DOCUMENTATION

ASSIGNMENT 1

STUDENT NAME: COMAN ALECSIA-COSTINA
GROUP: 30423

CONTENTS

1. Assignment Objective

The assignment's objective is to create and implement a system tailored for handling polynomial operations, focusing on single-variable polynomials with double coefficients.

- Design the Polynomial class
- Implement mathematical operations in the Operations Class
- Develop the Controller class for user interaction
- Desing the GUI Interface
- Integrate operations with controller
- Ensure proper flow and logic in Controller
- Test and debug the application
- Document the application

2. Problem Analysis, Modeling, Scenarios, Use Cases

- PROBLEM ANALYSYS

A polynomial is an algebraic expression that combines constants and variables using addition, multiplication and exponentiation with non-negative integer powers. In the context of a single indeterminate, denoted as x, a polynomial can always be expressed in the form:

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

Here, $a_0, a_1, a_2, \ldots, a_n$, are constants and x represents an indeterminate value. The term "indeterminate" implies that x doesn't have a specific value.

A polynomial comprises specific terms, also known as monomials. For instance, $4x^3$ is a monomial where the coefficient is 4, x is the indeterminate, and the degree is 3. Combining several terms yields a polynomial: $4x^3 - 2x + 6$ has three terms with varying exponents: the first is of degree three, the second is of degree one, and the third is of degree zero.

An alternative polynomial representation involves a sequence of ordered pairs:

$$\{(a_0, 0), (a_1, 1), (a_2, 2), \ldots., (a_n, n)\}$$

Each pair $(i,\ a_i)$ corresponds to the term $a_i x^i$ within the polynomial. The pair consists of the coefficient of the i-th term and its corresponding exponent i. For example, the polynomial $4x^3 - 2x + 6$ can be represented as {(3, 4), (1, -2), (0, 6)}, where each pair denotes the degree and coefficient of a monomial in the polynomial.

This representation facilitates various polynomial operations, including addition, subtraction, multiplication, division, differentiation, and integration, making it a versatile approach for polynomial processing.

- MODELLING

To utilize the calculator, users are required to input the two desired polynomials into designated text fields. Six operations are available:

- Addition
- Subtraction
- Multiplication
- Division
- Integration of both polynomials
- Derivation of both polynomials

The outcome of the selected operation will be showcased at the bottom of the screen. Subsequently, users can opt for another operation with the previously entered polynomials or input new ones, repeating the aforementioned procedure.

- SCENARIOS

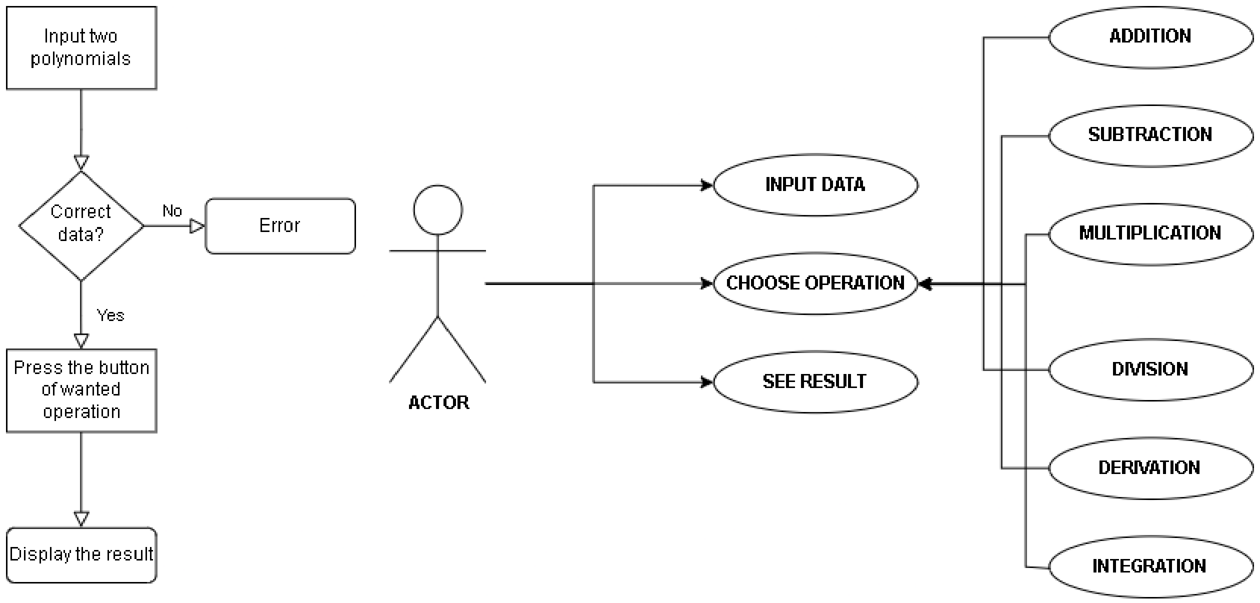
The use case methodology in system analysis involves identifying, defining and documenting the interactions between users/actors and a system, to understand its functional requirements.

To utilize the Java polynomial calculator, users need to follow a few straightforward steps. Most operations involve two polynomials, except for integration and derivation. The interface features two text fields where users input each polynomial via the keyboard. Below these fields, there's an example showing the correct format for polynomial input, such as _-_2x^3_+_1 (where

"_" indicates a blank space). Users must adhere to this format for optimal program functionality; otherwise, a pop-up window will prompt them to rewrite the polynomials correctly.

After correctly inputting both polynomials, users can select from a range of operations such as addition, subtraction, multiplication, and division by pressing the corresponding button. This action internally processes the inputted polynomials based on the chosen operation. The results are then shown at the bottom of the screen beneath the corresponding operation buttons. Additionally, for division, a label appears to display the remainder, while for integration and derivation, two text fields are utilized to display both polynomials.

- <u>USE CASE DIAGRAM</u>

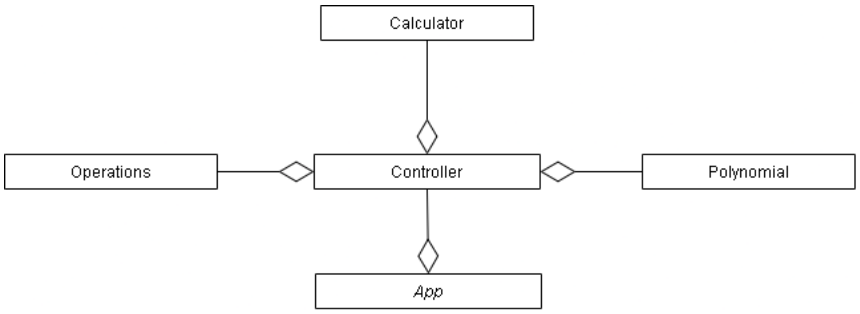


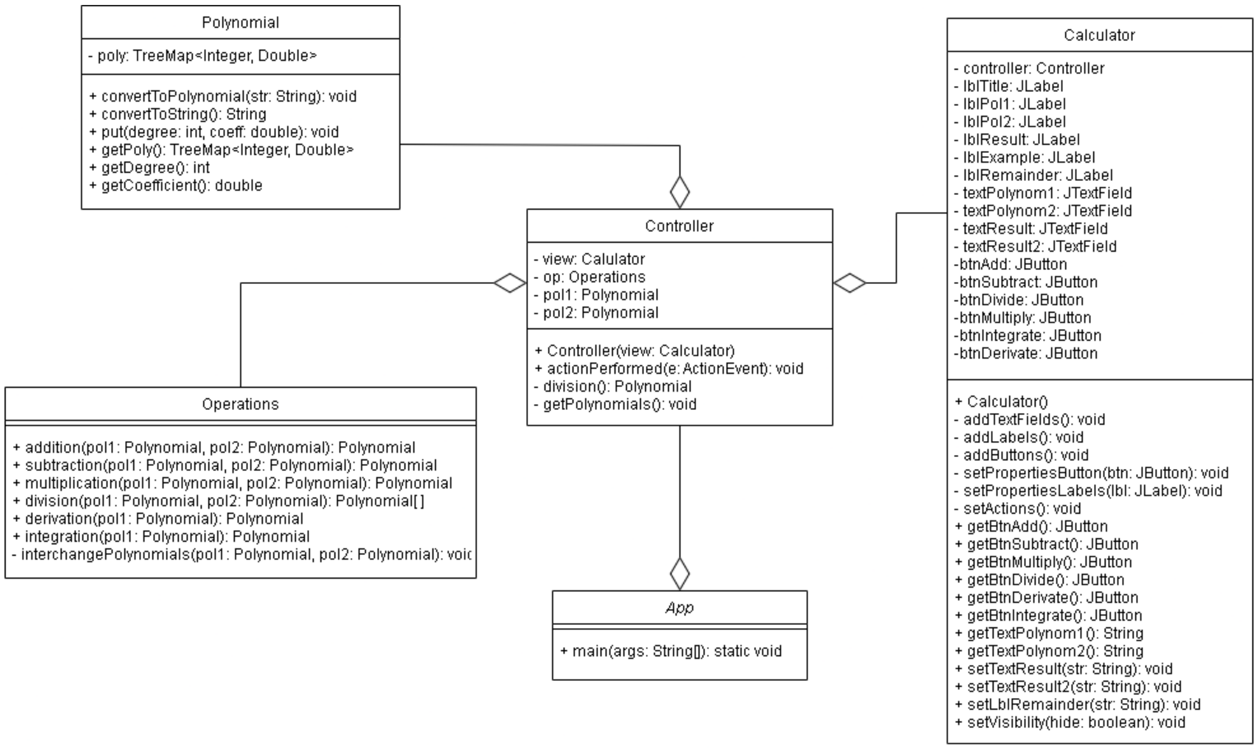
The stickman depicted in the diagram symbolizes the actor, which is the user interacting with the application. Connected to the stickman are three ovals representing its potential actions: inputting the polynomials, selecting the operation to perform with the inputted data, and viewing the resulting output.

3. Design

- <u>CLASS DIAGRAM</u>

A class diagram is a visual representation of the structure and relationships of classes in a system or software application. It shows the classes, attributes, methods, and associations between classes, providing a high-level overview of the system's architecture and design.




Polynomial
- poly: TreeMap<Integer, Double>
+ convertToPolynomial(str: String): void
+ convertToString(): String
+ put(degree: int, coeff: double): void
+ getPoly(): TreeMap<Integer, Double>
+ getDegree(): int
+ getCoefficient(): double
Controller
- view: Calulator
- op: Operations
- pol1: Polynomial
- pol2: Polynomial
+ Controller(view: Calculator)
+ actionPerformed(e: ActionEvent): void
- division(): Polynomial
- getPolynomials(): void
Operations
+ addition(pol1: Polynomial, pol2: Polynomial): Polynomial
+ subtraction(pol1: Polynomial, pol2: Polynomial): Polynomial
+ multiplication(pol1: Polynomial, pol2: Polynomial): Polynomial
+ division(pol1: Polynomial, pol2: Polynomial): Polynomial[ ]
+ derivation(pol1: Polynomial): Polynomial
+ integration(pol1: Polynomial): Polynomial
- interchangePolynomials(pol1: Polynomial, pol2: Polynomial): voic
App
+ main(args: String[]): static void
Calculator
- controller: Controller
- lblTitle: JLabel
- lblPol1: JLabel
- lblPol2: JLabel
- lblResult: JLabel
- lblExample: JLabel
- lblRemainder: JLabel
- textPolynom1: JTextField
- textPolynom2: JTextField
- textResult: JTextField
- textResult2: JTextField
-btnAdd: JButton
-btnSubtract: JButton
-btnDivide: JButton
-btnMultiply: JButton
-btnIntegrate: JButton
-btnDerivate: JButton
+ Calculator()
- addTextFields(): void
- addLabels(): void
- addButtons(): void
- setPropertiesButton(btn: JButton): void
- setPropertiesLabels(lbl: JLabel): void
- setActions(): void
+ getBtnAdd(): JButton
+ getBtnSubtract(): JButton
+ getBtnMultiply(): JButton
+ getBtnDivide(): JButton
+ getBtnDerivate(): JButton
+ getBtnIntegrate(): JButton
+ getTextPolynom1(): String
+ getTextPolynom2(): String
+ setTextResult(str: String): void
+ setTextResult2(str: String): void
+ setLblRemainder(str: String): void
+ setVisibility(hide: boolean): void

- <u>OOP DESIGN</u>

The object-oriented programming (OOP) design of the application follows key principles and concepts that are fundamental to software engineering:

- *Abstraction* is achieved through classes like Polynomial, Operations, Controller, and Calculator, which hide complex implementation details and expose only essential functionalities to the user.
- *Encapsulation* is utilized to encapsulate data (polynomial coefficients and degrees) and methods (polynomial manipulation operations) within the Polynomial class, ensuring data integrity and promoting modular code organization.
- *Inheritance* is employed in the Operations class, which inherits functionalities from the Polynomial class to perform mathematical operations such as addition, subtraction, multiplication, division, derivative, and integration on polynomials.
- *Polymorphism* is demonstrated in the Controller class, where different button actions trigger polymorphic behavior based on the type of mathematical operation selected by the user. This allows for flexibility and extensibility in handling user interactions.

The application is designed with a modular structure, where each class represents a distinct module responsible for specific functionalities. This promotes code reusability, maintainability, and scalability.

The classes exhibit high cohesion by focusing on single responsibilities (e.g., polynomial manipulation, mathematical operations, user interface handling), leading to more understandable and maintainable code.

• DATA STRUCTURES

The main data structure used is the **TreeMap<Integer, Double>** for representing the polynomials. This data structure is employed within the Polynomial class to store and manipulate polynomial expressions efficiently.

The *poly* field in Polynomial class is declared as a TreeMap<Integer, Double> to store the terms in a specific way: integer keys represent the degree of each monomial, while double values represent the coefficient associated with each term.

The TreeMap data structure offers benefits I considered advantageous for my application. This structure stores elements in a sorted order based on their keys. This makes it ideal for scenarios like sorting polynomial terms by their degrees in descending order. Also, because it uses Red-Black Tree internally, elements can be easily accessed, by key.

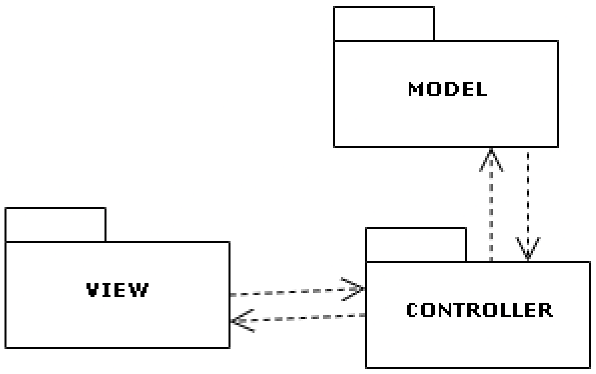
- <u>PAKAGES</u>

MVC stands for Model-View-Controller, which is a software design pattern commonly used for developing user interfaces. The methodology aims to separate the concerns of an application into three interconnected components.

The *Model* represents the data and business logic of the application. It encapsulates the data and defines how it can be accessed and manipulated. In MVC, the Model notifies the View of any changes to the data so that the View can update accordingly.

The *View* is responsible for presenting the user interface to the user. It displays the data from the Model and allows users to interact with it. In MVC, the View observes the Model and updates itself whenever the Model changes.

The *Controller* acts as an intermediary between the Model and the View. It receives user input from the View, processes it (e.g., performs data validation or business logic operations), and updates the Model accordingly. The Controller also updates the View based on changes in the Model.



My application adheres to the Model-View-Controller (MVC) architectural pattern, chosen for its ability to enhance system functionality comprehension and code readability. The Application class, housing the static main function that initiates the program, is kept distinct from the three main packages: Model, Controller, and View.

In the View package, the Calculator class is responsible for crafting the Graphical User Interface (GUI), ensuring a straightforward and user-friendly interaction experience. The interface is designed to clearly present available options to the user, facilitating easy utilization of the calculator's functionalities.

The Model package comprises the core logic of the application, containing the Polynomial class, which serves as the foundational object within the system, and the Operations class, facilitating all permissible polynomial operations accessible to the user.

Within the Controller package resides the Controller class, serving as the intermediary between the View and the Model. It captures user input from button presses, determines the subsequent action based on the input, and orchestrates the display of outcomes on the calculator's GUI, thus facilitating seamless interaction between the user, the application's logic, and the interface.

4. Implementation

- <u>POLYNOMIAL CLASS</u>

The Polynomial class is central to the polynomial calculator, organizing polynomial terms by degrees using a TreeMap for efficient manipulation. It converts string-based inputs to TreeMap format, validates correctness, and handles error messages, ensuring data integrity. It also converts polynomials back to strings, allows access to key data, and enhances the user experience with clear feedback on input errors.

- ***private TreeMap<Integer, Double> poly*:** A TreeMap that stores the polynomial terms as key-value pairs, where the key represents the degree of the term and the value represents the coefficient.
- ***convertToPolynomial(String str):*** Converts a string representation of a polynomial into the TreeMap format used by the class. Handles parsing of coefficients and degrees from the input string. It first standardizes the string format by replacing " - " with " + -" and then splits the modified string into individual monomials based on the plus sign. It then iterates through each monomial to extract its coefficient and degree, handling cases with or without "x^" (indicating variable terms) and cases with or without "x" (variable terms with implicit degrees or constant terms). The method includes error checking to ensure valid numerical values and proper polynomial formatting, providing error messages and halting execution if any issues are encountered.
- ***private void showError(String message):*** Displays an error message using a JOptionPane dialog box in case of invalid input or format errors during polynomial conversion.
- ***public String convertToString():*** Converts the polynomial stored in the TreeMap back to its string representation. Constructs the polynomial string by iterating through the TreeMap and appending coefficients and degrees as necessary.
- ***public TreeMap<Integer, Double> getPoly():*** Returns the TreeMap containing the polynomial terms.
- ***public int getDegree():*** Returns the highest degree (key) of the polynomial stored in the TreeMap.
- ***public double getCoefficient():*** Returns the coefficient corresponding to the highest degree term in the polynomial.

- ***public void put(int degree, double coeff):*** Inserts a new term (degree and coefficient pair) into the TreeMap representing the polynomial.

  - OPERATIONS CLASS

This class contains methods for performing mathematical operations on polynomials, such as addition, subtraction, multiplication, division, derivation and integration.

- ***public Polynomial addition(Polynomial pol1, Polynomial pol2):*** Performs polynomial addition by adding the coefficients having the same variable's exponent: it takes two polynomials, pol1 and pol2, and performs polynomial addition on them. It initializes a new polynomial result as a copy of pol1. Next, it iterates through the terms of pol2 using a lambda expression. For each term in pol2, it checks if the same degree term exists in pol1. If it does, it adds the coefficients together and updates result. If the resulting coefficient is zero, it removes the term from result. If the degree does not exist in pol1, it adds the term directly to result. Finally, it returns the updated result polynomial after the addition operation.
- ***public Polynomial subtraction(Polynomial pol1, Polynomial pol2):*** Performs polynomial subtraction by subtracting the coefficients having the same variable's exponent.
- ***public Polynomial multiplication(Polynomial pol1, Polynomial pol2):*** Performs polynomial multiplication by multiplying each term of one polynomial with each term of the other one; the result order is equal with the sum of the polynomials' highest orders.
- ***private void interchangePolynomials(Polynomial pol1, Polynomial pol2*)**:** Interchanges two Polynomial objects if the first polynomial's degree is less than the second polynomial's degree.
- ***public Polynomial[] division(Polynomial pol1, Polynomial pol2):*** Performs polynomial division, using the division polynomial algorithm, and returns both the quotient and remainder as a Polynomial array.
- ***public Polynomial derivative(Polynomial pol1****):* Calculates the derivative of a polynomial by multiplying the coefficient with the exponent and subtracting one from the variable's degree.
- ***public Polynomial integrate(Polynomial pol1):*** Calculates the integral of a polynomial by dividing the coefficient with the exponent incremented by one.

All methods return the result as a new Polynomial object and iterate through the polynomials with the forEach((d,c) -> {…}) lambda expression.

- CONTROLLER CLASS

The Controller class acts as the intermediary between the user interface (View) and the mathematical operations (Model) in the polynomial calculator application. It handles user actions, triggers corresponding operations, and updates the view with the results.

- ***private Calculator view:*** Stores an instance of the Calculator class from the View package for GUI interaction.
- ***private Operations op*****:** Stores an instance of the Operations class from the Model package for performing polynomial operations.
- ***private Polynomial pol1*** and ***private Polynomial pol2*****:** Store instances of Polynomial objects for storing polynomial data and performing operations.
- ***public Controller(Calculator v):*** Constructor method that initializes the Controller with a Calculator instance.
- ***public void actionPerformed(ActionEvent e):*** Implements the ActionListener interface to handle user actions and trigger corresponding operations based on the button clicked in the GUI.
- ***private Polynomial division():*** Performs polynomial division, updates the view with the remainder, and returns the quotient.
- ***private void getPolynoms():*** Retrieves polynomial inputs from the view, converts them to Polynomial objects, and stores them in pol1 and pol2.

- APP CLASS

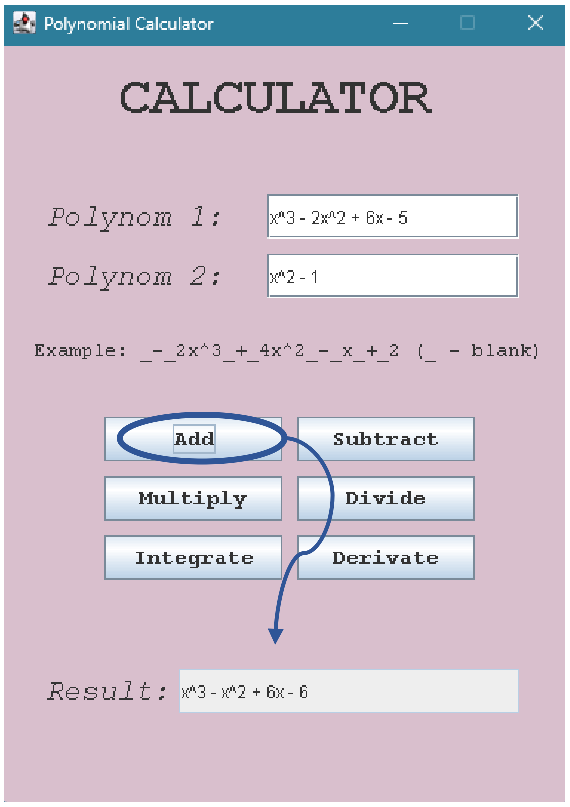
The App class serves as the entry point for the polynomial calculator application. It creates an instance of the Calculator class, sets up its appearance and properties, and makes it visible to the user.

- ***public static void main(String[] args):*** The main method of the application, responsible for launching the calculator interface; it instantiates a Calculator object, setting its size and position, along with other properties.

- CALCULATOR CLASS (GUI)

The Calculator class represents the graphical user interface (GUI) of the polynomial calculator application. It includes text fields for polynomial inputs, buttons for various mathematical operations, and labels to display results and instructions. The class interacts with the Controller class to handle user actions and update the view based on the operations performed.

- ***private Controller controller****:* Stores an instance of the Controller class to handle user actions and operations.
- ***private JLabel lblTitle, lblPol1, lblPol2, lblResult, lblExample, lblRemainder*****:** Labels for the calculator title, polynomial inputs, result display, example format, and remainder display.
- ***private JTextField textPolynom1, textPolynom2, textResult, textResult2:*** Text fields for entering polynomial inputs and displaying results.
- ***public Calculator():*** Constructor method that sets up the GUI components, layout, and actions for the calculator interface.
- ***private void addTextFields():*** Adds text fields for polynomial inputs and result display to the GUI.
- ***private void addLabels():*** Adds labels for polynomial inputs, result display, example format, and remainder display to the GUI.
- ***private void addButtons():*** Adds buttons for mathematical operations (addition, subtraction, multiplication, division, integration, derivative) to the GUI.
- ***private void setPropertiesButton(JButton btn):*** Sets properties (font, style) for the calculator buttons.
- ***private void setPropertiesLabels(JLabel lbl):*** Sets properties (font, style) for the calculator labels.
- ***private void setActions****():* Sets action listeners for the calculator buttons to trigger corresponding operations in the Controller class.
- ***Getters and setters*** methods for accessing and updating GUI components such as buttons, text fields, labels, and visibility settings (getBtnAdd(), getTextPolynom1(), setTextResult(), setLblRemainder(), etc.).

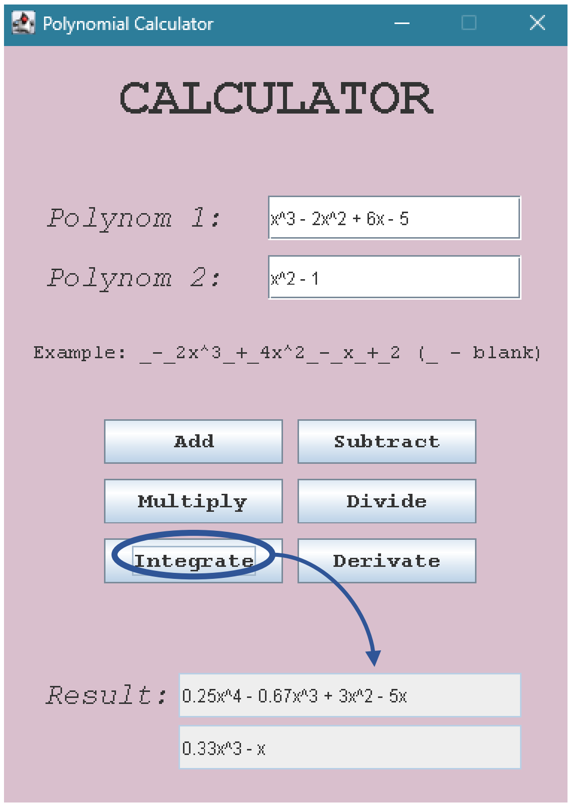


x^3 - 2x^2 + 6x - 5

x^2 - 1

Example: _-_2x^3_+_4x^2_-_x_+_2 (_ - blank)

x^3 - x^2 + 6x - 6



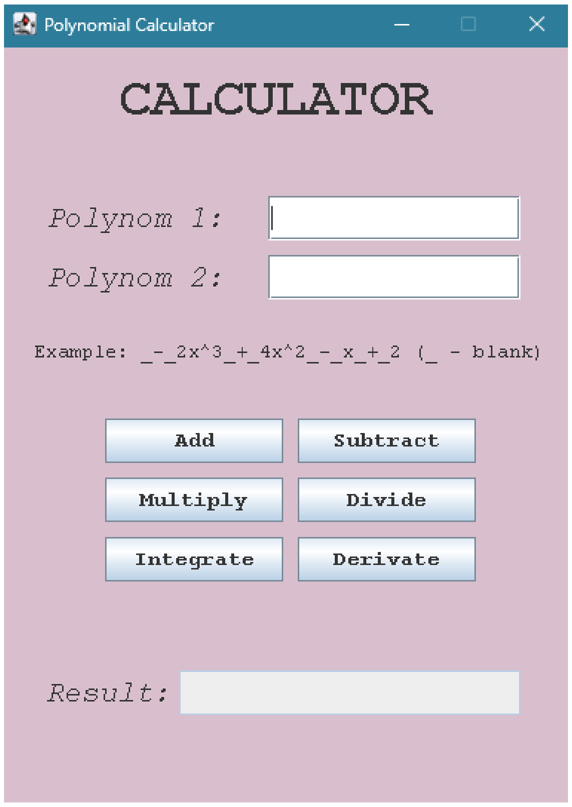
x^3 - 2x^2 + 6x - 5

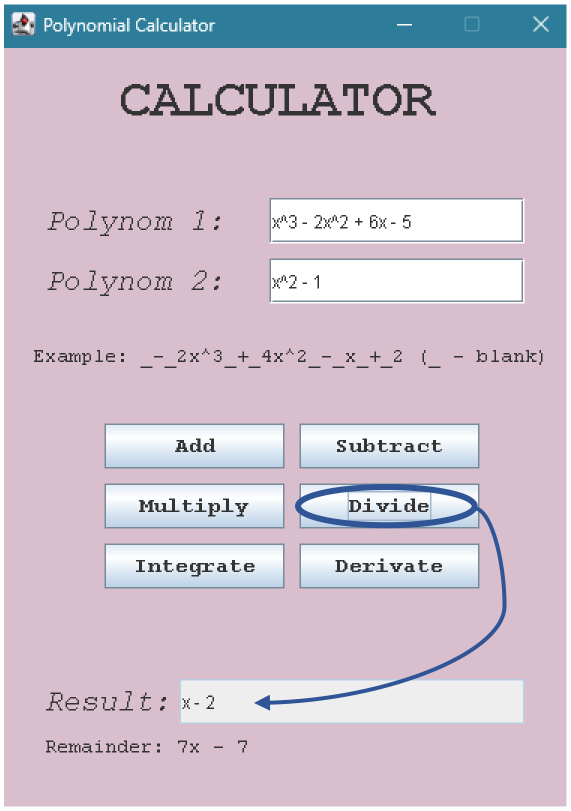
x^2 - 1

Example: _-_2x^3_+_4x^2_-_x_+_2 (_ - blank)

0.25x^4 - 0.67x^3 + 3x^2 - 5x

0.33x^3 - x





x^3 - 2x^2 + 6x - 5

x^2 - 1

x - 2

Remainder: 7x - 7

5. Results

JUnit is a popular testing framework for Java that is used to write and run unit tests applications. It provides annotations and APIs to define test cases, set up test environments, execute tests, and assert expected outcomes. Each of the six operations was subjected to an independent test, facilitated by a method annotated with @BeforeEach to establish the two polynomials necessary for executing the test operations.

- *additionTest():* This test case checks the addition operation of polynomials. It creates two Polynomial objects pol1 and pol2 with specific polynomial expressions, then invokes the addition() method from the Operations class to perform the addition operation. The expected result is compared against the actual result using assertEquals(), ensuring that the addition operation produces the correct polynomial sum.
- *subtractionTest()*
- *multiplicationTest()*
- *divisionTest()*
- *derivationTest()*
- *integrationTest()*

Each test case follows a similar structure of setting up input data, performing the operation under test, and asserting that the computed result matches the expected output. These tests collectively provide comprehensive coverage of the polynomial operations implemented in the Operations class, ensuring their correctness and functionality.

6. Conclusions

The completion of this assignment, developing a polynomial calculator, has expanded my knowledge in polynomial mathematics, Java language, OOP and GUI design, data structures like TreeMap, exception handling, and unit testing with JUnit. It has also increased my understanding of software development practices and the integration of mathematical algorithms into practical applications.

Potential developments for the polynomial calculator application could include implementing advanced polynomial algorithms for increased functionality and incorporating error handling mechanisms to improve user experience. It could involve computing the roots of polynomials and generating graphical plots for them and even enabling the application to handle negative values for exponents.

7. Bibliography

Programming Techniques – Lectures of prof. Cristina POP

Model-View-Controller: https://www.techtarget.com/whatis/definition/model-view-controller-MVC

Diagram: https://www.draw.io/