DOCUMENTATION

ASSIGNMENT 1

STUDENT NAME: Boboș Răzvan-Andrei
GROUP: 30425

CONTENTS

1. Assignment Objective

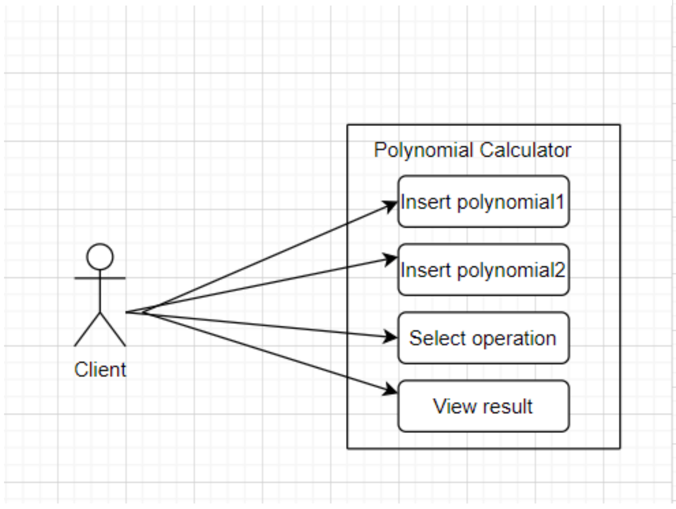
Main objective

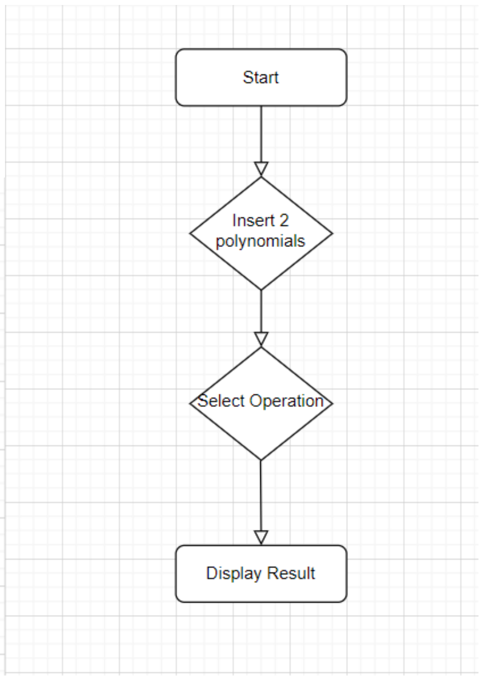
Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation to be performed and view the result.

Sub-objectives

1. Analyze the problem and identify requirements
2. Design the polynomial calculator
3. Implement the polynomial calculator
4. Test the polynomial calculator

2. Problem Analysis, Modeling, Scenarios, Use Cases





Functional requirements:

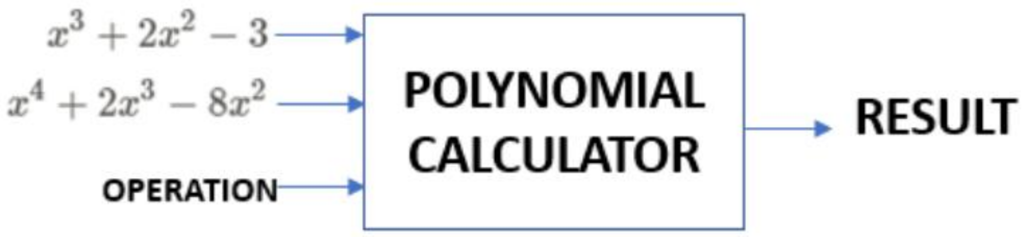
- The polynomial calculator should allow users to insert polynomials
- The polynomial calculator should allow users to select the mathematical operation
- The polynomial calculator should add two polynomials

Non-Functional requirements:

- The polynomial calculator should be intuitive and easy to use by the user
- The polynomial should have a good-looking interface

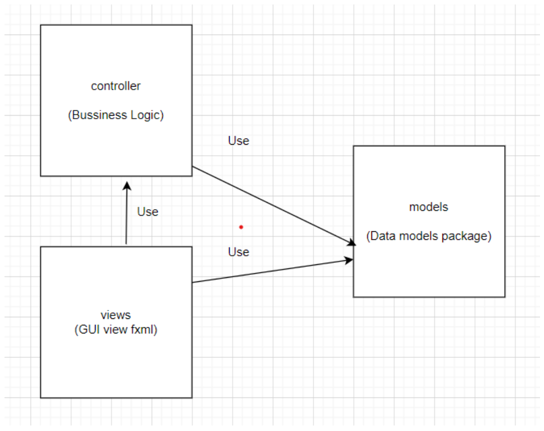
3. Design

Overall System Design



Division into sub-systems/packages

The calculator project is divided into 3 major packages providing different functionalities.



Division into classes

To design the object-oriented architecture of the polynomial calculator application, we can break down the functionality into several classes representing different components of the system. Here's a conceptual overview of the OOP design:

Monomial Class

Represents a single term in a polynomial, consisting of a coefficient and an exponent.
It consists helpful methods used in Operations class(eg. Add, which adds monomial single values, method helpful in Operations.java -> add)

Polynomial Class

Represents a polynomial, consisting of a collection of monomials(Map).

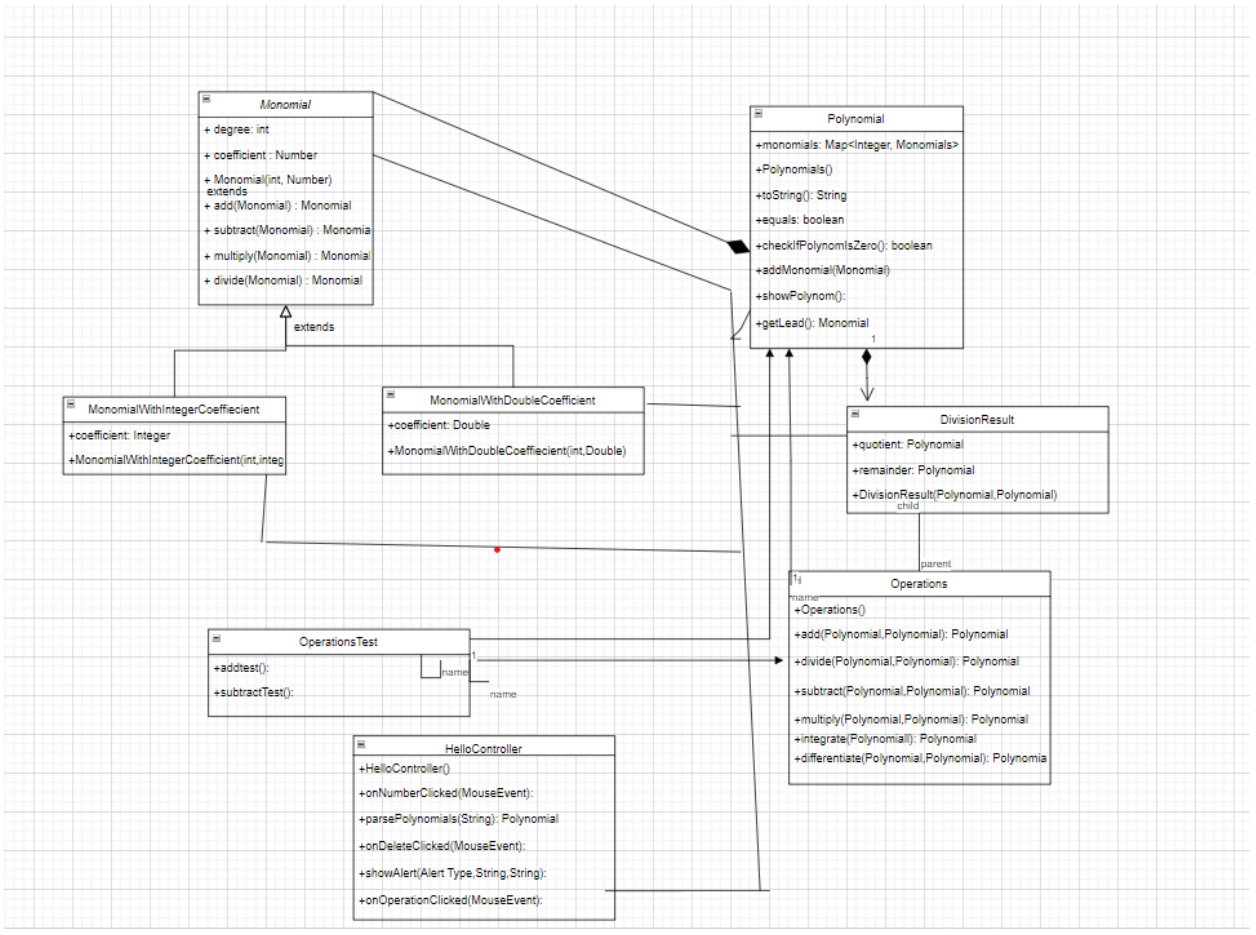
It implements helpful methods used in Opeartions class and also for handling the user input and displaying the polynomial result.

Operations Class
Represents the methods for making calculations on polynomials.(eg. Add, multiply, differentiate etc.)

HelloController class
Main class responsible for coordinating interactions between the user interface and the polynomial manipulation logic

OperationsTest class
Class implemented using jUnit, which tests if the Operations.java class methods provide a correct implementation.

Here's how these classes might relate to each other in a basic UML class diagram:


Monomial
+ degree: int
+ coefficient : Number
+ Monomial(int, Number)
extends
+ add(Monomial) : Monomial
+ subtract(Monomial) : Monomia
+ multiply(Monomial) : Monomial
+ divide(Monomial) : Monomial
extends
MonomialWithIntegerCoeffiecient
+coefficient: Integer
+MonomialWithIntegerCoefficient(int,integ
MonomialWithDoubleCoefficient
+coefficient: Double
+MonomialWithDoubleCoeffiecient(int,Double)
Polynomial
+monomials: Map<Integer, Monomials>
+Polynomials()
+toString(): String
+equals: boolean
+checkIfPolynomIsZero(): boolean
+addMonomial(Monomial)
+showPolynom():
+getLead(): Monomial
1
DivisionResult
+quotient: Polynomial
+remainder: Polynomial
+DivisionResult(Polynomial,Polynomial)
child
parent
Operations
1
name
+Operations()
+add(Polynomial,Polynomial): Polynomial
+divide(Polynomial,Polynomial): Polynomial
+subtract(Polynomial,Polynomial): Polynomial
+multiply(Polynomial,Polynomial): Polynomial
+integrate(Polynomiall): Polynomial
+differentiate(Polynomial,Polynomial): Polynomia
OperationsTest
+addtest():
+subtractTest():
name
1
name
HelloController
+HelloController()
+onNumberClicked(MouseEvent):
+parsePolynomials(String): Polynomial
+onDeleteClicked(MouseEvent):
+showAlert(Alert Type,String,String):
+onOperationClicked(MouseEvent):

Used Data Structures and Defined Interfaces

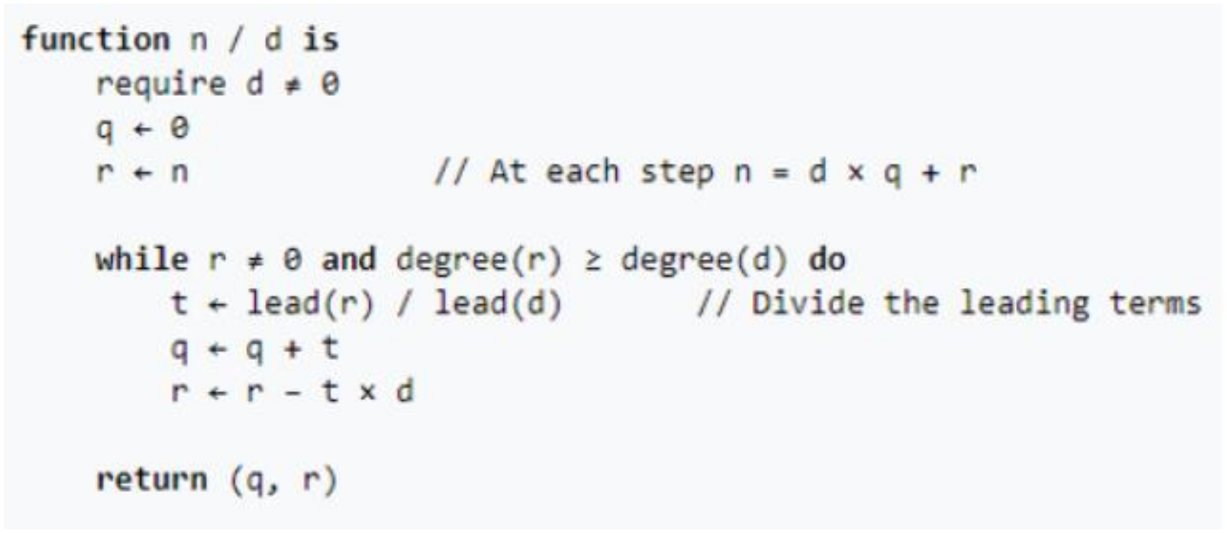
- A **Map** is utilized to efficiently store monomials, associating each monomial's degree with its corresponding coefficient, enabling quick access and manipulation of polynomial terms based on their degrees. Additionally, **Map** facilitates coefficient lookup, allowing seamless retrieval of coefficients using their respective degrees within the polynomial structure.

- The **Number** class is employed to represent coefficients of monomials, offering flexibility to handle both integer and floating-point coefficients within the polynomial structure. This abstraction simplifies coefficient management and computation, accommodating various numeric types seamlessly throughout polynomial operations.

- The **DivisionResult** class is tailored for division operations, encapsulating both the quotient and remainder components of polynomial division. This specialized class streamlines division computations, ensuring accurate handling and representation of division results within polynomial operations, enhancing clarity and precision in division-related functionalities.

Notable Used Algorithms

The division algorithm is implemented to perform polynomial division, generating quotient and remainder polynomials through systematic evaluation of polynomial terms. This algorithmic process facilitates efficient division operations, enabling precise decomposition of polynomials and yielding meaningful **DivisionResult** instances encapsulating both quotient and remainder components for further analysis or processing within the polynomial framework.



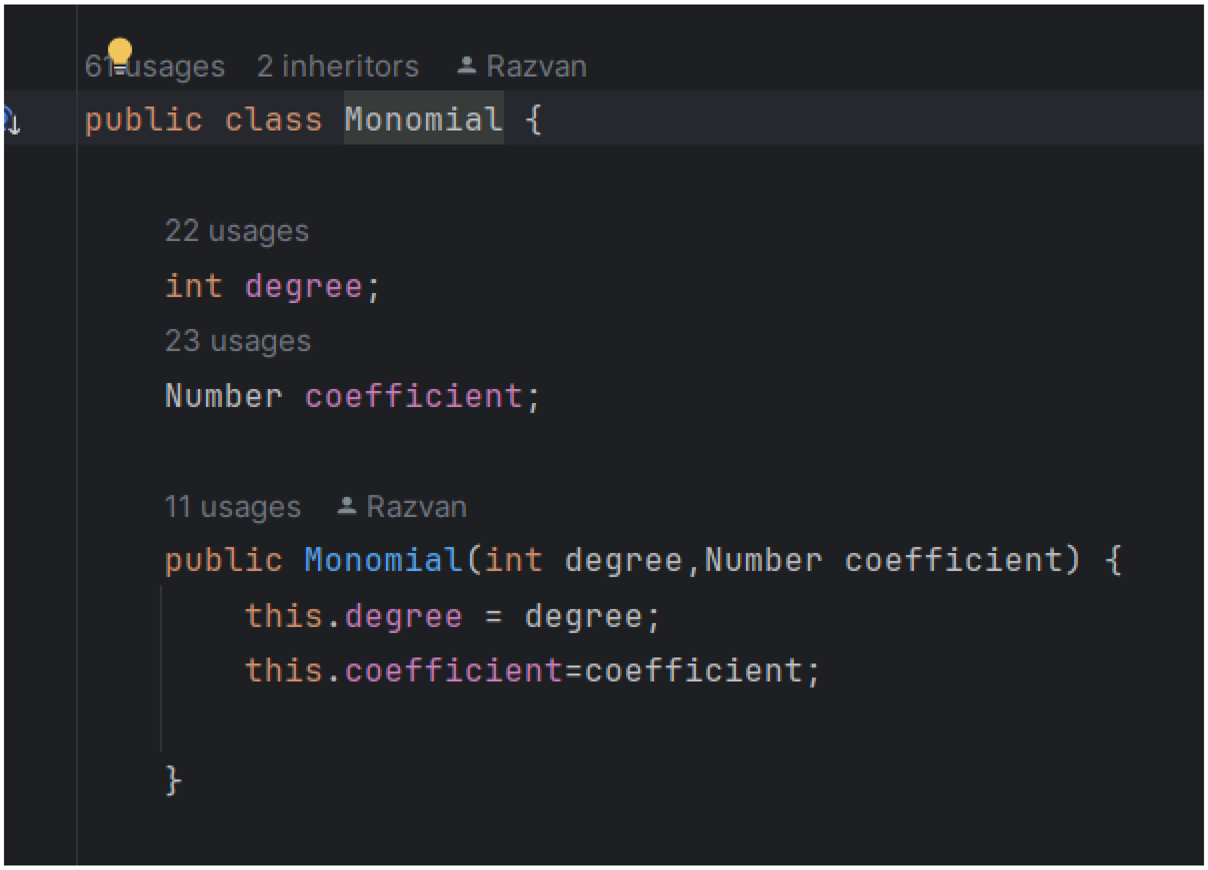
```
function n / d is
    require d ≠ 0
    q ← 0
    r ← n               // At each step n = d × q + r

    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

To comprehend the implementation thoroughly, we will delve into each class's implementation in detail, examining how they contribute to the overall functionality of the project.

Monomial.java

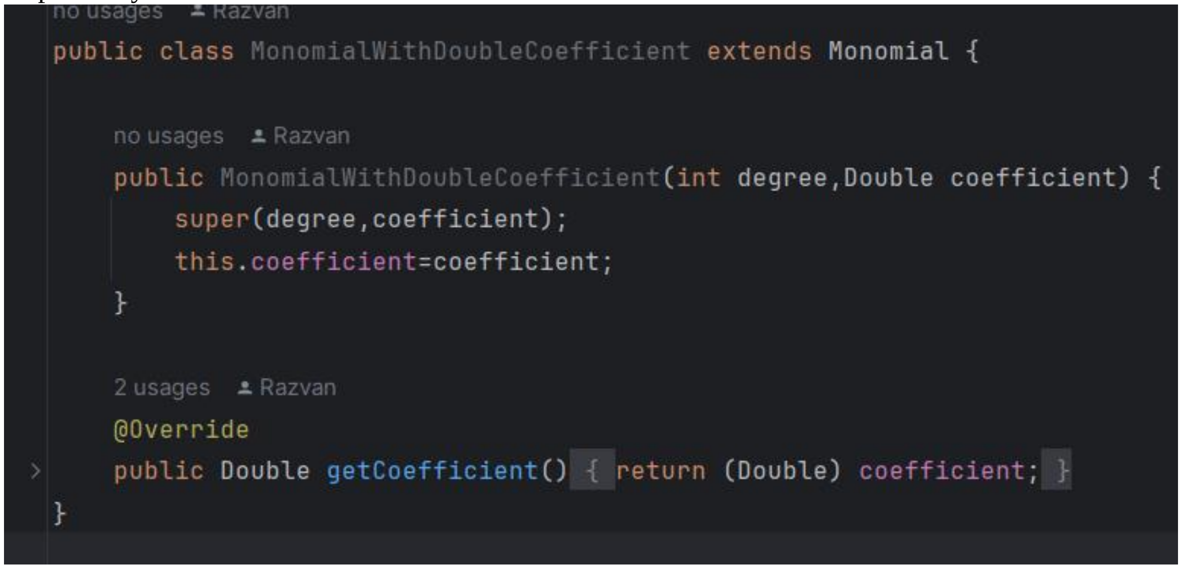
The **Monomial.java** class serves as the foundational building block of the polynomial project, encapsulating the essential characteristics of a monomial within a polynomial expression. By defining attributes such as the degree and coefficient, this class establishes the fundamental representation of individual terms within polynomials



```
61 usages   2 inheritors   Razvan
public class Monomial {

    22 usages
    int degree;
    23 usages
    Number coefficient;

    11 usages   Razvan
    public Monomial(int degree,Number coefficient) {
        this.degree = degree;
        this.coefficient=coefficient;

    }
```

MonomialWithDoubleCoefficient.java/MonomialWithIntegerCoefficient.java

The **MonomialWithDoubleCoefficient.java** and **MonomialWithIntegerCoefficient**.java classes extend the Monomial.java class, specializing in monomials with double and integer coefficients, respectively.



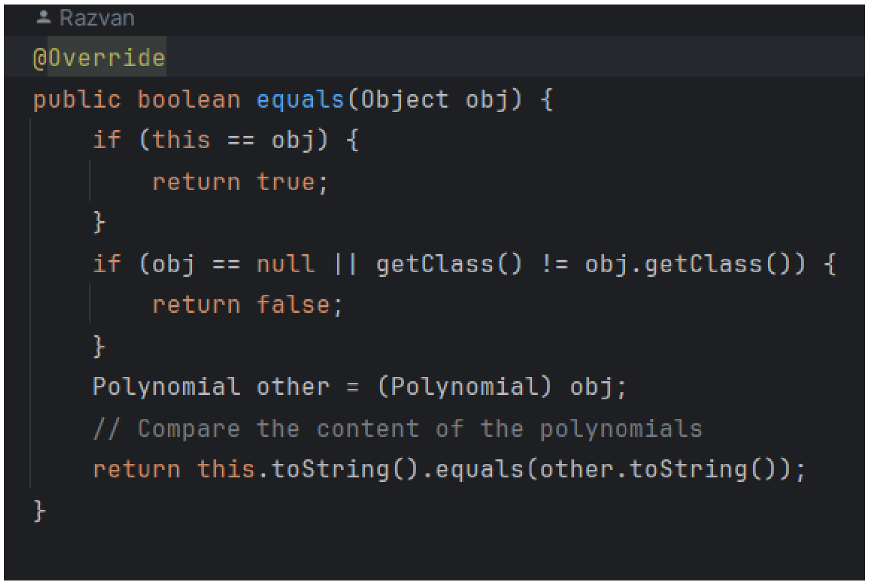
```
no usages   Razvan
public class MonomialWithDoubleCoefficient extends Monomial {

    no usages   Razvan
    public MonomialWithDoubleCoefficient(int degree,Double coefficient) {
        super(degree,coefficient);
        this.coefficient=coefficient;
    }

    2 usages   Razvan
    @Override
    public Double getCoefficient() { return (Double) coefficient; }
}
```

Polynomial.java

The Polynomial.java class encapsulates the functionalities necessary for managing polynomial expressions within the project.

Notable methods:

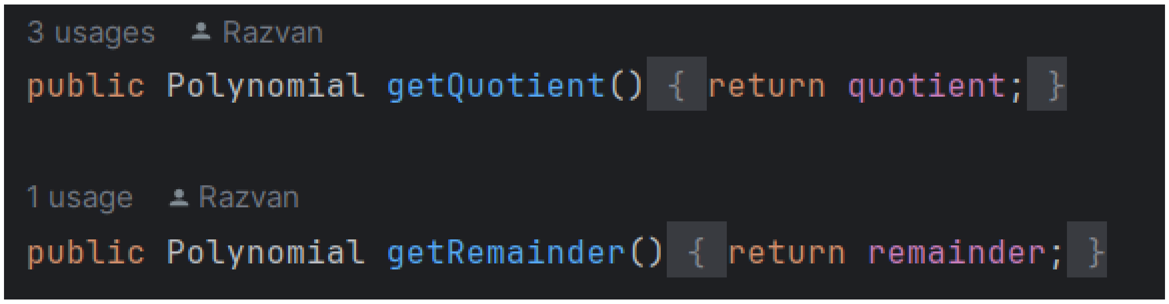
- checkIfPolynomIsZero() verifies if the polynomial is a zero polynomial by examining its monomials and their coefficients.
- The getLead() method retrieves the leading monomial of the polynomial, essential for polynomial operations.
- The toString() method generates a string representation of the polynomial, allowing for easy printing and comparison.
- Finally, the equals() method overrides the default equality comparison, ensuring proper comparison of polynomial objects based on their content rather than reference (used in OperationsTest.java).



```
Razvan
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null || getClass() != obj.getClass()) {
        return false;
    }
    Polynomial other = (Polynomial) obj;
    // Compare the content of the polynomials
    return this.toString().equals(other.toString());
}
```

DivisionResult.java

The **DivisionResult.java** class is pivotal in the polynomial project, serving as a container for the results of polynomial division operations.



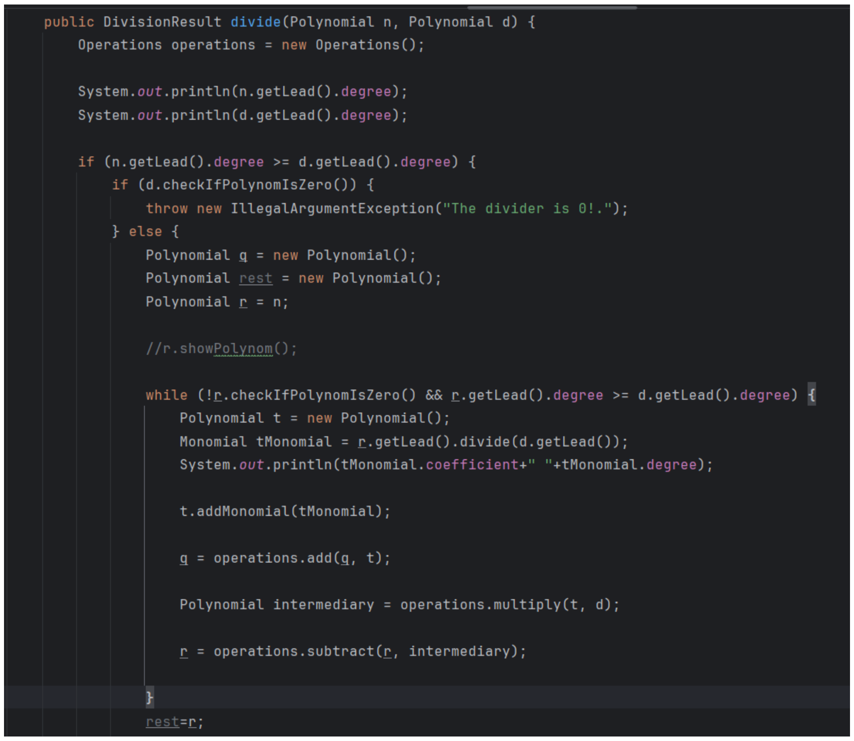
```
3 usages   Razvan
public Polynomial getQuotient() { return quotient; }

1 usage   Razvan
public Polynomial getRemainder() { return remainder; }
```

Operations.java

The **Operations.java** class serves as a central hub for performing various polynomial operations within the project, encapsulating functionalities for polynomial addition, subtraction, multiplication, division, differentiation, and integration.

Notable methods:

- divide(Polynomial,Polynomial): implements polynomial division, leveraging a systematic algorithm to produce both quotient and remainder polynomials.



```java
public DivisionResult divide(Polynomial n, Polynomial d) {
    Operations operations = new Operations();

    System.out.println(n.getLead().degree);
    System.out.println(d.getLead().degree);

    if (n.getLead().degree >= d.getLead().degree) {
        if (d.checkIfPolynomIsZero()) {
            throw new IllegalArgumentException("The divider is 0!.");
        } else {
            Polynomial q = new Polynomial();
            Polynomial rest = new Polynomial();
            Polynomial r = n;

            //r.showPolynom();

            while (!r.checkIfPolynomIsZero() && r.getLead().degree >= d.getLead().degree) {
                Polynomial t = new Polynomial();
                Monomial tMonomial = r.getLead().divide(d.getLead());
                System.out.println(tMonomial.coefficient+" "+tMonomial.degree);

                t.addMonomial(tMonomial);

                q = operations.add(q, t);

                Polynomial intermediary = operations.multiply(t, d);

                r = operations.subtract(r, intermediary);

            }
            rest=r;
```

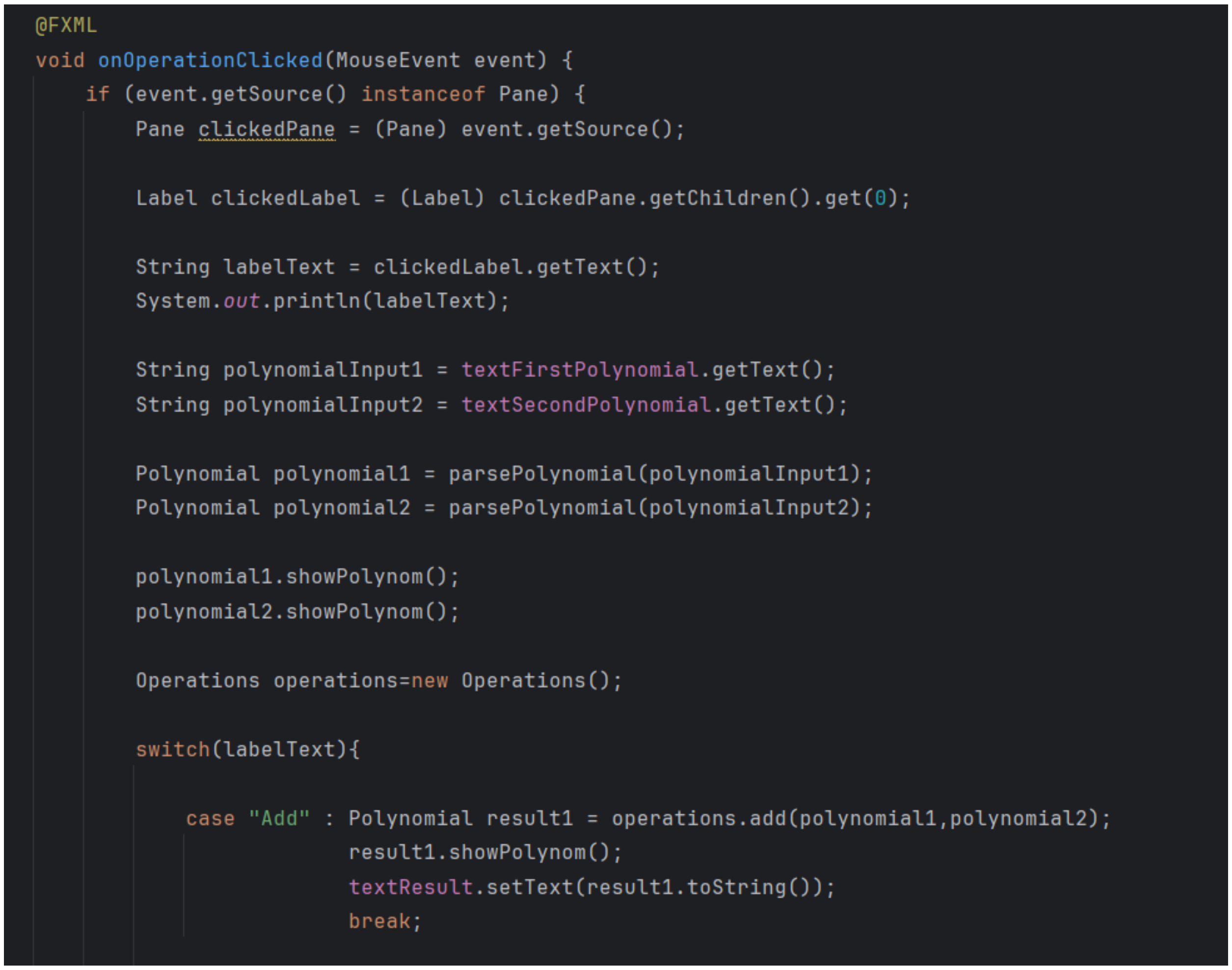
HelloController.java

The HelloController.java class is a vital component of the polynomial project, responsible for orchestrating user interactions and coordinating polynomial operations within the graphical user interface.

Notable methods:

- The **parsePolynomial()** method extracts coefficients and exponents from the input polynomial string, ensuring correct formatting and handling edge cases such as empty inputs or invalid monomial formats.

- The **onOperationClicked()** method responds to user clicks on operation buttons, retrieves input polynomials, invokes corresponding polynomial operations, and updates the result display, enhancing user interaction and facilitating real-time feedback.



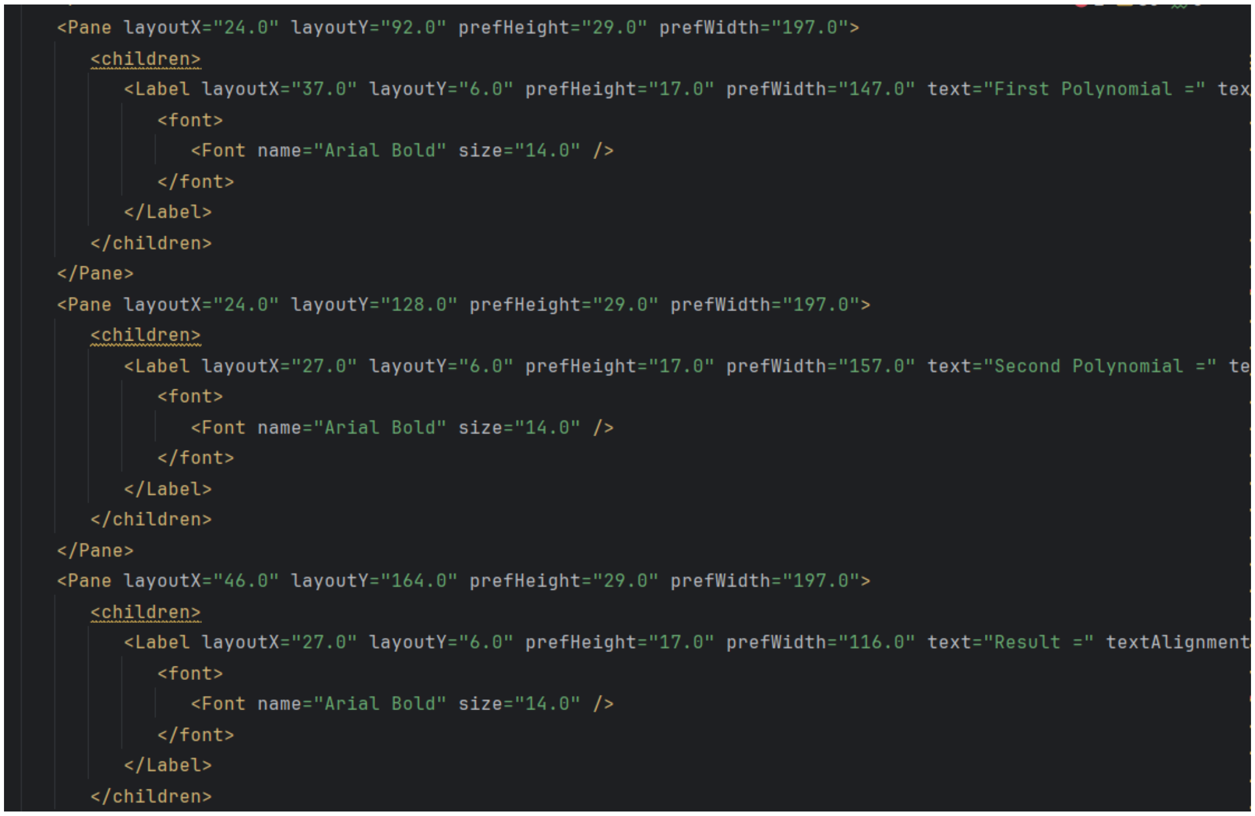
```java
@FXML
void onOperationClicked(MouseEvent event) {
    if (event.getSource() instanceof Pane) {
        Pane clickedPane = (Pane) event.getSource();

        Label clickedLabel = (Label) clickedPane.getChildren().get(0);

        String labelText = clickedLabel.getText();
        System.out.println(labelText);

        String polynomialInput1 = textFirstPolynomial.getText();
        String polynomialInput2 = textSecondPolynomial.getText();

        Polynomial polynomial1 = parsePolynomial(polynomialInput1);
        Polynomial polynomial2 = parsePolynomial(polynomialInput2);

        polynomial1.showPolynom();
        polynomial2.showPolynom();

        Operations operations=new Operations();

        switch(labelText){

            case "Add" : Polynomial result1 = operations.add(polynomial1,polynomial2);
                         result1.showPolynom();
                         textResult.setText(result1.toString());
                         break;
```

HelloApplication.java

The **HelloApplication.java** class initializes the JavaFX application, loads the graphical user interface defined in the hello-view.fxml file, and sets up the scene with a specific size.

hello-view.fxml

hello-view.fxml defines the layout and structure of the graphical user interface (GUI) for the Polynomial Calculator application using JavaFX. It contains various UI components such as labels, text fields, and panes arranged within a VBox layout container.



```
<Pane layoutX="24.0" layoutY="92.0" prefHeight="29.0" prefWidth="197.0">
   <children>
      <Label layoutX="37.0" layoutY="6.0" prefHeight="17.0" prefWidth="147.0" text="First Polynomial =" tex
         <font>
            <Font name="Arial Bold" size="14.0" />
         </font>
      </Label>
   </children>
</Pane>
<Pane layoutX="24.0" layoutY="128.0" prefHeight="29.0" prefWidth="197.0">
   <children>
      <Label layoutX="27.0" layoutY="6.0" prefHeight="17.0" prefWidth="157.0" text="Second Polynomial =" te
         <font>
            <Font name="Arial Bold" size="14.0" />
         </font>
      </Label>
   </children>
</Pane>
<Pane layoutX="46.0" layoutY="164.0" prefHeight="29.0" prefWidth="197.0">
   <children>
      <Label layoutX="27.0" layoutY="6.0" prefHeight="17.0" prefWidth="116.0" text="Result =" textAlignment
         <font>
            <Font name="Arial Bold" size="14.0" />
         </font>
      </Label>
   </children>
```

hello-view.css

CSS file defines the styling for the graphical user interface (GUI) components of the Polynomial Calculator application using JavaFX.

5. Results

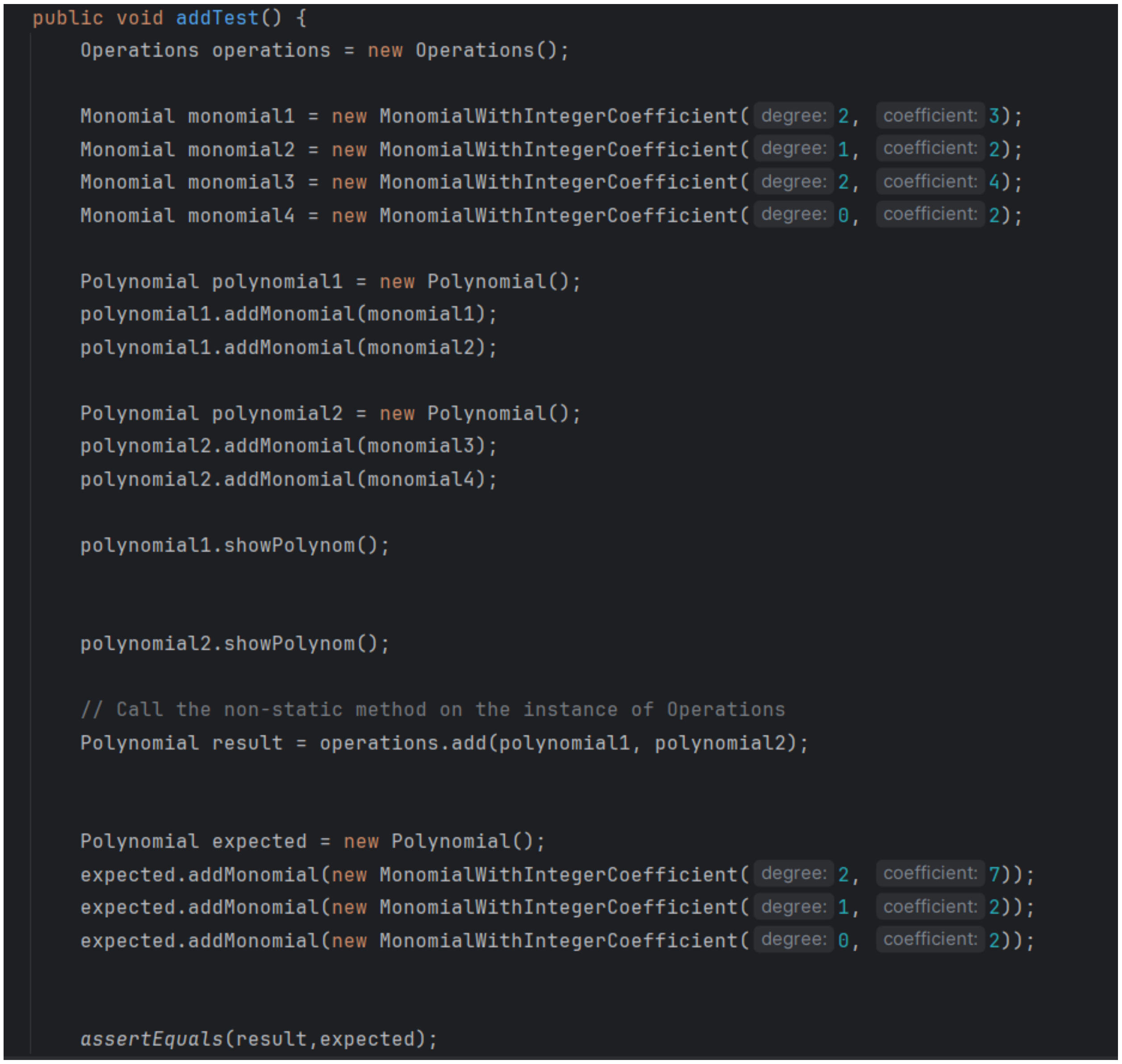
I conducted **two** tests, one focusing on addition and the other on subtraction, using these testing scenarios:

addTest

- In this scenario, two polynomials are created, each containing multiple monomials with integer coefficients.
- The polynomials are structured to represent mathematical expressions.
- The **add** method of the **Operations** class is called with these polynomials as parameters.
- The expected result is a new polynomial obtained by adding corresponding monomials from both input polynomials.
- The test asserts that the result of the addition operation matches the expected polynomial.

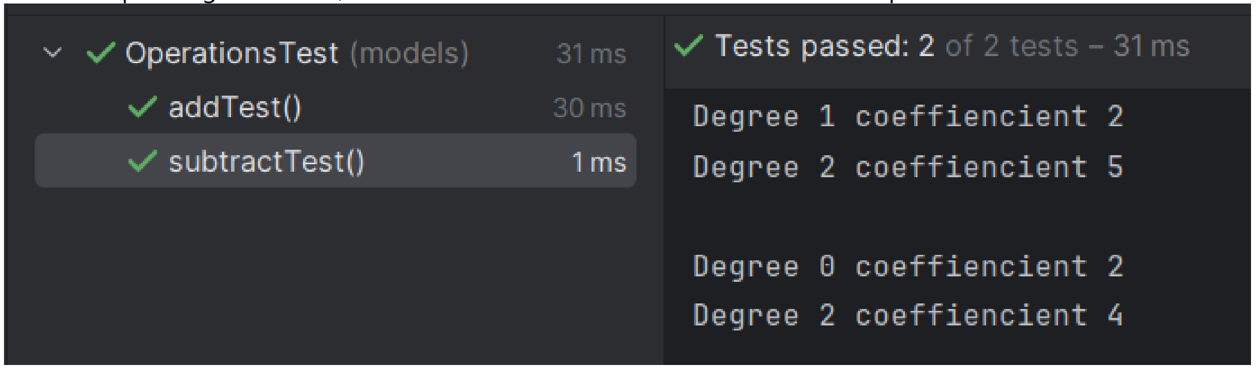
subtractTest

- Similar to the addition scenario, two polynomials are created with multiple monomials, representing mathematical expressions.
- The **subtract** method of the **Operations** class is invoked with these polynomials as arguments.
- The expected outcome is a new polynomial obtained by subtracting corresponding monomials from the first polynomial by those of the second polynomial.
- The test verifies whether the result of the subtraction operation aligns with the expected polynomial.



```java
public void addTest() {
    Operations operations = new Operations();

    Monomial monomial1 = new MonomialWithIntegerCoefficient( degree: 2,  coefficient: 3);
    Monomial monomial2 = new MonomialWithIntegerCoefficient( degree: 1,  coefficient: 2);
    Monomial monomial3 = new MonomialWithIntegerCoefficient( degree: 2,  coefficient: 4);
    Monomial monomial4 = new MonomialWithIntegerCoefficient( degree: 0,  coefficient: 2);

    Polynomial polynomial1 = new Polynomial();
    polynomial1.addMonomial(monomial1);
    polynomial1.addMonomial(monomial2);

    Polynomial polynomial2 = new Polynomial();
    polynomial2.addMonomial(monomial3);
    polynomial2.addMonomial(monomial4);

    polynomial1.showPolynom();


    polynomial2.showPolynom();

    // Call the non-static method on the instance of Operations
    Polynomial result = operations.add(polynomial1, polynomial2);


    Polynomial expected = new Polynomial();
    expected.addMonomial(new MonomialWithIntegerCoefficient( degree: 2,  coefficient: 7));
    expected.addMonomial(new MonomialWithIntegerCoefficient( degree: 1,  coefficient: 2));
    expected.addMonomial(new MonomialWithIntegerCoefficient( degree: 0,  coefficient: 2));


    assertEquals(result,expected);
```

After incorporating JUnit tests, the outcomes for the addition and subtraction operations are as follows:



```
✓ OperationsTest (models)   31 ms
    ✓ addTest()             30 ms
    ✓ subtractTest()         1 ms
```

```
✓ Tests passed: 2 of 2 tests – 31 ms

Degree 1 coeffiencient 2
Degree 2 coeffiencient 5

Degree 0 coeffiencient 2
Degree 2 coeffiencient 4
```

6. Conclusions

Conclusions: After completing this assignment, several conclusions can be drawn. Firstly, the implementation of unit tests using JUnit provided valuable insights into ensuring the correctness and robustness of the codebase. Secondly, the separation of functionality into distinct packages facilitated better code organization and maintainability. Finally, developing a desktop application allowed for a practical application of object-oriented principles and data modeling.

Learnings: Through this assignment, I gained proficiency in several key areas. Firstly, I learned effective strategies for separating code into packages, enhancing code readability and scalability. Additionally, I acquired skills in parsing user input, an essential aspect of developing interactive applications. Moreover, integrating data models within a desktop application framework deepened my understanding of object-oriented design principles and their practical applications.

Future Developments: Looking ahead, there are several avenues for further development and improvement. One potential area is the expansion of supported operations beyond addition and subtraction, enriching the application's functionality. Additionally, enhancing input parsing capabilities to support double values would improve the application's versatility and user-friendliness. Furthermore, refining the graphical user interface (GUI) to provide a more intuitive and visually appealing user experience represents another promising direction for future development efforts.

7. Bibliography

1. Fundamental Programming Techniques: A1_S1, A1_S2, A1_S3
2. OOP – Marius Joldos
3. https://youtu.be/IZCwawKILsk?si=1a1T8uV8iof2JGQE
4. https://www.youtube.com/@Randomcode_0
5. https://stackoverflow.com/
6. https://www.geeksforgeeks.org/