Technical University of Cluj-Napoca
Programming Techniques
Assignment 1

Polynomials Calculator

Teacher: Prof. Ioan Salomie
Teacher Assistant: Dr. Viorica Chifu
Student: Domokos Robert - Ștefan
Group: 30421

Content

1. Objective

Design and implement a polynomial calculator with a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation (addition, subtraction, multiplication, division, differentiation, integration) to be performed and view the result. Secondary objectives:

- Analyze the problem and identify requirements
- Design the polynomial calculator
- Implement the polynomial calculator
- Test the polynomial calculator

2. Analysis

The analysis of this assignment consists of the functional requirements, the modeling, the scenarios and the use cases.

- Functional requirements

The functional requirements consists of the correctly implementation of the operations between two polynomials(addition, subtraction, multiplication, division) or one polynomial (differentiation or integration), while also displaying in the Result section the correct answer as a polynomial with integer coefficients or double in case of the division and integration operations.

Addition: a0x^0 + a1x^1 + a2x^2 + … + b0x^0 + b1x^1 + b2x^2 + … = (a0+b0)x^0 + (a1+b1)x^1 + (a2+b2)x^2 +…

Subtraction: a0x^0 + a1x^1 + a2x^2 + … - b0x^0 - b1x^1 - b2x^2 - … = (a0-b0)x^0 + (a1-b1)x^1 + (a2-b2)x^2 +…

Differentiation: d(a0x^0 + a1x^1 + a2x^2 + …) = a1*1 x^0 + a2*2 x^1 + …

Integration: I(a0x^0 + a1x^1 + a2x^2 + …) = a0/1 x^1 + a1/2 x^2 + a2/3 x^3 + …

Multiplication: (a0x^0 + a1x^1 + a2x^2 + …) *(b0x^0 + b1x^1 + b2x^2 - …) = a0b0 * x^0 + (a0b1 + a1b0) *x^1 + (a0b2 + a1b1 + a2b0) * x^2 + …

Division: a0x^0 + a1x^1 + a2x^2 + …) / (b0x^0 +b1x^1 + b2x^2 + …) = …

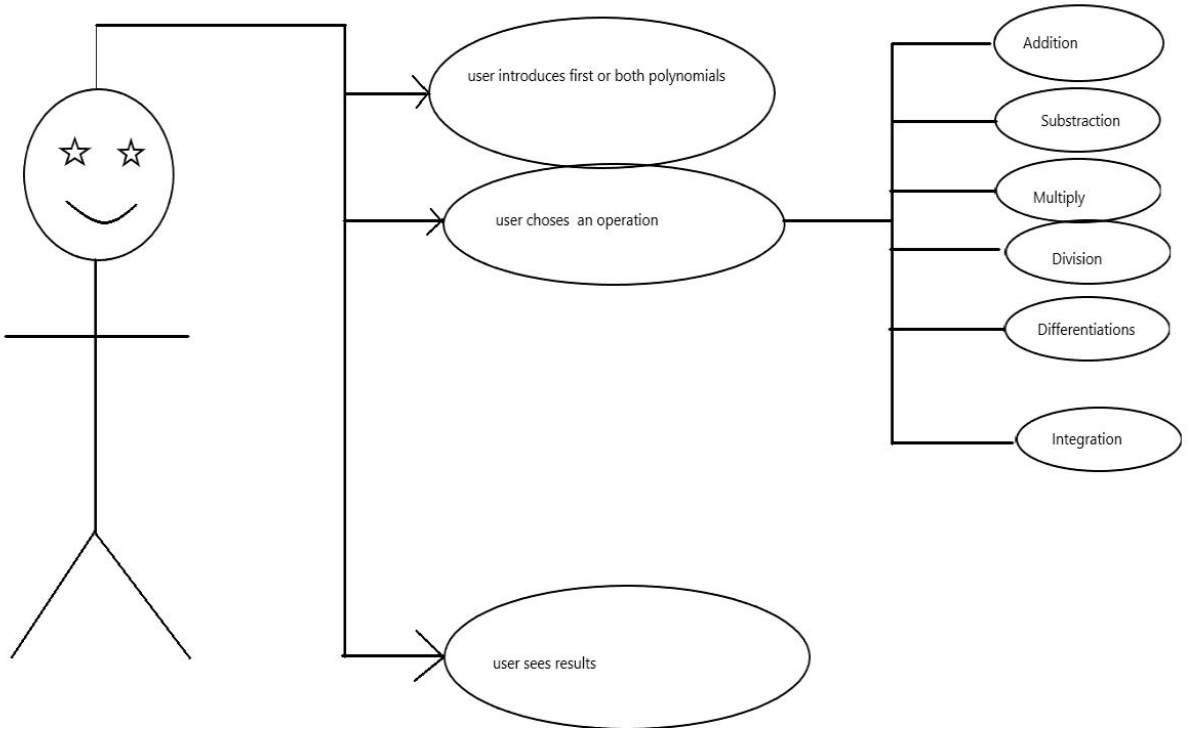
- Modeling

As for modelling the problem the user will be able to use the operations of the calculator by introducing in the interface the wanted polynomials. After the user fetches the wanted polynomials, the application can store them as an ArrayList of monomials for each polynomial, then he can choose a specific operation to perform, such as: Addition of two polynomials, Subtraction of two polynomials, Multiplication of two polynomials, Division of two polynomials, Differentiation of a polynomial and Integration of a polynomial. The result would be displayed on the Result field in the interface.

- Scenarios

As for scenarios, there are several ways for a person to use the application such that he or she can use it as mentioned in the “Use case” part of the documentation or introduce the polynomials in a wrong way. For that I used a JLabel that shows how the monomials should be written such that the application would work properly and satisfy the user.

- Use case



The use case represents the event steps between the actor and the system to achieve the goal of the app. Here the user introduces first the polynomials, he can choose only the first or both, depends on which operations he wants to perform (addition, subtraction, multiply, division, differentiation, integration), and then chooses an operation and presses the button for that operation to see the result.

3. Design

The design part of the project/assignment consists of the design decisions, UML diagrams, relationships, data structures, class design, packages and user interface;

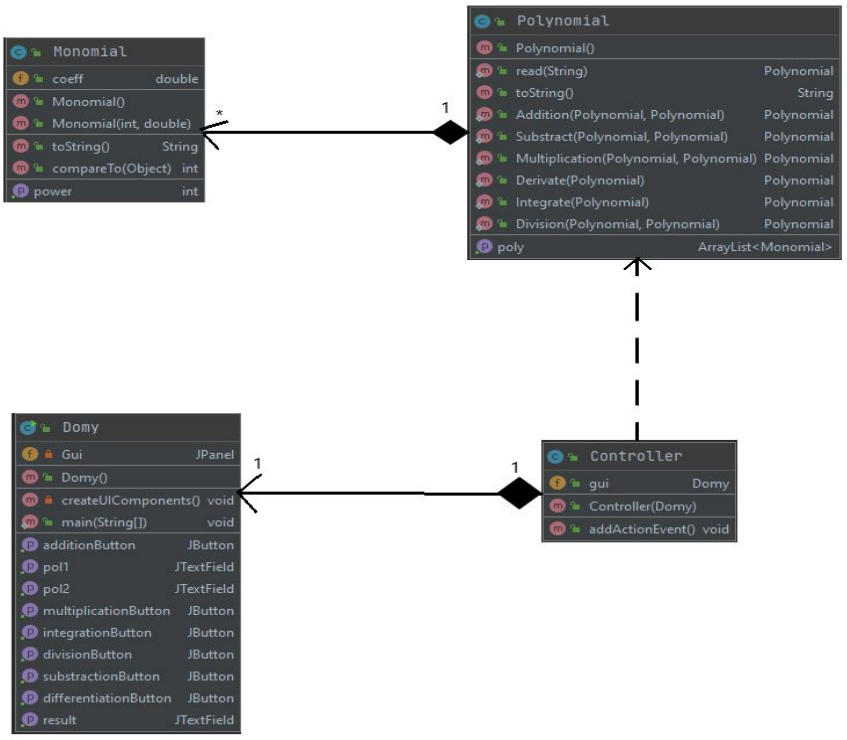
- Design decision

For the design, we have a program represented by a black box that has as inputs two polynomials and an operation and as output one polynomial which represents the result. I made the classes separated according to the Model View Controller paradigm:

- Model representing the classes that model the data used in the program;
- View representing the classes that are used in creating the interfaces or GUIs of the program;
- Controller representing the class or classes that control behaviour of the application;

These packages of classes are represented in the Packages part of the documentation.

- UML Diagram and relationships



The UML diagram shows that the classes Monomial, Polynomial, Domy (GUI) and Controller have different relationships. The relationships represented or shown are:

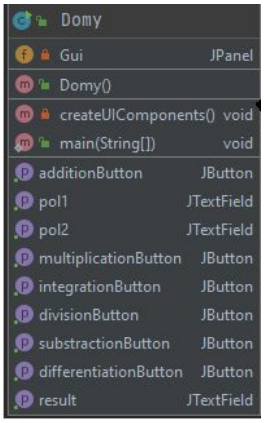
- Polynomial:Monomial is a One to Many relationship and a Dependency because the polynomials contain one or more monomials;
- Domy (GUI): Controller is a One to One relationship and a Dependency;
- Polynomial: Controller has a Dependency relationship;

- Data structures

The data structures used in this design for the polynomials calculator are: int, double, ArrayList, String,

- Packages

As already mentioned in the Design decisions section of the documentation the packages used are:



- **View Package:**

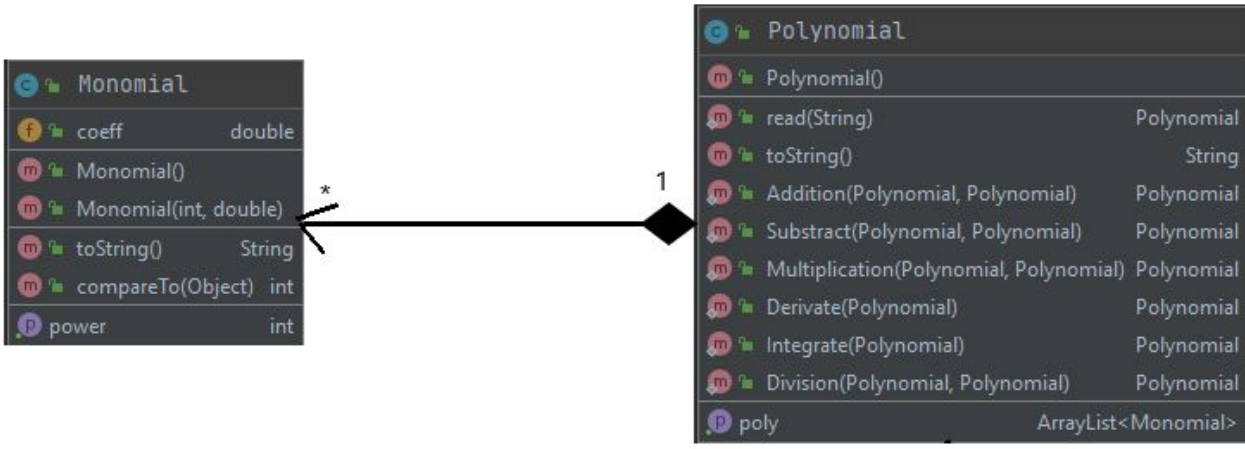
This package is made of everything that is related to the GUI/ interface and also contains a Main class that starts the execution of the application and shows the user interface. In my project the view package is

represented by the Domy.java class (GUI), that has fields for all parts of the GUI, from buttons to text fields and is in relationship with the Controller Package that sets the listener for each button.

This package was created using the GUIForm where I could easily better visualise the graphical user interface which uses the Java Swing.

- **Controller Package:**

This package uses the GUI in order to be able to edit the interface according to what the operation chosen is. The Controller Package has a method that is able to directly give a list of polynomials given in the input for easier use.



- **Model Package**

This package contains 2 classes, Monomial and Polynomial.

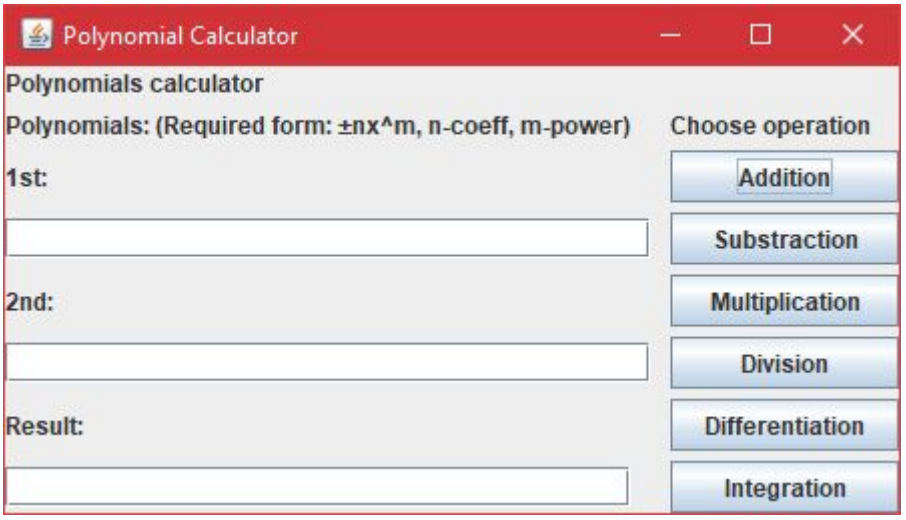
Monomial: This class represents one group or one term from the polynomial. Every group is used to perform all the required operations. This class contains the coeff(double), power(int) variables and the methods Monomial(), Monomial(int, double), toString(), compareTo(Object) that are also represented in the Implementation section of this documentation.

Polynomial: This class represents the list of the monomials that are currently composing the given polynomial. Every polynomial contains an ArrayList of monomials. It also contains the methods Polynomial(), read(String), toString(), and the methods for each operation (Addition, Subtraction, Multiplication, Differentiation, Integration, Division which is not yet implemented).

- User Interface

I made the user interface as friendly as I could for this assignment with respect to the requirements, but it can always improve. The user interface consists of the three text fields, each representing the first polynomial, second polynomial and the resulting polynomial. Beside the close button, the application has a button for every operation (Addition, Subtraction, Multiplication, Division, Differentiation, Integration) that he wants to choose.

After introducing the polynomials and pressing one of the operations chosen by the user the result would appear in the Result text field.



4. Implementation

- Monomial.java

This is the class representing the monomials having a coefficient(double coeff) and a power(int power).

```
package com.company;

public class Monomial implements Comparable {
    public int power;
    public double coeff;

    //...
    public Monomial(){...}

    //...
    public Monomial(int power, double coeff) {...}

    //...
    public int getPower() { return power; }

    //...
    @Override
    public String toString() {...}

    //...
    @Override
    public int compareTo(Object o) { return ((Monomial)o).getPower()-getPower(); }
}
```

Monomial(): This constructor is used for initialization.

```
public Monomial(){
    this.power = 0;
    this.coeff = 0;
}
```

Monomial(int power, double coeff): This constructor is used for setting the power and coeff.

```
public Monomial(int power, double coeff) {
    this.power = power;
    this.coeff = coeff;
}
```

getPower(): This method gives me the power of the monomial.

```
public int getPower() {
    return power;
}
```

toString(): The method toString() converts the monom in a string, for the integration I consider it if the coeff is a rational number to be represented as double if not as an integer. It also puts a sign before every monomial even if it is the first one and it has a positive coefficient.

```
@Override
public String toString() {
    if((int)coeff == coeff)
    {
        if (coeff < 0)
            return (int) coeff + "x^" + power;
        else if (coeff > 0)
            return "+" + (int) (coeff) + "x^" + power;
        else
            return "";
    }
    else {
        if (coeff < 0)
            return coeff + "x^" + power;
        else if (coeff > 0)
            return "+" + coeff + "x^" + power;
        else
            return "";
    }
}
```

compareTo(Object o): The method compareTo is used for sorting in polynomial the monomials so that it gives the polynomial sorted

```
@Override
public int compareTo(Object o) {
    return ((Monomial)o).getPower()-getPower();
}
```

- Polynomial.java

This is the class representing the polynomials by an ArrayList of monomials.

```
package com.company;

import java.util.ArrayList;
import java.util.Collections;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Polynomial {

    ArrayList<Monomial> poly = new ArrayList<>();

    public Polynomial() {}

    public ArrayList<Monomial> getPoly() { return poly; }

    public static Polynomial read(String expr) {...}

    @Override
    public String toString() {...}

    public static Polynomial Addition(Polynomial A, Polynomial B) {...}

    public static Polynomial Substract(Polynomial A, Polynomial B) {...}

    public static Polynomial Multiplication(Polynomial A, Polynomial B) {...}

    public static Polynomial Derivate(Polynomial A) {...}

    public static Polynomial Integrate(Polynomial A) {...}
    public static Polynomial Division(Polynomial A,Polynomial B){...}

}
```

Polynomial(): This is the constructor of this class.

```
public Polynomial() {}
```

getPoly(): This method is the getter for the polynomials.

```
public ArrayList<Monomial> getPoly() {
    return poly;
}
```

read(String expr): This method returns the polynomial written by the user in the required form which is recognized with the help of the Regular Expression (which recognize the groups that contain +/-nx^m where n is the coeff and m the power).
The Regex pattern "`([-+]?\\d+)x\\^(-?\\d+)`" is composed of:

- "d+" - recognises the integers of one or more decimals;
- "x\\^" - recognise the string "x^" from the monomial;
- "[-+]? or -?" - recognise the sign '-' or '+' if there is present in the monomial;

```
public static Polynomial read(String expr) {
    Polynomial pol = new Polynomial();
    Pattern p = Pattern.compile("([-+]?\\d+)x\\^(-?\\d+)");
    Matcher m = p.matcher(expr);
    while (m.find()) {
        Monomial mol = new Monomial();
        mol.coeff = Double.parseDouble(m.group(1));
        mol.power = Integer.parseInt(m.group(2));
        pol.getPoly().add(mol);
    }
    return pol;
}
```

toString(): This method returns a string of monomials from the polynomial ArrayList.

```
@Override
public String toString() {
    if (poly.isEmpty())
        return "";

    String str = "";
    for (Monomial mono : poly) {
        str = str + mono;
    }
    return str;
}
```

Addition(A,B): This method adds the two polynomials by adding together the coefficients of the monomials with the same power and adding the remaining monomials.

```
public static Polynomial Addition(Polynomial A, Polynomial B) {
    Polynomial S = new Polynomial();
    for (Monomial mono1 : A.getPoly())
        for (Monomial mono2 : B.getPoly())
            if (mono1.power == mono2.power) {
                S.getPoly().add(new Monomial(mono1.power, coeff: mono1.coeff + mono2.coeff));
            }
    for (Monomial mono1 : B.getPoly()) {
        int ok = 0;
        for (Monomial mono2 : A.getPoly())
            if (mono1.power == mono2.power) {
                ok = 1;
                break;
            }
        if (ok == 0)
            S.getPoly().add(new Monomial(mono1.power, mono1.coeff));
    }
    for (Monomial mono1 : A.getPoly()) {
        int ok = 0;
        for (Monomial mono2 : B.getPoly())
            if (mono1.power == mono2.power) {
                ok = 1;
                break;
            }
        if (ok == 0)
            S.getPoly().add(new Monomial(mono1.power, mono1.coeff));
    }
    Collections.sort(S.getPoly());
    return S;
}
```

Substract(A,B): This method substracts the B polynomial from the A polynomial by reversing the sign for each monomial from B and then perform the addition of these two.

```
public static Polynomial Substract(Polynomial A, Polynomial B) {
    Polynomial S = new Polynomial(),B1=B;
    for (Monomial mono1 : B1.getPoly()) {
        mono1.coeff = -mono1.coeff;
    }
    for (Monomial mono1 : A.getPoly())
        for (Monomial mono2 : B1.getPoly())
            if (mono1.power == mono2.power) {
                S.getPoly().add(new Monomial(mono1.power, coeff: mono1.coeff + mono2.coeff));
            }
    for (Monomial mono1 : A.getPoly()) {
        int ok = 0;
        for (Monomial mono2 : B1.getPoly())
            if (mono1.power == mono2.power)
                ok = 1;
        if (ok == 0)
            S.getPoly().add(new Monomial(mono1.power,mono1.coeff));
    }
    for (Monomial mono1 : B1.getPoly()) {
        int ok = 0;
        for (Monomial mono2 : A.getPoly())
            if (mono1.power == mono2.power)
                ok = 1;
        if (ok == 0)
            S.getPoly().add(new Monomial(mono1.power, mono1.coeff));
    }
    Collections.sort(S.getPoly());
    return S;
}
```

Multiplication(A,B): This method multiplies A and B polynomials by multiplying every monomial from A with every monomial from B.

```
public static Polynomial Multiplication(Polynomial A, Polynomial B) {
    Polynomial P = new Polynomial();
    for (Monomial mono1 : A.getPoly())
        for (Monomial mono2 : B.getPoly()) {
            Monomial prod = new Monomial();
            prod.coeff = mono1.coeff * mono2.coeff;
            prod.power = mono1.power + mono2.power;
            P.getPoly().add(prod);
        }
    Collections.sort(P.getPoly());
    Polynomial P1 = new Polynomial();
    Monomial mono = new Monomial( power: -1,  coeff: -1);
    for (Monomial mono1 : P.getPoly()) {
        if (mono.power == mono1.power) {

            mono.coeff = mono.coeff + mono1.coeff;
        } else {
            mono = mono1;
            P1.getPoly().add(mono);
        }
    }

    return P1;
}
```

Derivate(A): This method derivates the polynomial A.

```
public static Polynomial Derivate(Polynomial A) {
    Polynomial A1 = A;
    for(Monomial mono1: A1.getPoly())
    {
        mono1.coeff = mono1.power*mono1.coeff;
        mono1.power = mono1.power - 1;
    }
    return A1;
}
```

Integrate(A): This method integrates the polynomial A.

```
public static Polynomial Integrate(Polynomial A) {
    Polynomial A1 = A;
    for(Monomial mono1: A1.getPoly()){
        mono1.power = mono1.power + 1;
        mono1.coeff = mono1.coeff/mono1.power;

    }
    return A1;
}
```

Division(A,B): This method is not yet implemented.

```
public static Polynomial Division(Polynomial A,Polynomial B){
    Polynomial S=new Polynomial();
    S.getPoly().add(new Monomial( power: 0, coeff: 0));
    return S;
}
```

- Domy.java

This class represents the GUI interface of the application. I used JSwing to implement this part. Every JButton, JTextField and JPanel has a getter.

```
package com.company;

import javax.swing.*;

public class Domy {
    private JButton additionButton;
    private JButton divisionButton;
    private JButton substractionButton;
    private JButton multiplicationButton;
    private JButton differentiationButton;
    private JButton integrationButton;
    private JTextField pol1;
    private JTextField pol2;
    private JPanel Gui;
    private JTextField result;

    public JButton getAdditionButton() { return additionButton; }
    public JButton getDivisionButton() { return divisionButton; }
    public JButton getSubstractionButton() { return substractionButton; }
    public JButton getMultiplicationButton() { return multiplicationButton; }
    public JButton getDifferentiationButton() { return differentiationButton; }
    public JButton getIntegrationButton() { return integrationButton; }
    public JTextField getPol1() { return pol1; }
    public JTextField getPol2() { return pol2; }
    public JTextField getResult() { return result; }
    private void createUIComponents() { }

    public Domy(){
        Controller cont= new Controller( gui: this);
        cont.addActionEvent();
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame( title: "Polynomial Calculator");
        frame.setContentPane(new Domy().Gui);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```

The main method here forms the JFrame with all the buttons and labels mentioned above.
Here the constructor **Domy()** makes a new Controller for this GUI by using the Controller.java class and adding the ActionEvents for each button to the GUI.

- Controller.java

In this class I used the method **addActionEvent()** to add the ActionListener to each button so that it can display in the Result field from the interface the result for each operation.

```java
package com.company;


public class Controller {
    public Domy gui ;

    public Controller(Domy gui) {
        this.gui = gui;
    }

    public void addActionEvent() {

        gui.getAdditionButton().addActionListener(e -> {

            Polynomial A = Polynomial.read(gui.getPol1().getText());
            Polynomial B = Polynomial.read(gui.getPol2().getText());
            gui.getResult().setText(Polynomial.Addition(A,B).toString());
        });
        gui.getSubstractionButton().addActionListener(e -> {
            Polynomial A = Polynomial.read(gui.getPol1().getText());
            Polynomial B = Polynomial.read(gui.getPol2().getText());
            gui.getResult().setText(Polynomial.Substract(A,B).toString());
        });
        gui.getMultiplicationButton().addActionListener(e -> {
            Polynomial A = Polynomial.read(gui.getPol1().getText());
            Polynomial B = Polynomial.read(gui.getPol2().getText());
            gui.getResult().setText(Polynomial.Multiplication(A,B).toString());
        });
        gui.getDivisionButton().addActionListener(e -> {
            gui.getResult().setText("This feature is not available!");
        });
        gui.getIntegrationButton().addActionListener(e -> {
            Polynomial A = Polynomial.read(gui.getPol1().getText());;
            gui.getResult().setText(Polynomial.Integrate(A).toString());
        });
        gui.getDifferentiationButton().addActionListener(e -> {
            Polynomial A = Polynomial.read(gui.getPol1().getText());
            gui.getResult().setText(Polynomial.Derivate(A).toString());
        });
    }
}
```

- Tests.java

This class is represented in the 5. Results section of the documentation.

5. Results

In order to verify the correctness of the operation functions I used the JUnit platform by making a test function for each function that represents the operations with polynomials in the Polynomial.java.

Tests.java

```java
import com.company.Polynomial;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class Tests {

    @Test
    void testAddition() {...}
    @Test
    void testSubstraction() {...}
    @Test
    void testMultiplication() {...}
    @Test
    void testDerivate() {...}
    @Test
    void testIntegration() {...}
    @Test
    void testDivision() {...}

}
```

testAddition()

Adds the two given polynomials ("4x^2+3x^1+3x^0" and "3x^2+2x^1") and checks if the resulting polynomial is the correct one("+7x^2+5x^1+3x^0").

```java
@Test
void testAddition() {
    Polynomial A,B;
    A=Polynomial.read("4x^2+3x^1+3x^0");
    B=Polynomial.read("3x^2+2x^1");
    assertEquals( expected: "+7x^2+5x^1+3x^0",Polynomial.Addition(A,B).toString());
}
```

testSubstraction()

Subtracts the second polynomial ("3x^2+2x^1") from the first polynomial("4x^2+3x^1") and checks if the resulting polynomial is the correct one("+1x^2+1x^1").

```java
@Test
void testSubstraction() {
    Polynomial A,B;
    A=Polynomial.read("4x^2+3x^1");
    B=Polynomial.read("3x^2+2x^1");
    assertEquals( expected: "+1x^2+1x^1",Polynomial.Substract(A,B).toString());
}
```

testMultiplication()

Multiplies the two given polynomials (“4x^2+3x^1” and “3x^2+2x^1”) and checks if the resulting polynomial is the correct one (“+12x^4+17x^3+6x^2”).

```
@Test
void testMultiplication() {
    Polynomial A,B;
    A=Polynomial.read("4x^2+3x^1");
    B=Polynomial.read("3x^2+2x^1");
    assertEquals( expected: "+12x^4+17x^3+6x^2",Polynomial.Multiplication(A,B).toString());
}
```

testDerivate()

Derivates the given polynomial (“4x^2+3x^1”) and checks if the resulting polynomial is the correct one (“+8x^1+3x^0”).

```
@Test
void testDerivate() {
    Polynomial A;
    A=Polynomial.read("4x^2+3x^1");
    assertEquals( expected: "+8x^1+3x^0",Polynomial.Derivate(A).toString());
}
```

testIntegration()

Integrates the given polynomial (“3x^2+4x^1”) and checks if the resulting polynomial is the correct one (“+1x^3+2x^2”).

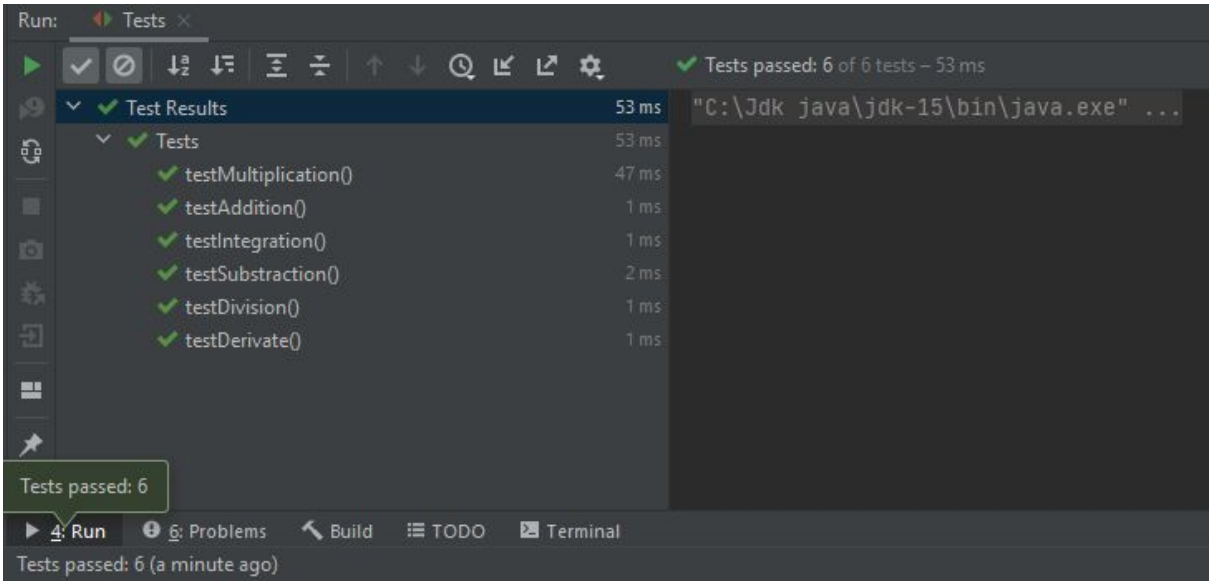
```
@Test
void testIntegration() {
    Polynomial A;
    A=Polynomial.read("3x^2+4x^1");
    assertEquals( expected: "+1x^3+2x^2",Polynomial.Integrate(A).toString());
}
```

testDivision()

As mentioned before the division function doesn’t work yet so the test just checks if the resulting polynomial is empty.

```
@Test
void testDivision() {
    Polynomial A,B;
    A=Polynomial.read("4x^2+3x^1");
    B=Polynomial.read("4x^2+3x^1");
    assertEquals( expected: "",Polynomial.Division(A,B).toString());
}
```

After running the class Tests.java we can see that all the operations are giving the results that we want.



6. Conclusions

In conclusion, the problems and the requirements that form this assignment challenged and improved my Java and Object Oriented Programming skills that could end up being useful in the future internships that I intend to apply for.

At the same time, it was my first chance to use the JUnit testing platform, which surprised me how much easier was to work and modify my code. Another java feature that I didn't use before was the Java Regular Expressions, which helped me read the polynomials in a much simpler and easier way that I was accustomed to. The GUIFrom had a great impact in the time allotted to this project because if I had to do it by writing it could take longer than expected, also another great thing is that it uses the Java Swing, that makes it portable between operating systems.

Some of the improvements to that application I made could be:

- Finishing the division operation so that could work;
- Improving the already existing operations so that they could run quicker;
- Making a more user-friendly interface that could be used even in the future.

7. Bibliography

https://en.wikipedia.org/wiki/Use_case/
https://regex101.com/
https://en.wikipedia.org/wiki/Regular_expression/
https://en.wikipedia.org/wiki/Object-oriented_programming/
https://www.smartdraw.com/uml-diagram/
https://stackoverflow.com/questions/214741/what-is-a-stackoverflowerror/
https://stackoverflow.com/questions/15363706/how-to-program-this-gui-in-java-swing/
https://www.uml-diagrams.org/dependency.html/