DOCUMENTATION

ASSIGNMENT 1

STUDENT NAME: Stoica Sergiu

GROUP: 30226

# CONTENTS

# **Assignment Objective** ................................................................................... 3

# **Problem Analysis, Modeling, Scenarios, Use Cases** .................................. 4

# **Functional / Non** – Functional Requirements …………………………………. 4

# 2.2 **Use cases** …………………………………………………………………………………… 5

# **Design** ............................................................................................................. 7

# OOP Design ………………………………………………………………………… 7

# UML Package and Class Diagrams ………………………………………….. 7

# Data Structures …………………………………………………………………… 9

# Used algorithms ………………………………………………………………….. 9

# **Implementation** ............................................................................................. 10

# Polinom Class …………………………………………………………………….. 10

# Operations Class ………………………………………………………………... 11

# Calc\_Controller Class ………………………………………………………….. 12

# Calculator Polinoame Application and the GUI …………………………. 13

# **Results** ........................................................................................................... 14

# **Conclusions** .................................................................................................. 15

# **Bibliography** ..................................................................................................16

# Assignment Objective

The main objective of this assignment is to develop a polynomial calculator capable of performing standard arithmetic operations on polynomials, allowing for both numerical and symbolic computations.

|  |  |  |
| --- | --- | --- |
| **Sub-objective** | **Description** | **Documentation Section** |
| Requirements Analysis | Identify functional and non-functional requirements for the polynomial calculator. | Section 2: Problem Analysis, Scenario Modeling, Use Cases |
| System Design | Create an object-oriented design for the application including UML class and package diagrams. | Section 3: Design |
| Implementation | Develop the necessary classes and methods to handle polynomial operations. | Section 4: Implementation |
| User Interface Development | Design and implement an intuitive user interface that facilitates user interaction with the calculator. | Section 4: Implementation |
| Testing and Validation | Conduct unit testing to ensure the correctness of polynomial operations and validate the application against user requirements. | Section 5: Results |
| Documentation | Prepare comprehensive documentation to detail the development process and support future maintenance and upgrades. | Entire Document |

# Problem Analysis, Modeling, Scenarios, Use Cases

**Functional Requirements:**

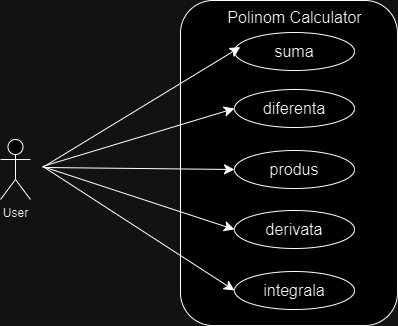
* The system must allow input of polynomials as algebraic expressions.
* The system must support addition, subtraction, multiplication, differentiation, and integration of polynomials.
* The system must validate the input and handle invalid polynomial expressions gracefully.
* The system must display the resulting polynomial after an operation in a standardized algebraic format.
* The system must store the state of the current polynomial to allow for multiple operations.

**Non - Functional Requirements:**

* Calculations must be completed within 1 second to ensure a responsive user experience.
* The user interface (UI) should be intuitive, allowing users to input and edit polynomials without needing to consult a manual.
* The system should have an error rate of less than 0.01% for all operations, ensuring high accuracy in calculations.
* The application's design should allow for the addition of new polynomial operations (e.g., division) without significant refactoring.

**Use Cases:**

* **Use Case 1: Perform Operation with 2 polynomials**
  + Actor: User
  + Goal: To perform an arithmetic operation on polynomials.
  + Steps:
    1. User inputs the first polynomial.
    2. User inputs the second polynomial.
    3. The user selects an operation that requires 2 polynomials, located on the 1st row.
    4. The data from user is being checked.
    5. If the input is good, the system displays the resulting polynomial, otherwise an error is shown.
  + The operations for this action are addition, subtraction, multiplication.
* **Use Case 2: Perform Operation with 1 polynomial**
  + Actor: User
  + Goal: To perform an arithmetic operation on polynomial.
  + Steps:
    1. User inputs the first polynomial.
    2. User selects an operation that requires 1 polynomial, located on the 2nd row.
    3. The data from user is checked.
    4. If the input is good, the system displays the resulting polynomial, otherwise an error is shown.
  + The operations for this action are differentiation, integration.



# Design

**OOP Design**

The application can be structured around three main components, following the MVC (Model-View-Controller) design pattern:

* Model (Polinom and Operations classes)
* View (User Interface)
* Controller (Calc\_Controller class)

The Controller class in the Model-View-Controller (MVC) architecture serves as the intermediary between the View and the Model. It effectively bridges user interactions from the View with data manipulation and logic in the Model.

The View deals only with the graphical interface and user interaction, the Model handles the data and business logic. This design pattern facilitates easier maintenance and scalability of the application, as changes to the business logic or the user interface can be made with minimal impact on the other parts.

**UML Package and Class Diagrams**

* **Model Package**: Contains the **Polinom and Operation** classes, deals with data manipulation.
* **View Package**: Manages user interaction, displaying results, and accepting input.
* **Controller Package**: Includes the **Calc\_Controller** class which serves as an intermediary between the View and Model, processing user commands and invoking the necessary operations on the Model.

A screenshot of a computer program

Description automatically generated

**Data Structures**

* **Tree Map<Number, Number>**: Used in the **Polinom** class to store polynomial terms, ensuring they are ordered by their degrees which facilitates operations like addition, subtraction, and multiplication.

**Used Algorithms**

* **Addition/Subtraction**: Iterate through the terms of both polynomials, adding/subtracting coefficients of terms with the same degree.
* **Multiplication**: For each term in one polynomial, multiply it by every term in the other polynomial, combining like terms.
* **Derivative**: Apply the power rule (multiply the coefficient by the degree and decrease the degree by one) to each term.
* **Integral**: For each term, divide the coefficient by the new degree (original degree + 1) and increase the degree by one.

# Implementation

**Polinom Class**

* **Fields**:
  + **termeni**: A TreeMap storing polynomial terms with keys as degrees and values as coefficients.
* **Important Methods**:
  + **addTerm(Number coeficient, Number grad)**: Adds a term to the polynomial.
  + **isValidPolynomial(String polynomial)**: Validates the input polynomial string.
  + **parseAndAddTerms(String polynomial)**: Parses a string to add terms to the polynomial.
  + **forEach(BiConsumer<Number, Number> action)**: Applies a given action to each term of the polynomial.
  + **toString()**: Overrides to provide a string representation of the polynomial.
  + **toStringImpartiri()**: Enhanced string representation, particularly for division results.

**Operations Class**

* **Important Methods**:
  + **suma**(Polinom polinom1, Polinom polinom2): This method represents the addition operation. It takes two Polinom objects as arguments and returns a new Polinom object that is the sum of the input polynomials. The method iterates over each term of the polynomials and adds the corresponding coefficients of like degrees.
  + **diferenta**(Polinom polinom1, Polinom polinom2): The subtraction operation is encapsulated within this method. It functions similarly to the suma method but subtracts the coefficients of the second polynomial from the first.
  + **produs**(Polinom polinom1, Polinom polinom2): This method computes the product of two polynomials. It implements the distributive property of multiplication over addition, multiplying each term of the first polynomial by every term of the second polynomial and summing the results.
  + **derivat**(Polinom polinom1): This method calculates the derivative of a polynomial. It applies the power rule for differentiation to each term of the polynomial, multiplying the coefficient by the degree and decrementing the degree by one.
  + **integrala**(Polinom polinom1): This method finds the indefinite integral of a polynomial. It divides each coefficient by the incremented degree of its term, applying the reverse operation of the power rule for integration.

**Calc\_Controller Class**

* **Fields**:
  + **p1\_text**: A TextField object for the user to input the first polynomial.
  + **p2\_text**: A TextField object for the user to input the second polynomial.
  + **suma**: A Button object that, when clicked, triggers the addition of two polynomials.
  + **produs**: A Button object that, when clicked, triggers the multiplication of two polynomials.
  + **rezultat\_text**: A Text object to display the result of the polynomial operations.
* **Important Methods**:
  + **sumaPolinoame**(): When the 'suma' button is clicked, this method is invoked. It reads the input polynomials from the p1\_text and p2\_text fields, creates Polinom objects, and adds the terms. It then calls the Operations.suma method with these objects and sets the rezultat\_text with the result.
  + **diferentaPolinoame**(): This method is triggered by a dedicated UI element for subtraction. It similarly processes the inputs and utilizes the Operations.diferenta method to compute the difference between the polynomials, updating the result on the interface.
  + **produsPolinoame**(): Activated when the 'produs' button is clicked, this method handles the multiplication of polynomials. It gathers the inputs, feeds them into the Operations.produs method, and presents the product polynomial in the result display area.
  + **derivataPolinom**(): This method is called to compute the derivative of a polynomial entered in p1\_text. It constructs a Polinom object from the input, calls the Operations.derivat method, and displays the derived polynomial.
  + **integralaPolinom**(): This method responds to the UI control for integration. It processes the polynomial from p1\_text, invokes the Operations.integrala method, and shows the integrated polynomial with the constant of integration denoted by '+C'.

**Calculator Polinoame Application and the GUI**

The CalculatorPolinoame class extends the JavaFX Application class and serves as the main entry point for the polynomial calculator's graphical user interface (GUI). The class is responsible for loading the FXML document that defines the layout and connecting it with the associated controller, Calc\_Controller.

start(Stage stage):

* FXMLLoader: Utilizes FXMLLoader to load the FXML file, which contains the definition of the GUI.
* Scene: Creates a Scene object with the loaded layout, setting its dimensions to 600 by 400 pixels.
* Stage: Sets the title of the application window to "Calculator polinoame!" and displays the scene.

The GUI described by the FXML document provides a user-friendly interface for polynomial operations. It is structured as follows:

* VBox Container: The root container organizing the layout in a vertical column with padding and spacing for aesthetic appeal.
* Title Label: Displays the title of the application "Calculator polinoame" in the Courier New font, size 18, centered at the top.
* Polynomial Input Fields:
  + Two TextField components labeled "Polinomul 1:" and "Polinomul 2:", allowing users to enter two polynomials for operations, with example prompts provided.
* Operation Buttons:
  + "Suma", "Diferenta", and "Produs" buttons are laid out in a FlowPane, each tied to their respective operation methods in the controller.
  + Another set of buttons, "Derivata" and "Integrala", specifically operate on "Polinomul 1" for derivative and integral calculations.
* Result Display:
  + A Text component labeled "Rezultat :" is used to display the result of the operations. The result font matches the input labels for consistency**.**

# Results

**Test Method Summaries:**

* **testSuma()**: This test validates the addition of two polynomials. The test ensures that coefficients are correctly added and the degrees are managed accurately. The test case asserts that **3x^2 + 4x + 5x^2 + 3 = 8x^2 + 4x + 3.**
* **testDiferenta()**: The subtraction test case verifies that polynomial terms are appropriately subtracted from one another and that terms with a zero coefficient are removed. The test case checks that (**5x^3 + 2x^2 + x) – (4x^3 + x + 1) = x^3 + 2x^2 -1** and confirms the absence of the x term.
* **testProdus()**: This test case ensures that the multiplication of two polynomials results in the correct degree and coefficient for each term in the resulting polynomial. It checks if **(3x^2 - x + 1) \* (x – 2) = (3x^3 – 7x^2 + 3x - 2).**
* **testDerivata()**: This test checks the derivative operation, ensuring that the derivative is taken correctly for each term, applying the power rule accurately. The result **3x^2 - 4x + 6** is expected, and the size of the terms is verified.
* **testIntegrala()**: The integration test verifies that each term of the polynomial is integrated correctly, with the coefficients divided by the incremented degree. It checks for the correct terms **0.25x^4**, **1.33x^3**, and **5x**, and confirms the size of the resulting polynomial.

**JUnit Assertions:**

The test methods make use of several JUnit assertion methods:

* **assertEquals**: This assertion checks that two values are equal. It is used to compare the expected result with the actual result from polynomial operations.
* **assertFalse**: This assertion verifies that a condition is false. It is used to ensure that certain terms (like a term with a zero coefficient) do not exist in the result.
* **Assert.assertEquals (size check)**: In addition to checking for specific coefficients, some tests verify the number of terms in the resulting polynomial to ensure that terms with zero coefficients are not included erroneously.

# Conclusions

The development and testing of the polynomial calculator application have illustrated the critical importance of careful design, implementation, and testing in software development. Through the use of object-oriented principles, we constructed a robust application capable of performing a variety of operations on polynomials, including addition, subtraction, multiplication, differentiation, and integration. The application's architecture, divided into model, view, and controller components, adheres to the Model-View-Controller (MVC) pattern, promoting separation of concerns and enhancing maintainability.

**Lessons Learned**

* **Object-Oriented Design**: The assignment reinforced the value of object-oriented design principles such as encapsulation, inheritance, and polymorphism. By treating polynomials as objects, we were able to encapsulate their properties and behaviors, facilitating a clear and modular design.
* **Regular Expressions**: The use of regular expressions for validating polynomial inputs highlighted the power of regex in parsing and validating complex string patterns, an invaluable tool in software development.
* **Unit Testing**: The application of JUnit for unit testing demonstrated the importance of testing in the development process. Writing tests for each core functionality ensured that the application logic was reliable and met the requirements specifications. This practice not only helps in early bug detection but also in maintaining code quality over time.
* **Error Handling**: Implementing error handling mechanisms taught us the significance of anticipating and managing potential runtime errors, providing a more robust and user-friendly application experience.

**Future Developments**

Looking ahead, there are several avenues for further development and enhancement of the polynomial calculator application:

* **Graphical Representation**: Integrating a feature to graphically represent polynomials.
* **Complex Numbers Support**: Extending the application to support operations on polynomials with complex coefficients would broaden the tool's applicability to more advanced mathematical problems.

# Bibliography

GeeksforGeeks. (n.d.). *Regular Expressions in Java*. Retrieved from https://www.geeksforgeeks.org/regular-expressions-in-java/

JUnit. (n.d.). JUnit 5 User Guide. Retrieved from

https://junit.org/junit5/docs/current/user-guide/

Baeldung. (n.d.). *Introduction to JUnit 5*. Retrieved from

https://www.baeldung.com/junit-5