MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

PROGRAMMING TECHNIQUES

ASSIGNMENT 1

POLYNOMIAL CALCULATOR

Reckerth Daniel Peter

30434/2

Contents

I. Assignment Objective

The aim of this project is to facilitate the design and implementation of a polynomial calculator, a desktop application which allow users to perform operations on their supplied polynomial using a graphical user interface. The functionalities and operations that can be performed consist of classical polynomial operations, i.e. polynomials addition, subtraction, multiplication and division, but also integration and derivation of one polynomial.

In order to fulfil this primary objective, some secondary objectives have been taken into account. Most of them resulted from questions that have been faced during the analysis process. They are:

— designing the model components of the system: how to model the real-world polynomial, monomial and operations concepts used for data manipulation
— designing the view, i.e. how should a polynomial calculator interface look like
— designing the controller, i.e. how should we connect the view with the model of our application in order for the user to successfully interact with the system
— decomposing the problem intro model-view-controller design

II. The problem analysis, modelling, scenarios, use cases

1. Problem analysis

The central element of our application is the polynomial. We present a short theoretical background of what a polynomial is. In order to do that, we have to firstly define the concept of monomial.

A monomial, also called power product, is nothing but a product of power with nonnegative integer exponents, or, in other words, a product of variables, possibly with repetitions, multiplied by a nonzero constant, called the coefficient of the monomial [Wikipedia]. Therefore, a monomial has the following form:

$$ax^n, a \in \mathbb{Z}, n \in \mathbb{N}$$

Thus, a polynomial is an algebraic expression consisting of monomials grouped according to certain patterns. They are sums of monomials, where the polynomial's degree is given by the degree (power) of the highest monomial. The general form of a polynomial is the following:

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

We prefer this representation of a polynomial instead of others (like ordered pairs) due to our implementation characteristics. Also, we had considered this exact representation, with non-increasing order of monomial degrees. Thus, a polynomial starts from its highest monomial as first term.

2. Modelling the problem

When we talk about modelling, we talk about high-level abstractions, devoid of the complexity and low-level details.

Therefore, we refer to the user's perspective of interacting with the system. This is achieved via a graphical user interface (GUI), which provides the necessary capabilities for the user to realize his wishes with regard to our project. There the user can type two polynomials from keyboard or use the virtual keyboard on the GUI, namely the number pad. Then the result of the chosen operation will be displayed on the screen.

We have taken decision in what to be kept and what to be discarded, that is abstractization. We have also followed a well-known architectural pattern, namely the Model View Controller in order to divide the workflow of our application. More of this will be detailed in chapter III regarding the designing decisions. By analysing the real-world data, we have come to decide that in order to represent our data we have to model the following concepts: monomial, polynomial as a list of monomials and operations.

The polynomials that the user inputs will be verified for correctness, as per the form given above. Therefore, the user must input a valid polynomial before any of the operations (addition, subtraction, etc.) can occur. If this is not the case, the user will be notified. This is necessary because what the user enters is modelled by us in the data module. This is the duty of the controller.

Below we present the GUI, the interface with the help of which the user interacts with our system.

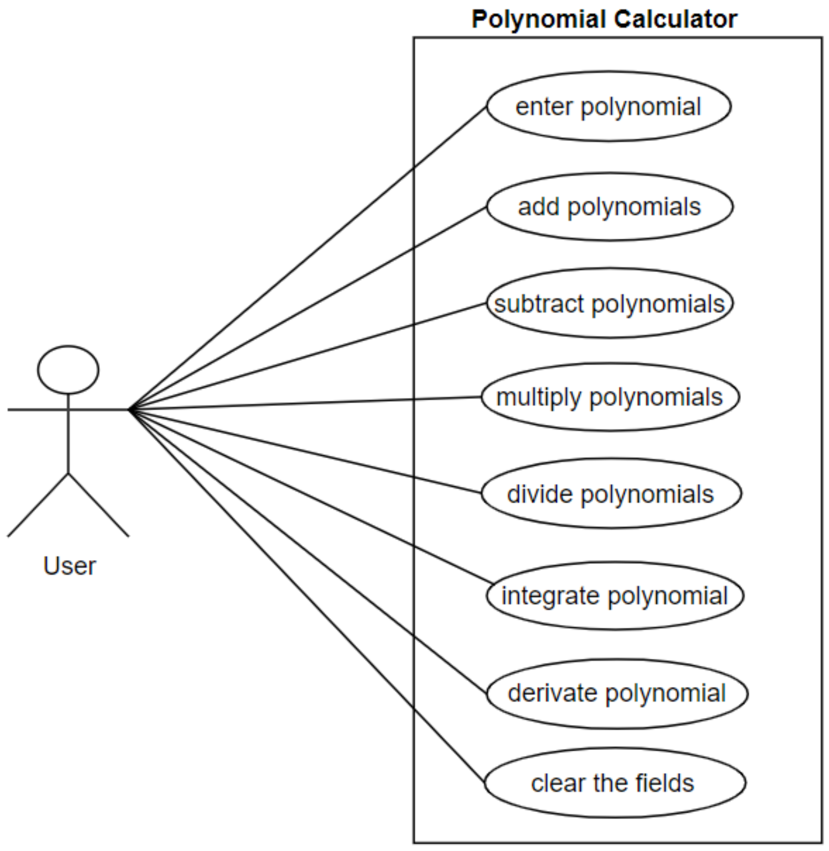
3. Use case scenarios

Scenarios represent different actions that can be taken by the user of the application. Together with a use case, they represent a list of actions or event steps typically defining the interactions between a role (known in UML as an actor) and the system in order to achieve a goal. [Wikipedia]

Therefore, a use case is an important and indispensable requirement analysis technique. Some actions the user can take (the operations), which we talked before are:

- write polynomial
- the addition of two polynomials
- the subtraction of two polynomials
- the multiplication of two polynomials
- the division of two polynomials
- the derivative of one polynomial
- the integration of one polynomial

Below are presented the use case of our system:



III. Design

1. Design Decisions

As mentioned before, our secondary objectives refer to decoupling the data access in accordance with the MVC design pattern. This is very powerful design methodology as it divides our system in three main parts: the model, the view and the controller.

The model represents data and the guidelines that govern access to and updates of this data. The model of our system incorporates our main secondary objective: data classes and data manipulation. The classes which make up the model of our systems are the `Polynomial, Monomial, Operation`. More details about these classes will be explained later.

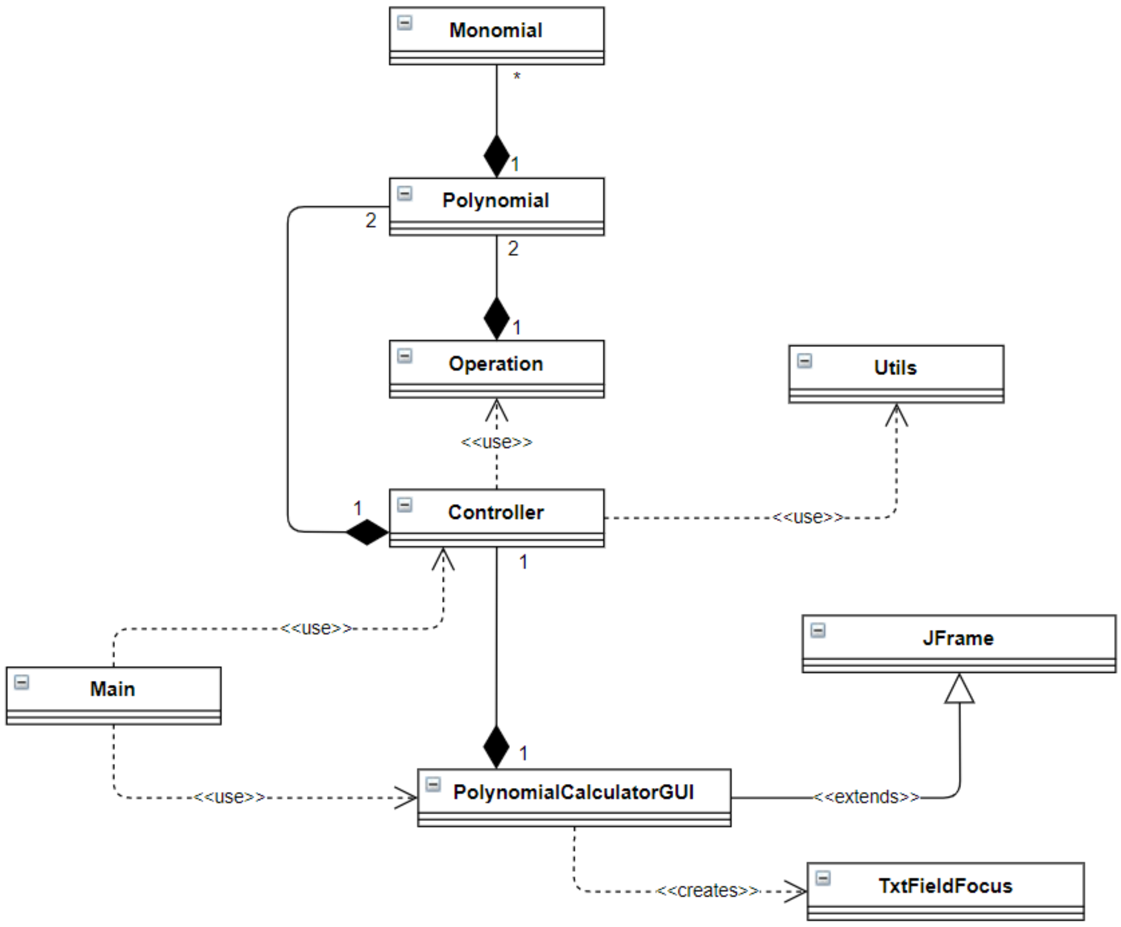
The view renders the contents of the model. It specifies exactly how the model data should be presented. Once the user interacts with the view, the view recognizes that a GUI action occurred, by using a listener method, found in the controller. The view calls the appropriate method on the controller. If the model changes, the view must also update the representations. The view has been realized by using the Swing JFC in order to create our window-based application.

The controller translates the user interactions within the view and calls the model to perform on them. The controller is registered with both the model and the view. Its main purpose it to facilitate communicates between the two, therefore decoupling more the model from the view. The control access both the model and the view, updating and notifying both of them.

2. UML Diagrams

The use case diagram was presented before. In this section we will present the class and package diagrams.

2.1 Class Diagram



This represents the whole class diagram of our system. Classes Polynomial, Monomial, Operation represent the model, classes PolynomialCalculatorGUI represents the view while class Controller is the controller. As it can be seen all are depended on the class Main in which our application runs.

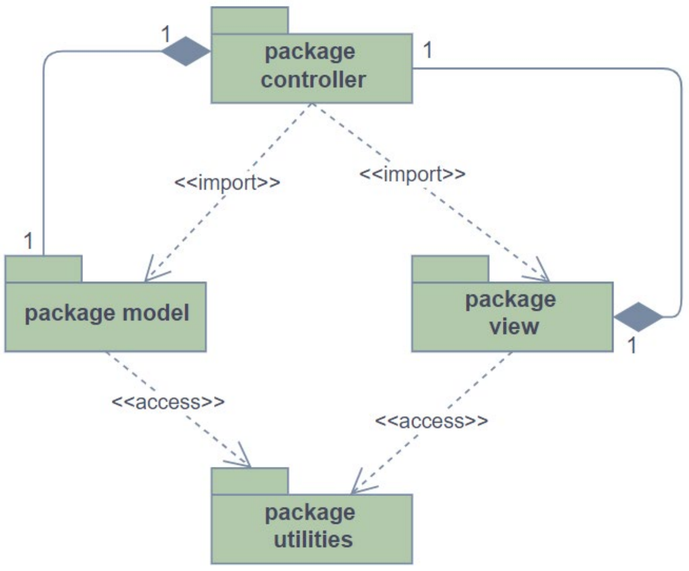
Between the Polynomial class and the Monomial one exists a composition relationship, because the polynomial class contain a list of monomials. In the Operation class we also have two Polynomial fields, respectively the quotient and remainder used for the division algorithm.

The PolynomialCalculatorGUI extends the java JFrame class and also inside it creates a new class TxtFieldFocus which implements the FocusListener interface, used to override the methods focusGained() and focusLost() for the two polynomial text fields in the GUI, in order for us to know which field is focused by the user. We also have a composition relationship between the PolynomialCalculatorGUI class and the Controller class as we need a controller object which will act as the ActionListener for the JButtons.

In the Controller class we also have two Polynomial fields, in which we set the validated and manipulated polynomials received as strings from the user. In the class methods we use the Operation object in order to perform the operation, thus a dependency between the Controller class and Operation one exists. The same relationship is established between the controller class and the Utils one, because we use the utilities static methods in order to validate user input.

2.2 Package diagram

Below is shown the package diagram. To be noted that besides the main model view controller packages we also have a utility package.



3. Data structures

Regarding the used data structures, we have used ArrayList objects in order to add monomial in a list. They were preferred instead of simple array structures due to their simplicity. They are more powerful at providing access to the elements, and also insertion and deletion from them is straightforward.

4. Designing of Classes

Although the given polynomial is made up of integer coefficients operations like division or integration can result in real coefficients after they are performed. In order to solve this problem, the coefficient field in the Monomial class is neither an integer (int) nor a real one (float) but a java language Number object. We use it's method intValue() for the integer outcomes and floatValue() for the real ones. In addition, we have decided to implement the basic operation methods in the Monomial class too, in order to add, subtract, divide, multiply two monomials or to derivate and integrate a monomial.

In the Polynomial class we simply deal with the list of the monomial objects, constructing the polynomial, printing it, order it by descending or ascending powers. In the Operation class however, we have implemented all the operation algorithms and there we have used the basic operation methods in order to add, subtract, etc. the monomials.

The Util class deals with utility functions, for example it has methods for validating user input in which we deal with the pattern matching using regular expressions. Also, we provided a padding method in which we pad with zero monomials the missing monomials in the given polynomial string in order to perform addition, subtraction.

In the View class, with the help of GUI form we have declared all of our text fields, buttons, implemented methods for setting and getting the text out of text fields and most important we have registered the controller as the action listener.

In the controller we have made the link between the GUI and the model. That means that we have implemented here the method actionPerformed() in which we decide what happens when an action occurs in the GUI. We also have implemented the focus listeners for the text field area in order to know which text field was selected with the help of a JTextField auxiliary registering which text field was previously selected.

5. Packages

The main objective of using packages in our project was to organize the modules and group related classes together. This need also resides in the MVC principle we followed.

By this mean, we have three packages: model, controller, and view which groups the related classes. We also have a utility package as shown in the package diagram

In the model package we have the classes: Monomial, Polynomial, Operation.

In the view package we have: PolynomialCalculatorGUI

In the controller package we have the Controller class.

In the utility package we have the Utils cass.

6. Algorithms

No particular algorithms were implemented, except for the division operation inside the Polynomial class.

The algorithm implemented here is known as polynomial short division, or Blomqvis'ts method. It uses the same principle as of the known division, the polynomial long division but the remainder is determined differently (mentally calculated and then set).

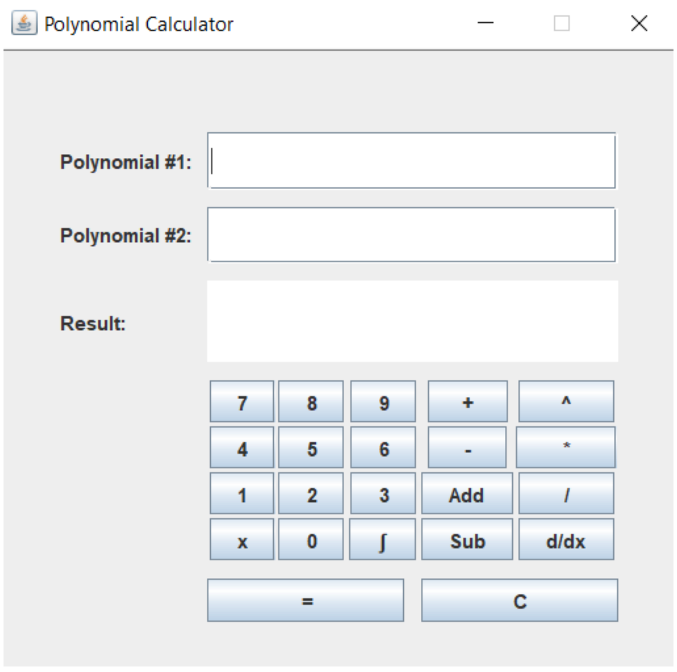
+, -, x represent polynomial arithmetic and / represents a simple division of two terms. It works also for degree(n) < degree(d) with the result being trivial [Wikipedia]. It can be observed that the remainder is determined with successive values, and at first is initialized with the numerator from which subtractions are applied.

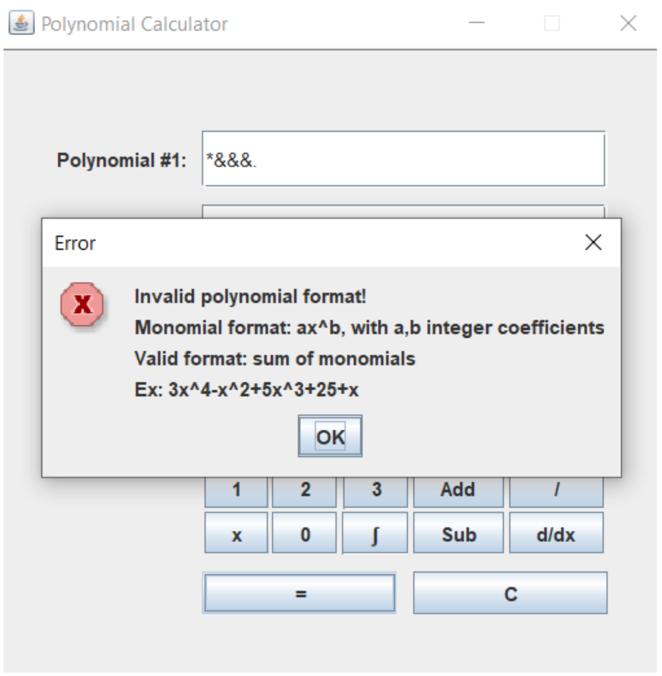
Its pseudocode is:

```
function n / d is
      require d ≠ 0
      q ← 0
      r ← n        // At each step n = d × q + r

while r ≠ 0 and degree(r) ≥ degree(d) do
      t ← lead(r) / lead(d)      // Divide the leading terms
      q ← q + t
      r ← r − t × d

return (q, r)
```

7. Interface

As we said we have used a GUI form but also code for implementing the other functionalities of the GUI. Below are shown some pictures of the GUI and invalid user input:

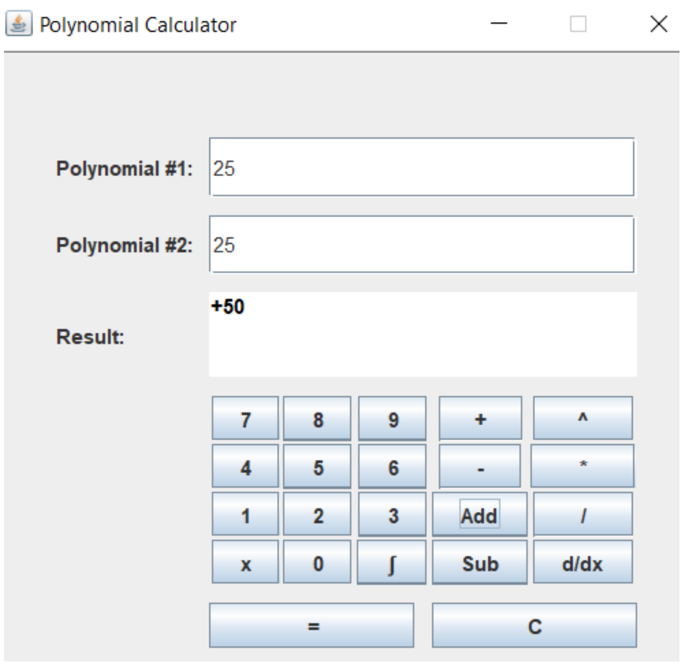



Polynomial #1: *&&&.

Error

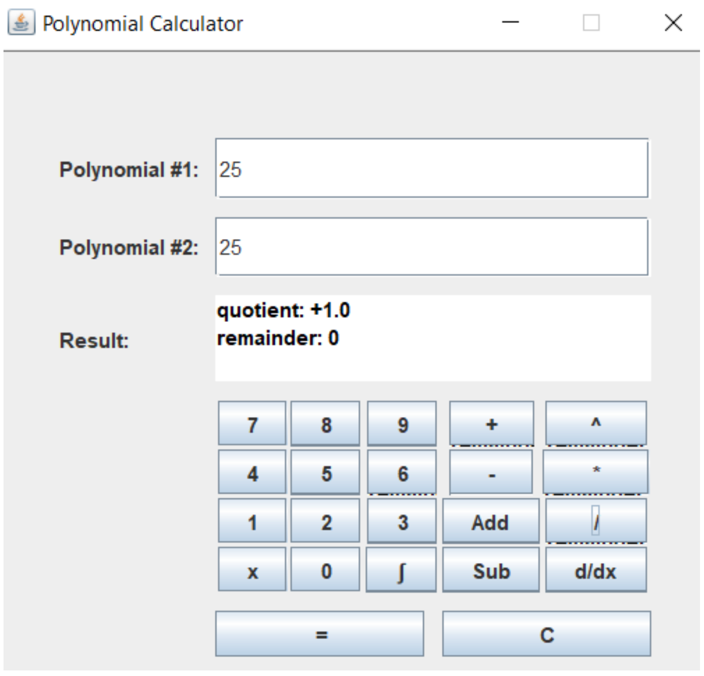
Invalid polynomial format!
Monomial format: ax^b, with a,b integer coefficients
Valid format: sum of monomials
Ex: 3x^4-x^2+5x^3+25+x

The results are shown in a read-only JTextArea field


Polynomial #1: 25

Polynomial #2: 25

Result: +50


Polynomial #1: 25

Polynomial #2: 25

Result: quotient: +1.0
remainder: 0

IV. Implementation

We now present the implementations of the classes with their fields and methods. Because the MVC principle was followed we will divide this section following the MVC components.

1. The Model classes

1.1) Monomial Class: `public class Monomial`

Models the concept of the real-world monomial.

	Name	Description
Fields	`private int power`	the exponent of the monomial
	`private Number coefficient`	the coefficient as Number
Constructors	`public Monomial()`	empty constructor
	`public Monomial(int power, Integer coefficient)`	constructor for integer coefficient
	`public Monomial(int power, Float coefficient)`	constructor for real coefficient
Methods	`public int getPower()`	return power
	`public void setPower()`	set power
	`public Number getCoefficient()`	return coefficient as Number
	`public int getIntegerCoefficient()`	return the integer value of the coeff.
	`public float getRealCoefficient()`	return the real value of the coeff.
	`public void setCoefficient(Number coefficient)`	set the coefficient by a Number obj.
	`private boolean hasEqualPower(Monomial m)`	check if 2 monomial have equal power
	`public Monomial addIntegerCoeffMonomials(Monomial m)`	add two monomials
	`public Monomial subtractIntegerCoeffMonomials(Monomial m)`	subtract two monomial
	`public Monomial multiplyRealCoeff(Monomial m)`	multiply monomials by real coefficient; used for division
	`public Monomial multiplyIntegerCoeff(Monomial m)`	normal multiplication of 2 monomial
	`public void derivate()`	derivate monomial
	`public void integrate()`	integrate monomial
	`public Monomial divide(Monomial m)`	divide a monomial by another

1.2) Polynomial class: public class Polynomial

	Name	Description
Fields	`private ArrayList<Monomial> polynomial`	the array list of monomials
Constructors	`public Polynomial(int power)`	construct an empty polynomial by a given power
	`public Polynomial()`	empty constructor
	`public Polynomial(ArrayList<Monomial> polynomial)`	construct a polynomial from a list of monomials
Methods	`public ArrayList<Monomial> getPolynomial()`	return the polynomial list
	`public void addMonomial(Monomial m)`	add a monomial in the list to the specified index
	`public Monomial getMonomial(int index)`	return the monomial on the specified index
	`public boolean isZero()`	check if the whole list has value zero (coeff=0)
	`public void reduce()`	removes the zero monomials
	`public void sortPolynomialByDescendingPower()`	sorts the list by descending power
	`public void sortPolynomialByAscendingPower()`	sorts the list by ascending power
	`public int getPolynomialDegree()`	returns the highest monomial power
	`public String printPolynomial()`	returns a string of the polynomial

1.3) Operation class: public class Operation

	Name	Description
Fields	`private Polynomial quotient`	the quotient resulted from division
	`private Polynomial remainder`	the remainder resulted from division
Constructors	`public Operation()`	empty constructor
Methods	`public Polynomial add(Polynomial p, Polynomial q)`	return the addition polynomial
	`public Polynomial subtract(Polynomial p, Polynomial q)`	return the subtraction polynomial
	`public Polynomial multiply(Polynomial p, Polynomial q)`	return the multiplication polynomial
	`public Polynomial derivate(Polynomial p)`	return the derivate polynomial
	`public Polynomial integrate(Polynomial p)`	return the integration polynomial
	`public void divide(Polynomial p, Polynomial q)`	perform division
	`public Polynomial getQuotient()`	return the quotient
	`public Polynomial getRemainder()`	return the remainder

2. **The View class:** public class PolynomialCalculatorGUI extends JFrame

	Name	Description
Fields	`private JPanel mainPanel, numberPadPanel`	panel fields
	`private JLabel polynomial1JLabel, polynomal2JLabel,resultJlabel`	label fields
	`private JTextField poly1TxtFld, poly2TxtFld, prevSelectedTextField`	text fields
	`private JButton nb0Button, … nb9Button, xButton, powerButton, addSignButton, minusSignButton, clearButton, equalButton, integrateButton, addButton, subtractButton, divideButton, derivateButton, multiplyButton`	button fields
	`Controller controller;`	the controller object
	`private JTextArea resultTxtArea`	text area
Constructors	`public PolynomialCaculatorGUI(String title)`	constructor, sets title
Methods	`private void createUIComponents()`	custom create, e.g. set text area not editable
	`getters for all fields`	getters for all the fields
	`public void showError(String msg)`	shows an error message dialog
	`public void showWarning(Sring msg)`	shows a warning message dialog
	`public void addActionListeners(ActionListener e)`	register controller as action listener
	`public void setPoly1TxtFldText()`	set the text in text field 1
	`public void setPoly2TxtFldText()`	set the text in text field 2
	`public void addFocusListener(FocusListener f)`	adds focus listener
	`public void appendResultTxtArea(String txt)`	append text in text area
	`public void setResultTextAreaText(String txt)`	set the text area text
	`public boolea polyFldsEmpty()`	true if both text fields are empty
	`public void setPrevSelectedTxtFld(JTextField txtFld)`	set the previously focusable text field
	`public String getPoly1TxtFldText(), getPoly2TxtFldText()`	return the text in both the text fields

3. The Controller Class: public class Controller implements ActionListener

	Name	Description
Fields	`private Polynomial p, q`	polynomial fields
	`private final PolynomialCalculatorGUI calcView`	final GUI component
Constructors	`public Controller(PolynomialCalculatorGUI calcView)`	constructor, add focus listeners

Methods	`private void performNbButtonAction(Object src)`	called in actionPerformed(); append text in fields for buttons 0-9
	`private void performUtilitiesButtonsAction(Object src)`	called in actionPerformed(); append text in fields for "x, ^, +, -" and also clear the fields
	`private boolean validateInputs(Object src)`	getting the string inputs verifies if they are valid; assign them to p, q
	`private void performOperationButtonsAction(Object src)`	calls the Operation methods for the two polynomials, if valid they are displayed else and error/warning message is shown
	`public void actionPerformed(ActionEvent e) <<override>>`	calls all above methods
	`public class TxtFieldFocus implements FocusListener <<override>> methods focusGained(FocusEvent e) and focusLost(FocusEvent e)`	implements the focus listeners for the 2 text fields

4. The Utility class: public class Utils

	Name	Description
Methods	`public static Polynomial paddedPolynomial(Polynomial p, int degree)`	pads a polynomial p with empty monomials given a degree
	`private static int sign(Matcher m)`	called in validateUserIP(), set the signs
	`public static Polynomial validateUserInput(String userStr)`	validates user input with the help of regular expressions

V. Results

For operations validations some Junit testing was done. The results are marked as "pass/fail" are presented below. We have used parametarized tests in order to re-use the same inputs. We have six static methods which return an Object[] array and seven parameterized tests with the given methods source from those 6.

Operation	Polynomials	State
non valid parameters	"AAA", "37x.3", "37.3x", "*&&*!"	pass
valid addition	• "25","25","+50" • "-3x","3x","0" • "","","0" • "3x^4+2x^2+x+25", "2x^4-x^2-x+5", "+5x^4+x^2+30" • "2x^2+75x^7+13x^3-2", "4x^3+15x^15", "+15x^15+75x^7+17x^3+2x^2-2"	pass
valid subtraction	"25", "25", "0" "3x", "3x", "0" "-3x", "3x", "-6x" "3x^4+2x^3-1x^2+x", "3x^5-3x^4+2x^3-25", "-3x^5+6x^4-x^2+x+25" "-2x^2+x+50", "-3x^15-2x+30x^2", "+3x^15-32x^2+3x+50"	pass
valid multiplication	"25", "0", "0" "25", "25", "+625" "-3x", "3x", "-9x^2" "2x+3", "2x-3", "+4x^2-9" "3x^4-2x^3+0x^2-4", "2x-x^2", "-3x^6+8x^5-4x^4+4x^2-8x"	pass
valid derivation	"25", "0" "3x", "+3"	pass

	"", "0" "0", "0" "3x^3+4x^2-x", "+9x^2+8x-1"	
valid integration	"25", "+25.0x" "3x", "+1.5x^2" "", "0" "0", "0" "3x^3+4x^2-x", "+0.75x^4+1.3333334x^3-0.5x^2"	pass
valid division	"25", "25", "+1.0", "0" "3x^3-2x^2+5", "x^2-1", "+3.0x-2.0", "+3x+3" "x^3+3x^2-4x-12", "x^2+x-6", "+x+2.0", "0" "0", "3x^3-4x^4", "0", "0" "3x^7-5-2x^2", "2x^3", "+1.5x^4", "-2x^2-5" "3x^7-2x^4-4x^2", "25", "+0.12x^7-0.08x^4-0.16x^2", "0"	pass

VI. Conclusions

This project proved to be a difficult challenge because there was no previous experience working on such a scale. Many difficult aspects were faced like building a GUI, working by the MVC principle, using GIT and implementing testing units.

The time management and organization were a difficult part for me, because I stumble upon things for many hours. However, in the end I believe I received a lot of new experience about these aspects, but the project is very far from being a successful one, even if everything works accordingly.

There are many aspects to be improved:

- a stronger cohesion between the modules
- better GUI coding style (because it was hard-coded and not used drag-and-drop interfaces like I was used); one example here is that for the project to work, after entering a polynomial in a text field the user must press enter, otherwise the GUI does not work
- better integration of OOP principles
- better development of the MVC principle; there wasn't very clear how to implement the controller – and I believe this was the most difficult part, where to place the action listeners exactly as there are two main principles: pull and push

In conclusion, I believe that even with these drawbacks the experience gained from delivering this project (even as late deadline) proved to be a good and essential exercise.