DOCUMENTATION

Assignment Nr. 1

Polynomial Calculator

NUME STUDENT: Bar Luca-Narcis
GRUPA:30421

CUPRINS

1. Assignment Objectives

Main Objective: 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.
Secondary Objectives:
-Organisation through classes
-Designing the algorithm(solution) using OOP Principles
-Formulating use-case scenarios
-Implementing and testing the solutions

2. Problem analysis, modelling, scenarios, use-cases

The problem requires both mathematical knowledge(in order to implement the necessary operations) and object-oriented programming principles, in order to create and manipulate objects that are used in our mathematical computations.

From analyzing the problem, we establish that we need a set of input data and a set of output data. The input is represented by two polynomials in all of the cases, except for differentiation and integration, where we only take one of the(specifically, the first one). The desired operation is also part of the input, as it is up to the user to select what they want to be done with the polynomials. The input polynomials and the operation determine an output, which is displayed in the form of a polynomial(and in the case of division, we display it as Result and Remainder). The polynomials must contain the coefficient and the degree for each monomial it has(e.g 5x^4 +3x^1 +2x^0). I have chosen this method of input, because it offers the user a more detailed look on how exactly the operations are performed. Also, the monomials must be in order of their degree. And in case the degree of the first polynomial is smaller than the one of the second one, an error is displayed(errors are also displayed in the cases of incorrect input formats/values).

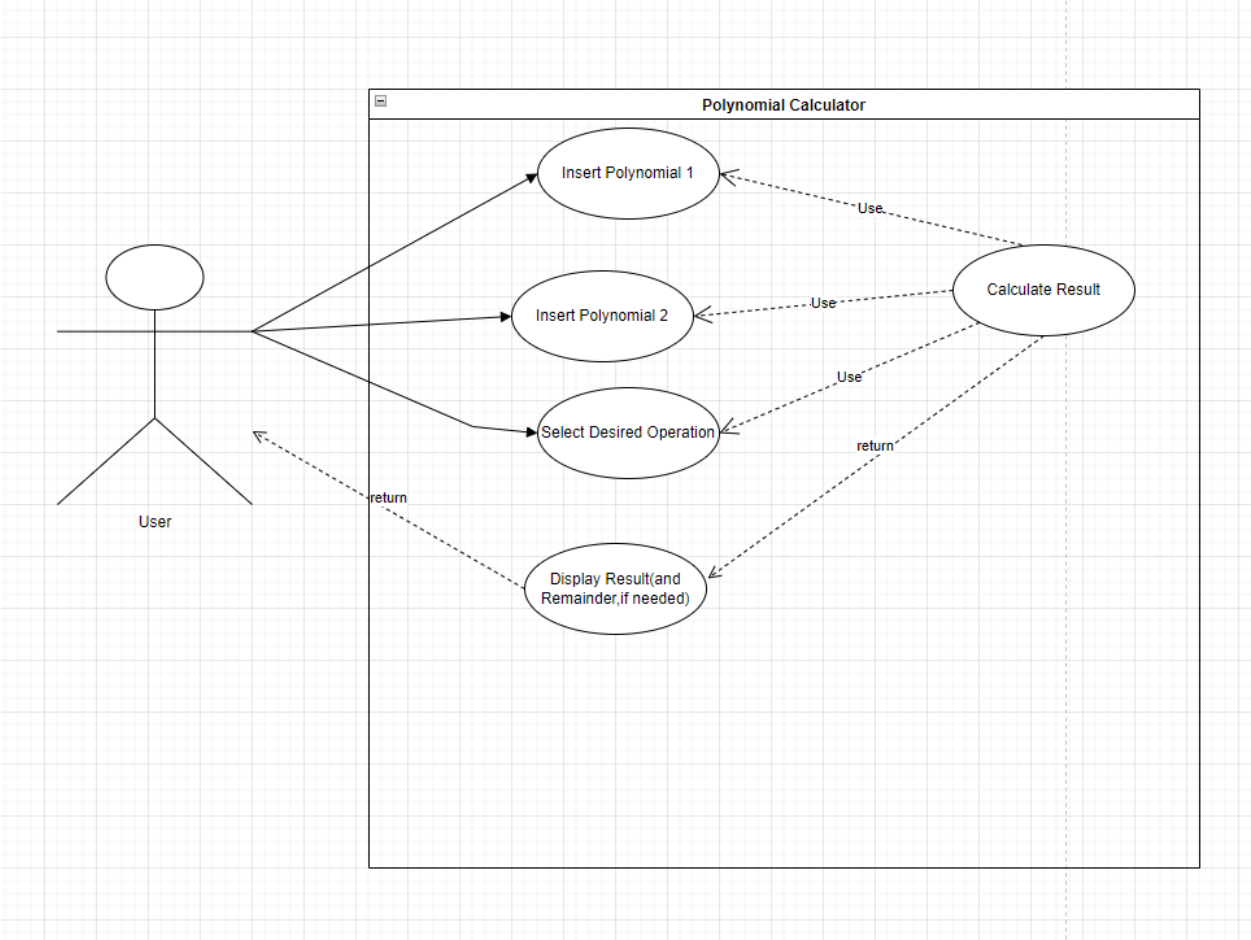
After the data is introduced, it is transformed and stored in the specific structures in the program. Then, operations are performed on said structures and the result is sent to the corresponding structure(or structures in case of division).

Real coefficients will be used in the special cases of division and integration, because they are much easier to read than fractions, as the objective of the program is for it to be as practical and easy to use as possible.

Down here , we can observe the use-case diagram of the project:

It is easily seen that ,at first,the user inserts the polynomials and selects the desired operation

Then, the calculator uses the inputs to calculate the result. The result is the sent to the display, which returns the information to the user, in a visual manner.



3. Design

For the design of the calculator , I have created 13 classes: *Monom, Polinom, Bipol, Addition, Sub,*

Div, Multiply, Result, PolCalc, Deriv, Integrate, PolView and PolController ; and also an interface called *IOp*, implemented by all of the classes that represent an operation. The graphical interface is designed using an MVC architectural model. The Model is represented by the *Polinom, Monom , Bipol* and the operation classes, the View is represented by *PolView* and the Controller is represented by *PolController*.

Monom -> has 2 instance variables: coeff(float) and degree(int) . The coefficient is a float in order to display the correct result in the cases of division and integration. It also contains the getters and setters for its variables and implements 2 methods: toString() and compareTo(Monom m) (used to check if degree of first polynomial is greater than the second).

Polinom -> contains a single instance variable – *elem*, of *type List<Monom>* because we need to store a variable number of monomials inside our polynomial. It has 2 constructors and implements the getter and setter for *elem* , and impletents *addMonom(Monom m) and toString();*

IOp-> is an interface which contains the method compute(Polinom p1, Polinom p2);

Addition, Sub, Deriv, Integrate, Div, Multiply -> classes that implement IOp and implement the compute(Polinom p1, Polinom p2) method in their own way, for the corresponding operation.

Result-> has one variable result of type *Polinom,* used when working with the values obtained from the compute(…) method. It has 2 constructors and a getter and setter

Bipolinom-> extends *Result()* and has a variable *rest* of type *Polinom*, used for operations where we need both the result and the remainder. It also has a constructor and a getter and setter.

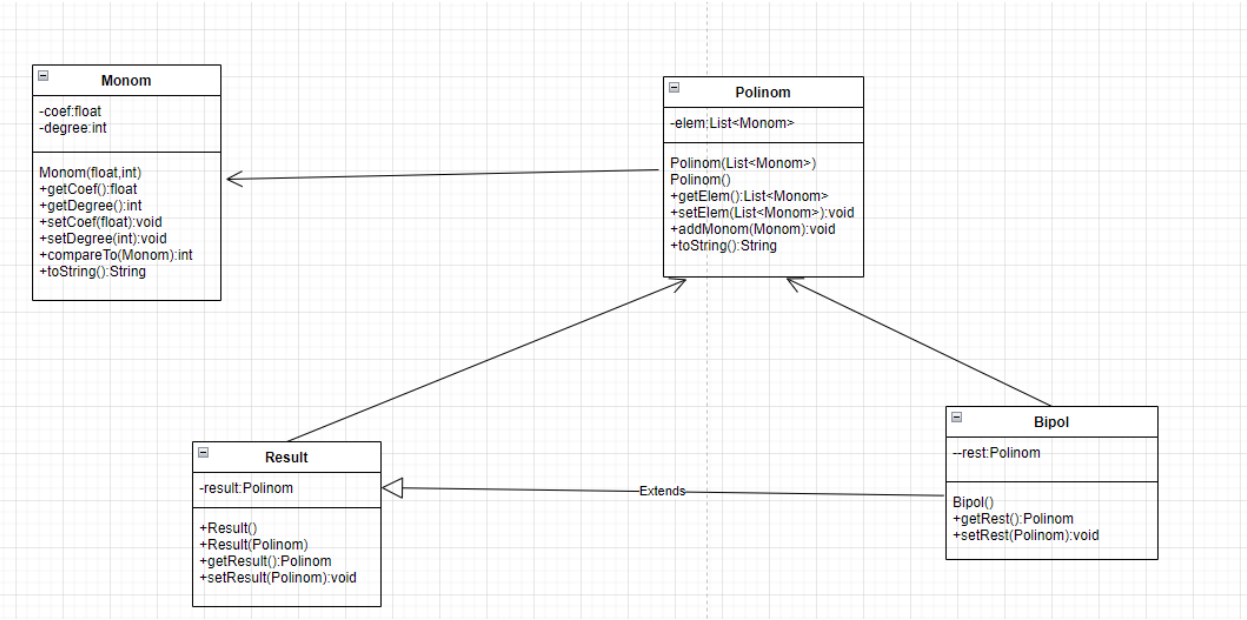
All of the above represent the Model of the interface.

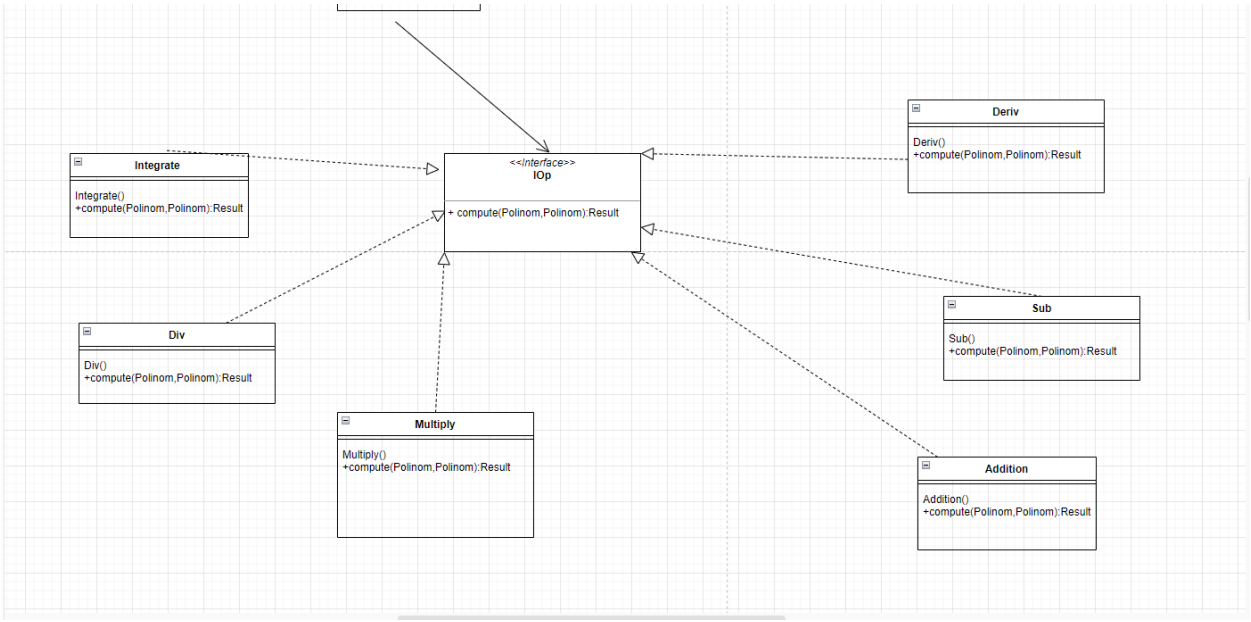
PolView->represents the View of the interface,and contains all of the Swing elements. It has 4 JTextFields(2 editable for introducing Polynomials and 2 uneditable for Result and Remainder) , 5JLabels (for Polynomial 1,Polynomial 2, Result,Rest and Operations) and 5 Jpanels to keep a good alignment of the interface components. It also contains getters and setters for the two Polynomials, in order to retrieve the user's input and then return it as output. There are also 6 JButtons,each with their own Listener method, in order to select the desired operation to be performed.

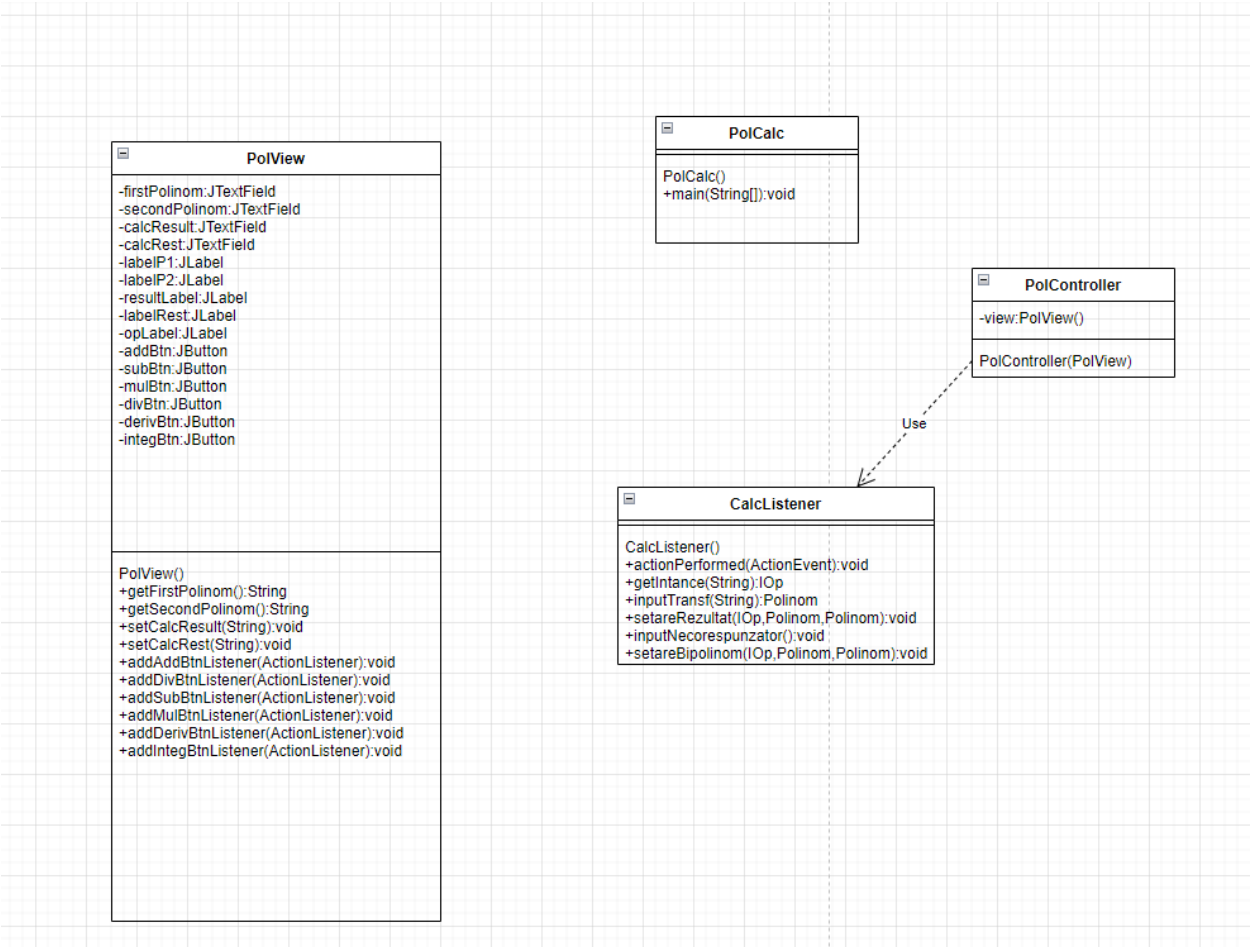
PolController-> represents the Cotroller which decides which steps are followed by the Model. It contains an instance variable *view*, which establish the connection between this class and *PolView*. The constructor contains all of the listeners for the buttons,which are then added to the view. It also contains an internal class called CalcListener which implements actionLister and contains the methods: *actionPerformed(ActionEvent e), getInstance(String s),inputTransf(String s), setareREzultat(IOp op, Polinom p1, Polinom p2), setareBipolinom(IOp op, Polinom p1, Polinom p2),*

inputNecorespunzator().

And down here, we can observe the UML class diagram of the project:




Integrate
Integrate()
+compute(Polinom,Polinom):Result
<<Interface>>
IOp
+ compute(Polinom,Polinom):Result
Deriv
Deriv()
+compute(Polinom,Polinom):Result
Div
Div()
+compute(Polinom,Polinom):Result
Multiply
Multiply()
+compute(Polinom,Polinom):Result
Sub
Sub()
+compute(Polinom,Polinom):Result
Addition
Addition()
+compute(Polinom,Polinom):Result


PolView
-firstPolinom:JTextField
-secondPolinom:JTextField
-calcResult:JTextField
-calcRest:JTextField
-labelP1:JLabel
-labelP2:JLabel
-resultLabel:JLabel
-labelRest:JLabel
-opLabel:JLabel
-addBtn:JButton
-subBtn:JButton
-mulBtn:JButton
-divBtn:JButton
-derivBtn:JButton
-integBtn:JButton
PolView()
+getFirstPolinom():String
+getSecondPolinom():String
+setCalcResult(String):void
+setCalcRest(String):void
+addAddBtnListener(ActionListener):void
+addDivBtnListener(ActionListener):void
+addSubBtnListener(ActionListener):void
+addMulBtnListener(ActionListener):void
+addDerivBtnListener(ActionListener):void
+addIntegBtnListener(ActionListener):void
PolCalc
PolCalc()
+main(String[]):void
PolController
-view:PolView()
PolController(PolView)
Use
CalcListener
CalcListener()
+actionPerformed(ActionEvent):void
+getIntance(String):IOp
+inputTransf(String):Polinom
+setareRezultat(IOp,Polinom,Polinom):void
+inputNecorespunzator():void
+setareBipolinom(IOp,Polinom,Polinom):void

4. Implementation

Here I will describe the implementation decisions for the methods of the classes, alongside the algorithms used. I have also provided some code snippets(from classes that weren't too lengthy)

Monom:

compareTo()-> implemented to sort monomials in the order of their grade

```
public int compareTo(Monom m) {
    if (this.getDegree() > m.getDegree())
        return -1;
    return 1;

}
```

toString()-> used for the output, since it converts the monomial to a String to be displayed

```
public String toString() {
    String s ="";
    String coefS = Float.toString(getCoef());
    int coefI = 0;
    coefI = (int) getCoef();
    if (getCoef()-coefI==0.0f) {

        if(coefI==0) s = " ";
        else if(coefI>0) s="+"+coefI+"x^"+getDegree();
        else s=coefI+"x^"+getDegree();}
    else {
        if(getCoef()==0) s = " ";
        else if(getCoef()>0) s="+"+getCoef()+"x^"+getDegree();
        else s=getCoef()+"x^"+getDegree();
    }

    return s;
}
```

Polinom:

addMonom(Monom m) -> adds a new monomial to the ones that are already forming the polynomial

toString()-> for output. Inside this,the toString() from Monom is called

```
public String toString() {

    String s="";
    if(elem.isEmpty() )  s="0";
    else {
        for(Monom i:elem) {

            s=s+i.toString();
        }
        if(s.charAt(0)=='+')
            s=s.substring(1,s.length());
    }
    if(s.charAt(0)==' ') s="0";
    return s;

}
```

Addition:

compute(Polinom p1,Polinom p2) ->returns a Result type object. It adds p1 and p2. The degree of every monomial in p1 is checked side by side with the ones in p2. If equal,the sum of their coefficients becomes the coefficient of the new monomial that is added to the result(if the sum >0). A second run-through is needed, to avoid losing the monomials from p2 that have a different degree than the ones in p1. In this second run-through, in case of equality, there will be no new monomial, since duplicates can't be allowed.

Sub:

compute(Polinom p1,Polinom p2)->returns a Result type object. Every monomial of p2 is multiplied by -1,then the sum between the new p2 and p1 is computed.

```
public class Sub implements IOp {

    public Result compute(Polinom p1, Polinom p2) {
        Result r=new Result();
        for(Monom i:p2.getElem()) {   /**
         multiplication of each monomial of p2 with -1
         **/
            i.setCoef(i.getCoef()*(-1));
        }
        IOp add=new Addition();
        /**
         *
         */
        r=add.compute(p1,p2); /**
         addition between p1 and -p2
         *
         */
        return r;

    }

}
```

Multiplication:

compute(Polinom p1,Polinom p2)-> returns a Result type object. The 2 polynomials are iterated through and the coefficient for every monomial in p1 is multiplied with the coefficient for every monomial in p2 and degrees are added together. After every monomial in p1 has been multiplied,we perfom an addition to make sure that there are no duplicates.

```
public class Multiply implements IOp {

    public Result compute(Polinom p1, Polinom p2) {
        Result r = new Result();
        Polinom q = new Polinom();

        IOp add = new Addition();
        for (Monom i : p1.getElem()) {
            /**
         * Multiplication of each coeff of p1 with each coeff of p2
         */
            Polinom p = new Polinom();
            for (Monom j : p2.getElem()) {
                Monom monom = new Monom( coef: i.getCoef() * j.getCoef(),  degree: i.getDegree() + j.getDegree());
                p.addMonom(monom);
            }
            r = add.compute(p, q);/**
             *Sum of the partial results from above
             */

            q = r.getResult();

        }
        return r;
```

Div:

compute(Polinom p1,Polinom p2)->returns a Result type object. First, we check if the degree of p1 is 0. If it is, the result is 0. Otherwise, while degree of p1 is greater than degree of p2, we do the division. A new monomial is created,whose coefficient is the result of the division of the first monomial of p1 and first monomial pf p2. And the degree is their difference. The new monomial is added to a polynomial,which is then multiplied with p2 and the result is substracted from p1. The new result and remainder are set.

Deriv:

compute(Polinom p1,Polinom p2)-> returns a Result type object. If the polynomial's degree is different from 0, then we iterate through p1's monomials and each is multiplied by its degree, and 1 will be substracted from the degree

```
ublic class Deriv implements IOp{

   public Result compute(Polinom p1, Polinom p2) {
       Result r=new Result();
       p2=new Polinom();
       if(p1.getElem().get(0).getDegree()!=0){
           for(Monom i:p1.getElem()) {
               Monom m=new Monom( coef: i.getCoef()*i.getDegree(), degree: i.getDegree()-1);
               /**
                * power remains -1 for non-x monomials,but is still displayed as 0
                */
               p2.addMonom(m);
           }
       }
       p1=p2;
       r.setResult(p1);
       return r;
   }
```

Integrate:

compute(Polinom p1,Polinom p2)->returns a Result type object. P1's iterated through and the coefficient of each monomial is divided by (degree+1) and 1 is added to the degree.

```
public class Integrate implements IOp{

   public Result compute(Polinom p1, Polinom p2) {
       Result r=new Result();
       p2=new Polinom();
       for(Monom i:p1.getElem())
       {
           p2.addMonom(new Monom( coef: i.getCoef()/(i.getDegree()+1), degree: i.getDegree()+1));
       }
       p1=p2;
       r.setResult(p1);
       return r;

   }
```

PolController:

actionPerformed(ActionEvent e)-> facilitates communication with PolView(). Data is taken as string and then is transformed in an object of type Polinom by inputTransf(String s). Using getInstance(s), we receive the name of the button(operation) pressed as a String, and it is then checked. If the operation is Division, then a check is performed, to make sure that the divided polynomial is different than 0 and if the degree of the divided polynomial is indeed greater than the one of the second polynomial. If the conditions are not met, the user will be signaled that there is an

error. Otherwise, the result and the remainder will be displayed. If the operation is any other one except division, the result is displayed(if input data is correct, of course).

inputTransf(String s)-> transforms input string into a Polinom

```
public Polinom inputTransf(String s) {
    Pattern p = Pattern.compile("(-?\\b\\d+)[xX]\\^(-?\\d+\\b)");
    Matcher m = p.matcher(s);
    Monom monom;
    Polinom pol = new Polinom();
    float c = 0;
    int d = 0;
    while (m.find()) {
        c = Float.valueOf(m.group(1));
        d = Integer.valueOf(m.group(2));
        monom = new Monom(c, d);
        pol.addMonom(monom);

    }
    return pol;
```

```
public IOp getInstance(String s) {
    switch (s) {
        case "+":
            return new Addition();
        case "-":
            return new Sub();
        case "*":
            return new Multiply();
        case ":":
            return new Div();
        case "Differentiate":
            return new Deriv();
        case "Integrate":
            return new Integrate();
        default:
            return null;

    }
}
```

setareRezultat(IOp op,Polinom p1,Polinom p2)->sets the corresponding value for the result and sets the remainder as NULL.

setareBipolinom(IOp,Polinom p1, Polinom p2)->sets the corresponding values for both the result and the remainder.

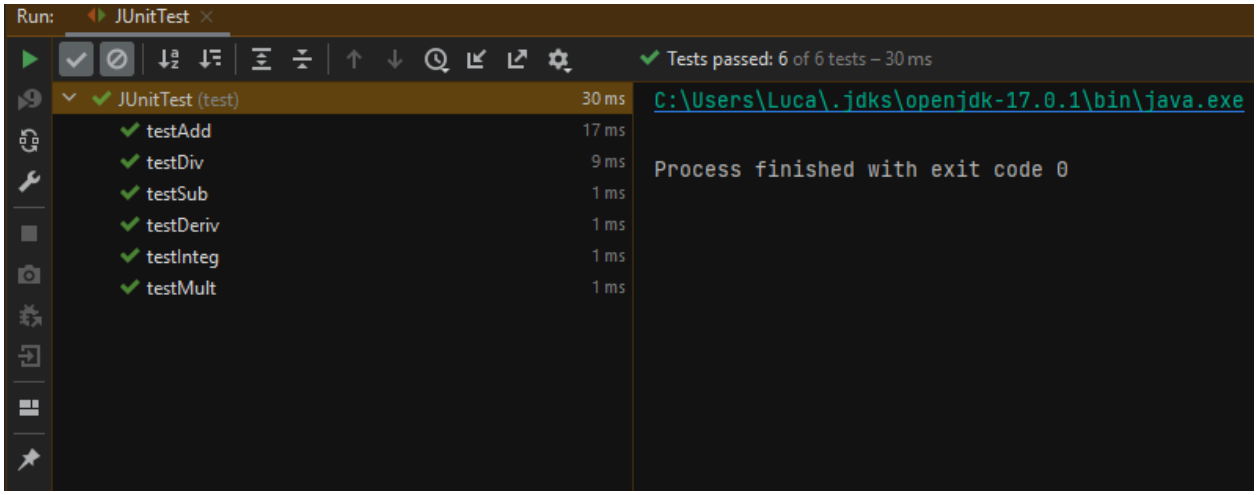
PolView:

As described in Chapter 3,the PolView class represents the interface of the project. Basically all it does is to “build” the interface, and make sure it can offer the input data to the controller and model, and the receive the results from them , in order to display them to the user.

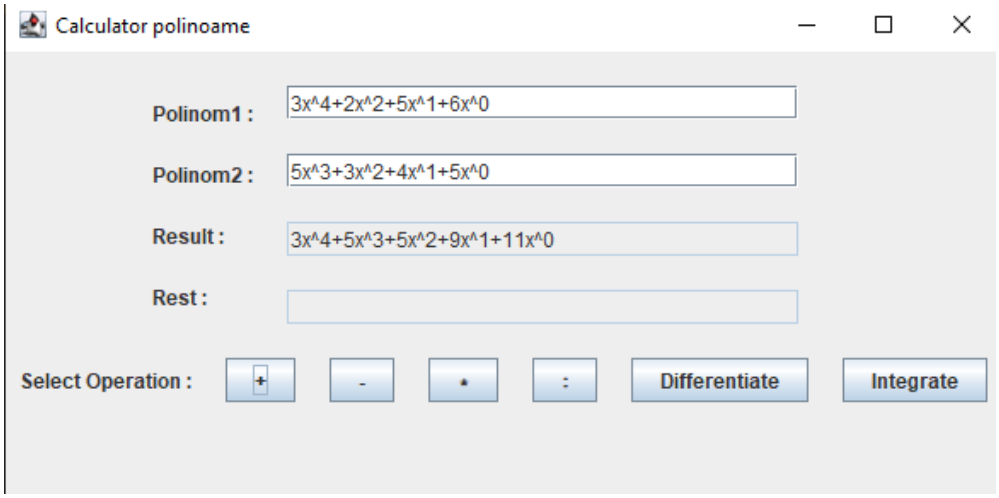
5. Results

Before the final modifications in the implementations, I used Junit Tests to make sure that everything is working accordingly.

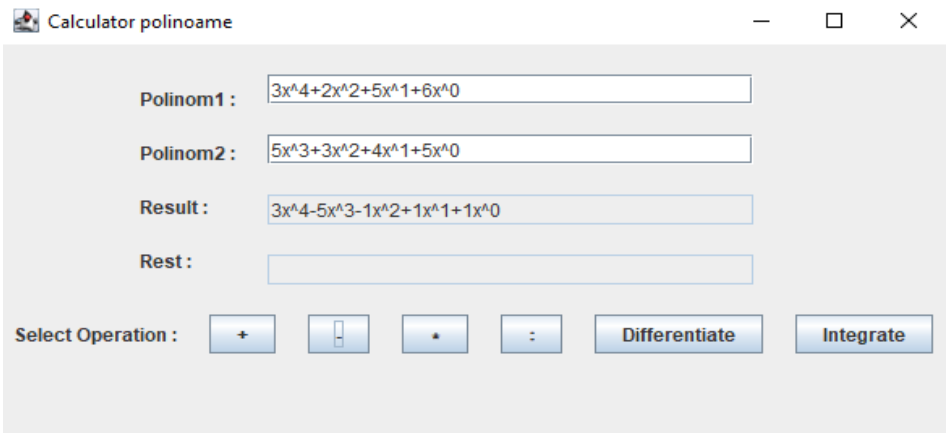
I have created 6 test methods, one for each operation. I've also used the method assertTrue(Boolean condition) to check if the condition was true, and therefore if the test was passed. And a snippet:



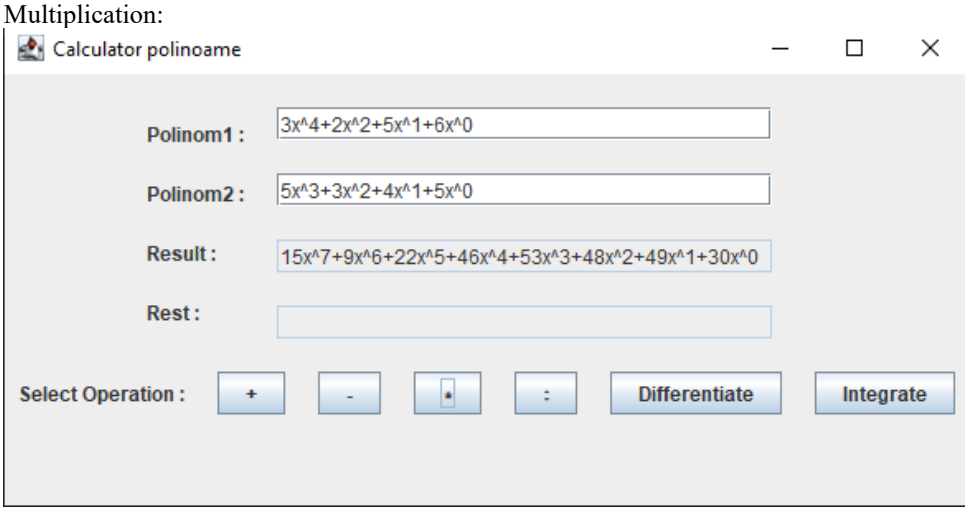
Addition:



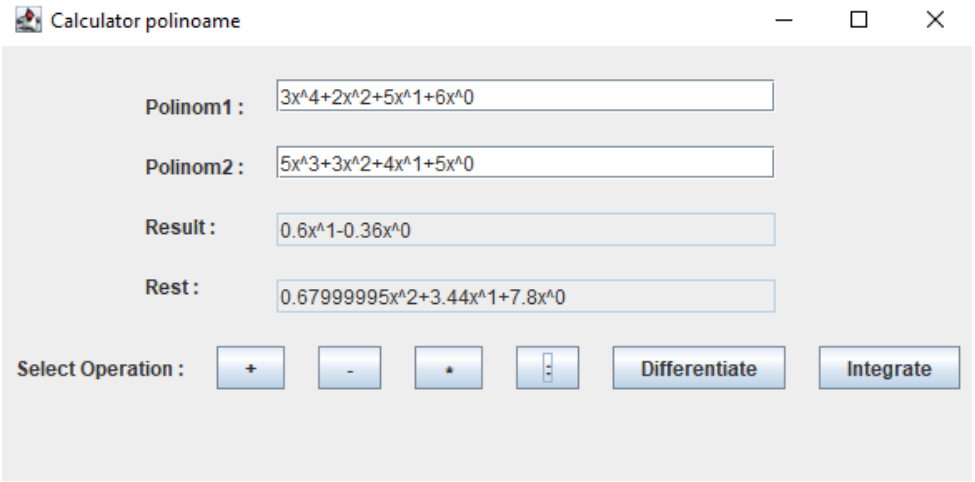
Substraction:



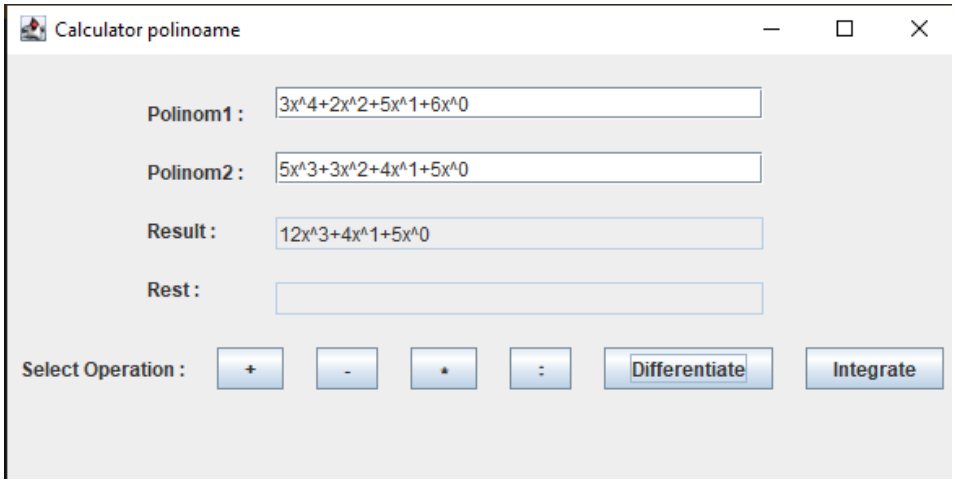
Multiplication:



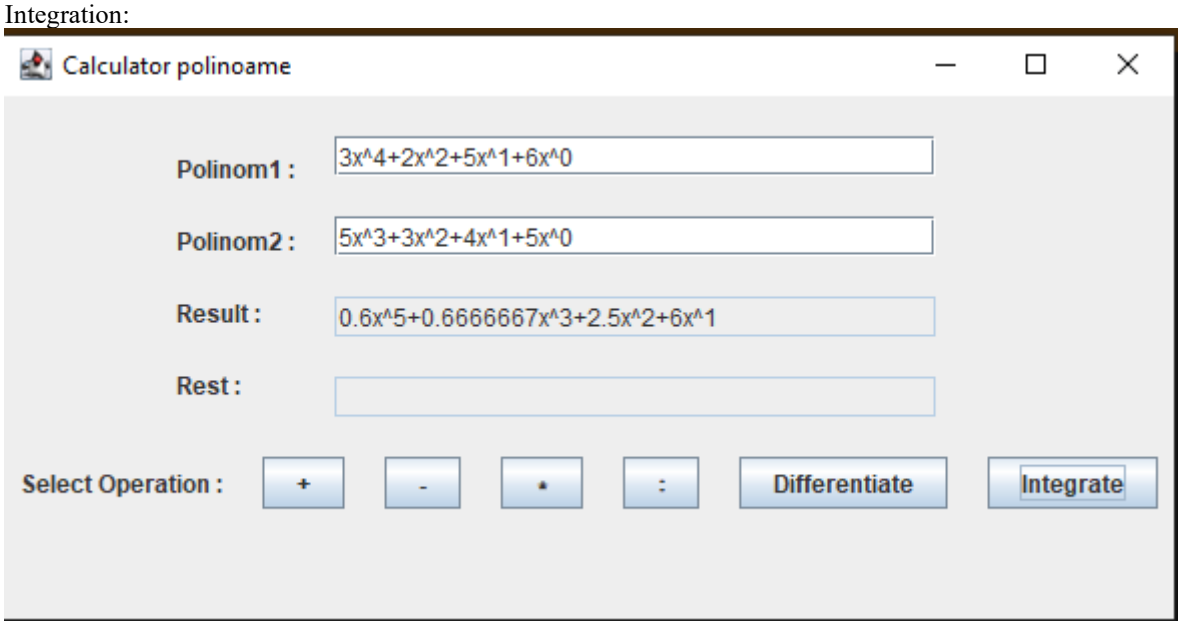
Division:



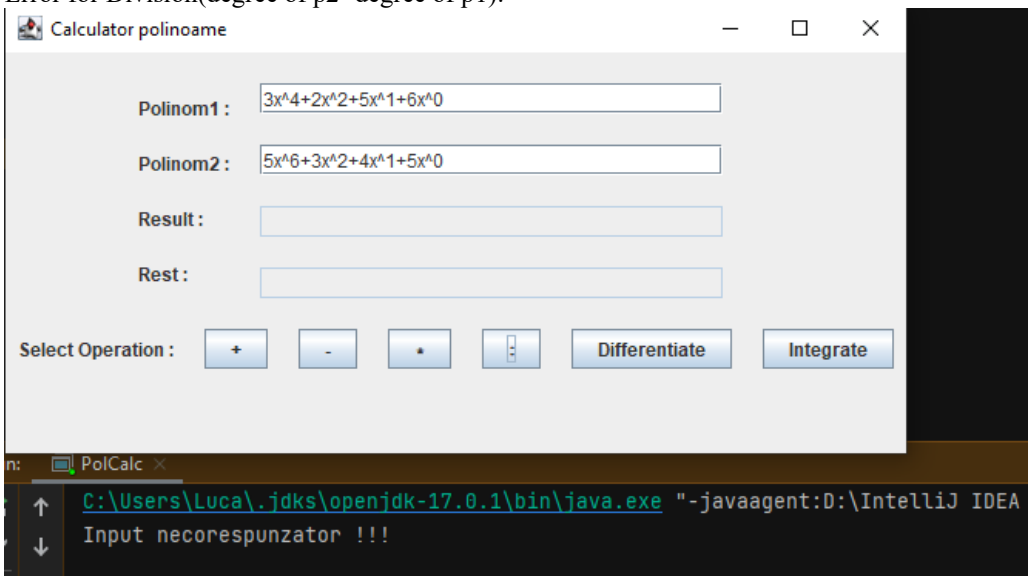
Differentiation:



Integration:



Error for Division(degree of p2>degree of p1):



6.Conclusions

I consider that after doing this assignment, I've laid more bricks on the foundation of OOP knowledge that I previously had, and I've learned how to do UML diagrams and use-case diagrams more efficiently. It also helped in the refreshing of some mathematical concepts. Improvements that could be made are:

- A prettier interface
- A Clear Button
- A warning if you introduce 0.

7.Bibliography

- https://www.geeksforgeeks.org/
- https://www.jetbrains.com/help/idea
- https://www.javatpoint.com/
- https://dsrl.eu/courses/pt/materials/A1_Support_Presentation.pdf