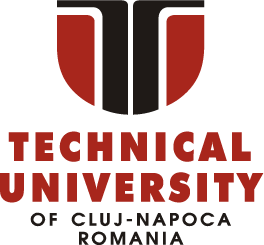
**TECHNICAL UNIVERSITY OF CLUJ-NAPOCA**

2nd year of study, Computer Science



Laboratory Work – Assignment 2

Order Management

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 . . . . . . . . . . . . . . . . . . . . . . . . . .page 3

1.2 Personal approach . . . . . . . . . . . . . . . . . . . . . . . .page 3

**2. Problem description**

2.1 Problem analysis . . . . . . . . . . . . . . . . . . . . . . . . .page 3

2.2 Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page 4

2.3 Scenarios . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page 4

2.4 Use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .page 5

**3. Projection**

3.1 UML diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . page 8

3.2 Data structures . . . . . . . . . . . . . . . . . . . . . . . . . . page 8

3.3 Class projections . . . . . . . . . . . . . . . . . . . . . . . . page 8

3.4 Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 16

3.5 Relationships . . . . . . . . . . . . . . . . . . . . . . . . . . . page16

3.6 Packages . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 17

3.7 Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 17

3.8 Graphical User Interface . . . . . . . . . . . . . . . . . . page18

**4. Implementation and testing** . . . . . . . . . . . . . . . . . . . . . . . .page 18

**5. Results** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 18

**6. Conclusions** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 19

**7. Bibliography** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . page 22

**1. Introduction**

1.1 Task objectives

The task of the assignment is defined as it follows: “Consider an application OrderManagement for processing customer orders. The application uses (minimally) the following classes: Order, OPDept (Order Processing Department), Customer, Product, and Warehouse. The classes OPDept and Warehouse use a BinarySearchTree for storing orders”. In other words, one should create a user interactive system (by means of console interaction or Graphical User Interface) that is able to manipulate products and orders and to perform basic operations on them.

1.2 Personal approach

This documentation paper aims to present a way of solving the problem of Order Management. 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 products and orders such as: adding a new product, removing a product, modifying the characteristics of a product, placing an order, visualizing the placed orders. This type of operations were chosen due to the fact that they are the most used and the most important operations regarding Order Management.

**2. Problem description**

2.1 Problem analysis

The analysis of a problem starts from examining the real model of Order Management 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 products and orders. 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 real model of products and orders, for the implemented model to be as close to the real one. The problem domain is defined by the real definition of a product or an order.

To a product, there is associated a name, a price, an available stock and a description. One should see that the price and the stock must be unsigned integers, and that the description might be missing or unnecessary. Also, an order must contain information about both the product and the customer that placed it, and also the number of products the customer required.

This idea will also be developed when implementing the constructor of the classes. Furthermore, it is important to be mentioned that the warehouse contains a collection of products. Similarly, the Order Department has a collection of placed orders.

2.2 Modeling

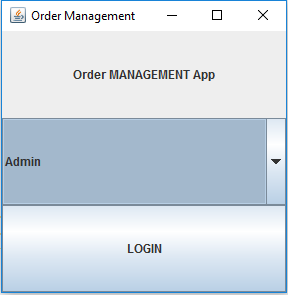
Based on the information presented above, I thought on the actual implementation of the solution. I started to think about what classes are required and how classes should be organized in packages. Also, I realized the need of having an abstract class for the customer / administrator frames, because of the possibility of them being similar.

2.3 Scenarios

The scenarios are as it follows: the user can choose to log in as administrator or as customer (or guest). Then, a new frame is open. Depending on the log in information, the user can choose from the available operations to be performed on the input products or orders, i.e. adding a new product, removing a product, modifying the characteristics of a product, placing an order, visualizing the placed orders. The user has the possibility of changing the log in information by logging out and choosing the other option. If this happens, the operations will mold around the new inputs and available operations.

2.4 Use cases

The use cases are strongly related to the user running the application. Therefore, the user should choose to log in as administrator or as guest, i.e. customer.



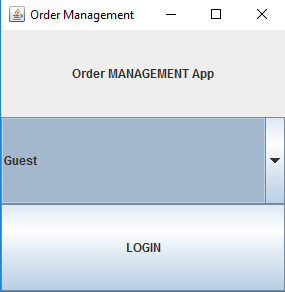


Illustration 1 : Graphical User Interface ( logging in )

Then, the user should choose the information that suits best to the situation. Then, depending on the operation desired to be done, the user can choose from the following: adding a new product, removing a product, modifying the characteristics of a product, placing an order, visualizing the placed orders.

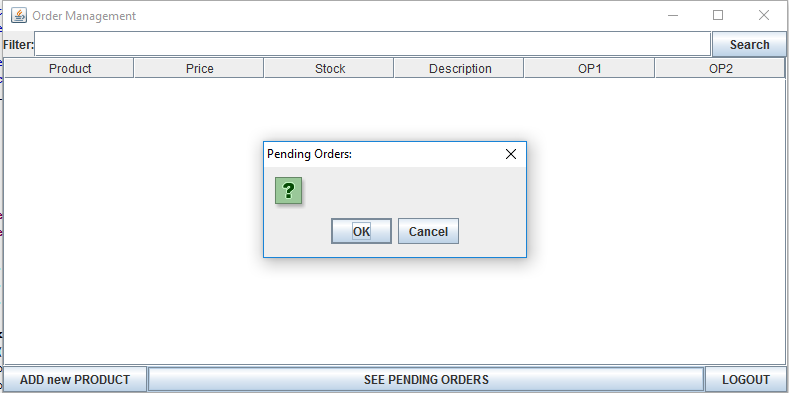
****

Illustration 2 : Graphical User Interface ( admin VIEW )

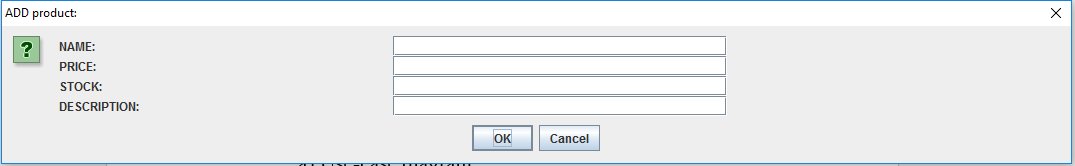


Illustration 3 : Graphical User Interface ( admin VIEW – adding a new product )

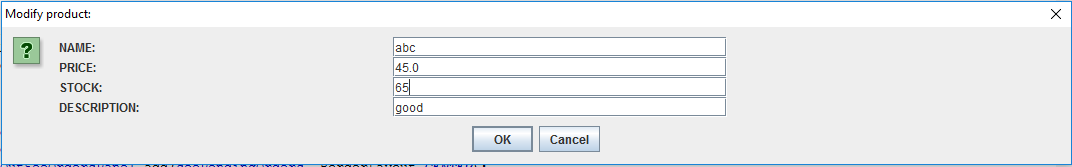


Illustration 4 : Graphical User Interface ( admin VIEW – modifying an existing product )

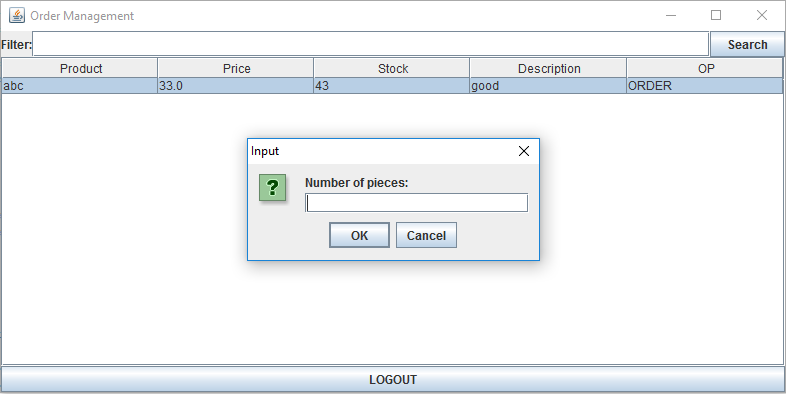


Illustration 5 : Graphical User Interface ( customer VIEW – introducing desired number of pieces)

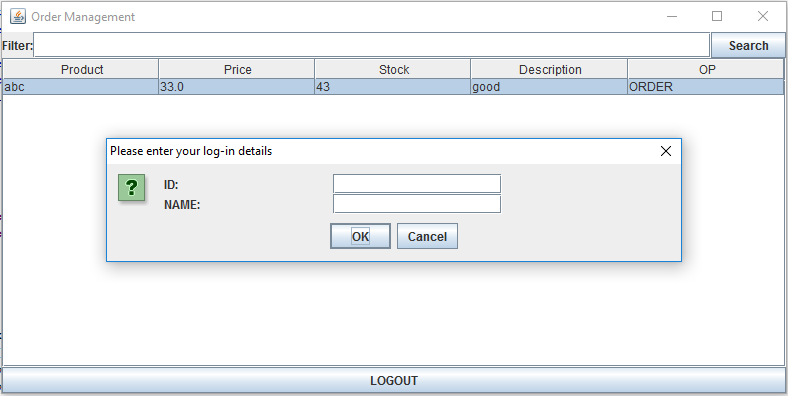
****

Illustration 6 : Graphical User Interface ( customer VIEW – account info)

To be mentioned is the fact that there are used a lot of interactive “JOptionPane” for requesting user – input. For example, when adding a new product, modifying one already existing, when providing the log in information, when visualizing the pending orders, and so on.

**3. Projection**

3.1 UML diagrams

a) Use-case diagram

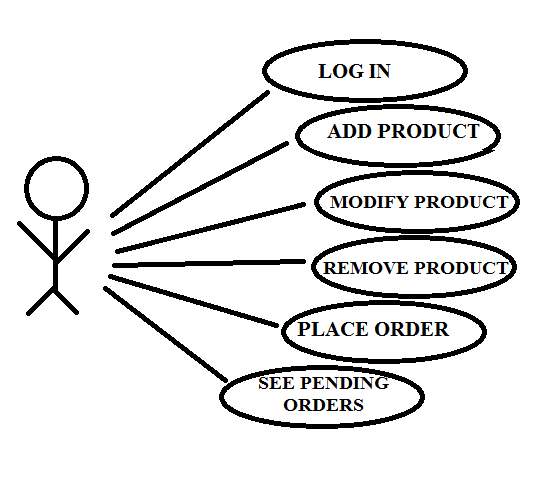


Illustration 7 : Use case diagram

The use case diagram presents the actor, which is the user that interacts with the application. He can perform several actions such as adding a new product, removing a product, modifying the characteristics of a product, placing an order, visualizing the placed orders.

b) Class Diagram

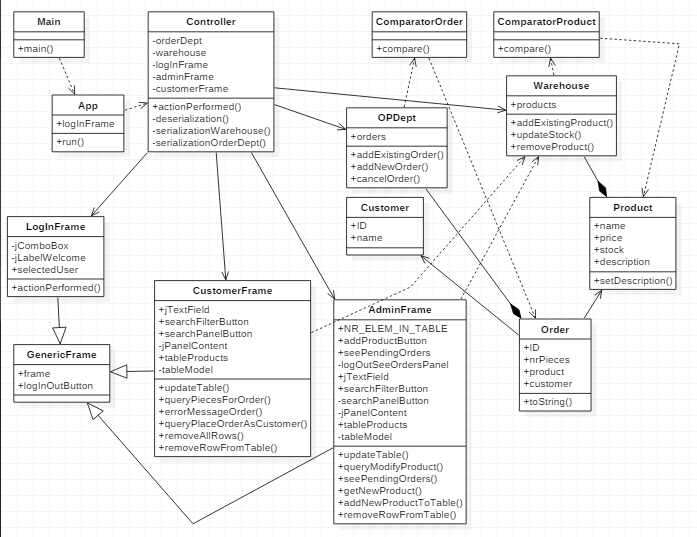


Illustration 8 : Class diagram

As presented in the class diagram, there are various relationships between the classes of the system. For example, there is one abstract class “GenericFrame”, which is extended by three classes: “LogInFrame”, “CustomerFrame” and “AdminFrame”. Between class “Warehouse” and class “Product” there is a composition relationship, just because a warehouse instance does not have sense without products. Also, between class “OPDept” and “Order” there exists an composition relation, because class “OPDept” has field of type “Order”, and the relation is very strong, as the one previously mentioned.

c) Sequence Diagrams

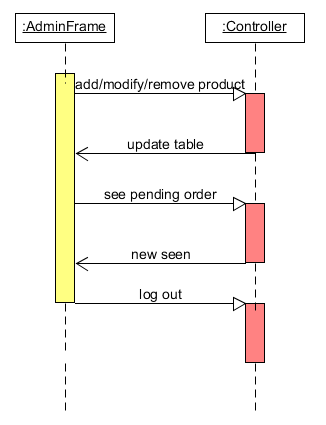


Illustration 9 : Sequence diagram #1

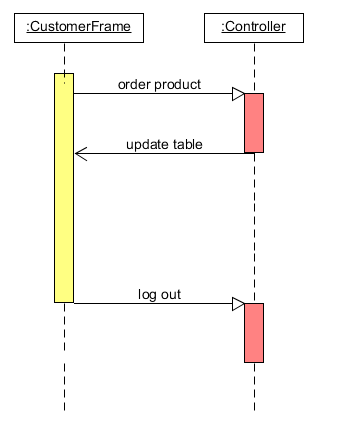


Illustration 10 : Sequence diagram #2

d) Activity Diagram

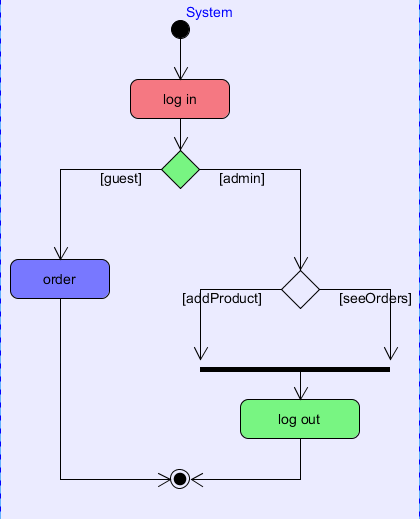


Illustration 11 : Activity diagram

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 ( for example, Product, Order, Customer 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, JLabel, JPanel, JComboBox and JOPtionPane.

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: “order.management.IO”, “order.management.models” and “order.management.main”. The mentioned packages are to be described next, along with the classes belonging to them.

1. Package “order.management.IO” – contains the design related to the user interface (GUI and console).

* class **ConsolePrinter** - for displaying the order/customers in the console. The main purpose for using such a class is for better tracing of the content of the order/customers.

The class describes a single method:

* public void printOrder/customer(Order/customer p) – writes in the console, in decreasing order of the degrees, the coefficients of a given order/customer.
* class DegreeFrame - implementing ChangeListener is for establishing the degrees of the two input order/customers. 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 order/customer “P1” to a default value.
* private static final int INITIAL\_DEGREE2 = 6; -for initializing the degree of order/customer “P2” to a default value.
* private static final int MAX\_GRADE = 20; -for setting the maximum degree a order/customer 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 order/customer “P1”.
* private JSlider jSliderDegree2; -for modifying the degree of order/customer “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 order/customers 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 order/customers.
* 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 order/customers.
* public JButton[] arithOp; -the buttons related to the arithmetical operations.
* public JButton backButton; -if the user wants to change the degree of a input order/customer, there exists the possibility of going back to the “degreeFrame” and setting another values for the degrees of the two input order/customers.
* public BasicArrowButton[] buttonsP1; -for changing the coefficients of the order/customer “P1”. The user can either increase or decrease the integer coefficient.
* public BasicArrowButton[] buttonsP2; -for changing the coefficients of the order/customer “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 order/customer “P1”.
* private JPanel[] jPanelButtonsP2; -contain the buttons used for changing the coefficients of the order/customer “P2”.
* private JLabel[] jLabelCoeffP1; -contain the actual coefficients of the input order/customer “P1”.
* private JLabel[] jLabelCoeffP2; -contain the actual coefficients of the input order/customer “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(Order/customer p1, Order/customer p2) –updates the displayed coefficients of the two input order/customers.
* public void updateOutput(Order/customer p1) - updates the displayed coefficients of the resulted order/customer.
* public void updateDivideButton(Order/customer p1, Order/customer p2) –verifies if the two input order/customers can perform division. In case they do not, the button is disabled, i.e. has no effect on the output.

1. 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 order/customers. 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 order/customers. 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.

1. package “order.management.models”

* 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 order/customers.
* private ProcessingFrame processingFrame; -for modifying coefficients and performing operations.
* private Order/customer p1, p2, result; -the input order/customers and the output order/customer 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 order/customer “P1”.
* private int findSourceP1(Object source) –verifies if the source of the event if among the buttons for modifying the coefficients of order/customer “P2”.
* private int findSourceArithOp(Object source) –verifies if the source of the event if among the buttons for performing arithmetical operations on the order/customers.
* 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 “Order/customer” and interface “Coefficient” there is a composition relationship, just because a order/customer instance does not have sense without coefficients. Also, between class “Controller” and “Order/customer” there exists an aggregation relation, because class “Controller” has three essential fields of type “Order/customer”, 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: “Order/customer”, “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 order/customers 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 order/customers, as it follows:

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

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 order/customers. 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 order/customers, 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 order/customer 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 order/customers.

**6. Conclusions**

The application can be further developed, by adding some other operations on order/customers. 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 order/customers. 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.