DOCUMENTATION

ASSIGNMENT *Polynomial Calculator*

STUDENT NAME: Gozman-Pop Maria-Eliza
GROUP: 30424

CONTENTS

1. Assignment Objective

The main objective is to build a user-friendly polynomial calculator app, with math functions for adding, subtracting, multiplying, dividing, and differentiating polynomials accurately and quickly, making math tasks easier for everyone.

Sub-objectives:

- **Problem Analysis and Requirements Identification:** Analyze requirements for polynomial calculator, including supported operations, input formats, error handling, and GUI specifications.
- **Design Phase:** Create detailed specifications for application architecture, GUI layout, parsing algorithms, mathematical operations, and error handling mechanisms.
- **Implementation:** Translate design specifications into executable code, developing GUI using Java Swing, backend logic for parsing and operations, and integrating components.
- **Testing:** Execute comprehensive testing strategy, including unit tests for individual components, integration tests for system functionality, and user acceptance testing for usability validation.

2. Problem Analysis, Modeling, Scenarios, Use Cases

2.1.

Functional Requirements:

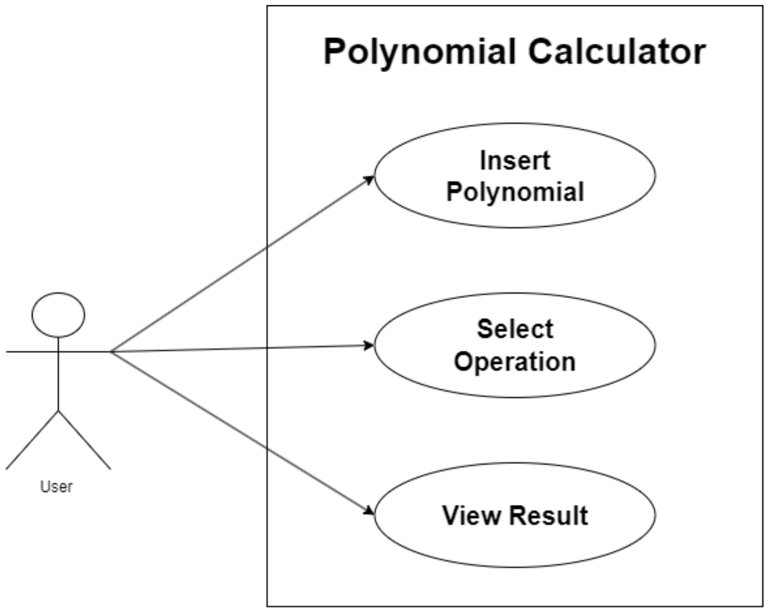
- The polynomial calculator should provide users with the capability to insert polynomial expressions into designated input fields.
- The polynomial calculator should allow users to select the desired operation.
- The polynomial calculator should be able to perform the addition of two polynomials.
- The polynomial calculator should be able to perform the subtraction of two polynomials.
- The polynomial calculator should be able to perform the multiplication of two polynomials.
- The polynomial calculator should be able to perform the division of two polynomials.
- The polynomial calculator should be able to perform the derivation of two polynomials.
- The polynomial calculator should be able to perform the integration of two polynomials.
- The polynomial calculator should display the result of each operation accurately.
- The polynomial calculator should detect and handle errors, providing informative messages to users for invalid inputs or mathematical errors.
- The polynomial calculator should accurately represent polynomials, handling terms with coefficients, exponents, and constant terms appropriately.

Non-functional Requirements:

- The polynomial calculator GUI should be intuitive and easy to use, with clear labels and input fields.

- The polynomial calculator should perform mathematical operations efficiently, even for large polynomials, to ensure a smooth user experience.
- The system should handle errors gracefully, preventing crashes and providing clear error messages to users.
- The polynomial calculator should be scalable to accommodate future enhancements or expansions, to be designed in a modular and extensible manner, allowing for easy integration of new features or updates without significant rework.

2.2. Use Cases:



- User to Insert Polynomial:
  - the user interacts with the system through designated input fields, namely polynomial1Text and polynomial2Text.
  - these input fields allow the user to input polynomial expressions using standard mathematical notation, being able to input terms with coefficients, variables, and exponents.
- User to Select Operation:
  - the system provides buttons corresponding to different mathematical operations: addition, subtraction, multiplication, division, differentiation, and integration.
  - The user selects the desired operation by clicking on the corresponding button.
- User to View Result:
  - the result of the mathematical operation is displayed in the designated area, labeled resultLabel.
  - the result should be presented in a readable format, maintaining the standard polynomial notation (as a polynomial expression).

3. Design

3.1.

The Polynomial Calculator Application demonstrates several key principles of object-oriented programming (OOP), facilitating a robust and flexible design:

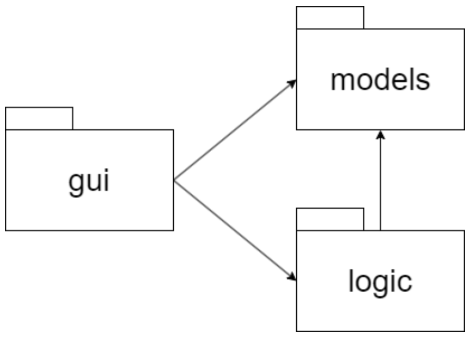
- **Abstraction:** It is achieved through the use of classes, interfaces, and methods, hiding implementation details from users and focusing on essential characteristics. Classes such as Polynomial, Monomial, and DivisionResult abstract real-world concepts related to polynomial mathematics, providing clear interfaces for interaction.

- **Encapsulation:** It is employed to bundle data (e.g., polynomial terms) and methods (e.g., polynomial operations) into cohesive units, ensuring data integrity and promoting modular design. Each class encapsulates its state and behavior, exposing only necessary interfaces to interact with external components.

- **Polymorphism:** It is leveraged through method overriding and interface implementation, allowing objects of different types to be treated uniformly. Interface-based design enables polymorphic behavior, allowing classes to adhere to common contracts while providing specialized implementations.

- **Modularity:** The application is organized into modular packages, each containing related classes and functionalities. Modular design promotes code reuse, maintainability, and scalability by encapsulating related components and minimizing dependencies between modules.

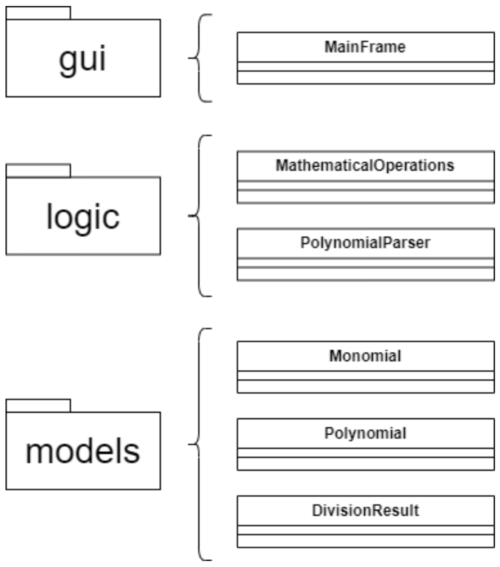
3.2. Division into packages



- The gui package depends on the logic package because the GUI classes (e.g., MainFrame) use classes from the logic package (e.g., MathematicalOperations) to perform mathematical calculations based on user input.
- The gui package also depends on the models package because GUI components may interact with the data model classes (e.g., Polynomial, Monomial) to display polynomial expressions and results.

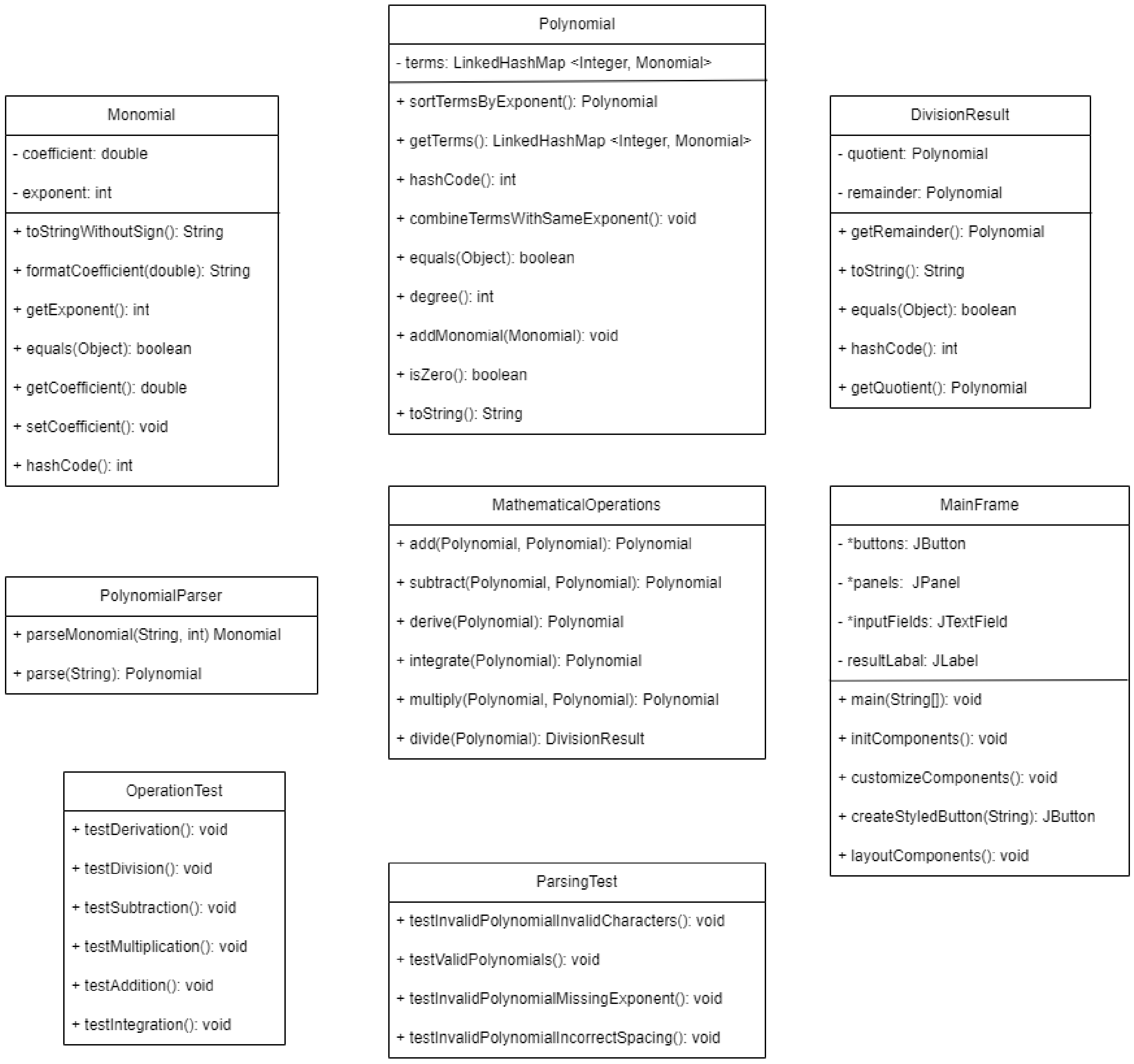
- The logic package depends on the models package to perform mathematical operations (e.g., addition, multiplication) on polynomial objects defined in the models package.

3.3. Division into classes



- **gui Package:**
  - MainFrame: This class represents the main graphical user interface (GUI) window of the Polynomial Calculator Application, dealing primarily with the presentation layer of the application, including Swing components and event handling.
- **logic Package:**
  - MathematicalOperations: This class contains static methods for performing mathematical operations on polynomials, encapsulating the core logic and algorithms of polynomial arithmetic, independent of the user interface.
  - PolynomialParser: This class is responsible for parsing polynomial expressions entered by the user into Polynomial objects, handling the parsing and interpretation of polynomial data, which is a domain-specific task.
- **models Package:**
  - Polynomial: This class represents a polynomial object, consisting of a collection of Monomial objects, aligning with the object-oriented principle of encapsulation, as it represents a fundamental data structure within the application's domain.
  - Monomial: This class represents a single term within a polynomial, consisting of a coefficient and an exponent, encapsulating the properties and behavior of individual monomials, and representing a fundamental building block of polynomials.
  - DivisionResult: This class represents the result of polynomial division, consisting of a quotient and a remainder polynomial. It encapsulates the outcome of polynomial division operations, representing a domain-specific result object within the polynomial arithmetic domain.

3.4. Division into routines


Polynomial
- terms: LinkedHashMap <Integer, Monomial>
+ sortTermsByExponent(): Polynomial
+ getTerms(): LinkedHashMap <Integer, Monomial>
+ hashCode(): int
+ combineTermsWithSameExponent(): void
+ equals(Object): boolean
+ degree(): int
+ addMonomial(Monomial): void
+ isZero(): boolean
+ toString(): String
Monomial
- coefficient: double
- exponent: int
+ toStringWithoutSign(): String
+ formatCoefficient(double): String
+ getExponent(): int
+ equals(Object): boolean
+ getCoefficient(): double
+ setCoefficient(): void
+ hashCode(): int
DivisionResult
- quotient: Polynomial
- remainder: Polynomial
+ getRemainder(): Polynomial
+ toString(): String
+ equals(Object): boolean
+ hashCode(): int
+ getQuotient(): Polynomial
MathematicalOperations
+ add(Polynomial, Polynomial): Polynomial
+ subtract(Polynomial, Polynomial): Polynomial
+ derive(Polynomial): Polynomial
+ integrate(Polynomial): Polynomial
+ multiply(Polynomial, Polynomial): Polynomial
+ divide(Polynomial): DivisionResult
MainFrame
- *buttons: JButton
- *panels: JPanel
- *inputFields: JTextField
- resultLabal: JLabel
+ main(String[]): void
+ initComponents(): void
+ customizeComponents(): void
+ createStyledButton(String): JButton
+ layoutComponents(): void
PolynomialParser
+ parseMonomial(String, int) Monomial
+ parse(String): Polynomial
OperationTest
+ testDerivation(): void
+ testDivision(): void
+ testSubtraction(): void
+ testMultiplication(): void
+ testAddition(): void
+ testIntegration(): void
ParsingTest
+ testInvalidPolynomialInvalidCharacters(): void
+ testValidPolynomials(): void
+ testInvalidPolynomialMissingExponent(): void
+ testInvalidPolynomialIncorrectSpacing(): void

4. Implementation

4.1. Class Description

- **MainFrame:**

This class represents the main graphical user interface (GUI) of the polynomial calculator application. It extends the JFrame class and contains components such as text fields, buttons, and panels arranged using Swing components. The MainFrame class handles user interactions, such as inputting polynomials, selecting mathematical operations, and displaying results. It utilizes action listeners to respond to user input and performs corresponding mathematical operations through the MathematicalOperations class. The GUI is divided into input panels, keyboard panels, and operation panels to organize the components efficiently.

The methods used are:

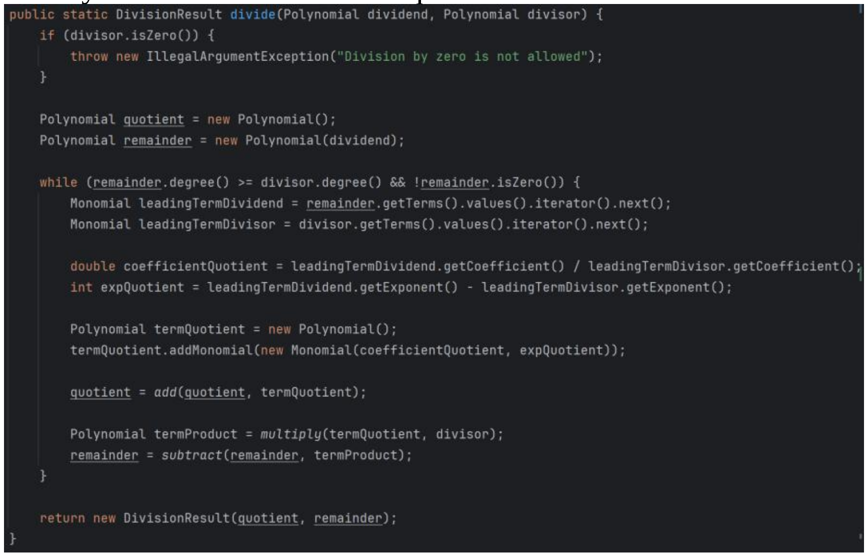
- public MainFrame(String name): Constructor for the MainFrame class.
- private void initComponents(): Initializes all the Swing components used in the GUI.
- private void layoutComponents(): Arranges the initialized components within the GUI.
- private void customizeComponents(): Customizes the appearance of the initialized components.
- private JButton createStyledButton(String text): Creates and configures a styled JButton.
- private class KeyboardButtonListener implements ActionListener: Handles actions performed by keyboard buttons.
- private class TextFieldMouseListener extends MouseAdapter: Handles mouse click events on text fields.
- private class OperationButtonListener implements ActionListener: Handles actions performed by operation buttons.
- public static void main (String[] args): Main method to launch the application.

- **MathematicalOperations:**

This class provides static methods to perform mathematical operations on polynomials, including addition, subtraction, multiplication, division, differentiation, and integration. Each method takes polynomial objects as input and returns the result of the corresponding operation. The class ensures proper handling of polynomial arithmetic, such as combining like terms and sorting terms by exponent. Error handling is implemented to handle scenarios such as division by zero and invalid polynomial formats.

The methods used are:

- public static Polynomial add (Polynomial polynomial1, Polynomial polynomial2): Computes the sum of two polynomials.
- public static Polynomial subtract (Polynomial polynomial1, Polynomial polynomial2): Computes the difference between two polynomials.
- public static Polynomial multiply (Polynomial polynomial1, Polynomial polynomial2): Computes the product of two polynomials.
- public static DivisionResult divide (Polynomial dividend, Polynomial divisor): Divides one polynomial by another and returns the quotient and remainder.



```java
public static DivisionResult divide(Polynomial dividend, Polynomial divisor) {
    if (divisor.isZero()) {
        throw new IllegalArgumentException("Division by zero is not allowed");
    }

    Polynomial quotient = new Polynomial();
    Polynomial remainder = new Polynomial(dividend);

    while (remainder.degree() >= divisor.degree() && !remainder.isZero()) {
        Monomial leadingTermDividend = remainder.getTerms().values().iterator().next();
        Monomial leadingTermDivisor = divisor.getTerms().values().iterator().next();

        double coefficientQuotient = leadingTermDividend.getCoefficient() / leadingTermDivisor.getCoefficient();
        int expQuotient = leadingTermDividend.getExponent() - leadingTermDivisor.getExponent();

        Polynomial termQuotient = new Polynomial();
        termQuotient.addMonomial(new Monomial(coefficientQuotient, expQuotient));

        quotient = add(quotient, termQuotient);

        Polynomial termProduct = multiply(termQuotient, divisor);
        remainder = subtract(remainder, termProduct);
    }

    return new DivisionResult(quotient, remainder);
}
```

(divide() method in MathematicalOperations class)

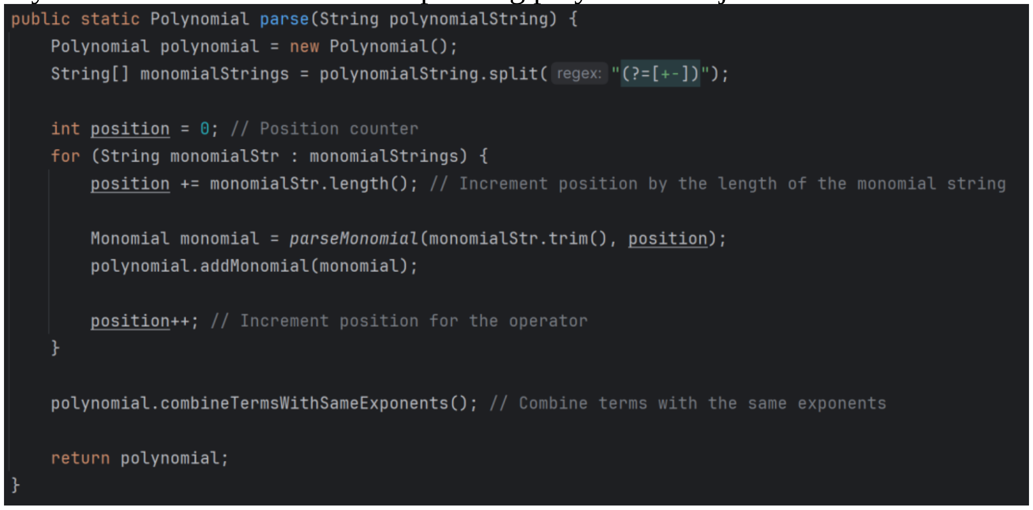
- public static Polynomial derive (Polynomial polynomial): Computes the derivative of a polynomial.
- public static Polynomial integrate (Polynomial polynomial): Computes the integral of a polynomial.

- **PolynomialParser:**

The PolynomialParser class parses string representations of polynomials into polynomial objects. It extracts monomials from polynomial strings, parses their coefficients and exponents, and constructs polynomial objects accordingly. Regular expressions are used to match monomial patterns and extract relevant information. The parser ensures robust handling of input strings and throws exceptions for invalid polynomial formats.

The methods used are:

- public static Polynomial parse (String polynomialString): Parses a string representation of a polynomial and returns the corresponding polynomial object.



```java
public static Polynomial parse(String polynomialString) {
    Polynomial polynomial = new Polynomial();
    String[] monomialStrings = polynomialString.split( regex: "(?=[+-])");

    int position = 0; // Position counter
    for (String monomialStr : monomialStrings) {
        position += monomialStr.length(); // Increment position by the length of the monomial string

        Monomial monomial = parseMonomial(monomialStr.trim(), position);
        polynomial.addMonomial(monomial);

        position++; // Increment position for the operator
    }

    polynomial.combineTermsWithSameExponents(); // Combine terms with the same exponents

    return polynomial;
}
```

(parse() method in PolynomialParser class)

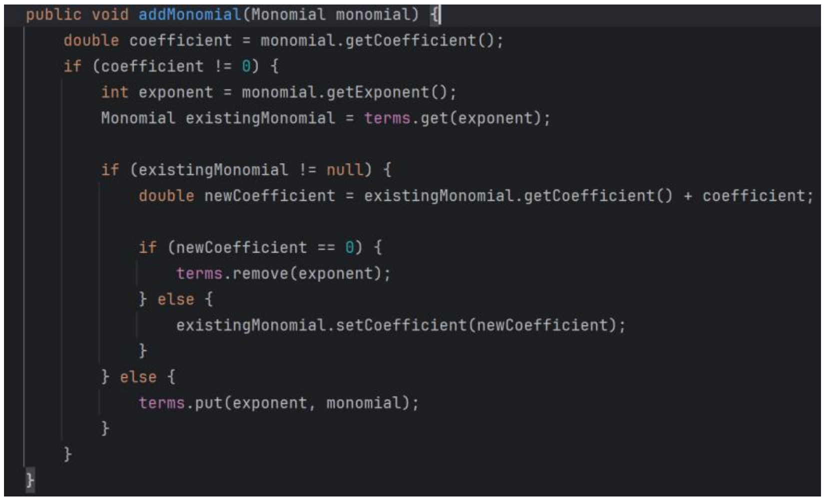
- private static Monomial parseMonomial(String monomialString, int position): Parses a string representation of a monomial and returns the corresponding monomial object.

- **Polynomial:**

This class represents a polynomial as a collection of monomials stored in a linked hash map. It provides methods to add, subtract, multiply, and manipulate polynomials, such as combining like terms and sorting terms by exponent. The class implements functionality to calculate the degree of a polynomial and determine if it is zero. Polynomial objects are used extensively throughout the application for mathematical computations and display purposes.

The methods used are:

- public LinkedHashMap<Integer, Monomial> getTerms(): Returns the terms of the polynomial.
- public int degree() : Returns the degree of the polynomial.
- public boolean isZero(): Checks if the polynomial is zero.
- public String toString(): Returns a string representation of the polynomial.
- public void addMonomial(Monomial monomial): Adds a monomial to the polynomial. If a monomial with the same exponent already exists, their coefficients are combined.



```
public void addMonomial(Monomial monomial) {
    double coefficient = monomial.getCoefficient();
    if (coefficient != 0) {
        int exponent = monomial.getExponent();
        Monomial existingMonomial = terms.get(exponent);

        if (existingMonomial != null) {
            double newCoefficient = existingMonomial.getCoefficient() + coefficient;

            if (newCoefficient == 0) {
                terms.remove(exponent);
            } else {
                existingMonomial.setCoefficient(newCoefficient);
            }
        } else {
            terms.put(exponent, monomial);
        }
    }
}
```

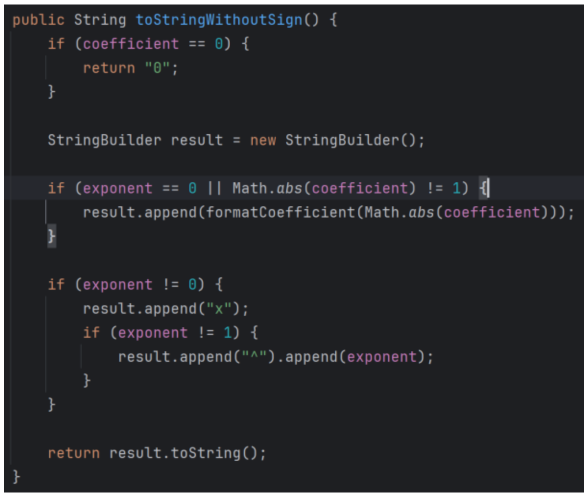
(addMonomial() method in Polynomial class)

- public Polynomial sortTermsByExponent(): Sorts the terms of the polynomial by exponent in descending order.
- public void combineTermsWithSameExponents(): Combines terms of the polynomial with the same exponent.

- **Monomial:**

The Monomial class represents individual terms within a polynomial, consisting of a coefficient and an exponent. It encapsulates the data and behavior associated with monomials, such as retrieving coefficient and exponent values, and formatting monomial strings. The class ensures proper handling of monomial arithmetic and provides methods to modify coefficient values.

- public double getCoefficient(): Returns the coefficient of the monomial.
- public int getExponent(): Returns the exponent of the monomial.
- public void setCoefficient(double newCoefficient): Sets a new coefficient for the monomial.
- public String toStringWithoutSign(): Returns a string representation of the monomial without the sign.



```
public String toStringWithoutSign() {
    if (coefficient == 0) {
        return "0";
    }

    StringBuilder result = new StringBuilder();

    if (exponent == 0 || Math.abs(coefficient) != 1) {
        result.append(formatCoefficient(Math.abs(coefficient)));
    }

    if (exponent != 0) {
        result.append("x");
        if (exponent != 1) {
            result.append("^").append(exponent);
        }
    }

    return result.toString();
}
```

(toStringWithoutSign() method in Monomial class)

- **DivisionResult:**

This class encapsulates the result of polynomial division, consisting of quotient and remainder polynomials. It provides methods to retrieve the quotient and remainder polynomials, as well as a string representation of the division result. Instances of DivisionResult are used to represent the outcome of polynomial division operations.

The methods used are:

- public Polynomial getQuotient(): Returns the quotient polynomial.
- public Polynomial getRemainder(): Returns the remainder polynomial.
- @Override public String toString(): Returns a string representation of the division result.

- **OperationTest**

The OperationTest class is responsible for testing the various mathematical operations implemented in the MathematicalOperations class, including addition, subtraction, multiplication, division, differentiation, and integration. These tests ensure that the implemented operations produce correct results for different scenarios and edge cases.

The methods used are:

- testAddition();
- testSubtraction();
- testMultiplication();
- testDivision();
- testDerivation();
- testIntegration();

- **ParsingTest**

The ParsingTest class is responsible for testing the functionality of the PolynomialParser class, which parses polynomial strings into polynomial objects. These tests ensure that the parsing logic correctly handles valid polynomial strings and appropriately detects and handles invalid input.

The methods used are:

- testValidPolynomial();
- testInvalidPolynomialMissingExponent();
- testInvalidPolynomialInvalidCharacters();
- testInvalidPolynomialIncorrectSpacing();

4.2. Graphical User Interface

The user interface of the polynomial calculator is implemented using Java Swing. The UI consists of various input fields, buttons, and display areas where users can input polynomial expressions, select operations, and view results.

The backend logic of the polynomial calculator is implemented using Java classes. These classes handle polynomial operations such as addition, subtraction, multiplication, division, differentiation, and integration. The calculator parses polynomial expressions entered by the user, performs the requested operation using the appropriate method, and displays the result back to the user.

The MainFrame class serves as the main graphical user interface component of the polynomial calculator program. It organizes the layout of UI components, handles user interactions, and delegates operations to other classes responsible for computation and processing of polynomial expressions.

- **Graphical User Interface Setup:**
  - The MainFrame class extends JFrame, which is a top-level container for Swing components, and it initializes the main window of the application.
  - Within the MainFrame, text fields are provided for users to input polynomial expressions. Additionally, buttons for selecting operations (e.g., addition, subtraction) and initiating calculations are included.

- **Event Handling:**
  - The MainFrame class implements action listeners to handle user interactions with UI components. Action listeners are attached to buttons, allowing the program to respond when users click on them.
  - When a user clicks on a button (e.g., "Add"), the corresponding action listener method is invoked. The MainFrame class captures these events and triggers the appropriate operations or actions.

- **Delegation to Calculation Logic:**
  - Based on the user's selection of polynomial operations (e.g., addition, subtraction), the MainFrame class delegates the computation tasks to helper classes responsible for performing the selected operation.
  - Before performing operations, the MainFrame class may parse the polynomial expressions entered by the user to ensure they are in the correct format. It utilizes a parsing utility or regular expressions for this task.

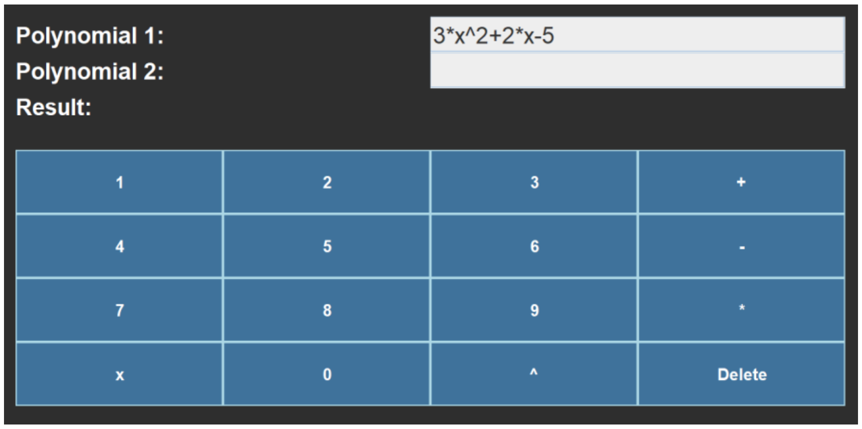
- **Error Handling and Output Display:**
  - If an error occurs during computation (e.g., invalid input, division by zero), the MainFrame class handles these exceptions. It displays error messages to alert the user about the issue and prompts them to correct the input.
  - Upon successful completion of a computation, the MainFrame class presents the result of the operation in the designated output area of the GUI. It updates the display with the computed polynomial expression for the user to view.

User Guide:

1. **Inputting Polynomial Expressions:**

- Entering Polynomials: Users can input polynomial expressions in a standard format using the text field and the keyboard provided. The format follows conventional polynomial notation, where terms are represented as coefficients followed by variables raised to a power.

  Example: 3*x^2+2*x-5

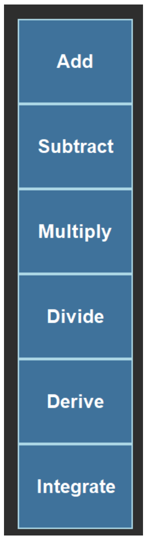

Polynomial 1: 3*x^2+2*x-5

Polynomial 2:

Result:

(Input text fields and keyboard)

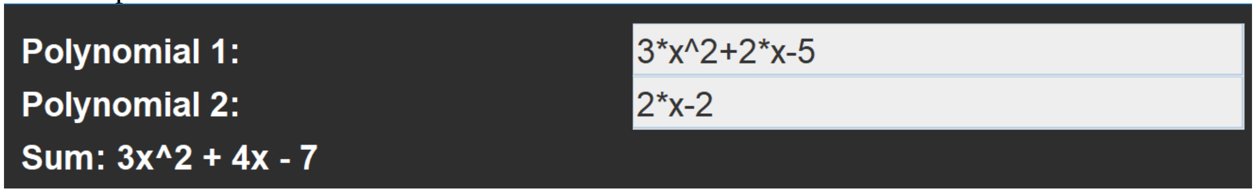
- Supported Operations: Users can select the operation they want to perform from the available options, including addition, subtraction, multiplication, division, differentiation, and integration.



(Operation buttons)

2. **Performing Operations:**

- Selecting Operation: After entering the polynomial expressions and selecting the desired operation, users can initiate the calculation by clicking on any of the operation buttons
- Viewing Results: The result of the operation is displayed in the output area below the input fields. Users can see the computed polynomial expression representing the result of the operation.


Polynomial 1: 3*x^2+2*x-5

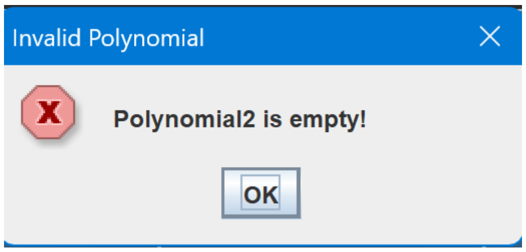
Polynomial 2: 2*x-2

Sum: 3x^2 + 4x - 7

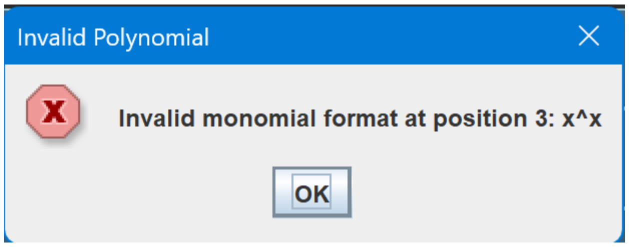
(Inputs and output after calculating the sum)

3. **Error Handling**

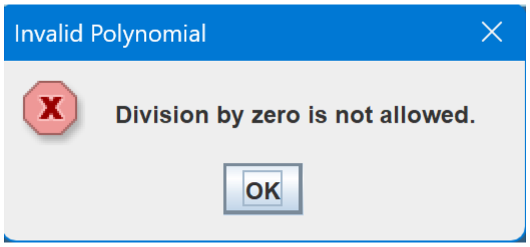
- Invalid Input: If users enter an invalid polynomial expression or encounter an error during calculation (such as division by zero or invalid input format), an error message is displayed to inform the user about the issue.


Invalid Polynomial

Polynomial2 is empty!


Invalid Polynomial

Invalid monomial format at position 3: x^x


Invalid Polynomial

Division by zero is not allowed.

(Possible input error responses)

5. Results

The polynomial calculator underwent testing to ensure its functionality, reliability, and user-friendliness. Tests were designed to cover all aspects of the application, including mathematical operations and user input validation. Results from the testing phase provide insights into the calculator's performance and effectiveness in handling various scenarios. Issues identified during testing were addressed to enhance the application's stability and accuracy, resulting in a more robust and dependable tool for polynomial calculations.

- **OperationTest Class**

Test Methods:

- testAddition():
  Purpose: This method tests polynomial addition.
  Test Cases:
  - Addition of polynomials with terms of the same degree.
  - Addition with a polynomial containing zero coefficients.
  - Addition with a polynomial containing negative coefficients.

- testSubtraction():
  Purpose: This method tests polynomial subtraction.
  Test Cases:
  - Subtraction of polynomials with terms of the same degree.
  - Subtraction with a polynomial containing zero coefficients.
  - Subtraction with a polynomial containing negative coefficients.

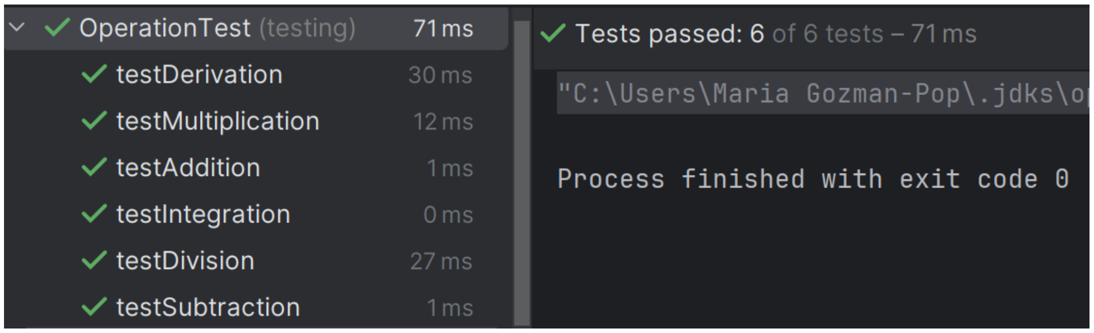
- testMultiplication():
  Purpose: This method tests polynomial multiplication.
  Test Cases:
  - Multiplication of polynomials with terms of the same degree.
  - Multiplication with an empty polynomial.
  - Multiplication with a polynomial containing zero coefficients.
  - Multiplication with a polynomial containing negative coefficients.

- testDivision():
  Purpose: This method tests polynomial division.
  Test Cases:
  - Simple division.
  - Division with a non-zero remainder.
  - Division with a zero dividend.

- testDerivation():
  Purpose: This method tests polynomial differentiation.
  Test Cases:
  - Derivation of a simple polynomial.
  - Derivation of a polynomial with zero coefficients.
  - Derivation of a polynomial with negative coefficients.
  - Derivation of a constant polynomial.
  -

- testIntegration():

Purpose: This method tests polynomial integration.
Test Cases:
  - Integration of a simple polynomial.
  - Integration of a polynomial with negative coefficients.
  - Integration of a polynomial with zero coefficients.



```
✓ OperationTest (testing)    71 ms
    ✓ testDerivation         30 ms
    ✓ testMultiplication     12 ms
    ✓ testAddition            1 ms
    ✓ testIntegration         0 ms
    ✓ testDivision           27 ms
    ✓ testSubtraction         1 ms
```

```
✓ Tests passed: 6 of 6 tests – 71 ms
"C:\Users\Maria Gozman-Pop\.jdks\o

Process finished with exit code 0
```

(Operation Tests Passed)

- **ParsingTest Class**

Test Methods:

- testValidPolynomial():

  Test Case:
  - Parsing a valid polynomial string into a polynomial object.

- testInvalidPolynomialMissingExponent():.

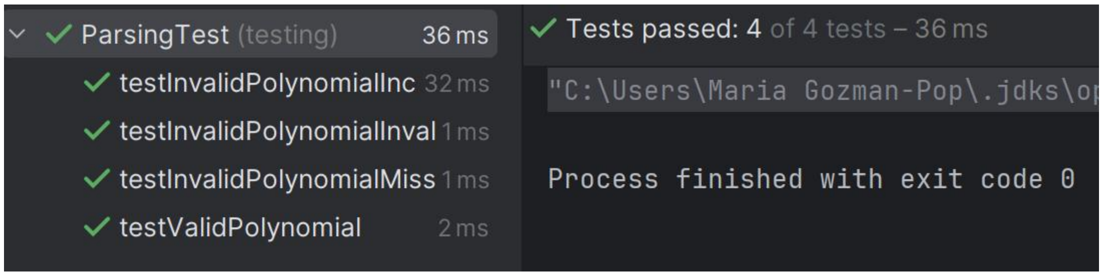
  Test Case:
  - Parsing an invalid polynomial string with a missing exponent, expecting an IllegalArgumentException to be thrown.

- testInvalidPolynomialInvalidCharacters():

  Test Case:
  - Parsing an invalid polynomial string with invalid characters, expecting an IllegalArgumentException to be thrown.

- testInvalidPolynomialIncorrectSpacing():

  Test Case:
  - Parsing an invalid polynomial string with incorrect spacing, expecting an IllegalArgumentException to be thrown.



```
✓ ParsingTest (testing)             36 ms   ✓ Tests passed: 4 of 4 tests – 36 ms
    ✓ testInvalidPolynomialInc      32 ms
    ✓ testInvalidPolynomialInval     1 ms   "C:\Users\Maria Gozman-Pop\.jdks\op
    ✓ testInvalidPolynomialMiss      1 ms
    ✓ testValidPolynomial            2 ms   Process finished with exit code 0
```

(Parsing Tests passed)

6. Conclusions

The following should be presented: conclusions, what you have learned from the assignment, future developments.
-specify what we have achieved (and future work)

Throughout this project, several valuable lessons have been learned. Firstly, implementing a graphical user interface (GUI) in Java has provided insights into event-driven programming and user interaction. Understanding how to design and structure GUI components to create a better user experience has been an important aspect of this learning process. Additionally, working with polynomial arithmetic and parsing has deepened the understanding of fundamental mathematical concepts in computer science. Parsing user input to manipulate polynomials requires careful attention to detail and error handling, which has required problem-solving skills and attention to edge cases. Moreover, collaborating on a larger project involving multiple classes and interactions has emphasized the importance of code organization, readability, and modularity. This project has highlighted the significance of designing classes with clear responsibilities and interfaces to facilitate code maintenance and extension.

Looking ahead, future developments could involve the addition of more advanced mathematical functions such as root-finding algorithms or polynomial factorization, and enhancing the functionality of the polynomial calculator, such as adding support for more advanced operations like polynomial evaluation, polynomial interpolation, or graph plotting. Additionally, improvements to the user interface and error handling could enhance the overall user experience.

7. Bibliography

The following references were consulted during the implementation of the project:

1. https://dsrl.eu/courses/pt/materials/PT_2024_A1_S1.pdf
2. https://dsrl.eu/courses/pt/materials/PT_2024_A1_S2.pdf
3. https://dsrl.eu/courses/pt/materials/PT_2024_A1_S3.pdf
4. https://www.geeksforgeeks.org/introduction-to-junit-5/
5. https://www.javatpoint.com/java-swing/