DOCUMENTATION

ASSIGNMENT 1

- POLYNOMIAL CALCULATOR-

STUDENT NAME: Pitaru Alexandra-Elena
GROUP: 30423

CONTENTS

1. Assignment Objective

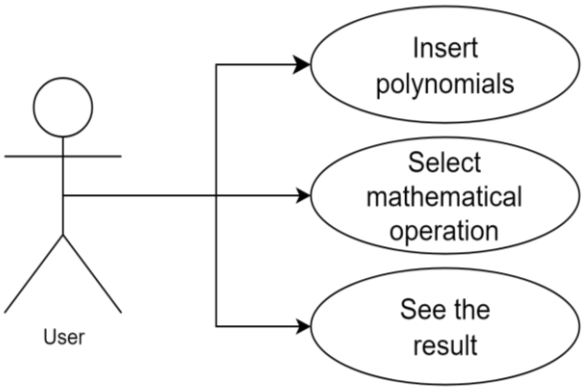
- Main objective of the assignment:

Create and develop a polynomial calculator application that includes both a graphical user interface and the underlying logic for performing mathematical operations on polynomials.

- The sub-objectives:
    - Analyze the problem and identify requirements (Section 2)
        - Gather user requirements
        - Analyze the mathematical operations
        - Identify edge cases
    - Design the polynomial calculator (Section 3)
        - User interface design
        - Algorithm design
        - Data model design
        - Error handling design
    - Implement the polynomial calculator (Section 4)
        - Code for user interface components
        - Implement back-end logic
        - Implement error handling
    - Test the polynomial calculator (Section 5)

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, subtract, multiply or divide two polynomials
    - The polynomial calculator should derivate or integrate one polynomial

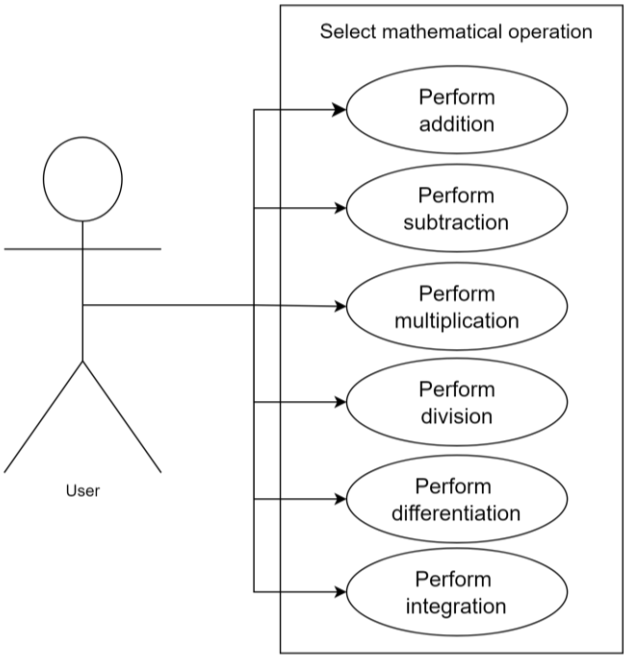


1. **Insert polynomials**

   Actor: User

   Steps:

- User opens the polynomial calculator app
- User selects the text fields to input polynomials
- User enters the polynomial as a string



2. **Select mathematical operation**

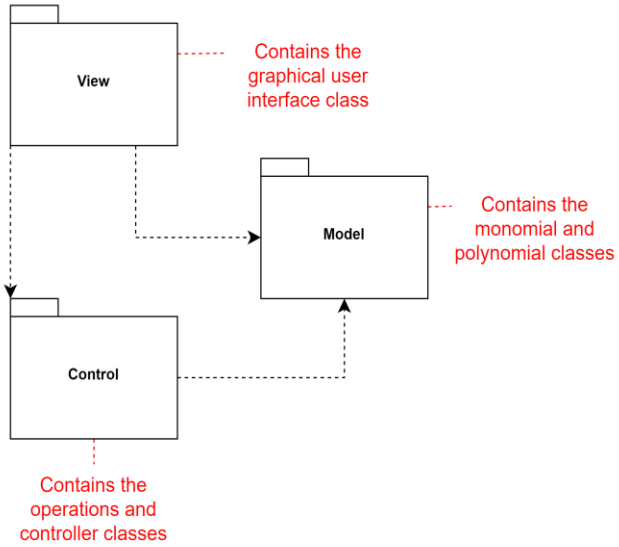
   Actor: User

   Steps:

   - The user decides which operation he wants to perform on the polynomials
   - The user presses the button corresponding to the desired operation

- Non-Functional Requirements:
  - Error handling: Error such as wrong format polynomials or division by 0 should be handled and an appropriate error message should be provided
  - Usability: The polynomial calculator should be intuitive and easy to use by the user
  - User Interface Design: The polynomial calculator's user interface should be visually appealing, with clear and explicit labeling of buttons and controls.

3. Design



1. **The Control-Model-View (CMV)** architectural pattern is a design paradigm that separates the concerns of an application into three distinct components: Control, Model, and View.

- **Model (M):**

The Model represents the underlying data and business logic of the application. It encapsulates the application's state and behavior, providing methods to manipulate and access data without concern for how it will be presented to the user. In my project, the Monomial and Polynomial classes in the model package fulfill this role. They define the structure associated with polynomial expressions.

- **View (V):**

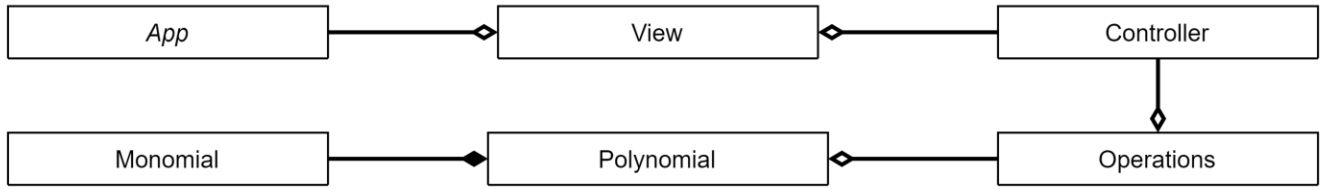
The View is responsible for presenting the application's user interface to the user. It translates the data provided by the Model into a user-friendly format and displays it to the user. In my project, the View class in the view package represents the graphical user interface (GUI) where users interact with the polynomial calculator. It displays polynomial expressions and allows users to input data and select operations.

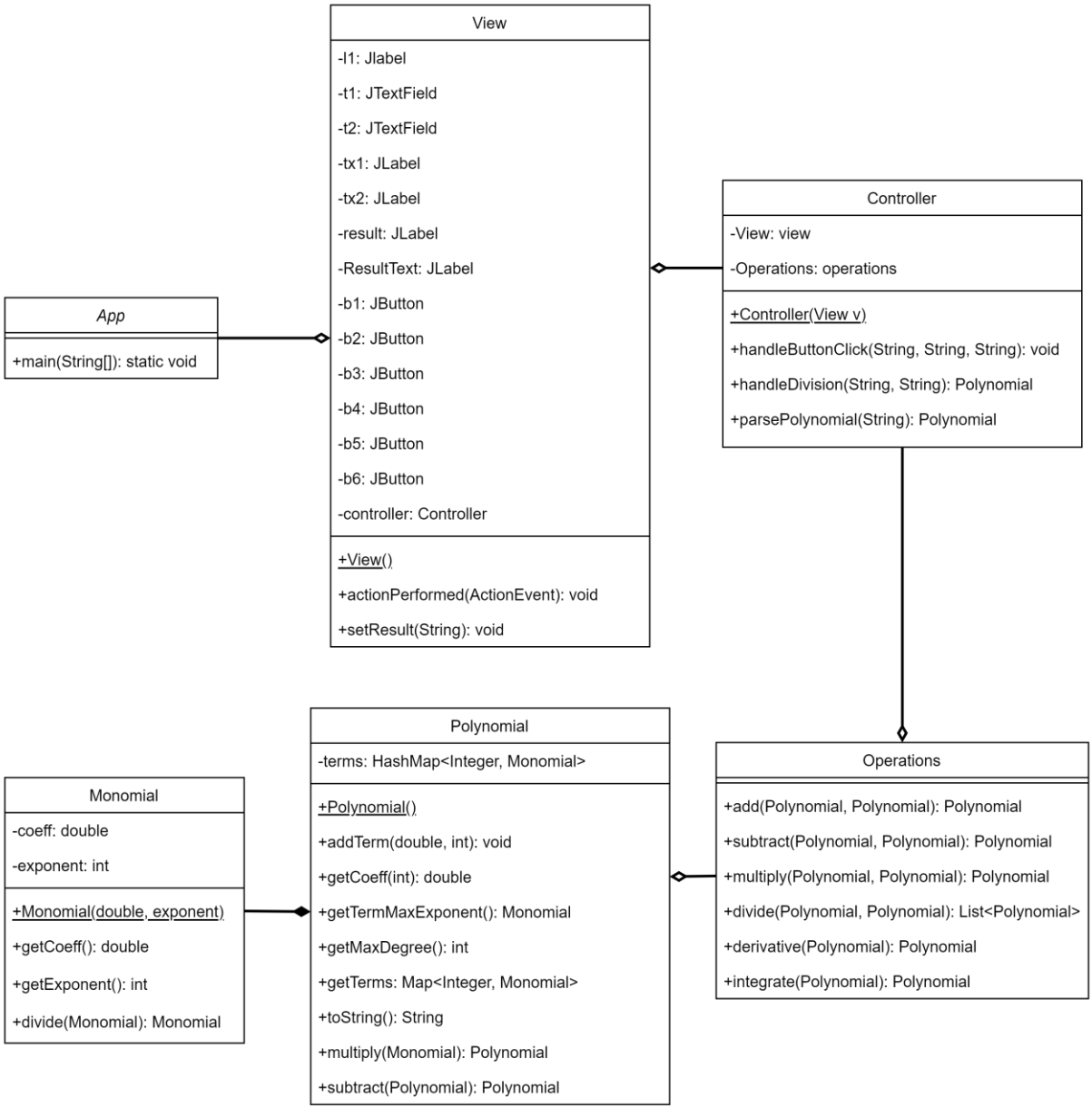
- **Control (C):**

The Control acts as an intermediary between the Model and View components. It coordinates user input, retrieves data from the Model, and updates the View accordingly. It contains the application's logic for processing user requests and initiating actions based on those requests. In my project, the Operations class in the control package serves as the Control component. It provides methods for performing mathematical operations on polynomials and coordinates the interaction between the Model (polynomials) and the View (GUI).

By adopting the CMV pattern in my project, I have achieved a clear separation of concerns, making my code easier to understand, maintain, and extend. Each component focuses on a specific aspect of the application, promoting code reusability and modularity.

2. Class Diagram





3. Used data structures

In my project I used primitive data types: integers and doubles but also HashMap, ArrayList and Arrays. I worked with HashMap in the Polynomial class to store the terms of the polynomial. The ArrayList was used in the Operations class to store the result of the polynomial division. I also used the new created objects Monomial and Polynomial.

4. Used Algorithms

a. Arithmetic Operations: Addition, subtraction, multiplication, division of polynomials are implemented using algorithms to manipulate monomials and polynomials.

b. Calculus Operations: Derivative and integral computations involve algorithms to differentiate and integrate polynomials.

c. Parsing: The parsing algorithm is used to convert polynomial expressions entered by the user into polynomial objects that can be processed by the calculator.

d. Display Formatting: Algorithms are used to format and display polynomial expressions and results in a user-friendly manner, both in the console and in the graphical user interface (GUI).

- **Addition:**

  The sum of the two polynomials is obtained by storing the first polynomial in the result, then iterating through the second polynomial, if there is already a term with the same exponent as the current term, the coefficients are added together, and if there is no term with that coefficient, the term is simply added to the result.

- **Subtraction:**

  The subtraction of the two polynomials is obtained by storing the first polynomial in the result, changing the sign of the coefficients in the second polynomial then iterating through the second polynomial, if there is already a term with the same exponent as the current term, the coefficients together, and if there is no term with that coefficient, the term is simply added to the result.

- **Multiplication:**

  The product of two polynomials is obtained by multiplying term by term and combining the result.

- **Division:**

  This method performs polynomial division by iteratively dividing the leading terms of the dividend (N) and the divisor (D). It calculates the quotient and remainder by dividing the leading terms and updates the dividend with the result of subtracting the product of the divisor and quotient from the current remainder. This process continues until the degree of the remaining polynomial (remainder) is less than the degree of the divisor (D). Finally, it returns a list containing the remainder followed by the quotient.

- **Differentiation:**

  It iterates through each term of the polynomial and applies the power rule of differentiation.
  For each term, it multiplies the coefficient by the exponent and decrements the exponent by one.

- **Integration:**

  It iterates through each term of the polynomial and applies the power rule of integration.
  For each term, it divides the coefficient by the exponent plus one and increments the exponent by one.

4. Implementation

I. Class Description

1) The Model:

a. Polynomial Class:

The Polynomial class represents a polynomial and contains a HashMap to store monomials.

Constructor:

public Polynomial(): Default constructor initializes the polynomial.

Methods:

public void addTerm(double coeff, int exponent): Adds a monomial to the polynomial.

public void removeTerm(int exponent): Removes a monomial from the polynomial.

public double getCoeff(int exponent): Returns the coefficient of a monomial with the given exponent.

public Monomial getTermWithMaxExponent(): Returns the monomial with the highest exponent.

public int getMaxDegree(): Returns the highest degree of the polynomial.

Arithmetic Operations:

public Polynomial multiply(Monomial monomial): Multiplies the polynomial by a monomial.

public Polynomial subtract(Polynomial poly): Subtracts another polynomial from this polynomial.

Utility Methods:

public String toString(): Converts the polynomial to a string representation.

b. Monomial Class:

The Monomial class represents a single monomial in the polynomial.

Constructor:

public Monomial(double coeff, int exponent): Initializes the monomial with the given coefficient and exponent.

Methods:

public double getCoeff(): Returns the coefficient of the monomial.

public int getExponent(): Returns the exponent of the monomial.

2) The Control:

a) Operations Class:

The Operations class contains methods to perform arithmetic operations on polynomials.

Methods:

public Polynomial add(Polynomial poly1, Polynomial poly2): Adds two polynomials.

public Polynomial subtract(Polynomial poly1, Polynomial poly2): Subtracts one polynomial from another.

public Polynomial multiply(Polynomial poly1, Polynomial poly2): Multiplies two polynomials.

public List<Polynomial> divide(Polynomial N, Polynomial D): Divides one polynomial by another.

public Polynomial derivative(Polynomial poly1): Computes the derivative of a polynomial.

public Polynomial integrate(Polynomial poly1): Computes the integral of a polynomial.

b) Controller Class:

The Controller class handles user interactions and coordinates actions between the model and the view.

Constructor:

public Controller(View v): Initializes the controller with a reference to the view.

Methods:

public void handleButtonClick(String actionCommand, String polynomial1, String polynomial2): Handles button clicks and performs corresponding operations.

3) The View:

a) View Class:

The View class represents the graphical user interface (GUI) of the application.

Constructor:

public View(): Initializes the GUI components and sets up the layout.

Methods:

Accessor and Mutator methods for operands and results.

Action listeners for GUI buttons to perform arithmetic operations and display results.

4) The Application:

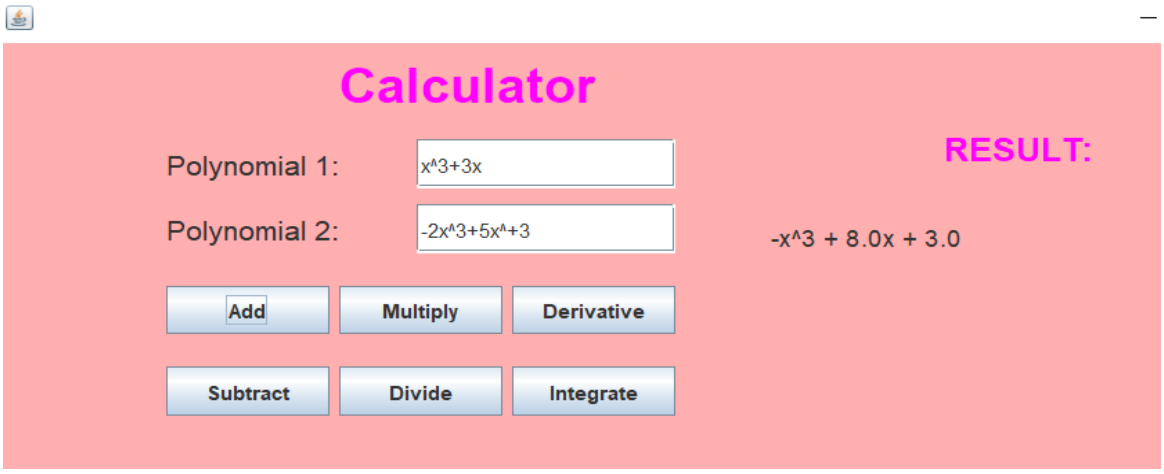
a) App Class:

The App class is the entry point of the application.

Main Method:

public static void main(String[] args): Instantiates the View and Controller classes to start the application.

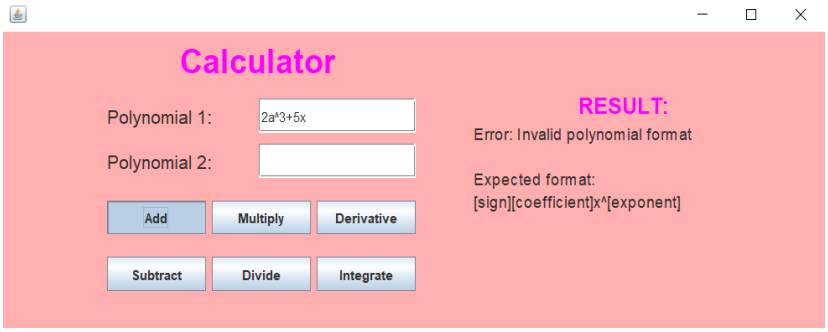
II. Graphical User Interface description



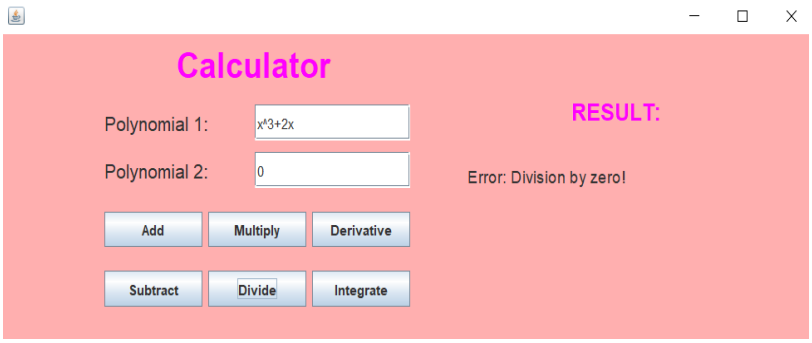
The Graphical User Interface represents the connection between the user and the application.

The interface is intuitive and easy to use. The user inputs the two polynomials that he wants to perform operations on in the two text fields. Then he presses the button corresponding to the desired operation and the result will appear on the screen, under the "RESULT" title.

The two polynomials inserted should respect some conditions. The expected format of a polynomial is: <sign><coefficient>x^<exponent>. If no sign('+' or '-') is explicitly mentioned between terms, it will be considered as a plus sign.



→ If the polynomial doesn't respect the format, an error message will inform the user about it:



→ There is also a warning for division by zero

5. Results

I used JUnit for unit testing.

```xml
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>RELEASE</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

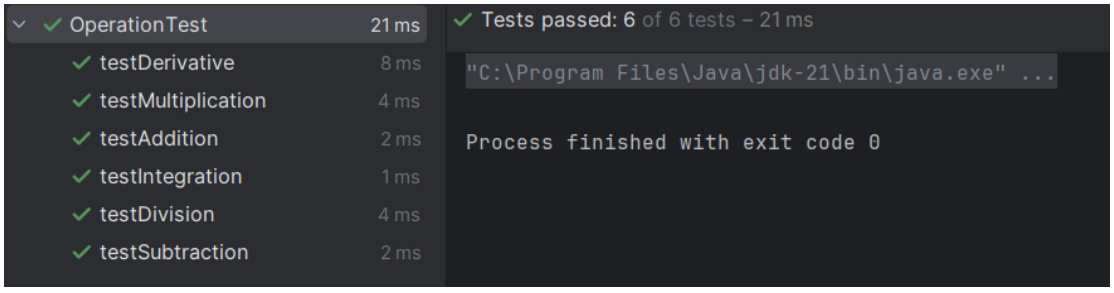
JUnit dependencies included in the *pom.xml* file

```java
import org.example.control.Operations;
import org.example.model.Polynomial;
import org.junit.Test;
import java.util.List;
import static org.junit.Assert.assertEquals;
```

I imported JUnit in my testing class

```java
@Test
public void testAddition() {
    Operations operations = new Operations();

    // Create instances of Polynomial representing the polynomials
    Polynomial poly1 = new Polynomial();
    poly1.addTerm( coeff: 1,  exponent: 2);  // x^2
    poly1.addTerm( coeff: 3,  exponent: 1);  // 3x

    Polynomial poly2 = new Polynomial();
    poly2.addTerm( coeff: 1,  exponent: 3);  // x^3
    poly2.addTerm( coeff: 2,  exponent: 2);  // 2x^2

    // Call the add method with the Polynomial instances
    Polynomial result = operations.add(poly1, poly2);

    // Define the expected result polynomial
    Polynomial expected = new Polynomial();
    expected.addTerm( coeff: 1,  exponent: 3);  // x^3
    expected.addTerm( coeff: 3,  exponent: 2);  // 3x^2
    expected.addTerm( coeff: 3,  exponent: 1);  // 3x

    // Assert that the result polynomial matches the expected polynomial
    assertEquals(expected, result);
}
```

This is an example of a testing method. I implemented sutraction, division, multiplication, derivative and integration similarly.



All tests have been passed, so the calculator works correctly.

6. Conclusions

Reflecting on the project, I've likely identified areas for improvement and areas where I can further enhance my skills. This project serves as a learning experience, highlighting both my strengths and areas for growth in software development.

This project helped me reinforce my knowledge of Object-Oriented Principles, application of Data Structures and Algorithms and Error handling. For me, working for this assignment was challenging but provided continuous learning and improvement in these areas:

- Algorithm Design: I've learned to design algorithms for polynomial operations such as addition, subtraction, multiplication, division, differentiation, and integration.
- Data Structures: I've become familiar with data structures like HashMaps, ArrayLists, and arrays, which are essential for storing and manipulating polynomial terms efficiently.
- User Interface Development: I've developed skills in creating graphical user interfaces (GUIs) using Java Swing.
- Software Design Patterns: I have employed design patterns like Model-View-Controller (MVC) to organize my code.
- Testing and Debugging: I've practiced testing my code and debugging any issues that arise, ensuring that my application works correctly and reliably.

In addition to the skills mentioned earlier, I have learned about regular expressions (regex) and the Pattern Matcher in Java. These are powerful tools for pattern matching and string manipulation, which I have used in my project for parsing polynomial input strings.

```
Pattern termPattern = Pattern.compile( regex: "([-+]?\\d*x\\^\\d+|[-+]?\\d*x|[-+]?\\d+)");

Matcher matcher = termPattern.matcher(polynomialString);
```

Future Developments:

- History: The calculator could maintain a history of operations performed, allowing users to review previous calculations.
- Memory functions: Users may want to store and recall polynomials for later use, so the calculator could provide memory functions such as memory storage and retrieval.
- Customization options: Users may want to customize the appearance or behavior of the calculator, such as adjusting the display settings or defining custom mathematical functions.
- Cross-platform compatibility: The calculator could be designed to run on multiple platforms and devices, such as desktop computers, tablets, and smartphones, with consistent functionality and user experience across different environments.
- Collaboration features: The calculator might support collaboration features, allowing multiple users to work on the same polynomial or share calculations in real-time.
- Export and import: Users may want to export or import polynomials from external sources, so the calculator could provide functionality for importing polynomial data from files or exporting calculations to other formats.

7. Bibliography

1. *Teach Yourself Java in 21 days – Laura Lemay :* *https://www.cs.cmu.edu/afs/cs.cmu.edu/user/gchen/www/download/java/LearnJava.pdf.*
2. *What are Java classes? - www.tutorialspoint.com*
3. *What is REGEX? - https://www.computerhope.com/jargon/r/regex.htm*
4. *Using pattern matcher - https://www.geeksforgeeks.org/matcher-pattern-method-in-java-with-examples/*
   -https://www.w3schools.com/java/java_regex.asp

5. *Java Swing tutorial:* *https://www.youtube.com/watch?si=_o7Ptnb7abwLHRrg&v=GG950kz9p5Y&feature=youtu.be*

6. *Lectures*