

# DOCUMENTATION

## ASSIGNMENT 1

STUDENT NAME: Tudorache Ioan  
GROUP:30226

# CONTENTS

1. Assignment Objective.....	3
2. Problem Analysis, Modeling, Scenarios, Use Cases .....	3
3. Design .....	10
4. Implementation.....	14
5. Results.....	21
6. Conclusions .....	22
7. Bibliography .....	24

## 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**

- ✓ **Analyze the problem and identify requirements.**
  - Understand the purpose of the polynomial calculator and gather requirements from stakeholders. Categorize requirements into functional (what the system should do) and non-functional (qualities the system should have).
- ✓ **Design the polynomial calculator.**
  - Create a blueprint for the system, including the user interface design, system architecture, and algorithms for polynomial operations. Ensure the design is scalable, modular, and user-friendly.
- ✓ **Implement the polynomial calculator.**
  - Translate the design into code using programming languages and development tools. Develop the user interface, implement polynomial operations, and integrate different components. Follow coding standards and best practices.
- ✓ **Test the polynomial calculator.**
  - Verify the functionality and quality of the system through testing. This includes unit testing to validate functional and non-functional aspects.

## 2. Problem Analysis, Modeling, Scenarios, Use Cases

- **Functional requirements:**

- The polynomial calculator should allow users to insert polynomials.

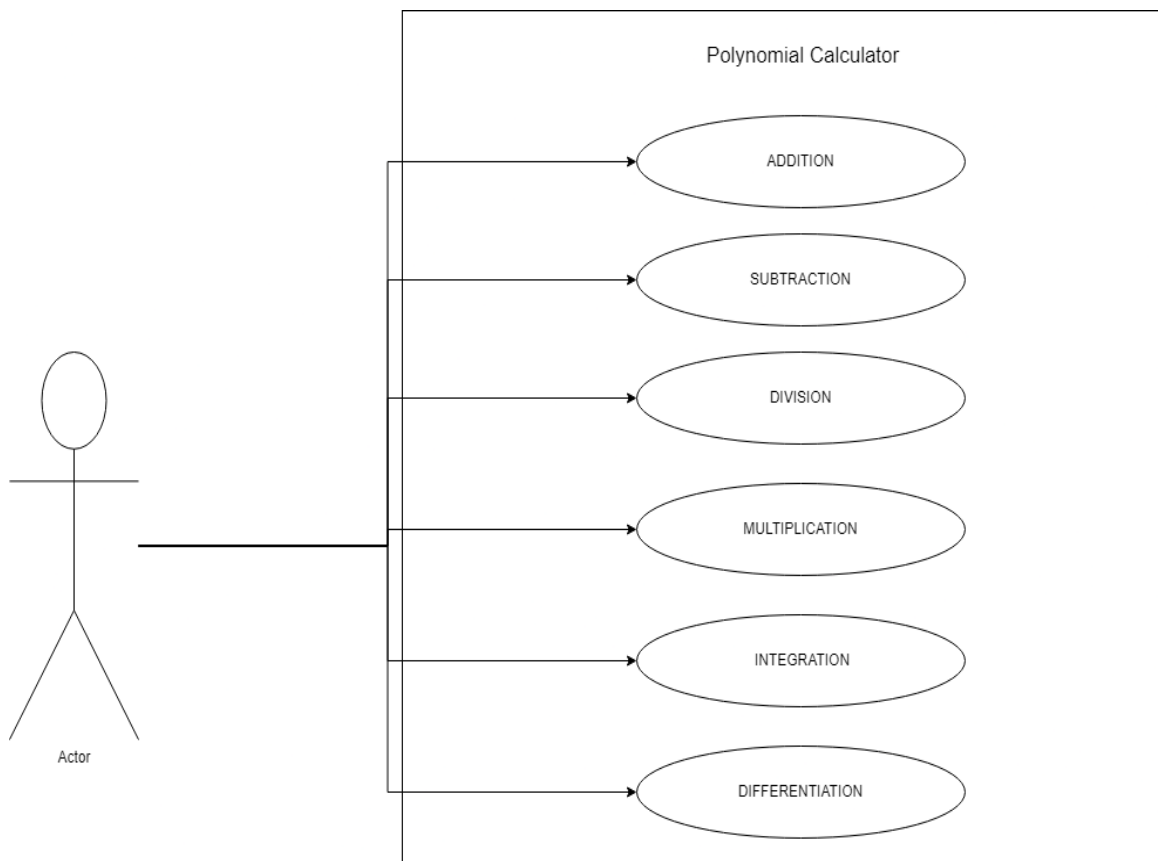
- The polynomial calculator should allow users to select the mathematical operation.
- The polynomial calculator should add two polynomials.
- The calculator should provide functionality to determine the degree of a polynomial.
- The calculator should handle errors gracefully, such as invalid polynomial input or division by zero.
- The calculator should be able to subtract one polynomial from another.
- The calculator should be able to multiply two polynomials together.
- The calculator should be able to divide one polynomial by another.
- The calculator should be able to compute the indefinite integral of a polynomial with respect to the variable of the polynomial.
- The calculator should be able to compute the derivative of a polynomial with respect to the variable of the polynomial.

- **Non-Functional requirements:**

- The polynomial calculator should be intuitive and easy to use by the user.
- The calculator should be reliable and robust, minimizing the likelihood of errors or crashes during operation.
- The calculator should be scalable to handle a large number of polynomial terms or operations without significant degradation in performance.
- The user interface of the calculator should be responsive, providing smooth interaction and feedback to user inputs.

- The calculator should be designed and implemented in a modular and maintainable manner, allowing for easy updates, bug fixes, and enhancements in the future.
- The calculator should be accompanied by comprehensive documentation, including user guides and developer documentation, to aid users in understanding its features and functionality.
- The calculator should perform polynomial operations efficiently, providing results in a timely manner even for complex calculations.

- **Use Case Diagram:**



- **Use Case Description:**

**Use Case:** Addition

**Primary Actor:** User

**Main Success Scenario:**

- 1) The user selects “Option 1”.
- 2) The user inserts the 2 polynomials in the graphical user interface.
- 3) The user selects the “Addition” operation.
- 4) The polynomial calculator evaluates the polynomial.
- 5) The polynomial calculator performs the addition of the two polynomials and displays the result.

**Alternative Sequence:** Incorrect polynomials

- The user inserts incorrect polynomials (e.g. with 2 or more variables).
- The scenario returns to step 1.

**Use Case:** Subtraction

**Primary Actor:** User

**Main Success Scenario:**

- 1) The user selects “Option 1”.
- 2) The user inserts the 2 polynomials in the graphical user interface.

- 3) The user selects the “Subtraction” operation.
- 4) The polynomial calculator evaluates the polynomial.
- 5) The polynomial calculator performs the subtraction of the two polynomials and displays the result.

**Alternative Sequence:** Incorrect polynomials

- The user inserts incorrect polynomials (e.g. with 2 or more variables).
- The scenario returns to step 1.

**Use Case:** Multiplication

**Primary Actor:** User

**Main Success Scenario:**

- 1) The user selects “Option 1”.
- 2) The user inserts the 2 polynomials in the graphical user interface.
- 3) The user selects the “Multiplication” operation.
- 4) The polynomial calculator evaluates the polynomial.
- 5) The polynomial calculator performs the multiplication of the two polynomials and displays the result.

**Alternative Sequence:** Incorrect polynomials

- The user inserts incorrect polynomials (e.g. with 2 or more variables).

- The scenario returns to step 1.

**Use Case:** Division

**Primary Actor:** User

**Main Success Scenario:**

- 1) The user selects “Option 1”.
- 2) The user inserts the 2 polynomials in the graphical user interface.
- 3) The user selects the “Division” operation.
- 4) The polynomial calculator evaluates the polynomial.
- 5) The polynomial calculator performs the division of the two polynomials and displays the result.

**Alternative Sequence:** Incorrect polynomials

- The user inserts incorrect polynomials (e.g. with 2 or more variables).
- The scenario returns to step 1.

**Use Case:** Integration

**Primary Actor:** User

**Main Success Scenario:**

- 1) The user selects “Option 2”.



- 2) The user inserts a polynomial in the graphical user interface.
- 3) The user selects the “Integration” operation.
- 4) The polynomial calculator evaluates the polynomial.
- 5) The polynomial calculator performs the integration of the polynomial and displays the result.

**Alternative Sequence:** Incorrect polynomials

- The user inserts an incorrect polynomial (e.g. with 2 or more variables).
- The scenario returns to step 1.

**Use Case:** Differentiation

**Primary Actor:** User

**Main Success Scenario:**

- 1) The user selects “Option 2”.
- 2) The user inserts a polynomial in the graphical user interface.
- 3) The user selects the “Differentiation” operation.
- 4) The polynomial calculator evaluates the polynomial.
- 5) The polynomial calculator performs the differentiation of the polynomial and displays the result.

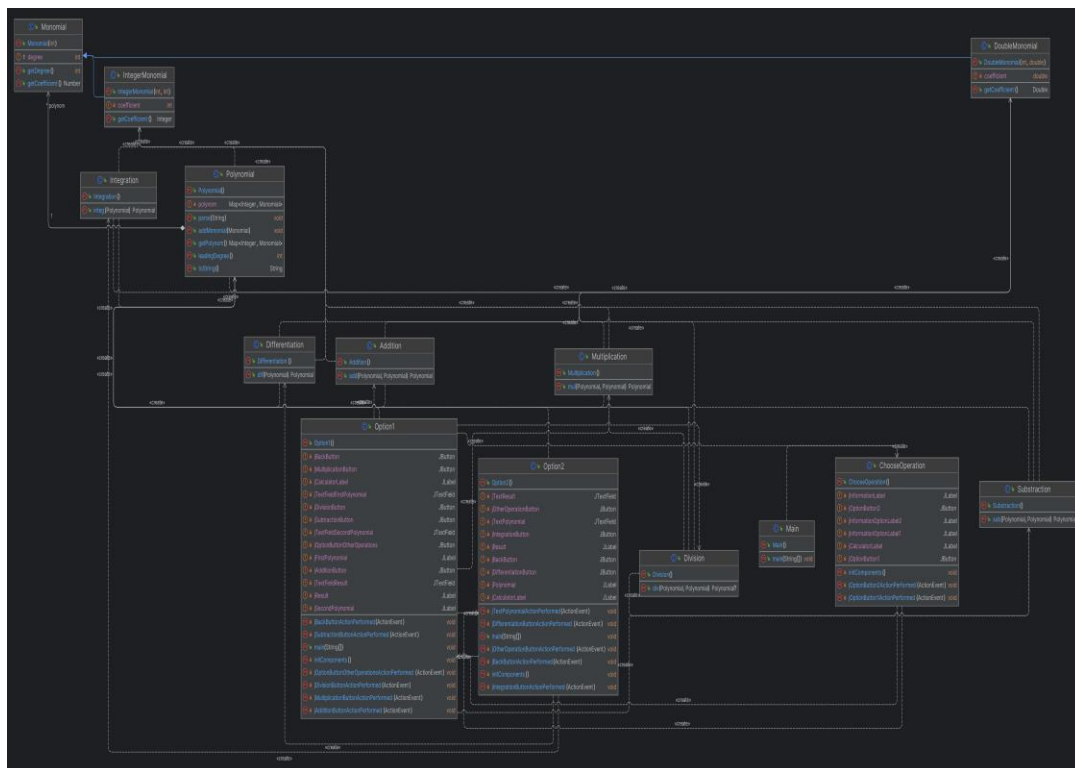
**Alternative Sequence:** Incorrect polynomials

- The user inserts an incorrect polynomial (e.g. with 2 or more variables).
- The scenario returns to step 1.

### 3. Design

- **UML Diagram:**

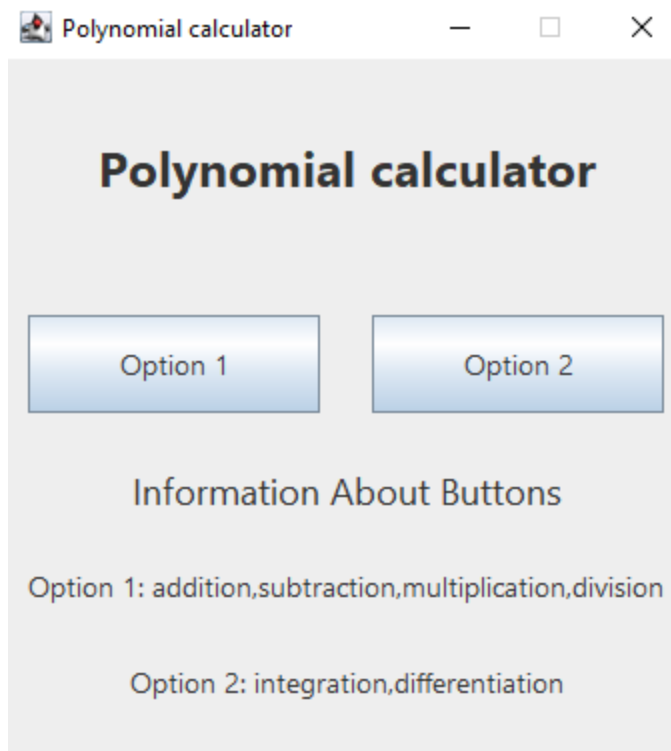
- The UML class diagram illustrates a software application for polynomial operations, comprising classes like Monomial and Polynomial for representing polynomial terms and expressions, along with Operation classes for mathematical operations like Integration and Differentiation. UI Components represent elements for user interaction, while the Main Class serves as the entry point. Relationships between classes, such as inheritance and association, depict how they interact within the system, indicating a comprehensive solution for polynomial manipulation.



- **Graphic Design:**

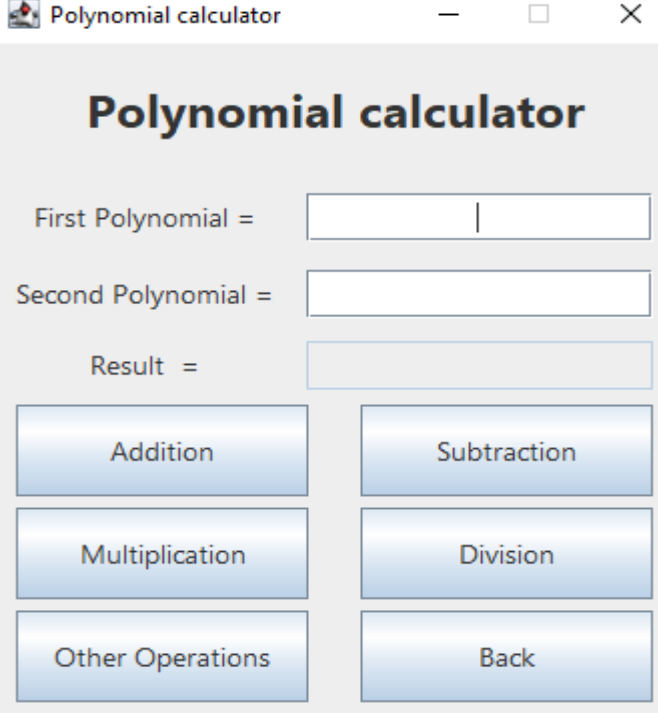
- **Main Menu Interface:**

- Serving as the primary navigation hub, this interface offers users a clear choice between arithmetic operations and differentiation/integration. With large, easy-to-click buttons and descriptive text, users can easily navigate to their desired functionality. The simplicity of the layout and intuitive design language make it easy for users to make informed choices and access the tools they need efficiently.



- **Arithmetic Operations Interface:**

- Here, users can perform arithmetic operations on two separate polynomials. The interface maintains consistency with the first screen, providing fields for both polynomials and displaying results within the same window. The layout ensures intuitive input and immediate verification of results.



The image shows a window titled "Polynomial calculator" with standard window controls (minimize, maximize, close). The interface has a light gray background and a title bar. The main content area is titled "Polynomial calculator" in a large, bold, black font. Below the title, there are three input fields: "First Polynomial =", "Second Polynomial =", and "Result =", each followed by a text box. The "Result" field is currently empty and has a light gray background. Below the input fields, there are six buttons arranged in a 3x2 grid. The buttons are labeled "Addition", "Subtraction", "Multiplication", "Division", "Other Operations", and "Back". The buttons have a blue gradient and a slight shadow effect.

**Polynomial calculator**

First Polynomial =

Second Polynomial =

Result =

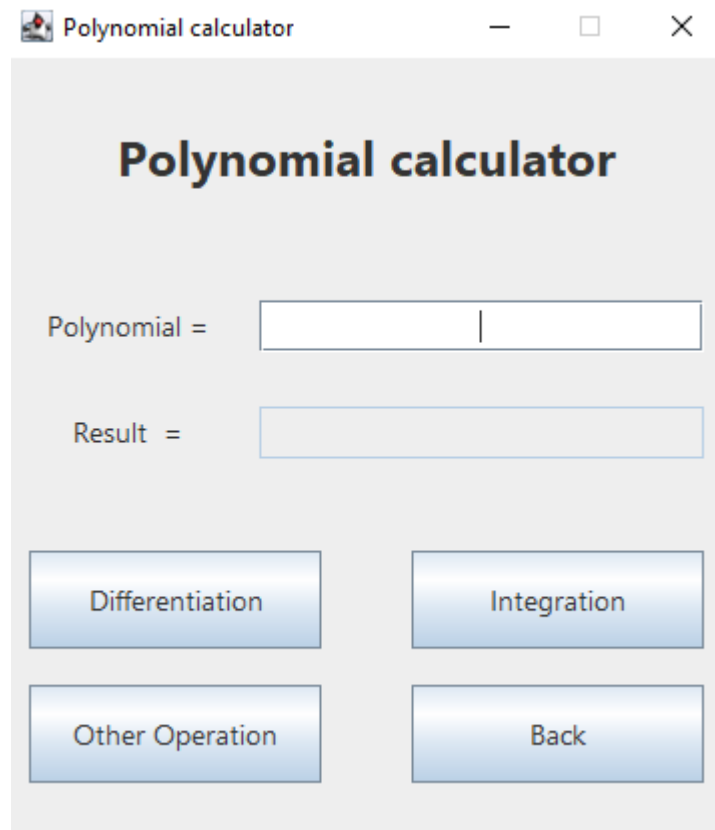
Addition Subtraction

Multiplication Division

Other Operations Back

- **Differentiation and Integration Interface:**

- This interface focuses on differentiation and integration operations, crucial in calculus. Users input a single polynomial and choose to either differentiate or integrate it. The layout is minimalistic, with clear labels for input and results, prominent operation buttons, and easy navigation back to previous options.



The image shows a window titled "Polynomial calculator" with standard window controls (minimize, maximize, close). The interface has a light gray background. At the top, the title "Polynomial calculator" is displayed in a bold, black font. Below the title, there are two input fields. The first is labeled "Polynomial =" and is a white box with a vertical cursor. The second is labeled "Result =" and is a white box. Below these fields, there are four buttons arranged in a 2x2 grid. The top-left button is labeled "Differentiation", the top-right button is labeled "Integration", the bottom-left button is labeled "Other Operation", and the bottom-right button is labeled "Back". All buttons have a blue gradient and a thin black border.

- **Data Structures:**

- The  $\text{Map} \langle \text{Integer}, \text{Monomial} \rangle$  data structure is employed within the Polynomial class to organize polynomial terms by their degrees, facilitating efficient storage and retrieval. This organization aids in various polynomial operations such as addition, subtraction, division

and multiplication, where terms with matching degrees can be easily located and combined. By utilizing Map, polynomial manipulation methods can efficiently access and modify terms based on their degrees, ensuring accurate and streamlined computation.

## 4. Implementation

### Polynomial Class:

- **Fields:**
  - **polynom:** A **Map** storing monomials with their degrees as keys.
- **Important Methods:**
  - **public Polynomial():** Constructor initializing the polynomial as an empty map.
  - **public void addMonomial(Monomial mon):** Method to add a monomial to the polynomial.
  - **public Map<Integer, Monomial> getPolynom():** Method to retrieve the polynomial map.
  - **public int leadingDegree():** Method to find the highest degree in the polynomial.
  - **public void parse(String polynomial):** Method to parse a polynomial string and populate the polynomial map with monomials.
  - **public String toString():** Method to convert the polynomial to a string representation.

### Monomial Class:

- **Fields:**
  - **degree:** An integer representing the degree of the monomial.
- **Important Methods:**
  - **public Monomial(int degree):** Constructor initializing the degree of the monomial.

- **public int getDegree():** Method to retrieve the degree of the monomial.
- **public abstract Number getCoefficient():** Abstract method to be implemented by subclasses to retrieve the coefficient of the monomial.

### **IntegerMonomial Class:**

- **Fields:**
  - **coefficient:** An integer representing the coefficient of the monomial.
- **Important Methods:**
  - **public IntegerMonomial(int degree, int coefficient):** Constructor initializing the degree and coefficient of the monomial.
  - **public Integer getCoefficient():** Method to retrieve the coefficient of the monomial.

### **DoubleMonomial Class:**

- **Fields:**
  - **coefficient:** A double representing the coefficient of the monomial.
- **Important Methods:**
  - **public DoubleMonomial(int degree, double coefficient):** Constructor initializing the degree and coefficient of the monomial.
  - **public Double getCoefficient():** Method to retrieve the coefficient of the monomial.

### **Addition Class:**

- **Fields:**
  - **None**
- **Important Methods:**
  - **public static Polynomial add(Polynomial a, Polynomial b):** Static method that performs polynomial addition. It takes two Polynomial objects as input and returns the result of adding them together as a new Polynomial object. This method iterates through the monomials of

each input polynomial, adding corresponding monomials with the same degree. If a monomial with the same degree already exists in the result polynomial, the coefficients are summed. If not, the monomial is simply added to the result polynomial.

### **Subtraction Class:**

- **Fields:**
  - **None**
- **Important Methods:**
  - **public static Polynomial sub(Polynomial a, Polynomial b):** Static method that performs polynomial subtraction. It takes two Polynomial objects as input and returns the result of subtracting the second polynomial from the first as a new Polynomial object. This method iterates through the monomials of each input polynomial, subtracting corresponding monomials with the same degree. If a monomial with the same degree already exists in the result polynomial, the coefficients are subtracted. If not, the monomial from the second polynomial is multiplied by -1 and added to the result polynomial.

### **Multiplication Class:**

- **Fields:**
  - **None**
- **Important Methods:**
  - **public static Polynomial mul(Polynomial a, Polynomial b):** Static method that performs polynomial multiplication. . It iterates through each term of the first polynomial and multiplies it with each term of the second polynomial, generating monomials with degrees as the sum of the corresponding degrees and coefficients as the product of the coefficients. The resulting polynomial contains the sum of all these multiplied monomials, representing the product of the original polynomials.



### Division Class:

- **Fields:**
  - **None**
- **Important Methods:**
  - **public static Polynomial div(Polynomial dividend, Polynomial divisor):** The Static method that performs polynomial division. method implements polynomial long division to divide two Polynomial objects. It iteratively subtracts multiples of the divisor from the dividend until the degree of the remaining polynomial is less than the divisor. It handles various scenarios including cases where the dividend has a lower degree or is zero and provides feedback if the divisor is not valid.

### Differentiation Class:

- **Fields:**
  - **None**
- **Important Methods:**
  - **public static Polynomial dif(Polynomial a):** Static method that performs polynomial differentiation. It takes a Polynomial object **a** as input and returns the result of differentiating it with respect to its variable as a new Polynomial object. The method iterates through the monomials of the input polynomial and applies the power rule of differentiation to each term. If the degree of the term is zero, indicating a constant, it adds a monomial with degree zero and coefficient zero to the result polynomial to maintain consistency. The differentiated polynomial is then returned as the result.

### Integration Class:

- **Fields:**
  - **None**

- **Important Methods:**
  - **public static Polynomial integ(Polynomial a):** Static method that performs polynomial integration. This method integrates a Polynomial object using the power rule. It iterates through each term, applying integration rules based on its degree. If the term is constant, it adjusts accordingly. Finally, it returns the integrated Polynomial object.

### **ChooseOperation Class:**

- **Fields:**
  - **jCalculatorLabel:** JLabel displaying the title "Polynomial calculator".
  - **jOptionButton1:** JButton for selecting Option 1 (arithmetic operations).
  - **jOptionButton2:** JButton for selecting Option 2 (integration and differentiation).
  - **jInformationLabel:** JLabel displaying information about the buttons.
  - **jInformationOptionLabel1:** JLabel providing information about Option 1 (arithmetic operations).
  - **jInformationOptionLabel2:** JLabel providing information about Option 2 (integration and differentiation).
- **Important Methods:**
  - **private void initComponents():** Method for initializing and setting up the components of the JFrame.
  - **private void jOptionButton1ActionPerformed(java.awt.event.ActionEvent evt):** ActionPerformed method for handling button click events on Option 1 button. It creates and displays an instance of the Option1 frame while hiding the current frame.
  - **private void jOptionButton2ActionPerformed(java.awt.event.ActionEvent evt):** ActionPerformed method for handling button click events on Option 2

button. It creates and displays an instance of the Option2 frame while hiding the current frame.

### **Option1 Class:**

- **Fields:**
  - **jCalculatorLabel:** JLabel displaying the title "Polynomial calculator".
  - **jAdditionButton:** JButton for performing addition operation.
  - **jSubtractionButton:** JButton for performing subtraction operation.
  - **jMultiplicationButton:** JButton for performing multiplication operation.
  - **jDivisionButton:** JButton for performing division operation.
  - **jFirstPolynomial:** JLabel for displaying "First Polynomial =" text.
  - **jSecondPolynomial:** JLabel for displaying "Second Polynomial =" text.
  - **jTextFieldFirstPolynomial:** JTextField for entering the first polynomial.
  - **jTextFieldSecondPolynomial:** JTextField for entering the second polynomial.
  - **jResult:** JLabel for displaying "Result =" text.
  - **jTextFieldResult:** JTextField for displaying the result of operations.
  - **jOptionButtonOtherOperations:** JButton for accessing other operations (Option2).
  - **jBackButton:** JButton for returning to the main menu.
- **Important Methods:**
  - **private void initComponents():** Method for initializing and setting up the components of the JFrame.
  - **private void jAdditionButtonActionPerformed(java.awt.event.ActionEvent evt):** ActionPerformed method for performing addition operation when the addition button is clicked.

- **private void**  
**jSubtractionButtonActionPerformed(java.awt.event.ActionEvent evt):** ActionPerformed method for performing subtraction operation when the subtraction button is clicked.
- **private void**  
**jMultiplicationButtonActionPerformed(java.awt.event.ActionEvent evt):** ActionPerformed method for performing multiplication operation when the multiplication button is clicked.
- **private void**  
**jDivisionButtonActionPerformed(java.awt.event.ActionEvent evt):** ActionPerformed method for performing division operation when the division button is clicked.
- **private void**  
**jOptionButtonOtherOperationsActionPerformed(java.awt.event.ActionEvent evt):** ActionPerformed method for accessing other operations (Option2) when the corresponding button is clicked.
- **private void**  
**jBackButtonActionPerformed(java.awt.event.ActionEvent evt):** ActionPerformed method for returning to the main menu when the back button is clicked.

## **Option2 Class:**

- **Fields:**
  - **jCalculatorLabel:** JLabel displaying the title "Polynomial calculator".
  - **jDifferentiationButton:** JButton for performing differentiation operation.
  - **jIntegrationButton:** JButton for performing integration operation.
  - **jPolynomial:** JLabel for displaying "Polynomial =" text.
  - **jTextPolynomial:** JTextField for entering the polynomial.
  - **jResult:** JLabel for displaying "Result =" text.
  - **jTextResult:** JTextField for displaying the result of operations.

- **jOtherOperationButton**: JButton for accessing other operations (Option1).
- **jBackButton**: JButton for returning to the main menu.
- **Important Methods**:
  - **private void initComponents()**: Method for initializing and setting up the components of the JFrame.
  - **private void jDifferentiationButtonActionPerformed(java.awt.event.ActionEvent evt)**: ActionPerformed method for performing differentiation operation when the differentiation button is clicked.
  - **private void jIntegrationButtonActionPerformed(java.awt.event.ActionEvent evt)**: ActionPerformed method for performing integration operation when the integration button is clicked.
  - **private void jOtherOperationButtonActionPerformed(java.awt.event.ActionEvent evt)**: ActionPerformed method for accessing other operations (Option1) when the corresponding button is clicked.
  - **private void jBackButtonActionPerformed(java.awt.event.ActionEvent evt)**: ActionPerformed method for returning to the main menu when the back button is clicked.

## 5. Results

- The provided testing scenarios are aimed at verifying the correctness of the parsing functionality or mathematical operations in the polynomial representation module. The tests ensure that when a polynomial string is parsed, it correctly creates a polynomial object with the expected monomials or that mathematical operations are performed correctly.

- Utilizing the JUnit testing framework, each test method outlines a specific scenario where either parsing or mathematical operations are conducted. The test cases are structured to compare the output of the functionality under test against manually determined expected outcomes.
- The use of unit tests like these helps ensure the correctness of the code by validating its behavior against expected outcomes. It provides confidence that the parsing functionality works as intended.
- These tests are crucial for confirming that the polynomial calculator behaves as expected, accurately handling polynomial expressions and mathematical computations. By systematically verifying the correctness of parsing or mathematical operations, these testing scenarios contribute significantly to the reliability and robustness of the polynomial representation module within the calculator application.

✓ Test_Methods	167 ms
✓ OperationsTest	157 ms
✓ subTest()	87 ms
✓ addTest()	15 ms
✓ integTest()	14 ms
✓ mulTest()	13 ms
✓ difTest()	13 ms
✓ divTest()	15 ms
✓ ParseTest	10 ms
✓ parseTest()	10 ms

## 6. Conclusions

The following should be presented: conclusions, what you have learned from the assignment, future developments.

The development of the polynomial calculator with a graphical interface is a significant achievement in providing users with a convenient tool for polynomial operations. Through the completion of this assignment, several key insights and learnings have emerged, paving the way for future enhancements and developments.

### What I Have Learned from the Assignment:

1. **Requirements Analysis:** Properly understanding and analyzing the requirements of a software project is crucial for its success. Through this assignment, I learned the importance of capturing both functional and non-functional requirements to ensure that the final product meets the needs of its users.

2. **Design and Implementation:** Designing the architecture and implementing the functionality of the polynomial calculator taught me about the importance of modularity, scalability, and user experience. Breaking down complex tasks into smaller, manageable components and ensuring clear communication between different modules is essential for building robust software systems.
3. **User Interface Design:** Developing a graphical interface for the polynomial calculator involved considerations of usability, intuitiveness, and visual design. I learned about the significance of user-centered design principles and the importance of creating interfaces that are easy to navigate and understand.
4. **Mathematical Operations:** Working on mathematical operations such as polynomial addition, subtraction, multiplication, division, integration, differentiation, and evaluation deepened my understanding of these concepts and their implementation in software.
5. **Testing:** Testing the polynomial calculator helped me understand the importance of rigorous testing to ensure the correctness and reliability of software systems. It highlighted the need for both unit tests and integration tests to validate the functionality of individual components and the system.

#### **Future Developments:**

1. **Enhanced Functionality:** The polynomial calculator can be further expanded to support additional mathematical functions, such as trigonometric functions, logarithmic functions, and exponential functions, providing users with a more comprehensive tool for mathematical analysis.
2. **Improved User Interface:** Future developments could focus on refining the graphical interface of the calculator to make it more visually appealing and user-friendly. This could involve incorporating advanced graphical elements, customization options, and interactive features to enhance the user experience.
3. **Performance Optimization:** Optimizing the performance of the polynomial calculator to handle larger polynomial expressions and complex mathematical operations efficiently would be a valuable future development. This could involve implementing algorithms for efficient polynomial manipulation and leveraging parallel processing techniques for computation-intensive tasks.

**In conclusion,** the development of the polynomial calculator with a graphical interface has been a rewarding experience, providing valuable insights into software development processes, mathematical concepts, and user interface design principles. Moving forward, there are numerous opportunities for further enhancements and developments to make the calculator even more powerful, user-friendly, and versatile.

## 7. Bibliography

1. Creating Diagrams - <https://app.diagrams.net/>
2. Information about the project - <https://dsrl.eu/courses/pt/>
3. How to make regex? - <https://regexr.com/7tini>