DOCUMENTATION

ASSIGNMENT 1

STUDENT NAME: MUNTIAN MARIANA-IONELA
GROUP: 30423

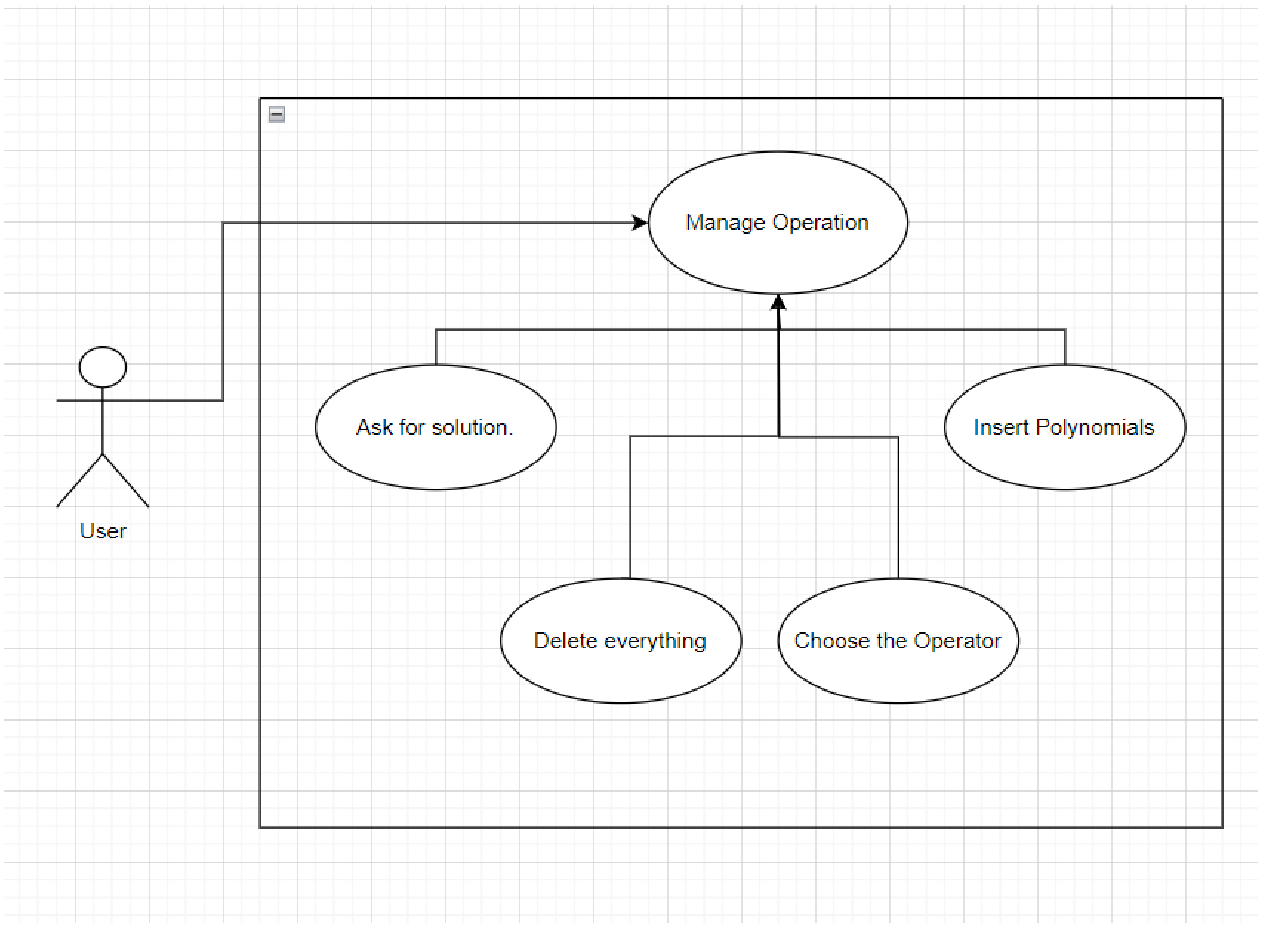
CONTENTS

1. Assignment Objective

The main objective of this assignment is to gain proficiency in building an application with a graphical user interface (GUI) in an OOP manner that implements algorithms for fundamental polynomial operations, including addition, subtraction, multiplication, division, integration, and differentiation. These operations will be interactively utilized by the user within the GUI interface.

Sub-objective	***Description***	***Section***
1. Analyze the problem and identify requirements	*The problem is analyzed from the point of view of the use cases and the functional requirements.*	*2. Problem Analysis, Modeling, Scenarios, Use Cases*
2. Design the polynomial calculator	*UML packages, UML diagram, the data structures and the algorithms that are used are presented.*	*3. Design*
3. Implement the polynomial calculator	*The implementation of the interface and the structure of each class are described.*	*4. Implementation*
4. Test the polynomial calculator	*A testing class is created for each class within the application to verify its functionality and ensure correctness through various test cases.*	*5.Results*

2. Problem Analysis, Modeling, Scenarios, Use Cases



Reference: [Lecture_2_UML_Diagrams.pdf]

Use Case 0: Manage Operation	
Brief Description	*The user must follow the steps required at Basic Flow in order to find the solution for the operation on her/his polynomials.*
Parent Scenario	-
Actor(s)	*The User*
Priority	*High*
Trigger	*Open the application*
Pre-conditions	-
Post-conditions	-
Basic Flow	*Step 1: Insert Polynomial 1* *Step 2: Insert Operation (+ - / *)* *Step 3: Insert Polynomial 2* *Step 4: Ask for solution* *Step 5: Delete everything*
Alternate Flow(s)	*Step 1: Insert Polynomial 1* *Step 2: Insert Operation (integration, derivative)* *Step 3: Ask for solution* *Step 4: Delete everything*
Exception Flow(s)	*Step 1: Insert Polynomial 1 (wrong format)* *Step 2: Delete everything*

	Step 3: Retake the steps from the Basic/Alternate Flow

Use Case 1: Insert Polynomial	
Brief Description	*The polynomials provided by the user must have the following format: ax^power+bx^power+…+d.* *After the 1st polynomial is introduced, the user must write the operation that he/she wants to perform on the polynomial and then press enter.* *After the 2nd polynomial is introduced, the user should press '='.*
Parent Scenario	*Manage Operation*
Actor(s)	*The User*
Priority	*High*
Trigger	*Open the application and press the buttons designed in the GUI.*
Pre-conditions	-
Post-conditions	*Choose the operation*
Basic Flow	*Step 1: Insert the coefficient (if none is inserted, 1 will be the default one)* *Step 2: Insert the variable 'x'* *Step 3: Insert the symbol '^'* *Step 4: Insert the power* *Step 4: Insert the sign (-,+) for the next monomial if any*
Alternate Flow(s)	*Step 1: Insert the coefficient (if none is inserted, 1 will be the default one)* *Step 2: Insert the variable 'x'*
Exception Flow(s)	*Step 1: Insert Polynomial 1* *(wrong format: coefficient after x or consecutive symbol "xx")* *Step 2: Delete everything* *Step 3: Retake the steps from the Basic/Alternate Flow*

Use Case 2: Choose the operator	
Brief Description	*After the 1st polynomial is introduced, the user must write the operation that he/she wants to perform on the polynomial and then press enter.* *If the operation is integration/derivative, the user must press '=' instead of enter.*
Parent Scenario	*Manage Operation*
Actor(s)	*The User*
Priority	*High*
Trigger	*Open the application and press the buttons designed in the GUI: + , - ,* , / , dx, integration*
Pre-conditions	*Insert Polynomial*
Post-conditions	*Press '=' to find the result*
Basic Flow	*Step 1: Insert the polynomial* *Step 2: Insert the operation symbol* *Step 3: Press enter if the operators are + , - ,* , /,*
Alternate Flow(s)	*Step 1: Insert the polynomial* *Step 2: Insert the operation symbol*
Exception Flow(s)	-

Use Case 3: Ask for solution	
Brief Description	*After the polynomials and the operation between them were introduced, the user should press "=" to find the result.*
Parent Scenario	*Manage Operation*
Actor(s)	*The User*
Priority	*Medium*
Trigger	*Press "=" button*
Pre-conditions	*Insert Polynomials and Choose the operator*

Post-conditions	-
Basic Flow	*Step 1: Press '='*
Alternate Flow(s)	-
Exception Flow(s)	-

Use Case 4: Delete everything	
Brief Description	*If the user wants to delete the polynomials or the operator, he/she must press "Clear" button. If there existed a format error, the user must press "Clear".*
Parent Scenario	*Manage Operation*
Actor(s)	*The User*
Priority	*High*
Trigger	*Press "Clear" button*
Pre-conditions	-
Post-conditions	-
Basic Flow	*Step 1: Press 'Clear'*
Alternate Flow(s)	-
Exception Flow(s)	-

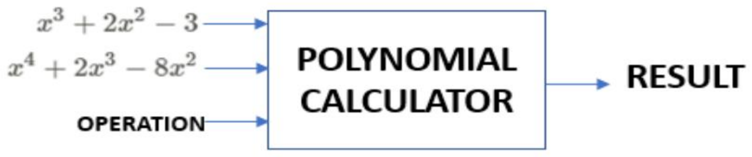
Functional requirements:

- *The polynomial calculator should allow users to insert polynomials*
- *The polynomial calculator should allow users to select the mathematical operation*
- *The polynomial calculator should add, subtract, multiply, divide two polynomials and integrate and derivate a polynomial.*

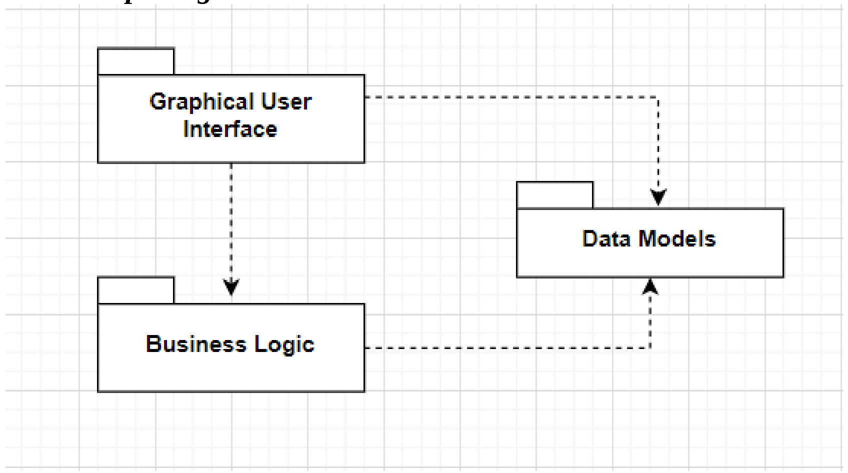
Non-Functional requirements:

- *The polynomial calculator should be intuitive and easy to use by the user*
- *The polynomial calculator should have a adequate interface.*

3. Design

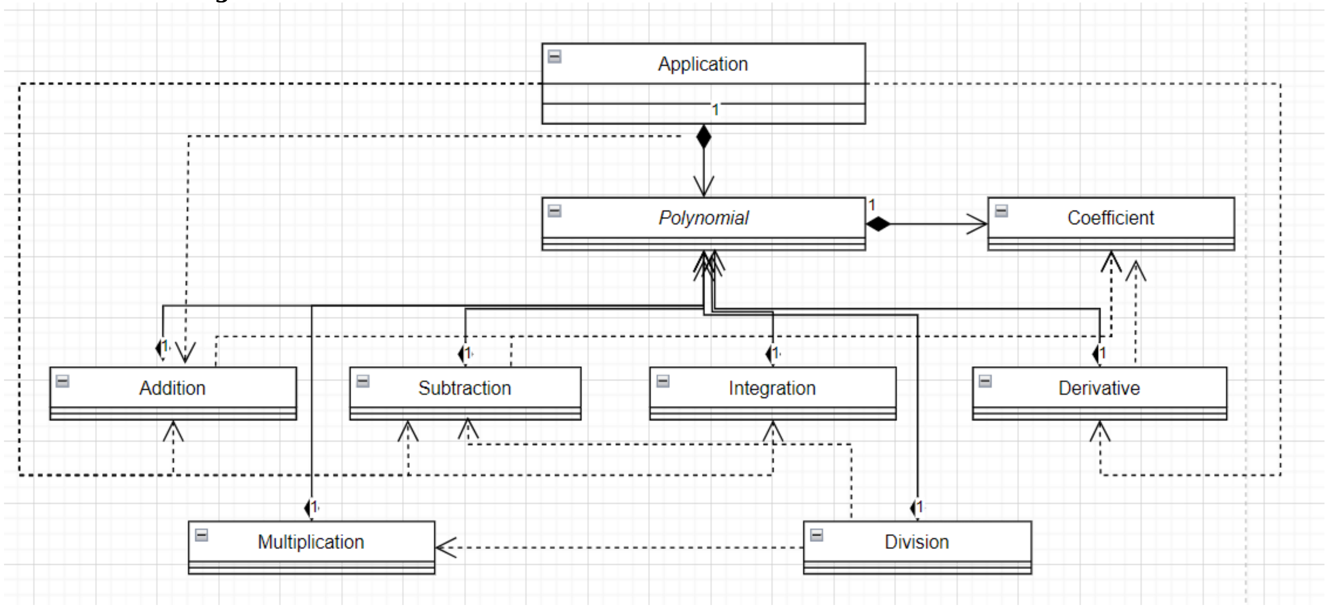


- ***UML packages***



Reference: https://dsrl.eu/courses/pt/materials/PT_2024_A1_S1.pdf

- ***Class Diagram***



Reference: Lecture_2_UML_Diagrams.pdf

- ***Used Data Structures***

A HashMap is used to associate the coefficient of a monomial with its power. The power represents the key of the map and the coefficient represents the value of the map. If collision occurs (more monomials with the same power exist), the sum between the coefficients and the result will represent the coefficient.

The coefficient is an instance of a class called Coefficient, which has 2 fields (numerator and denominator). This approach allows the display of non-integer coefficients.

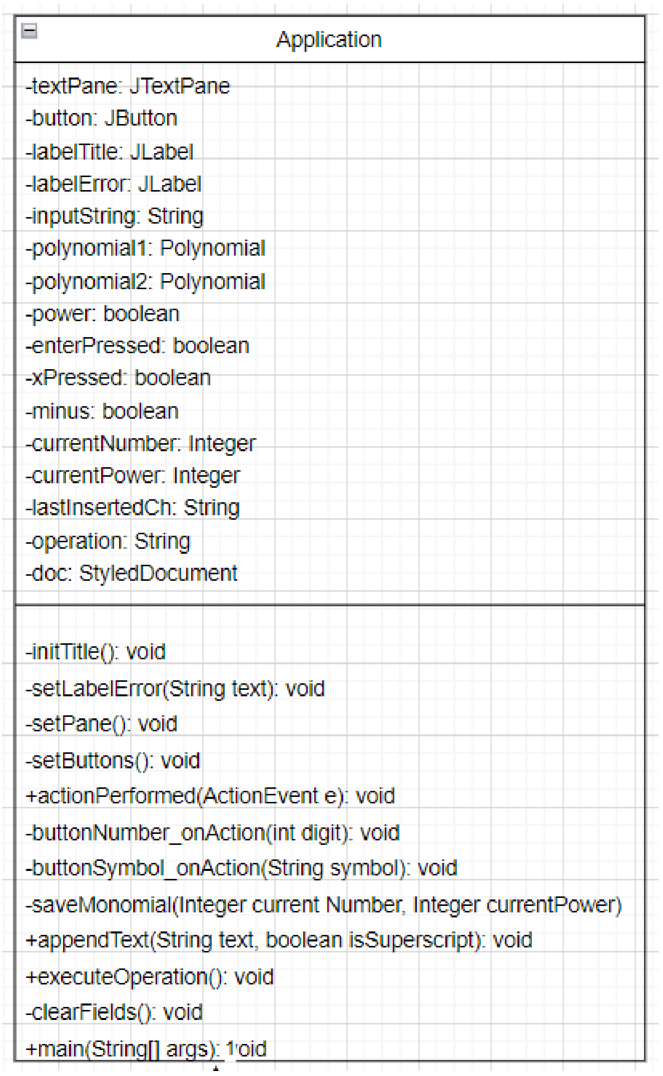
- ***Used Algorithms***

Pseudocode for division operation:

```
polynomialDivision(dividend, divisor):
        quotient = new Polynomial()
        remainder = new Polynomial()

        if divisor is zero:
            display "Division by zero is not possible!"
            return

        maxDividend = getFirstMonomial(dividend)
        maxDivisor = getFirstMonomial(divisor)

        while maxDividend is not null and degree of maxDividend >= degree of maxDivisor:
            term = divisionMonomials(maxDividend, maxDivisor)
            multiplicationPolynomial = multiplyPolynomials(divisor, term)
            dividend = subtractPolynomials(dividend, multiplicationPolynomial)
            maxDividend = getFirstMonomial(dividend)

        quotient = dividend
        remainder = dividend
        quotient.removeNullTerms()
        remainder.removeNullTerms()
```

4. Implementation

a) Implementation of the graphical user interface



The Graphical user interface is designed using Swing.
Fields:

- *private static JTextPane textPane: Text pane used to display input and results.*
- *private JButton button: Button used for user interaction.*
- *private JLabel labelTitle: Label for the title of the application.*
- *private JLabel labelError: Label to display error messages.*

Constructor:

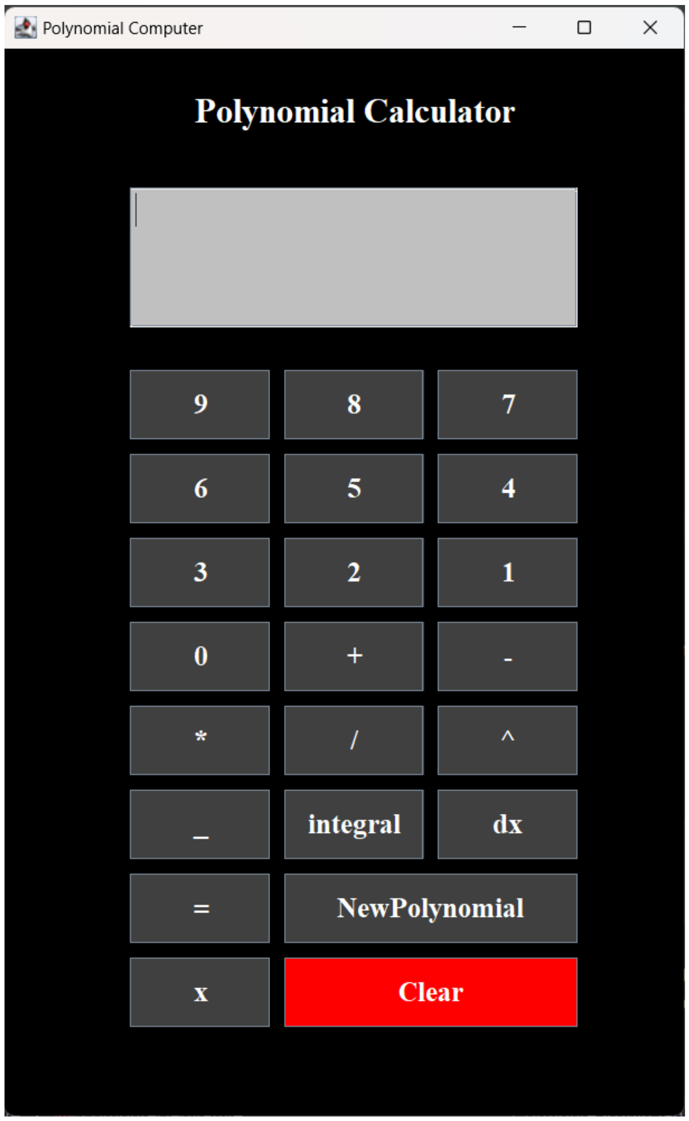
- *Initializes the GUI components such as the title, text pane, and buttons.*

Methods:

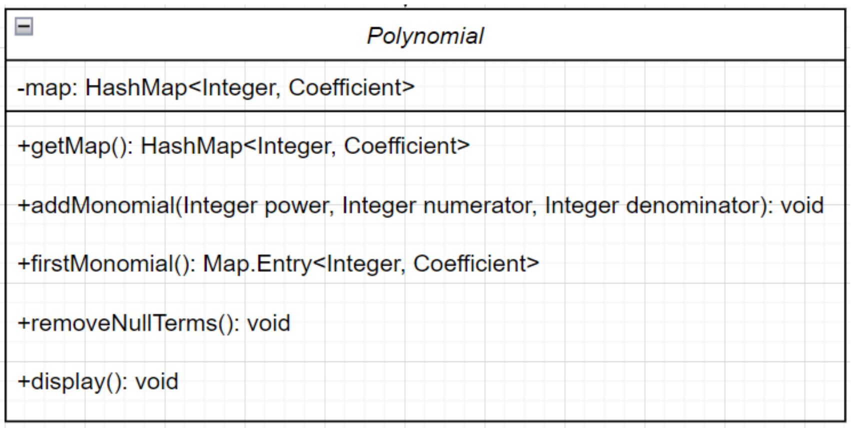
- *initTitle(): Initializes the title label.*
- *setLabelError(String text): Sets and displays error messages.*
- *setPane(): Configures the text pane for displaying input and results.*
- *setButtons(): Sets up the buttons for user interaction.*
- *actionPerformed(ActionEvent e): Handles actions performed by the user, such as button clicks.*
- *buttonNumber_onAction(int digit): Handles actions when a number button is clicked.*
- *buttonSymbol_onAction(String symbol): Handles actions when a symbol button is clicked.*

- *saveMonomial(Integer currentNumber, Integer currentPower): Saves a monomial based on user input.*
- *appendText(String text, boolean isSuperscript): Appends text to the text pane, allowing for superscript styling.*
- *executeOperation(): Executes polynomial operations based on user input.*
- *clearFields(): Clears all input fields and resets the application state.*

Main Method:
- *Initializes the application and makes it visible.*



b) *Implementation of the classes from Data Models package*


Polynomial
-map: HashMap<Integer, Coefficient>
+getMap(): HashMap<Integer, Coefficient>
+addMonomial(Integer power, Integer numerator, Integer denominator): void
+firstMonomial(): Map.Entry<Integer, Coefficient>
+removeNullTerms(): void
+display(): void

Fields:

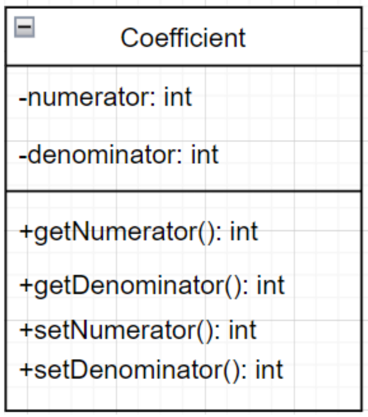
- *'private HashMap<Integer, Coefficient> map: A HashMap to store the coefficients of the polynomial, where the key represents the power of the term and the value is a Coefficient object representing the numerator and denominator.*

Constructor:

- *public Polynomial(): Initializes the polynomial by creating a new HashMap to store coefficients.*

Methods:

- *public HashMap<Integer, Coefficient> getMap(): Getter method to retrieve the HashMap representing the polynomial.*
- *public void addMonomial(Integer power, Integer numerator, Integer denominator): Adds a monomial to the polynomial. If a monomial with the same power already exists, the coefficients are added.*
- *public Map.Entry<Integer, Coefficient> firstMonomial(): Finds and returns the monomial with the greatest power from the polynomial.*
- *public void removeNullTerms(): Removes monomials with zero coefficients from the polynomial.*
- *public void display(): Displays the polynomial by appending its terms to the text pane in the GUI. It formats the terms with appropriate signs and displays them in descending order of power.*



Fields:

- *private int numerator: Represents the numerator of the coefficient.*
- *private int denominator: Represents the denominator of the coefficient.*

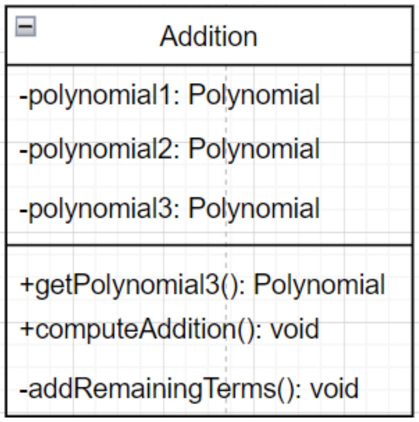
Constructor:

- *public Coefficient(int numerator, int denominator): Initializes a coefficient with the given numerator and denominator.*

Methods:

- *public int getNumerator(): Returns the numerator of the coefficient.*
- *public int getDenominator(): Returns the denominator of the coefficient.*
- *public void setNumerator(int numerator): Sets the numerator of the coefficient.*
- *public void setDenominator(int denominator): Sets the denominator of the coefficient.*

c) Implementation of the classes from the Business Logic package



Fields:

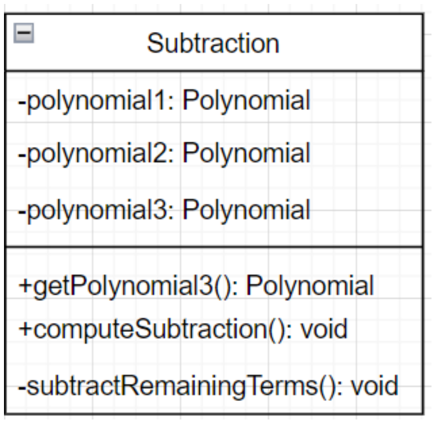
- *private Polynomial polynomial1: Represents the first polynomial.*
- *private Polynomial polynomial2: Represents the second polynomial.*
- *private Polynomial polynomial3: Represents the resulting polynomial after addition.*

Constructors:

- *public Addition(Polynomial polynomial1, Polynomial polynomial2): Initializes the Addition object with two polynomials provided as parameters. It also initializes the resulting polynomial polynomial3.*

Methods:

- *public Polynomial getPolynomial3(): Getter method to retrieve the resulting polynomial after addition.*
- *public void computeAddition(): Computes the addition of the two polynomials. It iterates through the terms of the first polynomial (polynomial1) and checks if the second polynomial (polynomial2) contains a term with the same power. If it does, it adds the coefficients appropriately, considering whether they have the same denominator. If there are no matching terms, it simply adds the term from polynomial1 to the resulting polynomial polynomial3.*
- *private void addRemainingTerms(): After adding terms from polynomial1, this method adds any remaining terms from polynomial2 to polynomial3. This ensures all terms from both polynomials are considered in the addition process.*


Subtraction
-polynomial1: Polynomial
-polynomial2: Polynomial
-polynomial3: Polynomial
+getPolynomial3(): Polynomial
+computeSubtraction(): void
-subtractRemainingTerms(): void

Fields:

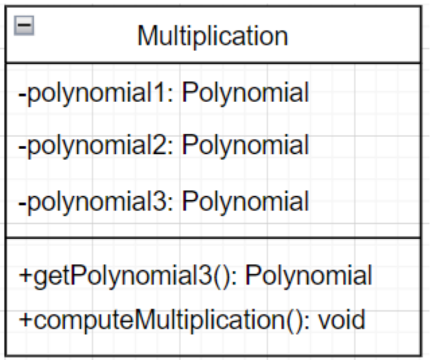
- *private Polynomial polynomial1: Represents the minuend polynomial (the polynomial from which the other polynomial will be subtracted).*
- *private Polynomial polynomial2: Represents the subtrahend polynomial (the polynomial that will be subtracted from the minuend).*
- *private Polynomial polynomial3: Represents the resulting polynomial after subtraction.*

Constructors:

- *public Subtraction(Polynomial polynomial1, Polynomial polynomial2): Initializes the Subtraction object with two polynomials provided as parameters. It also initializes the resulting polynomial polynomial3.*

Methods:

- *public Polynomial getPolynomial3(): Getter method to retrieve the resulting polynomial after subtraction.*
- *public void computeSubtraction(): Computes the subtraction of the second polynomial from the first polynomial. It iterates through the terms of the first polynomial (polynomial1) and checks if the second polynomial (polynomial2) contains a term with the same power. If it does, it subtracts the coefficients appropriately, considering whether they have the same denominator. If there are no matching terms, it simply adds the term from polynomial1 to the resulting polynomial polynomial3. After that, it subtracts remaining terms from polynomial2 by negating their coefficients and adding them to polynomial3.*
- *private void subtractRemainingTerms(): After subtracting terms from polynomial1, this method subtracts any remaining terms from polynomial2 by negating their coefficients and adding them to polynomial3. This ensures all terms from both polynomials are considered in the subtraction process.*



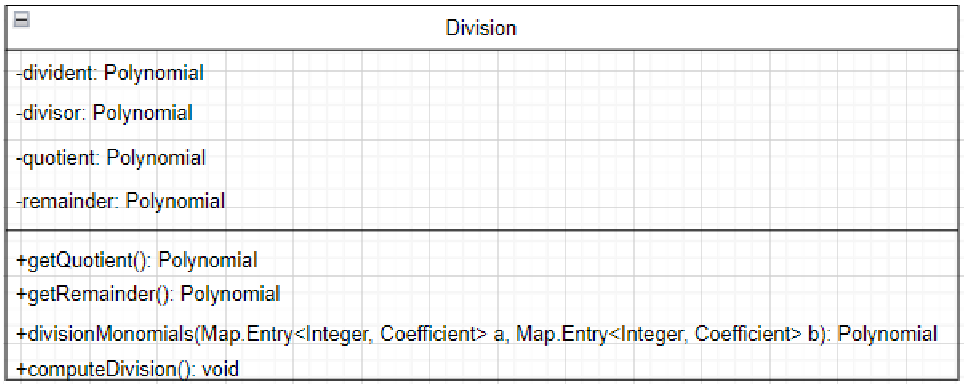
Fields:

- *private Polynomial polynomial1: Represents the first polynomial.*
- *private Polynomial polynomial2: Represents the second polynomial.*

- *private Polynomial polynomial3: Represents the resulting polynomial after multiplication.*

Constructors:
- *public Multiplication(Polynomial polynomial1, Polynomial polynomial2): Initializes the Multiplication object with two polynomials provided as parameters. It also initializes the resulting polynomial polynomial3.*

Methods:
- *public Polynomial getPolynomial3(): Getter method to retrieve the resulting polynomial after multiplication.*
- *public void computeMultiplication(): Computes the multiplication of the two polynomials. It iterates through the terms of both polynomials (polynomial1 and polynomial2) and multiplies each term from polynomial1 with each term from polynomial2. It then adds the resulting powers and multiplies the numerators and denominators of the coefficients. If the resulting power already exists in polynomial3, it adds the terms directly instead of creating a linked list and then summing them.*


Division

-divident: Polynomial
-divisor: Polynomial
-quotient: Polynomial
-remainder: Polynomial

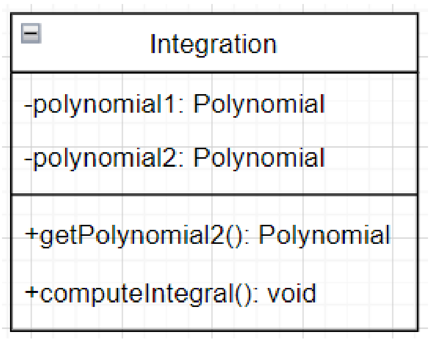
+getQuotient(): Polynomial
+getRemainder(): Polynomial
+divisionMonomials(Map.Entry<Integer, Coefficient> a, Map.Entry<Integer, Coefficient> b): Polynomial
+computeDivision(): void

Fields:
- *private Polynomial dividend: Represents the dividend polynomial (the polynomial to be divided).*
- *private Polynomial divisor: Represents the divisor polynomial (the polynomial by which the dividend will be divided).*
- *private Polynomial quotient: Represents the quotient polynomial (the result of polynomial division).*
- *private Polynomial remainder: Represents the remainder polynomial (the remainder after polynomial division).*

Constructors:
- *public Division(Polynomial polynomial1, Polynomial polynomial2): Initializes the Division object with two polynomials provided as parameters. It also initializes the quotient and remainder polynomials.*

Methods:
- *public Polynomial getQuotient(): Getter method to retrieve the quotient polynomial after division.*
- *public Polynomial getRemainder(): Getter method to retrieve the remainder polynomial after division.*
- *public Polynomial divisionMonomials(Map.Entry<Integer, Coefficient> a, Map.Entry<Integer, Coefficient> b): Computes division between two monomials. It multiplies the numerator of the first monomial by the denominator of the second monomial and vice versa to get the numerator and denominator of the result, then adds the result to the quotient polynomial.*
- *public void computeDivision(): Computes the division of the dividend polynomial by the divisor polynomial. It iterates through the terms of the dividend polynomial and performs long division by*

iterratively subtracting the product of the divisor and a term from the dividend polynomial. The result is stored in the quotient polynomial, and the remainder is stored in the remainder polynomial.



Fields:

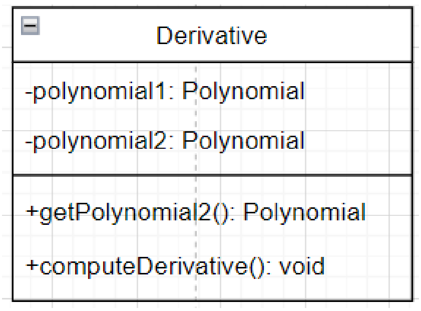
- *private Polynomial polynomial1: Represents the polynomial for which the integral is computed.*
- *private Polynomial polynomial2: Represents the resulting polynomial after integration.*

Constructors:

- *public Integral(Polynomial polynomial1): Initializes the Integral object with the polynomial for which the integral will be computed. It also initializes the resulting polynomial.*

Methods:

- *public Polynomial getPolynomial2(): Getter method to retrieve the resulting polynomial after integration.*
- *public void computeIntegral(): Computes the integral of the given polynomial. It iterates through the terms of the polynomial and applies the power rule of integration. For each term, it increments the power by 1, and adjusts the coefficient accordingly to represent the integral of that term. The resulting terms are added to the resulting polynomial.*



Fields:

- *private Polynomial polynomial1: Represents the polynomial for which the derivative is computed.*
- *private Polynomial polynomial2: Represents the resulting polynomial after differentiation.*

Constructors:

- *public Derivative(Polynomial polynomial1): Initializes the Derivative object with the polynomial for which the derivative will be computed. It also initializes the resulting polynomial.*

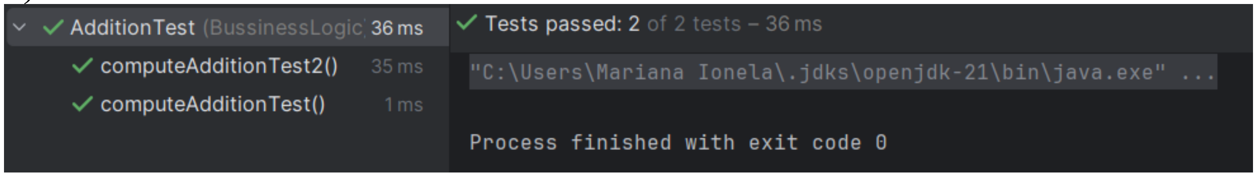
Methods:

- *public Polynomial getPolynomial2(): Getter method to retrieve the resulting polynomial after differentiation.*

- *public void computeDerivative(): Computes the derivative of the given polynomial. It iterates through the terms of the polynomial and applies the power rule of differentiation. For each term, it multiplies the coefficient by the power, decrements the power by 1, and adds the resulting term to the resulting polynomial. If the power becomes zero after differentiation, it adds a constant term representing the derivative of a constant.*

5. Results

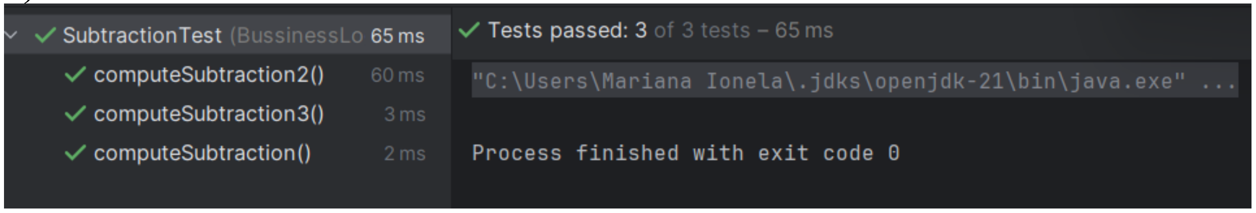
a) Addition



```
✓ AdditionTest (BussinessLogic) 36 ms     ✓ Tests passed: 2 of 2 tests – 36 ms
    ✓ computeAdditionTest2()   35 ms      "C:\Users\Mariana Ionela\.jdks\openjdk-21\bin\java.exe" ...
    ✓ computeAdditionTest()     1 ms
                                          Process finished with exit code 0
```

Test scenario : polynomials with negative coefficients are tested.
Test scenario: polynomials with positive coefficients are tested.

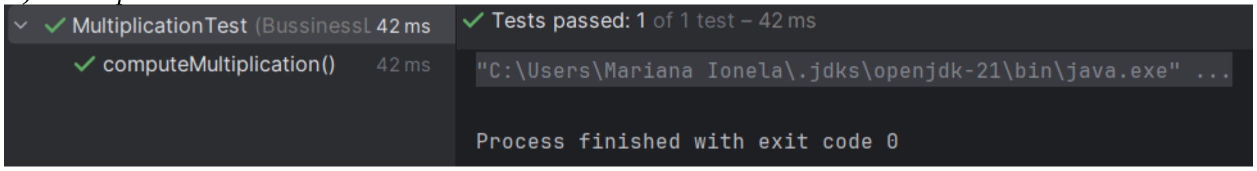
b) Subtraction



```
✓ SubtractionTest (BussinessLo 65 ms     ✓ Tests passed: 3 of 3 tests – 65 ms
    ✓ computeSubtraction2()   60 ms      "C:\Users\Mariana Ionela\.jdks\openjdk-21\bin\java.exe" ...
    ✓ computeSubtraction3()    3 ms
    ✓ computeSubtraction()     2 ms      Process finished with exit code 0
```

Test scenario : polynomials with positive coefficients are tested.
Test scenario: polynomials with negative coefficients are tested.
Test scenario: polynomials with different powers are tested.

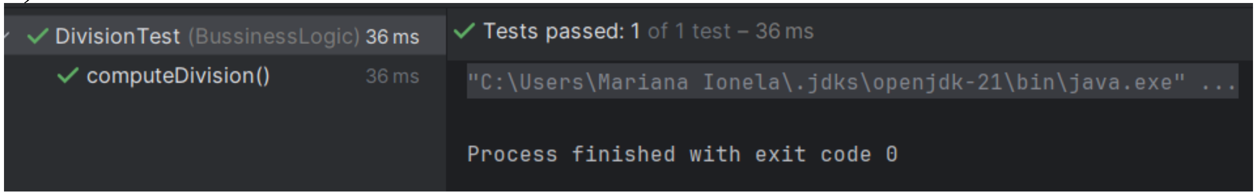
c) Multiplication



```
✓ MultiplicationTest (BussinessL 42 ms     ✓ Tests passed: 1 of 1 test – 42 ms
    ✓ computeMultiplication()   42 ms      "C:\Users\Mariana Ionela\.jdks\openjdk-21\bin\java.exe" ...

                                           Process finished with exit code 0
```

Test scenario : polynomials with positive coefficients are tested.

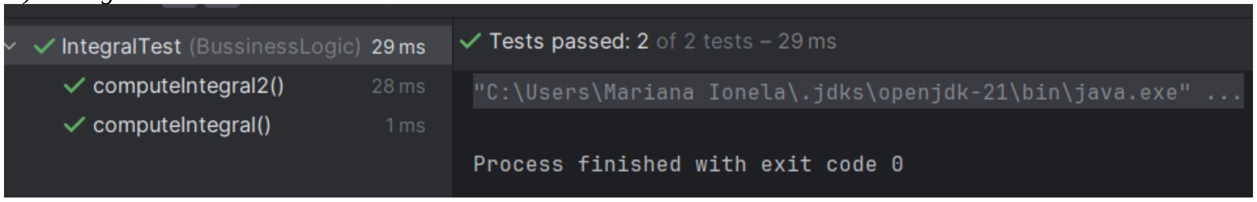
d) Division



```
✓ DivisionTest (BussinessLogic) 36 ms     ✓ Tests passed: 1 of 1 test – 36 ms
    ✓ computeDivision()         36 ms     "C:\Users\Mariana Ionela\.jdks\openjdk-21\bin\java.exe" ...

                                          Process finished with exit code 0
```

Test scenario : polynomials with positive coefficients are tested.

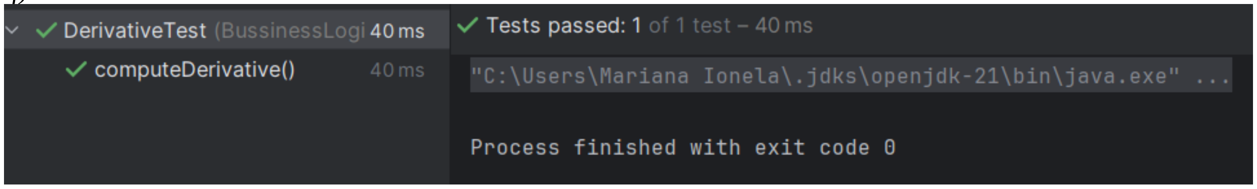
e) Integral



```
✓ IntegralTest (BussinessLogic) 29 ms    ✓ Tests passed: 2 of 2 tests – 29 ms
    ✓ computeIntegral2()        28 ms    "C:\Users\Mariana Ionela\.jdks\openjdk-21\bin\java.exe" ...
    ✓ computeIntegral()          1 ms
                                         Process finished with exit code 0
```

Test scenario : polynomials with positive coefficients are tested.
Test scenario: polynomials with power 0 are tested.

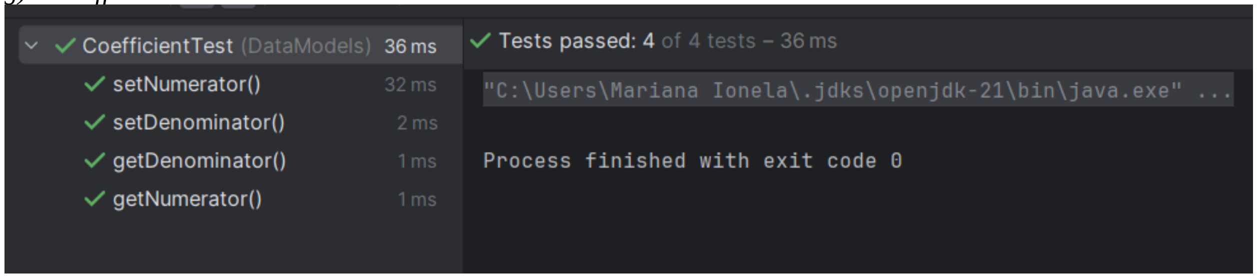
f) Derivative



```
✓ DerivativeTest (BussinessLogi 40 ms    ✓ Tests passed: 1 of 1 test – 40 ms
    ✓ computeDerivative()       40 ms    "C:\Users\Mariana Ionela\.jdks\openjdk-21\bin\java.exe" ...

                                         Process finished with exit code 0
```

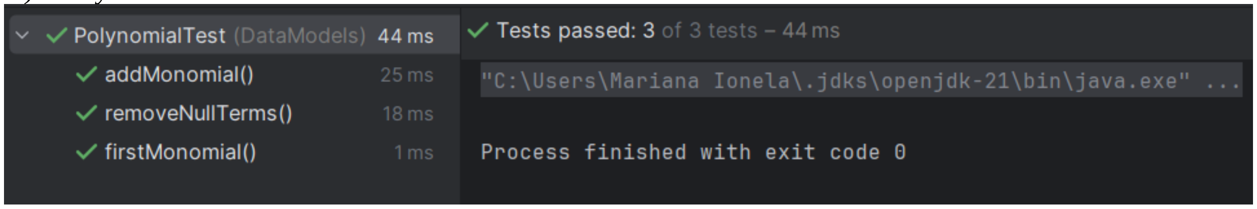
Test scenario : polynomials with positive coefficients are tested.

g) Coefficient



```
✓ CoefficientTest (DataModels) 36 ms    ✓ Tests passed: 4 of 4 tests – 36 ms
    ✓ setNumerator()           32 ms    "C:\Users\Mariana Ionela\.jdks\openjdk-21\bin\java.exe" ...
    ✓ setDenominator()          2 ms
    ✓ getDenominator()          1 ms    Process finished with exit code 0
    ✓ getNumerator()            1 ms
```

h) Polynomial



```
✓ PolynomialTest (DataModels) 44 ms    ✓ Tests passed: 3 of 3 tests – 44 ms
    ✓ addMonomial()           25 ms    "C:\Users\Mariana Ionela\.jdks\openjdk-21\bin\java.exe" ...
    ✓ removeNullTerms()       18 ms
    ✓ firstMonomial()          1 ms    Process finished with exit code 0
```

6. Conclusions

Some of the lessons learned by doing the project are:

- *Enhanced skills in error handling and implementing checks for edge cases such as division by zero, invalid polynomial syntax, etc.*
- *Object-Oriented Design: Gained insights into object-oriented design principles by encapsulating functionality within classes, promoting code reusability, and maintaining modularity.*
- *GUI Development: Developed proficiency in creating graphical user interfaces (GUI) using Java Swing for displaying polynomial operations and results.*

Future developments:

- *The possibility to introduce coefficients that are non-integers.*
- *The possibility to modify dynamically the input.*
- *The calculator should make simplifications before displaying the final result*
- *The calculator should display all the mathematical steps required to solve the problem.*

7. Bibliography

1. *What are the conventions for UML Diagram?*
   file:///C:/Users/Mariana%20Ionela/AppData/Local/Temp/8fb6fe23-4c75-4b7e-bb9a-6e0ffbe68791_Lecture_2_UML_Diagrams.zip.791/Lecture_2_UML_Diagrams.pdf

2. *What are the conventions for UML Packages Diagram?*
   https://dsrl.eu/courses/pt/materials/PT_2024_A1_S1.pdf

3. *What is the Use Cases diagram?*
   file:///C:/Users/Mariana%20Ionela/AppData/Local/Temp/8fb6fe23-4c75-4b7e-bb9a-6e0ffbe68791_Lecture_2_UML_Diagrams.zip.791/Lecture_2_UML_Diagrams.pdf