

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH



**TECHNICAL UNIVERSITY**  
OF CLUJ-NAPOCA

**FACULTY OF AUTOMATION AND COMPUTER SCIENCE**  
**COMPUTER SCIENCE DEPARTMENT**

Programming Techniques  
Assignment 1

# Polynomials Calculator

Teacher: prof. Ioan Salomie  
Teacher Assistant: Cristina Bianca Pop  
Student: Bucur Alexandra  
Group: 30424

2020-2021

## Contents

1. Assignment objective.....	3
2. Problem analysis, modeling, scenarios, use cases.....	4
Analyzing the problem .....	4
Modelling .....	4
Scenarios and Use cases .....	4
3. Design.....	6
Design decisions .....	6
UML diagrams.....	7
Data Structures .....	8
Class Design .....	8
Relationships, packages.....	9
Algorithms .....	10
User Interfaces .....	10
4. Implementation .....	13
Monomial Class.....	13
Polynomial Class .....	14
Operations Class .....	14
MainController Class.....	15
UI Class .....	15
App Class.....	15
5. Results.....	16
6. Conclusions.....	18
Further implementation .....	18
7. Bibliography .....	19

# 1. Assignment objective

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. Consider the polynomials of one variable and integer coefficients. It should have a dedicated graphical interface through which the user can insert polynomials, select the mathematical operation to be performed and view the result.

Sub-objectives:

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

## 1. Analyze the problem and identify requirements

This part will analyze the requirements and decide the modelling, scenarios and use cases. As functional requirements the user, which is also the primary actor of the project, should be able to insert 2 polynomials and perform several types of operations, such as: addition, subtraction, multiplication, division, derivation and integration. It will be presented in detail in [part 2](#) of the documentation.

## 2. Design the polynomial calculator

This part will analyze the design of the polynomial calculator. It will decide on a certain design pattern, decide on package configurations and the design of the classes.

It will also present the OOP design of the application, the UML class and package diagrams, the data structures used, the defined interfaces and the algorithms used (if applicable) will be presented.

This part will be presented in detail in [part 3](#).

## 3. Implement the polynomial calculator

This part will present in detail the implementation of the project. Each class will be described with important fields and methods. The implementation of the user interface will be described. This will be explained later, in [part 4](#).

## 4. Test the polynomial calculator

This part will present the result of the testing of the application. It will be presented later, in [part 5](#).

## 2. Problem analysis, modeling, scenarios, use cases

### Analyzing the problem

A polynomial can be written in the form:

$$P(X) = a_n * X^n + a_{n-1} * X^{n-1} + \dots + a_1 * X + a_0$$

where  $a_0, a_1, a_2, \dots, a_n$  are constants and  $x$  is the variable.

The polynomial consists of a list of certain terms, also called monomials, for example  $9x^{10}$  is a monomial. So, we can say that polynomial is composed of one or more monomials, in the form  $a_i x^i$ , where  $i$  is the degree of the monomial.

Such a representation will allow performing operations, such as: addition, multiplication, subtraction, division, derivation and integration.

### Modelling

The user will be able to perform certain operations available in the calculator by introducing in the dedicated interface some polynomials. For the derivation and integration, the first polynomial will be considered.

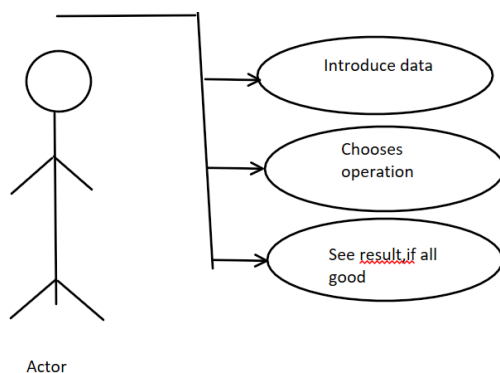
The user will also have an info button in order to learn how to use the calculator at the beginning. That button will display the necessarily information regarding the use of the polynomial calculator.

Available operations:

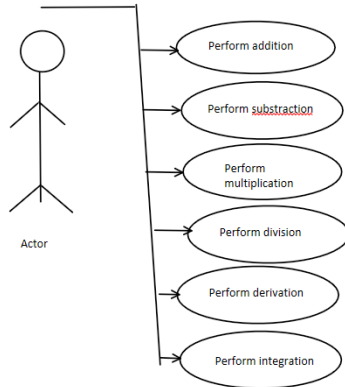
- Addition of two polynomials
- Subtraction of two polynomials
- Multiplication of two polynomials
- Division of two polynomials
- Differentiation of a polynomial
- Integration of a polynomial

The result of the chosen operation will be displayed in the interface, in the designated place. If the polynomial that is introduced, it will display a proper error message. The operations that cannot be computed will be signaled properly, also.

### Scenarios and Use cases



The above use case represents the basic form of functioning of the program. At first the user (actor) will input the data (the 2 polynomials), after this, the user will choose between the available operations and will see the result of the operation. If one of the polynomial is not correct, the application will signal this. Moreover if one operation cannot be performed (such as division by zero), the calculator will display an error and no result will be presented in this case.



Basic operations that the user can choose between.

### Addition

- **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 OK buttons for both polynomials in order to be validated
3. The user selects the “addition” operation
4. The polynomial calculator performs the addition of the two polynomials and displays the result

**Alternative Sequence:** Incorrect polynomials

- The user inserts incorrect polynomials (e.g. with 2 or more variables)
- An error message will be displayed for the polynomial that was incorrect
- The scenario returns to step 1

### Subtraction

- **Use Case:** subtract polynomials: like addition, but polynomials are subtracted

### Multiply

- **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 OK buttons for both polynomials in order to be validated
3. The user selects the “multiplication” operation
4. The polynomial calculator performs the multiplication of the two polynomials and displays the result

**Alternative Sequence:** Incorrect polynomials

- The user inserts incorrect polynomials (e.g. with 2 or more variables)
- An error message will be displayed for the polynomial that was incorrect
- The scenario returns to step 1

### Division

- **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 OK buttons for both polynomials in order to be validated
3. The user selects the “division” operation
4. The polynomial calculator performs the division of the two polynomials and displays the result

**Alternative Sequence:**

- Incorrect polynomials
  - The user inserts incorrect polynomials (e.g. with 2 or more variables)
  - An error message will be displayed for the polynomial that was incorrect
  - The scenario returns to step 1
- Impossible to perform division
  - The divider is 0 => division by 0 is impossible->error will be displayed.
  - The scenario returns to step 1

**Derivation**

- **Use Case:** derivate polynomials

**Primary Actor:** user

**Main Success Scenario:**

1. The user inserts 1 polynomial in the graphical user interface.
2. The user clicks on OK button for polynomial1 in order to be validated
3. The user selects the “derivate” operation
4. The polynomial calculator performs the differentiation of the polynomial and displays the result

**Alternative Sequence:** Incorrect polynomials

- The user inserts incorrect polynomials (e.g. with 2 or more variables)
- An error message will be displayed for the polynomial that was incorrect
- The scenario returns to step 1

**Integrate**

- **Use Case:** Integrate polynomials

**Primary Actor:** user

**Main Success Scenario:**

1. The user inserts 1 polynomial in the graphical user interface.
2. The user clicks on OK button for polynomial1 in order to be validated
3. The user selects the “Integrate” operation
4. The polynomial calculator performs the integration of the polynomial and displays the result

**Alternative Sequence:** Incorrect polynomials

- The user inserts incorrect polynomials (e.g. with 2 or more variables)
- An error message will be displayed for the polynomial that was incorrect
- The scenario returns to step 1

## 3. Design

### Design decisions

- This project has a MVC design pattern., which divides the application into three areas: processing, output and input.

**Context**

- Many software systems deal with finding data from a repository and displaying the data to the users through a graphical user interface (GUI)
- **Disadvantages:**

- The GUI changes more often than the business logic implementation -> if they are implemented in the same class then each time the GUI changes the business logic is changed
- The business logic can not be reused
- The code is complex and difficult to maintain

### Model components

- The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application.
- main idea: encapsulates core data and functionality.

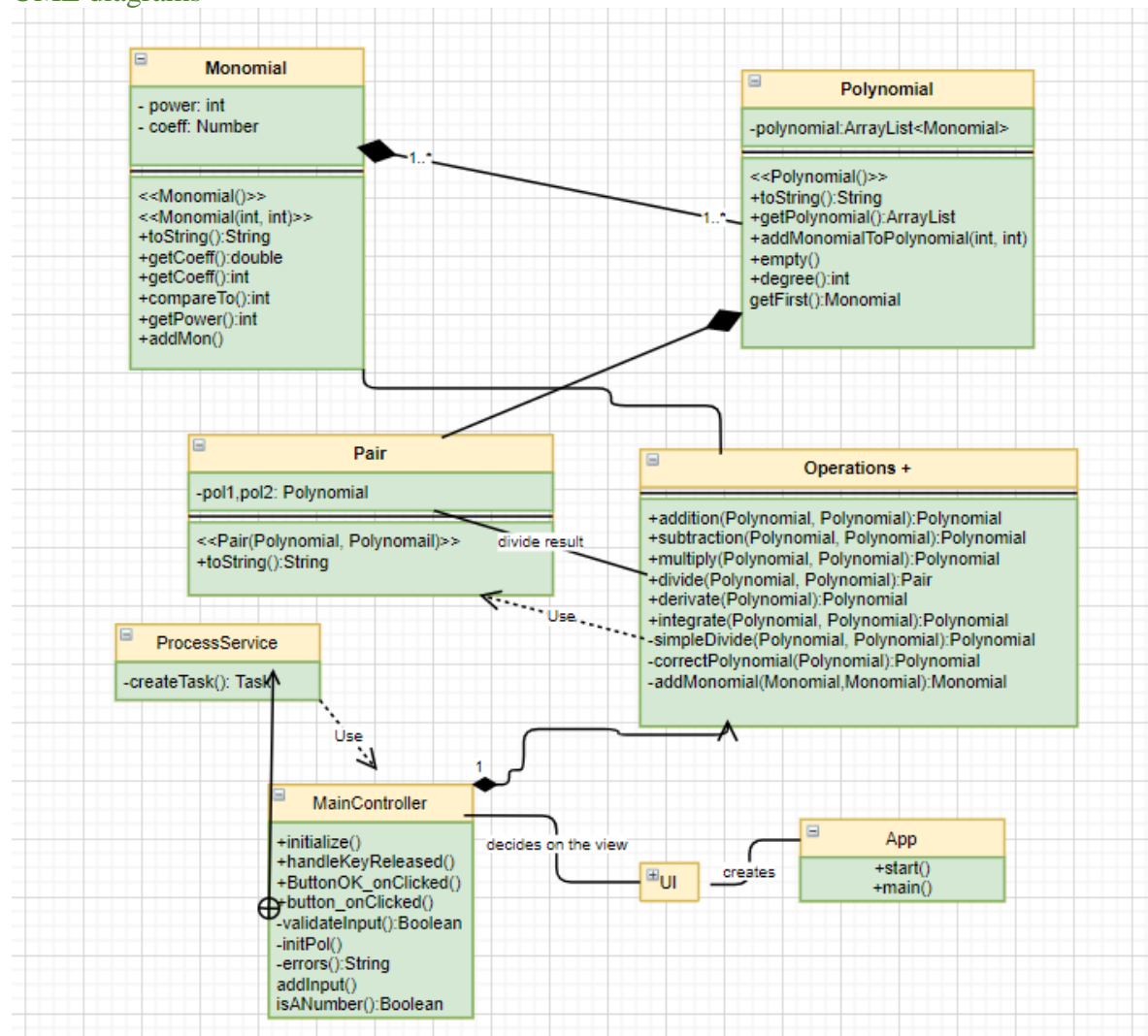
### View components

- Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
- main idea: display information to the user - obtains the data it displays from the model

### Controller

- Each view has an associated controller component. Controllers receive input, usually as events that denote mouse movement, activation of mouse buttons or keyboard input. Events are translated to service requests, which are sent either to the model or to the view.
- main idea: accepts input and converts it to commands for the model or view.

### UML diagrams



## Data Structures

The data structures that have been used in this application are either primitive data types, such as integers or doubles, or the type Number that will allow a certain number to be easily converted into any primitive data type, such as integer, float, doubles and so on.

The application also uses a newly created class, called Monomial that represents a part of the polynomial and has a coefficient and a power. Due to this, the Polynomial class has a list of monomials. For the list implementation, I chose to use an ArrayList type, instead of a simple array, because they are more efficient from the point of view of memory management and provide a faster access to their content. The code is also easier to understand and cleaner. Moreover, the size of an ArrayList is not fixed so we do not have to worry about incrementing the length of the array or getting an “Overflow” exception.

## Class Design

The whole idea of splitting your program into classes is based on a general rule named divide and conquer. The application divides a problem into smaller problems and then those small classes solve the simple and well-known problems.

Because I followed the MVC architecture, my program consists of 4 parts: Main package, Controller Package, Model Package, View Package.

- 1) The Model package – contains the logic of the application

### **Monomial Class**

Fields:

- power:int
- coeff: Number

Constructors:

- public Monom (int power, int coeff) : the constructor that initializes the monomials with the transmitted coefficient and exponent
- public Monom() : default constructor

Methods:

- public int getPower() : returns the degree of the monomial
- public void setPower(int power) : allows us to set the degree of a monomial
- public double getCoefficient() : returns the coefficient of the monomial
- public void setCoefficient(int coefficient) : allows us to set the coefficient of the monomial
- public void setCoefficient(double coefficient) : allows us to set the coefficient of the monomial
- public void toString() : displays one monomial on the screen
- public void compareTo(Object o): compares 2 monomials

### **Polynomial Class**

Fields:

polynomial: ArrayList<Monomial>

Constructor:

- public Polinom() : default constructor

Methods:

- public ArrayList<Monom> getPolinom() : returns the actual polynomial, as an ArrayList of Monoms



- public void getCanonicalForm() : orders the monomials in the “natural” order, highest degree in front
- public void toString() : displays the polynomial
- public empty(): removes all monomials from a polynomial
- public degree(): calculates the deg of a polynomial
- public getFirst(): returns monomial with highest degree

### Operations class

- no constructors, all operations that are needed for the polynomial, presented in detail in [part 4](#)

### Pair Class

- this class is used for returning both the quotient and remainder during the division operation.

Fields:

pol1,pol2: Polynomials

Constructors:

public Pair(p1,p2):set both polynomials

Methods:

Public String toString(): returns a pair of polynomials in a String format

- 2) The View package– contains the graphical interface.

### UI Class

- JavaFX code, displays all buttons and labels on the screen.

- 3) The Control package– this contains the linking between the model and the view

### MainController Class

- deals with events for buttons and labels, but also validating data.

Most important method: validateData()

- has a subclass **ProcessService** that help displaying labels for a certain amount of time

- 4) The Main package

### App class

- App class which runs the start() method in order to “turn on” the application.

## Relationships, packages

Java packages help in organizing multiple modules and group together related classes and interfaces.

In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently relating the user interface to underlying data models. The MVC pattern is widely used in program development with programming languages such as Java, Smalltalk, C, and C++.

The application, which is implementing the Model-View-Controller architectural design pattern, will have 4 packages: Controller Package, Model Package, View Package and Main Package.

**The Main package:**– contains a single class, named App, which contains the main() method and will be the one that calls and produces the page for the user interface.

**The Controller package:** -Will have only one class, MainController.

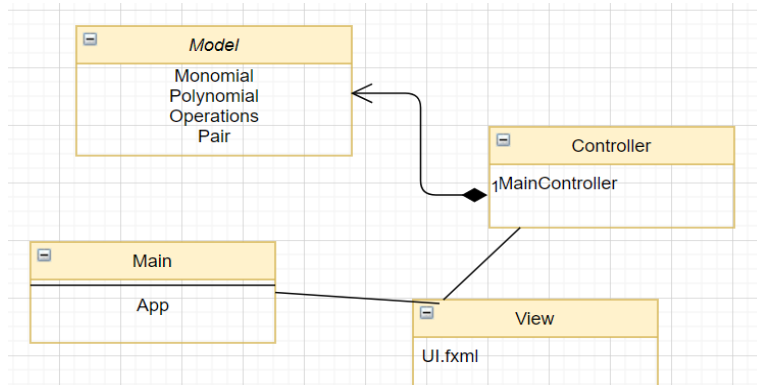
- This class will deal with linking the user interface (available in View package) and the implementation in the backend of the application (available in Model package).

- it interconnects the model and the view

**The View package:** – Contains a single class which represents the appearance of the user interface (GUI)

**The Model package:**

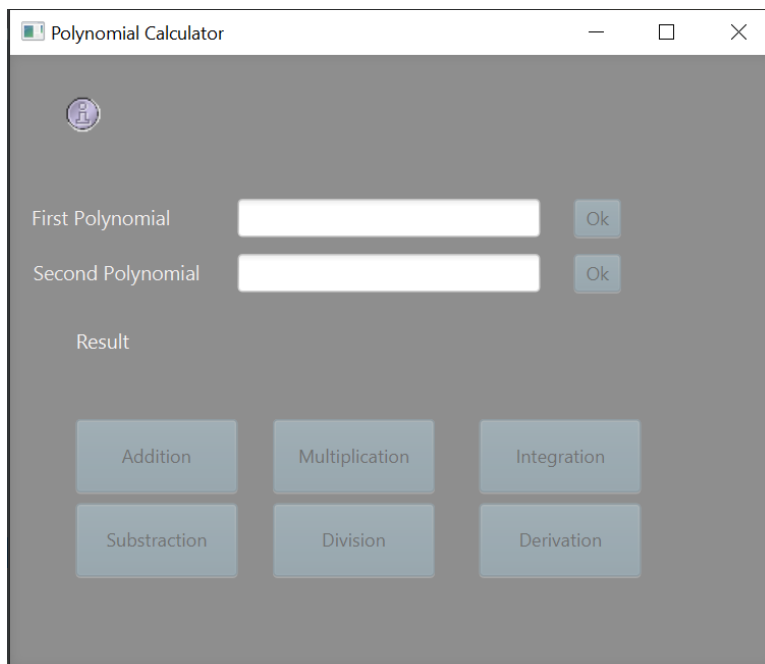
- Contains classes that deal with the model of the problem
- Defines the classes for Polynomial and Polynomial
- Computes the actual operations on the polynomials



## Algorithms

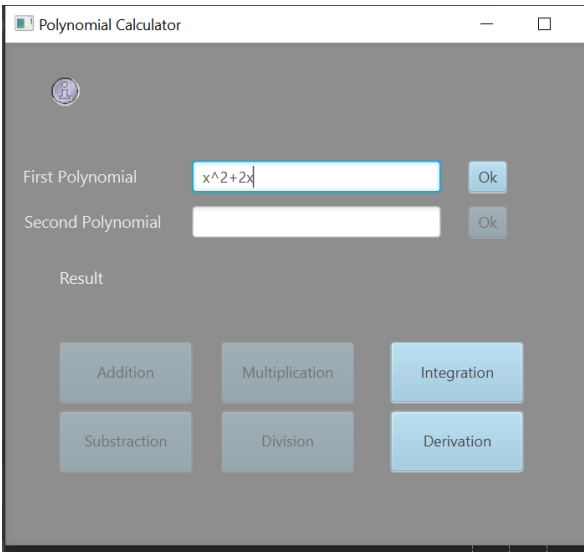
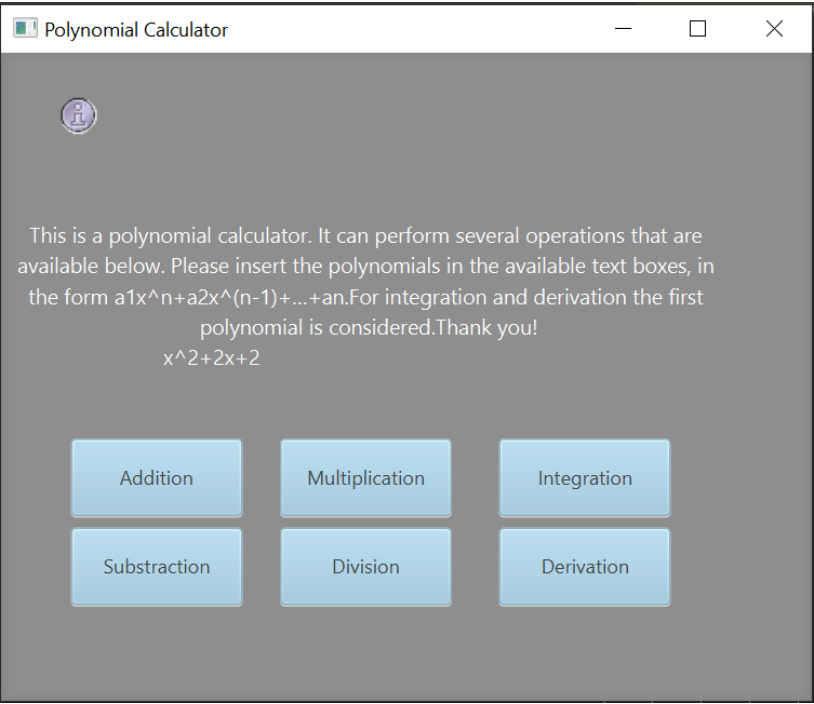
The algorithms regarding the operations of the polynomials are presented in detail when describing the Operations class.

## User Interfaces

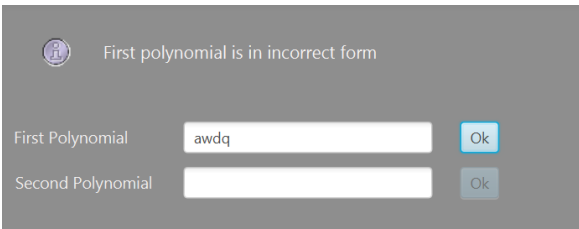


At the beginning the interface that the user is able to see will look like this. All buttons are disabled because no polynomial was inserted. There is only one button that is available, the one in the upper left corner.

By pressing that button, information about the polynomial will be displayed, for an amount of time.



As the data for the polynomials are introduced more buttons will appear to be available. So, now, the user will be able to perform the operations desired.

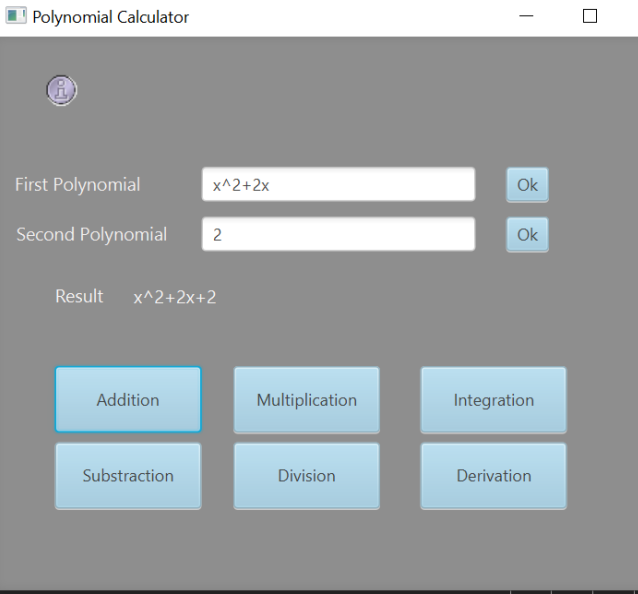


First polynomial is in incorrect form

First Polynomial

Second Polynomial

Moreover, on the upper part, there will be displayed error messages, if the inserted polynomials are not correct, or certain operations cannot be computer(ex: division by zero).



Polynomial Calculator

First Polynomial

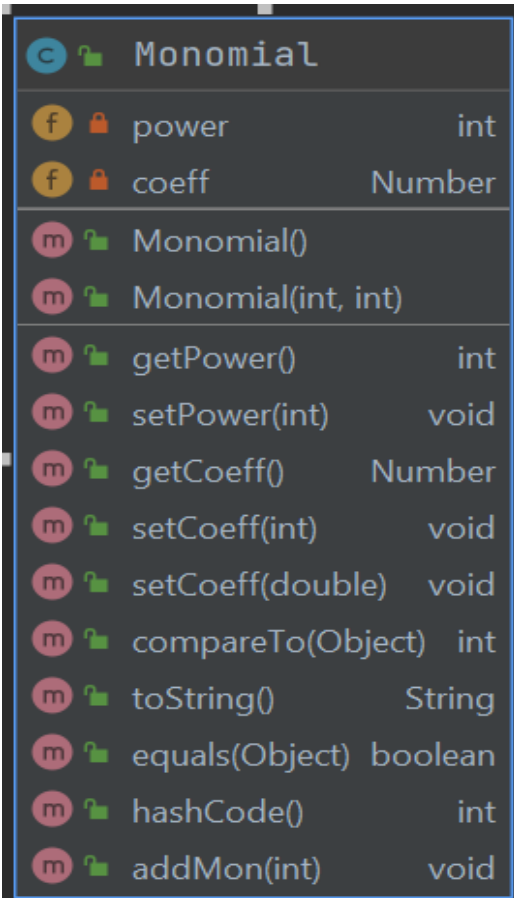
Second Polynomial

Result  $x^2+2x+2$

Addition	Multiplication	Integration
Substraction	Division	Derivation

## 4. Implementation

### Monomial Class



<b>c</b>	<b>Monomial</b>	
<b>f</b>	power	int
<b>f</b>	coeff	Number
<b>m</b>	Monomial()	
<b>m</b>	Monomial(int, int)	
<b>m</b>	getPower()	int
<b>m</b>	setPower(int)	void
<b>m</b>	getCoeff()	Number
<b>m</b>	setCoeff(int)	void
<b>m</b>	setCoeff(double)	void
<b>m</b>	compareTo(Object)	int
<b>m</b>	toString()	String
<b>m</b>	equals(Object)	boolean
<b>m</b>	hashCode()	int
<b>m</b>	addMon(int)	void

A polynomial is composed by one or more terms, which are called monomials. This class has two instance variables, an int exponent and a Number coefficient.

This class has 2 fields, one for the power (int) and one for the coeff (Number). It is of type Number because during the operations some coefficient might become doubles in order to obtain the correct result.

This class has 2 Constructors, one with no parameter and one with both of its fields( power and coefficient).

There are some getters and setters for all the fields, but the most important one is the toString() method that displays the monomial in a readable manner.

The compareTo() and equals() methods also help in deciding whether 2 monomials are equal or not, they were overridden to only take into consideration the power of the monomial. They are used in sorting 1 polynomial in order to obtain the canonical form and in order to perform the right operation for 2 monomials of the same degree.

## Polynomial Class

Polynomial	
polynomial	ArrayList<Monomial>
Polynomial()	
toString()	String
getPolynomial()	ArrayList<Monomial>
setPolynomial(ArrayList<Monomial>)	void
addMonomialToPolynomial(Monomial)	void
addMonomialToPolynomial(int, int)	void
getCanonicalForm()	void
empty()	void
degree()	int
getFirst()	Monomial

The Polynomial class contains an ArrayList of Monomials that create the actual polynomial.

At the beginning the polynomial that is created is empty. Besides the getters and setters, the important methods are: addMonomialToPolynomial(), getCanonicalForm(), empty(), degree() and getFirst().

- addMonomialToPolynomial() method simply adds in the array one more monomial, so the size of the ArrayList increases. There are 2 such methods, but with different parameters.
- getCanonicalForm() sorts the polynomial in decreasing order, comparing only the powers
- degree() computes the degree of the polynomial, after it was sorted-> used for division
- getFirst()-> returns the monomial with the biggest degree

## Operations Class

Operations	
add(Polynomial, Polynomial)	Polynomial
subtract(Polynomial, Polynomial)	Polynomial
addMonomial(Monomial, Monomial, int)	Monomial
multiply(Polynomial, Polynomial)	Polynomial
correctPolynomial(Polynomial)	Polynomial
derivate(Polynomial)	Polynomial
integrate(Polynomial)	Polynomial
divide(Polynomial, Polynomial)	Pair
simpleDivide(Polynomial, Polynomial)	Polynomial

This class deals with all the operations regarding the polynomials. It is an abstract class since it does not need instantiation. All methods display The polynomials in canonical form.

- add()

This method performs the addition of 2 polynomials, based on the addition done in mathematics.

It goes through 1 of the polynomials and if there exists a monomial with the same power in the second polynomial, then it will add the; otherwise the monomial from polynomial 1 will be added as it is in the result. At the end all not used monomials from polynomial 2 will be added to the result as they are.

- subtract()

Similar to add method, but instead of adding 2 monomials, it subtracts them.

- Multiply()

This method is based on multiplication on mathematics. It will go and multiply each monomial from the first polynomial with each monomial from the second polynomial.

The multiplication of 2 monomials: the power of the result becomes the sum of power of the 2 monomials and the coefficient of the result becomes the multiplication of coefficient of the 2 monomials.

At the end of there are in the resulted polynomial some monomials with the same degree, they will be added and the polynomial will be transformed in the canonical form.

- Divide()

The divide method test to see if the division can be performed(no division with 0). If so SimpleDivision() will perform the cases in which the second polynomial is 1 coefficient or if the second polynomial has a degree bigger than the first one.

The rest of the division is done regarding the mathematical rules:

**Step 1** - Order the monomials of the two polynomials  $P$  and  $Q$  in descending order according to their degree.

**Step 2** – Divide the first monomial of  $P$  to the first monomial of  $Q$  and obtain the first term of the quotient

**Step 3** – Multiply the quotient with  $Q$  and subtract the result of the multiplication from  $P$  obtaining the remainder of the division

**Step 4** – Repeat the procedure from step 2 considering the remainder as the new dividend of the division, until the degree of the remainder is lower than  $Q$ .

- Derivate()

Each monomial is computed having in consideration the rules from mathematics regarding derivation.

$$\frac{d}{dx}(a_n * X^n + a_{n-1} * X^{n-1} + \dots + a_1 * X + a_0) = n * a_n * X^{n-1} + (n-1) * a_{n-1} * X^{n-2} + \dots + a_1$$

- Integrate()

Each monomial is computed having in consideration the rules from mathematics regarding integration.

$$\int a_n * X^n + a_{n-1} * X^{n-1} + \dots + a_1 * X + a_0 = \int a_n * X^n dx + \int a_{n-1} * X^{n-1} dx + \dots + \int a_1 * X dx + \int a_0 dx$$

## MainController Class

This contains the linking between the model and the view This is a very important class because it acts on both model and view. It controls the data flow into model object and updates the view whenever data changes.

It is included in the Controller package and mainly contains methods for the functioning of the buttons/labels and all that can be displayed onto the screen, so that the user can see.

It has no constructor, only an initialize() method that sets the buttons to disabled at the beginning of the application. The rest of the methods are events and eventHandlers for the buttons.

## UI Class

The UI class is written in JAVAFX and is responsible for the appearance of the application. The view corresponds with the controller through some functions that triggers the functionality of the buttons and everything that is displayed there.

## App Class

This class contains the main() of the application. At runtime the application will start from this point.

## 5. Results

The application is an user friendly and helpful application to perform basic polynomial operations, as long as the user obeys the input conventions and it is familiar with polynomial operations such as: addition, subtraction, multiplication, division, differentiation and integration. The Junit testing represents another package from the application, especially dedicated to testing. Several methods were tested, with successful results.

```
@Before
public void initData() {
    monomial = new Monomial( power: 2, coeff: 2);
    monomial2 = new Monomial( power: 0, coeff: 2);
    monomial3 = new Monomial( power: 2, coeff: 1);
    monomial4 = new Monomial( power: 2, coeff: 0);
    monomial4.setCoeff(1.23);
    monomial5 = new Monomial( power: 1, coeff: 2);
    monomial5.setCoeff(1.0);
    monomial6 = new Monomial( power: 0, coeff: -1);
}

@Test
public void testToStringMonomial1() {
    String result = monomial.toString();
    Assert.assertEquals( message: "Values differ", result, actual: "+2x^2");
}

@Test
public void testToStringMonomial2() {
    String result = monomial2.toString();
    Assert.assertEquals( message: "Values differ", result, actual: "+2");
}
```

For the Monomial class, the important methods, such as toString() and compareTo() were tested, the rest were only getters and setters. Different cases were taken into consideration, such as: displaying “x” as “x”, not “1x^1”, or displaying a constant correctly. On the left is just an example. The initData() method is performed each time, before a test.

The Operation class was also tested with successful results. All cases were taking into consideration, such as division by zero, resulting a zero after a subtraction, or other special cases. The code below demonstrates on the left:

```
@Test
public void testSubtract() {
    result = Operations.subtract(p1, p2);
    Assert.assertEquals( expected: "-5x^5+4x^2+9x-7", result.toString());
}

@Test
public void testSubtractResultsZero() {
    result = Operations.subtract(p1, p1);
    Assert.assertEquals( expected: "0", result.toString());
}

@Test
public void testMultiply() {
    result = Operations.multiply(p1, p2);
    Assert.assertEquals( expected: "20x^7+10x^6+15x^5+20x^3+26x^2-x+30", result.toString());
}
```

```
@Test
public void testIntegrateOnlyIntegersAsResult() {
    p2.empty();
    p2.addMonomialToPolynomial( power: 0, coeff: 1);
    p2.addMonomialToPolynomial( power: 2, coeff: 3);
    p2.addMonomialToPolynomial( power: 3, coeff: 4);
    result = Operations.integrate(p2);
    Assert.assertEquals( expected: "x^4+x^3+x", result.toString());
}

@Test
public void testIntegrateRandomPolynomial() {
    result = Operations.integrate(p2);
    Assert.assertEquals( expected: "0.83x^6-3.5x^2+10x", result.toString());
}

@Test
public void testDivideDivisionByZero() {
    p2.empty();
    Pair pair = new Pair();
    pair = Operations.divide(p1, p2);
    Assert.assertEquals( expected: "Quotient: impossible Remainder: impossible", pair.toString());
}
```

The Polynomial class and Controller class also contain their own Test Classes, in which the most important methods are verified, all with successful results.



What is to be tested	Input Data	Expected Output Data	Actual Output Data	Test Result
Monomial toString()	2 2	+2x <sup>2</sup>	+2x <sup>2</sup>	Success
Monomial toString()	0 2	+2	+2	Success
getCanonicalForm()	1+8x <sup>3</sup> -8x <sup>9</sup> +9x	-8x <sup>9</sup> +8x <sup>3</sup> +9x+1	-8x <sup>9</sup> +8x <sup>3</sup> +9x+1	Success
empty()	1+8x <sup>3</sup> -8x <sup>9</sup> +9x	0	0	Success
Degree()	-8x <sup>9</sup> +8x <sup>3</sup> +9x+1	9	9	Success
Degree() for empty pol	0	0	0	Success
getFirst()	2x <sup>2</sup> +1+100x <sup>4</sup>	100x <sup>4</sup>	100x <sup>4</sup>	Success
Addition	P1= 2x+4x <sup>2</sup> +3 P2= 5x <sup>5</sup> +10-7x	5x <sup>5</sup> +4x <sup>2</sup> -5x+13	5x <sup>5</sup> +4x <sup>2</sup> -5x+13	Success
Subtraction	P1= 2x+4x <sup>2</sup> +3 P2= 5x <sup>5</sup> +10-7x	-5x <sup>5</sup> +4x <sup>2</sup> +9x-7	-5x <sup>5</sup> +4x <sup>2</sup> +9x-7	Success
Multiplication	P1= 2x+4x <sup>2</sup> +3 P2= 5x <sup>5</sup> +10-7x	20x <sup>7</sup> +10x <sup>6</sup> +15x <sup>5</sup> - 28x <sup>3</sup> +26x <sup>2</sup> - x+30	20x <sup>7</sup> +10x <sup>6</sup> +15x <sup>5</sup> - 28x <sup>3</sup> +26x <sup>2</sup> -x+30	Success
Derivation of the 1 <sup>st</sup> Polynomial	P1= 2x+4x <sup>2</sup> +3	8x+2	8x+2	Success
Derivation of the 2 <sup>nd</sup> Polynomial	P2= 5x <sup>5</sup> +10-7x	25x <sup>4</sup> -7	25x <sup>4</sup> -7	Success
Multiplication with 0	P1=0 P2=5x <sup>4</sup>	0	0	Success
Subtraction where result is 0	P1= 2x+4x <sup>2</sup> +3 P2= 2x+4x <sup>2</sup> +3	0	0	Success
Derivation of constants	10	0	0	Success
Integrate()	P2= 1+2x <sup>3</sup> +3x <sup>4</sup>	x <sup>4</sup> +x <sup>3</sup> +x	x <sup>4</sup> +x <sup>3</sup> +x	Success
Integrate()	P2= 5x <sup>5</sup> +10-7x	0.83x <sup>6</sup> -3.5x <sup>2</sup> +10x	0.83x <sup>6</sup> -3.5x <sup>2</sup> +10x	Success
Divide() by 0	P1 = 2x+4x <sup>2</sup> +3 P2 = 0	Quotient: impossible Remainder: impossible	Quotient: impossible Remainder: impossible	Success
Divide()	P1 = 3x <sup>2</sup> +5x+2 P2 =2x+1	Quotient: 1.5x+1.75 Remainder: 0.25	Quotient: 1.5x+1.75 Remainder: 0.25	Success
ValidateData()	x <sup>2</sup> + 5x	true	true	Success
ValidateData()	2	true	true	Success
ValidateData()	Xx; x <sup>2</sup> +a; x <sup>2</sup> ++56	false	false	Success

## 6. Conclusions

This project was a good exercise in remembering the OOP concepts learned in the first semester, but also learning some new concepts regarding design patterns, and the MVC architectural pattern.

It also helped me understand that even though some operations are easily computed by the human brain, here, when writing an algorithm or writing code we have to think of all possible cases in order to make an application as good as possible.

Time management and the steps of designing the project are important. It is not a good practice to directly start coding and then, later, change many things in the code because you realized they are not right.

### Further implementation

- Make the interface more appealing.
- Make the program allow polynomials for any variable, not only "x".
- Compute the value of a polynomial in a certain point
- Find the roots of the polynomial
- Implement not only polynomials, but functions such as sin, cos, exponent, power and create a calculator for such functions.

## 7. Bibliography

- <https://app.diagrams.net/>
- [https://www.youtube.com/watch?v=UI6lqHOVHic&ab\\_channel=Lucidchart](https://www.youtube.com/watch?v=UI6lqHOVHic&ab_channel=Lucidchart)
- [https://docs.oracle.com/javafx/2/get\\_started/jfxpub-get\\_started.htm](https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm)
- <https://www.baeldung.com/javafx>
- <https://docs.oracle.com/javase/tutorial/essential/regex/>
- <https://www.baeldung.com/regular-expressions-java>
- <https://www.vogella.com/tutorials/JUnit/article.html>
- <https://www.baeldung.com/junit-5> <https://google.github.io/styleguide/javaguide.html>