DOCUMENTATION

ASSIGNMENT *ASSIGNMENT_NUMBER*

STUDENT NAME: TĂNASE LUISA ELENA
GROUP: 30422

CONTENTS

1. Assignment Objective

Main objective:

- The main objective is to 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.

Sub-objectives:

- Parse a string in order to extract coefficients and exponents for each monomial in a polynomial.
- Implement addition, subtraction, multiplication and division of two polynomials. Also, include differentiation and integration of a polynomial as possible operations that could be selected and computed.
- Implement a method that converts a map with exponents and their corresponding coefficients into a polynomial string to be displayed as the result.
- Design the interface of the polynomial calculator with two input fields, a box for selecting the desired operation to be performed, a "compute" button and a label to display the result or possible errors.

2. Problem Analysis, Modeling, Scenarios, Use Cases

Functional Requirements:

- The polynomial calculator should allow the user to insert 2 polynomials as strings, just like writing them an paper;
- The polynomial calculator should allow the user to select the desired mathematical operation to be performed on the input polynomials;
- After clicking the "Compute" button, the result or the possible errors should be displayed in a label;
- The polynomial calculator should add, subtract, multiply and divide 2 polynomials. Also, it could differentiate and integrate the first polynomial. In these cases, the second polynomial is not required to be inserted;

Non-Functional Requirements:

- The polynomial calculator should be intuitive and easy to use;
- The polynomial calculator should provide a quick response to use inputs;
- The polynomial calculator should be designed to handle efficiently polynomials with a large number of terms;
- It should handle edge cases gracefully (invalid inputs, division by zero, etc…);

Use Case: add polynomials
Primary Actor: user
Main Success Scenario:
1. The user inserts the 2 polynomials in the graphical user interface.
2. The user selects the “add” option from the possible operations
3. The user clicks on the “Compute” button
4. The polynomial calculator performs the addition of the two polynomials and displays the result
Alternative Sequence: Incorrect input polynomials
- The user inserts incorrect polynomials
- The polynomial calculator returns as a result the error message “invalid input”
- The scenario returns to step 1

Use Case: subtract polynomials
Primary Actor: user
Main Success Scenario:
1. The user inserts the 2 polynomials in the graphical user interface.
2. The user selects the "subtract" option from the possible operations
3. The user clicks on the "Compute" button
4. The polynomial calculator performs the subtraction of the two polynomials and displays the result
Alternative Sequence: Incorrect input polynomials
- The user inserts incorrect polynomials
- The polynomial calculator returns as a result the error message "invalid input"
- The scenario returns to step 1

Use Case: multiply polynomials
Primary Actor: user
Main Success Scenario:
1. The user inserts the 2 polynomials in the graphical user interface.
2. The user selects the "multiply" option from the possible operations
3. The user clicks on the "Compute" button
4. The polynomial calculator performs the multiplication of the two polynomials and displays the result
Alternative Sequence: Incorrect input polynomials
- The user inserts incorrect polynomials
- The polynomial calculator returns as a result the error message "invalid input"
- The scenario returns to step 1

Use Case: divide polynomials
Primary Actor: user
Main Success Scenario:
1. The user inserts the 2 polynomials in the graphical user interface.
2. The user selects the "divide" option from the possible operations
3. The user clicks on the "Compute" button
4. The polynomial calculator performs the division of the first polynomial by the second polynomial
Alternative Sequence:
1) Incorrect input polynomials
- The user inserts incorrect polynomials
- The polynomial calculator returns as a result the error message "invalid input"
- The scenario returns to step 1
2) Second polynomial equal with zero
- The polynomial calculator returns as a result the error message "Cannot divide by zero"
- The scenario returns to step 1

Use Case: differentiation of a polynomial
Primary Actor: user
Main Success Scenario:
1. The user inserts the polynomial in the first polynomial input text from the graphical user interface.
2. The user selects the “derivative” option from the possible operations
3. The user clicks on the “Compute” button
4. The polynomial calculator performs the differentiation of the polynomial and displays the result
Alternative Sequence: Incorrect input polynomial
- The user inserts an incorrect polynomial
- The polynomial calculator returns as a result the error message “invalid input”
- The scenario returns to step 1

Use Case: integration of a polynomial
Primary Actor: user
Main Success Scenario:
1. The user inserts the polynomial in the first polynomial input text from the graphical user interface.
2. The user selects the “integration” option from the possible operations
3. The user clicks on the “Compute” button
4. The polynomial calculator performs the integration of the polynomial and displays the result
Alternative Sequence: Incorrect input polynomial
- The user inserts an incorrect polynomial
- The polynomial calculator returns as a result the error message “invalid input”
- The scenario returns to step 1

3. Design

Overall System Design

Division into packages

Division into classes

UML Diagram

Used data structures:

- In the Polynomial class, we use a HashMap having as key-value the exponent-coefficient pairs (integer-double)

Used algorithms:

- We used the algorithm for dividing two polynomials

- For the rest of the operations (addition, subtraction, multiplication, differentiation and integration) we used basic mathematical rules

4. Implementation

View class

The View class extends the JFrame class, which is a top-level container for displaying a window in a Java Swing application. The View class encapsulates all the GUI components necessary for the user to interact with the polynomial calculator.

Components:

contentPanel: This panel serves as a container to hold all the GUI components such as labels, text fields, combo boxes, buttons, and result displays.

firstPolynomialTextField: A text field where the user can input the first polynomial.

firstPolynomialLabel: A label accompanying the firstPolynomialTextField to indicate its purpose.

secondPolynomialTextField: A text field where the user can input the second polynomial.

secondPolynomialLabel: A label accompanying the secondPolynomialTextField to indicate its purpose.

operationsComboBox: A combo box allowing the user to select the operation to perform on the polynomials. It includes the options: addition, subtraction, multiplication, division, derivative, and integration.

computeButton: A button that triggers the computation of the selected operation on the entered polynomials.

resultValueLabel: A label used to display the result of the computation.

Controller: An instance of the Controller class is instantiated within the View class to handle user interactions, such as button clicks.

Methods:

prepareGui(): This method initializes and configures the GUI components. It sets up the layout, adds labels, text fields, combo boxes, buttons, and result display components to the content panel.

getOperationsComboBox(): This method returns the operations combo box, allowing external classes to access it.

getResultValueLabel(): This method returns the result value label, allowing external classes to access it.

getFirstPolynomial(): This method retrieves the text entered into the first polynomial text field.

getSecondPolynomial(): This method retrieves the text entered into the second polynomial text field.

Controller class

The Controller class serves as the controller component in the Model-View-Controller (MVC) architectural pattern for the polynomial calculator application. It handles user inputs from the GUI (View) and delegates the processing to the underlying logic (Operations) before updating the View with the result.

Dependencies:

View: This class interacts with the GUI components and listens for user actions.

Operations: An instance of this class is used to perform various operations on polynomials.

Constructor: Controller(View v): Constructs a new Controller instance, initializing it with the provided View instance. It also initializes the Operations object with which it is in a composition uml relationship.

ActionListener Implementation: The Controller class implements the ActionListener interface to listen for and handle action events triggered by user interactions in the GUI (View).

actionPerformed(ActionEvent e): This method is invoked when an action event occurs, such as a button click. It performs the following steps:

1) Retrieves the action command associated with the event.
2) If the command is "COMPUTE", indicating that the user wants to perform a computation, then go to step 3:
3) Retrieves the selected operation from the View's operations combo box.
4) Parses the polynomials entered by the user from the text fields.
5) Performs the corresponding operation using the Operations object.
6) Updates the View's result label with the computed result.

If an error occurs during the computation or parsing of input, it updates the View's result label with an error message.

Operations class

The Operations class provides static methods for performing various mathematical operations on polynomials. These operations include addition, subtraction, multiplication, division, differentiation, and integration.

Dependencies:

Polynomial: This class represents a polynomial and is used as the data structure for the input and output of operations.

Methods:

isZeroPolynomial(Polynomial p): Checks if the given polynomial is a zero polynomial (checks if all coefficients are zero).

add(Polynomial p1, Polynomial p2): Adds two polynomials p1 and p2 and returns the result as a new polynomial. In order to do this, we map over the coefficients of p1 and we add them in the result polynomial. Then, we map over the coefficients of p2 and we check if we already have the

corresponding exponent in the result. If so, we add the coefficients, else we add the new coefficient for that exponent. If a coefficient becomes zero, then we remove that exponent-coefficient pair from the result.

subtract(Polynomial p1, Polynomial p2): Subtracts polynomial p2 from polynomial p1 and returns the result as a new polynomial. Similarly with the addition implementation, in order to do the subtraction, we map over the coefficients of p1 and we add them in the result polynomial. Then, we map over the coefficients of p2 and we check if we already have the corresponding exponent in the result. If so, we subtract the coefficients of p2 from the coefficient of p1, else we add the new coefficient for that exponent with the sign changed. If a coefficient becomes zero, then we remove that exponent-coefficient pair from the result.

multiply(Polynomial p1, Polynomial p2): Multiplies two polynomials p1 and p2 and returns the result as a new polynomial. In order to do this, we map over the coefficient of p1 and of p2 in a nested loop. For each term, the new exponent is the sum of the exponents from p1 and p2 and the new coefficient is the product of the coefficients from p1 and p2. If we already have such an exponent in the result, then we overwrite its coefficient with the sum of the new coefficient and the already existing one.

divide(Polynomial p1, Polynomial p2): Divides polynomial p1 by polynomial p2 using polynomial long division and returns the quotient and remainder as a formatted string. In order to do this, we used the algorithm presented in the pseudocode from section 3.

differentiation(Polynomial p): Computes the derivative of the polynomial p and returns the result as a string. For each term (monomial), the coefficient becomes the product between the exponent and the coefficient, and the exponent decreases with 1.

integrate(Polynomial p): Computes the indefinite integral of the polynomial p and returns the result as a string, including the constant of integration. For each term (monomial), the coefficient becomes the coefficient divided bt the exponent incremented by 1 and the exponent then gets incremented by 1. We concatenate “+C” to the resulted string, since the it is the result of an indefinite integral.

5. Results

The OperationsTest class contains test methods to verify the correctness of the operations implemented in the Operations class. Each test method focuses on a specific operation and provides a scenario to ensure the functionality is working as expected.

Scenarios:

addTest(): Tests the addition operation.
Input: Two polynomials p1= 3x^3+2x^2+x-5 and p2= x^2-2x+1.
Expected Output: The result of adding p1 and p2 together, which is 3x^3+3x^2-x-4.
Explanation: Parses two polynomials, adds them together using the Operations.add() method, and compares the result with the expected polynomial.

subtractTest(): Tests the subtraction operation.
Input: Two polynomials p1= 3x^3+2x^2+x-5 and p2= x^2-2x+1.
Expected Output: The result of subtracting p2 from p1, which is 3x^3+x^2+3x-6.
Explanation: Parses two polynomials, subtracts p2 from p1 using the Operations.subtract() method, and compares the result with the expected polynomial.

multiplyTest(): Tests the multiplication operation.
Input: Two polynomials p1= 3x^3+2x^2+x-5 and p2= x^2-2x+1.
Expected Output: The result of multiplying p1 and p2, which is 3x^5-4x^4-5x^2+11x-5.
Explanation: Parses two polynomials, multiplies them together using the Operations.multiply() method, and compares the result with the expected polynomial.

divideTest(): Tests the division operation.
Input: Two polynomials p1= x^3-2x^2+6x-5 (dividend) and p2= x^2-1 (divisor).
Expected Output: The quotient and remainder of dividing p1 by p2, which is Q = 1.0x^1-2.0x^0 and R = 7.0x^1-7.0x^0.
Explanation: Parses two polynomials, performs polynomial long division using the Operations.divide() method, and compares the result with the expected quotient and remainder.

differentiationTest(): Tests the differentiation operation.
Input: A polynomial p= 3x^3+2x^2+x-5.
Expected Output: The derivative of p, which is 9x^2+4x+1.
Explanation: Parses a polynomial, computes its derivative using the Operations.differentiation() method, and compares the result with the expected derivative.

integrationTest(): Tests the integration operation.
Input: A polynomial p= 4x^3+3x^2+x-5.
Expected Output: The indefinite integral of p, which is 1.0x^4+1.0x^3+0.5x^2-5.0x^1 + C.
Explanation: Parses a polynomial, computes its indefinite integral using the Operations.integrate() method, and compares the result with the expected integral.

6. Conclusions

What have I learned:

- The model-view-controller architectural pattern
- To build simple interfaces (GUI) using Java Swing
- Regex
- Testing with Junit

Future developments:

- GUI improvements
- Adding more possible operations, like finding the roots of the polynomials
- Add functionality to solve polynomial equations
- Integrate a history functionality, to store previous calculations

7. Bibliography

1. *PT Course from TUCN*
2. *Regex for polynomial expression – Stack Overflow: https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression*