FUNDAMENTAL PROGRAMMING TECHNIQUES

ASSIGNMENT 1
SOLUTION DESCRIPTION DOCUMENT

POLYNOMIAL CALCULATOR

Student: Mihai Cristina-Mădălina
Teaching assistant: Chifu Viorica

Faculty of Automation and Computer Science
Computer Science Department (En)
2nd year of study, gr. 30424

Content:

1. Objective

- **The main objective** of this laboratory work is to **design and implement a polynomial calculator** with a **dedicated graphical interface** through which the user can enter polynomials, select the operation to be performed (i.e. addition, subtraction, multiplication, division, differentiation, integration) and display the result.
- The project will implemented as a **Maven Project** using **Eclipse application**.

Secondary objective:

- Splitting the project into packages (3.1.);
- Designing Graphical User Interface according to the Model View Controller (MVC) architectural pattern (3.2.);
- Implementing Monomial and Polynomial classes (3.3.);
- Implementing operations to be performed on polynomials (4.1.);
- Use Regular Expressions and pattern matching to check the validity of the user input and extract the polynomial coefficients (4.1.).

2. Problem analysis

2.1. Overview

A polynomial is an **expression** that can be built from **constants and symbols** called indeterminates or variables by means of addition, multiplication and exponentiation to a **non-negative integer power**. Two such expressions that may be transformed, one to the other, by applying the usual properties of **commutativity**, **associativity** and **distributivity** of addition and multiplication are considered as defining the same polynomial.

A polynomial in a single indeterminate x can always be written (or rewritten) in the form

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

where $a_0, \ldots, a_n$ are constants and ***x*** is the indeterminate. The word "indeterminate" means that ***x*** represents no particular value, although any value may be substituted for it. [1]

The calculator will find the result of the sum, difference, multiplication and division of two polynomials of any rank. It will also be able to compute the differentiation and integration of a polynomial.

2.2. Modeling the problem

The **functioning** of the polynomial calculator is shown in Fig. 1 and explained below:

Step 1: The calculator takes as input a polynomial of form: $a_n x\hat{}\, n + a_{n-1} x\hat{}\,(n-1) + \ldots + a_1 x + a_0$ and the operation selected by the user which needs to be performed (the order of monomials is not important).

Step 2: The input is processed and checked for its validity. The program checks if any operation was selected as well. If the input is incorrect or the operation was not selected, an error message is displayed to the user. Otherwise, the program goes to the step 3.

Step 3: As long as there is a valid input and an operation is required, the calculator will make the necessary computations and will display the result.

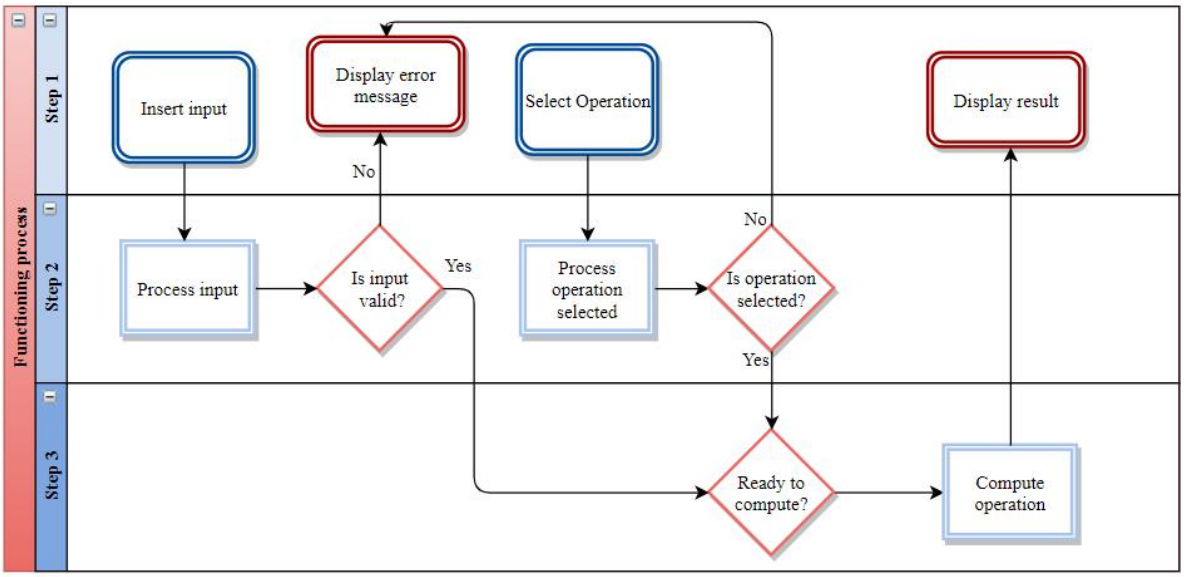


Fig. 1 Functioning process

2.3. Use-case analysis

The application shown in Fig. 2 is used as follows:
1. Run the application.
2. Insert the first polynomial. Is the input valid? If Yes, go to step 4. If Not, go to step 3.
3. Display error message and go to step 2.
4. Select operation. Does the operation requires a second polynomial? If Yes, go to step 5. If Not, go to step 6.
5. Insert 2nd polynomial. Is the input valid? If Yes, go to step 7. If Not, go to step 3.
6. Compute operation on one polynomial (differentiation or integration) and go to step 8.
7. Compute operation on 2 polynomials (addition, subtraction, multiplication, division).
8. Display result.
9. Stop.

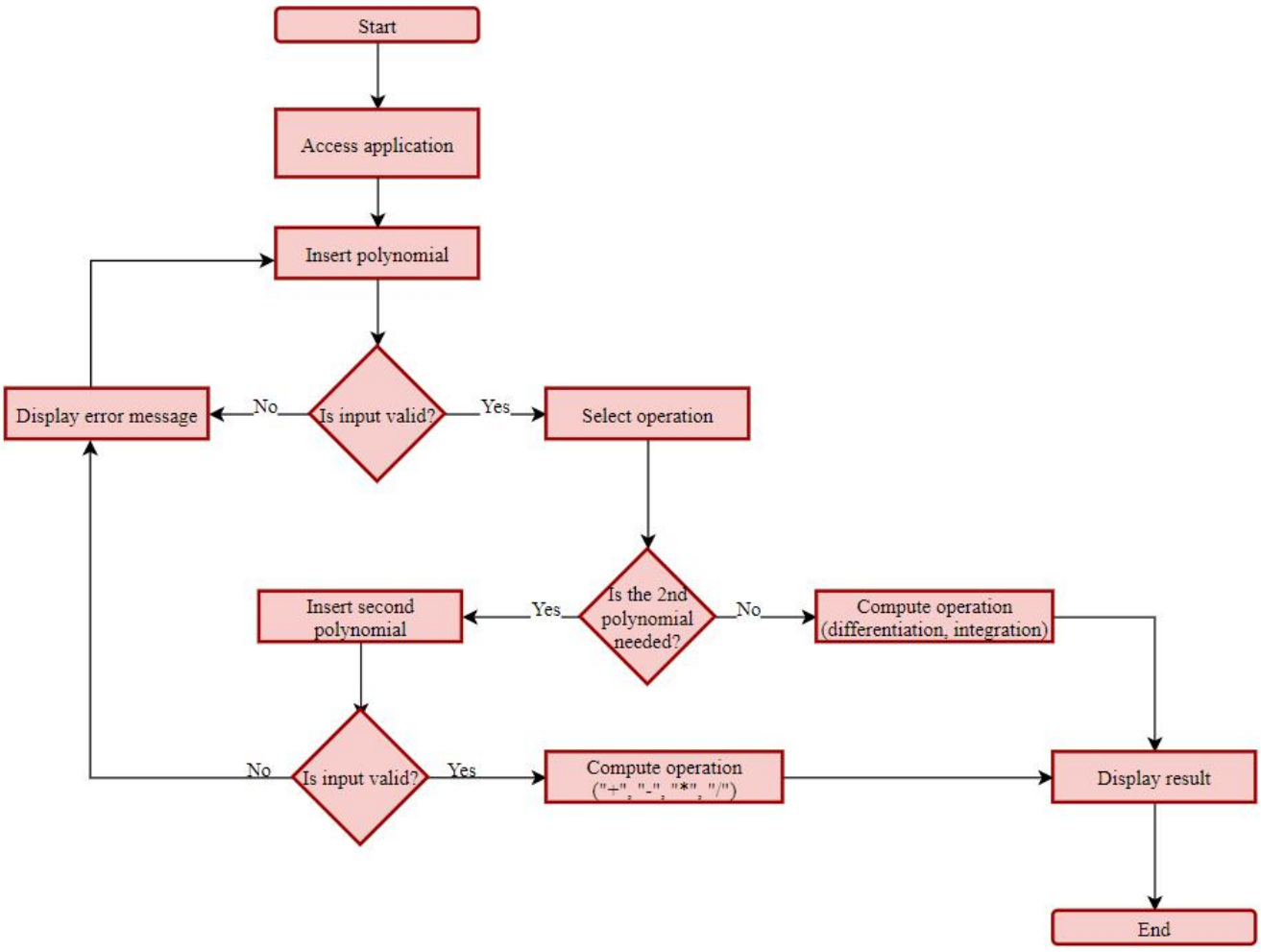


Fig. 2 Use-case flowchart

3. Object-oriented approach

3.1. Strategy

This assignment satisfies the object-oriented programming implementation by using encapsulation and working mainly with the implemented Monomial and Polynomial classes. Furthermore, it has been used the Model-View-Controller architectural pattern for designing the calculator.

Therefore, the project is divided in 4 big packages: controller, main, model, view. This approach was used in order to separate the user interface into a View (which creates the display and call the Model when necessary to get information) and Controller (which responds to user requests, interacting with both the View and Controller when necessary). (see Fig. 3)

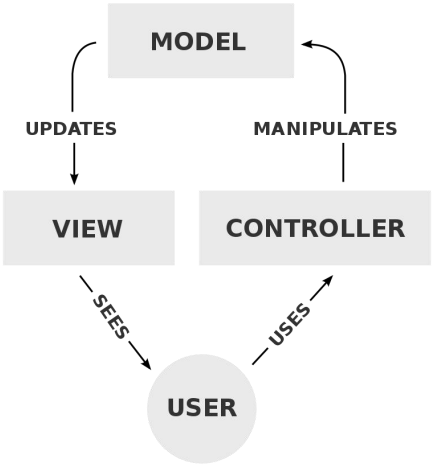


Fig. 3

In our case, in the "main" package, model, view and controller are created and passed to the parts that need them, so there is only one copy of each. (see Fig. 4)

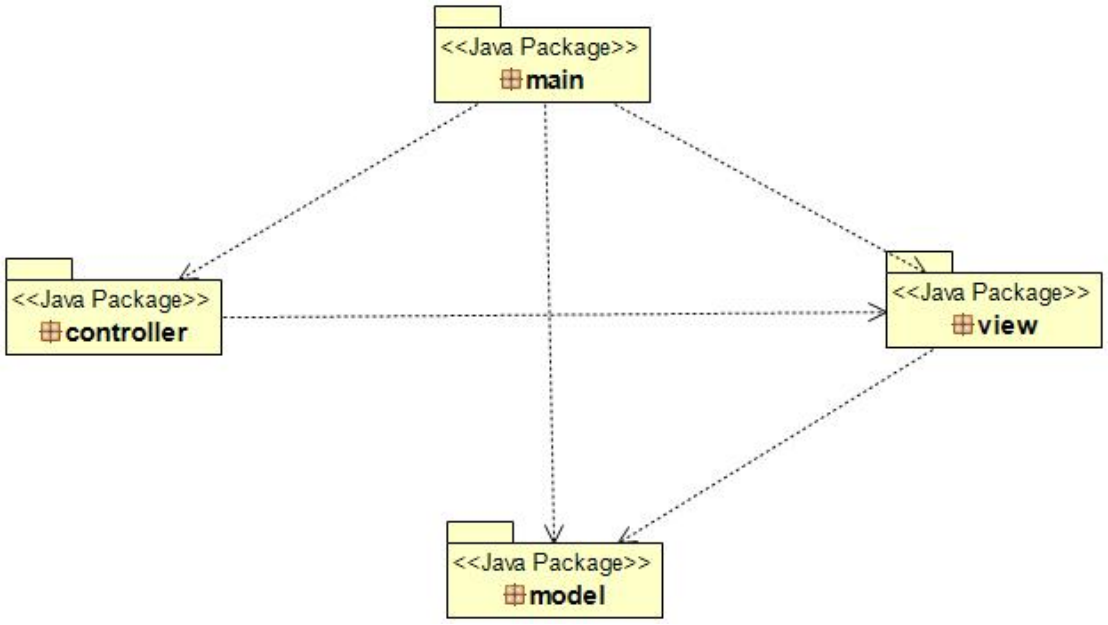


Fig. 4 UML diagram of the project packages

3.2. UML diagrams

In the following section, there will be presented the UML diagrams of each package.

As shown in Fig. 5, the CalculatorModel class is dependent on the Polynomial class, which is related to Monomial. The Model is independent on the user interface and its purpose is to call the operation which needs to be computed and store the result which will be returned when required, due to the encapsulation usefulness.
The Polynomial class includes the methods which compute the operations needed and other additional methods (i.e. getRank(), toString(), transformInStandardForm()) which will be described in the *Class design* section (3.3.). The Monomial class is used just to store and return the coefficients and powers of the polynomials.

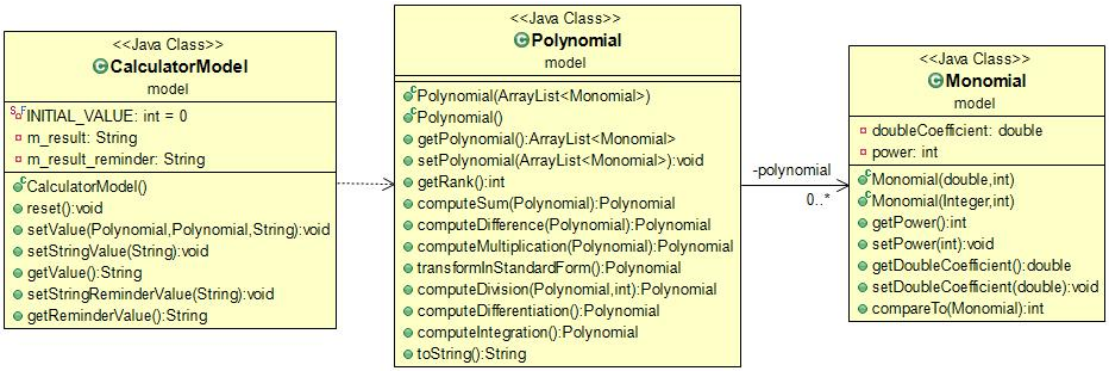


Fig. 5 UML diagram of model package

Fig. 6 shows the implementation of the user interface. It includes text fields, buttons and a combo box and action listeners for buttons as well. Its purpose is to send to the model the requirements of the user and display the result set back by the model.

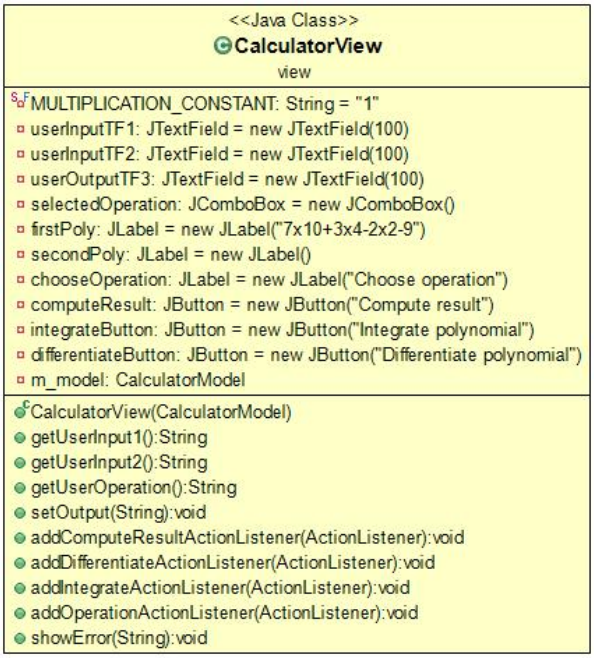


Fig. 5 UML diagram of view package

The controller process the user requests. It is implemented as an observer because it registers listeners that are called when the view detects a user interaction. Based on the user request, the controller calls the methods in the view and model to accomplish the requested action. In this case, the controller has 3 integrated listeners, one for each button of the interface. Moreover, the controller is responsible for checking the validity of the user input string through the method *matchRegEx(String):Polynomial* which will be discussed in section 4.1. (i.e. polynomial). (see Fig. 6)

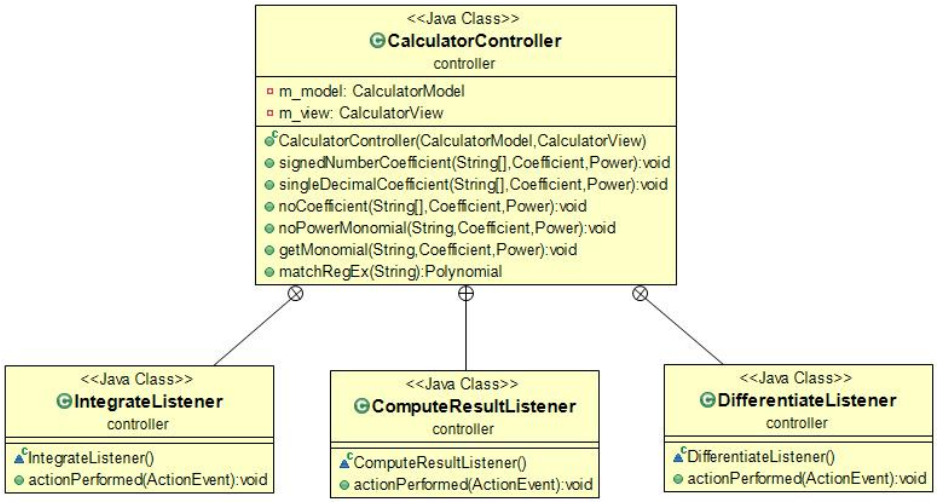


Fig. 6 UML diagram of controller package

In Fig. 7, the whole project is presented. It can be easily noticed that the packages are strongly related to each other.

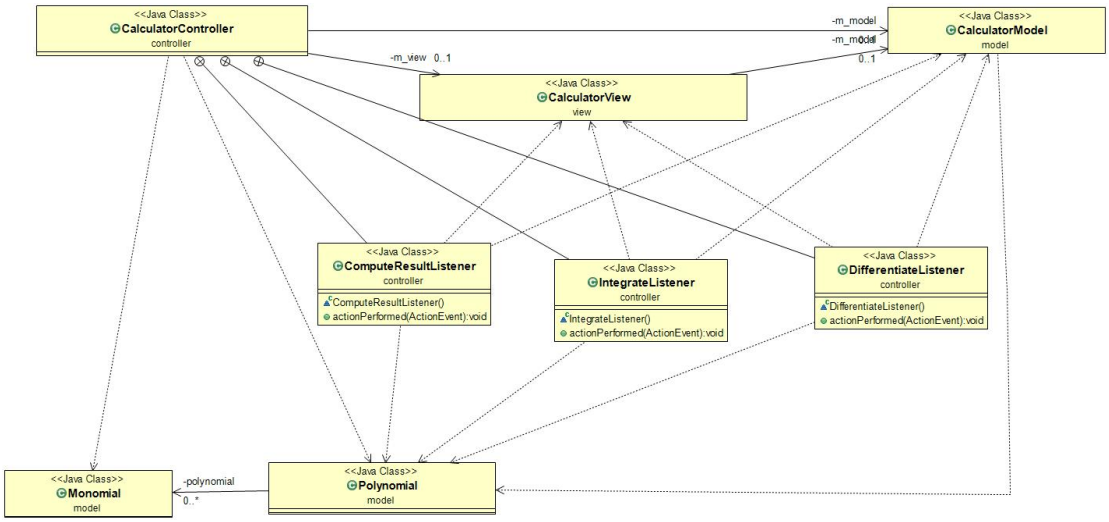
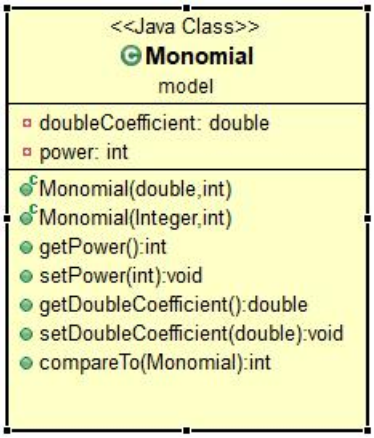


Fig. 7 UML diagram of all packages

3.3. Class design

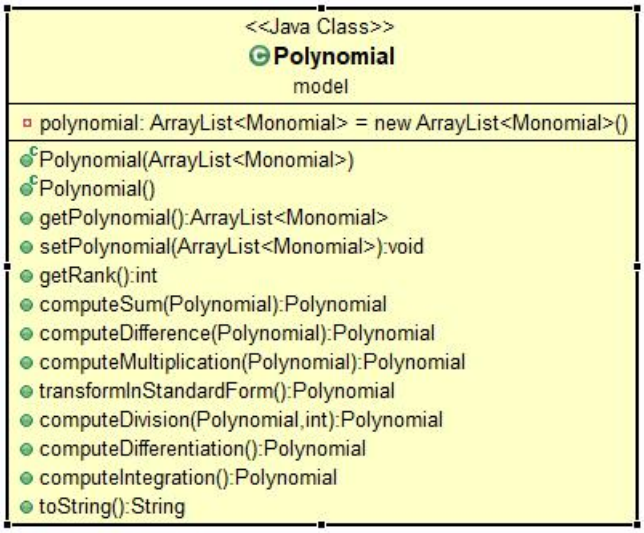
According to the design considerations, two main classes has been implemented, i.e. Monomial and Polynomial.

The Monomial class has been designed such way that it uses just the variables "coefficient" as double and "power" as integer. Due to the Regular Expression and pattern matching, the coefficients of the received polynomial from the input will be accepted just if they are integers. So the problem request is fulfilled and the implementation gets easier. It also implements the Comparable interface which will help us order the monomials in descending order of powers.



The Polynomial class is composed of an array list of monomials. The class includes the crucial methods (i.e. computeSum, computeDifference etc.) and some auxiliary methods used to compute the crucial ones easier.

- Method *getRank()* returns the rank of the polynomial.
- Method *transformInStandardForm()* fulfills the polynomial with the missing elements where the coefficient is 0 (if any). For example, if the given input polynomial is: 7x^3+1, then this method transforms it into: 7x^3+0x^2+0x^1+1x^0. So the computations gets easier.
- Finally, the method *toString()* transforms the polynomial into a user friendly polynomial string.



3.4. User interface design (UI)

For the UI, there has been declared basic elements to design a friendly, easy-to-access interface:

- 1 field to enter the user 1st polynomial, predefined with a random polynomial example;
- 1 field to insert the 2nd polynomial if necessary;
- 1 button for computing differentiation of the first polynomial;
- 1 button for computing integration of the first polynomial;
- 1 combo box for selecting one of the remaining operations: Addition, Subtraction, Multiplication, Division;
- 1 button for computing the result after selecting an operation and inserting 2 polynomials.
- 1 non-editable field for displaying the result for the user. (see Fig. 8)

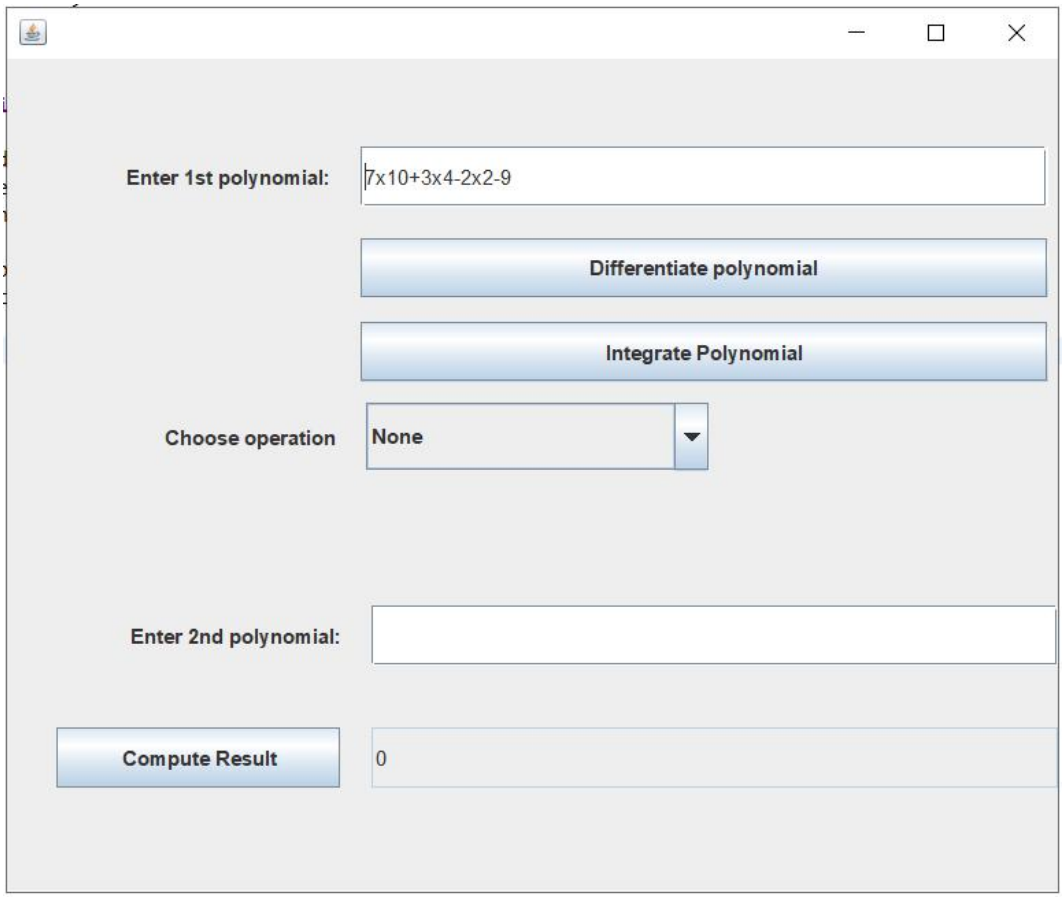


Fig. 8 User Interface

All the above listed elements have been arranged on the frame using Layout Managers.

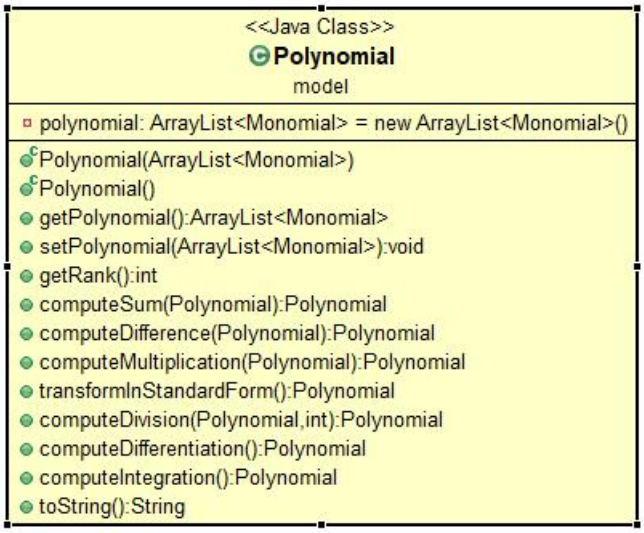
A layout manager is an object that implements the LayoutManager interface and determines the size and position of the components within a container. Although components can provide size and alignment hints, a container's layout manager has the final say on the size and position of the components within the container.[2]

Among the many possibilities, the GroupLayout seemed to fit best for the assignment at hand. GroupLayout is a layout manager that was developed for GUI builders such as Matisse, the GUI builder provided with the NetBeans IDE. Although the layout manager was originally designed to suit the GUI builder needs, it also works well for manual coding.

4. Implementation

4.1. Classes and methods description

The most striking class is the Polynomial one. It includes the methods for computing all the crucial operations. Taking this into consideration, a description of the computing approach is described below.



We have the following methods, each one provided with a short description of the implementation and exemplification:

- *computeSum(Polynomial):Polynomial*

Description: A new polynomial "sum" is created. It is given the a set of monomials with coefficient 0 and maximum power equal to the maximum rank between the polynomials given as input. To this polynomial, there are added sequentially polynomial number 1 and polynomial number 2 coefficient by coefficient. The method returns the "sum".

E.g.:

Polynomial 1	+x^**7**+2x^5+9x^3
Polynomial 2	+2x^**7**+x^4-x^2
sum	+0.0x^**7**+0.0x^6+0.0x^5+0.0x^4+0.0x^3+0.0x^2+0.0x^1+0.0x^0
sum + Polynomial 1	+**1**.0x^7+0.0x^6+**2**.0x^5+0.0x^4+**9**.0x^3+0.0x^2+0.0x^1+0.0x^0
sum + Polynomial 2	+**3**.0x^7+0.0x^6+**2**.0x^5+**1**.0x^4+**9**.0x^3-**1**.0x^2+0.0x^1+0.0x^0
sum (displayed)	3.0x^7+2.0x^5+x^4+9.0x^3-x^2

- *computeDifference(Polynomial):Polynomial*

Description: A new polynomial "difference" is created. It is given the a set of monomials with coefficient 0 and maximum power equal to the maximum rank between the polynomials given as input. To this polynomial, there is added polynomial number 1 and then subtracted polynomial number 2 coefficient by coefficient. The method returns the "difference".

E.g.:

Polynomial 1	+x^**7**+2x^5+9x^3
Polynomial 2	+2x^7+x^4-x^2
difference	+0.0x^**7**+0.0x^6+0.0x^5+0.0x^4+0.0x^3+0.0x^2+0.0x^1+0.0x^0
difference + Polynomial 1	+**1**.0x^7+0.0x^6+**2**.0x^5+0.0x^4+**9**.0x^3+0.0x^2+0.0x^1+0.0x^0
difference - Polynomial 2	-**2**.0x^7+0.0x^6+**2**.0x^5-**1**.0x^4+**9**.0x^3+**1**.0x^2+0.0x^1+0.0x^0
difference (displayed)	-2.0x^7+2.0x^5-x^4+9.0x^3+x^2

- *computeMultiplication(Polynomial):Polynomial*

Description: A new polynomial "multiplication" is created. It is given the a set of monomials with coefficient 0 and maximum power equal to the sum of the ranks of the polynomials given as input. We iterate through each polynomial and, for each pair of monomials, add their power and multiply their coefficient. We set the coefficient obtained to the multiplication polynomial.

E.g.:

Polynomial 1	+x^**2**+3
Polynomial 2	+2x^**3**+x^2
multiplication	+0.0x^**5**+0.0x^4+0.0x^3+0.0x^2+0.0x^1+0.0x^0
Polynomial 1 * Polynomial 2	+2.0x^5+1.0x^4+6.0x^3+3x^2
multiplication	+**2.0**x^5+**1.0**x^4+**6.0**x^3+**3.0**x^2+0.0x^1+0.0x^0
multiplication (displayed)	+2.0x^5+x^4+6.0x^3+3.0x^2

- *computeMultiplication(Polynomial, int):Polynomial*

Description: As long as the rank of the first polynomial is greater then the rank of the second one, we compute the coefficient and power of each monomial of the result (stored in a new Polynomial variable "divisionMonomial"). The 1st polynomial is then multiplied by this monomial and subtracting from its own old value. Also, its element with maximum rank is removed. According to integer parameter given, if 0, the method will return the quotient and the remainder, otherwise.

E.g.:

Polynomial 1	+2x^**3**+x^2
Polynomial 2	+x^**2**+3.0x^0
divisionMonomial	+2.0x^1
Polynomial 1 * divisionMonomial	+2.0x^3+6.0x^1
Polynomial 1 -= Polynomial 1 * divisionMonomial	+1.0x^2-6.0x^1
divisionMonomial	+1.0x^0
Polynomial 1 * divisionMonomial	+1.0x^2+3.0x^0
Polynomial 1 -= Polynomial 1 * divisionMonomial	-6.0x^1-3.0x^0
Quotient (divisionMonomial)	+2.0x^1+1.0x^0
Remainder (Polynomial 1)	-6.0x^1-3.0x^0

- *computeDifferentiation():Polynomial*

Description: For each monomial of the polynomial, the coefficient is changed to the value of the current coefficient times the value of the power. The power is decremented with 1.

E.g.:

Polynomial 1	+x^**2**+3
Polynomial 1	+2.0x^1+0.0x^0
Polynomial 1 (displayed)	+2.0x

- *computeIngetration():Polynomial*

Description: For each monomial of the polynomial, the coefficient is changed to the value of the current coefficient devided by the value of the (power + 1). The power is incremented with 1.

E.g.:

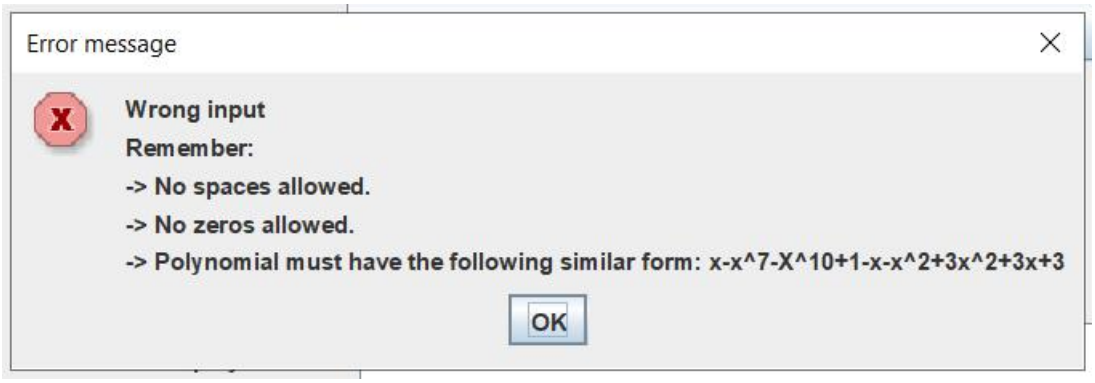
Polynomial 1	+x^**1**+3
Polynomial 1	+0.5x^2+3.0x^1
Polynomial 1 (displayed)	+0.5x^2+3.0x

Another important method is *matchRegEx(String):Polynomial*, which uses Regular Expressions[3]. It takes as input a String and returns a Polynomial. This method has two roles: one is to check the validity of the user input using patterns and matchers, and the other one is to split the input into monomials and extract the coefficient and power of each one. For checking the validity, two patterns were needed: one for checking the correctness of the input and one for finding errors in the input string. For extracting the monomials characteristics, more methods have been implemented due to the various way of writing a polynomial with coefficient 0 or 1 and power 0 or 1. The table below shows for which pattern the method was used:

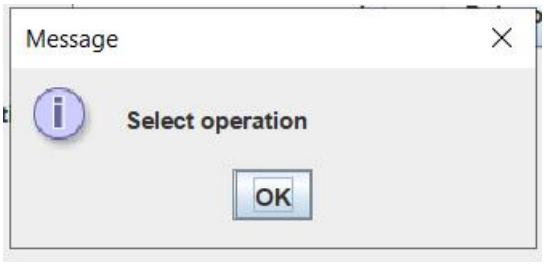
Method implemented	**Pattern target**
noPowerMonomial(String, Coefficient, Power)	x, -x, +x, ax (a is an integer)
noCoefficient(String[], Coefficient, Power)	(+-)x^b (b is an integer)
singleDecimalCoefficient(String[], Coefficient, Power)	-x^b, +x^b, ax^b (a is a decimal, b is an integer)
signedNumberCoefficient(String[], Coefficient, Power)	ax^b (a, b are integer)

4.2. User Interface description

The user interface is easy to work with. It just require the user to enter the desired polynomial(s) and select an operation and/or press buttons for requested result. There are some restrictions though. Only polynomials with non-zero coefficients are allowed. Each monomial follows the pattern “ax^b”, where a may be a signed integer or could be missing and b is a natural number and could be missing as well. More than that, no spaces are allowed. If any of this restrictions is violated, then an error message is displayed on the screen.

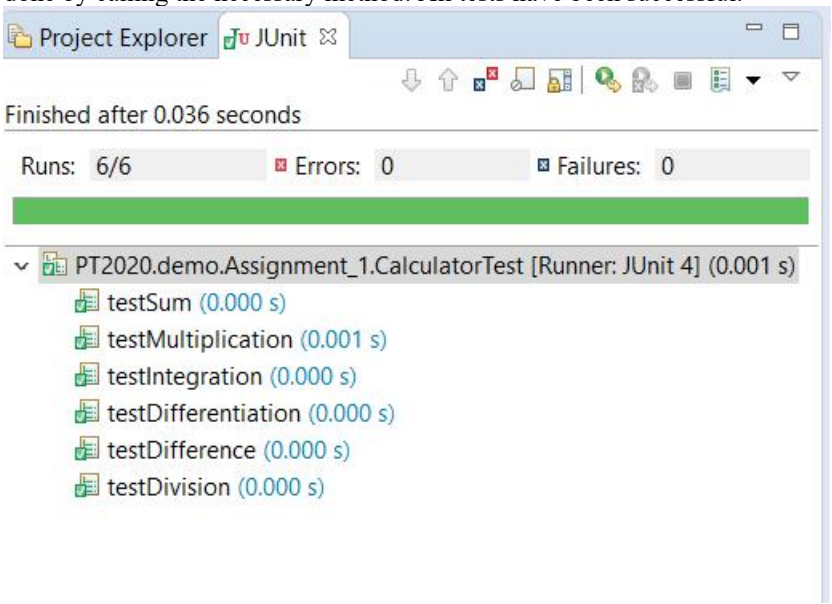


Some other error frames are displayed as well in case the user does not select an operation.



5. Unit Testing

Testing the calculator has been done using **JUnit 4[4].** In class *CalculatorTest* , have been tested methods for computing operations: *testSum(), testDifference(), testMultiplication(), testDivision(), testDifferentiation(), testIntegration().* In each of this methods, there have been created one or two polynomial(s) and a String according to the desired result. Then method *assertEquals(String, String)* has been called to check the coincidence between our desired result declared in the string and the actual computation done by calling the necessary method. All tests have been successful.



6. Conclusions

6.1. Lessons learned

While working on this assignment, I have found myself fascinated by the Java Style conventions [5] at first sight. Then, progressing into the project, I have learned:

- ✓ how to use arrays properly;
- ✓ how to iterate through arrays in a more professional way;
- ✓ how to write redundant code into reusable sub-methods;
- ✓ how to work with packages;
- ✓ how to cope with the Model View Controller pattern;
- ✓ how to use Regular Expressions patterns and matchers;
- ✓ how to use JUnit for testing my methods;

6.2. Possible future improvements

Due to the simplicity of the project, the monomial characteristics and the user interface flexibility, there can be added many features such as:

- ✓ taking coefficients as floating point numbers;
- ✓ give value to the indeterminate **x** and compute the result;
- ✓ finding the roots of the polynomial of rank 2 or 3;
- ✓ designing more fields for inserting the polynomials;
- ✓ showing the step which the input takes to get to the output;
- ✓ computing a graph for the polynomial at hand;
- ✓ correct automatically the input if some basic errors occur (i.e. if an non-desired character is introduced).

7. Bibliography

- [1] Polymial:
  https://en.wikipedia.org/wiki/Polynomial
- [2] Layout Manager:
  https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html
  https://docs.oracle.com/javase/tutorial/uiswing/layout/using.html
- [3] Regular Expressions Pattern:
  https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html
- [4] JUnit:
  http://users.utcluj.ro/~igiosan/Resources/POO/Lab/12-Testarea_Unitara.pdf
- [5] Java Style:
  https://google.github.io/styleguide/javaguide.html