Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Department of Computer Science

2016/2017

Programming Techniques

Homework 3

Warehouse Management

Lecturer: Prof. Ioan Salomie

Teaching Assistant: Claudia Pop

Student: Acu D. Raul-Mihai

Group: 30425

‘

Table of Contents

1. Problem specification

Consider an order management application 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:

1. Domain specific classes: Order, Customer and Product

2. Business Logic (warehouse-specific processing) classes: OrderProcessing, WarehouseAdministration, ClientAdministration

3. Presentation classes: GUI related classes

4. Data access classes: Database access related classes

a. System`s requirements

a. Analyze the proposed application, determine the structure and behavior of its classes and draw an extended UML class diagram.
b. Design, implement and test the application classes. Use javadoc for documenting classes.
c. Define, design and implement a system of utility programs (examples: reports for under-stock, totals, filters, etc.).
d. Design and implement a comprehensive demo driver for the order management application.

Graphical interface:

o Window for client operations: add new client, edit client, delete client, view all clients in a table (JTable)

o Window for product operations: add new product, edit product, delete product, view all product in a table (JTable)

• Create a product order for a client: the application user will be able to select an existing product, select an existing client, and insert a desired quantity for the product to create a valid order. In case that there are not enough products, an under stock message will be displayed. After the order is finalized, the product stock is decremented.

• Use relational databases for storing the data for the application, minimum three tables: Client, Product and Order.

• Documentation

System`s description

The system provides an Graphical User Interface (GUI) which can be used for processing for processing customer orders for a warehouse. Relational databases are used to store the products, the clients and the orders.

The most common architecture pattern is the layered architecture pattern, otherwise known as the n-tier architecture pattern. This pattern is the de facto standard for most Java EE applications and therefore is widely known by most architects, designers, and developers. The layered architecture pattern closely matches the traditional IT communication and organizational structures found in most companies, making it a natural choice for most business application development efforts.

Components within the layered architecture pattern are organized into horizontal layers, each layer performing a specific role within the application (e.g., presentation logic or business logic). Although the layered architecture pattern does not specify the number and types of layers that must exist in the pattern, most layered architectures consist of four standard layers: presentation, business, persistence, and database. In some cases, the business layer and persistence layer are combined into a single business layer, particularly when the persistence logic (e.g., SQL or HSQL) is embedded within the business layer components. Thus, smaller applications may have only three layers, whereas larger and more complex business applications may contain five or more layers.

Each layer of the layered architecture pattern has a specific role and responsibility within the application. For example, a presentation layer would be responsible for handling all user interface and browser communication logic, whereas a business layer would be responsible for executing specific business rules associated with the request. Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request. For example, the presentation layer doesn't need to know or worry about *how* to get customer data; it only needs to display that information on a screen in particular format. Similarly, the business layer doesn't need to be concerned about how to format customer data for display on a screen or even where the customer data is coming from; it only needs to get the data from the persistence layer, perform business logic against the data (e.g., calculate values or aggregate data), and pass that information up to the presentation layer.

2. Usages

a. Clients management

The user is able to add new clients, edit the information of the existing ones, or, remove some clients from the database. However, some constraints are being used for keeping the database precise and clear.

b. Products management

The products are being stored in the warehouse using relational databases. The database table contains information about the product, such as: name, price and quantity left on the stock.

c. Orders management

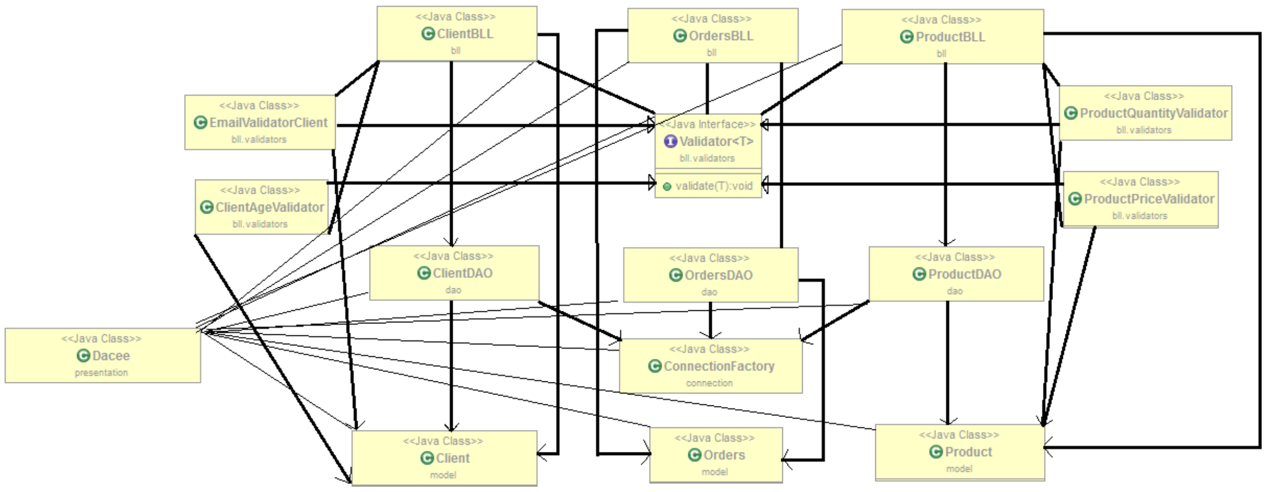
The order table uses two foreign keys, one for the client and one for the product. In this manner, the two tables are connected, being able to subtract data in order to create a bill containing the name of the client and the product he ordered, and the total price.

3. Design

a. UML Class Diagram

The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.

Structure diagrams emphasize the things that must be present in the system being modeled. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems



b. Packages

Package **bll**:

In computer software, business logic or domain logic is the part of the program that encodes the real-world business rules that determine how data can be created, stored, and changed. It is contrasted with the remainder of the software that might be concerned with lower-level details of managing a database or displaying the user interface, system infrastructure, or generally connecting various parts of the program.

Package **dao**:

In computer software, a data access object (DAO) is an object that provides an abstract interface to some type of database or other persistence mechanism. By mapping application calls to the persistence layer, the DAO provides some specific data operations without exposing details of the database. This isolation supports the Single responsibility principle. It separates what data access the application needs, in terms of domain-specific objects and data types (the public interface of the DAO), from how these needs can be satisfied with a specific DBMS, database schema, etc. (the implementation of the DAO).
Although this design pattern is equally applicable to the following:

1. -most programming languages;
2. - most types of software with persistence needs;
3. - most types of databases

It is traditionally associated with Java EE applications and with relational databases (accessed via the JDBC API because of its origin in Sun Microsystems' best practice guidelines "Core J2EE Patterns" for that platform).

The advantage of using data access objects is the relatively simple and rigorous separation between two important parts of an application that can but should not know anything of each other, and which can be expected to evolve frequently and independently. Changing business logic can rely on the same DAO interface, while changes to persistence logic do not affect DAO clients as long as the interface remains correctly implemented. All details of storage are hidden from the rest of the application (see information hiding). Thus, possible changes to the persistence mechanism can be implemented by just modifying one DAO implementation while the rest of the application isn't affected. DAOs act as an intermediary between the application and the database. They move data back and forth between objects and database records. Unit testing the code is facilitated by substituting the DAO with a test double in the test, thereby making the tests non-dependent on the persistence layer.
In the non specific context of the Java programming language, Data Access Objects as a design concept can be implemented in a number of ways. This can range from a fairly simple interface that separates the data access parts from the application logic, to frameworks and commercial products. DAO coding paradigms can require some skill. Technologies like Java Persistence API and Enterprise JavaBeans come built into application servers and can be used in applications that use a JavaEE application server. Commercial products like TopLink are available based on Object-relational mapping (ORM). Popular open source ORM products include Doctrine, Hibernate, iBATIS and JPA implementations such as Apache OpenJPA.

Package **connection**:

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is part of the Java Standard Edition platform, from Oracle Corporation. It provides methods to query and update data in a database, and is oriented towards relational databases. A JDBC-to-ODBC bridge enables connections to any ODBC-accessible data source in the Java virtual machine (JVM) host environment.

Package **model**:

This model contains the actual objects representation inside the database, the tables being considered as classes, and their columns as attributes.

4. Using and testing the system

a. User interface

The graphical user interface (GUI), is a type of user interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, instead of text-based user interfaces, typed command labels or text navigation.

Designing the visual composition and temporal behavior of a GUI is an important part of software application programming in the area of human–computer interaction. Its goal is to enhance the efficiency and ease of use for the underlying logical design of a stored program.

Such an user interface, is implemented in the **ui** package, by using the **Dacee** class.
The user interface is based on the properties of the above mentioned packages. All the objects we need are declared as attributes of the class and they are initialized in the constructor **Dacee()**.

- **JPanel** for the main window application:

```
private JPanel Panel1 ; .
private JPanel panelClients ; .
private JPanel panelProducts ; .
private JPanel panelOrders ; .
private JPanel panelAddClient ; .
private JPanel panelEditClient ; .
private JPanel panelRemoveClient ; .
private JPanel panelAddProduct ; .
private JPanel panelEditProduct ; .
private JPanel panelRemoveProduct ; .
private JPanel panelAddOrder; .
private JPanel panelRemoveOrder; .
private JPanel panelEditOrder; .
```

- **JLabels** for the written text
- **JTextFields** used for communicating with the database

```
private JTextField txtOrderAddIdclient; .
private JTextField txtOrderAddIdproduct; .
private JTextField txtOrderAddQuantity; .
private JTextField txtOrderEditIdclient; .
private JTextField txtOrderEditIdproduct ; .
private JTextField txtOrderEditQuantity ; .
private JTextField txtOrderEditID ; .
private JTextField txtOrderRemoveID ; .
private JTextField txtClientAddName ; .
```

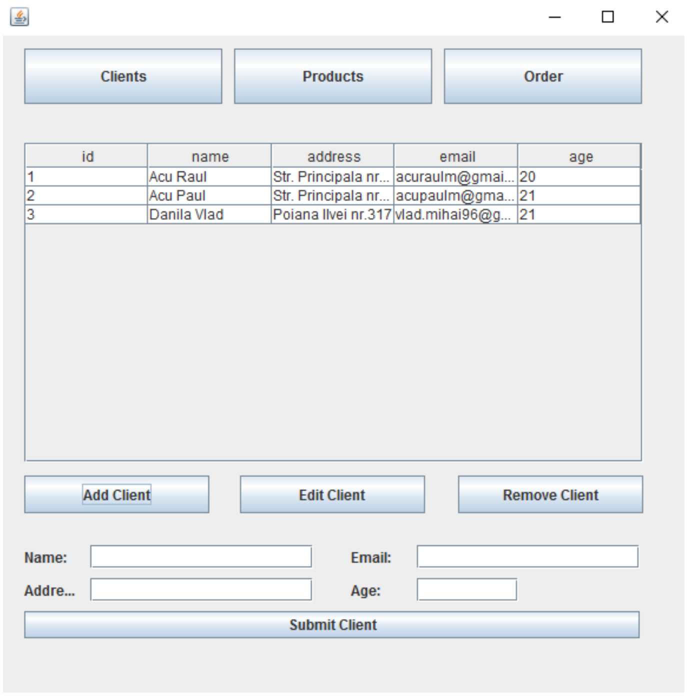
private JTextField txtClientAddAddress ; .
private JTextField txtClientAddEmail ; .
private JTextField txtClientAddAge ; .
private JTextField txtClientEditName ; .
private JTextField txtClientEditAddress ; .
private JTextField txtClientEditEmail ; .
private JTextField txtClientEditAge ; .
private JTextField txtClientEditID ; .
private JTextField txtClientRemoveID ; .
private JTextField txtProductRemoveID ; .
private JTextField txtProductEditName ; .
private JTextField txtProductEditPrice ; .
private JTextField txtProductEditQuantity ; .
private JTextField txtProductAddName ; .
private JTextField txtProductAddPrice ; .
private JTextField txtProductAddQuantity ; .
private JTextField txtProductEditID;

- **JTables** for displaying the tables` content .
  public static JTable *tableProducts*; .
  public static JTable *tableOrders*; .
  public static JTable *tableClients*; .
- **Various methods** that contribute to the element`s properties,

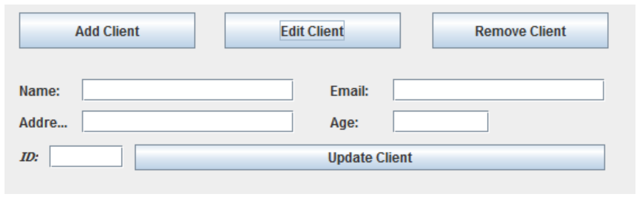
are responsible of adding them to the interface and describing their behavior:

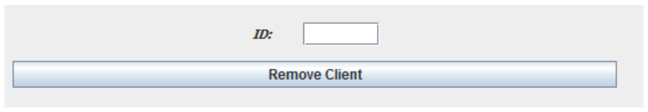
- *interfaceProperties()* – sets the Location, Size and Title of the JFrame.
- *initComponents()* – sets the properties of the elements described before; .
- *addComponents()* – adds the elements to the main window application; .
- *addListeners()* – assigns each button the according action; .
- *invisibleAll()* – hides most of the useless panels .

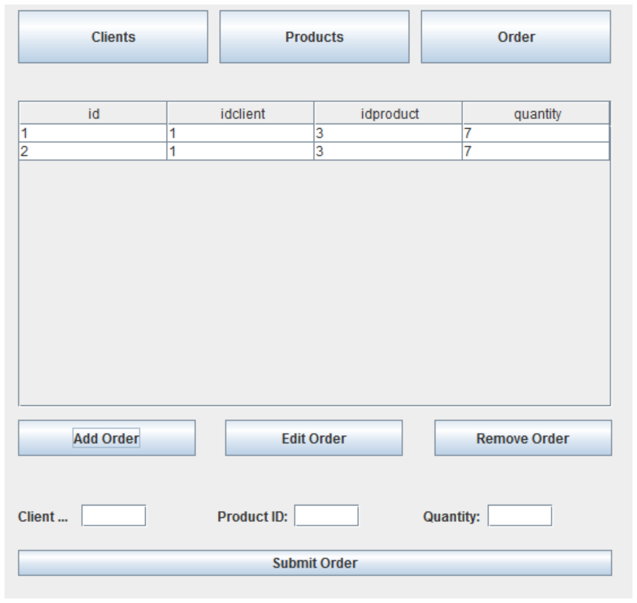
The actual **Graphical User Interface** is presented below.



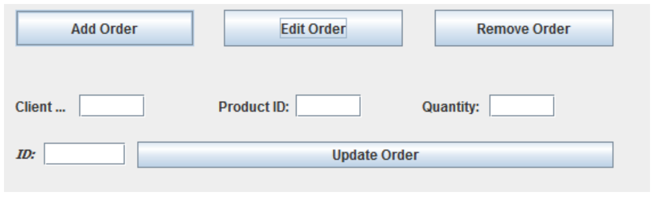
id	name	address	email	age
1	Acu Raul	Str. Principala nr...	acuraulm@gmai...	20
2	Acu Paul	Str. Principala nr...	acupaulm@gma...	21
3	Danila Vlad	Poiana Ilvei nr.317	vlad.mihai96@g...	21







id	idclient	idproduct	quantity
1	1	3	7
2	1	3	7



5. Conclusion

a. Advantages and Disadvantages

➢ **Advantages**

- Easy to use;
- Plain and simple, nothing to be confused by.

➢ **Disadvantages**

- One client cannot order more than a product at a time.
- It might fail if given wrong inputs;

b. Future development: ways of improving the system

- Desigining a better relational database;
- Trying to use as less memory as possible;
- Trying to improve the response time;
- Designing a more user-interactive interface;