DOCUMENTATION

ASSIGNMENT No. 1

STUDENT: Mureșan Salomeea
GROUP: 30424

CONTENTS

1. Assignment objective

i. The main objective of this project is to implement a polynomial calculator with dedicated graphical interface which the user can insert polynomials, select the mathematical operation (addition, subtraction, multiplication, derivative and integration) to be performed and view the result.

ii. The secondary objectives are:

objective	description	chapter
Implement MVC device pattern	The Model-View-Controller is a well-known software architectural pattern ideal to implement user interfaces on computers by dividing an application intro three interconnected parts. Main goal of Model-View-Controller, also known as MVC, is to separate internal representations of an application from the ways information are presented to the user.	3. Design: 3.2. Package Diagram
Find a way to represent the polynomial in Java	I decided to consider the polynomial as a list of monomials (each having an integer coefficient and power).	3. Design: 3.1. Theoretical considerations
Design the user interface and the controller	I created it using Java Swing.	
Implement business logic	Code is written in Java.	

2. Problem analysis, modeling, scenarios, use cases

As functional requirements, the polynomial calculator should be able to:

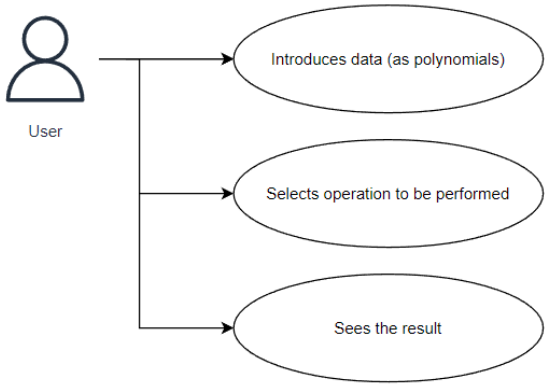
- Read one or two polynomials, given as input from the user;
- Compute the specified operation, chosen by the user by clicking one of the buttons from the GUI;
- Display the result correctly on the screen.

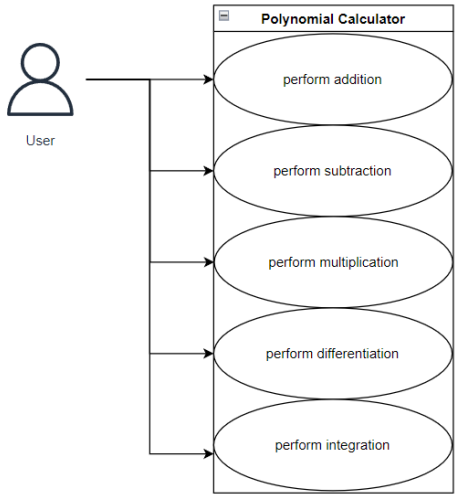
The user only has access to the graphical user interface, so all interactions with the user will be made through it.

a. Use Case Diagrams

The actor (in our case, the user that interacts with the application) can perform several actions on the input polynomials:

- Addition
- Subtraction
- Multiplication
- Differentiation (for the first polynomial)
- Integral (also for the first polynomial)





Use case: **ADD, SUBTRACT, MULTIPLY**
Primary actor: user
Success scenario steps:
1. The user introduces the first polynomial, each term written as: (+/- integer coefficient) x ^ (integer power)
2. The user introduces the second polynomial, respecting the input pattern
3. The user clicks on a button having the symbol + OR – OR *
4. The calculator checks the input text and converts it to an object from class Polynomial
5. The calculator computes the operation selected by the user
6. The result is displayed

Use case: **INTEGRATE, DIFFERENTIATE**
Primary actor: user
Success scenario steps:
1. The user introduces the first polynomial, each term written as: (+/- integer coefficient) x ^ (integer power)
2. The user clicks on a button having the symbol containing the words “differentiate” or “integrate”
3. The calculator checks the input text and converts it to an object from class Polynomial
4. The calculator computes the operation selected by the user
5. The result is displayed

3. Design

3.1. Theoretical considerations

A polynomial expression is an expression that can be built from constants and symbols (called variables or indeterminates) by means of addition, multiplication and exponentiation to a non-negative power. A polynomial in a single indeterminate x can always be written (or rewritten) in the form:

$a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_2 x^2 + a_1 x + a_0$

where $a_0, \ldots a_n$ are constants that are called coefficients of the polynomial, and x is the indeterminate. The world “indeterminate” means that x represents no particular value, although any value may substitute it. This can also be expressed more concisely by using summation notation:

$$\sum_{k=0}^{n} a_k x^k$$

Arithmetic:

Polynomials can be added or subtracted using the associative law of addition, grouping all their terms together into a single sum, possibly followed by reordering, using the commutative law, and combining of terms with the same power. When polynomials are added or subtracted together, the result is another polynomial.

Polynomials can also be multiplied. To expand the product of two polynomials into a sum of terms, the distributive law is repeatedly applied, which results in each term of one polynomial being multiplied by every term of the other.

Calculating differentials and integrals of polynomials is particularly simple, compared to other kinds of functions. The differential of the polynomial:

$$P = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

with respect to x is the polynomial :

$$na_n x^{n-1} + (n-1)a_{n-1}x^{n-2} + \cdots + 2a_2 x + a_1 = \sum_{i=1}^{n} i a_i x^{i-1}.$$

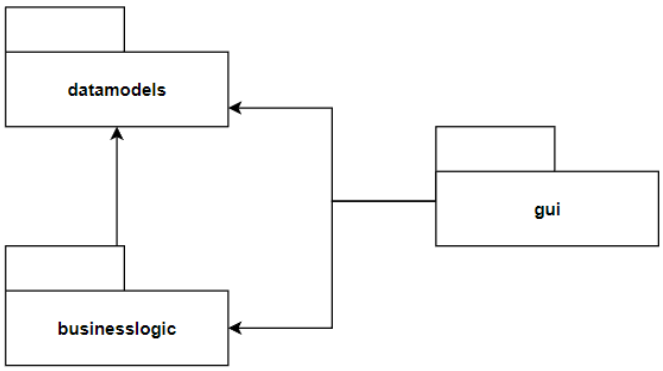
3.2. Package diagram

In object-oriented programming (OOP) development, model-view-controller (MVC) is the name of a methodology for successfully and efficiently relating the user interface to the underlying data models. This proposes three main areas to be used in the development of the application:

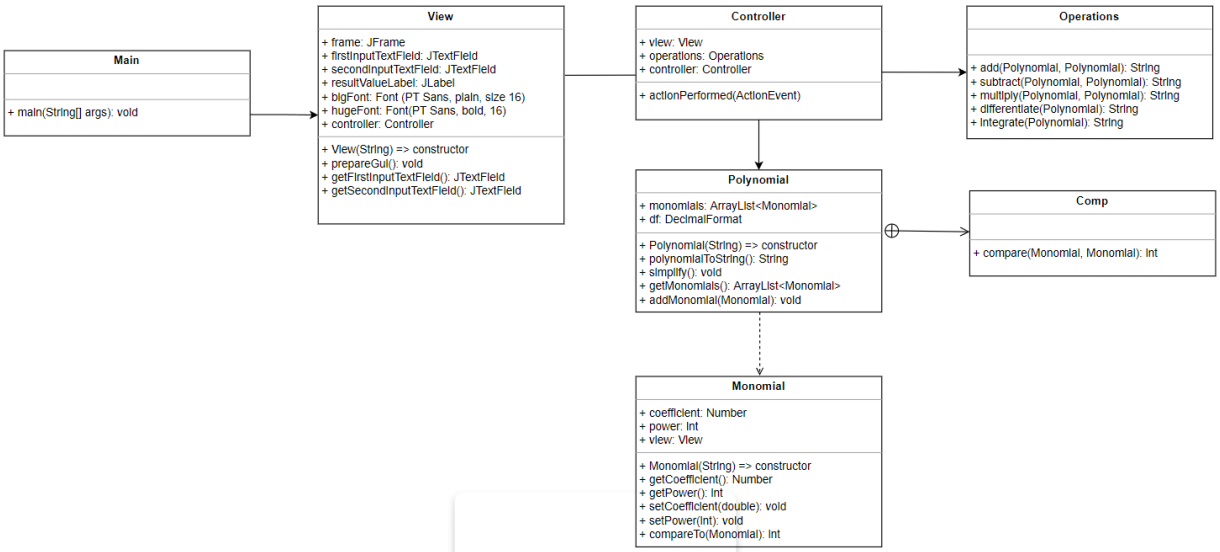
- Model: represents the logical structure of the data used by the software application.
- View: represents a collection of elements used in the user interface (everything that the user can see and interact with; for example, buttons, text boxes, labels, scrollers etc.).
- Controller: represents the connection between the model and the view.

This polynomial calculator is designed based on this MVC methodology, having the following packages:

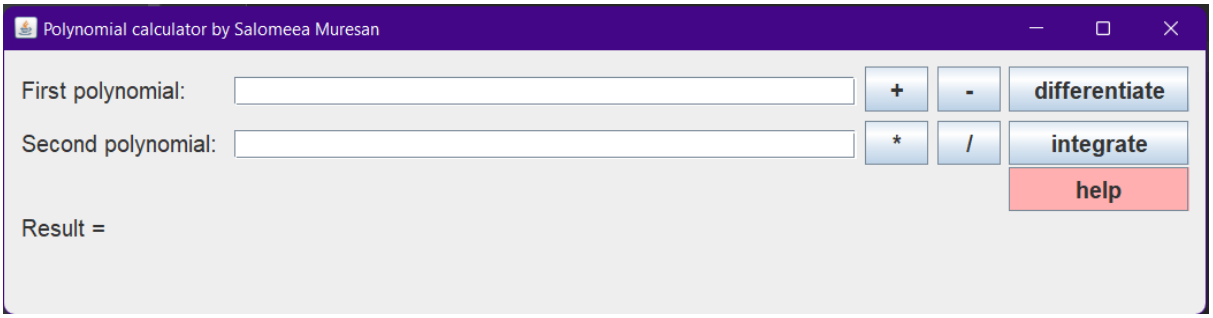
- gui (Graphical User Interface): contains 2 classes, View and Controller
- datamodels: contains 2 classes, Polynomial and Monomial
- businesslogic: contains 1 class, Operations, which implements the addition, subtraction, multiplication, differentiation and integration of the inputs



3.3. Class diagram



3.4. Design



4. Implementation

In this part of the documentation, each class will be discussed and have its functionalities presented.

4.1. View

View is responsible for arranging the window/s that will be seen by the user, essentially designing the layout of the application. In order to create a user-friendly interface, I used GroupLayout. GroupLayout works with the horizontal and vertical layouts separately. The layout is defined for each dimension independently. You do not need to worry about the *vertical* dimension when defining the *horizontal* layout, and vice versa, as the layout along each axis is totally independent of the layout along the other axis.

When focusing on just one dimension, you only must solve half the problem at one time. This is easier than handling both dimensions at once. This means, of course, that each component needs to be defined twice in the layout. If you forget to do this, GroupLayout will generate an exception. GroupLayout uses two types of arrangements: sequential and parallel, combined with hierarchical composition.

1. With **sequential** arrangement, the components are simply placed one after another, just like BoxLayout or FlowLayout would do along one axis. The position of each component is defined as being relative to the preceding component.

```
layout.setHorizontalGroup(layout.createSequentialGroup()
      .addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
            .addComponent(firstInputLabel)
            .addComponent(secondInputLabel)
            .addComponent(freeSpaceLabel)
            .addComponent(resultLabel))
.addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
            .addComponent(firstInputTextField)
            .addComponent(secondInputTextField)
            .addComponent(freeSpaceLabel)
            .addComponent(resultValueLabel))
.addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
            .addComponent(addButton)
            .addComponent(multiplyButton)
            .addComponent(freeSpaceLabel))
.addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
            .addComponent(subtractButton)
            .addComponent(divideButton)
            .addComponent(freeSpaceLabel))
.addGroup(layout.createParallelGroup(GroupLayout.Alignment.LEADING)
            .addComponent(differentiateButton)
            .addComponent(integrateButton)
            .addComponent(helpButton)));
```

2. The second way places the components in **parallel**—on top of each other in the same space. They can be baseline-, top-, or bottom-aligned along the vertical axis. Along the horizontal axis, they can be left-, right-, or center-aligned if the components are not all the same size.

```
layout.setVerticalGroup(layout.createSequentialGroup()
      .addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
            .addComponent(firstInputLabel)
            .addComponent(firstInputTextField)
            .addComponent(addButton)
            .addComponent(subtractButton)
            .addComponent(differentiateButton))
.addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
            .addComponent(secondInputLabel)
            .addComponent(secondInputTextField)
            .addComponent(multiplyButton)
            .addComponent(divideButton)
            .addComponent(integrateButton))
.addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
            .addComponent(freeSpaceLabel)
            .addComponent(freeSpaceLabel)
            .addComponent(freeSpaceLabel)
            .addComponent(freeSpaceLabel))
            .addComponent(helpButton)
.addGroup(layout.createParallelGroup(GroupLayout.Alignment.BASELINE)
            .addComponent(resultLabel)
            .addComponent(resultValueLabel)));
```

Each button has set an action command, so the controller will be able to complete the tasks it needs to do:

```
addButton.setActionCommand("ADD");
subtractButton.setActionCommand("SUBTRACT");
multiplyButton.setActionCommand("MULTIPLY");
divideButton.setActionCommand("DIVIDE");
differentiateButton.setActionCommand("DIFFERENTIATE");
integrateButton.setActionCommand("INTEGRATE");
helpButton.setActionCommand("HELP");

addButton.addActionListener(this.controller);
subtractButton.addActionListener(this.controller);
multiplyButton.addActionListener(this.controller);
divideButton.addActionListener(this.controller);
differentiateButton.addActionListener(this.controller);
integrateButton.addActionListener(this.controller);
helpButton.addActionListener(this.controller);
```

All these instructions are written in the method prepareGui().

- prepareGui () method is the one in charge with the display (positioning the buttons, labels, text boxes)
- View (String) method is a constructor that initializes the frame of the GUI and also calls method prepareGui ().

4.2. Controller

As it is implied from its name, the Controller controls the software application. Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the DataModel component and interact with the Views to render the final output. For example, the user will handle all the interactions and inputs from the View and perform operations on one or two polynomials.

Controller gets called by the view when one of the buttons is clicked.

```
public void actionPerformed(ActionEvent e)
```

- view attribute holds the refference to the caller view
- operations attribute is an object from the class Operations

4.3. Monomial

This class represents a monomial. A monomial is, roughly speaking, a polynomial which has only one term. Knowing this definition, the atributes of the class are:

- coefficient, declared as an object of the abstract class Number in the java.lang.package. The main purpose of Number class is to provide methods to convert the numeric value in question to various primitive types (int, double are the primitive types used in this project);
- power, which holds the (integer) value of the monomial.
- Monomial(String): constructor

```
public Monomial(String monomial){
    String COEFFICIENT_PATTERN = "([+-][0-9]+)";
    String POWER_PATTERN = "[\\^]([0-9]+)";

    Pattern coefficientPattern = Pattern.compile(COEFFICIENT_PATTERN);
    Pattern powerPattern = Pattern.compile(POWER_PATTERN);

    Matcher coefficientMatcher = coefficientPattern.matcher(monomial);
    Matcher powerMatcher = powerPattern.matcher(monomial);

    if(coefficientMatcher.find() && powerMatcher.find()){
        this.coefficient=Integer.parseInt(coefficientMatcher.group(1));
        this.power=Integer.parseInt(powerMatcher.group(1));
        System.out.println("good");
    }
}
```

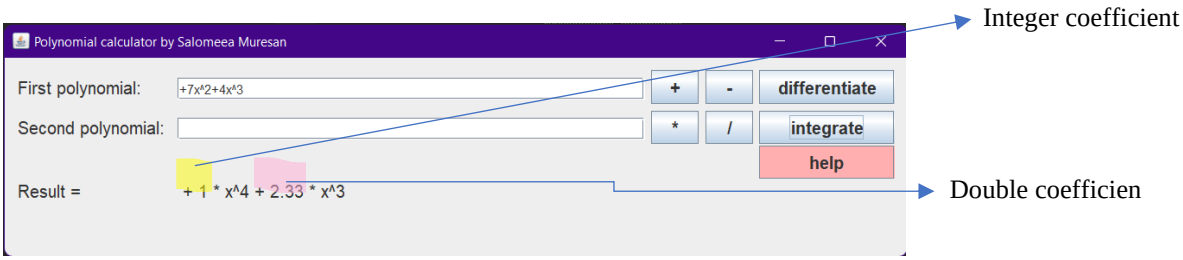
-

A monomial, taking in consideration the Monomial class, can be written as: (coefficient)*$x^{(power)}$.

4.4. Polynomial

This class represents a polynomial as a collection of monomials.

- monomials: a list (ArrayList) of monomials that put together results the polynomial in question
- df (abbreviation from DecimalFormat), which is set in such a way that all double values coefficients will be printed with only 2 decimals.



- polynomialToString () method creates a small text, converting a Polynomial class object into something that the user can understand and work with. This method is used for displaying the correct result after all the calculations this polynomial calculator made.
- simplify () method is useful mostly when the polynomial in question has more terms, but some of them have matching powers. This method goes through the whole list of monomials and if it finds two terms with the same power, it adds the coefficients and removes one of the monomials from the list (the monomial that hasn't changed its coefficient).
- getMonomials (returns the ArrayList of monomials)
- addMonomial (Monomial) method adds a new monomial to the already implemented ArrayList
- Polynomial (String): constructor

```
public Polynomial(String polynomial) throws ParseException {
    this.monomials= new ArrayList<>();
    Monomial monomial;
    String POLYNOMIAL_PATTERN = "([+-][^+-]+)";
    Pattern pattern = Pattern.compile(POLYNOMIAL_PATTERN);
    Matcher matcher = pattern.matcher(polynomial);
    while(matcher.find()){
```

```
            monomial=new Monomial(matcher.group(1));
            this.addMonomial(monomial);
        }
    }
```

4.5. Comp

This class is used only as comparator between two monomials, taking in consideration their power.

```
public class Comp implements Comparator<Monomial> {
    @Override
    public int compare(Monomial o1, Monomial o2) {
        return o1.compareTo(o2);
    }
}
```

This must be done in order to be able to sort the list of monomials with only one single line:

```
Collections.sort(this.monomials,new Comp());
```

4.6. Operations

This class implements each of the 5 operations that can be done on 2 polynomials.

ADDITION:

```
public static String add(Polynomial first, Polynomial second) throws ParseException {

    for(Monomial m1 : first.getMonomials()){
        boolean thereIs = false;
        for (Monomial m2: second.getMonomials()) {
            if(m1.getPower()==m2.getPower()){
                thereIs = true;
                m2.setCoefficient(m2.getCoefficient().intValue()+m1.getCoefficient().intValue());
            }
        }
        if(!thereIs){
            second.addMonomial(m1);
        }
    }
    second.simplify();
    return second.polynomialToString();
}
```

This code goes through both polynomials, finds the monomials with the same power and then performs the addition in m1. If, when going through the second polynomial list of monomials, the monomial will be added to the first polynomial, keeping the sign of the coefficient unchanged.

SUBTRACTION:

```
public String subtract(Polynomial first, Polynomial second) throws ParseException {
    for(Monomial m2 : second.getMonomials()){
        m2.setCoefficient((-1)*m2.getCoefficient().doubleValue());
    }
    return add(first, second);
}
```

The subtraction operation is based on addition, but this time, we must change the signs of the coefficients of each monomial, so if the coefficient is +5, it will be transformed in -5 and vice versa.

MULTIPLICATION

```
public String multiply(Polynomial first, Polynomial second) throws ParseException {
    Polynomial result = new Polynomial("");
    for(Monomial m1 : first.getMonomials()){
        for (Monomial m2: second.getMonomials()) {
            Monomial mResult = new Monomial("");
            mResult.setPower((m1.getPower()+m2.getPower()));
            mResult.setCoefficient(m1.getCoefficient().intValue()*m2.getCoefficient().intValue());
            result.getMonomials().add(mResult);
        }
    }
    result.simplify();
    return result.polynomialToString();
}
```

This method multiplies the *first monomial from* the first polynomial with all the other monomials from the second polynomial, and *then the second*, and so on. After all monomials have been multiplied, the result will be simplified (using the method. simplify()).

DIFFERENTIATE

```
public String differentiate (Polynomial polynomial) throws ParseException {
    for(Monomial m : polynomial.getMonomials()){
        m.setCoefficient(m.getCoefficient().intValue()*m.getPower());
        m.setPower((m.getPower()-1));
    }
    polynomial.simplify();
    return polynomial.polynomialToString();
}
```

The differentiation method will be applied only to the first polynomial. Considering the input coefficients and powers integers, the differentiation will be easy to compute:

We know that $(x^a)` = a*x^{a-1}$, so we apply the same formula to each monomial of the input polynomial.

INTEGRATION

```
public String integrate (Polynomial polynomial) throws ParseException {
    for(Monomial m : polynomial.getMonomials()){
        m.setCoefficient((m.getCoefficient().doubleValue()*(1.0/(m.getPower()+1))));
        m.setPower((m.getPower()+1));
    }
    polynomial.simplify();
    return polynomial.polynomialToString();
}
```

The integration method will be applied only to the first polynomial. Considering the input coefficients and powers integers, the integration will be easy to compute:

We know that $\int x^a dx = \frac{x^{a+1}}{a+1}$, so we apply the same formula to each monomial of the input polynomial.

5. Conclusions

I have learned new and useful things during the completion of this assignment: I learned about the MVC design pattern and how easy it can actually be to create a functionally GUI using this pattern. I really wish I knew this before, when I was working on other projects. I also quickly gained an understanding of Java Swing, in spite of the fact that I have never used this toolkit before. It was backbreaking at first to create the layout, but after a few days of trying, I finally figured it out.

Despite the polynomial calculator is functional, it has its own shortcomings. Overcoming these, the program should:

- cover extensively all the patterns for the inputs, so there wouldn't be so many restrictions for the user when giving the inputs.
- implement the division of the 2 polynomials.
- cover exception cases (ex. When the input text is not valid etc.).
- create a more friendly user interface.

6. Bibliography

https://ssaurel.medium.com/learn-to-make-a-mvc-application-with-swing-and-java-8-3cd24cf7cb10

https://dsrl.eu/courses/pt/

https://en.wikipedia.org/wiki/Polynomial

https://www.javatpoint.com/uml-class-diagram

https://docs.oracle.com/javase/tutorial/uiswing/