Technical University of Cluj-Napoca

Programming techniques

Laboratory assignment 2



**Orders Management Application**

Student: Sicobean Alexandra Maria

Group: 30421

1. **Objective**

The objective of the assignment is to design and implement an application for managing the client orders for a warehouse.

The sub-objectives of the assignment are:

* Analyzing the problem and identifying the requirements
* Designing the orders management application
* Implementing the orders management application
* Testing the orders management application

1. **Analysis, scenarios, use cases**

**Functional requirements:**

* The application should allow an employee to add a new client
* The application should allow an employee to add a new product
* The application should allow an employee to add a new order
* The application should allow an employee to edit a new client
* The application should allow an employee to edit a new product
* The application should allow an employee to edit a new order
* The application should allow an employee to delete a new client
* The application should allow an employee to delete a new product
* The application should allow an employee to delete a new order

**Non-functional requirements:**

* The application should be intuitive and easy to use

**Scenarios and use cases:**

**Use Case:** add client

**Primary Actor:** employee

**Main success Scenario:**

* The employee selects the Manage clients option
* The employee enters the client’s details
* The employee clicks on the “Add” button
* The application stores the client data in the database

**Use Case:** add product

**Primary Actor:** employee

**Main success Scenario:**

* The employee selects the Manage products option
* The employee enters the product’s details
* The employee clicks on the “Add” button
* The application stores the product data in the database

**Use Case:** add order

**Primary Actor:** employee

**Main success Scenario:**

* The employee selects the Manage orders option
* The employee enters the orders’s details
* The employee clicks on the “Place order” button
* The application stores the order data in the database

**Use Case:** edit client

**Primary Actor:** employee

**Main success Scenario:**

* The employee selects the Manage clients option
* The employee enters the client’s new details and id
* The employee clicks on the “Edit” button
* The application stores the new client data in the database

**Use Case:** edit product

**Primary Actor:** employee

**Main success Scenario:**

* The employee selects the Manage products option
* The employee enters the product’s new details and id
* The employee clicks on the “Edit” button
* The application stores the new product data in the database

**Use Case:** delete product

**Primary Actor:** employee

**Main success Scenario:**

* The employee selects the Manage products option
* The employee enters the product’s id
* The employee clicks on the “Delete” button
* The application deletes the product from the database

**Use Case:** generate bill

**Primary Actor:** employee

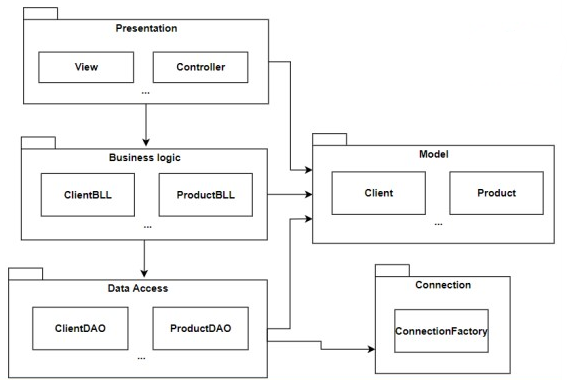
**Main success Scenario:**

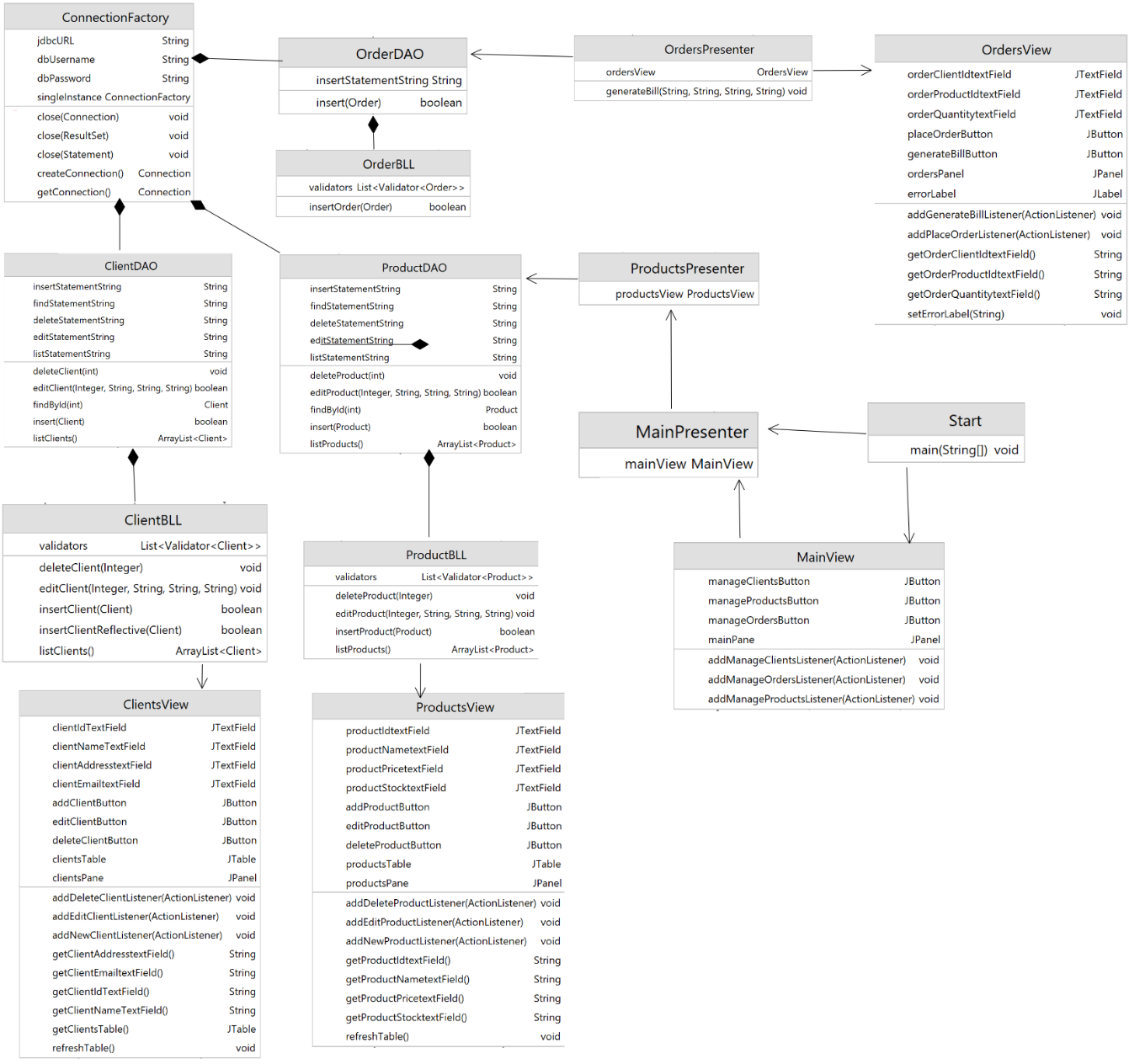
* The employee selects the Manage orders option
* The employee enters the order’s details
* The employee clicks on the “Generate bill” button
* The application generates a .txt file with the bill’s details

1. **Design**

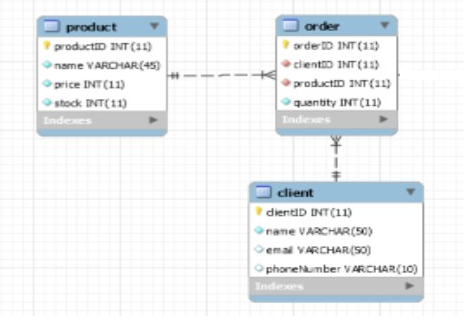
The structure of the project is similar to the one described in the assignment support materials. It closely follows the layered structure as it can be seen below:

**UML package diagram:**

****

**UML class diagram:**

**Database diagram:**

****

1. **Implementation**

**Classes:**

**BLL package:**

**ClientBLL**: This class is part of the business logic and verifies the validity of a client data before introducing it into the database

insertClient: This method verifies if the email is correct and, if yes, it introduces a new element into the database

deleteClient: This method deletes a client from the database based on its id

editClient: This method changes the attributes of a client

listClients: This method returns a list of clients in the database

**OrderBLL**: This class is part of the business logic and verifies the validity of an order data before introducing it into the database

insertOrder: This method inserts a new order into the database after verifying that its quantity is greater than 0

**ProductBLL**: This class is part of the business logic and verifies the validity of a product data before introducing it into the database

insertProduct: This method verifies if the price and stock is correct and, if yes, it introduces a new element into the database

deleteProduct: This method deletes a product from the database based on its id

editProduct: This method changes the attributes of a product

listProduct: This method returns a list of products in the database

**Connection package:**

**ConnectionFactory:** This class connects the application to the database

**DataAccess package:**

**ClientDAO:** This is the class that manages the raw actions between the application and the clients table. Here the sql statements are prepared and executed. It has the same methods as ClientBLL only before being verified through validators

**ProductDAO:** This is the class that manages the raw actions between the application and the product table. Here the sql statements are prepared and executed. It has the same methods as ProductBLL only before being verified through validators

**OrderDAO:** This is the class that manages the raw actions between the application and the order table. Here the sql statements are prepared and executed. It has the same methods as OrderBLL only before being verified through validators

**Model package:**

**Client:** This class is the model for the client entity. It contains attributes that will also be contained in the database.

**Product:** This class is the model for the product entity. It contains attributes that will also be contained in the database.

**Order:** This class is the model for the order entity. It contains attributes that will also be contained in the database.

**Presentation package:**

**ClientsPresenter:** This class is a presenter that connects the Client model class to the ClientsView and is used for any action that influences the management of clients.

**OrdersPresenter:** This class is a presenter that connects the Orders model class to the OrdersView and is used for any action that influences the management of clients.

**ProductsPresenter:** This class is a presenter that connects the Product model class to the ProductsView and is used for any action that influences the management of clients.

**MainPresenter**: This class is the main presenter. It links all the other presenters and views to the main one

**ClientsView:** This class is the view for managing the clients. It contains text fields for id, name, address and emails as well as buttons for adding, editing and deleting clients and a clients table.

**ProductsView**: This class is the view for managing the products. It contains text fields for id, name, price and stock as well as buttons for adding, editing and deleting products and a products table.

**OrdersView**: This class is the view for managing the orders. It contains text fields for clientids, productids and quantities as well as buttons for placing orders and generating bills.

**MainView**: This view is the main view. It is the first panel that is displayed and the user's first contact with the application. Through this view the user will choose the entity he wants to manage

**Validators package:**

**EmailValidator:** This class validates the given mail by using an email pattern

Validate: This is the method used to validate the given email

**QuantityValidator**: This class verifies the quantity provided by the user for an order

Validate: This method verifies that the quantity is a positive number

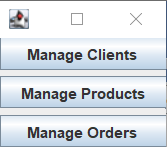
**StockValidator**: This class verifies the stock and price introduced by the user for a product

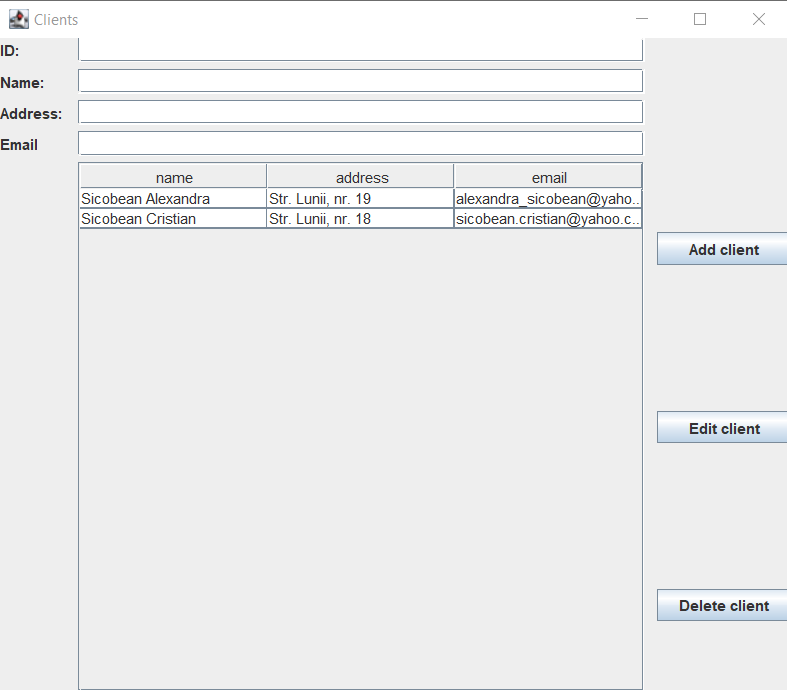
Validate: This method verifies if both the stock and the price of a product are greater than 0

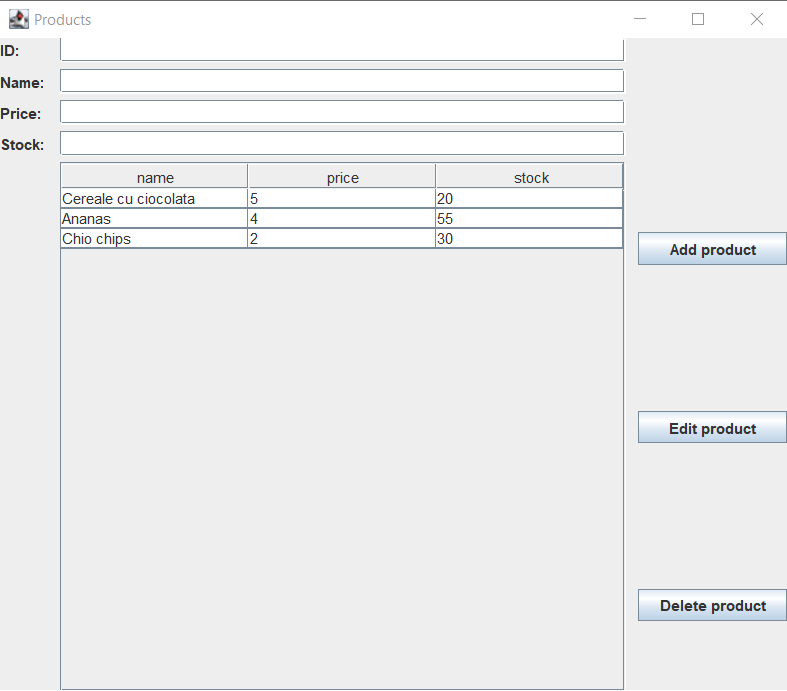
**Validator**: This interface is used to implement the email, quantity and stock validators

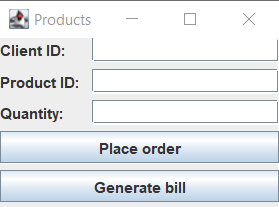
Validate: \* The method is implemented for each of the validators depending on the given parameter

**UI**:



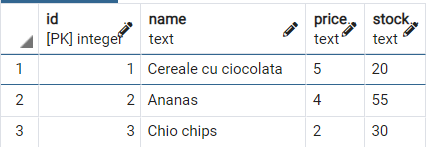




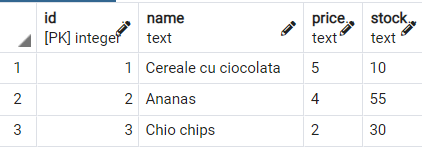


1. **Results**

Initial data:



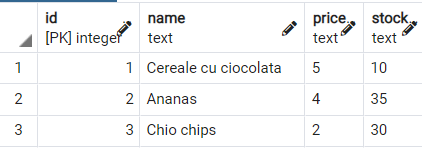
After placing an order from the client with id 2 for 10 Cereale cu ciocolata:



Bill:

Name: Sicobean Cristian  
Product: Cereale cu ciocolata  
Quantity: 10  
Final price: 50

After placing an order from the client with id 1 for 20 Ananas:



Bill:

Name: Sicobean Alexandra  
Product: Ananas  
Quantity: 20  
Final price: 80

1. **Conclusions**

The multitude of classes, packages and layers made this assignment harder to implement than last assignments. However, thanks to reflective techniques many hours of work were saved.

1. **Bibliography**

- http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html

-http://www.tutorialspoint.com/java/util/timer\_schedule\_period.htm

- http://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-andthreadpoolexecutor.html