DOCUMENTATION

ASSIGNMENT *1*

STUDENT NAME: DANCI PATRICIA IOANA

GROUP: 30424P

# CONTENTS

[1. Assignment Objective 3](#_Toc128043139)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 3](#_Toc128043140)

[3. Design 5](#_Toc128043141)

[4. Implementation 8](#_Toc128043142)

[5. Results 14](#_Toc128043143)

[6. Conclusions 18](#_Toc128043144)

[7. Bibliography 18](#_Toc128043145)

# Assignment Objective

This assignment has as the **main objective** the design and further implementation of a polynomial calculator displayed through a graphical user interface where the user is able to input polynomials and perform different mathematical operations.

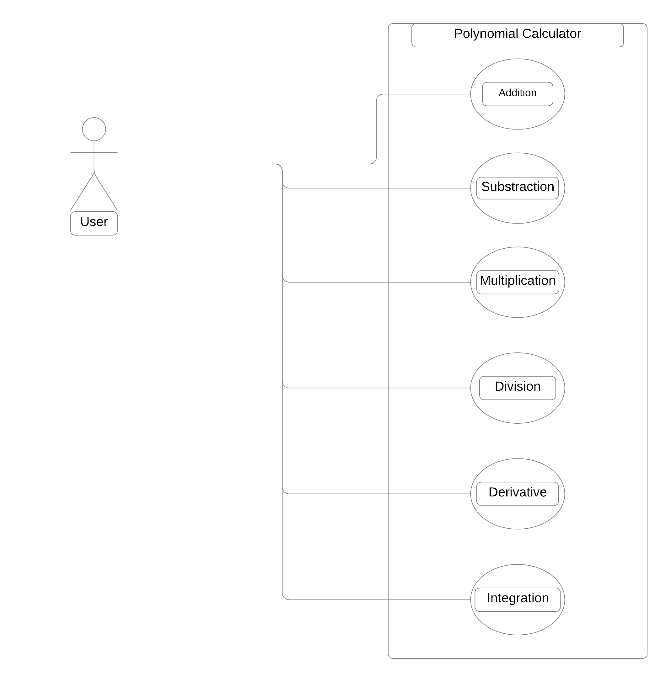
In order to achieve such functionality, the following sub-objectives need to be addressed:

* Analize the requirement
* Design the application itself
* Implement the operations
* Test the accuracy and correctness of the polynomial calculator

# Problem Analysis, Modeling, Scenarios, Use Cases

**Functional requirements:**

1. Allow insertion of first polynomial and second polynomial
2. Check input validity
3. Allow for selection of mathematical operations
4. Addition of two polynomials
5. Subtraction of two polynomials
6. Multiplication of two polynomials
7. Division of two polynomials
8. Derivative of a polynomial
9. Integration of a polynomial

**

Furthermore, based on these functional requirements, I will provide a use case diagram where the actor is being represented by the user, who interacts with the calculator system. The use cases in this diagram are given by the mathematical operations: add, substract, multiply, divide, derivate, integration.

**Non-functional requirements:**

1. The polynomial calculator should be intuitive and easy to use by the user

* Use Case: ADDITION
* Primary Actor: user
* Main Succes Scenario:

1. The user inserts 2 polynomials in the GUI
2. The user selects “ADDITION”
3. The polynomial calculator performs the operation and displays the result

* User Case: SUBTRACTION
* Primary Actor: user
* Main Succes Scenario:

1. The user inserts 2 polynomials in the GUI
2. The user selects “SUBTRACTION”
3. The polynomial calculator performs the operation and displays the result

* User Case: MULTIPLICATION
* Primary Actor: user
* Main Succes Scenario:

1. The user inserts 2 polynomials in the GUI
2. The user selects “MULTIPLICATION”
3. The polynomial calculator performs the operation and displays the result

* User Case: DIVISION
* Primary Actor: user
* Main Succes Scenario:

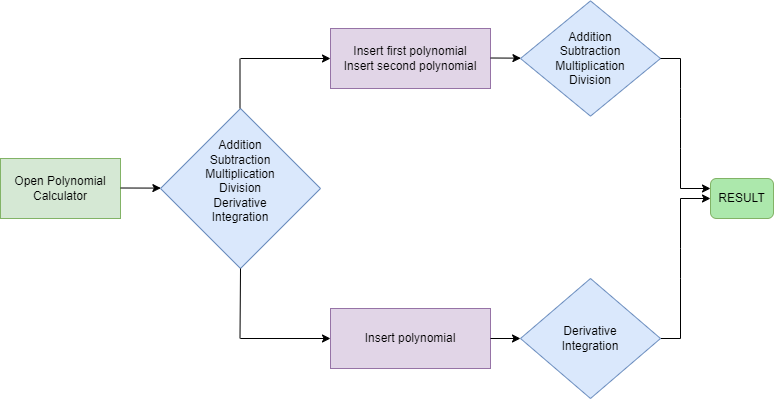
1. The user inserts 2 polynomials in the GUI
2. The user selects “DIVISION”
3. The polynomial calculator performs the operation and displays the result

* User Case: DERIVATIVE
* Primary Actor: user
* Main Succes Scenario:

1. The user inserts a polynomial in the GUI
2. The user selects “DERIVATIVE”
3. The polynomial calculator performs the operation and displays the result

* User Case: INTEGRATION
* Primary Actor: user
* Main Succes Scenario:

1. The user inserts a polynomial in the GUI
2. The user selects “INTEGRATION”
3. The polynomial calculator performs the operation and displays the result



Since derivative/integration operates on only one polynomial, whereas addition, subtraction, multiplication and division operate on two polynomials, the user must first decide which operation is the calculator required to perform. Afterwards, based on the previous decision, the user must insert either p1 and p2, or p and then choose amongst the available mathematical operations. The result will be displayed in a specific text field on the graphical user interface as well.

# Design

The class design of the polynomial calculator comprises four main components: the Interface class, responsible for managing the graphical user interface (GUI) elements such as text fields and buttons for user interaction; the Polynomial class, which encapsulates polynomial data and provides methods for accessing and manipulating polynomial coefficients; the Operations class, implementing algorithms for various polynomial operations like addition, subtraction, multiplication, division, differentiation, and integration; and the OperationTest class, dedicated to unit testing the functionality of the Operations class to ensure the accuracy and reliability of polynomial calculations. Additionally, the Main method serves as the entry point of the program, facilitating the instantiation of GUI components.

**POLYNOMIAL CALCULATOR**

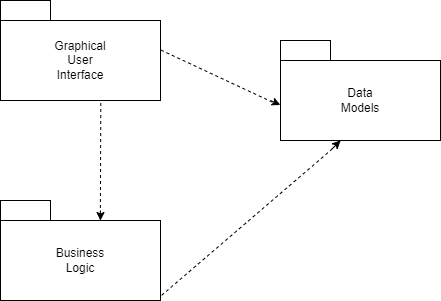
x²+2x+1

RESULT

x3-10 x²+4x+9

OPERATION

*Division into sub-systems:*



The **Graphical User Interface package** contains classes that embody the essence of user interaction. These classes define the interface elements, such as buttons, text fields and text labels, each positioned to enhance the user experience. Moreover, this package seamlessly integrates Action Listeners, dynamically responding to users’ selections of mathematical operations.

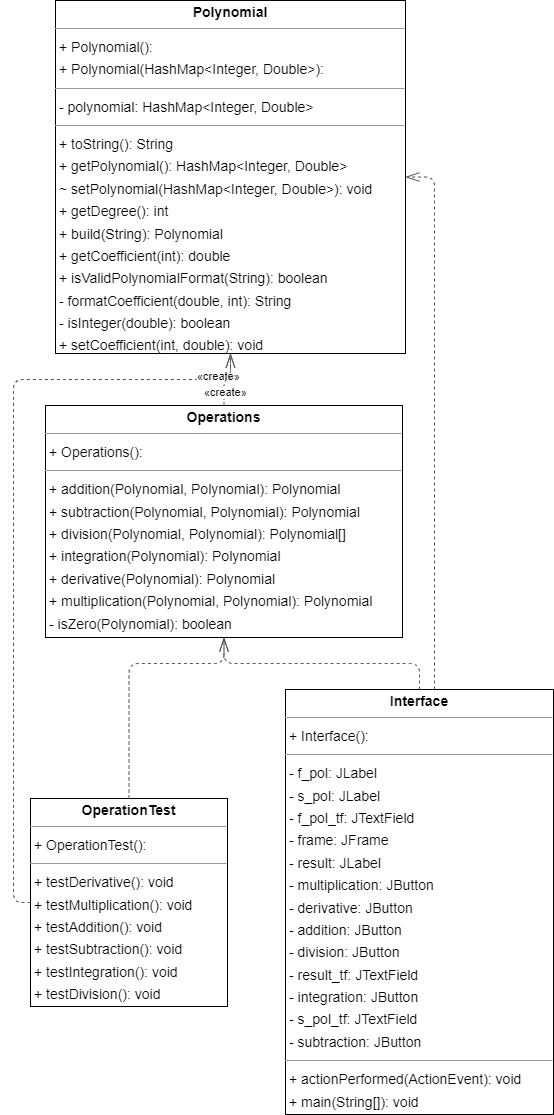
In the **Business Logic package**, the Operations class serves for mathematical computations. Here, algorithms for addition, subtraction, multiplication, division, derivation and integration converge, from basic arithmetic to advanced calculus methods.

As for the **Data Model package**, the Polynomial class stands as the blueprint of application data. It encapsulates the essence of the polynomial expressions, representing coefficients, degrees and terms with precision. Each instance of the Polynomial class embodies a unique mathematical entity.

*Data Structure Identification:*

1. HashMap -> Used in the Polynomial clas to represent polynomial terms, where the key represents the degree and the values represent the coefficients
2. HashSet -> Used in method Division from Operations class as an auxiliary representation of the keySets for iteration
3. String -> Strings are used to represent polynomial expressions, both as input from the user and as output due to the toString method implemented in the Polynomial class

*UML Class Diagram:*



# Implementation

* POLYNOMIAL CLASS

The Polynomial class represents mathematical polynomials, storing terms as mappings between degrees and coefficients. It offers methods for retrieving coefficients, setting coefficients, and generating string representations of polynomials. Additionally, it provides functionality for constructing polynomials from string inputs and validating polynomial formats. Regarding the **fields**, the Polynomial class contains a HashMap structure field.

* getDegree() method retrieves the degree of the polynomial, which is the highest power of the variable ‘x’ by using Collections.max() method in order to sort the keySets

public int getDegree() {

if (this.polynomial.isEmpty()) {

return 0;

}

return Collections.max(this.polynomial.keySet());

}

* getCoefficient(int degree) method retrieves the coefficient of the term with the specified degree in the polynomial; since it uses the getOrDefault method of the HashMap class, default value ‘0.0’ will be returned in case there is no such degree in the polynomial

public double getCoefficient(int degree) {

return this.polynomial.getOrDefault(degree, 0.0);

}

* toString() method generates a string representation of the polynomial, displaying each term with its coefficient and degree, formatted as a mathematical expression; this method also utilizes two helper methods: formatCoefficient(double coefficient, int degree) and isInteger(double value) in order to format the coefficient and check for particular cases such as “1” or “-1”, as well as to convert the coefficient to its integer value if possible

@Override

public String toString() {

StringBuilder stringBuilder = new StringBuilder();

List<Integer> sortedDegrees = new ArrayList<>(polynomial.keySet());

Collections.sort(sortedDegrees, Collections.reverseOrder());

// Handle other terms

for (int degree : sortedDegrees) {

if (degree == 0) {

continue; // Skip degree 0 as it's handled separately

}

double coefficient = polynomial.get(degree);

if (coefficient == 0) {

continue; // Skip terms with zero coefficient

}

if (stringBuilder.length() > 0) {

if (coefficient >= 0) {

stringBuilder.append(" + ");

} else {

stringBuilder.append(" - ");

coefficient = Math.abs(coefficient);

}

}

stringBuilder.append(formatCoefficient(coefficient, degree));

if (degree > 0) {

stringBuilder.append("x");

if (degree > 1) {

stringBuilder.append("^").append(degree);

}

}

}

// Handle constant term (degree 0) separately

if (polynomial.containsKey(0)) {

double constant = polynomial.get(0);

if (constant != 0) {

if (stringBuilder.length() > 0) {

if (constant >= 0) {

stringBuilder.append(" + ");

} else {

stringBuilder.append(" - ");

constant = Math.abs(constant);

}

}

stringBuilder.append(formatCoefficient(constant,0));

}

}

return stringBuilder.toString();

}

private String formatCoefficient(double coefficient, int degree) {

if (isInteger(coefficient)) {

if (coefficient == 1 || coefficient == -1) {

if (degree == 0) {

return Integer.toString((int) coefficient);

} else {

if (coefficient < 0) {

return "-";

} else {

return ""; // Don't display coefficient if it's 1

}

}

} else {

return Integer.toString((int) coefficient); // Convert to integer if possible

}

} else {

return Double.toString(coefficient);

}

}

private boolean isInteger(double value) {

return value == (int) value;

}

* build(String input) together with isValidPolynomialFormat(String input) methods construct the polynomial from a string input, parsing it when encountering the sign “+”; for this reason, I first replaced “-“ signs with “+-“, in order to retrieve negative coefficients with their sign as well; the second static method uses a regex pattern in order to validate the input; since it is a Boolean function, it returns true if the input is valid and false otherwise

public static Polynomial build(String input) {

HashMap<Integer, Double> poly = new HashMap<>();

input = input.replace("-", "+-");

String[] monomials = input.split("\\+");

for (String monomial : monomials) {

if (monomial.isEmpty()) {

continue;

}

double coefficient;

int degree;

if (!monomial.contains("x")) {

coefficient = Double.parseDouble(monomial);

degree = 0;

} else {

String[] parts = monomial.split("x");

if (parts.length == 0) {

coefficient = 1;

} else if (parts[0].equals("-")) {

coefficient = -1;

} else if (parts[0].isEmpty() || parts[0].equals("+")) {

coefficient = parts.length > 1 ? 1 : Double.parseDouble(parts[0]);

} else {

coefficient = Double.parseDouble(parts[0]);

}

degree = parts.length > 1 ? Integer.parseInt(parts[1].substring(1)) : 1;

}

poly.put(degree, coefficient);

}

return new Polynomial(poly);

}

public static boolean isValidPolynomialFormat(String input) {

String regex = "([+-]?\\d\*(\\.\\d\*)?(x)?(\\^\\d+)?)+";

return Pattern.matches(regex, input.trim());

}

The Polynomial class encapsulates the data (coefficients of the polynomial) and the methods that operate on this data. The data is represented as a HashMap and is private, meaning it can only be accessed through the methods provided by the class.

* OPERATIONS CLASS

The Operations class provides a collection of static methods for performing mathematical operations such as addition, subtraction, multiplication, division, derivative, and integration on polynomial objects. These operations are commonly used in polynomial algebra and are implemented to manipulate polynomials represented as hash maps of degrees and coefficients.

* addition(Polynomial p1, Polynomial p2) method computes the addition of two polynomials ‘p1’ and ‘p2’ by iterating over the terms of both polynomials and adding corresponding terms together in order to generate the sum

public static Polynomial addition(Polynomial p1, Polynomial p2) {

HashMap<Integer, Double> result = new HashMap<>();

for (Map.Entry<Integer, Double> term : p1.getPolynomial().entrySet()) {

result.put(term.getKey(), term.getValue());

}

for (Map.Entry<Integer, Double> term : p2.getPolynomial().entrySet()) {

result.merge(term.getKey(), +term.getValue(), Double::sum);

}

return new Polynomial(result);

}

* subtraction(Polynomial p1, Polynomial p2) method is similar to addition, except it computes the difference between the two polynomials; like in the previous case, this method iterates over the terms of both poynomials and subtracts the corresponding terms
* multiplication(Polynomial p1, Polynomial p2) method computes the product of the two polynomials ‘p1’and ‘p2’ by multiplying each term of ‘p1’ with each term of ‘p2’; the degree of the result is represented as the sum of the degrees, whereas the coefficient is computed by multiplying the coefficients of each term(monomial)
* division(Polynomial p1, Polynomial p2) method performs the long division algorithm, resulting in a quotient and a remainder polynomial; besides the computation itself, this method throws two IllegalArgumentExceptions: one in case the divisor is 0(zero) and one in case the degree of the divisor is larger than the degree of the dividend; this method also utilizes the data structure ‘HashSet’ in order to store the degrees(keySets) of the dividend and iterate through them to remove terms with coefficient 0

public static Polynomial[] division(Polynomial p1, Polynomial p2) {

if(p2.getDegree() > p1.getDegree()) {

throw new IllegalArgumentException("Degree of divisor > Degree of divident.");

}

if (p2.getPolynomial().isEmpty() || isZero(p2)) {

throw new IllegalArgumentException("The divisor cannot be zero.");

}

Polynomial quotient = new Polynomial();

Polynomial remainder = new Polynomial();

Polynomial dividend = new Polynomial(p1.getPolynomial());

while (!isZero(dividend) && dividend.getDegree() >= p2.getDegree()) {

int degreeQ = dividend.getDegree() - p2.getDegree();

double coeffQ = dividend.getCoefficient(dividend.getDegree()) / p2.getCoefficient(p2.getDegree());

Polynomial term = new Polynomial();

term.setCoefficient(degreeQ, coeffQ);

quotient = addition(quotient, term);

Polynomial subtractTerm = multiplication(term, p2);

dividend = subtraction(dividend, subtractTerm);

HashSet<Integer> degrees = new HashSet<>(dividend.getPolynomial().keySet());

for (int degree : degrees) {

if (dividend.getCoefficient(degree) == 0) {

dividend.getPolynomial().remove(degree);

}

}

}

remainder = dividend;

return new Polynomial[]{quotient, remainder};

}

* derivative(Polynomial p) method takes as parameter only one polynomial p and computes its derivative, which involves multiplying each term by its degree and reducing the degree by 1
* integration(Polynomial p) method calculates the indefinite integral of the polynomial ‘p’ by increasing the degree of each term by 1 and dividing the coefficient by the new degree

The Operations class provides static methods for performing operations on Polynomial objects. These methods take Polynomial objects as arguments and return a Polynomial object or an array of Polynomial objects. The internal details of how these operations are performed are hidden from the user, which is another form of encapsulation.

* INTERFACE CLASS

The Interface class serves as the graphical user interface (GUI) for the Polynomial Calculator application. This class extends JFrame to create a window-based interface. It contains text fields for inputting two polynomials, buttons for performing various mathematical operations (addition, subtraction, multiplication, division, derivative, and integration), and a text field to display the result of the operations;

**Fields:** *Text Fields (JTextField): f\_pol\_tf and s\_pol\_tf* are text fields where users can input the first and second polynomials, respectively. The result of the operations is displayed in the result\_tf text field.

*Labels (JLabel): f\_pol, s\_pol, and result* are labels indicating the purpose of the corresponding text fields.

*Buttons (JButton): addition, subtraction, multiplication, division, derivative, and integration* are buttons representing different mathematical operations that users can perform on the polynomials.

*Action Listeners*: The class implements the ActionListener interface to handle button clicks. When a button is clicked, the actionPerformed method is invoked to process the user's input and perform the appropriate operation using the Operations class.

**Action handling:** The actionPerformed method checks which button was clicked and performs the corresponding operation on the polynomials entered by the user.

It validates the format of the input polynomials using the *isValidPolynomialFormat* method of the Polynomial class and displays an error message if the format is invalid.

It then constructs Polynomial objects from the input strings and calls the appropriate static methods of the Operations class to perform the requested operation.

The result of the operation is displayed in the result\_tf text field.

***public void actionPerformed(ActionEvent e) {***

***if (e.getSource() == addition || e.getSource() == subtraction ||***

***e.getSource() == multiplication || e.getSource() == division ||***

***e.getSource() == derivative || e.getSource() == integration) {***

***String fPolynomial = f\_pol\_tf.getText();***

***String sPolynomial = s\_pol\_tf.getText();***

***if (!Polynomial.isValidPolynomialFormat(fPolynomial) || !Polynomial.isValidPolynomialFormat(sPolynomial)) {***

***result\_tf.setText("Error: Invalid format");***

***return;***

***}***

***// Proceed with operations***

***Polynomial p1 = Polynomial.build(fPolynomial);***

***Polynomial p2 = Polynomial.build(sPolynomial);***

***if (e.getSource() == addition) {***

***Polynomial result = Operations.addition(p1, p2);***

***result\_tf.setText(result.toString());***

***} else if (e.getSource() == subtraction) {***

***Polynomial result = Operations.subtraction(p1, p2);***

***result\_tf.setText(result.toString());***

***} else if (e.getSource() == multiplication) {***

***Polynomial result = Operations.multiplication(p1, p2);***

***result\_tf.setText(result.toString());***

***} else if (e.getSource() == division) {***

***try {***

Polynomial[] result = Operations.division(p1, p2);

***StringBuilder resultStr = new StringBuilder();***

***resultStr.append("Quotient: ").append(result[0].toString()).append("\n");***

***resultStr.append("Remainder: ").append(result[1].toString());***

***result\_tf.setText(resultStr.toString());***

***} catch (IllegalArgumentException ex) {***

***result\_tf.setText("Error: " + ex.getMessage());***

***}***

***} else if (e.getSource() == derivative) {***

***Polynomial result = Operations.derivative(p1);***

***result\_tf.setText(result.toString());***

***} else if (e.getSource() == integration) {***

***Polynomial result = Operations.integration(p1);***

***result\_tf.setText(result.toString());***

***}***

***}***

***}***

In the Interface class (in Interface.java), all the components (JLabel, JTextField, JButton, JFrame) are private. They can't be accessed directly from outside the class. Public methods are provided (actionPerformed()) in order to interact with these components.

# Results

Within the OperationTest class, I have utilized Junit testing in order to test all methods implemented; this method of testing is designed to ensur the correctness of polynomial operations; the class contains multiple test methods, each targeting a specific operation; within each testing method, polynomial inputs are set up, operations are executed through the Operations class and the results are compared against expected values using Junit assertions;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.junit.jupiter.api.Assertions.\*;

public class OperationTest {

@Test

public void testAddition() {

Polynomial p1 = new Polynomial();

Polynomial p2 = new Polynomial();

p1.getPolynomial().put(4, 3.0);

p1.getPolynomial().put(3, 2.0);

p1.getPolynomial().put(2, 5.0);

p2.getPolynomial().put(4, 2.0);

assertEquals(Operations.addition(p1,p2).toString(), "5x^4 + 2x^3 + 5x^2");

}

@Test

public void testSubtraction() {

Polynomial p1 = new Polynomial();

Polynomial p2 = new Polynomial();

p1.getPolynomial().put(4, 3.0);

p1.getPolynomial().put(3, 2.0);

p1.getPolynomial().put(2, 5.0);

p2.getPolynomial().put(4, 2.0);

assertEquals(Operations.subtraction(p1,p2).toString(), "x^4 + 2x^3 + 5x^2");

}

@Test

public void testMultiplication() {

Polynomial p1 = new Polynomial();

Polynomial p2 = new Polynomial();

p1.getPolynomial().put(4, 3.0);

p1.getPolynomial().put(3, 2.0);

p1.getPolynomial().put(2, 5.0);

p2.getPolynomial().put(4, 2.0);

assertEquals(Operations.multiplication(p1,p2).toString(), "6x^8 + 4x^7 + 10x^6");

}

@Test

public void testDerivative() {

Polynomial p1 = new Polynomial();

p1.getPolynomial().put(4, 3.0);

p1.getPolynomial().put(3, 2.0);

p1.getPolynomial().put(2, 5.0);

assertEquals(Operations.derivative(p1).toString(), "12x^3 + 6x^2 + 10x");

}

@Test

public void testIntegration() {

Polynomial p1 = new Polynomial();

p1.getPolynomial().put(4, 3.0);

p1.getPolynomial().put(3, 2.0);

p1.getPolynomial().put(2, 5.0);

assertEquals(Operations.integration(p1).toString(), "0.6x^5 + 0.5x^4 + 1.6666666666666667x^3");

}

@Test

public void testDivision() {

Polynomial p1 = new Polynomial();

Polynomial p2 = new Polynomial();

p1.getPolynomial().put(2, 1.0);

p1.getPolynomial().put(1, -2.0);

p1.getPolynomial().put(0, 1.0);

p2.getPolynomial().put(1, 1.0);

p2.getPolynomial().put(0, -1.0);

assertEquals(Operations.division(p1,p2)[0].toString(),"x - 1");

assertEquals(Operations.division(p1,p2)[1].toString(),"");

Polynomial p3 = new Polynomial();

Polynomial p4 = new Polynomial();

p3.getPolynomial().put(3, 1.0);

p3.getPolynomial().put(2, -2.0);

p3.getPolynomial().put(1, 6.0);

p3.getPolynomial().put(0, -5.0);

p4.getPolynomial().put(2, 1.0);

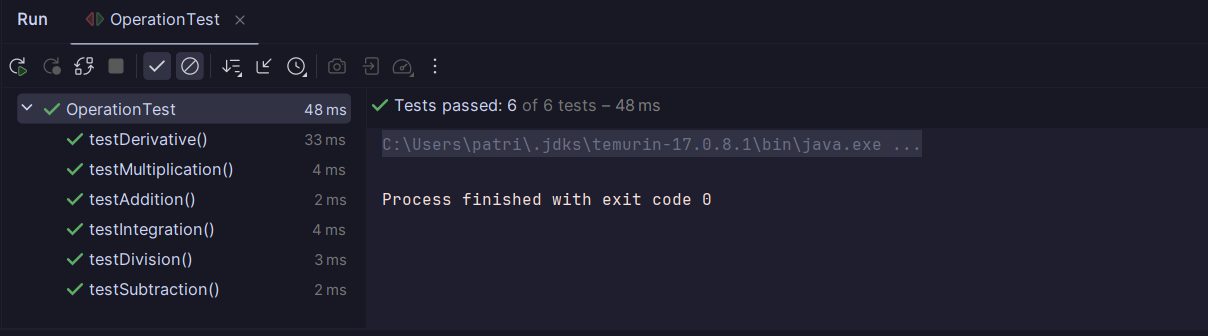
p4.getPolynomial().put(0, -1.0);

assertEquals(Operations.division(p3,p4)[0].toString(),"x - 2");

assertEquals(Operations.division(p3,p4)[1].toString(),"7x - 7");

}

}



# Conclusions

From this assignment, I've learned the importance of designing well-structured and modular code using object-oriented principles. Through the implementation of classes such as Polynomial, Operations, and Interface, I gained a deeper understanding of encapsulation, which is a fundamental concept in object-oriented programming. Additionally, I learned how to use Java Swing for creating graphical user interfaces (GUIs) and JUnit for writing unit tests to ensure the correctness of my code.

As for future development:

* Enhanced functionality: support more advanced mathematical operations such as root finding and factorization
* Improved User Experience: the GUI could be refined with additional features like error handling or additional digit buttons for input
* Optimization: improve the efficiency of operations, especially for large polynomials

Overall, this assignment provided valuable insights into software development practices, GUI design, and testing methodologies, laying a solid foundation for future projects and learning opportunities in Java programming and software engineering.

# Bibliography

<https://dsrl.eu/courses/pt/materials/PT_2024_A1_S1.pdf>

<https://stackoverflow.com/questions/36490757/regex-for-polynomial-expression>

<https://www.geeksforgeeks.org/encapsulation-in-java/>

<https://dsrl.eu/courses/pt/materials/PT_2024_A1_S3.pdf>

<https://www.mathsisfun.com/algebra/polynomials-division-long.html>

<https://stackoverflow.com/questions/32730600/dividing-polynomials-in-java>

<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>