Polynomial Calculator

Programming Techniques – First Project

Group: 30424

Student: Mureșan Alexandru

Teacher: Claudia Pop

Contents

1. Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

2. Description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

3. Diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

3. 1 Use-case diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

3. 2 Class diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

4. Implementation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

5. Implementation testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

6.Further developments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

7.Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

**1.Introduction**

Polynomials are of great importance in Mathematics and in Computer Science fields with a wide range of applicability.

Using Java for both the graphical interface and for background code I was able to develop a standalone application which deals with operation s applied to polynomials.

There are basic operations that can be performed on polynomials, as addition, subtraction and multiplication (i.e. which use two polynomials) but there are also more difficult operations such as differentiation and integration (which require only one polynomial).

Given the general form of any polynomial,

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

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . .. . . . . .. . . . . .. . . . . .. . . . .. . . . . . .. . . . . . .. . . . .. . . . . .. . . .. . . . .. . . . .. . .. . . . .. . . . .. . . .. . .. . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . .. . . . . . . .. . . . . . . . .. .. .. . . . . . . . . . . . . . . . .. . . .. . . . .. . .. . . . . . .. . . . .. . . .. . . . . . . .. . . .. . . .. . . . . .. . . . .. . . . . . . . . . . . .. . . . . . . .. . . . . .. . . . . . . . . .. . . . . . . . . .. . . . . . .. . . . . . .. . . . .. . . . . .. . . .. . . . . . . . . .. . .. . .. .. .. . . . .. . . .. . . .. . . . . . . .. . . . . . . . . . . .. . . . . . .

we can see that the problem can be solved using more than one method (i.e. ArrayList of monomials, composed of coefficient and degree, or array of coefficients).

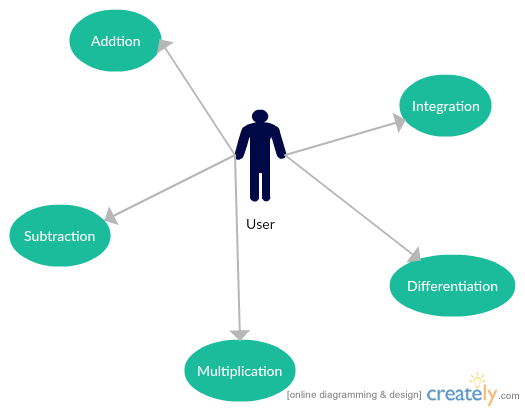
I chose the ArrayList implementation because it is more flexible in terms of describing the polynomial and it is also more object-oriented as you work with objects of a class, not character strings exclusively.

**2.Description**

The purpose of this assignment is to develop a standalone application which performs operations on polynomials. This project has all the operations implemented, without the division. We are required to work with polynomials written in textboxes, and we have to create a user-friendly interface and include the operations the user requires.

**3.Diagrams**

**3.1. Use-case diagrams**

 Use case diagrams are used to present a graphical overview of the functionality of a system. This use-case diagram is particularly showing the relation between the user and the different use-cases in which the user is involved.

The interface must be friendly, regarding the large variation of user’s experience.

Fig.1 Use-case diagram of Polynomial Calculator

**3.2 Class Diagram**

Class JFrame

Class Term

+degree: int;

+coefficient: float;

getDegree(): int;

setDegree(int): void;

getCoefficient(): float;

setCoefficient(float): void;

Class Polynomial

+terms: List<Term>;

getTerms(): List<Term>;

setTerms(List<Term>): void;

addTerm(Term): void;

polyFromString(String): void;

toString(): String;

clear(): void;

Class Operation

add(Polynomial, Polynomial):Polynomial;

sub(Polynomial, Polynomial): Polynomial;

mul(Polynomial, Polynomial): Polynomial;

differentiate(Polynomial): Polynomial;

integrate(Polynomial): Polynomial;

Class MainClass

+calc: Calculator;

Class Calculator

+panel: JPanel;

+addButton: JButton;

+subButton: JButton;

+intButton: JButton;

+ derButton: JButton;

+mulButton: JButton

+polynomial1Labe: JLabel;

+polynomial2Label: JLabel

+resultLabel: JLabel

+tbox1: JTextPane;

+tbox2: JTextPane;

+tbox3: JTextPane;

+Poly1: String;

+Poly2: String;

+p1: Polynomial;

+p2: Polynomial;

**4.Implementation**

This project is made following the OOP structure(classes, methods) and principes:

1. **Inheritance:** when one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism

2. **Polymorphism:** when one task is performed by different ways i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc.

3. **Abstraction:** hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing

4. **Encapsulation:** binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

It has the following packages:

-> **default**- the default package;

-> **polynomial.model**- the package that contains the classes used to implement the operations on polynomials and the Graphic User Interface of this project;. .

Polynomial.Model

* Calculator
* MainClass
* Operation
* Polynomial
* Term

**Term**

**public** **class** **Term**{{

**private** **int** degree;

**private** **float** coefficient;

}

In this class I define each term of a polynomial through its private coefficient and degree. I made getters and setters for each attribute so I would be able to receive or change their value whenever I want. Terms are stored in memory as monomials with coefficient\*x^ degree form.

**Polynomial**

**public** **class** **Polynomial**{{

**private** List<Term> terms;

}

In this class I define each polynomial as a list of terms, each term having the characteristics from class **Term** defined previously. Terms are unsorted, so the user can introduce whatever coefficient and degree he/she wants in whatever order the user likes. I also implemented a polyFromString(String) method for transforming the user input text in polynomial terms.

I improved the toString() method, so it would know to write the polynomials in a mathematical form, with ‘x’ and signs and to write 0 in the result if the polynomial is empty. Also the addTerm(Term) method allows me to add terms to the current polynomial, making the sum between the equal graded terms.

**Operation**

**public** **class** **Operation**{

}

In this class I define methods for every operation required, but division, adding the resulting coefficient and degree to a new term, which was added to the result polynomial. I will describe each method shortly, because I consider that they need a little more explanation, than the other methods in the other classes.

**Polynomial** **add**(**Polynomial** a, **Polynomial** b)

* This method makes the addition of two Polynomials, a and b;
* It goes through the list of polynomial b and verifies if the terms are equal in degree;
* If the terms are equal in degree, the coefficients are added, the degree is copied and the term is added in the resulting polynomial;
* It returns a Polynomial result value;

**Polynomial** **sub**(**Polynomial** a, **Polynomial** b)

* This method subtracts polynomial b from polynomial a, modifying the addition method by putting a “-“ sign in front of the coefficient;
* It goes through all the terms of the polynomial b and locally changes the coefficient;
* The method addTerm from class Polynomial does the rest by checking the dergree of the terms and adding the resembling terms;
* It returns a Polynomial result value;

**Polynomial** **mul**(**Polynomial** a, **Polynomial** b)

* This method multiplies two given polynomials a and b;
* It goes through the terms of both of the polynomials like in real situations: it takes the first term from the first polynomial and multiplies it with each term in the second polynomial and saves the result in a third polynomial;
* As in the previous cases, addTerm takes care of the creation of the result polynomial;
* It returns a Polynomial result value;

**public** **static** **Polynomial** **differentiate**(**Polynomial** a)

* This method returns the differential of a polynomial a;
* The implementation is a really simple one: multiply the degree with the current coefficient, you decrease the it and you get the derivative of the polynomial;
* Again, addTerm does all the job regarding addition of new term to the new, fresh created polynomial;
* The method returns a Polynomial result value;

**public** **static** **Polynomial** **integrate**(**Polynomial** a)

* This method returns the integral of the given polynomial a;
* Again, the implementation didn’t raise any problem, all you have to do is: divide coefficient to degree, increment degree and add the term to the polynomial;
* It also uses addTerm to add the new term to the result polynomial;
* The method returns a Polynomial result value;

**Calculator**

**public** **class** **Calculator** **extends** **JFrame** **implements** ActionListener{

**JPanel** panel = **new** JPanel();

**JButton** addButton = **new** JButton();

**JButton** subButton = **new** JButton();

**JButton** intButton = **new** JButton();

**JButton** derButton = **new** JButton();

**JButton** mulButton = **new** JButton();

**JLabel** polynomial1Label = **new** JLabel("Polynomial 1: ");

**JLabel** polynomial2Label = **new** JLabel("Polynomial 2: ");

**JLabel** resultLabel = **new** JLabel("Result: ");

**JTextPane** tbox1 = **new** JTextPane();

**JTextPane** tbox2 = **new** JTextPane();

**JTextPane** tbox3 = **new** JTextPane();

**String** Poly1 = **new** String();

**String** Poly2 = **new** String();

**Polynomial** p1 = **new** Polynomial();

**Polynomial** p2 = **new** Polynomial();

}

In this class the Graphic User Interface (GUI) is defined by extending the already existent **JFrame** class, that helps us create frames, panels, buttons, labels and text panes (there are more components that can be created using this class, but I’m referring to these because these are the only ones I use for this project) such that we can implement a user-friendly environment.

I set for each element bounds and thus I place each of them in their right place. I mention that no window builder was used in creating this GUI; each element was placed by coding its position. For buttons I added something extra: a background image for each of them and I had to make an ActionListener in which I an action for each of the buttons, so they would do exactly what the user thinks they do.

This class works with polyFromString and toString on each button action, as it has to convert the text from its two text panes into Polynomial objects and when the operation is made, the result has to be converted into text and given back to the

Third text pane. After each calculation the result polynomial has to be cleared so the next result, from the next calculation won’t be added to previous results (each result must be considered separately).

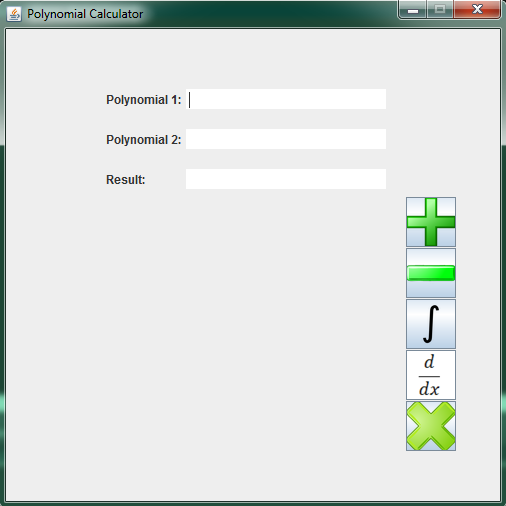
**MainClass**

**public** **class** **MainClass**{

}

This class doesn’t too much. In the main method in it I declared a Calculator object I needed in order to run my interface when the run button is hit.

**5. Implementation testing**

****

The picture above shows a glimpse of what the interface should look like.

The user should do the following in order to obtain the wanted results:

* Enter the terms of the first polynomial in the following form:”coefficientx^degree” separated by signs “+” or “-“;
* Enter the terms of the second polynomial in the exact same manner;
* Press on the button that has the representation of the desired operation;
* The result of the user’s operation will be displayed on the third text pane, the one labeled as “Result:”;
* You can perform the next operation without any refresh or delete or anything; this operation is made in the back programming for the user, so no worries;

Regarding the interface implementation, as I already said explicitly in the description of Calculator class, I used only code, no window builder no nothing design oriented. At first I didn’t know how to implement that but I did a little bit of research and I found out that it’s not that hard. So I positioned everything by the help of the mighty pixels and started making it look bearable.

**5.Results**

I would not say that I did a tremendous job with this Polynomial calculator, but I could say that I nailed it. It’s pretty clever, and object oriented enough, so I would say that I had a great amount of things to learn from this assignment. It wasn’t its best version but I think it deserves some credit.

**6.Further developments**

If I had time, I would totally try:

* To implement division operation on the two polynomials;
* To make it calculate roots of one polynomial, be them complex or not;
* To make a more user friendly interface, so there won’t be any creepy green buttons and any weird things in its window;
* To create separate windows for the operations that require 2 polynomials and for those that require only one;
* To select the number of polynomials the user wants to perform operations on (maximum and minimum numbers);

**7.Conclusions**

In conclusion, this project helped me understand some of the OOP fundamentals, and even if I wasn’t brilliant in this area of activity it didn’t raise me too many problems. From educational point of view, this was a good example of must do during this course.