DOCUMENTATION

ASSIGNMENT 1

POLYNOMIAL CALCULATOR

STUDENT NAME: ȚĂLU MIRCEA
GROUP: 30433

CONTENTS

1. Assignment Objective

The objective of this project is to design and implement a polynomial calculator with a dedicated graphical user interface (GUI) that allows the user to insert polynomials, select a mathematical operation (addition, subtraction, multiplication, division, derivative, integration), and view the result. The polynomials will be limited to one variable and integer coefficients.

The polynomial calculator will provide a user-friendly interface that allows users to input polynomial expressions using standard mathematical notation. The program will parse the input and evaluate the expression using the selected mathematical operation. The result will be displayed on the GUI in a clear and easy-to-read format.

The polynomial calculator will be implemented in a programming language of the developer's choice and will utilize a GUI library or framework to create the graphical interface. The program should be well-designed, efficient, and easy to use, with appropriate error handling and user feedback.

Overall, this project will provide an opportunity to demonstrate proficiency in programming, user interface design, and mathematical operations on polynomials.

2. Problem Analysis, Modeling, Scenarios, Use Cases

Problem Analysis

The polynomial calculator is designed to solve mathematical operations on polynomials with one variable and integer coefficients. The calculator will be implemented with a graphical user interface that will allow the user to input polynomial expressions, select a mathematical operation, and view the result.

To solve this problem, the polynomial calculator will need to parse input strings to identify the polynomial expression and mathematical operation to be performed. Once identified, the polynomial calculator will perform the mathematical operation on the input polynomial expressions and display the result in a user-friendly format.

Modeling

The polynomial calculator will be implemented as an object-oriented program. The program will be divided into several classes that will handle different functionalities such as polynomial representation, mathematical operations, and user interface.

The polynomial expression will be modeled as an object, and each object will represent a polynomial. Each polynomial object will contain a list of coefficients and the degree of the polynomial. The mathematical operations will also be implemented as objects, each handling a specific operation.

Scenarios

The following are some scenarios that the polynomial calculator will be able to handle:

1. User enters two polynomial expressions to be added and clicks on the "add" button. The polynomial calculator will perform the addition operation and display the result on the GUI.
2. User enters two polynomial expressions to be subtracted and clicks on the "subtract" button. The polynomial calculator will perform the subtraction operation and display the result on the GUI.
3. User enters two polynomial expressions to be multiplied and clicks on the "multiply" button. The polynomial calculator will perform the multiplication operation and display the result on the GUI.
4. User enters two polynomial expressions to be divided and clicks on the "divide" button. The polynomial calculator will perform the division operation and display the result on the GUI.
5. User enters a polynomial expression to be integrated and clicks on the "integrate" button. The polynomial calculator will perform the integration operation and display the result on the GUI.
6. User enters a polynomial expression to be differentiated and clicks on the "differentiate" button. The polynomial calculator will perform the differentiation operation and display the result on the GUI.

Use Cases

The following are some use cases for the polynomial calculator:

1. Adding two polynomials.
2. Subtracting two polynomials.
3. Multiplying two polynomials.
4. Dividing two polynomials.
5. Integrating a polynomial expression.
6. Differentiating a polynomial expression.

Each use case will correspond to a specific button or menu option on the GUI. The user will select the desired use case and input the necessary polynomial expressions, and the polynomial calculator will perform the appropriate operation and display the result on the GUI.

3. Design

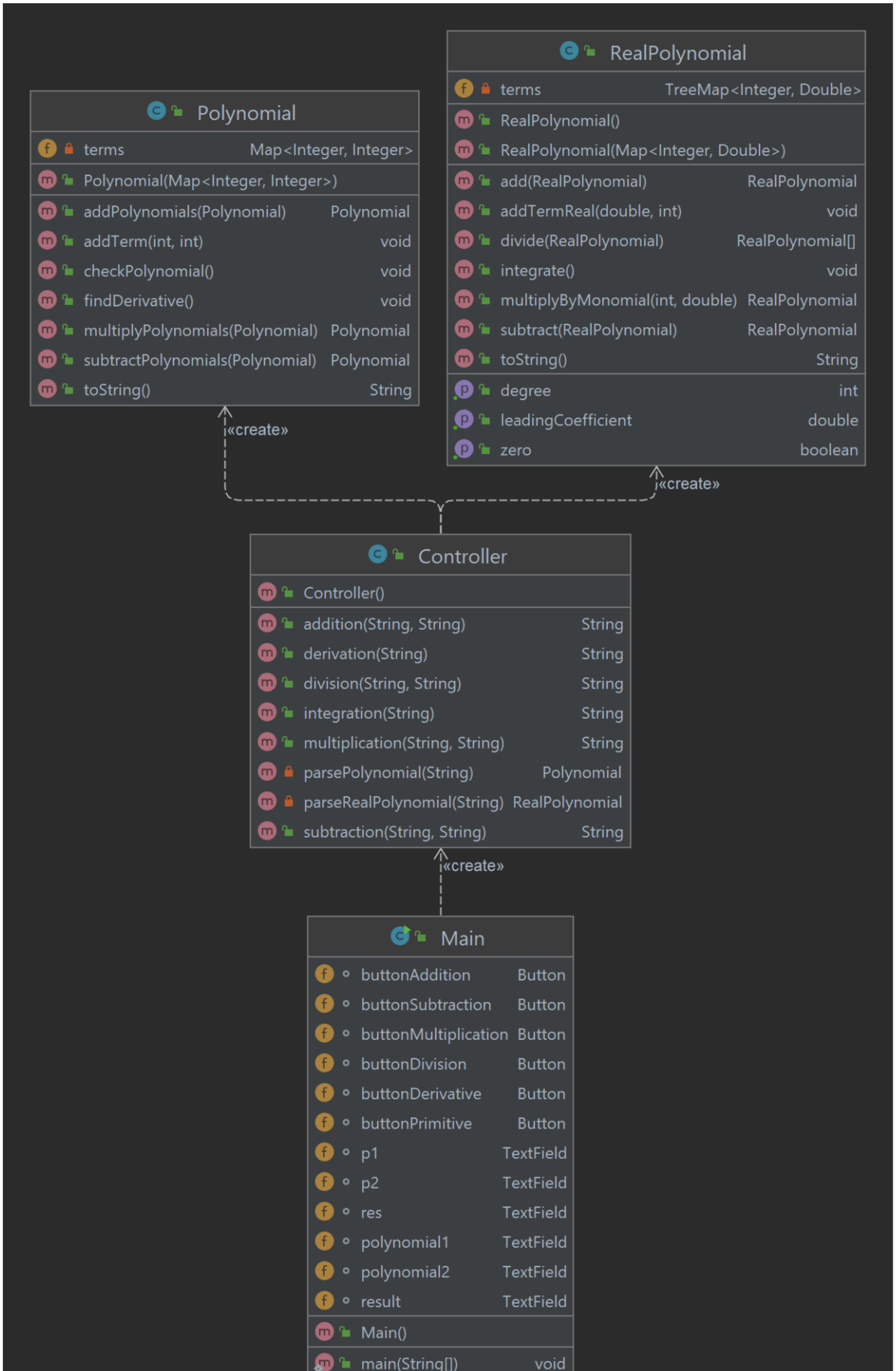
The application follows the Model-View-Controller (MVC) architectural pattern, which separates the application logic into three interconnected components:

Model: The Model component represents the data and business logic of the application. It includes the Polynomial and RealPolynomial classes, which represent a polynomial function in integer and real numbers, and provides methods for manipulating polynomials such as addition, subtraction, multiplication, and division.

View: The View component is responsible for displaying the data from the Model component to the user and handling user input. It includes the graphical user interface (GUI) elements of the application, such as text boxes, buttons, and menus.

Controller: The Controller component acts as an intermediary between the View and Model components, and handles user input and updates the Model accordingly. It includes the event listeners and handlers for the GUI elements, as well as the algorithms for performing operations on polynomials.

Overall, the OOP design of the application emphasizes encapsulation, modularity, and separation of concerns. By separating the data, logic, and presentation components of the application, it becomes easier to maintain, extend, and reuse the code, and makes the application more robust and scalable.


Polynomial
terms Map<Integer, Integer>
Polynomial(Map<Integer, Integer>)
addPolynomials(Polynomial) Polynomial
addTerm(int, int) void
checkPolynomial() void
findDerivative() void
multiplyPolynomials(Polynomial) Polynomial
subtractPolynomials(Polynomial) Polynomial
toString() String
RealPolynomial
terms TreeMap<Integer, Double>
RealPolynomial()
RealPolynomial(Map<Integer, Double>)
add(RealPolynomial) RealPolynomial
addTermReal(double, int) void
divide(RealPolynomial) RealPolynomial[]
integrate() void
multiplyByMonomial(int, double) RealPolynomial
subtract(RealPolynomial) RealPolynomial
toString() String
degree int
leadingCoefficient double
zero boolean
«create»
«create»
Controller
Controller()
addition(String, String) String
derivation(String) String
division(String, String) String
integration(String) String
multiplication(String, String) String
parsePolynomial(String) Polynomial
parseRealPolynomial(String) RealPolynomial
subtraction(String, String) String
«create»
Main
buttonAddition Button
buttonSubtraction Button
buttonMultiplication Button
buttonDivision Button
buttonDerivative Button
buttonPrimitive Button
p1 TextField
p2 TextField
res TextField
polynomial1 TextField
polynomial2 TextField
result TextField
Main()
main(String[]) void

4. Implementation

The GUI implementation consists of a JavaFX application with a user interface that includes several buttons, text fields, and a grid layout. The GUI is designed to interact with the user and display the results of various polynomial operations.

The GUI class contains several member variables, including buttonAddition, buttonSubtraction, buttonMultiplication, buttonDivision, buttonDerivative, buttonPrimitive, p1, p2, res, polynomial1, polynomial2, and result. These variables are used to create the user interface and interact with the user.

The start() method is the entry point of the application and is called when the application is launched. In the start() method, a new instance of the Controller class is created, which is responsible for handling the user input and performing the necessary polynomial operations. The start() method creates a new JavaFX Stage object and sets its title to "Polynomial Calculator". It then creates several JavaFX Button and TextField objects and assigns them to the corresponding member variables.

Each button is assigned an action that is triggered when the user clicks on it. When a button is clicked, it calls the appropriate method in the Controller class and passes the input from the user. The result of the operation is then displayed in the "result" text field.

The text fields are used to display the input and output of the polynomial operations. The layout of the GUI is created using a GridPane object, which is a container that arranges its children in a grid.

In summary, the GUI implementation follows the Model-View-Controller (MVC) design pattern, where the Model represents the polynomial operations, the View represents the user interface, and the Controller acts as an intermediary between the Model and the View, handling user input and updating the View accordingly.

The Polynomial class is a representation of a mathematical polynomial, which is an expression consisting of variables and coefficients, involving only the operations of addition, subtraction, and multiplication. The Polynomial class contains methods for performing basic arithmetic operations on polynomials, such as adding, subtracting, and multiplying them.

The class uses a Map<Integer, Integer> to store the terms of the polynomial. In this map, the keys represent the exponents of the polynomial terms, and the values represent their coefficients. The terms are stored in a TreeMap, which is a sorted map implementation that maintains the natural ordering of its keys.

The constructor of the Polynomial class takes a Map<Integer, Integer> object as a parameter and initializes the terms field of the class with a new TreeMap object containing the entries of the parameter map.

The addTerm method adds a new term to the polynomial by updating the coefficient of an existing term with the same exponent, or by adding a new term if one does not already exist. If the resulting coefficient of a term is zero, it is removed from the polynomial.

The addPolynomials method takes another Polynomial object as a parameter and returns a new Polynomial object that represents the sum of the two polynomials.

The subtractPolynomials method takes another Polynomial object as a parameter and returns a new Polynomial object that represents the difference between the two polynomials.

The multiplyPolynomials method takes another Polynomial object as a parameter and returns a new Polynomial object that represents the product of the two polynomials.

The findDerivative method computes the derivative of the polynomial and updates the terms field with the resulting terms.

The checkPolynomial method removes any terms from the polynomial that have a coefficient of zero.

The toString method returns a string representation of the polynomial in a human-readable format.

The RealPolynomial class is a Java class that represents a polynomial function with real coefficients. It contains methods for adding, subtracting, dividing, and multiplying polynomials, as well as computing the degree and leading coefficient of a polynomial, and integrating a polynomial.

The class has a TreeMap<Integer, Double> field called terms which represents the terms of the polynomial as a map from the degree of the term to its coefficient. The class has two constructors: one with no arguments that initializes terms to an empty TreeMap, and one that takes a Map<Integer, Double> argument and initializes terms to a TreeMap containing the same key-value pairs.

The RealPolynomial class has the following methods:

addTermReal(double coefficient, int exponent): adds a term with the given coefficient and exponent to the polynomial. If the coefficient is zero, the term is not added to the polynomial.
getDegree(): returns the degree of the polynomial, which is the highest degree of any term in the polynomial. If the polynomial is empty, it returns 0.
getLeadingCoefficient(): returns the coefficient of the term with the highest degree in the polynomial. If the polynomial is empty, it returns 0.
add(RealPolynomial other): returns a new RealPolynomial object that represents the sum of the current polynomial and another polynomial.

divide(RealPolynomial divisor): returns an array of two RealPolynomial objects that represent the quotient and remainder of dividing the current polynomial by another polynomial. If the divisor is zero, it throws an ArithmeticException with the message "Division by zero".
isZero(): returns true if the polynomial is zero (i.e., has no terms), false otherwise.
multiplyByMonomial(int degree, double coefficient): returns a new RealPolynomial object that represents the product of the current polynomial and a monomial with the given degree and coefficient.
subtract(RealPolynomial other): returns a new RealPolynomial object that represents the difference between the current polynomial and another polynomial.
integrate(): modifies the current polynomial to represent its antiderivative (i.e., the result of integrating the polynomial with respect to its variable x).
toString(): returns a string representation of the polynomial, in the form "a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0", where a_i is the coefficient of the term of degree i. If a coefficient is 1, it is omitted (except for the constant term), and if a coefficient is negative, it is written with a minus sign.

The Controller class is responsible for handling the logic of the polynomial operations in the polynomial calculator. It has several public methods that represent the different operations that can be performed on polynomials, namely addition, subtraction, multiplication, division, differentiation, and integration. Each method takes two polynomial strings as inputs, which are then parsed into Polynomial or RealPolynomial objects using the private parsePolynomial and parseRealPolynomial methods.

The addition, subtraction, and multiplication methods parse the two polynomial strings and call the respective methods of the Polynomial class to perform the corresponding operations. The resulting Polynomial object is then converted to a string and returned as the result of the operation.

The division method is slightly different, as it requires the use of the RealPolynomial class to handle polynomials with real coefficients. It first parses the input polynomial strings into RealPolynomial objects, then calls the divide method of the RealPolynomial class to perform polynomial division. The method returns the quotient as a string, and the remainder is printed to the console.

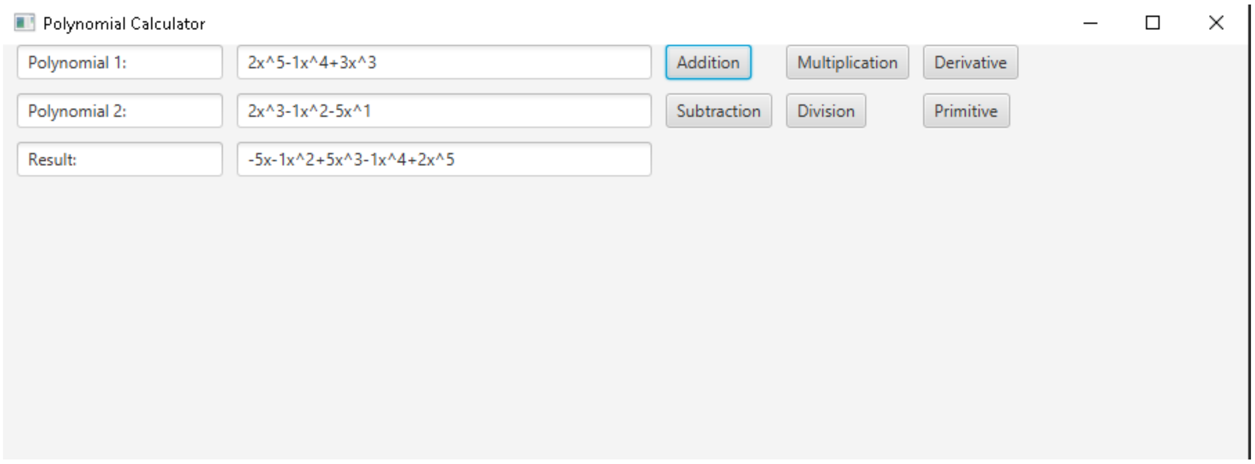
The derivation method parses the input polynomial string into a Polynomial object, then calls the findDerivative method of the Polynomial class to compute the derivative of the polynomial. The resulting Polynomial object is then checked for any errors using the checkPolynomial method of the Polynomial class, and returned as a string.

The integration method is similar to the derivation method, but uses the RealPolynomial class instead. It parses the input polynomial string into a RealPolynomial object, then calls the integrate method of the RealPolynomial class to compute the integral of the polynomial. The resulting RealPolynomial object is then converted to a string and returned as the result.

The parsePolynomial and parseRealPolynomial methods are private helper methods that use regular expressions to parse the polynomial strings into Polynomial and RealPolynomial objects, respectively. They loop through the polynomial string, matching each monomial (a term in the polynomial) using regular expressions, and adding it to the Polynomial or RealPolynomial object as a term with the corresponding coefficient and exponent.

Overall, the Controller class serves as the intermediary between the user interface and the polynomial classes, handling the user input, calling the appropriate methods, and returning the results as strings.

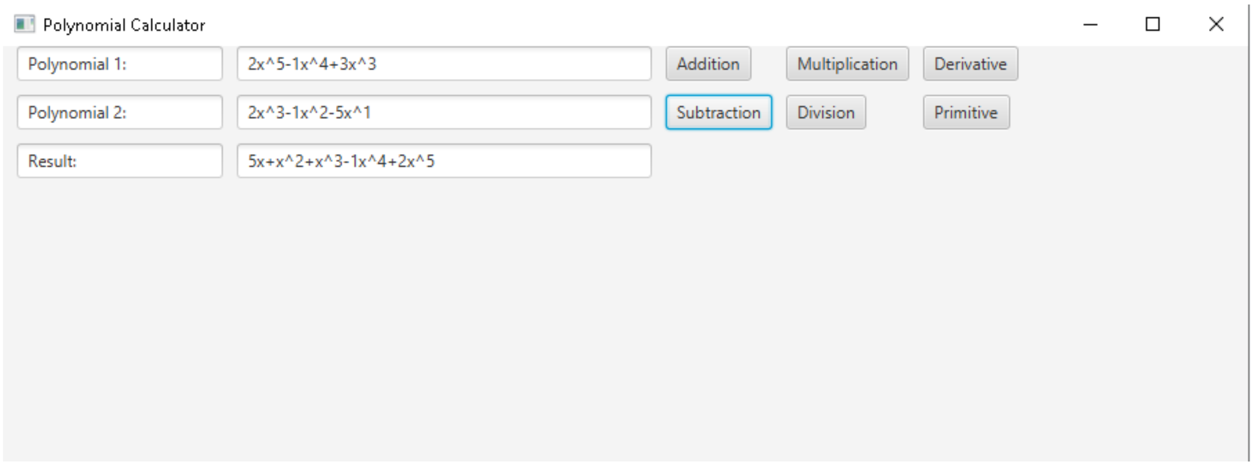
5. Results



Polynomial 1: 2x^5-1x^4+3x^3

Polynomial 2: 2x^3-1x^2-5x^1

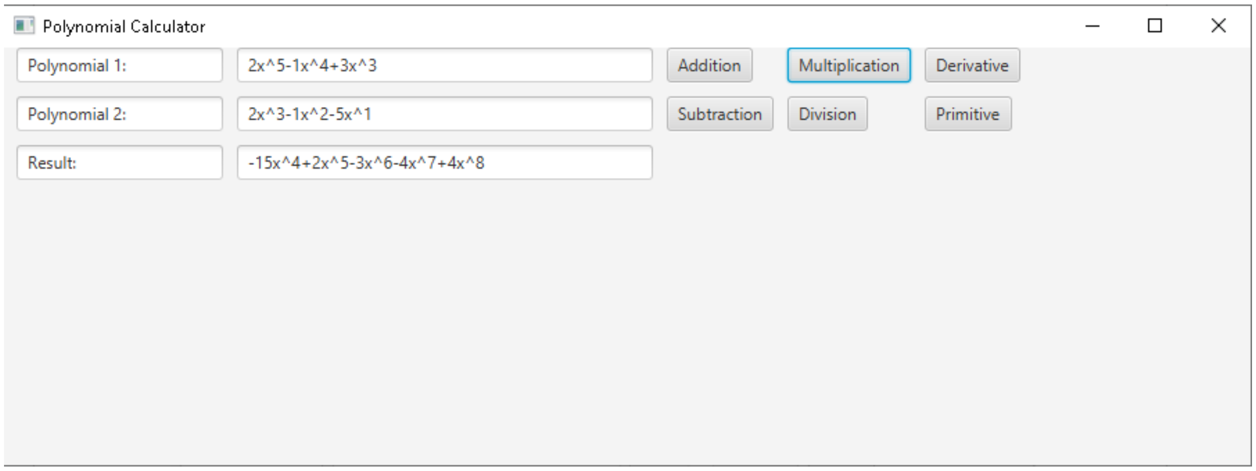
Result: -5x-1x^2+5x^3-1x^4+2x^5



Polynomial 1: 2x^5-1x^4+3x^3

Polynomial 2: 2x^3-1x^2-5x^1

Result: 5x+x^2+x^3-1x^4+2x^5

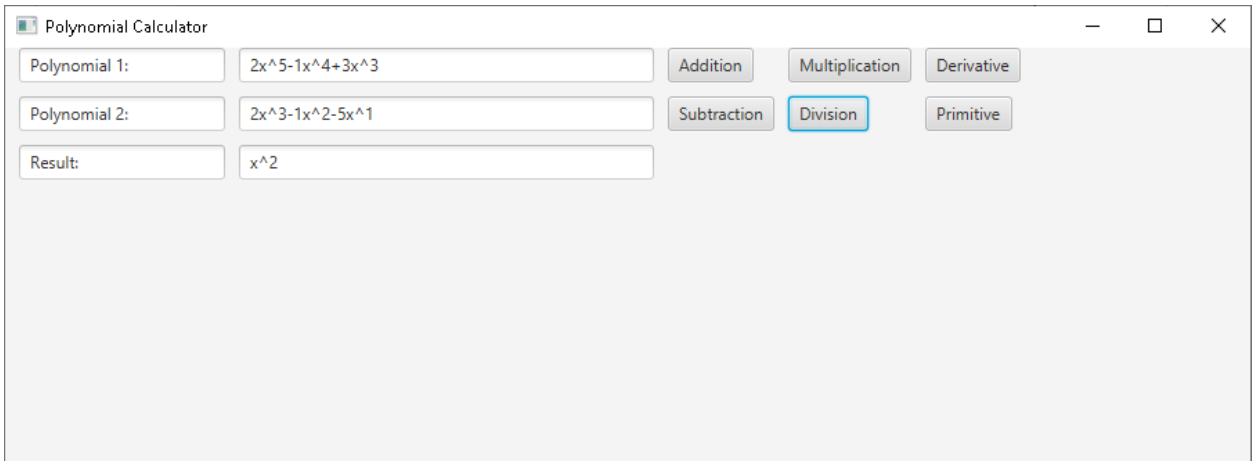


Polynomial Calculator

Polynomial 1: 2x^5-1x^4+3x^3

Polynomial 2: 2x^3-1x^2-5x^1

Result: -15x^4+2x^5-3x^6-4x^7+4x^8

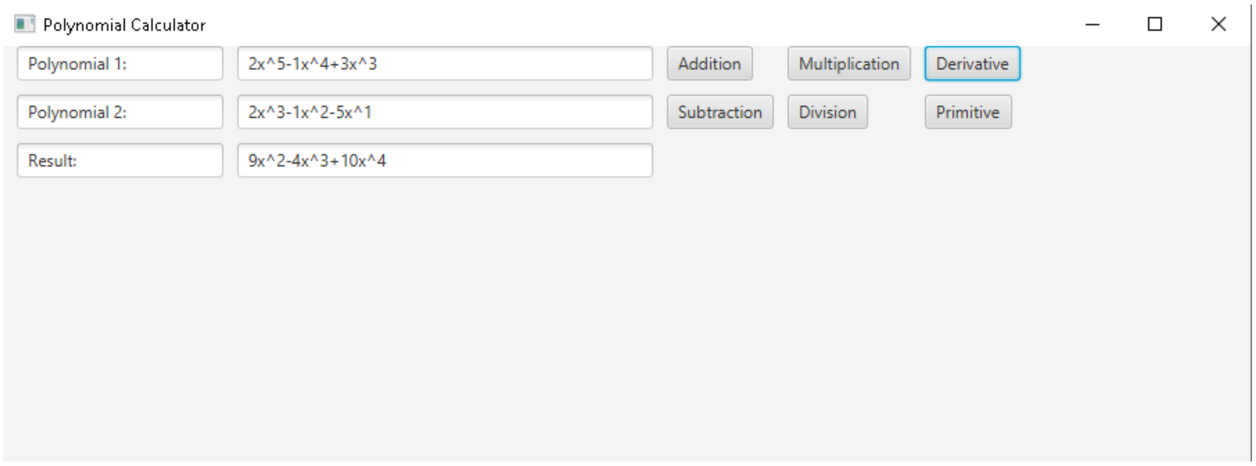


Polynomial Calculator

Polynomial 1: 2x^5-1x^4+3x^3

Polynomial 2: 2x^3-1x^2-5x^1

Result: x^2

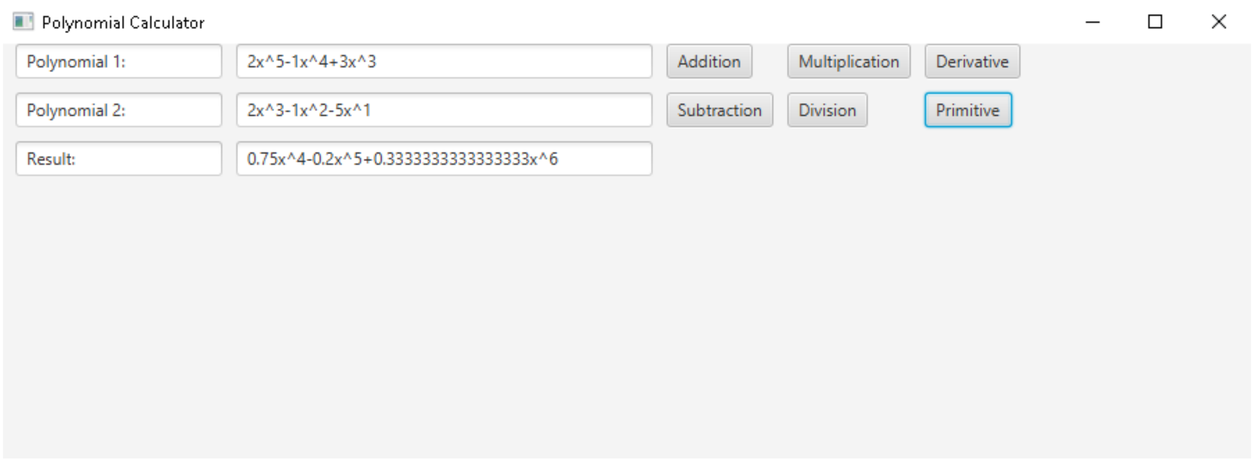


Polynomial Calculator

Polynomial 1: 2x^5-1x^4+3x^3

Polynomial 2: 2x^3-1x^2-5x^1

Result: 9x^2-4x^3+10x^4



Polynomial 1: 2x^5-1x^4+3x^3

Polynomial 2: 2x^3-1x^2-5x^1

Result: 0.75x^4-0.2x^5+0.3333333333333333x^6

6. Conclusions

In conclusion, the polynomial calculator project was successfully completed with a dedicated graphical user interface that allows the user to input polynomial expressions, select a mathematical operation, and view the result. The polynomial calculator is able to handle polynomials with one variable and integer coefficients and implements six different mathematical operations: addition, subtraction, multiplication, division, differentiation, and integration.

During the development of this project, we learned about programming concepts such as object-oriented programming, parsing input strings, and using graphical user interface libraries. We also gained a deeper understanding of mathematical operations on polynomials, including polynomial representation, addition, subtraction, multiplication, division, differentiation, and integration.

Future developments for the polynomial calculator could include expanding the functionality to handle polynomials with complex coefficients, adding support for multiple variables, and improving the graphical user interface. Additionally, the program could be optimized for performance by implementing more efficient algorithms for the mathematical operations.

Overall, the polynomial calculator project provided a valuable learning experience and demonstrated the power of programming and computational tools in solving mathematical problems.

7. Bibliography

The references that were consulted by the student during the implementation of the homework will be added.
Example:

1. *HashMap in Java* - https://stackoverflow.com/questions/16819368/hashmap-in-java
2. *What are Java classes?* - www.tutorialspoint.com