Homework #3 – Warehouse Management

**Contents:**

1. *Project Specification*
2. *Problem Analysis. Application Modelling. Scenarios. Use Cases*
   1. *Problem Analysis. Application Modelling*
   2. *Scenarios. Use Cases*
3. *Design* 
   1. *Relational Diagram*
   2. *Class Design* 
      1. *Main Class*
      2. *Controller Class*
      3. *DatabaseConnection Class*
      4. *ProductView Class*
      5. *ClientView Class*
      6. *Client Class*
      7. *Product Class*
      8. *Order Class*
      9. *AbstractDAO Class*
      10. *AbstractRepoClient Class*
      11. *AbstractRepoProduct Class*
4. *Using the application*
5. *Results*
6. *Conclusions and future developments*
7. Project Specification

The main objective of this project is to propose, design and implement an OrderManagement application aiming to process customer orders for a Warehouse. The implementation of this application contains the addition of new clients and products, which are then executed as an order in case the product is inside the Warehouse and the chosen client exists. Using this application we are able to see the products that are under-stock and modify their quantity. Also, we are able to add new orders, new products (in the Warehouse) or new clients. We can also, add, delete or update other product details. For designing the application, Java programming language was used together with the IntelliJ IDEA tool.

1. Problem Analysis. Application Modelling. Scenarios. Use Cases
   1. **Problem Analysis. Application Modelling**

As mentioned in the previous section, the main objective of the application is to design a user - friendly interface for computing some real life situations between a “client” and a “warehouse” with the help of a Order Management system and “splitting” the (background) tasks into Client operations, Product operations and Order operations. The implementation of the application requires the concept of “database”, since the details about the clients, products and order are stored inside a database, which plays the role of a virtual warehouse. The client is able to see the product and also the number of products available in the market. Based on this he can add an order with the products he or she wishes. The client is also able to see the price and the name for every product. we make distinction between Product Class and Client Class. Product Class contains methods like getters and setters of the attributes that a product owns: id, name, quantity and price. The Client class also contains methods like getters and setters of the attributes: id, name, address and age.

I**nput data**:

* Client Name
* Client Address
* Client Age
* Product Name
* Product Price
* Product Quantity
* Shipping Method

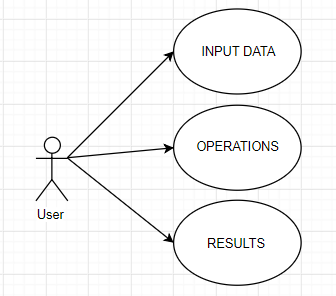
**Output data:**

* A log (updated in real time which consists in information about clients and queues)
* Image display (visual representation of servers and clients)
* Timer (updated in real time that shows how much of simulation time is left)

An exact description of the previous case is given below.

|  |  |
| --- | --- |
| Actors | Clients |
| Objectives | Warehouse management |
| General description | Client specifies the input data for the Warehouse management, since form is not automatically filled at first. He can also select a certain row from the JTable |
| Pre-condition | Client presses one of the CRUD buttons |
| Post-condition | If correct input data, data will be inserted in the table and synchronized with the MySql database. Otherwise an error message will pop up and the user needs to enter the data again. |

The using cases of the application are also presented in the next diagram:



It took up to 11 classes to create the application in a good, ordered manner:

1. **Main Class** - simple class containing the main(String[] args) method which marks the application start.
2. **Controller Class** - a class that deals with actions that need to be performed when certain buttons are pressed. It makes the connection between the “GUI” and the “Repository”.
3. **DatabaseConnection Class** – this class makes the connection to the MySql database, which stores the details about the clients, products and orders
4. **ProductView Class** - contains all the graphical user interface - frames, buttons, labels, textfields, etc for the products part.
5. **ClientView Class** - contains all the graphical user interface - frames, buttons, labels, textboxes, etc. for the client part
6. **Client Class -** this class is used for establishing the details for each client. In other words, it contains the getters, setters and constructors of each Client object
7. **Product Class -** this class is used for establishing the details for each product. In other words, it contains the getters, setters and constructors of each Product object
8. **Order Class -** this class is used for establishing the details for each order. In other words, it contains the getters, setters and constructors of each Order object
9. **AbstractDAO Class-** this class holds the project structure: the queries that need to be executed on the database, the Reflection part and some other important methods.
10. **AbstractRepoClient Class –** this class extends the AbstractDAO class, such that the Client can also call these methods through Reflection
11. **AbstractRepoProduct Class –** this class extends the AbstractDAO class, such that the Product can also call these methods through Reflection

3. Design

# 3.1 Relational Diagram

For better understanding of the classes usage I will give detailed explanations about the content and usage of every class.

O imagine care conține interior, computer, cer, laptop

Descrierea a fost generată cu un grad foarte mare de încredere

# 3.2 Class Design

In the following paragraphs, I will present a brief description of every class, together with the relationships that happen between them. As we can see, the most often used relationship is the dependency relationship and is denoted by *-- >* but there are also inheritance relationships that are present, denoted by 🡪.

# 3.2.1 Main Class

The Main Class contains only one method **public static void main(String[] args)**.This method facilitates the launch of the application creating a new ClientView and ProductView object.

# 3.2.2 Controller Class

The Controller class deals with actions that need to be performed when certain buttons are pressed. It makes the connection between the “GUI” and the “Repository”. This means, that this class works as an inter – mediator between what we see, the Graphical User Interface represented by the 2 classes (ProductView and ClientView) and the “working engines” behind them, stored in the Repository package. Used methods:

* **public List<Client> findAllClients();**
* **public Client findClientById(int id);**
* **public int insertClient(Client c);**
* **public int updateClient(Client c);**
* **public boolean deleteClient(int id);**
* **public List<Product> findAllProducts;**
* **public Product findProductById(int id);**
* **public int insertProduct(Product p);**
* **public int updateProduct(Product p);**
* **public boolean deleteProduct(int id);**
* **public void orderProduct(int clientID, int productID, int quantity, int shippingID);**

# 3.2.3 DatabaseConnection Class

This class deals with actions that need to be performed when using the database. This has a very important role because the details about the clients, products and orders are stored inside the MySql database. The tables used in this application are called Client, Orders, Product and ShippingMethod. The methods are declared as follows:

* **public DatabaseConnection();**
* **private Connection createConnection();**
* **public static Connection getConnection();**
* **public static void close(Connection connection);**
* **public static void close(Statement statement);**
* **public static void close(ResultSet resultSet);**

# 3.2.4 ProductView Class

The GUI Class contains only the default constructor together with the declarations of the buttons, textboxes and labels that will be displayed on the frame for the product part. Every graphical element is laid out using the **setBounds()** predefined method, since there is no predefined layout in this frame - **setLayout(null)**. All the elements that are used will be then added to the frame using the **this.add()** method. This class also contains the instruction that establishes the size of the frame (**setSize(1600,700)**) being a 1600 x 700 pixels window. Using **setTitle()** is used for setting a title, which is lastly displayed on the screen by the method **setVisible(true)**.

# 3.2.5 ClientView Class

The GUI Class contains only the default constructor together with the declarations of the buttons, textboxes and labels that will be displayed on the frame for the client part. Every graphical element is laid out using the **setBounds()** predefined method, since there is no predefined layout in this frame - **setLayout(null)**. All the elements that are used will be then added to the frame using the **this.add()** method. This class also contains the instruction that establishes the size of the frame (**setSize(1600,700)**) being a 1600 x 700 pixels window. Using **setTitle()** is used for setting a title, which is lastly displayed on the screen by the method **setVisible(true)**.

# 3.2.6 Client Class

This class is responsible with the structure of the Client. This means it sets the attributes of the Client object with the help of the getter and setter methods. To be able to work with the structure imposed by the application, some getters and setters are defined:

* **public int getClientID();**
* **public void setClientID(int clientID);**
* **public String getName();**
* **public void setName(String name);**
* **public String getAddress();**
* **public void setAddress(String address);**
* **public int getAge();**
* **public void setAge(int age);**

# 3.2.7 Product Class

This class is responsible with the structure of the Client. This means it sets the attributes of the Client object with the help of the getter and setter methods. To be able to work with the structure imposed by the application, some getters and setters are defined:

* **public int getProductID();**
* **public void setProductID(int productID);**
* **public String getName();**
* **public void setName(String name);**
* **public float getPrice();**
* **public void setPrice(float price);**
* **public int getQuantity();**
* **public void setQuantity(int quantity);**

# 3.2.8 Order Class

This class is responsible with the structure of the Client. This means it sets the attributes of the Client object with the help of the getter and setter methods. To be able to work with the structure imposed by the application, some getters and setters are defined:

* **public int getOrderID();**
* **public void setOrderID(int orderID);**
* **public int getClientID();**
* **public void setClientID(int clientID);**
* **public int getProductID();**
* **public void setProductID(int productID);**
* **public int getShippingID();**
* **public void setShippingID(int shippingID);**

# 3.2.9 AbstractDAO Class

This class is the actual engine of the application because it is responsible with the CRUD operations on each table (insert, update and delete a user, product or order) and dealing with the Reflection part, namely extracting the table headers using Reflection.

# 3.2.10 AbstractRepoClient Class

This class is responsible with inheriting the behavior of the AbstractDAO class. This means, that all the methods, constructors and attributes available to the AbstractDAO objects, are available to an AbstractRepoClient.

# 3.2.11 AbstractRepoProduct Class

This class is responsible with inheriting the behavior of the AbstractDAO class. This means, that all the methods, constructors and attributes available to the AbstractDAO objects, are available to an AbstractRepoProduct.

4. Using the application

O imagine care conține captură de ecran

Descrierea a fost generată cu un grad foarte mare de încredere

In the two pictures we can observe the 2 frames responsible for the Client and Product. Each one is showing the “menu” for each object together with the buttons that simulate the operations which help the management of the warehouse.

O imagine care conține captură de ecran

Descrierea a fost generată cu un grad foarte mare de încredere

Each entry inside the JTable is synchronized with the MySql table. The buttons represent the queries that we run on the MySql table. The results after pressing the buttons are shown inside a scrolling JPanel which holds the JTable of the application.

5.Results

As it can be already seen, the results have a dedicated area in the graphical user interface. By testing the application on different input data, we can notice that it assures a good approach towards warehouse management. The clients, products and orders that are inserted, deleted or updated can be seen inside the table after any button press. As the results should be conclusive, we must have much more details in the “transaction” process.

6.Conclusions and future developments

This system designed for processing and optimizing order operations for warehouse management is a very useful tool that we can use nowadays. A good idea for realizing a simulation between “client”, “product” and “order” is carefully observing the real-life process. Like many other applications, we can develop this program in different ways. As a future development, we can make an application This software program could be developed by creating filters, to display products having certain attributes, like name and quantity, see the date when they were added. We can implement a more detailed graphical user interface and we can display more information useful for the user, for example, to filter the products below a certain price.