DOCUMENTATION

PROJECT 1

NAME OF STUDENT: Jakab-Gyik Sarolta
GROUP: 30423

TABLE OF CONTENTS

1. Objectives

Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation (i.e., addition, subtraction, multiplication, division, derivative, integration) to be performed and view the result. We consider the polynomials of one variable and integer coefficients.

Sub-objectives:

- Analyze the problem and identify requirements – in chapter 2 of Documentation
- Design the polynomial calculator – chapter 3 of Documentation
- Implement the polynomial calculator – chapter 4 of Documentation
- Test the polynomial calculator -chapter 5 of Documentation

2. Problem analysis, modelling, scenarios, cases of utility

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 polynomial calculator should subtract two polynomials
The polynomial calculator should multiply two polynomials
The polynomial calculator should divide two polynomials
The polynomial calculator should derivate one polynomial
The polynomial calculator should integrate one polynomial
The polynomial calculator should output the result of the operations
The polynomial calculator should close on left clicking the upper right red button.

Use Case: *add polynomials*
Primary Actor: user
Main Success Scenario:
1. The user inserts the 2 polynomials in the graphical user interface.
2. The user clicks on the “add” button
3. The polynomial calculator performs the addition of the two polynomials and displays the result
Alternative Sequence: Incorrect polynomials - The user inserts incorrect polynomials – A window pops up indicating the wrong input - The scenario returns to step 1

Use Case: *subtract polynomials*
Primary Actor: user
Main Success Scenario:
1. The user inserts the 2 polynomials in the graphical user interface.
2. The user clicks on the “subtract” button
3. The polynomial calculator performs the subtraction of the two polynomials and displays the result
Alternative Sequence: Incorrect polynomials - The user inserts incorrect polynomials – A window pops up indicating the wrong input - The scenario returns to step 1

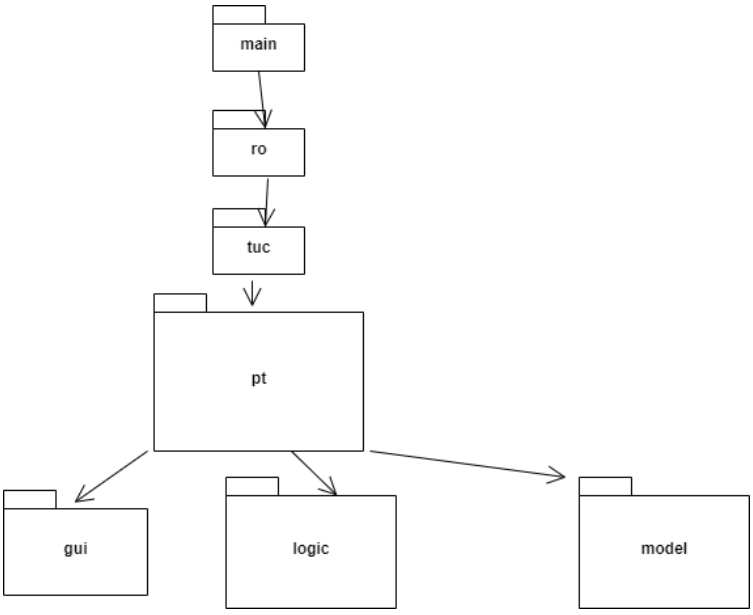
Use Case: *multiply polynomials*
Primary Actor: user
Main Success Scenario:
1. The user inserts the 2 polynomials in the graphical user interface.
2. The user clicks on the “multiply” button
3. The polynomial calculator performs the multiplication of the two polynomials and displays the result

Alternative Sequence: Incorrect polynomials - The user inserts incorrect polynomials – A window pops up indicating the wrong input - The scenario returns to step 1

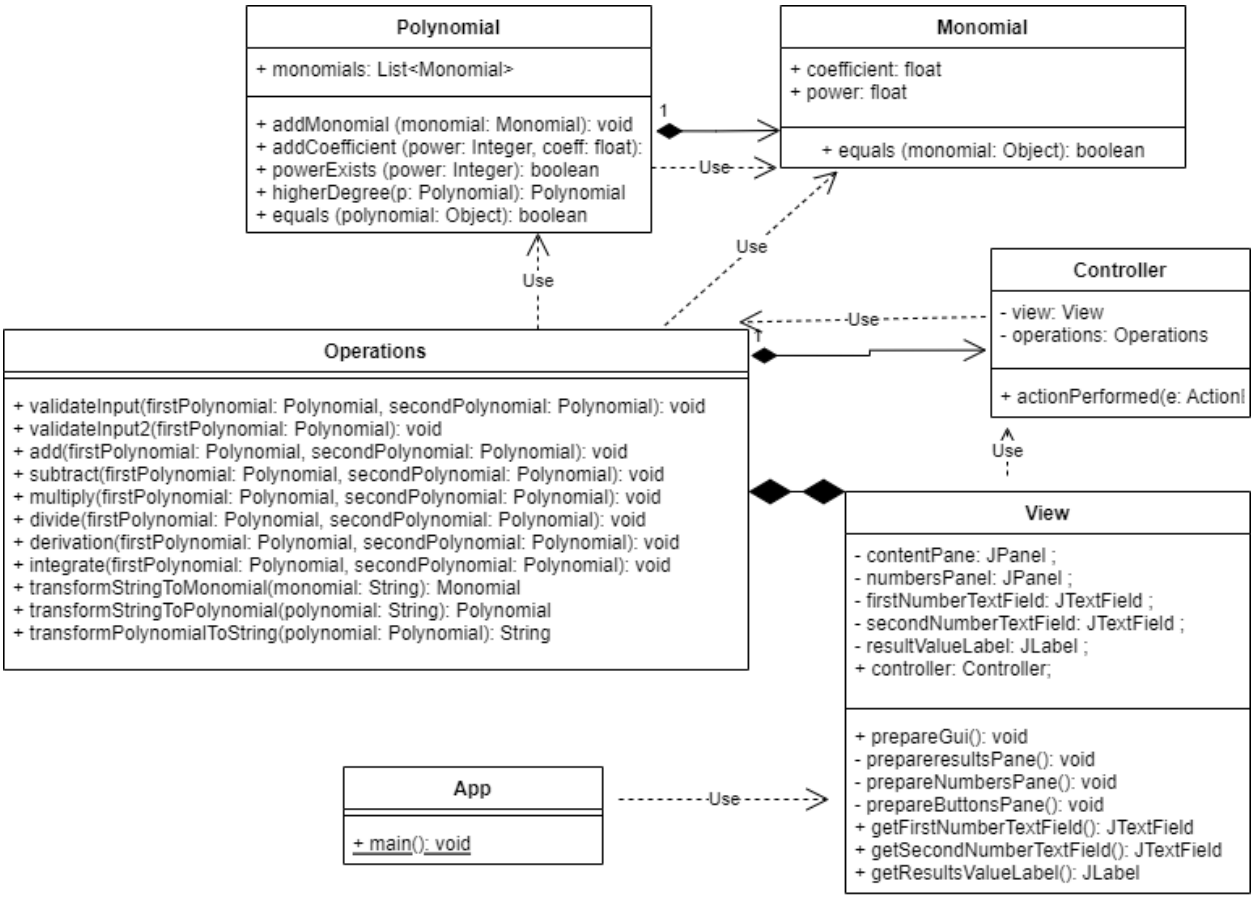
Use Case: *divide polynomials*
Primary Actor: user
Main Success Scenario:
1. The user inserts the 2 polynomials in the graphical user interface.
2. The user clicks on the "divide" button
3. The polynomial calculator performs the division of the two polynomials and displays the result
Alternative Sequence: Incorrect polynomials - The user inserts incorrect polynomials – A window pops up indicating the wrong input - The scenario returns to step 1

Use Case: *derivate polynomial*
Primary Actor: user
Main Success Scenario:
1. The user inserts 1 polynomial in the graphical user interface, in the first polynomial test field
2. The user clicks on the "derivation" button
3. The polynomial calculator performs the derivation of the polynomial and displays the result
Alternative Sequence: Incorrect polynomial - The user inserts incorrect polynomial – A window pops up indicating the wrong input - The scenario returns to step 1

Use Case: *integrate polynomial*
Primary Actor: user
Main Success Scenario:
1. The user inserts 1 polynomial in the graphical user interface, in the first polynomial test field
2. The user clicks on the "integrate" button
3. The polynomial calculator performs the integration of the polynomial and displays the result
Alternative Sequence: Incorrect polynomial - The user inserts incorrect polynomial – A window pops up indicating the wrong input - The scenario returns to step 1



~Package diagram~



~Class diagram~

3. Description

The package diagram indicates the Model-View-Controller type architecture used in the realization of the project. These 3 packages are all found inside the ro.tuc.pt package. The “gui” package (aka graphical user interface) contains both the Controller and the View parts of the indicated architecture. The Controller part is found inside the Controller class, while the View inside the View class, as the names suggest.

The Model part with the created data structures, Polynomial and Monomial classes, is found inside the model package. Also, the logic package belongs to the Model part of the architecture. This is where the Operations class executes the arithmetic operations indicated by the user of the application, the operations that the Controller signals for the Model.

The application doesn’t require any specific algorithms to be implemented, only the basic core operations that one could perform with a polynomial, such as addition, subtraction, multiplication, division (a quite challenging one), derivation and integration.

The application implements the ActionListener interface. The Controller class detects the pressing of any of the operation buttons and it calls a method from the Operations class to be executed, based on the chosen command/button.

4. Implementation

About every class in details:

Monomial class

The base of every polynomial is an array of monomials, therefore, to decompose the polynomial and to work with an array list of monomials makes the computations way easier to execute. Thus, this class is in relationship both with the Polynomial class, that creates an array list of monomials, and the Operations class, which calls the different getter and setter functions of the class.

A monomial has 2 attributes: the coefficient and the power. In this project we work with integer coefficients, however, the declared type if float. When computing the integration or division, it may occur that the coefficients of the result are real numbers and thus it is more accurate to represent them as floats. The power is always an integer number.

The methods that the class implements besides the constructors and the getter-setter methods is the overridden equals. This is needed for the Junit testing, when comparing the results of different operations. The function compares all the coefficients and powers of the 2 monomials and declares equality of both attributes have the same value.

```
@Override
public boolean equals(Object o) {
    if (this.getCoefficient() == ((Monomial)o).getCoefficient() && this.getPower() == ((Monomial)o).getPower()) return true;
    return false;
}
```

Polynomial class

Creates an array list of monomials and it is the data model that represents the base of the polynomial calculator. The data is manipulated as an instance of the polynomial class, at all times.

The methods that the class implements besides the constructors and the getter-setter methods are the addMonomial, addCoefficient, powerExists, higherDegree and the overridden equals.

The addMonomial method checks if the parameter exists already in the list and if not, then it adds the monomial to the array list.

The addCoefficient method checks if the power parameters is already an existing power of a polynomial in the array list, and if it is, then it changes the coefficient of that monomial accordingly. If the new coefficient would be 0, then it removes the monomial form the array list.

```java
public void addCoefficient(Integer power, float coeff){
    boolean found = false;
    int i = 0;
    while(i < monomials.size() && !found) {
        if (monomials.get(i).getPower() == power) {
            monomials.get(i).setCoefficient(monomials.get(i).getCoefficient() + coeff);
            if (monomials.get(i).getCoefficient() == 0) {
                monomials.remove(monomials.get(i));
            }
            found = true;
        }
        i++;
    }
    if(!found){
        System.out.println("Monomial not found in the polynomial :((");
    }
}
```

The powerExists method checks the power parameter in the array list. The higherDegree methods returns the polynomial with the higher degree out of the 2 polynomials. This method is used when dividing the polynomials for example: the first step of division is checking the degree and dividing the one with the higher degree with the one with the lower degree.
The equals method calls the monomial equals method for all monomials in the array lists and checks equality in pairs. If all pairs are identical, then the 2 polynomials are equal.

```java
@Override
public boolean equals(Object polynomial) {
    for (int i = 0; i < this.getMonomials().size(); i++){
        if (!((Polynomial)polynomial).getMonomials().get(i).equals(this.getMonomials().get(i))) return false;
    }
    return true;
}
```

Operations class

In this class one can find all the methods that a polynomial calculator can perform.
The most important functions are: add, subtract, multiply, divide, derivation and integrate.

Add method

```
public Polynomial add(Polynomial firstPolynomial, Polynomial secondPolynomial) {
    validateInput(firstPolynomial, secondPolynomial);
    if (firstPolynomial.higherDegree(secondPolynomial) == secondPolynomial){
        Polynomial p = firstPolynomial;
        firstPolynomial = secondPolynomial;
        secondPolynomial = p;
    }
    Polynomial sum = new Polynomial(firstPolynomial.getMonomials());
    for (Monomial m : secondPolynomial.monomials){
        boolean found = false;
        for (Monomial m1 : firstPolynomial.monomials){
            if (m.getPower().equals(m1.getPower())){
                found = true;
                sum.addCoefficient(m.getPower(), m.getCoefficient());
            }
        }
        if (!found){
            sum.addMonomial(m);
        }
    }
    return sum;
}
```

This method declares a sum local variable that will be returned at the end with the result of the addition. The function always starts by iterating through both the monomial arrays of the 2 polynomials. It adds to the result every monomial and of the monomial already exists, then simply changes the coefficient. By starting if with the polynomial having the higher degree, the order of the degrees in the result will be the decreasing order.

Subtract method

Similarly to the addition, this method also creates a sum local variable and returns it at the end. To perform the subtraction we will take the higher degree polynomial and subtract from that the one with the lower degree. This method works the same way as the add method, the major differnece is simply that the coefficients that change are multiplied with -1, for the operation to be correct.

Multiply method

```java
public Polynomial multiply(Polynomial firstPolynomial, Polynomial secondPolynomial) {
    validateInput(firstPolynomial, secondPolynomial);
    Polynomial product = new Polynomial();
    ArrayList<Monomial> monomials = new ArrayList<>();
    for (Monomial m: firstPolynomial.monomials){
        for (Monomial m1: secondPolynomial.monomials){
            if (product.powerExists(m.getPower() + m1.getPower())){
                product.addCoefficient( power: m.getPower() + m1.getPower(), coeff: m.getCoefficient() * m1.getCoefficient());
            } else{
                Monomial newMonomial = new Monomial( coefficient: m.getCoefficient() * m1.getCoefficient(),  power: m.getPower() + m1.getPower());
                product.addMonomial(newMonomial);
            }
        }
    }
    return product;
}
```

This method declares a product local variable to store the product of the multiplication. It iterates though the array lists and multiplies each element's coefficient with the other coefficient while adding the powers together. If the power already exists, then the coefficients will simply be changed in the resulting polynomial.

Divide method

```java
public Polynomial divide(Polynomial firstPolynomial, Polynomial secondPolynomial) {
    validateInput(firstPolynomial, secondPolynomial);
    Polynomial quotient = new Polynomial();
    Polynomial remainder;
    if (firstPolynomial.higherDegree(secondPolynomial) != firstPolynomial){
        Polynomial p = firstPolynomial;
        firstPolynomial = secondPolynomial;
        secondPolynomial = p;
    }
    Monomial m = firstPolynomial.monomials.get(0);
    while (m.getPower() >= secondPolynomial.monomials.get(0).getPower()){
        Monomial newMonomial = new Monomial();
        newMonomial.setPower(m.getPower() - secondPolynomial.monomials.get(0).getPower());
        float c = m.getCoefficient() / secondPolynomial.monomials.get(0).getCoefficient();
        newMonomial.setCoefficient(c);
        quotient.addMonomial(newMonomial);
        Polynomial p = new Polynomial();
        p.addMonomial(newMonomial);
        Polynomial p1 = multiply(p, secondPolynomial);
        remainder = subtract(firstPolynomial, p1);
        if(remainder.monomials.size() == 0){
            break;
        } else{
            m = remainder.monomials.get(0);
        }
    }
    return quotient;
}
```

Following the indicated steps of polynomial division, this method calculates the quotient and the remainder of the operation, however, it only returns the quotient to be printed. The division of polynomials has as condition for the first term to be greater or equal in degree with the second term. This inequality is checked after every division step in the condition of the while loop. After that the new coefficient and power of the next monomial of the quotient are calculated. The multiplication and subtraction methods are called to find the remainder after every step of this calculus. Then at the end of the loop, the remainder is checked, and if it is 0, that indicates the end of the operation and we exit the while loop.

Derivation method

```java
public Polynomial derivation(Polynomial poly
    validateInput2(polynomial);
    Polynomial res = new Polynomial();
    ArrayList<Monomial> monomials = new Arra

    for (Monomial m: polynomial.monomials){
        Monomial newMonomial = new Monomial();
        newMonomial.setPower(m.getPower() - 1);
        newMonomial.setCoefficient(m.getCoefficient() * m.getPower());
        monomials.add(newMonomial);
    }
    res.setMonomials(monomials);
    return res;
}
```

To perform derivation on a polynomial, we iterate through the array of monomials and decrease the power by 1 and multiply the coefficient with the initial power. It is simply following the derivation formula for x^n = n * x^(n-1). The result is returned in the res polynomial.

Integrate method

```java
public Polynomial integrate(Polynomial polynomial) {
    validateInput2(polynomial);
    Polynomial res = new Polynomial();
    ArrayList<Monomial> monomials = new ArrayList<>();

    for (Monomial m: polynomial.monomials){
        Monomial newMonomial = new Monomial();
        newMonomial.setPower(m.getPower() + 1);
        newMonomial.setCoefficient(m.getCoefficient() / newMonomial.getPower());
        monomials.add(newMonomial);
    }
    res.setMonomials(monomials);
    return res;
}
```

The integration method also follows the formula x^n => x^(n+1)/(n+1).
Iterating through the monomial array, the power increases by 1 and the coefficient gets divided by the new power. In this step it is quite helpful to have the coefficients declared as floats. It provides a precision in the resulting polynomial's coefficients.

Other methods

The validateInput and validateInput2 methods are called several times, before the execution of any operation. They check if the provided inputs are not null, so the user has provided a polynomial to perform the calculations on. If not, a window will pop up, telling the user that the input is incorrect. Then the user must type another input in the indicated text fields.

The transformStringToPolynomial and transformStringToMonomial methods are called every time when a new input is given. They transform the string into monomials at first, then adding these monomials to the array, the result is the correct polynomial.
The transformPolynomialToString method performs the exact opposite operation for printing the results on the screen. It writes floating point coefficients with 10^-2 precision.

```
public String transformPolynomialToString(Polynomial polynomial){
    String poly = "";
    for(Monomial m : polynomial.monomials){
        if (m.getCoefficient() != 0){
            if(m.getCoefficient() != 1 && m.getCoefficient() != -1){
                if(m.getCoefficient() > 0){
                    poly += " + ";
                }
                poly += String.format("%.2f", m.getCoefficient());
            }
            if(m.getPower() != 0){
                if(m.getCoefficient() == -1){
                    poly += "- x^";
                } else if(m.getCoefficient() == 1){
                    poly += " + x^";
                } else{
                    poly += " x^";
                }
                poly += Integer.toString(m.getPower());
            }
            if(m.getCoefficient() == 1 && m.getPower() == 0){
                poly += " + 1";
            }
            if(m.getCoefficient() == -1 && m.getPower() == 0){
                poly += "- 1";
            }
        }
    }
    //System.out.println("The polynomial is: " + poly.substring(0, poly.length() - 1));
    if (poly.length() != 0 && poly.substring(0, 3).equals(" + "))
    return poly.substring(3);
    else if(poly.length() != 0)
        return poly;
    else return "0";
}
```

In the case of tranforming back to string, the plus and minus signs must be analyzed carefully, and therefore there are 4 main cases: when the monomial has the coefficient 1 and power 0, we only write +1, when the coefficient is -1 and the power 0, we write -1. However, when the coefficient is not 1 we use the toString method to transform the integer into a string.

View class

This class extends JFrame that indicates the JavaSwing graphical user interface. With dividing the contentPane into 3 parts (numbersPane, buttonsPane, resultsPane), and using the GridLayout, we get the interface of the application. The construstor method calls the prepareGui method which initializes all these panes.

The prepareNumbersPane method writes on the screen the 2 labels to indicate the places of the text boxes for the inputs, and also it places the text boxes in the right place on the panel.
The prepareButtonsPane method places the 6 buttons for the different operations that the calculator could do. The actionlistener that is connected with all 6 of these buttons will help distinguish between the operations and indicated to the controller if one of them has been pressed by the user.
The prepareResultsPane simply places the label “Result:” and an empty string which indicates that no operations have been executed yet.
All the different background colors, borders and font features try to make the interface as interactive and as user friendly, as a calculator can be.

```
JButton computeButton1 = new JButton( text: "Add");
computeButton1.setFont(new Font( name: "Serif",  style: Font.ITALIC | Font.BOLD,  size: 13));
computeButton1.setForeground(Color.white);
computeButton1.setActionCommand("Add");
computeButton1.setBackground(Color.MAGENTA);
computeButton1.addActionListener(this.controller);
operationsPanel.add(computeButton1);
```

The getFirstNumberTextField and getSecondNumberTextField methods return the string input while the getResultValueLabel returns the result of the operations.

Controller class

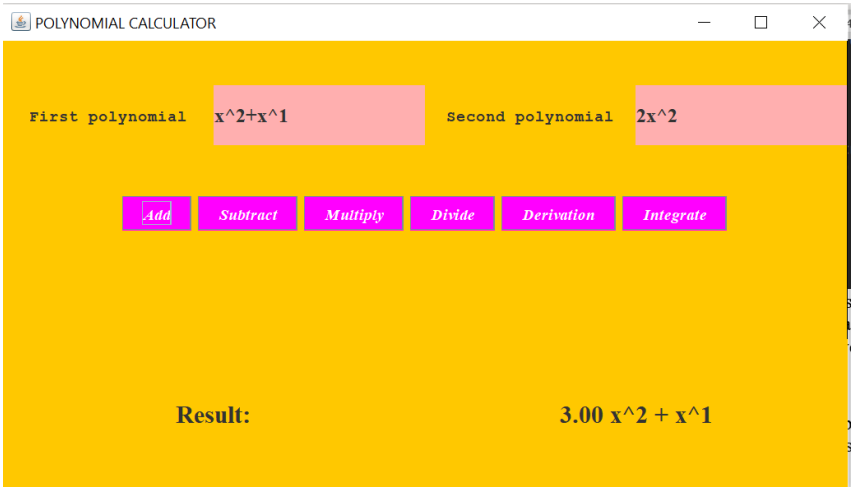
This class mainly focuses on the actionPerformed method. This method gets called every time the action listener detects user interaction with the interface. This method distinguishes between the different buttons’ commands and calls the indicated methods from the Operations class. It also transforms the input string and after the operations it transforms also the polynomial back to string.

```
switch(command){
    case "Add": result = operations.add(firstP, secondP);
        break;
    case "Subtract": result = operations.subtract(firstP, secondP);
        break;
    case "Multiply": result = operations.multiply(firstP, secondP);
        break;
    case "Divide": result = operations.divide(firstP, secondP);
        break;
    case "Derivation": result = operations.derivation(firstP);
        break;
    case "Integrate": result = operations.integrate(firstP);
        break;
}
```

5. Results

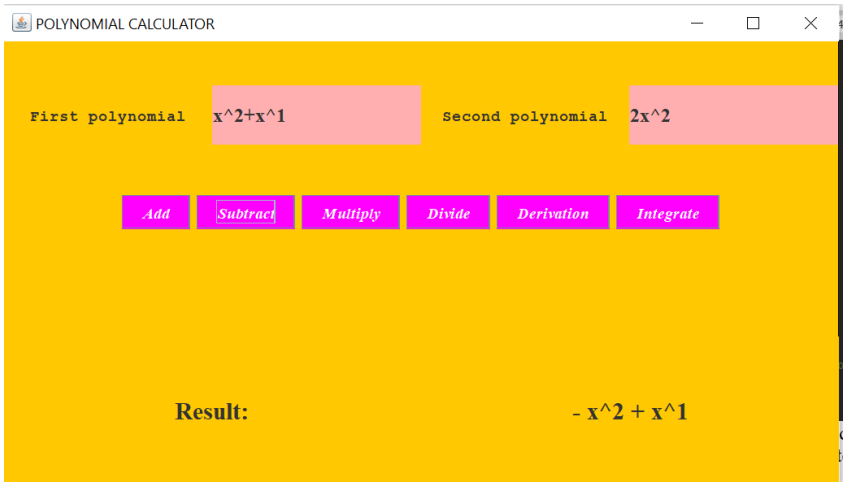
To realize unit testing, I've used Junit. The testing classes are AddOperationsTest, SubtractOperationsTest, MultiplyOperationsTest, DivideOperationsTest, DerivationOperationsTest, IntergationOperationsTest and for the input and output functions as well, there is InputTest and OutputTest.
The way the test classes were designed is using either @Test methods or parametrized testing, where we generate arguments with a method that will be fed to the method doing the assert operation.

For the add operation:



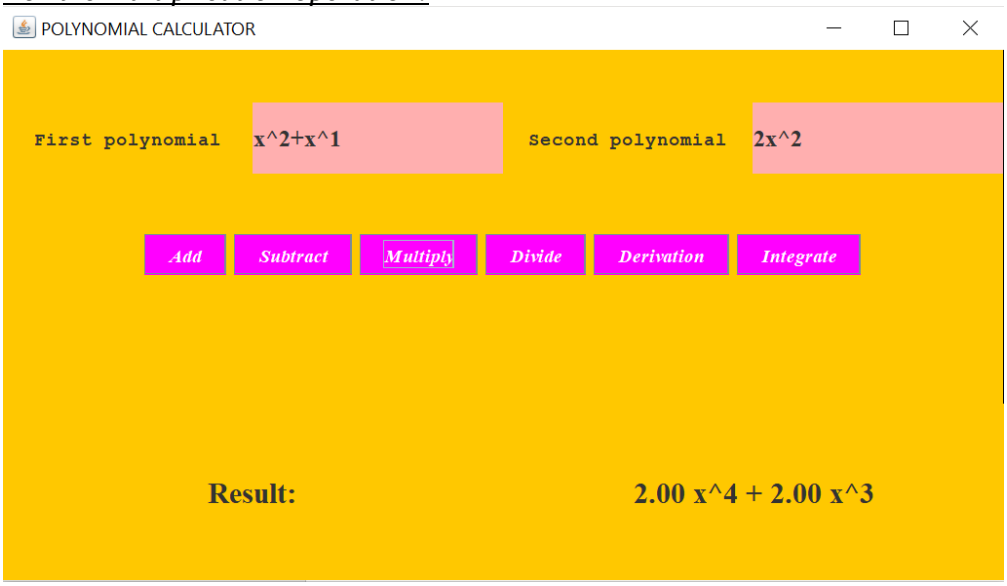
```java
public class AddOperationsTest {
    Operations operations = new Operations();

    @ParameterizedTest
    @MethodSource("provideInput")
    void testAddition(Polynomial p1, Polynomial p2, Polynomial p3){
        assertTrue(operations.add(p1, p2).equals(p3), message: "Correct addition");
    }


    private static ArrayList<Arguments> provideInput(){
        ArrayList<Arguments> argumentList = new ArrayList<>();

        Monomial monomial1 = new Monomial( coefficient: 2, power: 2);
        Monomial monomial2 = new Monomial( coefficient: 3, power: 4);


        ArrayList<Monomial> monomials1 = new ArrayList<>();
        ArrayList<Monomial> monomials2 = new ArrayList<>();
        ArrayList<Monomial> monomials3 = new ArrayList<>();
        monomials1.add(monomial1);
        monomials2.add(monomial2);
        monomials3.add(monomial1);
        monomials3.add(monomial2);
        Polynomial p1 = new Polynomial(monomials1);
        Polynomial p2 = new Polynomial(monomials2);
        Polynomial p3 = new Polynomial(monomials3);

        argumentList.add(Arguments.of(p1, p2, p3));
        return argumentList;
    }
```

For the subtract operation:

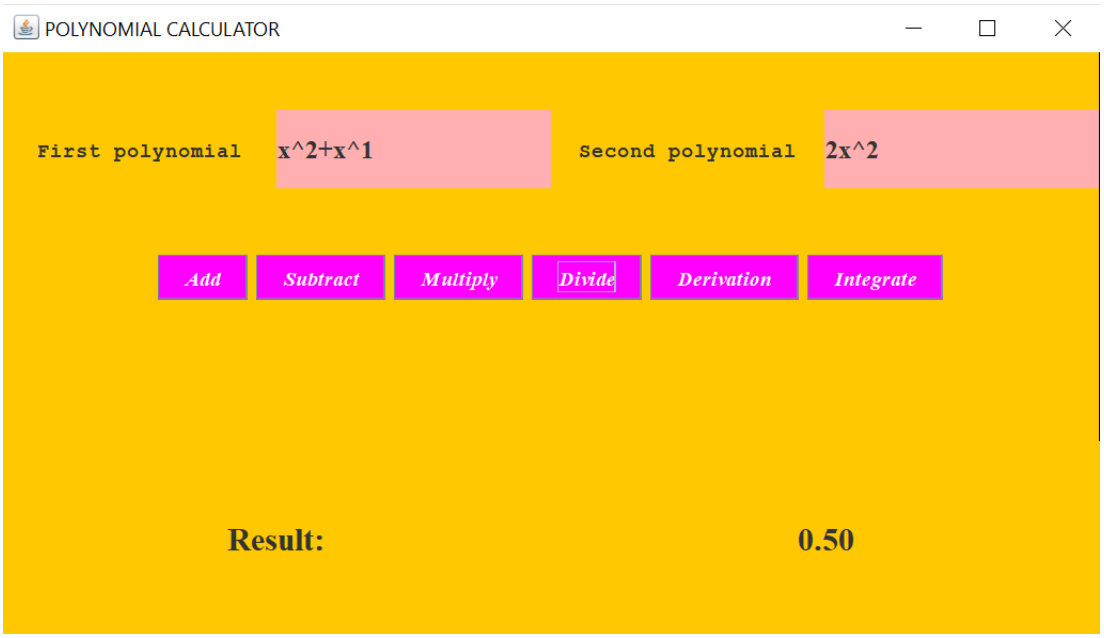


```java
public class SubtractOperationsTest {
    Operations operations = new Operations();

    @Test
    public void subtractTest(){
        Monomial monomial1 = new Monomial( coefficient: 3,  power: 4);
        Monomial monomial2 = new Monomial( coefficient: 2,  power: 4);
        Monomial monomial3 = new Monomial( coefficient: 1,  power: 4);

        ArrayList<Monomial> monomials1 = new ArrayList<>();
        ArrayList<Monomial> monomials2 = new ArrayList<>();
        ArrayList<Monomial> monomials3 = new ArrayList<>();
        monomials1.add(monomial1);
        monomials2.add(monomial2);
        monomials3.add(monomial3);
        Polynomial p1 = new Polynomial(monomials1);
        Polynomial p2 = new Polynomial(monomials2);
        Polynomial p3 = new Polynomial(monomials3);

        assertTrue(operations.subtract(p1, p2).equals(p3),  message: "Correct subtraction");
    }
}
```

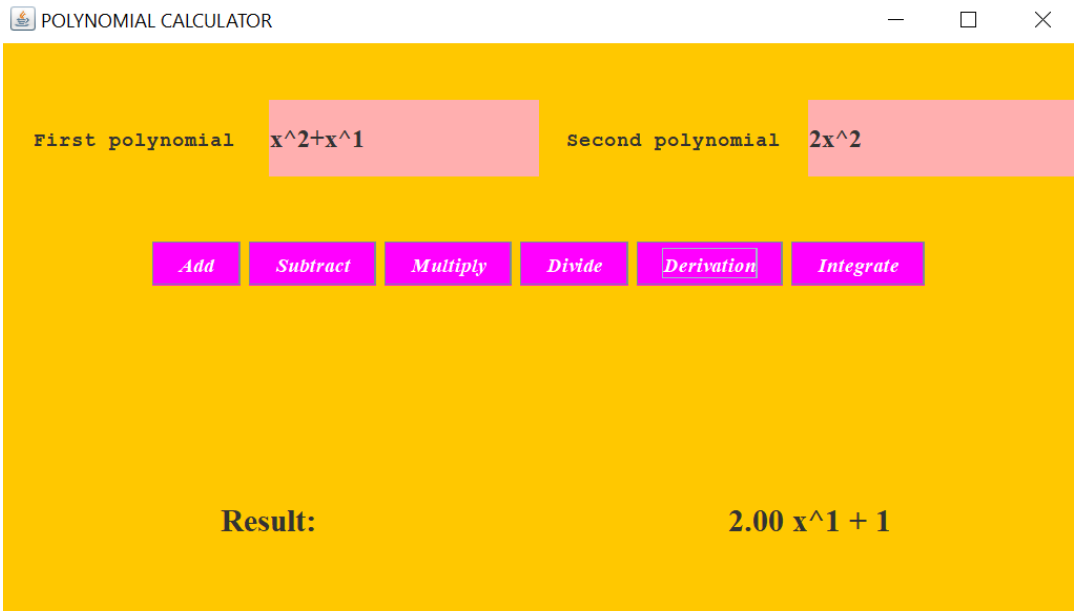
For the multiplication operation:



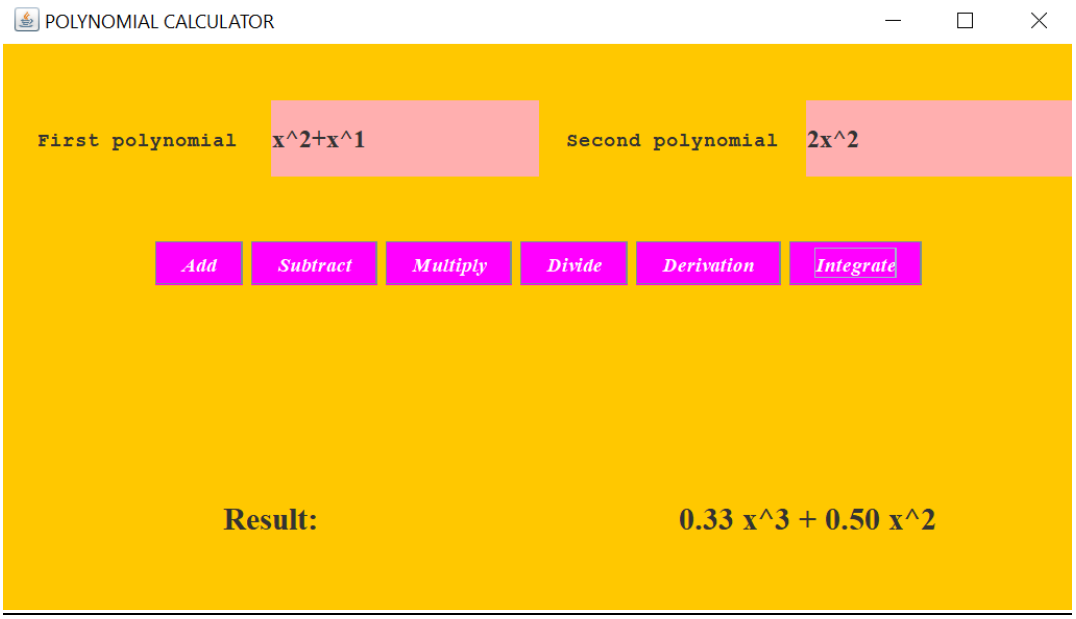
For the division operation:



For the derivation operation:



For the integration operation:



All the test cases show that the results of the operations are correct and the input and output transformations work correctly for different types of parameters.

6. Conclusion

All the code fragments presented above and the unit testing ensure that the project accomplishes the predefined objectives. The polynomial calculator has a complete front end and back end that make it possible to perform the addition, subtraction, multiplication, division, derivation and integration operations correctly.
However, there is plenty of room for improvement. The interface design could be developed further and different operations could be added as well. The power operation or the elementwise multiplication would be a great example for that. The calculator could find the roots of the polynomial and tell the user if they are integer numbers, real numbers or complex numbers. There are plenty of operations a calculator could implement. The ones already implemented are the basis of all the other operations and they would prove useful in case of further improvement.

As far as knowledge gain, I have never done such a complex project in Java, never even used Java Swing or Junit. This project represented a great challenge for me. I was really scared that I will not be able to accomplish the tasks and finish the project in time. However, I have never been more proud of myself, then when I realized that the computations work and the interface is also finished. It was an intense learning process but a rewarding one.

7. Bibliography

The resources for this project were the pdf-s provided by the professor, the presentation of the project and the pdf guide.
I have also tackled several problems with the help of different websites explaining different Java concepts.