GUI_Class, Polynomial, and OperationClass, are situated within the “org.main” package.



Interactions

- GUI_Class has an association with Polynomial, interacting with Polynomial objects (2 polynomial objects need to be created before performing operations)
- Since it operates on polynomial objects by offering a result, OperationClass has an association with Polynomial.
- DesignClass has no associations with other classes as it provides utility methods for GUI customization.



Data structure used in this application:

- HashMap - It maps the degree of each term to its corresponding coefficient when storing the polynomial terms. It is declared with the following signature: `HashMap<Integer, Double>`.

Other classes or tools used for the project:

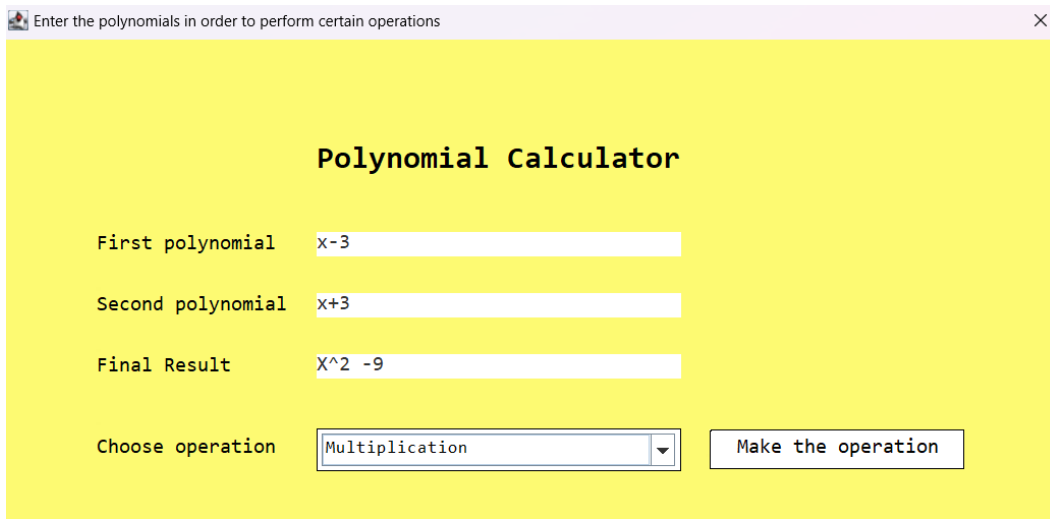
- StringBuilder – Its main purpose is to build strings efficiently, used when constructing the final polynomial result.
- RegEx – Regular Expressions were utilized in the Polynomial-Class to parse the polynomial string and extract its constituent terms for further processing.

IMPLEMENTATION

GUI_Class
<ul style="list-style-type: none">- mainPanel: JPanel- firstPolynomialText: JTextArea- operationChooser: JComboBox- makeTheOperationButton: JButton- secondPolynomialText: JTextArea- finalText: JTextArea- secondText: JLabel
<ul style="list-style-type: none">+ GUI_Class()+ main(String[]): void

GUI_Class represents the graphical user interface of the application. Its main objective is to construct and display a dialog window for the user interface. GUI_Class serves as the intermediary for user interactions with the application's interface, preparing the inputs to be stored in HashMaps. The interface includes two fields for entering polynomial expressions, one JComboBox which lets the user choose the desired operation and a button to display the final polynomial.

The UI of the project is simple and effectively allows users to calculate any polynomial with natural degrees and integer coefficients. In case of undesired inputs, a message will be displayed instead of a polynomial expression.



Enter the polynomials in order to perform certain operations

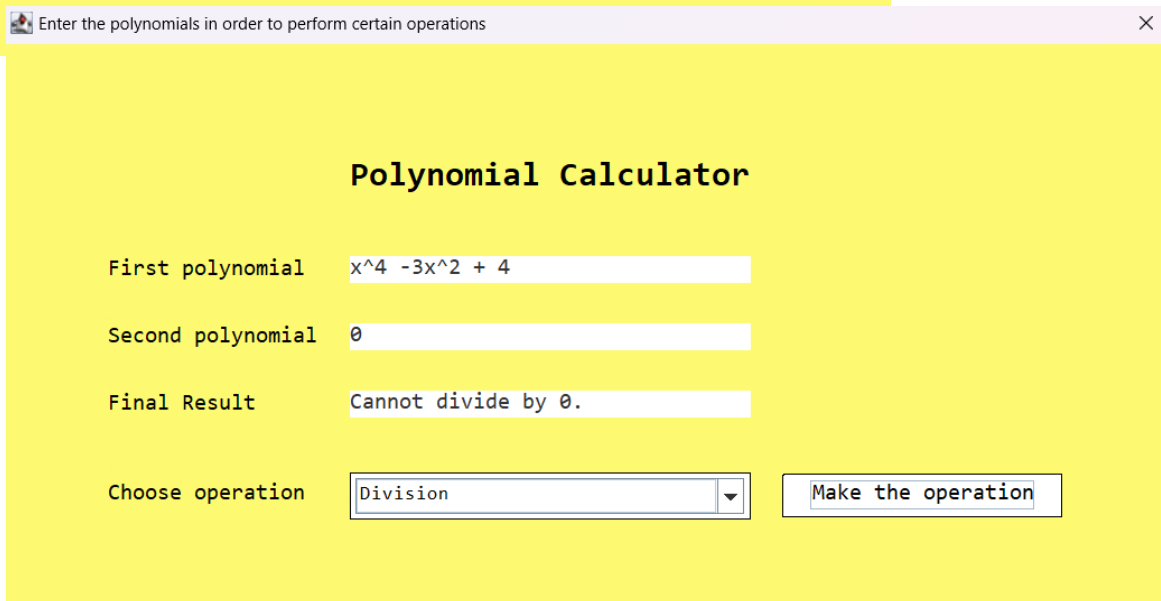
Polynomial Calculator

First polynomial

Second polynomial

Final Result

Choose operation



Enter the polynomials in order to perform certain operations

Polynomial Calculator

First polynomial

Second polynomial

Final Result

Choose operation

The class Polynomial is using the concept of encapsulation, maintaining data integrity, and hiding the important details of an object. It is vital for the project because it is used to read polynomial strings and store degrees and coefficients in corresponding hash maps which will be useful in further calculations, when using the OperationClass static methods.

The most significant method from this class is the storeInHashMaps method:

Polynomial
- poly1: String - coefficientMap1: HashMap
+ getPolynomialString(): String + getCoefficientMap(): HashMap + setCoefficientMap(HashMap): void + Polynomial(String) - addCoefficient(int, double, HashMap): void - storeInHashMaps(String, HashMap): void + readPolynomial(): void

```
private void storeInHashMaps(String poly, HashMap<Integer, Double> coefficientMap)
```

```
{
    poly = poly.replaceAll("\\s", ""); //remove all spaces
    Pattern pattern = Pattern.compile("[+-]?[^-+]+");
    Matcher matcher = pattern.matcher(poly);
    while (matcher.find()) {
        String term = matcher.group(1);
        if (!term.isEmpty()) {
            double coefficient;
            int degree = 0;
            if (term.contains("X") || term.contains("x")) {
                String[] parts = term.split("[xX]");
                if (parts.length > 0 && !parts[0].isEmpty())
                {
                    if(!parts[0].equals("+") && !parts[0].equals("-"))
                        coefficient = Double.parseDouble(parts[0]);
                    else if(parts[0].equals("+"))
                        coefficient = 1;
                    else
                        coefficient = -1;
                }
            }
            else
                coefficient = 1; //in case x is removed and there is nothing left
            if (parts.length > 1 && parts[1].contains("^")) {
```

```

        degree = Integer.parseInt(parts[1].substring(parts[1].indexOf("^") + 1));
    } else
        degree = 1;
    } else
        coefficient = Double.parseDouble(term);
    addCoefficient(degree, coefficient, coefficientMap);
}}}

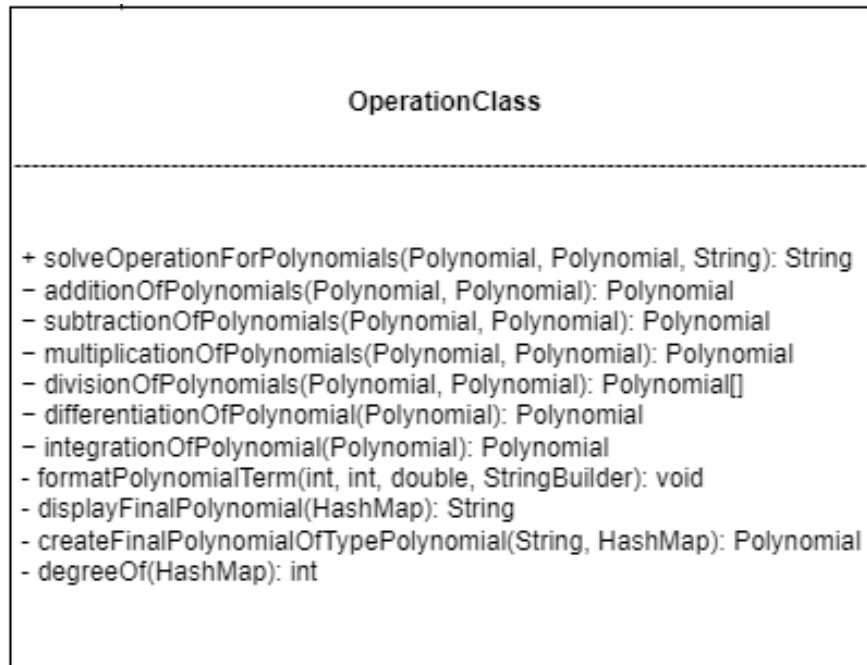
```

This code is used to parse the degrees and the coefficients of the polynomial expression into maps which are to be used later by calling static methods from OperationClass. This method uses regular expressions and pattern matching. Firstly, all the spaces from the string are removed. Then, we use the pattern “([+-]?[^-+]+)” to match individual polynomial terms, where “[+-]?” matches an optional sign character at the beginning of the term, and “[^-+]” matches one or more characters that are not '-' or '+', representing the coefficient and the possible representation of the degree (“^number”). For each matched term, the coefficient is extracted from the first part, and the degree from the second part, after a split is performed. Finally, the degrees and their corresponding coefficients are stored in the desired data structure.

DesignClass is used as the customizer for some objects from the GUI_Class. It provides public methods to customize the visual appearance of the Swing components. It has methods for modifying the aspect of buttons and combo-boxes. In addition, it helps removing partially the second text field from the graphical interface when the operation is set to “differentiation” or “integration”, since this operation can be applied to only one polynomial expression.

DesignClass
+ customizeButton(JButton): void + displaySecondPolynomial(JComboBox, JTextArea, JLabel): void + customizeChooser(JComboBox): void

The densest class from the project, “OperationClass”, contains the operation methods for addition, subtraction, multiplication, division, differentiation, and integration. All these methods are kept private. Since we are interested only in the result, the method solveOperationForPolynomials is the only public method from this class.



There are 3 helper methods: `formatPolynomialTerm`, `displayFinalPolynomial`, `createFinalPolynomialOfTypePolynomial`. All of them come in aid when calling the `solveOperationForPolynomials` method.

```

public static String solveOperationForPolynomials(Polynomial poly1, Polynomial poly2, String operation) {
    Polynomial finalPolynomial = null;
    Polynomial[] polynomialsDivision = new Polynomial[3];
    poly1.readPolynomial();
    poly2.readPolynomial();
    switch (operation) {
        case "Addition" -> finalPolynomial = additionOfPolynomials(poly1, poly2);
        case "Subtraction" -> finalPolynomial = subtractionOfPolynomials(poly1, poly2);
        case "Multiplication" -> finalPolynomial = multiplicationOfPolynomials(poly1, poly2);
    }
}
  
```

```

        case "Division" -> polynomialsDivision = divisionOfPolynomials(poly1, poly2);
        case "Differentiation" -> finalPolynomial = differentiationOfPolynomial(poly1);
        case "Integration" -> finalPolynomial = integrationOfPolynomial(poly1);
        default -> {System.out.println("This operation was not expected."); return "This operation was not expected";}
    }

    if (operation.equals("Division")) {
        if (polynomialsDivision[0] == null && polynomialsDivision[1] == null)
            return "Cannot divide by 0.";
        else if (polynomialsDivision[1] == null)
            return polynomialsDivision[0].getPolynomialString();
        else if (polynomialsDivision[0] != null)
            return polynomialsDivision[0].getPolynomialString() + " + (" + polynomialsDivision[1].getPolynomialString()
+ ")/(" + polynomialsDivision[2].getPolynomialString() + ")";
        } else if (operation.equals("Integration"))
            return finalPolynomial.getPolynomialString() + " + C";

    assert finalPolynomial != null;
    return finalPolynomial.getPolynomialString();
}

```

This method takes two Polynomial objects and a string “operation” which specifies the desired operation to execute. It initializes finalPolynomial and polynomialDivision variables. As its name suggests, polynomialDivision will store the three polynomials (when the operation string is equal to “Division”): quotient, remainder and divisor. If the result does not need a remainder, the last 2 array values will be set to null.

To perform and store the desired string, a switch statement is used regarding the operation performed. In case no standard operation is made, an error message (“This operation was not expected”) is displayed. If the operation is integration, it appends the constant of integration 'C' to the result. Ultimately, it returns the string representation of the final polynomial result.

RESULTS

By making sure the polynomial calculator performs correctly all the operations, it is important to consider some examples and expect some predefined results. In order to test the reading (storing into hash maps) and the operation methods, it is compulsory to create two separate classes in a separate test folder and make sure the results are the same.

For this section, a relevant code would be the addition function:

```
@org.junit.jupiter.api.Test
```

```
void additionOfPolynomials() {  
    Polynomial poly1 = new Polynomial("3x^2 + 2x + 5");  
    Polynomial poly2 = new Polynomial("4x^2 + 3");  
    Polynomial poly3 = new Polynomial("2x^3 + 4x^2 + x");  
    Polynomial poly4 = new Polynomial("3x^2 - 2x + 1");  
    Polynomial poly5 = new Polynomial("-X^2 - 3x + 76");  
    Polynomial poly6 = new Polynomial("-41 + 53x");  
    callRead(new Polynomial[]{poly1, poly2, poly3, poly4, poly5, poly6});  
    assertEquals("7X^2 + 2X + 8", OperationClass.solveOperationForPolynomials(poly1, poly2, "Addition"));  
    assertEquals("2X^3 + 7X^2 - X + 1", OperationClass.solveOperationForPolynomials(poly3, poly4, "Addition"));  
    assertEquals("-X^2 + 50X + 35", OperationClass.solveOperationForPolynomials(poly5, poly6, "Addition"));  
}
```

Using assertEquals, we are testing individual results and checking various cases in which the operation could perform incorrectly. Taking this code as an example, we perform similarly to the other operation methods.

Similarly, we are testing the read-method from the class Polynomial. In other words, we verify the correctness of storing degrees and coefficients in hash maps. The following structure is representative:

```
Polynomial poly1 = new Polynomial("3x^2 + 2x + 5");
poly1.readPolynomial();

HashMap<Integer, Double> expectedMap1_first = new HashMap<>();
    expectedMap1_first.put(2, 3.00);
    expectedMap1_first.put(1, 2.00);
    expectedMap1_first.put(0, 5.00);

assertEquals(expectedMap1_first, poly1.getCoefficientMap());
```

The main goal is to have all the cases tested and proved to be correct. Consequently, we can draw the conclusion of having a correct algorithm and that the application works as expected.

CONCLUSION

In conclusion, the polynomial calculator project provides a user-friendly tool for performing various polynomial operations. Through encapsulation and modular design, the project ensures clean code organization and ease of maintenance.

This project helped in understanding the clean organization of the code, separation of classes and packages by also strengthening the understanding of the concepts of OOP.

As future enhancements, the application can be improved in terms of UI: more buttons can be added such that the user can also choose to click for certain variables or numbers when entering the polynomial expressions.

BIBLIOGRAPHY

Lecture materials and other resources: <https://dsrl.eu/courses/pt/>

Understanding regular expressions: <https://www.computerhope.com/jargon/r/regex.htm>

Understanding flowchart symbols: <https://www.lucidchart.com/pages/flowchart-symbols-meaning-explained>