Polynomial Calculator Documentation

1. **Assignment objective**

Design and implement a system for polynomial computations. The polynomials have one variable and integer coefficients. A graphical user interface is also required. Use pattern matching and regular expressions for the user input. We must implement the following operations:

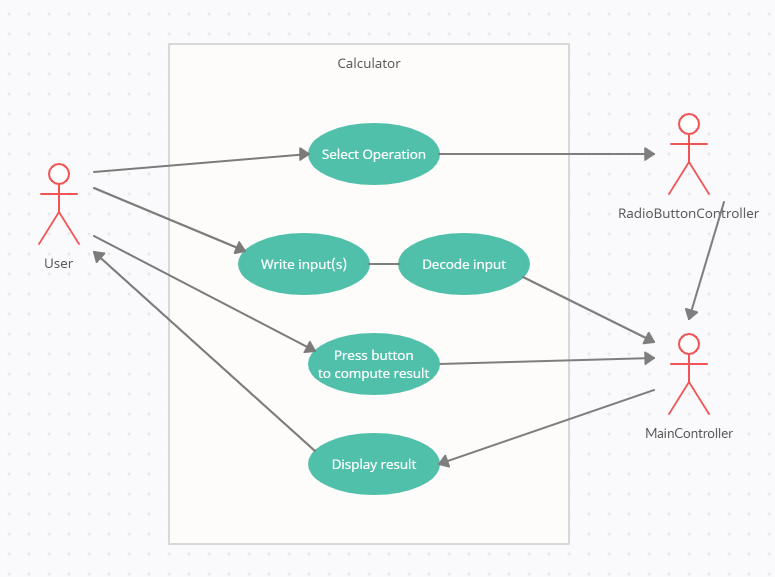
* Addition
* Subtraction
* Multiplication
* Division
* Derivation
* Integration

Steps:

* Create the monomial and polynomial classes
* Implement the mentioned operations
* Create the graphical interface
* Implement controllers to communicate between the frontend and backend
* Use pattern matching to extract the polynomial from the user input

1. **Problem analysis, modeling, scenarios, use cases**

**Use-case diagram:**



The actor is the User, which has to complete 3 actions:

1. He chooses the operation to be performed, which is taken by the RadioButtonController and converted to usable data.
2. He has to write the polynomial(s) in the text box(es), from where the string is converted to usable data using regex and sent to the MainController.
3. He has to push the button to calculate, which tells MainController to use the polynomials and the operation from the other controller in order to compute the result

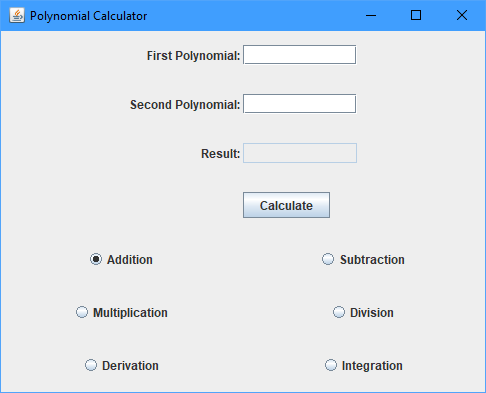
The result is shown to the User in a noneditable text box.

Depending on the chosen operation, a different number of input boxes or results will be shown and a different operation function will be called to compute the result.

Division gives us two results, which will be shown together in the result text box, the first one being the quotient, and the second one being the remainder. For derivation and integration, only one polynomial is needed, so the second text box and label will be hidden.

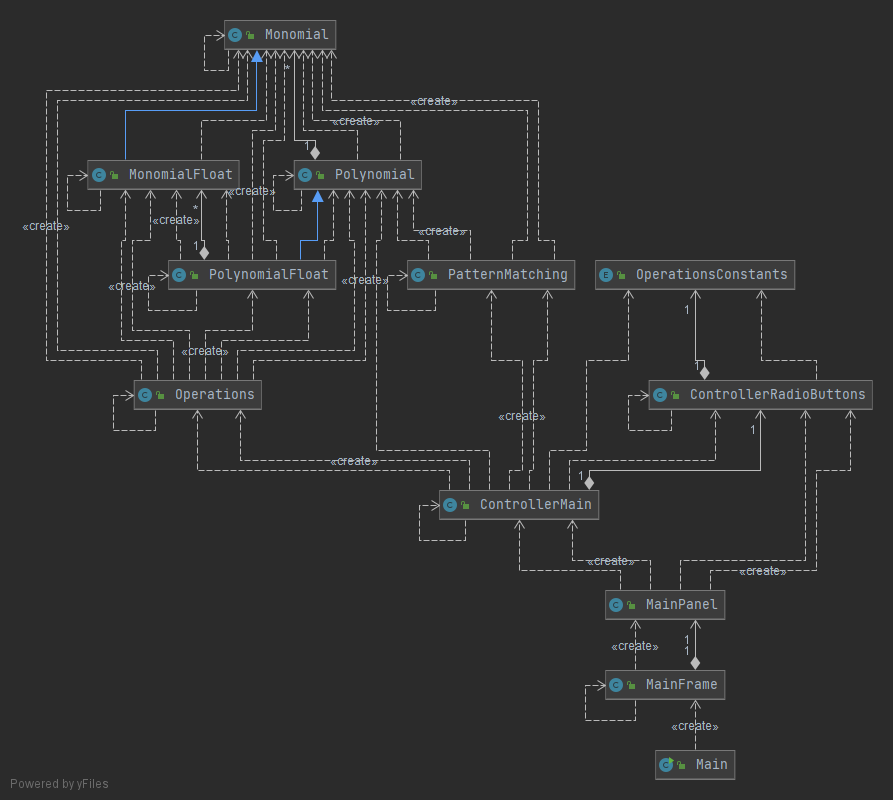
1. **Design**

**Graphical User Interface:**

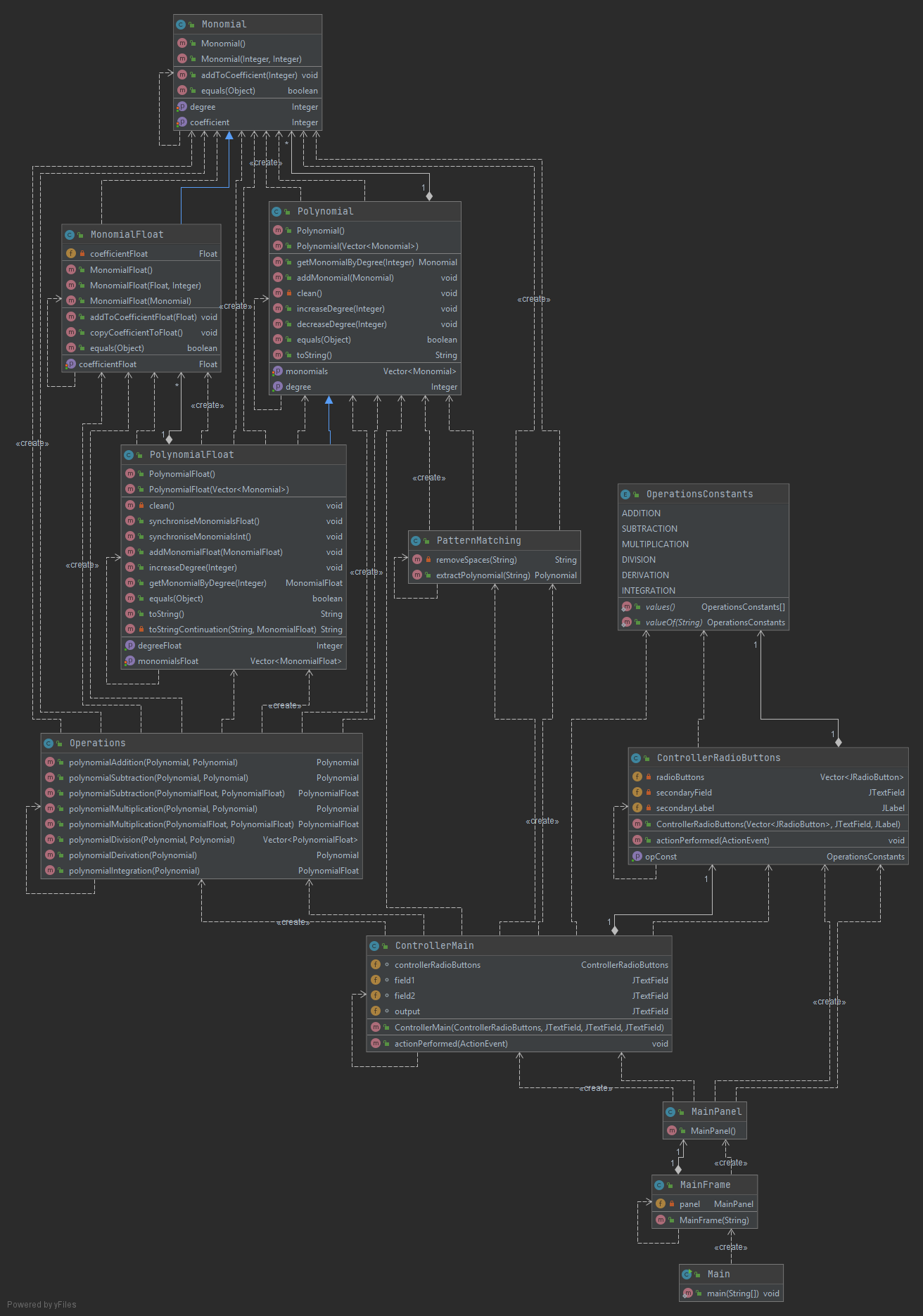


The GUI is user-friendly, it has two input text boxes and one output textbox that shows the result. The operation is chosen with the radio buttons and the result is updated only after “Calculate” button is pressed.

**UML Diagram:**



And the one that includes all the details:



**Data Structures:**

For data storage, I used primitive wrappers (like Integer, String, Float etc.) instead of raw primitive types, Vectors instead of arrays, and new classes such as Monomial and Polynomial.

Vectors are better than arrays because they implement List and Iterable interfaces, which makes them easier to work with. They allocate space dynamically, which is a great plus, and they are a RandomAccess Collection, which gives it all the good things that an array has.

Polynomial has only one variable, a Vector of Monomials, because the degree can be calculated from the size of the Vector. Monomial has an Integer coefficient and Integer degree.

PolynomialFloat and MonomialFloat are variants of Polynomial and Monomial adapted for non-Integer coefficients. They both extend their base class, PlynomialFloat also containing a Vector of MonomialFloat, which contains an extra variable: Float coefficientFloat.

**Packages:**

This project respects the OOP recommended design pattern, model-view-controller (MVC). Model contains all the logic that runs behind a GUI (basically the backend), View contains the elements of the GUI (the frontend), and Controller communicates between the two and applies some extra logic if needed. All of these are contained in the source folder (Application).

In my case, the packages are as follows:

* Application – contains the Main class, used to run the whole thing, and all the other packages
* Model – is comprised of 2 packages, DataModels and BussinessLogic, the first containing classes used for storing and managing data, and the second containing classes that process the data.
* View – this is the GUI package. It contains classes for the graphic interface.
* Control – called Controllers in my case, contains the controllers that connect the GUI with the logic

**Algorithms:**

The only notable algorithms apart from the operations are the pattern matching algorithm and the Monomial insertion.

Pattern matching is implemented by the class with the same name from BussinessLogic, in the sole public method extractPolinomials(). This takes the input string and matches it with the regular expression pattern “(([-]?[0-9]+)?[\*]?([-])?([a-zA-Z])?[\^]?([0-9]+)?)”. It will extract 5 groups for each match, the first being the expression as a whole (I think that I extract it twice for no reason, but it is a bit late to fix that), which is not very important, the second one is the signed coefficient, the third one is for a special case in which the coefficient is missing (equal to 1) and the sign is “-“ (for example “-x”), the fourth one is the letter that represents the variable and the fifth group is the power of the monomial. The main 3 groups are the second (the coefficient), the fourth (x) and the fifth (the degree).

From these groups we ca solve the following cases:

“-22\*x^2” and “-22x^2” - none of the main 3 groups are null => the coefficient is -22 and the degree is 2

“x^2” – the coefficient and the second (“-” group) groups are null => the coefficient is 1

“-x^2” – the coefficient group is null, but the second one is not => the coefficient is -1

“2x” – the degree group is null => the degree is 1

“-7” – the variable group is null => the degree is 0

And other combinations that respect the mathematical writing. Also, spaces don’t matter because they are removed before working with the string, “\*” is optional and “+” is not required if the coefficient is negative (“x^2+-2x” works but unnecessary +). Alternatively, you can write any nonalphanumeric character instead of “+” because it only works as a delimiter (apart from “\*-^” and spaces).

Monomial insertion in a polynomial uses a relatively simple, but very important algorithm. The monomials are not simply inserted at the end of the vector, instead they are placed on the position that matches their degree. If the degree of the polynomial is smaller than the monomial, the vector is supplemented with monomials having the coefficient 0 and the degree that matches the position, until the position of the monomial is reached. Monomials added to the vector do not overwrite the predecessor of their position, instead the coefficient is added to the previous one, which is 0 under normal circumstances but it can be another number, a behavior that makes polynomial addition trivial. While this kind of data storage is not very efficient in the case of a monomial with a high degree and nothing else, we can’t have more than 4 polynomials at all times, therefore on a small-size project like ours the pros outweigh the cons by a large margin.

The operations are straight forward. Addition only uses the aforementioned method to insert all monomials from one polynomial into the other and subtraction negates the coefficient when inserting. Multiplication takes each monomial from one polynomial and multiplies it with each monomial from the other, multiplying the coefficients and adding the degrees, and collecting all the results in one polinomial. Division uses the algorithm described at the laboratory. Derivation multiplies the coefficients with the degree, then shifts the monomials to the smaller position while decreasing all the degrees by one. Integration shifts the monomials to a higher position, increasing the degrees, and then divides the coefficient by the new degree. Of course, none of the polynomials given as parameters are modified (a good practice in most cases), instead a new one is created, filled and returned.

1. **Implementation**

BussinessLogic package - Contains the logic of the backend

**Classes:**

* **public class Operations**

Implements polynomial operations: addition, subtraction, multiplication, division, derivation, integration.

**Methods:**

* public Polynomial polynomialAddition(Polynomial polynomial1, Polynomial polynomial2):

Takes two Polynomial parameters and returns their sum.

* public Polynomial polynomialSubtraction(Polynomial polynomial1, Polynomial polynomial2):

Takes two Polynomial parameters and returns the difference between the first and the second.

* public Polynomial polynomialMultiplication(Polynomial polynomial1, Polynomial polynomial2):

Takes two Polynomial parameters and returns their product.

* public Vector<PolynomialFloat> polynomialDivision(Polynomial polynomial1, Polynomial polynomial2):

Takes two Polynomial parameters and returns the result of the first divided by the second. The returned vector contains the quotient on the first position and the remainder on the second. It uses the algorithm described at the laboratory, with the addition that is checks if the second parameter is "polynomial 0".

* public Polynomial polynomialDerivation(Polynomial polynomial):

Takes one Polynomial parameter and returns its derivative.

* public PolynomialFloat polynomialIntegration(Polynomial polynomial):

Takes one Polynomial parameter and returns its integral.

* private PolynomialFloat polynomialSubtraction(PolynomialFloat polynomial1, PolynomialFloat polynomial2):

Takes two PolynomialFloat parameters and returns the difference between the first and the second. It overloads the normal polynomialSubtraction() and does the same thing, but it works with PolynomialFloat. It is only needed for polynomialDivision(), therefore it is private.

* private PolynomialFloat polynomialMultiplication(PolynomialFloat polynomial1, PolynomialFloat polynomial2):

Takes two PolynomialFloat parameters and returns their product. It overloads the normal polynomialMultiplication() and does the same thing, but it works with PolynomialFloat. It is only needed for polynomialDivision(), therefore it is private.

* public class PatternMatching

Implements pattern matching of regular expressions.

**Methods:**

* private String removeSpaces(String input)

Takes a string and returns a copy without whitespaces.

* public Polynomial extractPolynomial(String input)

Takes a string and returns the polynomial described by it. It uses regex to extract every Monomial from the string and add them together in a Polynomial.

DataModels package – Contains the classes that store information

**Classes:**

* public class Monomial

Stores the coefficient and degree of a monomial.

**Attributes:**

* private Integer coefficient;
* private Integer degree;

**Methods:**

Setters and Getters and 2 constructors:

* public Monomial()
* public Monomial(Integer coefficient, Integer degree)
* public void addToCoefficient(Integer coefficient)

Adds an Integer to the coefficient.

* public class MonomialFloat extends Monomial

Float version of Monomial. Apart from the variables inherited from Monomial, it also stores a float coefficient.

**Attributes:**

* private Float coefficientFloat;

**Methods:**

Setters and Getters and 2 constructors:

* public MonomialFloat(Monomial monomial)
* public MonomialFloat(Float coefficient, Integer degree)
* public void addToCoefficientFloat(Float coefficientFloat)

Adds a Float to the coefficientFloat. Basically the Float version of addToCoefficient().

* public class Polynomial

Stores a vector of Monomials, with each monomial's position corresponding to their degree.

**Attributes:**

* private Vector<Monomial> monomials;

**Methods:**

Setters and Getters and 2 constructors:

* public Polynomial()
* public Polynomial(Vector<Monomial> vec)
* public Integer getDegree()

Gets the degree of the polynomial, using the size of the monomials vector (degree = monomials.size() – 1).

* public Monomial getMonomialByDegree(Integer degree)

Takes the Integer degree and returns the Monomial with that degree.

* public void addMonomial(Monomial monomial)

Adds a Monomial to the monomials Vector. The monomials are not simply inserted at the end of the vector, instead they are placed on the position that matches their degree. If the degree of the polynomial is smaller than the monomial, the vector is supplemented with monomials having the coefficient 0 and the degree that matches the position, until the position of the monomial is reached. Monomials added to the vector do not overwrite the predecessor of their position, instead the coefficient is added to the previous one.

* private void clean()

Reduces the size of the Polynomial to match its true degree

* public void increaseDegree(Integer howMuch)

Increases the degrees of all Monomials by the Integer parameter and shifts them to their new spots.

* public void decreaseDegree(Integer howMuch)

Decrease the degrees of all Monomials by the Integer parameter and shifts them to their new spots.

* public class PolynomialFloat extends Polynomial

Float version of Polynomial. Apart from the vector inherited from Polynomial, it also stores a MonomialFloat vector, with each monomialFloat's position corresponding to their degree.

**Attributes:**

* private Vector<MonomialFloat> monomialsFloat;

**Methods:**

Setters and Getters and 2 constructors:

* public PolynomialFloat()
* public PolynomialFloat(Vector<Monomial> vec)
* public Integer getDegreeFloat()

Gets the degree of the polynomial, using the size of the monomialsFloat vector (degree = monomialsFloat.size() – 1)

* public void addMonomialFloat(MonomialFloat monomialFloat)

Adds a MonomialFloat to the monomials Vector. Float version of addMonomial, does the same thing but with MonomialFloats in monomialsFloat vector.

* private void clean()

Reduces the size of the Polynomial to match its true degree, Float version.

* public void synchroniseMonomialsFloat()

Synchronises monomialsFloat with monomials. Because monomials and monomialsFloat are not bound by one another, they need to be synchronised when a change happened in one of them. This method copies all Monomials from monomials to monomialsFloat as MonomialFloats.

* public void synchroniseMonomialsInt()

Synchronises monomials with monomialsFloat. Because monomials and monomialsFloat are not bound by one another, they need to be synchronised when a change happened in one of them. This method copies all MonomialFloats from monomialsFloat to monomials as Monomials

* @Override public void increaseDegree(Integer howMuch)

Increases the degrees of all MonomialFloats by the Integer parameter and shifts them to their new spots. Float version of increaseDegree() from super.

* @Override public MonomialFloat getMonomialByDegree(Integer degree)

Takes the Integer degree and returns the MonomialFloat with that degree. Float version of getMonomialByDegree() from super.

* public enum OperationsConstants {ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION, DERIVATION, INTEGRATION}

Enumeration for operations.

Controllers package – Contains the controllers that connect the backend to the frontend

**Classes:**

* public class ControllerMain implements ActionListener

The main controller, assigned to the "Calculate" JButton. It takes input from the two JTextFields and the operation from the other controller, computes the result and shows it in the output JTextField.

**Attributes:**

* ControllerRadioButtons controllerRadioButtons;
* JTextField field1;
* JTextField field2;
* JTextField output;

**Methodes:**

* public ControllerMain(ControllerRadioButtons controllerRadioButtons, JTextField field1, JTextField field2, JTextField output)

The constructor.

* @Override public void actionPerformed(ActionEvent e)

Prints the result when the button is pressed. It takes input from the two JTextFields and converts it to Polynomials, then it takes the operation from the other controller, computes the result with an Operations instance and shows it in the output JTextField.

* public class ControllerRadioButtons implements ActionListener

Controller for all the JRadioButtons. It chooses the OperationsConstants that denotes the operation to be performed. Also, it hides the second input label and text box when not needed.

**Attributes:**

* private Vector<JRadioButton> radioButtons;
* private OperationsConstants opConst;
* private JTextField secondaryField;
* private JLabel secondaryLabel;

**Methods:**

* public ControllerRadioButtons(Vector<JRadioButton> radioButtons, JTextField secondaryField, JLabel secondaryLabel)

The constructor.

* public OperationsConstants getOpConst()

A getter.

* @Override public void actionPerformed(ActionEvent e)

Chooses the OperationsConstants when a radio button is clicked. Also, it hides the second input label and text box when not needed.

GUI package – implements the GUI

**Classes:**

* public class MainFrame extends JFrame

Main frame. Completes some initializations for the GUI. Creates the panel where all the interactions will take place.

**Methods:**

* public MainFrame(String title)

The constructor.

* public class MainPanel extends JPanel

The main panel that contains all the GUI elements. Creates the elements, sets their positions and binds them to their respective controllers.

**Methodes:**

* public MainPanel()

The constructor.

“Root” package – contains all the other packages and the Main class

**Classes:**

* public class Main

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

/\*\*

\* Starts a thread for the Swing graphical interface and completes the setup for the frame.

\*/

@Override

public void run() {

JFrame frame = new MainFrame("Polynomial Calculator");

frame.setSize(500, 400);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

});

}

1. **Results**

Test classes have been implemented using Junit (OperationsTest, PatternMatchingTest). They contain tests for all operations with polynomials and pattern matching for the input.

polynomialAdditionTest1(): -x^2+2\*x+1 + x^2+x = 3\*x+1

polynomialAdditionTest2(): 4\*x^5-3\*x^4+x^2-8\*x+1 + 3\*x^4-x^3+x^2+2\*x-1 = 4\*x^5-x^3+2\*x^2-6\*x

polynomialSubtractionTest1(): x^2+2\*x+1 - x^2+x = x+1

polynomialSubtractionTest2(): 4\*x^5-3\*x^4+x^2-8\*x+1 - 3\*x^4-x^3+x^2+2\*x-1 = 4\*x^5-6\*x^4+x^3-10\*x+2

polynomialMultiplicationTest(): 3\*x^2-x+1 \* x-2 = 3\*x^3-7\*x^2+3\*x-2

polynomialDivisionTest1(): x^3-2\*x^2+6\*x-5 : x^2-1 = x-2 remainder: 7\*x-7

polynomialDivisionTest2(): 2\*x^2+4\*x-2 : 2 = x^2+2\*x-1 remainder: 0

polynomialDerivationTest(): ( x^3-2\*x^2+6\*x-5 )' = 3\*x^2-4\*x+6

polynomialIntegrationTest(): S( x^3+4\*x^2+5 )dx = 1/4\*x^4+4/3\*x^3+5\*x

extractPolynomialTest(): "24\*X^4+x3 -7a ^23- 8x+7" => -7\*x^23+24\*x^4+x^3-8\*x+7

In the graphical interface, the results have the correct values for any polynomials tested. Division by 0 is handled by returning a null result for quotient and remainder. When the coefficient is not an integer (in division or integration), it will be shown as a float, but all floats with integer values are displayed as integers.

1. **Conclusion**

There were a lot of ways to implement the classes and methods and choosing one after comparing the pros and cons of all the others was an interesting experience that will come in handy. When it comes to the data models, maintaining the lists in order at all times proved to be a very advantageous decision and it showed me that making something good from the start will ease your work by a lot.

I also learnt how to make a proper graphical user interface in swing, since I had very little experience before. Also, I was reminded how useful debugging tools are.

When it comes to further improvements, I have a few things on my mind:

* Add an input filter / validator to refuse wrong inputs
* Improve the GUI (make it beautiful)
* Store coefficients as floating point (integer coefficients was a horrible restriction when it comes to division)
* Add more operations
* Clean the code up a bit

1. **Bibliography**

<https://www.youtube.com/watch?v=jUEOWVjnIR8> – very nice tutorial series for java swing

<https://docs.oracle.com/en/> - documentations

<https://www.geeksforgeeks.org/> - quick searches for small dilemmas

<https://regex101.com/> - regular expressions testing