Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2016-2017

Order Management System

Programming Techniques

Homework 3

Student Ieremiasș Viorel

Group 30422

Contents

[1. Problem Specification 3](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488628)

[2. Problem Analysis 3](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488629)

[2.1. General Overview 3](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488630)

[2.2. Input and Output 5](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488631)

[2.3. Use Cases 6](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488632)

[2.4. Scenarios 7](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488633)

[3. Design 9](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488634)

[3.1. Package design 9](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488635)

[3.2. Class Design 9](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488636)

[3.3. Data Structures 12](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488637)

[4. Conclusions & further improvements 13](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488638)

[5. Bibliography 13](file:///D:\IV\Documente\Facultate\An%202\Sem.%20II\Programming%20Techniques\PT2017_30422_Ieremias_Viorel\Documentatie%20HW3.docx#_Toc481488639)

# Problem Specification

**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

Other classes and packages can be added to implement the full functionality of the application.

**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.

# Problem Analysis

## General Overview

An order management system, or OMS, is a computer software system used in a number of industries for order entry and processing. Orders can be received from businesses, consumers, or a mix of both, depending on the products. Offers and pricing may be done via catalogs, websites, or broadcast network advertisements.

This application should be able to fulfill all the requirements that are basic for a system like this, that is, it must be able to display, search and keep track of orders, and implicitly of customers’ details and products’ basic properties. All the information is stored in a relational MySQL database, along with the information about the users which have access to the system. This way, all the data is easier to retrieve and accessed from different computers. In order to do this, specific methods for searching and inserting nodes into the tree will be developed, e.g. adding a product/ an order or deleting a product/an order. Specific classes will contain the implementation of the methods necessary to perform these operations.

Very important for an application like this is to keep a very accurate track of the transactions and modifications that apply to the data. As orders are made by clients or processed by workers, it is very important to constantly update the stocks and the sold quantities.

**Reflection Technique**

Reflection is a very powerful technique employed by many programming languages. It offers the possibility to observe and modify a program’s execution during its runtime. A program that uses reflection techniques can monitor the execution of a segment of code and act according to a desired goal related to that segment.

Reflection is generally accomplished by dynamically assigning program code and take decisions at runtime.

In Java language, reflection allows the programmer to obtain information about classes, interfaces, fields and methods without actually knowing their names at compile time. More than that, Java allow to instantiate a new object and invoke its methods. In some object-oriented programming languages, including Java, reflection can be used to override member accessibility rules. For example, reflection makes it possible to change the value of a field marked "private" in another class, other than the one the field belongs to.

The main reason why reflection techniques are used is because they offer the possibility to adapt a program to different situations in a dynamic way. The advantages lay in the fact that you can write more generic code, create testing facilities or in general obtain a considerable amount of knowledge about the classes and interfaces working with.

**Layered Architecture**

A multilayer or multitier architecture is a client-server architecture in which presentation, application processing and data management functions are accomplished by different packages of the program. By separating an application into separate tiers, there is the possibility to modify and make changes to a specific part of the program, without affecting the overall program, as long as the communications between the packages stays in the same protocols. Generally, the application is composed, as said before, of a presentation layer, handling the GUI and additional UI components, a logic model and processing package and a package that handles communication with the server and the database. Thus, the application is more flexible and the reusability is greatly increased.

## Input and Output

In order for this application to function according to real life situations, there is a different view for the person that administers the warehouse for example and the person that simply wants to place an order. Although for the moment these different perspectives are part of the same GUI and are delimited only by a distinction between the “Admin Space” and the “User Space”, the functions and capabilities offered by each of them is completely different, tailored to the needs and the roles of every person interacting with the system.

Products that are stored into the warehouse can be identified by a unique identification number, a name and a brief description. Additional information includes the price of a unit and the number of units in stock at that moment. Orders can be received according to the available products in the warehouse. A client can only order products that are available at that specific moment of time, in the limits of the stocks in that moment. When placing an order, if the client was a customer in the past, he/she is able to identify himself amongst the clients in the database. Otherwise, he/she will have to fill-in a form the name, address and a phone number. Situations like under-stock are reported. The order is not placed, but the user is warned that the number of products he desires to order is not available.

Clients are identified by a unique identification number, a name, a delivery address and a phone number. A new client can be registered into the application if he hasn’t placed any orders before.

Situations where user inputs wrong data or tries to modify something for which he/she does not have permission will be signaled accordingly with the corresponding error message, such that the application can recover from wrong user input situations.

According to the specified task requirements and the purpose of this application, the user will be able to:

- Add / modify products from the warehouse;

- Add/ modify clients for some specific reason;

- Add deliveries (orders);

- Add(increase) the stock in the case of under-stock;

- Filter data;

- Get information about registered clients;

- Update general information about products;

- Get information about deliveries;

- Get information about the stock (warehouse);

The administrator has special rights. Its main purpose is to review the placed orders, process them and manage the products from the database. The admin can also see all currently registered customers, all products on offer and all placed orders. After logging in, he gets to choose the area that he wants to visualize: orders, customers, products. Insert, delete and update are available for all the three areas. These operations are also very intuitive and simple, making it comfortable for the user. For inserting, the admin simply fills in the details of the product in the text fields. For updating, the desired row is selecting with a click, while the fields that are desired to be modified are filled in the previously mentioned text fields. The rest of the fields are to be ignored. Deletion takes place by click-selecting the row to be deleted and pressing the button. All the operations above take place instantly and the result can be observed in the table without refreshing or waiting times. Based on the task requirements, the admin can:

- View registered customers;

- View placed orders;

- View order details;

- Proceed orders;

- View products currently in the database;

- Add new products;

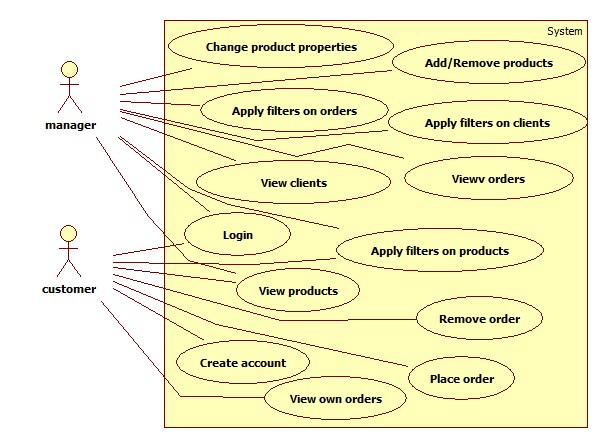
- Edit existing product information;

- Delete products;

## Use Cases

A use case defines the interactions between external actors and the system under consideration, to accomplish a goal. Actors must be able to make decisions. According to this definition we can identify the main actor to be the user of the application.

A use case represents specifying a sequence of actions that the system can perform by interacting with the system’s actors. The designed Warehouse Management system has two actors: the administrator and the customer. Both of them can be considered to be main actors, as their use cases generate observable results and do not interleave. For example, the admin can change the stock (add products, modify price or quantity of products), while the customer can only place orders. We can deduce the ways in which they can interact with the system to be the following: input the necessary data for placing and order (product identity, client identity, delivery details, ordered amount) for the customer and details about the products, orders and clients for the manager. These use cases refer to both the manager and the customer. The manager can add products, modify product properties (quantity or price), remove products, view all orders and all customers.

For a better understanding of the ideas stated above, I have created a use case diagram which describes the relationships between the actors of the system, as well as specifies the use cases.nd

## Scenarios

**Scenario I: Place order**

Summary: The customer selects the product he wants to order and checks out when he has finished.

Flow of events:

1. The customer starts the Order Management application.

2. The customer opens the “User space” window

3. From the first Combo Box, the customer chooses the product to order

4. If he ordered previously, the customer will find his name amongst the options of the second Combo Box, the one that selects the client

5. Otherwise, the user fills-in the required information in the text fields

6. The customer presses “SEND ORDER” button

**Scenario II: Insert Product**

Summary: The administrator adds a new product to the warehouse.

Flow of events:

1. The administrator starts the Order Management application;

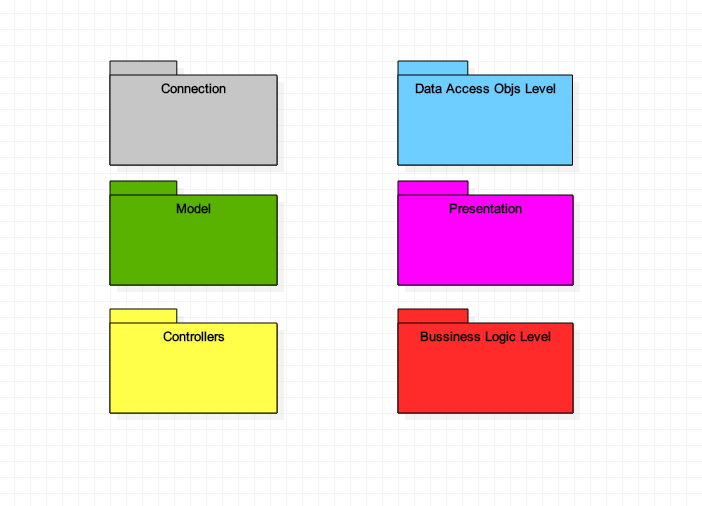
2. The admin opens the “Admin space” window;

3. The admin enters all the required about the product in the text fields;

4. The admin presses the “Insert” button

# Design

## Package design



## Class Design

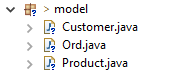
The application is structured on layers, each one handling a different matter.

**Connection**

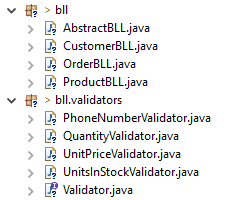
Contains a single class, ConnectionFactory, that creates the connection with the database, allowing extraction and insertion of data.

It uses the singleton design pattern – only one instance of ConnectionFactory can exist at any time. Thus, only one instance has access to the data inside, meaning that no synchronization or data inconsistency problems can appear. The static methos of ConnectionFactory open and close the connection and the different statements used to communicate with the database.

**Model**

Contains the classes that model the object that are part of the order management system, those being the customer, the order and the product. Each one of them contains specific fields, name, address, phone and an identification number, for example, in the case of the customer. The classes also include getters and setter for their fields and overwritten toString() methods.

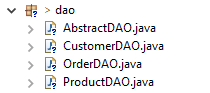
**Business Logic Level**

The business logic level is the one that makes the link between the model presented above and the deeper layers that communicate with the database.

The most important class of this package is the AbstractBLL class. It is a parameterized class, which means that the operations (insert, update, find, delete) are implemented on a generic class of objects. CustomerBLL, OrderBLL and ProductBLL extend the AbstractBLL class only to set the type of objects on which the operations are made, but there is no need to reimplement the capabilities.

The ‘validators’ sub package includes classes that all implement the “Validator” interface. These classes act as filter that check the correctness of the data that is to be inserted in the database.

**Data Access Objects**

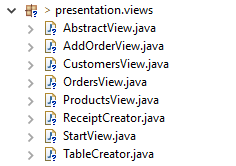
The DAOs are the objects that access the database through the connection opened by the ConnectionFactory. Again, as before, the AbstractDAO is a parameterized class, which means that the operations (insert, update, find, delete, create query statements, create objects) are implemented on a generic class of objects. CustomerDAO, OrderDAO and ProductDAO extend the AbstractDAO class only to set the type of objects on which the operations are made, but there is no need to reimplement the capabilities.

The capabilities of the DAO classes are quite complex, because it they need to cover the communication with the database. This is done by means of statements, objects of class PreparedStatement (used to send queries and commands to the database), respectively ResultSet (used to obtain the results of different operations).

The first step is to create the queries that the database, a MySQL one, can understand and relate to. Thus, methods like createSelectQuery(String field), createDeleteQuery(int id) and createInsertQuery(Object objToInsert) create and return the SQL queries that can be passed to the database server.

The CRUD (create – read – update – delete) specific operations, present in the business-logic level actually call methods in the DAOs, that handle the job of adding, modifying or deleting in the database, according to the needs of the BLL classes.

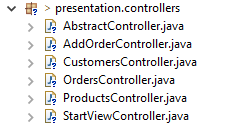
This implicitly means that the DAOs have to create and return objects resulted from a SELECT operation. Since AbstractDAO is a parameterized class, the only solution to obtain the class of T, instantiate it and fill all the fields it possesses is through reflection – a new instance of that type is created with an unparameterized constructor, that all the fields are accessed via the setter methods and written with the correct values.

**Presentation**

The presentation classes have the task to create and manage to UI for both of the types of users that can have access to the application: administrator and user. The view of the administrator is more complex to the one of the user. It includes separate windows for the products, orders and customers, a comprehensive set of operations and a detailed view over the situation of each of them. However, in order to avoid code duplication, all the tree presentation classes corresponding to the perspective of the administrator, CustomersView, ProductsView and OrdersView are versions of the same AbstractView class, a parameterized UI class that employs reflection techniques too, in order to display data in JTables and to pass objects to insert to the BLLs and from there to DAOs.

The TableCreator class is actually the one that contains the static class that, based on a set of objects it receives as arguments, creates the corresponding table.

StartView is the window initially display, the one that makes the distinction between the user and the admin rights and features. Finally, ReceiptCreator is employed to produces a text file of the order placed by a customer.

**Controllers**

Controllers are responsible for managing the relation between the model and the presentation classes. They notify the model when actions take place in the GUI and transmit the messages of the model to the GUI, to keep it updated. The main tool to accomplish this is the ActionListener, which observes when buttons are pressed, or any other events occur on the component of the windows.

## Data Structures

**Vectors**

Vectors could be useful for storing information about the data retrieved from the database, that has to be added into tables. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the vector has been created.

However, there are some disadvantages when using vectors, like the fact that the size of the vector has to be known and cannot be dynamically allocated.

**Lists**

An ordered collection (also known as a sequence). The user of this interface has precise control over where in the list each element is inserted. The user can access elements by their integer index (position in the list), and search for elements in the list. Unlike sets, lists typically allow duplicate elements. More formally, lists typically allow pairs of elements e1 and e2 such that e1.equals(e2), and they typically allow multiple null elements if they allow null elements at all. It is not inconceivable that someone might wish to implement a list that prohibits duplicates, by throwing runtime exceptions when the user attempts to insert them, but we expect this usage to be rare.

The List interface places additional stipulations, beyond those specified in the Collection interface, on the contracts of the iterator, add, remove, equals, and hashCode methods. Declarations for other inherited methods are also included here for convenience.

**Array Lists**

This Java class (member of the Java Collection Framework) has many methods for manipulating arrays (such as sorting and searching). Although these facilities exist and can be used, the structure of a binary search tree shall be implemented without them. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized).

# Conclusions & further improvements

I found this project a very interesting one, as it posed quite a few challenges and helped me understand a lot of aspects about designing and coding a real-life application.

First of all, it involved working with a database, and that is a very good point, because that is how most applications do. Communication with the database may seem intimidating, but the connector handles it in a way that is simple and straightforward for the user, while the concise, limited but very powerful SQL syntax allows any operation on the database in an intuitive manner.

Secondly, the layered architecture is a good example of organization, responsibility-delegation and level designing. It also made much clear what is the purpose and the function of each component of the system.

Finally, reflection techniques prove to be very useful in many situations when information about the attributes or capabilities of an instance or type is required at runtime.

Further development:

* Create a log-in window for normal user and for manager
* Allow more products in an order
* Alert manager when under-stock or over-stock occurs
* Implement data update directly on the table
* Improve text version of the receipt

# Bibliography

[https://en.wikipedia.org/](https://en.wikipedia.org/wiki/Reflection_(computer_programming))

<https://docs.oracle.com/>

<http://tutorialspoint.com/>

<https://www.javatpoint.com/java-tutorial/>