Technical University of Cluj-Napoca

Programming Techniques

Laboratory – Assignment 3

Database Management System

Student: Cristian Casian-Cristi

Group: 30422

1) Objective

Consider an application OrderManagement for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders.

Furthermore, the application uses (minimally) the following classes:

- Model classes - represent the data models of the application (for example Order, Client, Product)
- Business Logic classes - contain the application logic
- Presentation classes – classes that contain the graphical user interface
- Data access classes - classes that contain the access to the database

Other classes and packages can be added to implement the full functionality of the application.

a. Analyze the application domain, determine the structure and behavior of its classes and draw an extended UML class diagram.

b. Implement the application classes. Use javadoc for documenting classes.

c. Use reflection techniques to create a method that receives a list of objects and generates the header of the table by extracting through reflection the object properties and then populates the table with the values of the elements from the list.

2) Problem analysis, scenarios, use cases

- General overview

  This application should be able to fulfil all the requirements in order to display, modify, and keep track of orders, clients and products. These are stored in a relational MySQL database, along with the information about the users which have access to the system. This way, all the data is easier to retrieve and access from different computers.

- Input and Output

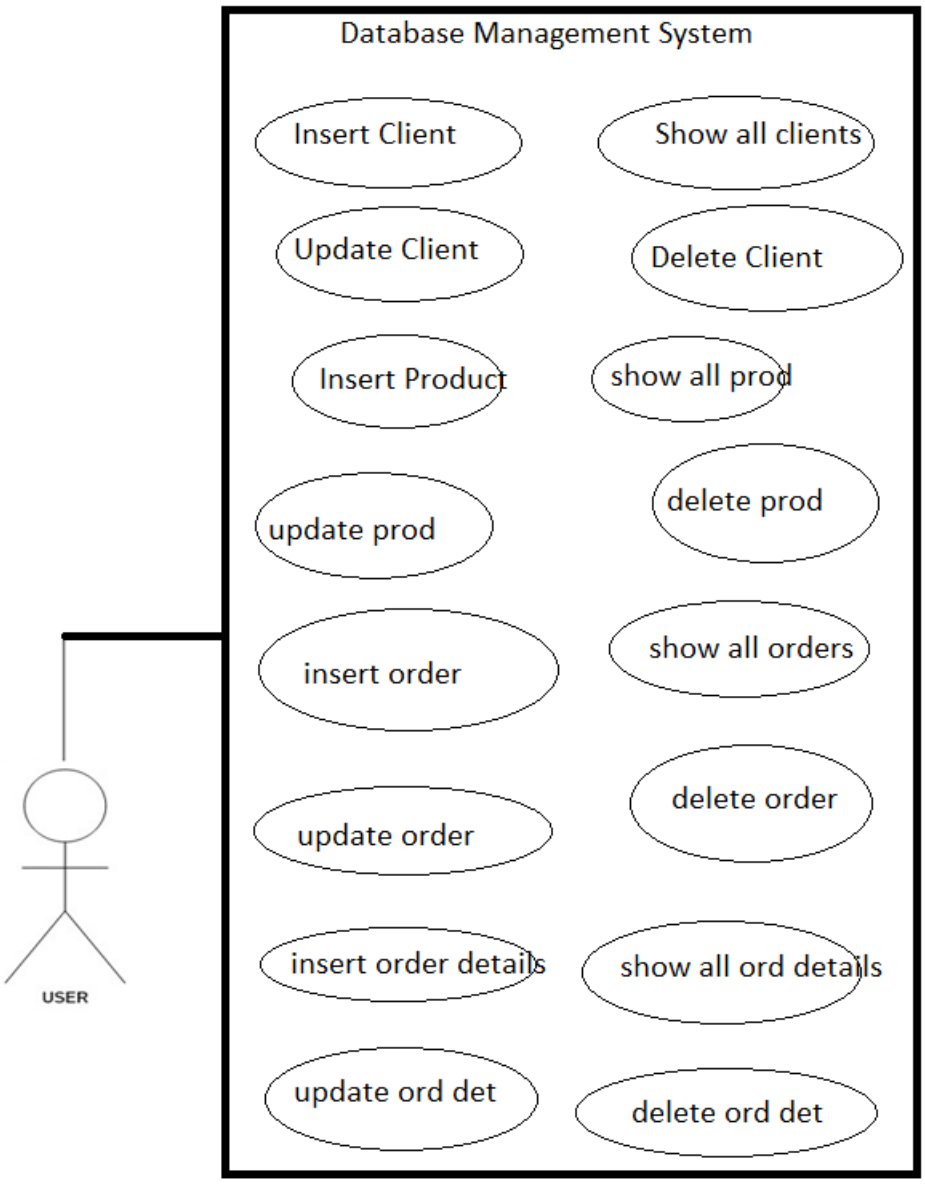
  When talking about the input in the application, the user can choose to manage 4 tables, Client, Product, Order and Order Details. All tables have the options defined as the CRUD operations, Create (insert entry) , Read (show all entries), Update

(modify entry) and Delete, and the user can introduce the values in the specific fields for any of the four different tables.

For example, for the Client table the user can introduce an ID for the client (if no id is provided the application will generate an auto-incremented ID), the name of the client, the Email address and the phone number. All fields have to obey the standard rules such as phone number has to have 10 digits (only digits), the name of the client can contain only letters and spaces, the email address has to contain the @ and the . symbols .

For the following tables, the input is similar but with different fields.

- Use Cases



a) Title: Add Client

Resume: The user can add a new Client by introducing the client details such as id, name, email address and phone number in the specified fields and then he/she should press the Insert button. If the client is created successfully, a pop-up will appear and say that the client was created, and in case of error another pop-up will state that there was an error somewhere.

Actors: User

b) Title: Delete Client

Resume: The user can remove a client by specifying the client ID in the dedicated field and then pressing the Delete button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

c) Title: Update Client

Resume: The user can edit the details of an existing client. He/She should fill all the fields representing client details with the new values and press the Update button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

d) Title: Add Product

Resume: The user can add products in order to be bought by the clients, the products have to be specified via Id, name, price and stock, meaning the total amount of pieces of that particular product. Each order decreases the stock because people buy those products from our market. Any cancelled/ deleted order will “put back” the amount of products involved in that order. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

e) Title: Delete Product

Resume: The user can remove a product by introducing the ID of the product he/she wants to delete and then press the delete button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

f) Title: Update Product

Resume: The user can edit the details of an existing product. He/She has to input all the details of the product he/she wants to modify and then press the Update button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

g) Title: Show Product

Resume: The user can display all the products pressing the Show All button. If a product is modified/ created/ deleted and the user wants to see that modification he/she has to press the Show All button again.

Actors: User

h) Title: Show Client

Resume: The user can display all the clients pressing the Show All button. If a client is modified/ created/ deleted and the user wants to see that modification he/she has to press the Show All button again.

Actors: User

i) Title: Show Order

Resume: The user can display all the order pressing the Show All button. If an order is modified/ created/ deleted and the user wants to see that modification he/she has to press the Show All button again.

Actors: User

j) Title: Insert Order

Resume: The user can insert a new order by specifying the id of the order, selecting the client id from the list of clients, selecting the product id and introducing the quantity that will be bought by the client. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

k) Title: Delete Order

Resume: The user can remove an order by introducing the ID of the order he/she wants to delete and then press the delete button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

l) Title: Update Order

Resume: The user can edit the details of an existing order. He/She has to input all the details of the order he/she wants to modify and then press the Update button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

m) Title: Insert OrderDetails

Resume: The user can insert a new order detail by specifying all the fields of the detail, such as the id, the order id , the delivery city and the delivery street of the client. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

n) Title: Delete OrderDetails

Resume: The user can remove an order detail by introducing the ID of the order detail he/she wants to delete and then press the delete button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

Actors: User

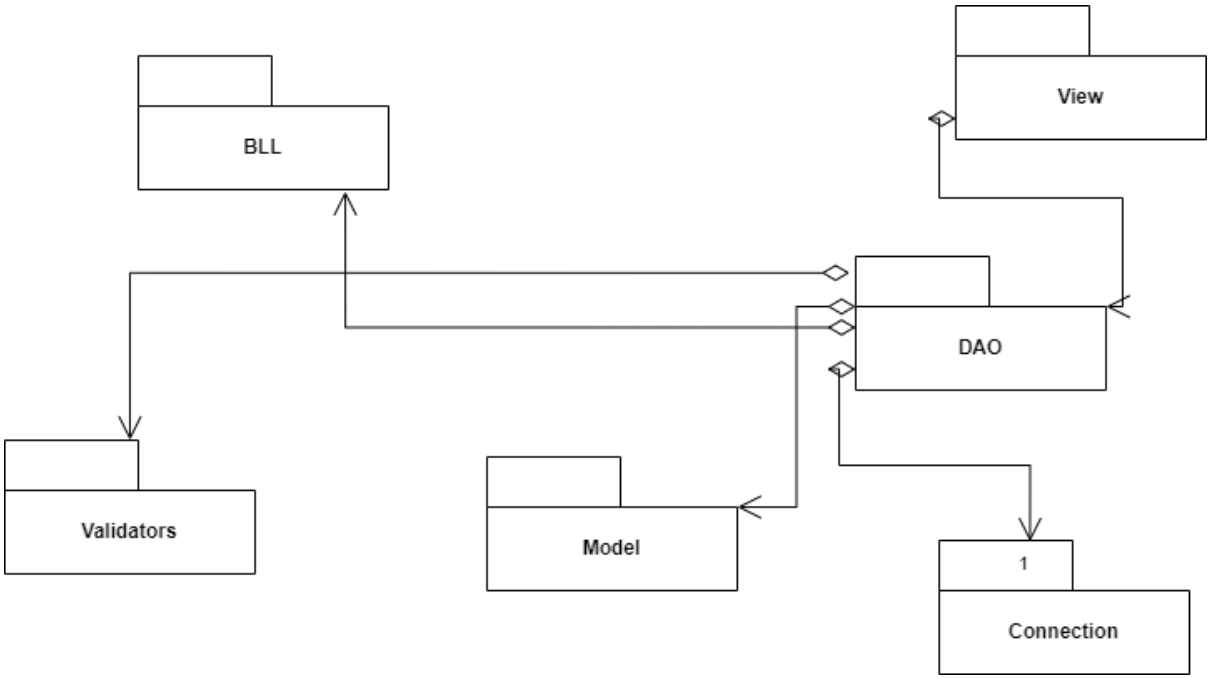
o) Title: Update OrderDetails

Resume: The user can edit the details of an existing order detail. He/She has to input all the details of the order detail he/she wants to modify and then press the Update button. If the operation was successful a pop-up will appear and state that. In case of error another pop-up will state the error.

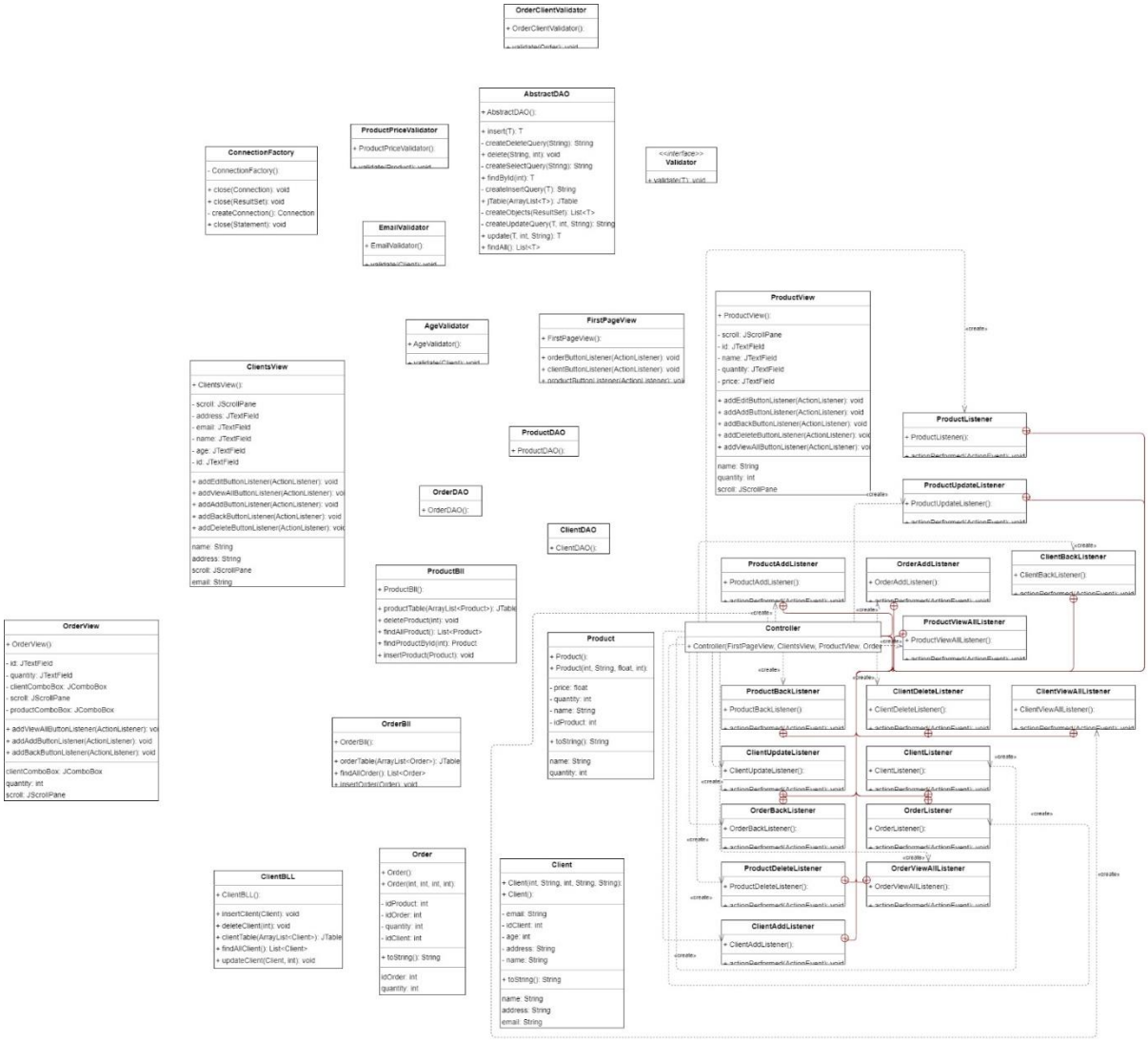
Actors: User

3) Implementation

- Package Diagram



- Class Diagram



- Data Structures

ArrayList – I chose to use this structure to keep my queues and to temporary hold my clients, because this kind of lists is useful for storing and accessing data ArrayList internally uses **dynamic array** to store the elements. Manipulation with ArrayList is slow because it internally uses array. If any element is removed from the array, all the bits are shifted in memory. This class can **act as a list** only because it implements List only.

- Packages

Java packages help in organizing multiple modules and group together related classes and interfaces.

In object-oriented programming development, model-view-controller (MVC) is the name of a methodology or design pattern for successfully and efficiently relating the user interface

to underlying data models. The MVC pattern is widely used in program development with programming languages such as Java, Smalltalk, C, and C++.

The MVC pattern has been heralded by many developers as a useful pattern for the reuse of object code and a pattern that allows them to significantly reduce the time it takes to develop applications with user interfaces.

The model-view-controller pattern proposes three main components or objects to be used in software development:

- *Model*, which represents the underlying, logical structure of data in a software application and the high-level class associated with it. This object model does not contain any information about the user interface.

- *View*, which is a collection of classes representing the elements in the user interface (all of the things the user can see and respond to on the screen, such as buttons, display boxes, and so forth)

- *Controller*, which represents the classes connecting the model and the view, and is used to communicate between classes in the model and view.

My project is based on Model – View – Controller Pattern, but I added some packages for the data access component and connection with the database. So my project consists of seven packages :

➢

- Class Design

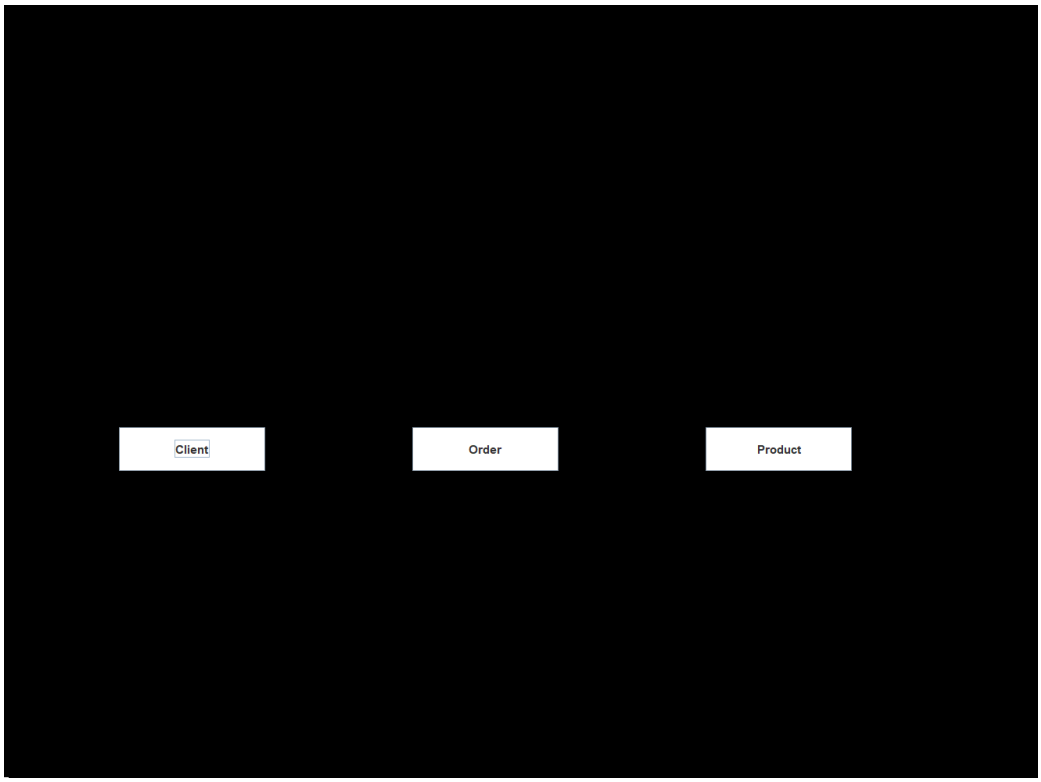
The whole idea of splitting your program into classes is based on a general rule named divide and conquer. This paradigm can be used almost everywhere: you divide a problem into smaller problems and then you solve these little, simple and well-known problems .

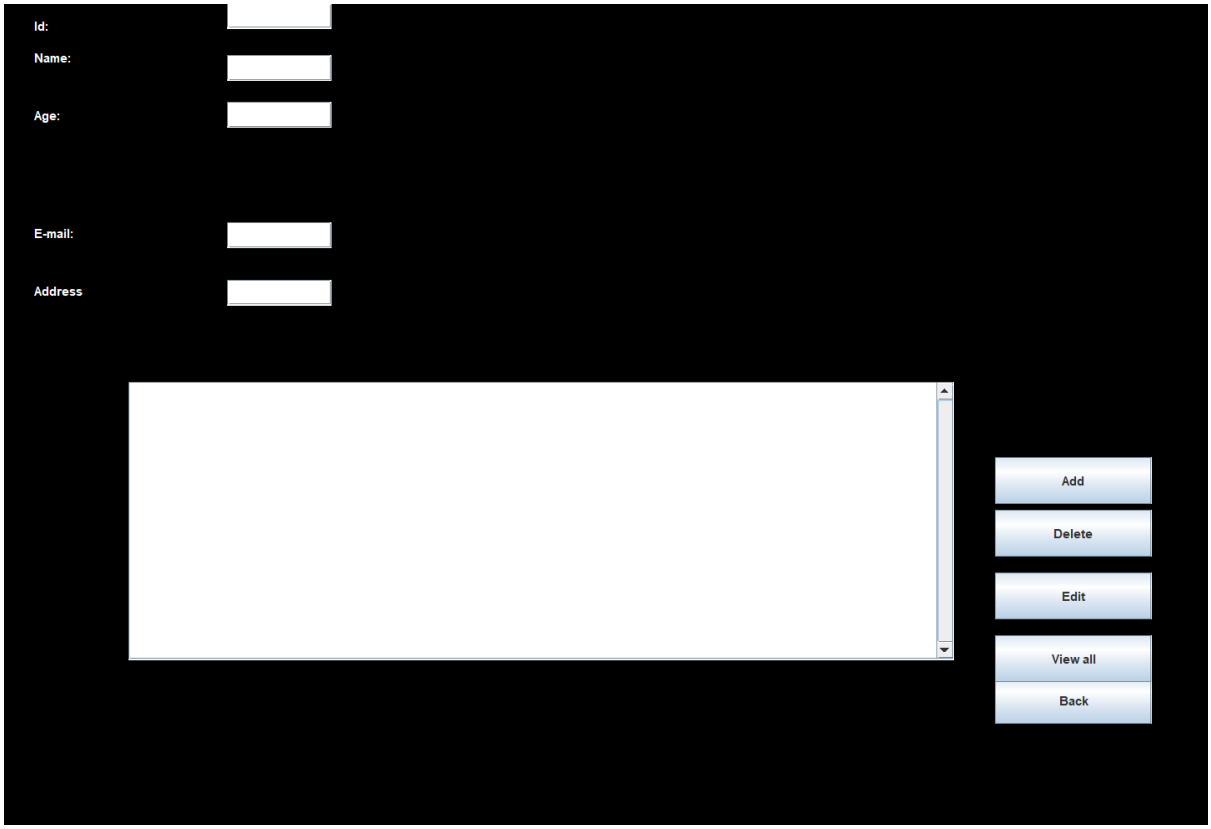
Dividing your program into classes is one of the types of division which started to become common in last decade. In this programming paradigm we model our problem by some objects and try to solve the problem by sending messages between these objects.

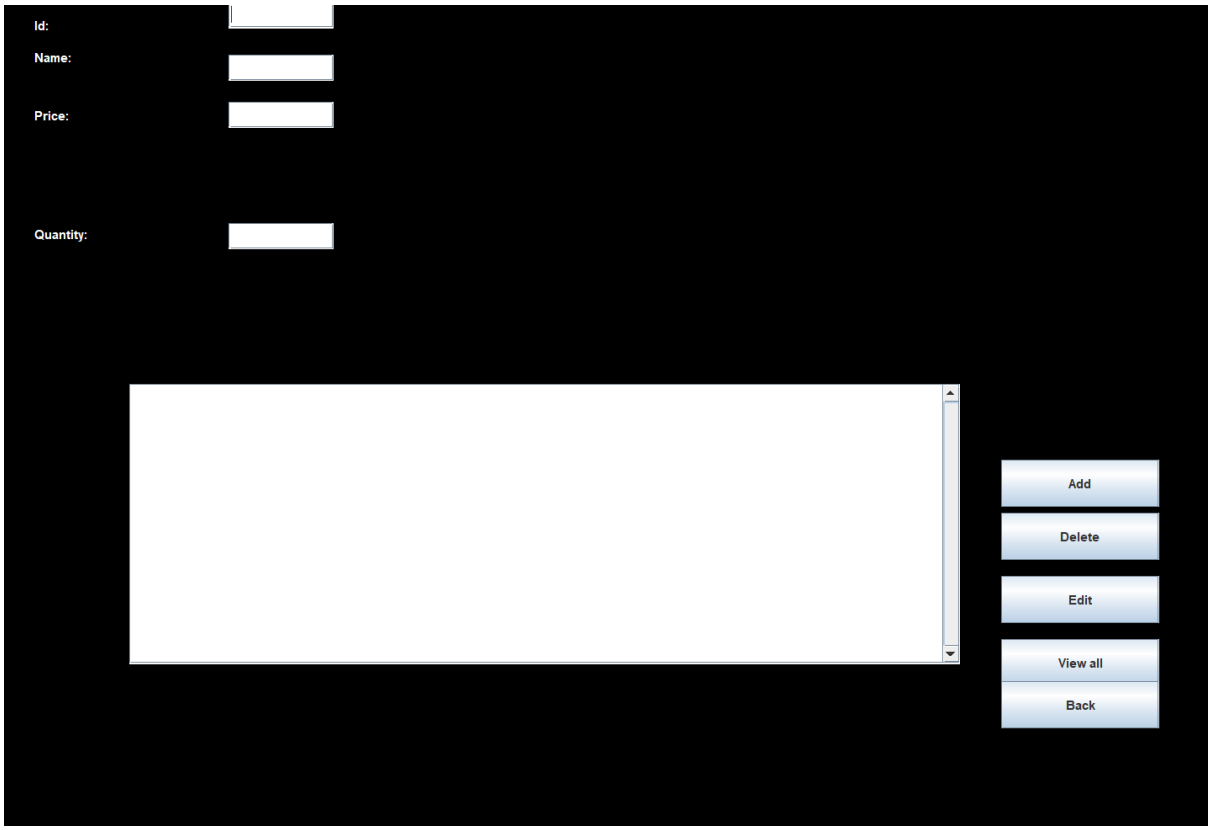
- Application
  - Application – starts the application
- Connection
  - ConnectionFactory- is responsible with the connection with the database, creating a statement. It also has methods to close the connection/ the statement.
- Controller
  - MainController- Initializes the UserInterface buttons and sets up the environment of the application.
- DAO
  - AbstractDAO - contains generic methods to obtain a table from a list of objects, a method to insert in the database a specific object, a method to delete from the database and one to update an existing entry of the database. All the generic methods use reflection techniques.
  - ClientDAO – extends the Generic DAO and uses the generic methods to implement CRUD operations on the Client table
  - OrderDAO– extends the Generic DAO and uses the generic methods to implement CRUD operations on the Order table
  - ProductDAO– extends the Generic DAO and uses the generic methods to implement CRUD operations on the Product table
- Model
  - Client – models the client table from the database
  - Order– models the order table from the database
  - Product– models the product table from the database
- Validators
  - ClientValidator – validates the information introduced in the text fields from the ClientInterface
  - OrderDetailsValidator validates the information introduced in the text fields from the OrderDetailsInterface
  - ProductValidator validates the information introduced in the text fields from the ProductInterface
- View
  - ClientInterface – Client frame, provides buttons, labels, text fields to interact with the clients table
    OrderInterface Order frame, provides buttons, labels, text fields to interact with the order table
  - ProductInterface Product frame, provides buttons, labels, text fields to interact with the product table
  - UserInterface - the main frame which is the “mother” of all the other frames in this application. Provides access to the other frames

4) Graphical User Interface

The interface is a very user friendly one, because all the buttons with which the user interacts say very clearly what they are supposed to do.

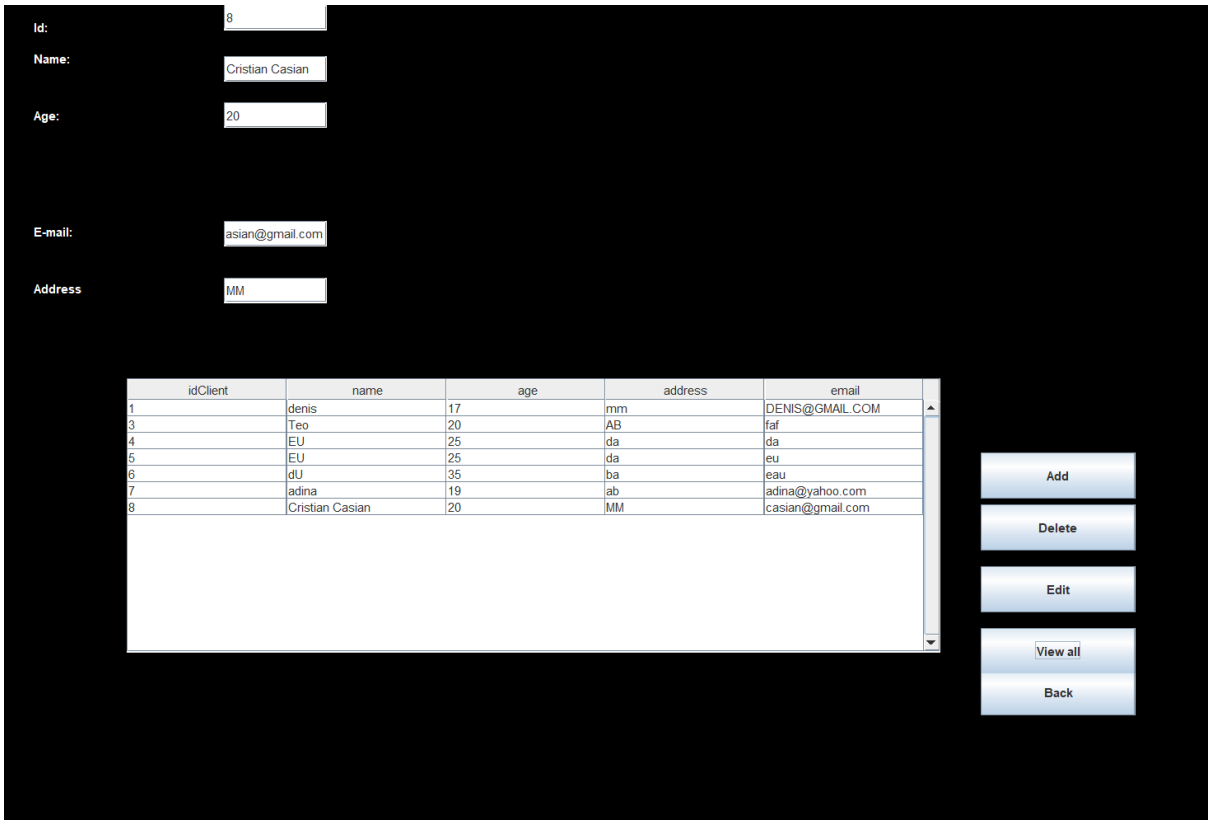



Id:
Name:
Age:
E-mail:
Address
Add
Delete
Edit
View all
Back


Id:
Name:
Price:
Quantity:
Add
Delete
Edit
View all
Back


Id:
idClient:
1
idProduct:
1
Quantity:
Add
View all
Back

5) Results


Id:
8
Name:
Cristian Casian
Age:
20
E-mail:
asian@gmail.com
Address
MM
idClient name age address email
1 denis 17 mm DENIS@GMAIL.COM
3 Teo 20 AB faf
4 EU 25 da da
5 EU 25 da eu
6 dU 35 ba eau
7 adina 19 ab adina@yahoo.com
8 Cristian Casian 20 MM casian@gmail.com
Add
Delete
Edit
View all
Back

6) Conclusions

This project was a good exercise in remembering the OOP concepts learned in the first semester, but also learning new ones, I found it very useful and challenging at first. There are a few learned things which I would present next.

First of all , time management is very, very, very, crucial, because a good organizational spirit helps you see things gradually and making things from time helps you a lot.

Secondly, modelling the problem in a right way from the beginning helps you to implement it faster.

Thirdly, I arrived at the conclusion that facing problems with your code and trying to make it work by yourself, through the mean of research, has the benefit of learning new concepts and a better use of the known ones. In the end, one of the most important things that I have learnt is to make my interfaces from code, because last semester I used WindowBuilder, and I realised that creating an interface is not so easy as it seems using WindowBuilder. After this project I can say that my interface building skills are up to date.

By the means of this project I managed to improve my knowledge about queues processing and how they are implemented on a system, I learned about Threads and about synchronization.

7) Bibliography

a. Object-Oriented Programming - Lecture Slides of prof. Marius JOLDOS
b. Programming Techniques – Lectures of prof. Cristina Pop
c. www.stackoverflow.com
d. www.wikipedia.org
e. http://whatis.techtarget.com/definition/model-view-controller-MVC
f. https://www.draw.io/ (for diagrams)