DOCUMENTATION

ASSIGNMENT *1*

STUDENT NAME: Duică Sebastian
GROUP: 30424

CONTENTS

1. Assignment Objective

Main Objective: The main objective of this assignment is to design and implement a polynomial calculator application with a dedicated graphical interface, allowing users to perform various mathematical operations on polynomials.

Sub-Objectives:

Sub-Objective	Section
Implement polynomial addition, subtraction, multiplication, division, derivative, and integration operations based on the provided code in the `org.example.logic` package.	Implementation
Design and implement a user-friendly graphical user interface (GUI) for user interaction using the classes defined in the `org.example.gui` package.	Design, Implementation
Utilize object-oriented programming (OOP) principles as demonstrated in the provided code for modularity and maintainability.	Design, Implementation
Model polynomials using HashMap data structure as illustrated in the `Polynomial` class in the `org.example.data` package.	Implementation
Incorporate regular expressions and pattern matching for extracting polynomial coefficients, as demonstrated in the `parseString` method of the `Polynomial` class.	Implementation
Conduct comprehensive testing using JUnit framework for unit testing, integrating test results into the `Results` section of the documentation.	Results

2. Problem Analysis, Modeling, Scenarios, Use Cases

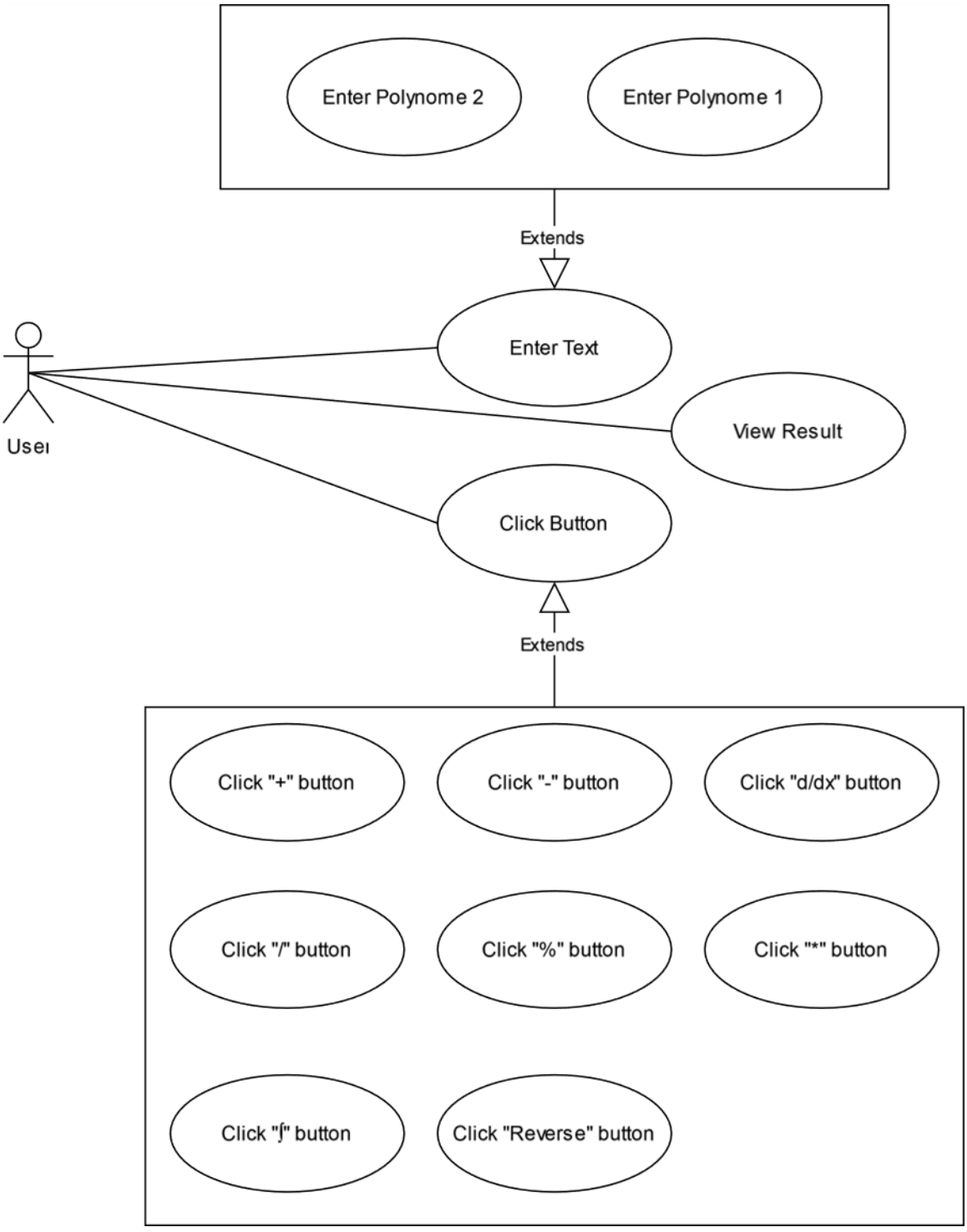

Figure 1 Use Case Diagram

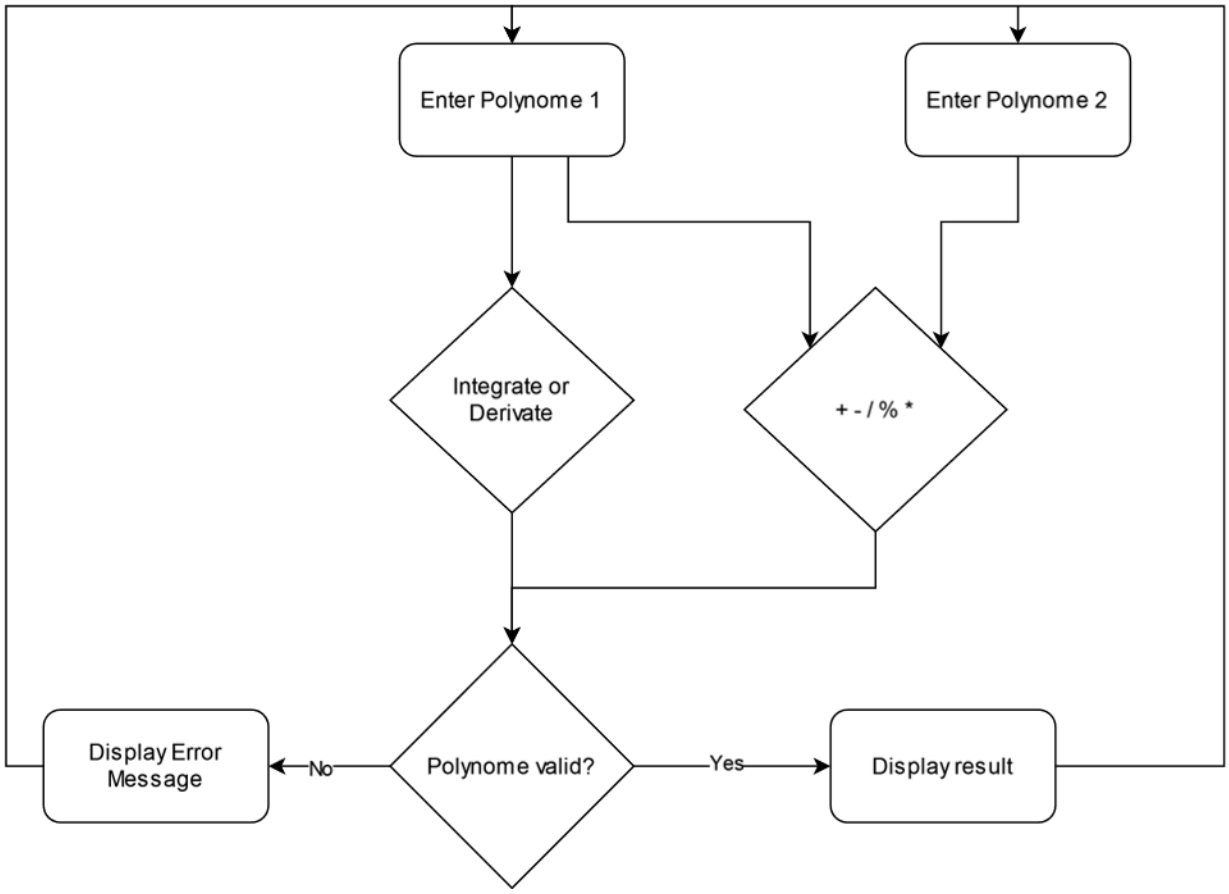

Figure 2 Flow Chart Diagram

3. Design

The main application is structured in 3 packages: gui, logic and data that are used to structure the code with the model view controller design pattern.

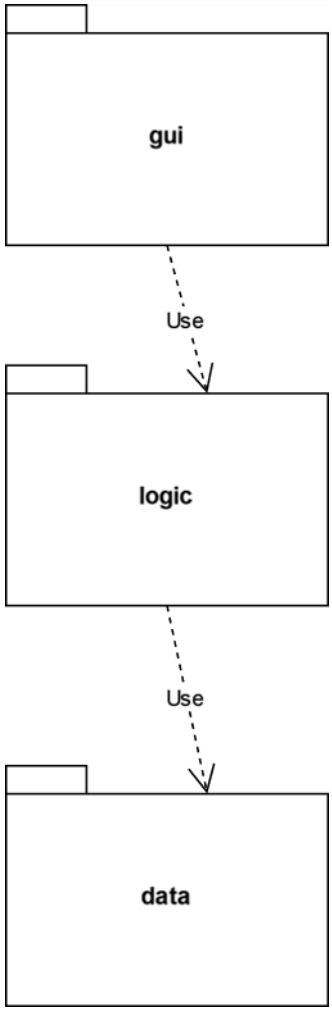

Figure 3 Package Diagram

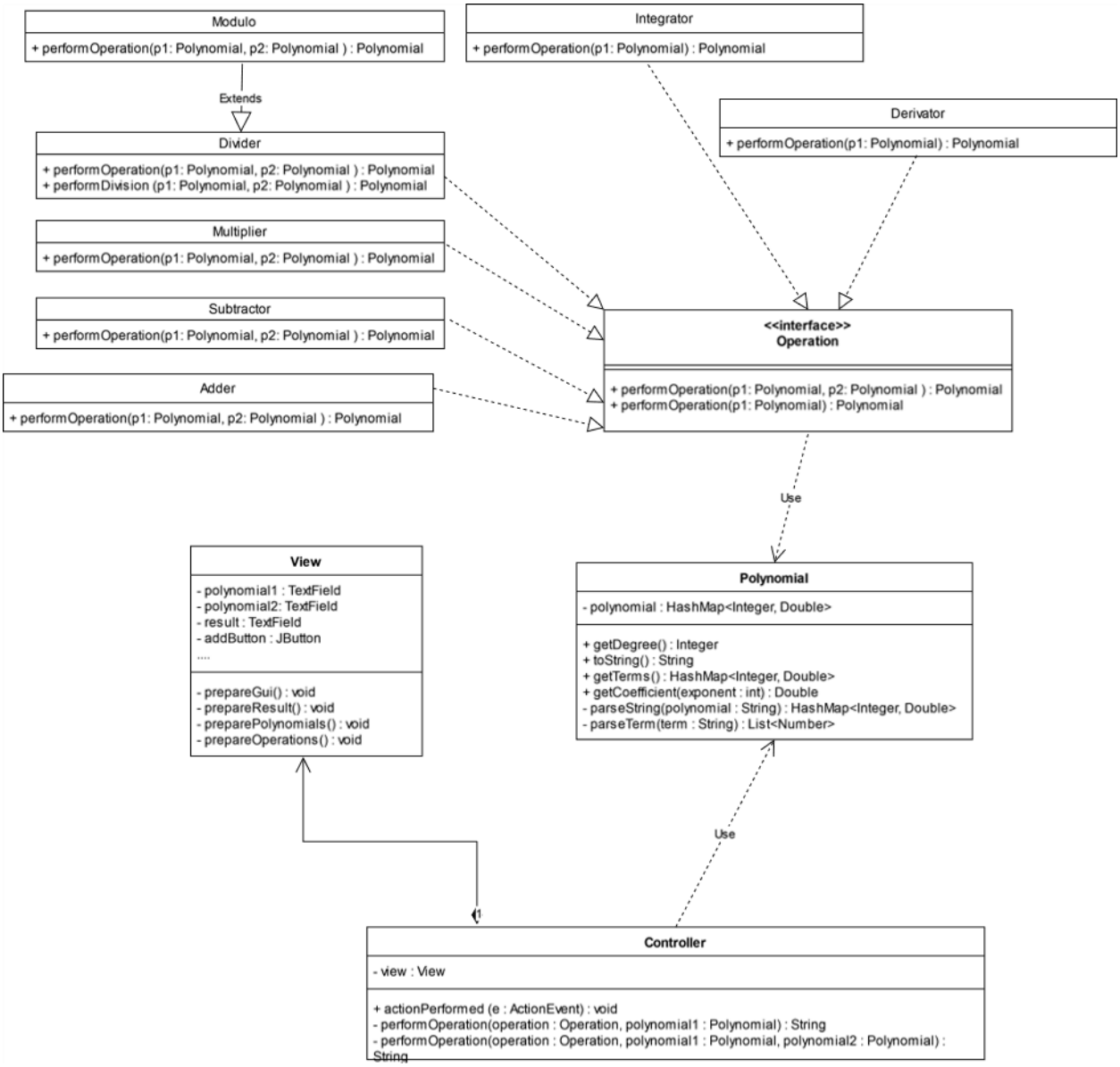

Figure 4 Class Diagram

The main data structure used to store and operate on the polynomials is the HashMap<Integer, Double> that is used to store all the coefficients of the polynomials. The reason behind this decision is because the hash map has a O(1) access time making the operations very efficient.

The Operation interface is used to abstract the implementation of every operations and it is used in the controller to perform every type of operation.

The only algorithm used in this application is the long division algorithm that is used for dividing polynomials, the pseudocode of the implementation is below:

```
function n / d is
    require d ≠ 0
    q ← 0
    r ← n             // At each step n = d × q + r

    while r ≠ 0 and degree(r) ≥ degree(d) do
        t ← lead(r) / lead(d)       // Divide the leading terms
        q ← q + t
        r ← r − t × d

    return (q, r)
```

4. Implementation

Polynomial (org.example.data)

- **Fields:**
  - polynomial: HashMap<Integer, Double> - Represents the polynomial as a mapping of exponents to coefficients.
- **Important Methods:**
  - Polynomial(String polynomial): Constructor that initializes the polynomial using a string representation.
  - getCoefficient(int exponent): Retrieves the coefficient for a given exponent.
  - getDegree(): Returns the degree of the polynomial.
  - toString(): Converts the polynomial to a human-readable string representation.
  - parseString(String polynomial): Parses the string representation of the polynomial.
  - parseTerm(String term): Parses a single term of the polynomial.

Controller (org.example.gui)

- **Fields:**
  - view: View - Reference to the associated View class.
- **Important Methods:**
  - actionPerformed(ActionEvent e): Handles actions performed by the user, such as button clicks.
  - performOperation(Operation operation, Polynomial polynomial1, Polynomial polynomial2): Performs a polynomial operation and returns the result.
  - performOperation(Operation operation, Polynomial polynomial1): Overloaded method to perform unary operations.

View (org.example.gui)

- **Fields:**
  - polynomial1: TextField - Input field for the first polynomial.
  - polynomial2: TextField - Input field for the second polynomial.
  - result: TextField - Display field for the result.

    - switchButton, addButton, subtractButton, divideButton, multiplyButton, derivativeButton, integrateButton, moduloButton: JButtons - Buttons for switching polynomials and performing operations.
- **Important Methods:**
    - getPolynomial1(), getPolynomial2(): Retrieves the input polynomials from the text fields.
    - setPolynomial1(String polynomial), setPolynomial2(String polynomial): Sets the input polynomials in the text fields.
    - setResult(String result): Sets the result in the result text field.
    - prepareGui(): Sets up the graphical user interface layout and components.
    - prepareResult(), preparePolynomials(), prepareOperations(): Helper methods for preparing different sections of the GUI.

Operations Interface (org.example.logic)

Description:

The Operations interface defines the contract for performing various polynomial operations. It serves as a blueprint for implementing different mathematical operations on polynomials.

Methods:

- performOperation(Polynomial p1, Polynomial p2): This method is intended for binary operations such as addition, subtraction, multiplication, division, and modulo operation. It takes two Polynomial objects as input and returns the result of the operation.
- performOperation(Polynomial p1): This overloaded method is used for unary operations such as derivative and integration. It takes a single Polynomial object as input and returns the result of the operation.

Implementation:

- The interface declares two default methods for performing polynomial operations.
- Each method throws an UnsupportedOperationException by default to ensure that implementing classes override them with specific functionality.
- Classes that implement this interface provide concrete implementations for polynomial operations, such as addition, subtraction, multiplication, division, derivative, and integration.
- By defining a common interface for polynomial operations, the code adheres to the principle of abstraction, allowing different implementations to be used interchangeably.
- Implementations of this interface, such as Adder, Subtractor, Multiplier, Divider, Derivator, Integrator, and Modulo, encapsulate the logic for performing specific polynomial operations.
- This interface promotes code reusability and modularity by separating concerns and providing a consistent API for performing polynomial operations.

Implementation of Graphical User Interface (GUI):

- The GUI is implemented using Java Swing components.
- The main window is represented by the View class, which extends JFrame.
- Input fields (TextField) are provided for users to enter polynomials.
- Buttons (JButton) are used for various operations such as addition, subtraction, multiplication, etc.
- The GUI layout is organized using JPanel containers with appropriate layouts (FlowLayout, BoxLayout) to arrange components.
- Event handling is done using ActionListener interface implemented in the Controller class. Each button click triggers the actionPerformed method, which processes the user's input and updates the GUI accordingly.
- Error handling is implemented to handle invalid input polynomials and display appropriate error messages to the user.
- The GUI provides a user-friendly interface for interacting with the polynomial calculator, allowing users to input polynomials, select operations, and view results in real-time.

5. Results

Description:

The Results section presents the testing scenarios and outcomes for each implemented polynomial operation. JUnit tests are used to ensure the correctness and reliability of the implemented functionalities.

JUnit Test Class: PolynomeCalculatorTest

- **Description:** This class contains multiple test methods, each testing a specific polynomial operation.
- **Methods:**
    - testAdd(), testSubtract(), testMultiplication(), testDivision(), testModulo(), testDerivation(), testIntegration(): Each method tests a particular polynomial operation using predefined input polynomials and asserts the expected result against the actual result obtained from the operation.

Test Scenarios:

- **Addition:** Tests addition of two polynomials, ensuring correct summation of coefficients for each term.
- **Subtraction:** Tests subtraction of one polynomial from another, ensuring correct deduction of coefficients for each term.
- **Multiplication:** Tests multiplication of two polynomials, ensuring correct expansion of terms and summation of coefficients.
- **Division:** Tests division of one polynomial by another, ensuring correct quotient is obtained.

- **Modulo Operation:** Tests the modulo operation of one polynomial by another, ensuring the correct remainder is obtained.
- **Derivation:** Tests derivation of a polynomial, ensuring correct differentiation of terms.
- **Integration:** Tests integration of a polynomial, ensuring correct integration of terms.

Implementation:

- Each test method constructs instances of the respective operation classes (e.g., Adder, Subtractor, Multiplier) and provides predefined input polynomials.
- The actual result of the operation is obtained by invoking the corresponding method (e.g., performOperation()) on the operation class instance.
- The expected result is then compared against the actual result using the assertEquals() method provided by JUnit.
- If the actual result matches the expected result, the test passes; otherwise, it fails, indicating a discrepancy in the implementation.

Test Coverage:

- The JUnit tests provide comprehensive coverage of all implemented polynomial operations, ensuring that each operation behaves as expected under various input scenarios.
- Test cases are designed to cover both typical and edge cases to validate the correctness and robustness of the implemented functionalities.

6. Conclusions

Project Summary:

- Designing and implementing a polynomial calculator with a graphical interface presented an engaging challenge, requiring the application of object-oriented programming principles and algorithmic logic.
- The project aimed to fulfill the requirements specified, including encapsulation, proper class design, graphical user interface implementation, and accurate polynomial operations.

Learnings:

- **Object-Oriented Design:** Through this project, a deeper understanding of object-oriented programming principles, such as encapsulation, inheritance, and polymorphism, was gained.
- **Algorithmic Thinking:** Implementing polynomial operations, such as addition, subtraction, multiplication, division, derivative, and integration, honed algorithmic thinking skills and problem-solving abilities.

- **Graphical User Interface Development:** Developing a graphical user interface using Java Swing expanded knowledge of GUI components, event handling, and layout management.
- **Testing Practices:** Employing JUnit for unit testing reinforced the importance of test-driven development (TDD) and ensured the correctness and robustness of the implemented functionalities.

Future Developments:

- **Enhanced Functionality:** Future iterations of the polynomial calculator could incorporate additional features such as support for more complex polynomials, advanced mathematical operations, and interactive graphing capabilities.
- **Improved User Experience:** User experience enhancements, including error handling, input validation, and graphical visualization of polynomial operations, could be implemented to make the application more user-friendly.
- **Optimization and Performance:** Optimization techniques could be applied to improve the efficiency and performance of polynomial operations, especially for large input datasets.
- **Platform Expansion:** Consideration could be given to porting the application to different platforms or integrating it with web-based technologies to reach a broader audience.

7. Bibliography

1. *Polynomial long division* https://en.wikipedia.org/wiki/Polynomial_long_division