

DOCUMENTATION

ASSIGNMENT ASSIGNMENT_1

STUDENT NAME: NECHITA FLORINA-ELENA
GROUP: 30424

CONTENTS

1. Assignment Objective	3
2. Problem Analysis, Modeling, Scenarios, Use Cases	4
3. Design.....	6
4. Implementation	7
5. Results.....	12
6. Conclusions	13
7. Bibliography	13

1. Assignment Objective

Design and build a polynomial calculator with an easy-to-use graphical interface. The interface should enable the user to enter polynomials, choose from a variety of mathematical operations (such as addition, subtraction, multiplication, division, derivative, and integration), and clearly display the resulting output.

(i) Main objective of the assignment: Develop a polynomial calculator with a graphical user interface using Java Swing or JavaFX that can perform mathematical operations on polynomials, including addition, subtraction, multiplication, division, derivative, and integration.

(ii) Sub-objectives:

- ❖ Implement polynomial modeling with Map
 - Model a polynomial using a Map with the degree of each term as the key and the coefficient of the term as the value
 - Addressed for Modeling the Polynomial in Polinom class in package model and in Operations class in package logic
- ❖ Use foreach instead of for loop
 - Iterate over collections using foreach loop
 - Addressed for modeling the Polynomial in Polinom class in package model and in Operations class in package logic
- ❖ Implement graphical user interface
 - Develop a user-friendly GUI using Java Swing
 - Addressed for GUI Development in View class in package gui
- ❖ Implement addition and subtraction operations
 - Add and subtract polynomials
 - Addressed in Operations class in package logic
- ❖ Implement classes and methods within specified limits
 - Restrict class and method size to maximum 300 and 30 lines, respectively
- ❖ Implement multiplication operation
 - Multiply polynomials
 - Addressed in Operations class in package logic
- ❖ Implement division operation
 - Divide polynomials
 - NOT addressed
- ❖ Implement derivative operation
 - Calculate the derivative of a polynomial
 - Addressed in Operations class in package logic
- ❖ Implement integration operation
 - Calculate the integral of a polynomial
 - Addressed in Operations class in package logic
- ❖ Use regular expressions and pattern matching
 - Extract polynomial coefficients using regular expressions and pattern matching, for Coefficient Extraction
 - NOT addressed
- ❖ Use Junit for testing
 - Implement Junit testing framework to test the polynomial calculator
 - Addressed for Testing and Debugging in App.test in package test using Maven

2. Problem Analysis, Modeling, Scenarios, Use Cases

Main Objective: Develop a polynomial calculator with the ability to input polynomials, perform various mathematical operations (addition, subtraction, multiplication, division, derivative, and integration), and show the results.

Sub-Objectives:

1. Implement a user interface that allows users to input polynomials.
2. Provide options for selecting mathematical operations, including addition, subtraction, multiplication, division, derivative, and integration.
3. Calculate the result of the selected mathematical operation and display it to the user.

Analysis

In mathematics, a polynomial is a type of expression that consists of variables, constants, and mathematical operations such as addition, multiplication, and raising variables to non-negative integer powers.

The general formula of a polynomial is:

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

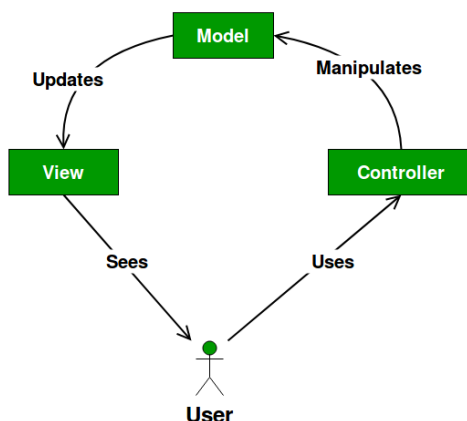
where x is the variable, n is a non-negative integer, and $a_n, a_{n-1}, \dots, a_2, a_1$, and a_0 are coefficients (constants) that determine the shape of the polynomial. The highest power of x that appears in the polynomial is called the degree of the polynomial.

A polynomial expression with one variable can be represented as a vector with p elements, where each element of the vector corresponds to the coefficient of the variable raised to a power equal to the index of the element in the vector. As an example, consider the vector $[2, -5, 0, 1]$, where $p = 4$.

This polynomial can be written as $2x^3 - 5x^2 + x$.

Instead of using a vector, a polynomial with one variable can be represented as a map where each key-value pair represents a term in the polynomial. The key corresponds to the power of the variable, while the value corresponds to the coefficient of the term. For instance, the polynomial expression $3x^4 - 2x^2 + 5x - 1$ can be represented as the following map: $\{4: 3, 2: -2, 1: 5, 0: -1\}$.

Scenario

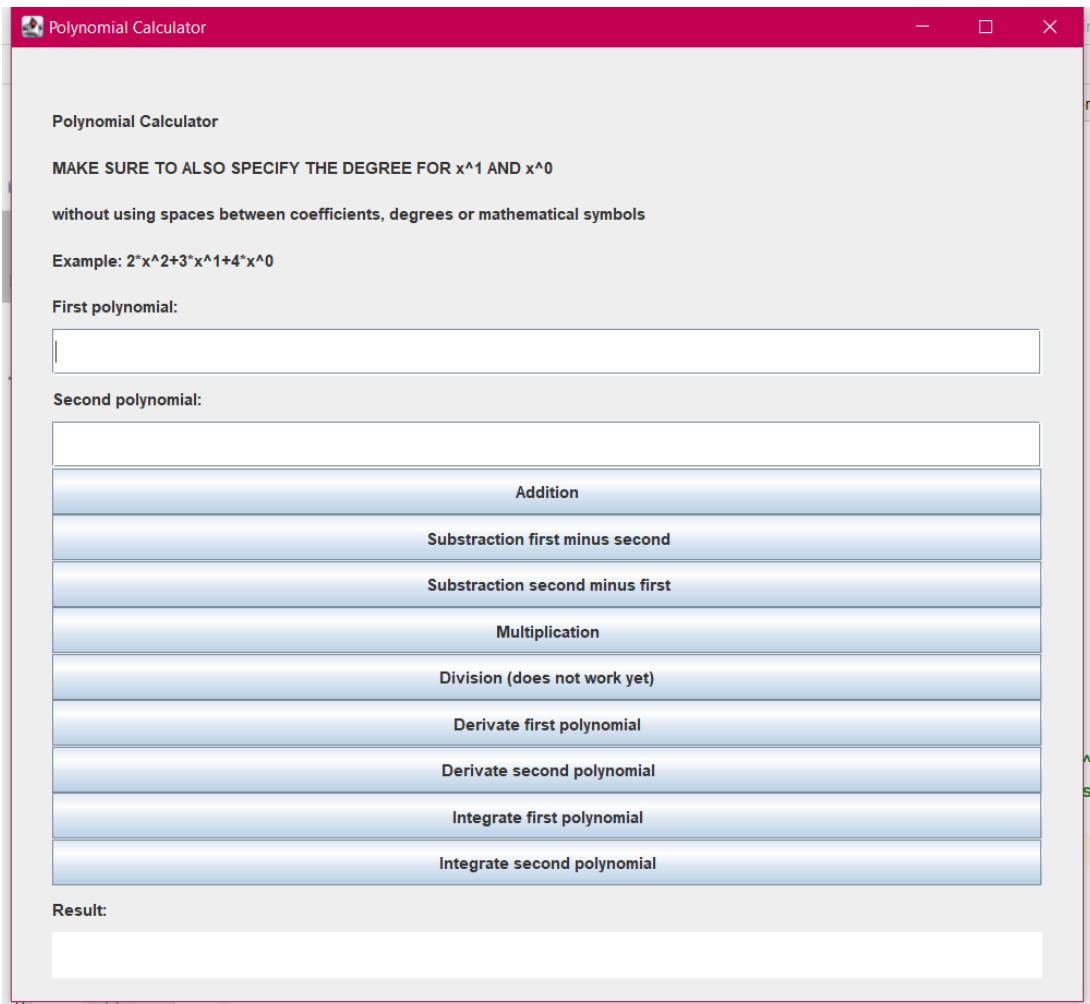


To perform operations on polynomials using the user interface, the user needs to input two polynomials.

Upon formatting the input of two polynomials as required, the user can utilize the graphical user interface to select a specific mathematical operation by clicking on the corresponding button. The calculated result will be displayed in the same window, mimicking a standard calculator.

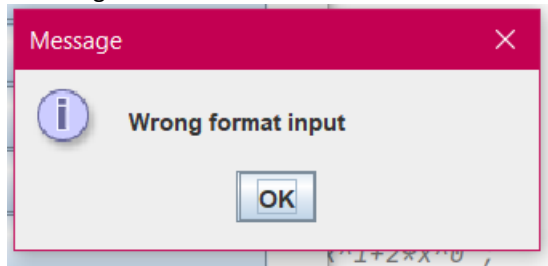
The project follows a classic MVC architecture with the gui package as the view and logic as the controller. The model manages data and logic, the view displays the model in the UI, and the controller converts user input into commands for the model. This creates a circular flow between the packages for efficient code re-use and parallel development.

The user is presented with a user-friendly and easy-to-navigate GUI (Graphical User Interface) upon starting the application. The following window is displayed:



The screenshot shows a window titled "Polynomial Calculator" with a magenta title bar. Inside, the text "Polynomial Calculator" is displayed. Below it, instructions read: "MAKE SURE TO ALSO SPECIFY THE DEGREE FOR x^1 AND x^0" and "without using spaces between coefficients, degrees or mathematical symbols". An example is given: "Example: 2*x^2+3*x^1+4*x^0". There are two input fields labeled "First polynomial:" and "Second polynomial:". Below these is a vertical stack of ten buttons: "Addition", "Subtraction first minus second", "Subtraction second minus first", "Multiplication", "Division (does not work yet)", "Derivate first polynomial", "Derivate second polynomial", "Integrate first polynomial", and "Integrate second polynomial". At the bottom, there is a "Result:" label and a large empty text box for the output.

As previously stated, the user enters two polynomials for performing the operations, and the result is displayed in the designated area for the result. If the user enters an invalid input, a warning window may appear:



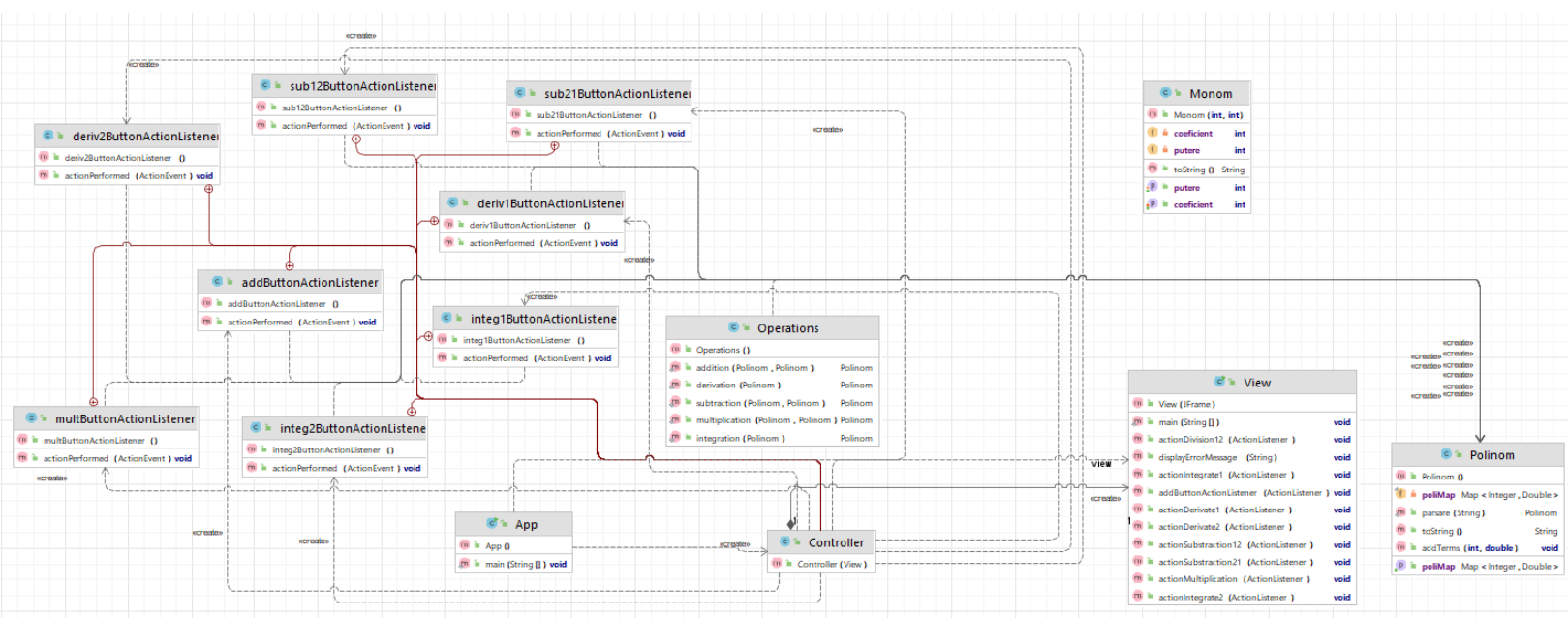
The polynomial resulting from the operation will be displayed in descending order of powers.

A use case is a technique utilized in system analysis to identify, clarify, and arrange system requirements by defining a sequence of interactions between users and systems in a specific environment, linked to a particular goal. Use cases are closely linked with user actions.

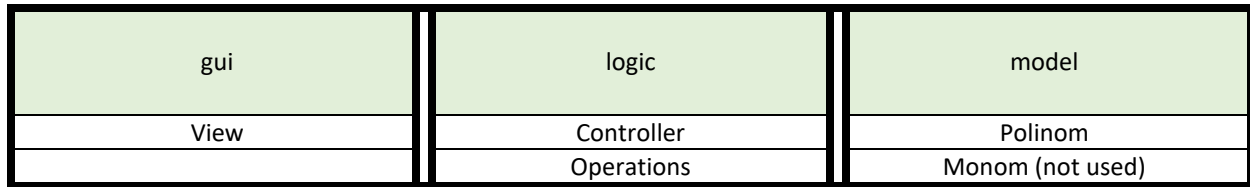
The resulting polynomial from subtraction can be obtained by subtracting one polynomial from the other, with both options of subtracting the first from the second and vice versa being available. Both polynomials can undergo the process of derivation or integration.

The following should be presented: OOP design of the application, UML package and class diagrams, used data structures, defined interfaces and used algorithms (if it is the case).

The UML (Unified Modeling Language) diagram presented below is a visual representation used to specify, document, and illustrate the software system. It includes all the Java classes utilized in the implementation of the project and their dependencies and interrelationships with one another. This diagram is known as the class diagram.



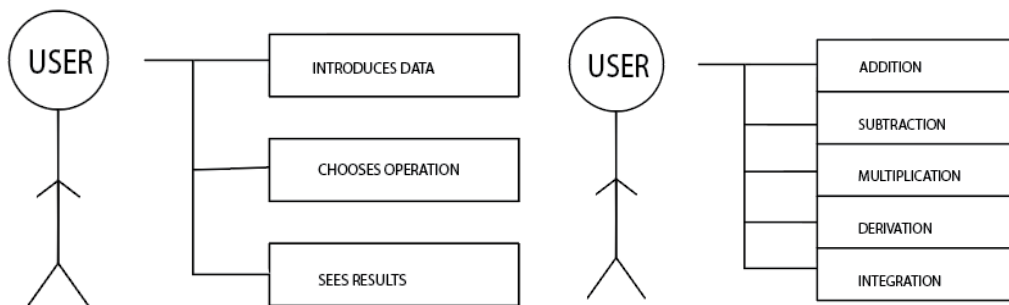
Package Diagram



The following diagram depicts the three primary packages and their corresponding classes. The gui package includes all the GUI components that the user interacts with. The model package comprises the Polinom class, and the logic package consists of the Controller class, which functions as the application's "brain," and the Operation class.

Use Case Diagram

The diagrams below illustrate the different scenarios or cases in which a user may interact with the system. These diagrams are known as use case diagrams.



4. Implementation

Each class will be described (fields, important methods). Also, the implementation of the graphical user interface will be described.

Packages

gui

The gui package comprises all the graphical user interface components employed to develop the visible user interface of the software system, which includes text fields, buttons, and labels, in the class View.

View		
View(JFrame)		
rules3	JLabel	
divisionButton	JButton	
firstPoly	JLabel	
resultLabel	JLabel	
poly1	JTextField	
integrate2Button	JButton	
subtraction12Button	JButton	
frame	JFrame	
rules	JLabel	
secondPoly	JLabel	
title	JLabel	
integrate1Button	JButton	
subtraction21Button	JButton	
result	JTextArea	
rules2	JLabel	
derivate1Button	JButton	
additionButton	JButton	
multiplicationButton	JButton	
derivate2Button	JButton	
poly2	JTextField	
panel	JPanel	
main(String[])	void	
actionDivision12 (ActionListener)	void	
displayErrorMessage (String)	void	
actionIntegrate1 (ActionListener)	void	
addButtonActionListener (ActionListener)	void	
actionDerivate1 (ActionListener)	void	
actionDerivate2 (ActionListener)	void	
actionSubtraction12 (ActionListener)	void	
actionSubtraction21 (ActionListener)	void	
actionMultiplication (ActionListener)	void	
actionIntegrate2 (ActionListener)	void	

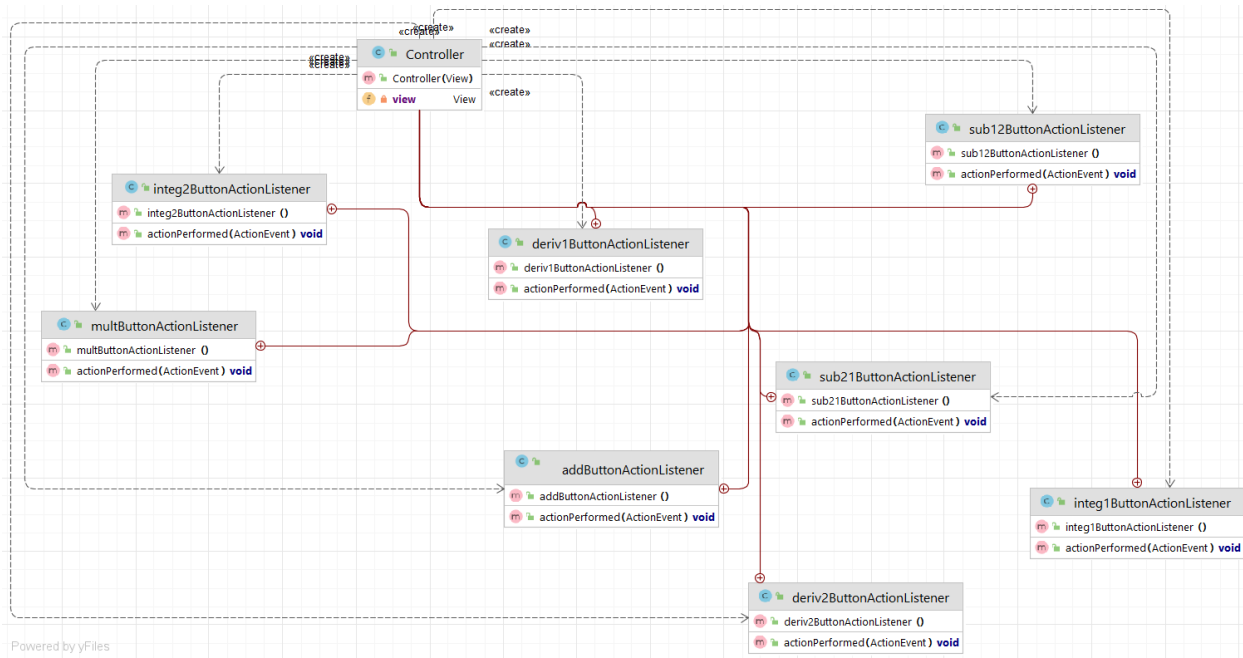
logic

Within the "logic" package, there are two classes: "Operations" and "Controller". The "Controller" class acts as an intermediary between the "Operations" class and the "View" class, facilitating communication and interaction between the backend and frontend of the application.

In the context of this application, the Operations class is responsible for performing mathematical operations on the Polinom objects. It contains methods for addition, subtraction, multiplication, division, integration, and differentiation. The Controller class, on the other hand, acts as an intermediary between the View and Operations classes.

When a user clicks on a button in the GUI, the corresponding event is captured by the View class. The View class then invokes a method in the Controller class, passing any required data as parameters. The Controller class then calls the appropriate method in the Operations class, performs the required operation on the Polinom objects, and returns the result back to the View class. Finally, the View class updates the relevant UI elements with the result.

Thus, the Controller class is responsible for linking the Operations class with the View class, and ensures that the appropriate methods are called at the right times to perform the desired mathematical operations and update the GUI accordingly.



The Controller class acts as an intermediary between the user interface (View) and the underlying business logic (Operations), handling user inputs and passing them to the appropriate operations while updating the View with the results. It is responsible for coordinating the flow of data between the user interface and the business logic.

This class implements the ActionListener interface and contains the actionPerformed method, which is executed when the "Addition" button is clicked on the user interface. It creates two new Polinom objects based on the user input in the text fields, catches any exceptions that may occur during the parsing process, and displays an error message if necessary. It then calls the addition method from the Operations class, passes in the two Polinom objects, and sets the resulting Polinom as the text of the result field on the user interface. Essentially, this class is responsible for handling the addition operation when the user clicks the "Addition" button.

```
1 usage  ▴ florina2002
public class addButtonActionListener implements ActionListener {
    ▴ florina2002
    public void actionPerformed(ActionEvent e) {
        Polinom firstPoli = new Polinom();
        Polinom secondPoli = new Polinom();

        try {
            firstPoli = Polinom.parsare(view.poly1.getText());
            secondPoli = Polinom.parsare(view.poly2.getText());

        } catch (Exception ex) {
            System.out.println(ex);

            view.displayErrorMessage(s "Wrong format input");
        }

        view.result.setText((Operations.addition(firstPoli, secondPoli)).toString());
    }
}
```

Operations		
m	Operations ()	
m	addition (Polinom, Polinom)	Polinom
m	derivation (Polinom)	Polinom
m	subtraction (Polinom, Polinom)	Polinom
m	multiplication (Polinom, Polinom)	Polinom
m	integration (Polinom)	Polinom

The Operations class contains static methods for performing various mathematical operations on Polinom objects. These operations include addition, subtraction, multiplication, integration, and derivation. Each method takes two Polinom objects as parameters and returns a new Polinom object as the result of the operation.

2 usages florina2002

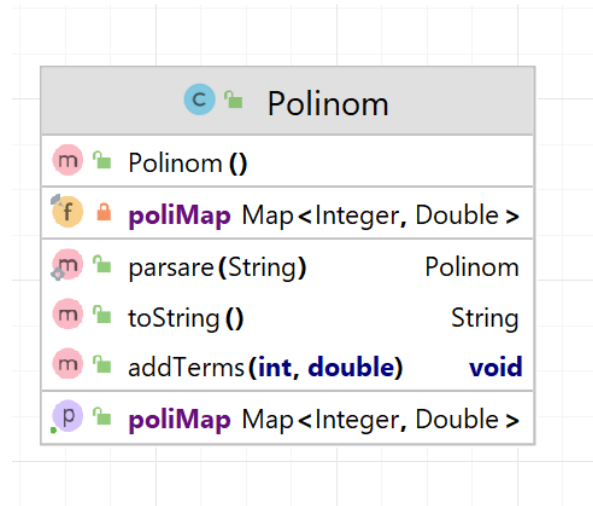
```
public static Polinom addition(Polinom polinom1, Polinom polinom2){
    Polinom polinom3 = new Polinom();
    for (Map.Entry<Integer, Double> entry : polinom1.getPoliMap().entrySet()){
        int degree = entry.getKey();
        double coefficient = entry.getValue();
        polinom3.addTerms(degree, coefficient);
    }
    for (Map.Entry<Integer, Double> entry : polinom2.getPoliMap().entrySet()){
        int degree = entry.getKey();
        double coefficient = entry.getValue();
        polinom3.addTerms(degree, coefficient);
    }
    return polinom3;
}
```

The addition method adds the coefficients of like terms in two polynomials and returns a new polynomial. The subtraction method subtracts the coefficients of like terms in two polynomials and returns a new polynomial. The multiplication method multiplies each term of one polynomial with each term of the other polynomial and returns a new polynomial. The integration method computes the antiderivative of a polynomial and returns a new polynomial. The derivation method computes the derivative of a polynomial and returns a new polynomial.

The class does not contain any instance variables and all of its methods are static, so it is not necessary to create an instance of the Operations class to use its methods.

model

The model package likely contains classes that define the domain objects of the application, which represent the data being processed or manipulated. In the context of the code provided, it contains the Polinom class, which represents a polynomial with its degree and coefficients stored in a `Map<Integer, Double>`. The Polinom class also has methods for adding, getting and removing terms.



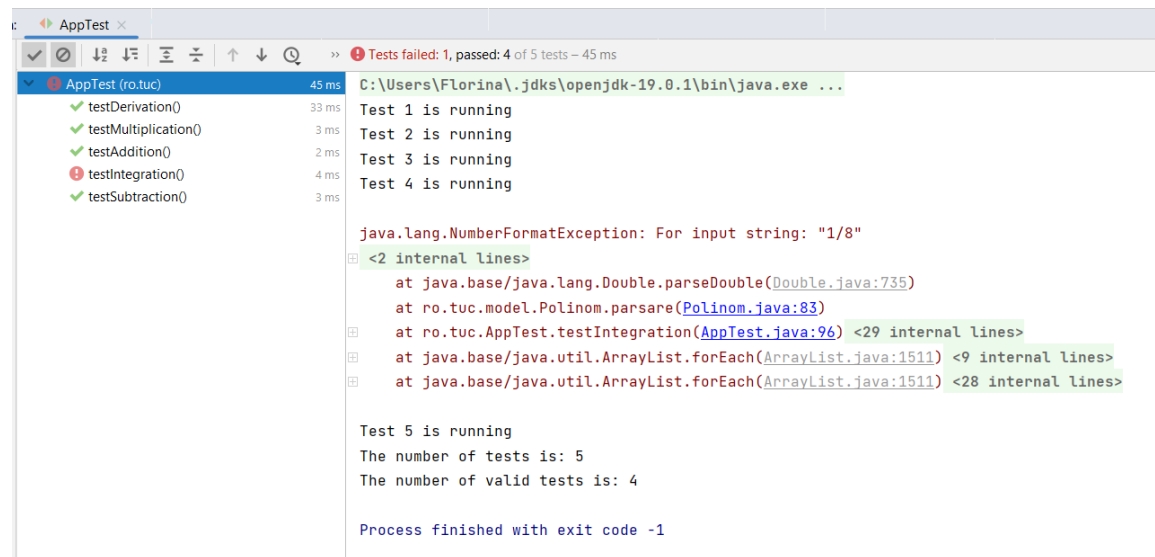
This is a Java class called "Polinom" that represents a polynomial function. Here's a brief explanation of each method in the class:

- `getPoliMap()`: This method returns the map that holds the degree-coefficient pairs of the polynomial.
- `Polinom()`: This is the constructor of the class that initializes an empty `TreeMap` to hold the degree-coefficient pairs of the polynomial. The `TreeMap` is created with a reverse order comparator, so that the degree values are sorted in decreasing order.
- `addTerms(int degree, double coefficient)`: This method adds a new term to the polynomial with the given degree and coefficient. If the coefficient is 0, the term is removed from the polynomial.
- `toString()`: This method overrides the default `toString()` method to return a string representation of the polynomial. The string contains the terms of the polynomial in descending order of degree, with each term separated by a plus or minus sign.
- `parsare(String poli)`: This static method takes a string representation of a polynomial as input and returns a `Polinom` object with the degree-coefficient pairs of the polynomial. It parses the input string using regular expressions to extract the degree and coefficient of each term, and then adds the terms to the `Polinom` object using the `addTerms()` method.

5. Results

JUnit tests are automated tests written in Java using the JUnit testing framework. They are designed to test individual units or components of code, such as methods or classes, in isolation from the rest of the code.

In the context of this project, JUnit tests can be used to test the functionality and correctness of specific methods or classes in the code. For example, a JUnit test could be written to test the addition operation of the Operation class, ensuring that it correctly adds two polynomials together and returns the expected result.



```
AppTest (ro.tuc) 45 ms
  ✓ testDerivation() 33 ms
  ✓ testMultiplication() 3 ms
  ✓ testAddition() 2 ms
  ✗ testIntegration() 4 ms
  ✓ testSubtraction() 3 ms

C:\Users\Florina\.jdk\openjdk-19.0.1\bin\java.exe ...
Test 1 is running
Test 2 is running
Test 3 is running
Test 4 is running

java.lang.NumberFormatException: For input string: "1/8"
    at java.base/java.lang.Double.parseDouble(Double.java:735)
    at ro.tuc.model.Polinom.parsare(Polinom.java:83)
    at ro.tuc.AppTest.testIntegration(AppTest.java:96)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)

Test 5 is running
The number of tests is: 5
The number of valid tests is: 4

Process finished with exit code -1
```

```
79
80 @Test
81 public void testDerivation() {
82     String poly1 = "5*x^7+6*x^4+7*x^1+8*x^0";
83     String result = "35.0*x^6+24.0*x^3+7.0*x^0";
84     p1 = Polinom.parsare(poly1);
85     rezultat1 = Polinom.parsare(result);
86     assertEquals(rezultat1.toString(), Operations.derivation(p1).toString());
87     testeValide++;
88 }
89
90 @Test
91 public void testIntegration() {
92     String poly1 = "5*x^7+6*x^4+7*x^1+8*x^0";
93     String result = "1/8*x^8+1/5*x^5+7/2*x^2+8*x^1"; //we use the fraction form to show that the correct result for coefficients is a double
94     p1 = Polinom.parsare(poly1);
95     rezultat1 = Polinom.parsare(result);
96     assertEquals(rezultat1.toString(), Operations.integration(p1).toString());
97     testeValide++;
98 }
99 }
```

The above code snippets show two JUnit tests for the derivation and integration operations of the application. The first test checks if the derivation operation on a given polynomial is correctly computed and produces the expected result. The polynomial and its expected derivative are represented as strings and parsed into Polinom objects. The test passes if the computed derivative polynomial matches the expected result. The second test checks the integration operation in a similar manner, by comparing the computed integral of a polynomial with the expected result. The coefficients in the expected result are represented as fractions to show that the correct result for

coefficients is a double. If the computed integral polynomial matches the expected result, the test is considered valid.

6. Conclusions

Through this project, I have learned the importance of proper software design and implementation, as well as the use of various tools and technologies in Java, such as JUnit.

In terms of future developments, there is always room for improvement and expansion of the Polynomial Calculator. One possible addition could be the implementation of additional operations, such as division, to further enhance the functionality of the calculator. Additionally, the GUI could be improved with more advanced features and a more modern design.

7. Bibliography

<https://www.geeksforgeeks.org/mvc-design-pattern/>