

Documentation

Assignment 3

Bodogae George Stefan
Group: 30424

CUPRINS

1.	Obiectivul temei.....	Error! Bookmark not defined.
2.	Analiza problemei, modelare, scenarii, cazuri de utilizare	3
3.	Proiectare.....	Error! Bookmark not defined.
4.	Implementare	Error! Bookmark not defined.
5.	Rezultate.....	Error! Bookmark not defined.
6.	Concluzii	Error! Bookmark not defined.
7.	Bibliografie.....	Error! Bookmark not defined.

1. Assignment's objective

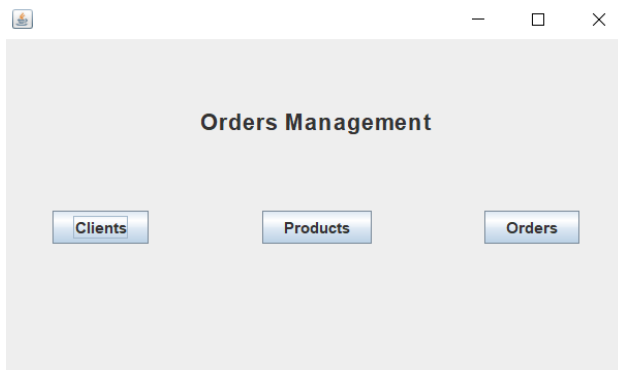
This assignment's objective is creating an order management app in java which interacts with an SQL database for storing and retrieving data. An order management app is really useful when dealing with the problems which occur when managing any kind of business that has a product. It allows the employees to easily keep track of the clients and client data, their product range and stock management and also of the orders which can be overwhelming if there is no organized system to manage them. The application is also useful for generating bills automatically from the database without having to manually enter each of the details.

2. Analisis, modeling, use cases

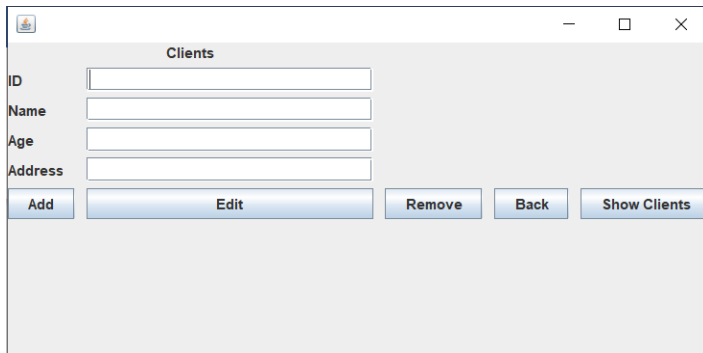
I started programing this application with ease of use in mind. The app has 4 windows which can be accessed by the user. The first one is the main window which is the starting point of the application. From here, the user can access the clients window, the products window and the orders window. The clients window has 4 formatted text fields in which the user can insert the ID, name, age and address of a new client. Then, there are 5 buttons for adding a new client, editing an existing client, removing an existing client, going back to the main window and showing a separate window that contains a table filled with all the clients. The second option the user has from the main window is accessing the product window. There are 4 formatted text fields where the user can input the ID, name, stock and price of a product and then there are 5 buttons for adding a new product, editing an existing product, removing an existing product, going back to the main window and opening a separate window which displays a table that contains all the products available in the database. The last option the user has from the main window is selecting the orders button, which opens the orders window. In this window the user can select a product and a client, the amount of product that the order will contain and set an order ID and then he has the option to make an order, which generates a bill as a text file, and he can also press the "Show Orders" button which displays a window that contains a table filled with all the orders. The "back" button also can be pressed to go back to the main window.

The following images represent the way the user is going to interact with the application, as well as the bill generated when making an order.

The following image represents the main window which has 3 buttons: "Clients", "Products" and "Orders". The user will press the buttons in any order, as the application allows coming back to this window as many times as it's needed.

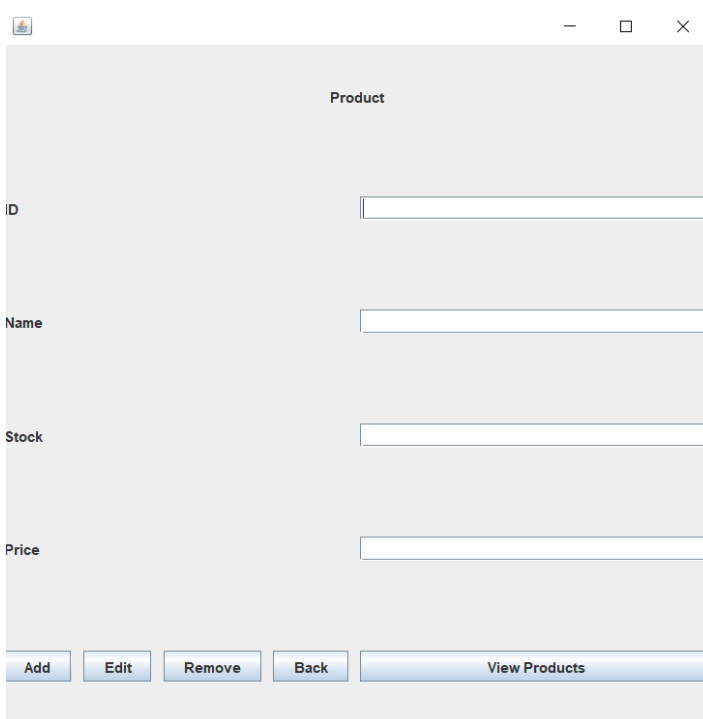


The following image represents the Clients window, which has 4 formatted text fields: "ID", "Name", "Age" and Address. The Clients window also contains 5 buttons: "Add", "Edit", "Remove", "Back" and "Show Clients".



The screenshot shows a window titled "Clients" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains four text input fields labeled "ID", "Name", "Age", and "Address" stacked vertically on the left. To the right of these fields is a large, empty rectangular area. At the bottom of the window, there are five buttons: "Add", "Edit", "Remove", "Back", and "Show Clients", arranged horizontally.

The following image represents the Product window, which has 4 formatted text fields: “ID”, “Name”, “Stock” and Price. The Product window also contains 5 buttons: “Add”, “Edit”, “Remove”, “Back” and “Show Products”.



The screenshot shows a window titled "Product" with a standard Windows-style title bar. The window contains four text input fields labeled "ID", "Name", "Stock", and "Price" stacked vertically on the left. To the right of these fields is a large, empty rectangular area. At the bottom of the window, there are five buttons: "Add", "Edit", "Remove", "Back", and "View Products", arranged horizontally.

The following image represents the orders window. In this window there are 2 combo boxes where the user can select a product and a client based on the ID of that object. There are 2 formatted text fields in which the user will input the amount of product that he/she wants to order and an ID for the order. There are 3 buttons: “Back”, “Order” and “Show Orders”.

The following image represents the table which is created when the user presses the “Show Products” button from the Products window. Similar tables are generated when the user presses “Show Clients” or “Show Orders” buttons from the Clients window or the Orders window.

id	pr_name	stock	price
3	Salata	9	12
5	Slapi	8	12
2	Bere	10	12
1	pita	0	12
4	Gin	2	12

The following image represents a bill generated in a text file when the user presses the “Order” button from the Order window. The text file contains the ID of the order, the ID, name and amount of product ordered, The ID and name of the client that made the order and the total price which is calculate by the formula: product price * amount.

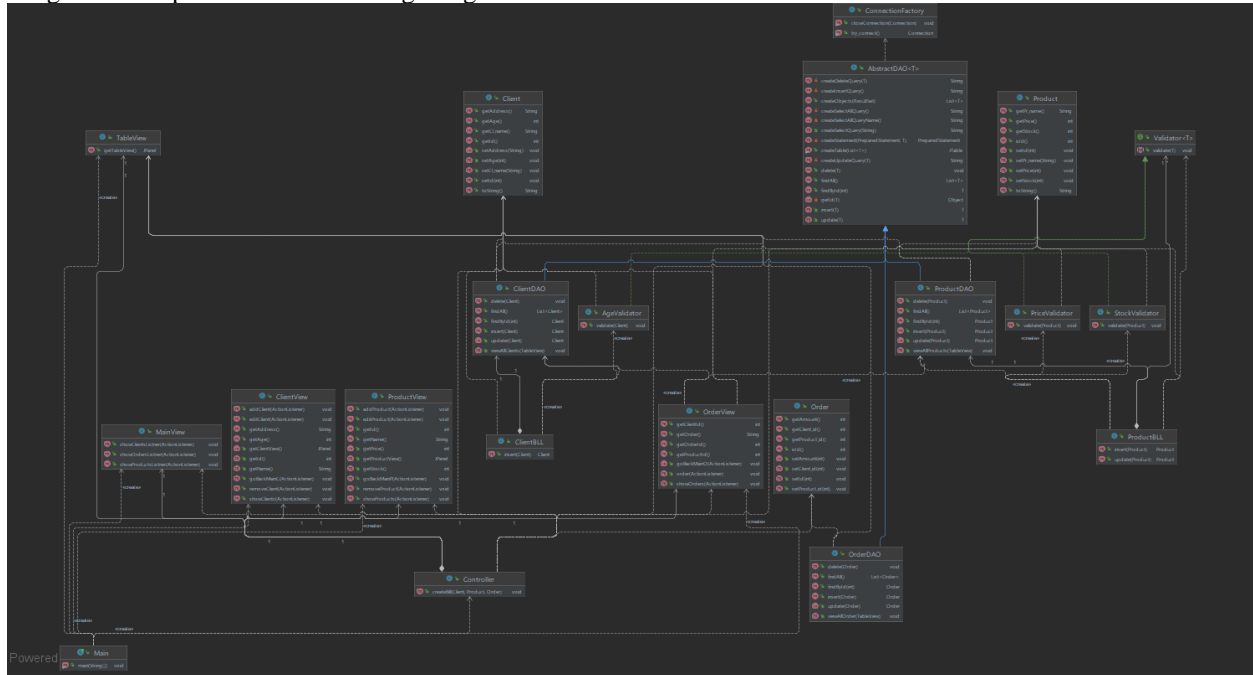
```

Order no.11
Product: Gin amount: 2
Client no.3: Gusteru
Total price: 24

```

3. Design

For this application I have chosen an OOP design, using a layered architecture to organize the classes in packages. There are 6 packages: BusinessLogic, Connection, Model, DataAccess, Controller and Presentation. The image below represents the Class diagram generated in IntelliJ:



The UML diagram shows each class with their corresponding methods and the links between the classes. This diagram allows the viewer to better understand the way the whole application works.

4. Implementation

1. The BusinessLogic package contains the following classes:
 - 1.1. `AgeValidator` implements the `Validator` interface. This class is used to verify that the client to be inserted has an age greater than 0. If it is not the case, a pop-up message is shown: "Invalid data".
 - 1.2. `PriceValidator` implements the `Validator` interface. This class is used to verify that the product inserted by the user has a price greater than 0. If the price is less than 0, a pop-up message is shown: "Invalid data".
 - 1.3. `Client BLL` class uses an `AgeValidator` object to validate the age of the client that is to be inserted and calls the `insert` method from `ClientDAO`.
 - 1.4. `ProductsBLL` class uses a `PriceValidator` object to validate the price of the product that is to be inserted and calls the `insert` method from `ProductDAO`.
2. The Connection package contains the `ConnectionFactory` class which contains 2 methods: public static `Connection try_connect()`, which establishes the connection with the postgres database and returns a `Connection` type object. The second method from this class is public static void `closeConnection(Connection connection)` which closes the connection received through a parameter.
3. The Model package contains 3 classes:

- 3.1. The Client class mirrors the Client table from the postgres database by having the same names for the class variables as the table has for its columns: id, cl_name, age, address. This class contains getters and setters, 2 constructors(one with parameters and one without) and a toString method that returns all the fields of a client concatenated into a string.
- 3.2. The Product class mirrors the Product table from the postgres database by having the same names for the class variables as the table has for its columns: id, pr_name, stock, price. This class contains getters and setters, 2 constructors(one with parameters and one without) and a toString method that returns all the fields of a product concatenated into a string.
- 3.3. The Order class contains the class variables(same as in the table): id, product_id, client_id, amount. It has getters and setters for all the field and 2 constructors: one with parameters and an empty one.
4. The DataAccess package contains 4 classes:
 - 4.1. The AbstractDAO is the main DAO class that creates and executes queries on the postgres database. Here are defined the methods that create generalized queries that get the fields of the object for which a query is executed through reflection so that it is ready to use for any type of object. This class also creates a list of objects to store the results of the query in it, as well as a table containing all the fields corresponding to a specific object. The main idea behind executing SQL queries by using AbstractDAO is first creating the query as a string with a method like "createInsertQuery()" which returns an insert query based on an object whose fields are extracted through reflection and then sending that query to the database through a statement with a method like insert(Object). The method public List<T> createObjects(Resultset resultset) gets a Resultset object passed as a parameter and through reflection assigns an object of a specific type the corresponding values based on the fields extracted from the database.
 - 4.2. The ClientDAO class has overwritten methods (@override) from the AbstractDAO that it extends, so that when a client query is made, the methods from the AbstractDAO class return a Client type object. In addition, ClientDAO has the "viewAllClients" method which creates a table containing a List of all the clients retrieved with a "findAll" query from the postgres database.
 - 4.3. The ProductDAO class has overwritten methods (@override) from the AbstractDAO that it extends, so that when a product query is made, the methods from the AbstractDAO class return a Product type object. In addition, ProductDAO has the "viewAllProducts" method which creates a table containing a List of all the products retrieved with a "findAll" query from the postgres database.
 - 4.4. The OrderDAO class has overwritten methods (@override) from the AbstractDAO that it extends, so that when an order query is made, the methods from the AbstractDAO class return an Order type object. In addition, OrderDAO has the "viewAllOrder" method which creates a table containing a List of all the orders retrieved with a "findAll" query from the postgres database.
5. The Controller package contains 2 classes:
 - 5.1. The Main class contains one method which is the starting point of the application, named main. This method creates a Controller type object which is used to control the whole program.\
 - 5.2. The Controller class is basically the central point of the application as it interacts with most of the other classes from the other packages. The constructor creates the following views: MainView, ClientView, ProductView, OrderView, TableView and initializes all the action listeners for the buttons that the user will interact with. Apart from the action listeners, there is also a method that creates a bill for each order in a text file that is being saved in the project folder.
6. The presentation package contains the following classes:
 - 6.1. The MainView class extends JFrame, it represents the first window of the application and it contains definition of buttons and actionListeners.
 - 6.2. The ClientView class extends JFrame, it represents the window of the application that allows the user to add, edit, remove and view clients and it contains definitions of buttons and actionListeners.
 - 6.3. The ProductView class extends JFrame, it represents the window of the application that allows the user to add, edit, remove and view products and it contains definitions of buttons and actionListeners.
 - 6.4. The OrderView class extends JFrame, it represents the window of the application that allows the user to add, edit, remove and view orders and it contains definitions of buttons and actionListeners.

6.5. The TableView class extends JFrame and it is used to create a JTable filled with Clients, Products or orders depending on the user's choice.

5. Conclusions

This application surely has its flaws and bugs, but in my opinion it is perfectly usable and can help maintain an order in a business that has flow of product. While programming this application I enhanced my OOP skills, I learned about layered architectures and how reflection works, how to do it and most importantly in my opinion: when it is useful. As in further developments, it would be nice to also save the time at which an order is made and register the employees as separate entities in the application so that they have the full access over the clients. As of right now the application is intended just for employees but in the future it might develop into an full ordering desktop inside a shop or factory.

6. Bibliography

[Java Create and Write To Files \(w3schools.com\)](https://www.w3schools.com/java/java_files.asp) – used for learning how to create files in Java
[uses reflection to get all the field names from java.sql.Types. : JDBC Data Type « Database SQL](https://www.java2s.com/java2s-core/JDBCDataTypes/JDBCDataTypes.html)

[JDBC « Java \(java2s.com\)](https://www.java2s.com/java2s-core/JDBCDataTypes/JDBCDataTypes.html) – learning about reflection and how to use it

[Introduction to Java Reflection - YouTube](https://www.youtube.com/watch?v=K8v8v8v8v8v) – learning about reflection

[Java Swing | JTable - GeeksforGeeks](https://www.geeksforgeeks.org/java-swing-jtable/) – Creating a table in Java Swing

[Connecting to the Database \(postgresql.org\)](https://www.postgresql.org/) – learning how to connect to a postgresql database