March, 2016

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

2nd year of study, Computer Science

Laboratory Work – Assignment 1

Polynomial Processing

Lecturer: Prof. Dr. Eng. Ioan Salomie

Teaching Assistant: Delia Balaj

Student: Diana-Gabriela Popescu

Group: 30425

Table of contents

1. Introduction

1.1 Task objectives

The task of the assignment is defined as it follows: “Propose, design and implement a system for polynomial processing. Consider the polynomials of one variable and integer coefficients”. In other words, one should create a user interactive system (by means of console interaction or Graphical User Interface) that is able to manipulate polynomials and to perform basic operations on them.

1.2 Personal approach

This documentation paper aims to present a way of solving the problem of polynomial processing. As for user interface, there will be developed algorithms for user to interact with the Graphical User Interface (GUI) and, for some methods, the output will also be displayed in the console. The solution is obtained by means of implementing several operations specific for polynomials such as: addition, subtraction, multiplication, division, integration and differentiation. This type of operations were chosen due to the fact that they are the most used and the most important operations regarding polynomials.

2. Problem description

2.1 Problem analysis

The analysis of a problem starts from examining the real mathematical model of polynomials or the model we confront with in the real world and passing the problem through a laborious process of abstractization. Hence we identify our problem domain and we try to decompose it in modules easy to implement. Always, having a good model will ease the way the operations are performed and will make more complex programs be clear to read and easy to maintain. My implementation of the solution started with trying to find a suitable and clear model for the polynomials. Also, due to the fact that the user has an important role in running the application, the Graphical

User Interface was also been given a high importance in the process of problem analysis.

So, first of all, one has to analyse the mathematical model of polynomials, for the implemented model to be as close to the real one. The problem domain is defined by the mathematical definition of a polynomial:

$$P(x) = a_n * x^n + a_{n-1} * x^{n-1} + \ldots + a_1 * x^1 + a_0 * x^0 ; a_n \neq 0 ; n \in \mathbb{N}$$

As the mathematical model of polynomials shows, a polynomial "P" is an expression consisting of variables and coefficients. For our specific task, the polynomial has only one variable ("x" $\in \mathbb{R}$), and a list of integer coefficients ($(a_n)_{n\geq 0}$; $a_n \in \mathbb{Z}$). To the polynomial also belongs a value "n", representing the degree of the polynomial. The degree also shows us how many coefficients the polynomial has, i.e. "n+1" coefficients ("a_i" with "i" from 0 to "n"). One should also observe the fact that, in order for the polynomial to be of degree "n", "a_n" should be different from zero. There is only one exception from this rule: when the polynomial's degree is zero, the coefficient a_0 can also be zero.

This idea will also be developed when implementing the division between the input polynomials. Furthermore, it is important to be mentioned that the result obtained after dividing two polynomials will be a new polynomial with real coefficients. As for the others operations, addition, subtraction and multiplication, if the coefficients of the input polynomials are integers, the result obtained by applying these operations on the inputs will again have integer coefficients and a degree related to the operation that is applied.

The addition and subtraction are done coefficient-by-coefficient on the terms having the same degree. The multiplication is done in steps, implying multiplication of each term from the first polynomial with each term from the second polynomial. The division is to be implemented with a specific developed algorithm, that is called " long (polynomial) division ". The integration and differention is only related to the monomials composing the polynomial.

2.2 Modeling

Based on the information presented above, I thought on the actual implementation of the solution. I started to think about what classes are required and how classes should be organized in packages. Also, I realized the need of having an interface for the coefficients' type, because of the possibility of them being either integer or real numbers.

2.3 Scenarios

The scenarios are as it follows: the user can choose the degree of the input polynomials. Then, the coefficients of the two input polynomials are set. Depending on the degree and the coefficients, the user can choose from the available six operations to be performed on the two input polynomials, i.e. addition, subtraction, multiplication, division, integration and differentiation. The user has the possibility of changing both the degree and the coefficients of the inputs. If this happens, the operations will mold around the new inputs.

2.4 Use cases

The use cases are strongly related to the user running the application. Therefore, the user should choose the degrees of the input polynomials.

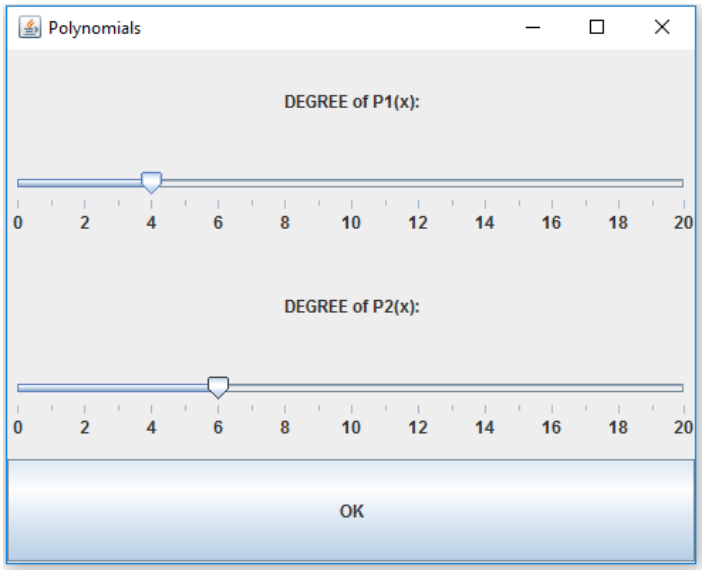

Illustration 1 : Graphical User Interface (setting degrees)

Then, the user should choose the coefficients related to each polynomial. Then, depending on the operation desired to be done, the user can choose from the following: addition, subtraction, multiplication, division, integration and differentiation.

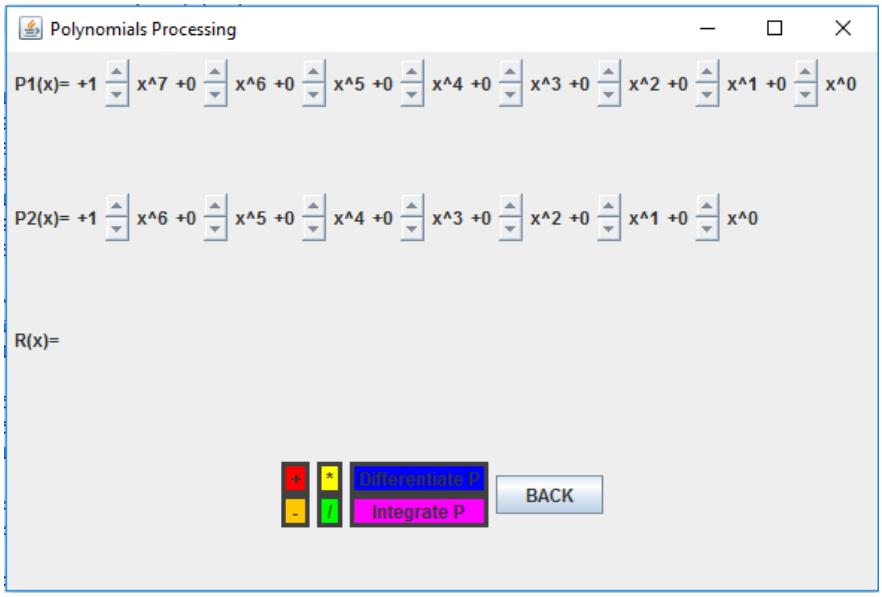

Illustration 2 : Graphical User Interface (setting coefficients and choosing operation)

3. Projection

3.1 UML diagrams

a) Use-case diagram

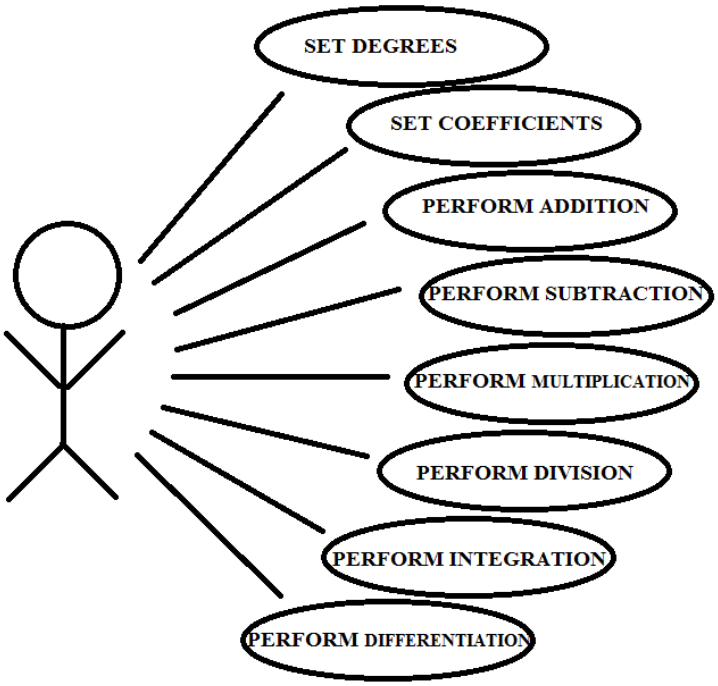

Illustration 3 : Use case diagram

The use case diagram presents the actor, which is the user that interacts with the application. He can perform several actions such as addition, subtraction, multiplication, division, integration and differentiation. The user can also set degrees and coefficients by means of the Graphical User Interface.

b) Class Diagram

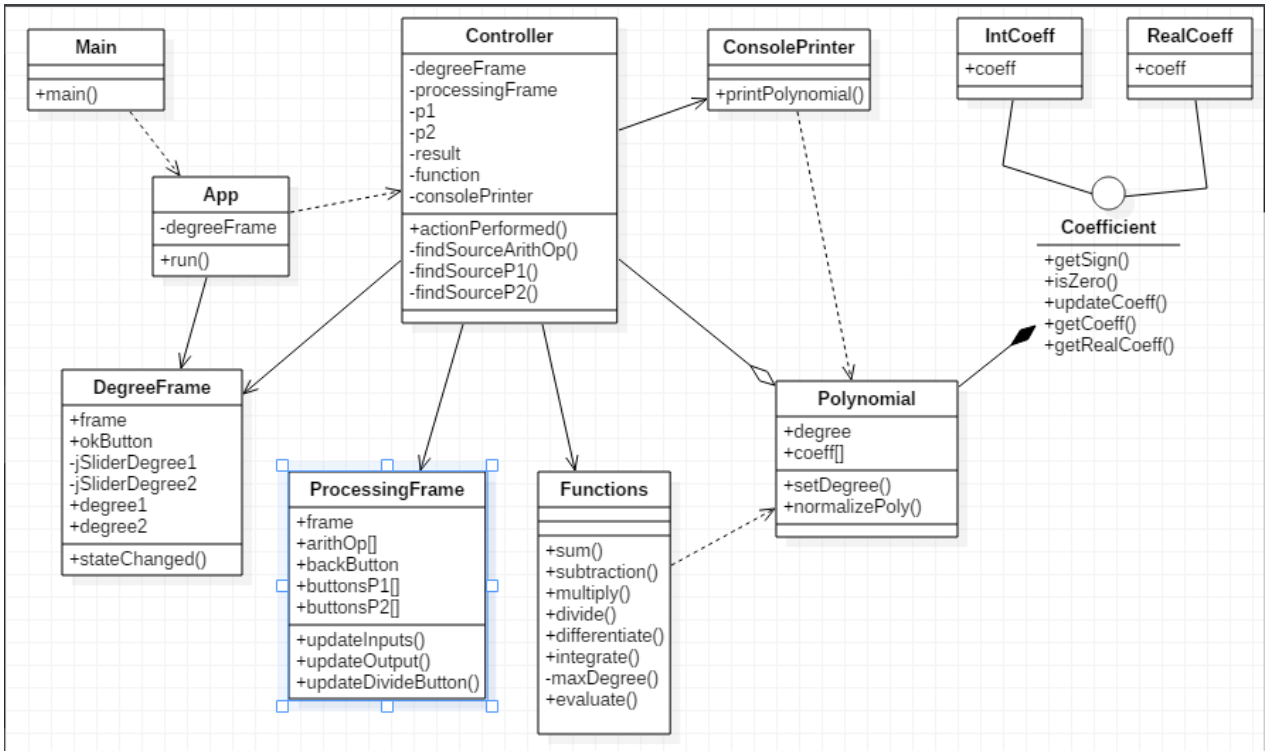

Illustration 4 : Class diagram

As presented in the class diagram, there are various relationships between the classes and interfaces of the system. For example, there is one interface “Coefficient”, which is implemented by two classes: “IntCoeff” and “RealCoeff”. Between class “Polynomial” and interface “Coefficient” there is a composition relationship, just because a polynomial instance does not have sense without coefficients. Also, between class “Controller” and “Polynomial” there exists an aggregation relation, because class “Controller” has three essential fields of type “Polynomial”, but the relation is not that strong as the one previously mentioned.

3.2 Data structures

The data structures used in the application are either primitive data types (int or double) or new types that are defined by the designed classes and interfaces (for example, Polynomial, Coefficient, Functions types). Furthermore, there were used arrays of the types presented above. Not to be forgotten is the use of GUI-like types, such as: JFrame, JButton, JSlider, JLabel, JPanel and BasicArrowButton.

3.3 Class projections

Class projection refers mainly to how the model was thought, how the problem was divided in sub-problems, each sub-problem representing more or less the introduction of a new class. For the beginning, it has to be mentioned that, in my design, I used four different packages for organizing the classes and the interfaces. These are: “poly.IO”, “poly.model”, “poly.operations” and “poly.processing”. The mentioned packages are to be described next, along with the classes and interfaces belonging to them.

a) Package “poly.IO” – contains the design related to the user interface (GUI and console).

- class **ConsolePrinter** - for displaying the polynomials in the console. The main purpose for using such a class is for better tracing of the content of the polynomials.
  The class describes a single method:
  - public void printPolynomial(Polynomial p) – writes in the console, in decreasing order of the degrees, the coefficients of a given polynomial.
- class DegreeFrame - implementing ChangeListener is for establishing the degrees of the two input polynomials. It describes 2 JSliders, which can be modified to obtain the final desired degree. Besides them, there are plenty of other instance variables, as described below:

- private static final int INITIAL_DEGREE1 = 4; -for initializing the degree of polynomial “P1” to a default value.
- private static final int INITIAL_DEGREE2 = 6; -for initializing the degree of polynomial “P2” to a default value.
- private static final int MAX_GRADE = 20; -for setting the maximum degree a polynomial can have.
- public JFrame frame; -the specific frame that has as purpose the establishment of the two degrees.
- private JSlider jSliderDegree1; -for modifying the degree of polynomial “P1”.
- private JSlider jSliderDegree2; -for modifying the degree of polynomial “P2”.
- private JLabel jLabelGrade1; -contains the text "DEGREE of P1(x):".
- private JLabel jLabelGrade2; -contains the text "DEGREE of P2(x):".
- public JButton okButton; -for going to the next frame where the operations upon the polynomials are done.
- public int degree1 = INITIAL_DEGREE1; -when the “okButton” is pressed, “degree1” is set to the last value of the “jSliderDegree1”.
- public int degree2 = INITIAL_DEGREE2; -when the “okButton” is pressed, “degree2” is set to the last value of the “jSliderDegree2”.

The constructor of the class instantiates and initializes the instance variables to the default values.

The class implements one method:

- public void stateChanged(ChangeEvent event) –has to deal with the events that occur when the values of the sliders are modified. Each time a new value is found, the variables “degree1” and “degree2” are modified.

- class ProcessingFrame - contains the main user interface of the application. It contains the arithmetical operation' buttons, and

also some other buttons for modifying the coefficients of the input polynomials.

- public static final int PLUS = 0; -constant used as index for the related arithmetical operation.
- public static final int MINUS = 1; -constant used as index for the related arithmetical operation.
- public static final int MUL = 2; -constant used as index for the related arithmetical operation.
- public static final int DIV = 3; -constant used as index for the related arithmetical operation.
- public static final int INTEGRATE = 4; -constant used as index for the related arithmetical operation.
- public static final int DIFF = 5; -constant used as index for the related arithmetical operation.
- public static final int NR_OF_OP = 6; -constant used for indicating the number of available arithmetical operations.
- private static final int LINE_BORDER_THICKNESS = 3; -constant used for setting the default thickness of the border for each arithmetical operation button.
- public JFrame frame; -the specific frame that has as purpose the establishment of the coefficients and performing the arithmetical operations upon the polynomials.
- public JButton[] arithOp; -the buttons related to the arithmetical operations.
- public JButton backButton; -if the user wants to change the degree of a input polynomial, there exists the possibility of going back to the “degreeFrame” and setting another values for the degrees of the two input polynomials.
- public BasicArrowButton[] buttonsP1; -for changing the coefficients of the polynomial “P1”. The user can either increase or decrease the integer coefficient.

- public BasicArrowButton[] buttonsP2; -for changing the coefficients of the polynomial “P2”. The user can either increase or decrease the integer coefficient.
- private JLabel jLabelP1; -contains the text "P1(x)=".
- private JLabel jLabelP2; -contains the text "P2(x)=".
- private JLabel jLabelP3; -contains the text "R(x)=".
- private JLabel jLabelResult; -for displaying the result obtained after performing an arithmetical operation.
- private JPanel jPanelLine1; -contains the data belonging the the 1st line of the frame.
- private JPanel jPanelLine2; -contains the data belonging the the 2nd line of the frame.
- private JPanel jPanelLine3; -contains the data belonging the the 3rd line of the frame.
- private JPanel jPanelLine4; -contains the data belonging the the 4th line of the frame.
- private JPanel jPanelPlusMinus; -contains the two specific arithmetical operations.
- private JPanel jPanelMulDiv; -contains the two specific arithmetical operations.
- private JPanel jPanelDiffInt; -contains the two specific arithmetical operations.
- private JPanel[] jPanelButtonsP1; -contains the buttons used for changing the coefficients of the polynomial “P1”.
- private JPanel[] jPanelButtonsP2; -contain the buttons used for changing the coefficients of the polynomial “P2”.
- private JLabel[] jLabelCoeffP1; -contain the actual coefficients of the input polynomial “P1”.
- private JLabel[] jLabelCoeffP2; -contain the actual coefficients of the input polynomial “P2”.

- private JLabel[] jLabelPowersOfX1; -contain a text representing the power of "x" related to the local degree and coefficient.
- private JLabel[] jLabelPowersOfX2; -contain a text representing the power of "x" related to the local degree and coefficient.

The constructor of the class instantiates and initializes the instance variables to the default values.

The described methods are:

- public void updateInputs(Polynomial p1, Polynomial p2) –updates the displayed coefficients of the two input polynomials.
- public void updateOutput(Polynomial p1) - updates the displayed coefficients of the resulted polynomial.
- public void updateDivideButton(Polynomial p1, Polynomial p2) –verifies if the two input polynomials can perform division. In case they do not, the button is disabled, i.e. has no effect on the output.

b) Package "poly.model"

- interface Coefficient - describes the capabilities of the two types of coefficients: integers or double. The methods are:
  - public char getSign();
  - public boolean isZero();
  - public void updateCoeff(int input);
  - public void updateCoeff(double input);
  - public int getCoeff();
  - public double getRealCoeff();
- class IntCoeff -implementing Coefficient interface describes the integer coefficients of the polynomials. It contains one and only instance variable:
  - public int coeff; - represents the actual integer value of the coefficient.

The constructor of the class instantiates and initializes the instance variable to the provided integer value.

The class implements the methods described by the "Coefficient" interface:

- public char getSign(); -returns '-' for negative values and '+' for positive ones (including zero).
- public boolean isZero(); -returns "true" is the value is zero and "false" otherwise.
- public void updateCoeff(int input); -updates the value of the coefficient with another input.
- public void updateCoeff(double input); -updates the value of the coefficient with another input. For this to happen, the "double" value is downcasted to "int".
- public int getCoeff(); -returns the value of the integer coefficient.
- public double getRealCoeff(); -returns the casted value of the integer coefficient.

- class RealCoeff - implementing Coefficient interface describes the real/double coefficients of the polynomials. It contains one and only instance variable:
  - public double coeff; - represents the actual integer value of the coefficient.

  The constructor of the class instantiates and initializes the instance variable to the provided "double" value.

  The class implements the methods described by the "Coefficient" interface:

  - public char getSign(); -returns '-' for negative values and '+' for positive ones (including zero).
  - public boolean isZero(); -returns "true" is the value is zero and "false" otherwise.
  - public void updateCoeff(int input); -updates the value of the coefficient with another input. For this to happen, the "int" value is casted to "double".

- public void updateCoeff(double input); -updates the value of the coefficient with another input.
- public int getCoeff(); -returns the downcasted value of the integer coefficient.
- public double getRealCoeff(); -returns the value of the real coefficient.

c) Package “poly.operations”

- class Functions - describes the methods that can be performed upon polynomials. These are:
  - public Polynomial sum(Polynomial p1, Polynomial p2) –returns a new polynomial resulted after summing up the coefficients of the parameters term-wise.
  - public Polynomial subtraction(Polynomial p1, Polynomial p2) –returns a new polynomial resulted after subtracting the coefficients of the parameters term-wise.
  - public Polynomial multiply(Polynomial p1, Polynomial p2) –returns a new polynomial resulted after multiplying the provided polynomials.
  - public Polynomial divide(Polynomial p1, Polynomial p2) –returns a new polynomial resulted after dividing the provided polynomials. The model of the method is the long-division algorithm of Euclid.
  - public Polynomial differentiate(Polynomial p1) – returns a new polynomial representing the first derivative of the given polynomial.
  - public Polynomial integrate(Polynomial p1) –returns a new polynomial representing the first primitive of the given polynomial.
  - private int maxDegree(int a, int b) –returns the maximum between the degrees of two polynomials.
  - public double evaluate(Polynomial p1, int x) –method used for evaluating the value of a polynomial for a certain value for “x”.

d) package “poly.processing”

- class Controller -implementing ActionListener describes the relationships between the GUI interface and the logical structures of the application.
  The instance variables are:
  - private DegreeFrame degreeFrame; -for establishing the degrees of the polynomials.
  - private ProcessingFrame processingFrame; -for modifying coefficients and performing operations.
  - private Polynomial p1, p2, result; -the input polynomials and the output polynomial obtained after one operation.
  - private Functions function; -instance of class “Functions” as reference to the implementation of the arithmetical operations.
  - private ConsolePrinter consolePrinter; -instance of class “ConsolePrinter” for using the specific method.

  The constructor of the class instantiates and initializes the instance variables to the default values.
  Implemented methods:
  - public void actionPerformed(ActionEvent event) - for dealing with the events that occur when interacting with the GUI.
  - private int findSourceP1(Object source) –verifies if the source of the event if among the buttons for modifying the coefficients of polynomial “P1”.
  - private int findSourceP1(Object source) –verifies if the source of the event if among the buttons for modifying the coefficients of polynomial “P2”.
  - private int findSourceArithOp(Object source) –verifies if the source of the event if among the buttons for performing arithmetical operations on the polynomials.
- class App - implementing Runnable describes the way the application runs. It has a “degreeFrame instance” variable:
  - private DegreeFrame degreeFrame;

The constructor of the class instantiates and initializes the instance variable to the given value.
Then, it instantiates a Controller object by calling its constructor with the instantiated “degreeFrame” as parameter, all this in one method:

- public void run()

- class Main - contains the main method. It runs the App.

```
public static void main(String[] args) {
    new App().run();
}
```

3.4 Interface

This section being already developed in detail until now, I will remember briefly some important facts. The user interface is mainly realized by means of “java.swing” package. I used instances of JFrame, JPanel, JButoon, JSlider, BasicArrowButton, JLabel. Also, I used FlowLayout and GridLayout for organizing components inside the frame and panels. The buttons and the sliders are the main way of the user interacting with the application.

3.5 Relationships

As presented in the class diagram, there are various relationships between the classes and interfaces of the system. For example, there is one interface “Coefficient”, which is implemented by two classes: “IntCoeff” and “RealCoeff”. Between class “Polynomial” and interface “Coefficient” there is a composition relationship, just because a polynomial instance does not have sense without coefficients. Also, between class “Controller” and “Polynomial” there exists an aggregation relation, because class “Controller” has three essential fields of type “Polynomial”, but the relation is not that strong as the one previously mentioned.

There are also some other dependencies and associations due to the fact that some methods use as parameter types of a different class.

3.6 Packages

The program is divided in four packages, as mentioned before. The package “poly.IO” contains classes: “DegreeFrame”, “ProcessingFrame”, “ConsolePrinter”. The package “poly.model” contains classes: “Polynomial”, “IntCoeff”, “RealCoeff” and the interface “Coefficient”. The package “poly.operations” contains class “Functions”. The package “poly.processing” contains classes: “Controller”, “App” and “Main”.

3.7 Algorithms

The main algorithms regarding polynomials are the following: addition, subtraction, multiplication, division, integration and differentiation. Using the mathematical model for each operation, one can design a suitable algorithm. The algorithms designed by me are contained by “Functions” class. They are designed as methods handling polynomials, as it follows:

- public Polynomial sum(Polynomial p1, Polynomial p2) –returns a new polynomial resulted after summing up the coefficients of the parameters term-wise.
- public Polynomial subtraction(Polynomial p1, Polynomial p2) – returns a new polynomial resulted after subtracting the coefficients of the parameters term-wise.
- public Polynomial multiply(Polynomial p1, Polynomial p2) – returns a new polynomial resulted after multiplying the provided polynomials.
- public Polynomial divide(Polynomial p1, Polynomial p2) –returns a new polynomial resulted after dividing the provided polynomials. The model of the method is the long-division algorithm of Euclid.
- public Polynomial differentiate(Polynomial p1) –returns a new polynomial representing the first derivative of the given polynomial.
- public Polynomial integrate(Polynomial p1) –returns a new polynomial representing the first primitive of the given polynomial.

3.8 Graphical User Interface

An important fact to be mentioned is the use of "java.swing" package. I used instances of JFrame, JPanel, JButoon, JSlider, BasicArrowButton, JLabel. Also, I used FlowLayout and GridLayout for organizing components inside the frame and panels. The buttons and the sliders are the main way of the user interacting with the application.

4. Implementation and testing

The implementation was done in Eclipse and it was also tested in this environment. However the program should maintain its portability. Concerning the code implementation I did not make use of laborious algorithms, but I have rather stayed faithful to the classical algorithms of computing polynomials. The personal touch in the implementation is also felt in the way the Graphical User Interface is thought.

Testing implies checking for any errors and warnings in the program or limitations of this program. Regarding the input polynomials, the user cannot introduce a non-integer coefficient, since the modification is strictly managed by use of buttons. The only possible errors are related to a possible wrong result of computation when applying some arithmetical operation. It is also developed the problem of "division by 0" in the division method. In this case, an exception is thrown, which can also be handled by the method. So, from this point of view, there are lots of errors that will be avoided this way. Other possible scenarios will be tried as future development.

5. Results

The application is user friendly and useful in performing basic polynomial operations such as: addition, subtraction, multiplication, division, differentiation and integration. As the application is developed on a Java platform, it is highly portable and allows it to run on several operating systems. The application is to be

used by anyone who is interested in performing these types of operations and has a basic knowledge about polynomials.

6. Conclusions

The application can be further developed, by adding some other operations on polynomials. For me personally, the design of this system helped me think about the way classes should be organized in packages. Also, at first, I thought and tried using ArrayList for memorizing the coefficients of the polynomials. But this way the code became to unclear and not suggestive at all. So I preffered using a more simple tool: arrays.

7. Bibliography

- http://staruml.io/
  -for downloading a tool for creating class diagrams.
- https://en.wikipedia.org/wiki/Long_division
  -for developing the division algorithm.
- http://www.purplemath.com/modules/polydiv2.htm
  -for developing the division algorithm.
- http://bellekens.com/2010/12/20/uml-composition-vs-aggregation-vs-association/
  -for better understanding of the relationships between classes.