Ian Chelaru
Technical University of Cluj-Napoca
2nd year Computer Science
Programming Techniques 2017/2018

# Assignment 3 – Specification

## 1. Assignment's objective

The main purpose of the assignment is to design and implement an application for processing customer orders for a warehouse. The application is written in java programming language. The integrated development environment (IDE) used is Eclipse.

Secondary objectives:

- create a relational database and use it in order to store the application data;
- implement operations for managing the application data;
- create a friendly graphical user interface (GUI) for a better interaction between the user and application.

## 2. Features

The application manages three types of entities: clients, products and orders. Each entity has its own attributes. The application uses a relation database in order to store all the information regarding the clients, products and orders. The data is not lost when the application is shut down. The user can perform operations on the application data: add, edit or delete a client, or a product, from the database. Others features allow the user to add orders and view the entire content from the database.

When running the application a frame called "Warehouse" appears on the screen (Figure 1). The user can select to operate on the clients, products or orders data (by pressing the corresponding button):

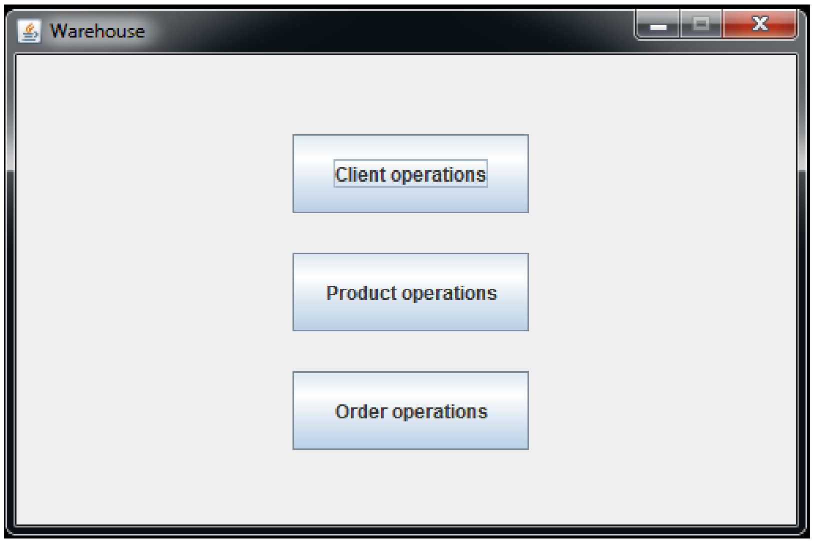


Figure 1 – Warehouse (created by StartView class)

A. Client operations

The possible operations are add, edit, delete a client or view all clients (Figure 2).

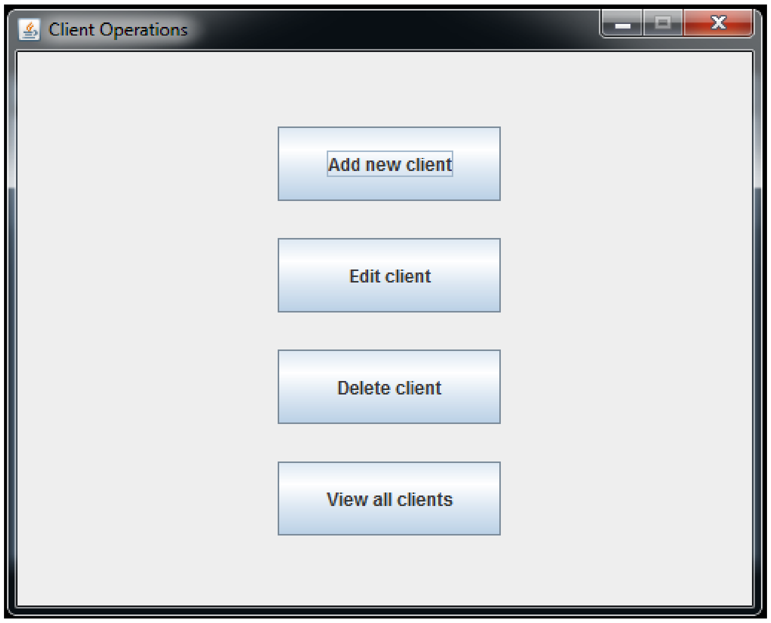


Figure 2 – Client Operations (created by WindowClientOperations class)

In order to add a client, the user has to press the “Add new client” button. A new frame named “Add new client” appears (Figure 3).

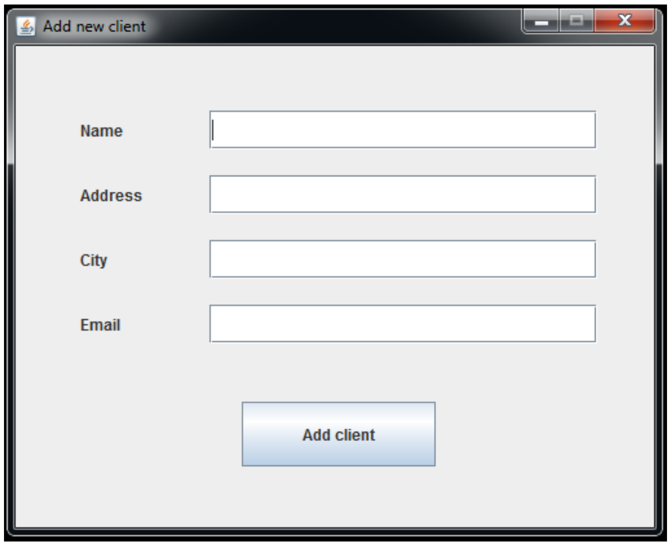


Figure 3 – Add new client (created by AddClientView class)

The client personal data (full name, the address and the city of the domicile, contact email address) must be filled in the corresponding text fields. Some of the client information can miss. By pressing the button "Add client", a new client will be stored in the client table from the database. The client can be identified by its id. The client id is automatically generated and is unique.

If the user wishes to edit a client, he/she will press the "Edit client" button. A new frame called "Edit client" appears (Figure 4).

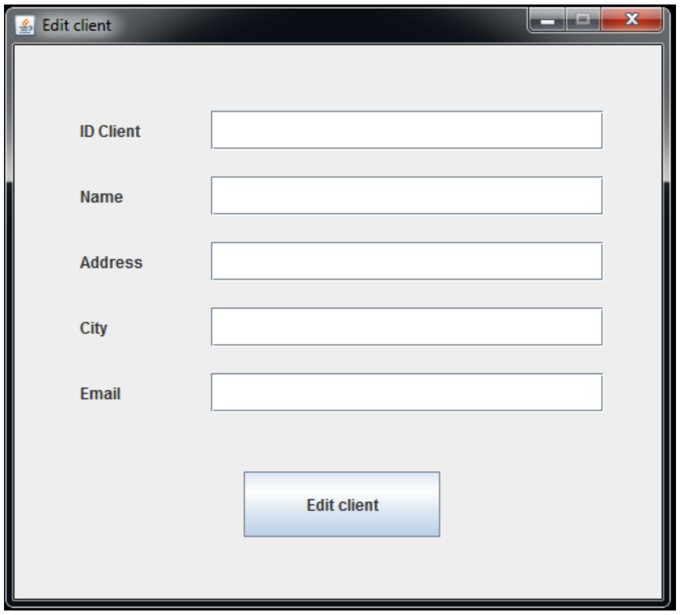


Figure 4 – Edit client (created by EditClientView class)

The user has to introduce the id of the client whose data will be modified. In the other text fields, the new information of the client should be introduced. In order for the user not to introduce data which is not desired to be modified the application lets the user to leave the field empty. If a field is left empty the information of the old client corresponding to that field will not be changed. Finally, the “Edit client” button has to be pressed for the operation to take place.

In order to delete a client, the user must press the “Delete client” button. A new frame named “Delete client” will appears (Figure 5). The user has to introduce the id of the client who will be deleted. By pressing the “Delete client” button, the client with the corresponding id will be permanently deleted from the client table.

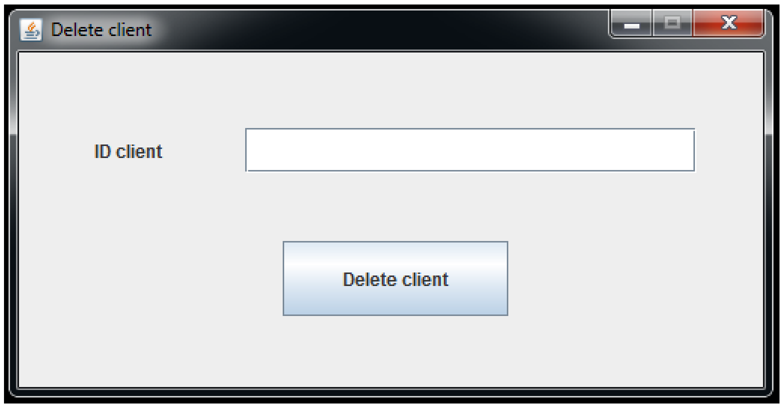


Figure 5 – Delete client (created by DeleteObjectView class)

The user can see the client table by pressing the “View all clients” button. A new frame called “Clients table” will appear and the table with all the clients will be displayed.

Errors:

If the id given by the user when editing a client does not correspond to any client from the table or the id is not specified an error will occur.

When deleting a client, if the id is not specified, an error will occur.

Information regarding the errors will be displayed in the console application. The functionality of the application is not affected. The client table suffers no change.

B. Product operations

The possible operations are add, edit, delete a product or view all products (Figure 6).

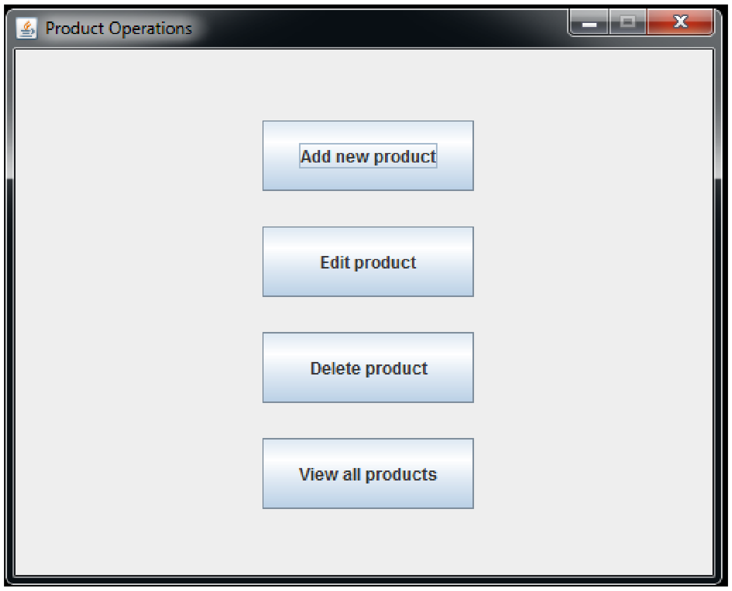


Figure 6 – Product Operations (created by WindowProductOperations class)

In order to add a product, the user has to press the “Add new product” button. A new frame named “Add new product” appears (Figure 7). The product characteristics (a general description of the product, the unit price and the number of products of this type which are available on the warehouse stock) should be filled in the respective fields. By pressing the button “Add product”, a new product will be stored in the product table from the database. The product can be identified by its id. The product id is automatically generated and is unique.

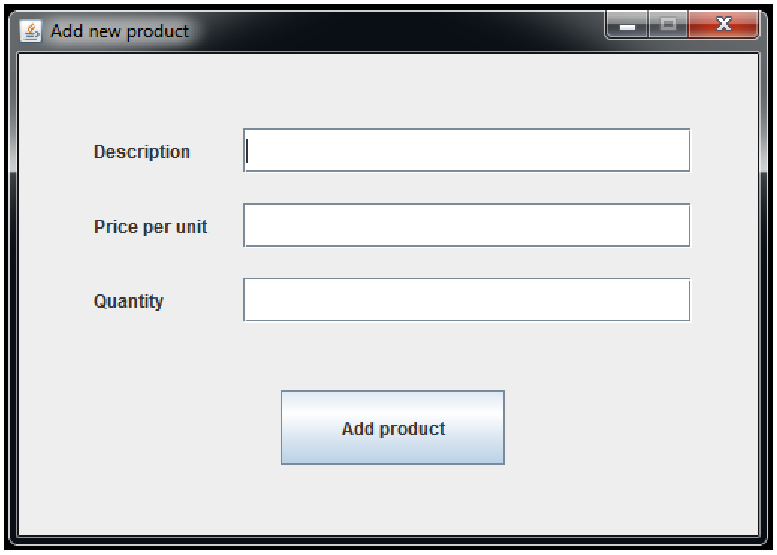


Figure 7 – Add new product (created by AddProductView class)

If the user wishes to edit a product, he/she will press the "Edit product" button. A new frame called "Edit product" appears (Figure 8). The user has to introduce the id of the product whose characteristics will be modified. In the other text fields, the new information of the product should be introduced. In order for the user not to introduce data which is not desired to be modified the application lets the user to leave the field empty. If a field is left empty the information of the old product corresponding to that field will not be changed. Finally, the "Edit product" button has to be pressed for the operation to take place.

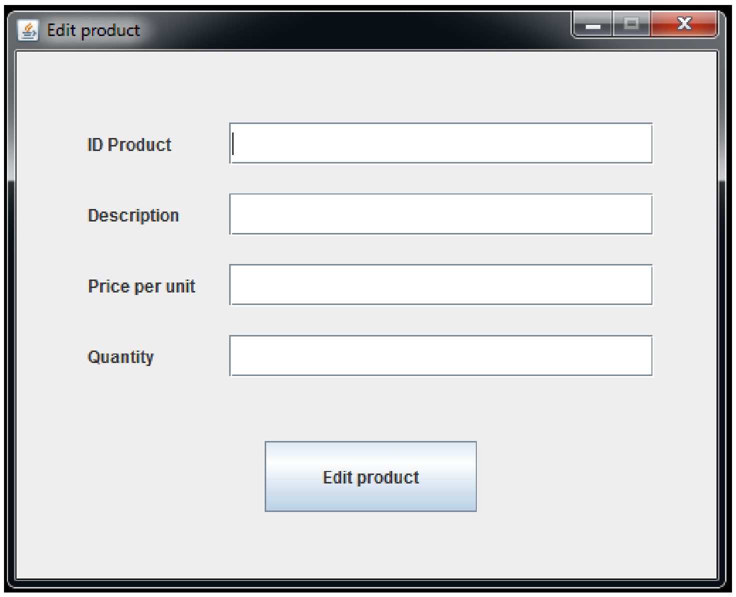


Figure 8 – Edit product (Created by EditProductView class)

In order to delete a product, the user must press the “Delete product” button. A new frame named “Delete product” will appear. The user has to introduce the id of the product which will be deleted. By pressing the “Delete product” button, the product with the corresponding id will be permanently deleted from the product table.

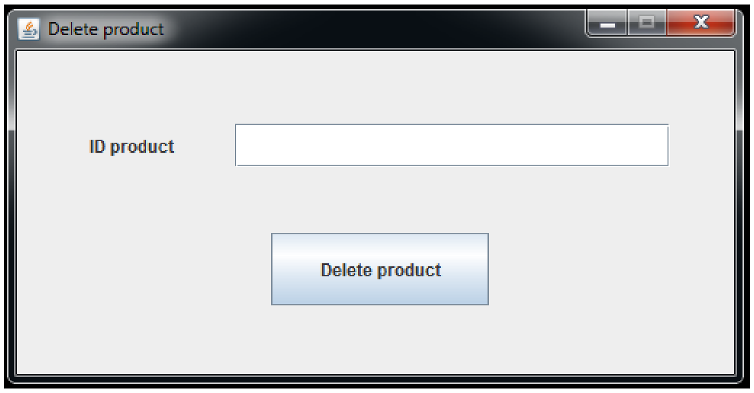


Figure 9 – Delete product (created by DeleteObjectView class)

The user can see the product table by pressing the “View all products” button. A new frame called “Products table” will appear and the table with all the products will be displayed.

Errors:

When adding or editing a product the “Price per unit” value must be a real number whereas the “Quantity” value must be an integer value. Otherwise, an error will occur.

If the id given by the user when editing a product does not correspond to any product from the table or the id is not specified an error will occur.

When deleting a product, if the id is not specified, an error will occur.

Information regarding the errors will be displayed in the console application. The functionality of the application is not affected. The product table suffers no change.

C. Orders operations

The possible operations are to create a product order for a client and to view all the orders made (Figure 10).

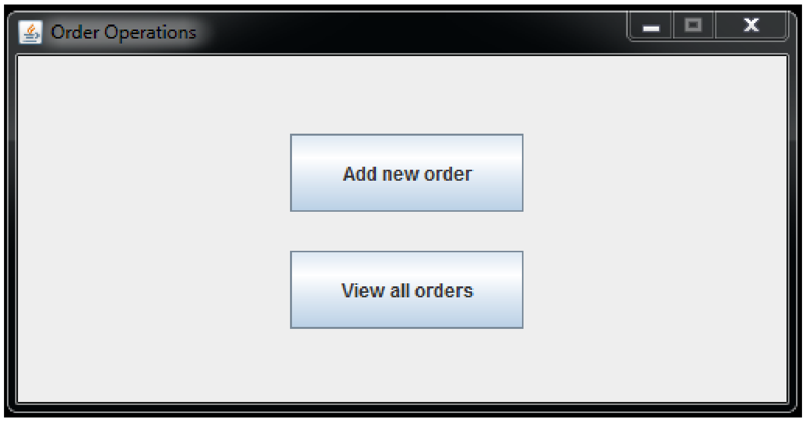


Figure 10 – Order Operations (created by WindowOrderOperations class)

To add an order the user must press the “Add new order” button. A new frame named “Add new order” appears (Figure 11). The user must select the id of the client who made an order. The id of the product and the amount ordered by the client must be specified. By pressing the button “Add order”, a new order will be stored in the order table from the database. The number of products (with the specified id) available on stock is decremented by the amount value. The order can be identified by its id. The order id is automatically generated and is unique.

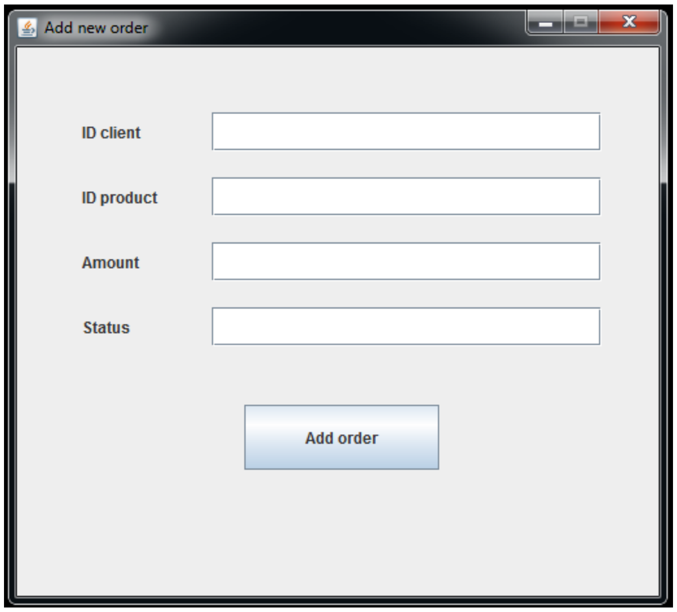


Figure 11 – Add new order (created by AddOrderView class)

The user can see the order table by pressing the “View all orders” button. A new frame called “Orders table” will appear and the table with all the orders will be displayed.

If the amount value is a negative number, then a warning message will be displayed on the frame.

If the amount value is bigger than the number of products on the stock, then a warning message will be displayed.

If the client id or product id does not correspond to any client, respective product, then a warning message will be displayed.

Errors:

If the values of the id client, id product and amount are not integer numbers, an error will occur.

Information regarding the errors will be displayed in the console application. The functionality of the application is not affected. The order table suffers no change.

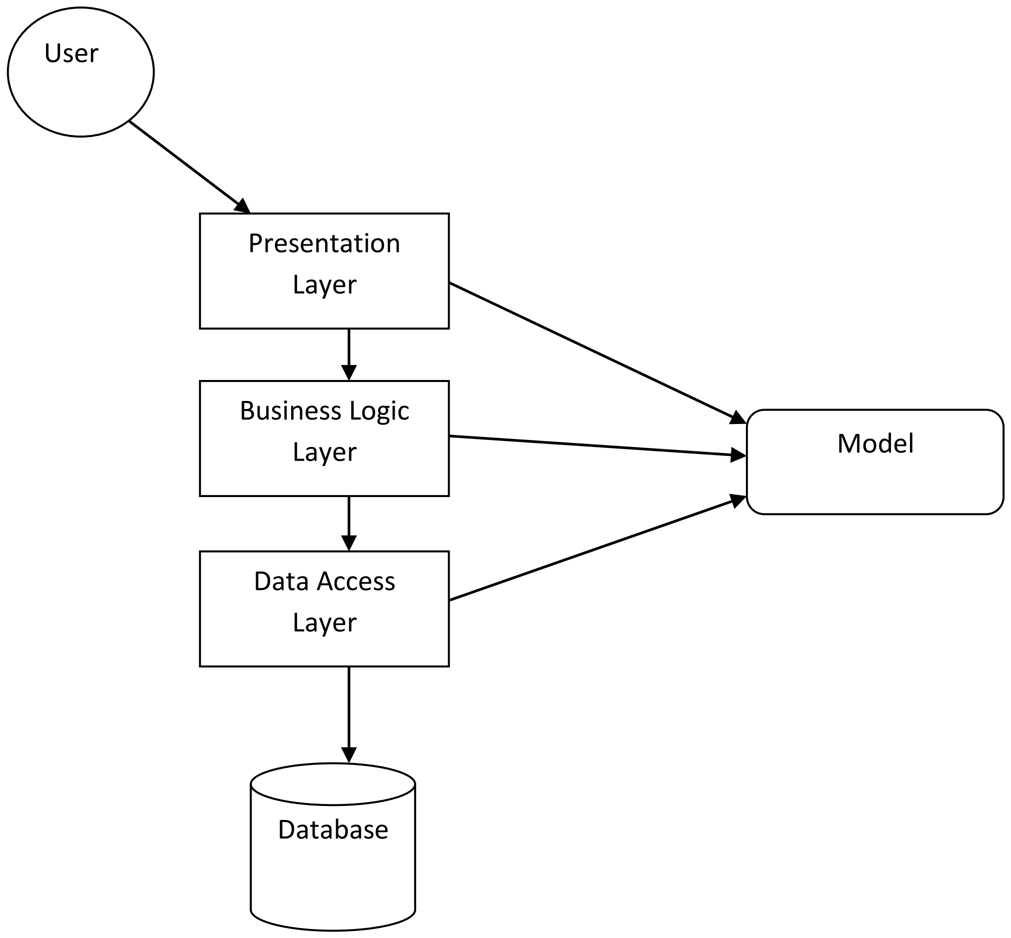
In order to exit the application, the user has to close the “Warehouse” frame.

Assumptions:

It is assumed that the input data given by the user is a valid one such that none of the above errors will occur.

## 3. Design

In order to design the application, a layered architecture is used. The application is split into three different layers (3-Tier Architecture): presentation layer, business logic layer and data access layer. Each layer has a special purpose and has access only to the layer below it.



The presentation layer is split into two parts: view and controller. The view contains the classes which implement the GUI. The purpose of the view is to take the input given by the user and to display the application outputs. The controller contains the main controller class which has the purpose to take the input from the GUI, call the business logic classes which compute the results and send the results to the GUI.

The business logic layer contains the classes which implement the business logic of the application. It receives information from the presentation layer and sends it to the data access layer.

The data access layer has the purpose to communicate with the database. The classes from this layer create the connection with the database, create and execute the queries and provide information to the upper layers regarding the data from the database.

The layers have access to the model. The model contains the classes which abstracts the objects which create the application domain and carry the data. The classes are mapped to the database tables.

Each part of the application is implemented in its own package or subpackage.

## UML Diagram

The UML Diagram is presented below (Figure 12).

Remark: there should be an association relation from the MainController class to each class form the view package.

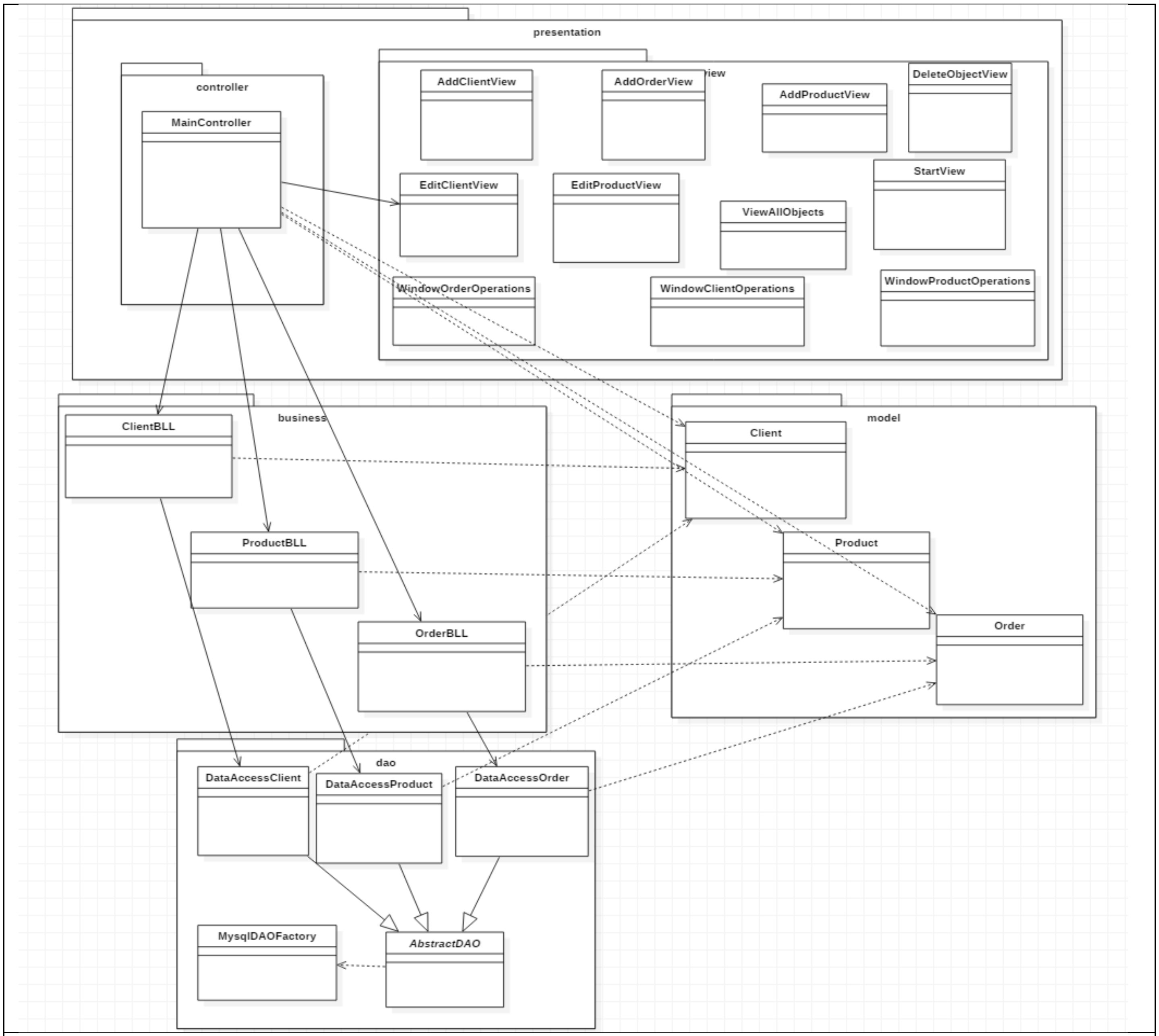
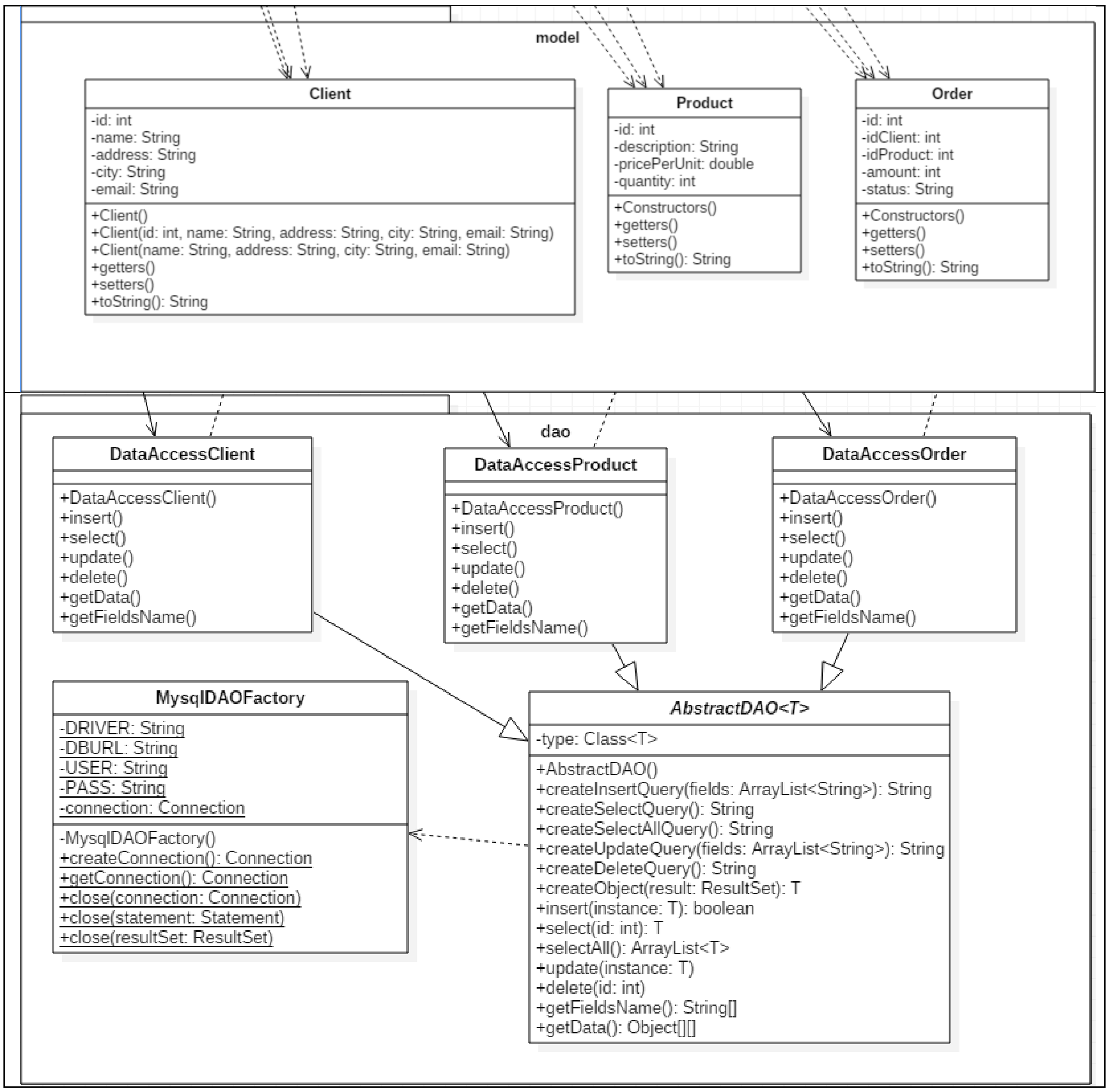
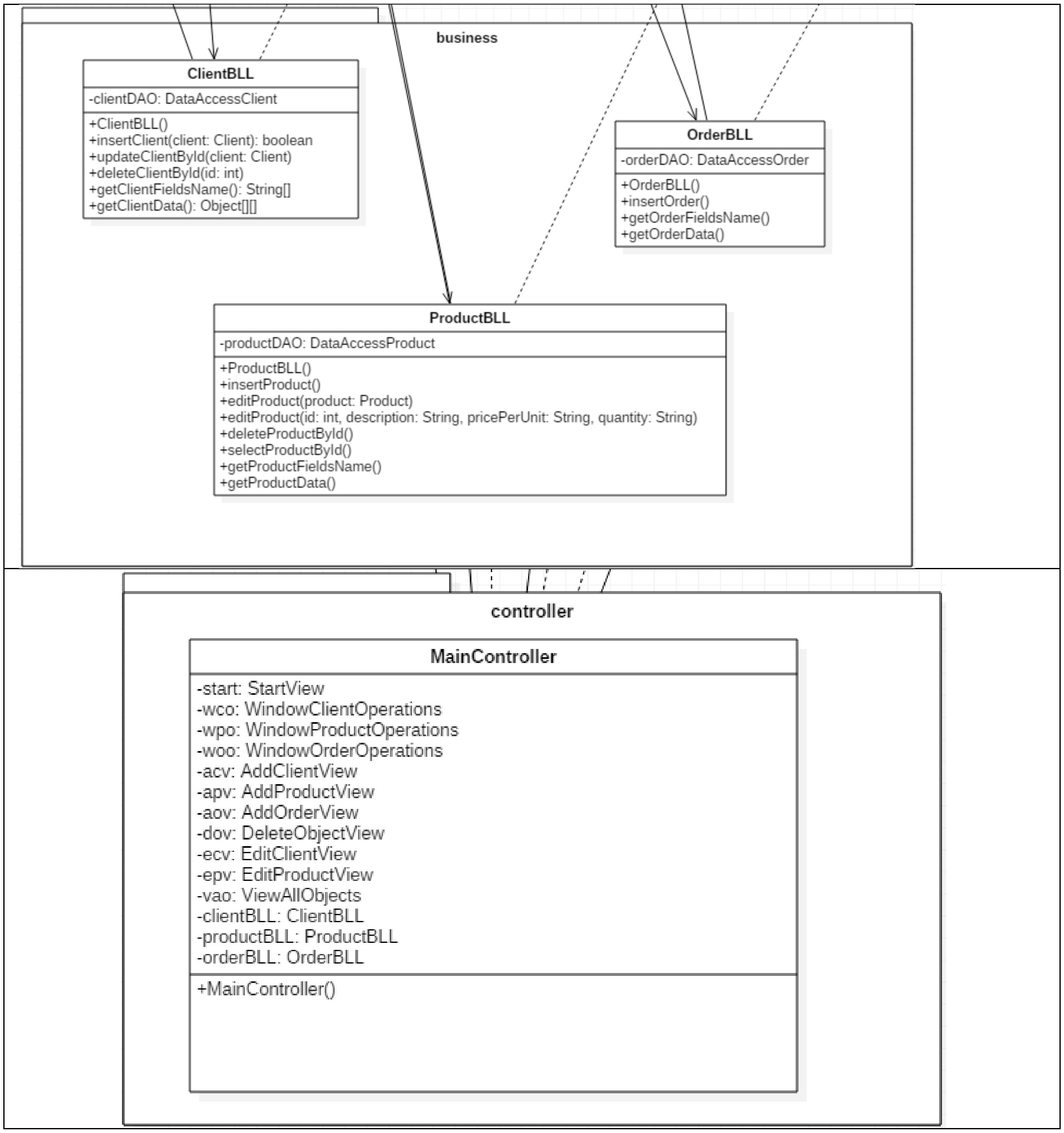


Figure 12 - UML Diagram



model
Client
-id: int
-name: String
-address: String
-city: String
-email: String
+Client()
+Client(id: int, name: String, address: String, city: String, email: String)
+Client(name: String, address: String, city: String, email: String)
+getters()
+setters()
+toString(): String
Product
-id: int
-description: String
-pricePerUnit: double
-quantity: int
+Constructors()
+getters()
+setters()
+toString(): String
Order
-id: int
-idClient: int
-idProduct: int
-amount: int
-status: String
+Constructors()
+getters()
+setters()
+toString(): String
dao
DataAccessClient
+DataAccessClient()
+insert()
+select()
+update()
+delete()
+getData()
+getFieldsName()
DataAccessProduct
+DataAccessProduct()
+insert()
+select()
+update()
+delete()
+getData()
+getFieldsName()
DataAccessOrder
+DataAccessOrder()
+insert()
+select()
+update()
+delete()
+getData()
+getFieldsName()
MysqlDAOFactory
-DRIVER: String
-DBURL: String
-USER: String
-PASS: String
-connection: Connection
-MysqlDAOFactory()
+createConnection(): Connection
+getConnection(): Connection
+close(connection: Connection)
+close(statement: Statement)
+close(resultSet: ResultSet)
AbstractDAO<T>
-type: Class<T>
+AbstractDAO()
+createInsertQuery(fields: ArrayList<String>): String
+createSelectQuery(): String
+createSelectAllQuery(): String
+createUpdateQuery(fields: ArrayList<String>): String
+createDeleteQuery(): String
+createObject(result: ResultSet): T
+insert(instance: T): boolean
+select(id: int): T
+selectAll(): ArrayList<T>
+update(instance: T)
+delete(id: int)
+getFieldsName(): String[]
+getData(): Object[][]


business
ClientBLL
-clientDAO: DataAccessClient
+ClientBLL()
+insertClient(client: Client): boolean
+updateClientById(client: Client)
+deleteClientById(id: int)
+getClientFieldsName(): String[]
+getClientData(): Object[][]
OrderBLL
-orderDAO: DataAccessOrder
+OrderBLL()
+insertOrder()
+getOrderFieldsName()
+getOrderData()
ProductBLL
-productDAO: DataAccessProduct
+ProductBLL()
+insertProduct()
+editProduct(product: Product)
+editProduct(id: int, description: String, pricePerUnit: String, quantity: String)
+deleteProductById()
+selectProductById()
+getProductFieldsName()
+getProductData()
controller
MainController
-start: StartView
-wco: WindowClientOperations
-wpo: WindowProductOperations
-woo: WindowOrderOperations
-acv: AddClientView
-apv: AddProductView
-aov: AddOrderView
-dov: DeleteObjectView
-ecv: EditClientView
-epv: EditProductView
-vao: ViewAllObjects
-clientBLL: ClientBLL
-productBLL: ProductBLL
-orderBLL: OrderBLL
+MainController()

# 4. Implementation

## GUI implementation

The GUI was implemented as a succession of frames. Each frame has a special purpose. There are eleven classes which implements the GUI (each one extends JFrame): AddClientView, AddOrderView, AddProductView, DeleteObjectView, EditClientView, EditProductView, StartView, ViewAllObjects, WindowClientOperations, WindowOrderOperations and WindowProductOperations.

The StartView class has the purpose to show to the user the main application features: operate on clients, products or orders. The attributes of the class are three buttons which open other frames (WindowClientOperations, WindowOrderOperations, WindowProductOperations). The class has a constructor with no parameter which initializes the frame and a method for each button which adds an action listener for the respective button.

The WindowClientOperations class has the purpose to present to the user the actual client operations which can be done: add, edit and delete a client and view all clients. The attributes of the class are four buttons which open other frames where the operations are set (AddClientView, EditClientView, DeleteObjectView, ViewAllObjects). The class has a constructor with no parameter which initializes the frame and method for each button which adds an action listener for the respective button.

The WindowProductOperations and the WindowOrderOperations classes have the same purpose and implementations like WindowClientOperations, but for their corresponding objects: products, respectively orders.

The classes which create frames where the add, edit and delete operations are set (AddClientView, AddOrderView, AddProductView, EditClientView, EditProductView, DeleteObjectView) have the same implementation. As the attributes, they have some text fields, where the user will write the inputs, and a button which performs the corresponding operation (the name of the classes are suggestive). Each class has a constructor which initializes the frame, getters for the text fields (in order for the controller to access the input) and one method which adds an action listener for the button.

The ViewAllObjects class has the purpose to display the tables from the database. The column names and the rows of data are received as parameters by the constructor. In this way, the class is used to display any kind of objects: clients, products or orders.

## 5. Conclusions

What I learn from the assignment:

- how to connect a java application to a database;
- how to generate and execute queries;
- what is and how to use the reflection technique;
- design and implement an application using a three–tier architecture;
- how to create and use a JTable.

Future developments:

- better field validation (for instance check if a city given by the user is an existing one);
- possibility to edit the status of an order;
- more information regarding the data (for example the client may have a phone number);
- other filters: possibility to select an object by other properties than id or view all the objects with a certain property.