

Technical University of Cluj-Napoca  
Faculty of Automation and Computer Science  
Computer Science Department

Assignment 2

-JAVA Project-

Coordinating: Claudia Pop

Student: Onaciu Andreea

Group 30414

Table of Contents

[1. Assignment Objectiv 3](#_Toc447016593)

[2. Problem Analysis 4](#_Toc447016594)

[2.1 Modeling 4](#_Toc447016595)

[2.2.1 Model 6](#_Toc447016596)

[2.2.2 Business Logic Layer 8](#_Toc447016597)

[2.2.3 Data Access Objects 10](#_Toc447016598)

[2.2.4 Connection 12](#_Toc447016599)

[2.2.5 Presentation Part 12](#_Toc447016600)

[2.2.5.1 Controller Part 12](#_Toc447016601)

[2.2.5.2 View Part 13](#_Toc447016602)

[2.2 Scenarios 14](#_Toc447016603)

[2.3 Use Cases 14](#_Toc447016604)

[3.Design (UML Diagram) 15](#_Toc447016605)

[4.Implementation 15](#_Toc447016606)

[5.Results 16](#_Toc447016607)

[6. Conclusions 16](#_Toc447016608)

[7.Future developments of the project 16](#_Toc447016609)

[8.Biography 16](#_Toc447016610)

## Assignment Objectiv

The task of the assignment was:

Consider an application OrderManagement for processing customer orders. The application uses (minimally) the following classes: Order, OPDept (Order Processing Department), Customer, Product, and Warehouse. The classes OPDept and Warehouse use a BinarySearchTree for storing orders.

a. Analyze the application domain, determine the structure and behavior of its classes, identify use cases.

*b. Generate use case diagrams, an extended UML class diagram, two sequence diagrams and an activity diagram.*

*c. Implement and test the application classes. Use javadoc for documenting the classes.*

*d. Design, write and test a Java program for order management using the classes designed at question c). The program should include a set of utility operations such as under-stock, over-stock, totals, filters, etc.*

The task states that the project must be able to perform some operations regarding order and product, client processing, for example:

-insert order;

-calculate price;

-generate invoice;

-insert product;

-insert order;

-delete product;

-delete customer;

-update customer;

-update product;

-show clients;

-show products;

In order

In order to store the information and to keep it update it properly I use a database, made in the program MySql Workbench. The problem was to find a clear and easy to mange system of tables. I decided that the most important tables needed to solve the task are the following:

* orderDetails table;
* product table;
* customer table;

Each table will hold information necessary to make a friendly view for the client, which in this case is the administrator of the shop.

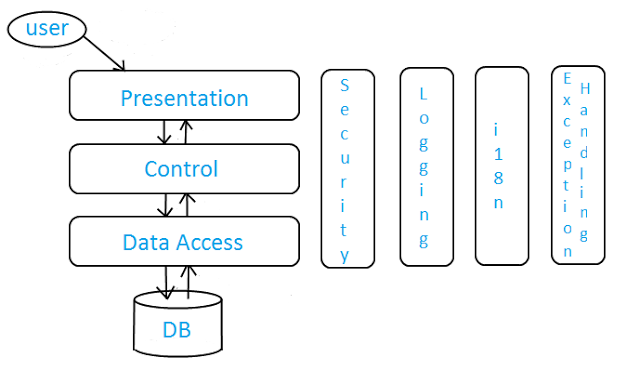
## Problem Analysis

## 2.1 Modeling

The process of modeling the problem begins with choosing a design pattern suitable for the requirements of the project. Given the fact that the assignments ask for a graphical user interface, one design pattern that matches these conditions is business pattern.

In a standard enterprise application which has a database and graphical UI (web or desktop), there are some typical layers which constitute the application. In this article we will mention those layers and give some instructions about them.

The general diagram is as below: 

[](http://4.bp.blogspot.com/-whbN41cW1MY/USSAXuBdShI/AAAAAAAAAfc/kA7g1D0tVVM/s1600/layers.png)

There are some important properties about that diagram:

There are vertical and horizontal layers. Vertical layers may be thought as general application service libraries which can paralelly work on all horizontal layers and independent from each other. However, a horizontal layer's subject of interest is only its neighbour (top and bottom) horizontal layers, and they work sequentially from top-to-bottom and bottom-to-top. Lastly, user can only interact with topmost horizontal layer.

Horizontal layers:

* Presentation: Every enterprise application has a UI, in fact graphical UI. This UI can be web or desktop based, which doesn't matter. The rule is simple. UI takes user action and sends it to the controller. And at the end it shows result taken from controller to the user. UI can be implemented according to MVC, MVP, MVVM or another approach.
* Control: Handles business logic of the application. Takes info from user and sends it to DB layer (DAO or ORM framework) and vice versa. Abstraction type of controller may vary (a separate control and business layers for example)  according to the application parameters or development patterns (MVC, MVP, MVVM, ...) but the main idea remains the same.
* Data Access: Database handling layer of the application. It may contain entity definitions, ORM framework or DB connection codes having SQL sentences, according to the abstraction decision. Its role is getting data from controller, performing data operation on database and sending results again to controller (if result exists). Database independence is a very important plus for this layer, which brings flexibility.

The presentation layer was done using the MVC pattern, i.e. the model-view-controller type of design.

* *Model*- The model represents data and the rules that govern access to and updates of this data. In enterprise software, a model often serves as a software approximation of a real-world process.
* *View* - The view renders the contents of a model. It specifies exactly how the model data should be presented. If the model data changes, the view must update its presentation as needed. This can be achieved by using a push model, in which the view registers itself with the model for change notifications, or a pull model, in which the view is responsible for calling the model when it needs to retrieve the most current data.
* *Controller* - The controller translates the user's interactions with the view into actions that the model will perform. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in an enterprise web application, they appear as GET and POSTHTTP requests. Depending on the context, a controller may also select a new view -- for example, a web page of results -- to present back to the user.

For each of these parts, I created a separate package, to organize and have a clean view of the project. Afterwards, the next step is to focus onto the structure of each part : model, view and controller.

### 2.2.1 Model

Starting with the model, which represents the base of the project, I had to decide how to organize in classes with attributes and capabilities the problem. The entities correspond to the tables, and have the same attributes like the columns of the tables. This makes the consistency of the design. The following classes construct the model part:

1. Product

The Product class was absolutely necessary as the main goal of the project is to have system capable to perform different operations on product. The product class is viewed more like a warehouse, because it stores also the quantity of the product.

Attributes:

Each product has quantity, price and name. The product will be identified in the rest of the project by its name. The quantity and the price must be positive values and this characteristic will be maintain at the insertion of a new project due to the business layer.

Capabilities:

-a constructor that sets all fields;

- setters and getters for each field;

1. Customer

Customer is also an essential entity of the product and represents the table with the same name.

Attributes:

Each customer is characterized by its name and by its email.

Capabilities:

-a constructor that sets all fields;

- setters and getters for each field;

1. Order

The order class represents an activity, but also an entity of the problem. The order would have contain customer and product as fields, but for a easier approach I decided to insert the following fields:

Attributes:

Each order will contain the customer name, the product name, how much of this product the customer ordered (quantity), and the total price of the order.

Capabilities:

-a constructor that sets all fields;

- setters and getters for each field;

- a method that generates an invoice, more exactly a .txt file having the name of the customer and containing details about the order like : name of the customer, name of the product, price and quantity.

### 2.2.2 Business Logic Layer

The Business Logic Layer package contains the classes that perform the connection between the queries from the Data Access Objects package and the Presentation package. It is used to validate the data that the user, in this case the administrator, sends as input through the interface.

1.Product Operations

The Product Operations class access the ProductDAO to return to the controller the needed information.

Attributes:

* ProductDAO object;

Capabilities:

The following methods where created :

-deleteProduct – that sends to de dao object the request to delete a product;

-getInfo – that returns a string matrix for the View frame to display the products;

-getProductNames – that returns a string vector containing the names of the all products, this is done by calling a dao query;

-insertProduct – that insert a valid product in the database by calling a dao query;

-updateProduct – that updates a product from the database containg valid information, also done by calling a query from the dao object;

-validateInsertProduct – validates if the attributes of the product that will be inserted is correct, which means it has a name full of letters, a positive price and a positive quantity;

2.Customer Operations

The Customer Operations class access the CustomerDAO to return to the controller the needed information.

Attributes:

* CustomerDAO object;

Capabilities:

The following methods where created :

-deleteCustomer – that sends to de dao object the request to delete a customer;

-getInfo – that returns a string matrix for the View frame to display the customers and their attributes;

-getCustomersNames – that returns a string vector containing the names of the all customers, this is done by calling a dao query;

-insertCustomer – that insert a valid customer in the database by calling a dao query;

-updateCustomer – that updates a customer from the database containg valid information, also done by calling a query from the dao object;

-validateCustomer – validates if the attributes of the customer that will be inserted is correct, which means it has a name full of letters, and the email contains the “@” character;

3.Order Operations

The Order Operations class access the OrderDAO and the ProductDAO, to return to the controller the needed information.

Attributes:

* ProductDAO object;
* OrderDAO object;

Capabilities:

The following methods where created :

-calculatePrice – that sends to the insertOrder method the price of the order, by multiping the price of the object (taken from a productDAO query) with the quantity given as an input in the frame by the user of the application;

-insertOrder – that insert a valid order in the database by calling a dao query;

-validateOrder – validates if the attributes of the order that will be inserted is correct, which means the quantity required exits in the database and it is positive;

### 2.2.3 Data Access Objects

The Data Access Objects package contains classes that perform the direct access of the database via queries, using the Sql libraries existing in JAVA. The classes are divided according to the tables in the database and respect the CRUD operations:

CRUD stands for Create, Read, Update and Delete. As required in the task, the on the customer and on the product all operation will be possible, while on the order table only the create operation which in this case stands for insert is possible.

1.ProductDAO

The ProductDAO class access the database using the connection establish in the ConnectionFactory to return to the needed information to the Business Logic Layer.

Attributes:

* connection object;
* a string representing the query;
* a ResultSet object which will hold the result set of the query;
* an ArrayList of products

Capabilities:

The following methods where created :

-deleteProduct – that deletes an product based on its name from the database;

-getProduct – that returns a product object representing a specific object that was searches by its name;

-getProducts – that returns an array of products objects, more exactly all the products from the database;

-insertProduct – that insert a valid product in the database;

-updateProduct – that updates a product from the database containg valid information;

2.CustomerDAO Operations

The CustomerDAO class access the database using the connection establish in the ConnectionFactory to return to the needed information to the Business Logic Layer.

Attributes:

* connection object;
* a string representing the query;
* a ResultSet object which will hold the result set of the query;
* an ArrayList of customers;

Capabilities:

The following methods where created :

-deleteCustomer – that deletes an customer based on its name from the database;

-getCustomers – that returns an array of customer objects, more exactly all the products from the database;

-insertCustomer – that insert a valid customer in the database;

-updateEmail – that updates a customer from the database containing a valid email;

2.OrderDAO Operations

The OrderDAO class access the database using the connection establish in the ConnectionFactory to return to the needed information to the Business Logic Layer.

Attributes:

* connection object;
* a string representing the query;
* a ResultSet object which will hold the result set of the query;
* an ArrayList of orders;

Capabilities:

The following methods where created :

-getOrders – that returns an array of orders objects, more exactly all the products from the database;

-insertOrder – that insert a valid order in the database;

### 2.2.4 Connection

Another part of the project was to create a connection between the database and the the project. This was done in this package by the Class Connection Factory which uses a DRIVER\_CLASS,a PASSWORD,a URL, A USER to connect to database.

### 2.2.5 Presentation Part

The presentation part controller the way the user can access the information and also creates the graphical interface of the project.

### 2.2.5.1 Controller Part

For making a link between the other layers and the view, the package controller part holds classes that communicate between those.

1. AbstractController

This class holds the main parts on which the problem works on, more precisely objects referring to: the frame, and the next frame that should be displayed when clicking a specific button.

1. MenuController

This class holds the action listeners for all the buttons from the menu, and realizes the access to all the other controllers and frames from the project, so here will be the place where all the other frames will return.

- *DeleteCustomerActionListener*

*- DeleteProductActionListener*

*- InsertCustomerActionListener*

*- InsertOrderActionListener*

*- ViewCustomerActionListener*

*- InsertProductActionListener*

*- ViewProductActionListener*

*- UpdateProductActionListener*

*- UpdateCustomerActionListener*

All these inner classes, implement the Action listener interface and they write the method actionPerformed() which has to decide what actions to take when a specific button is pressed. In our case the following actions take place: to the actual frame the visibility is set to false, and controller is transferred to another controller according to the button pressed;

1. DeleteCustomerController
2. DeleteProductController
3. InsertCustomerController
4. InsertProductController
5. InsertOrderController
6. ShowClientsController
7. ShowProductsController
8. UpdateCustomerController
9. UpdateProductController
10. MenuController

This class holds the main method that starts the application.

### 2.2.5.2 View Part

It contains the following classes:

1. DeleteCustumerFrame
2. DeleteProductFrame
3. InsertCustomerFrame
4. InsertProductFrame
5. InsertOrderFrame
6. ViewFrame
7. UpdateCustomerFrame
8. UpdateProductFrame
9. AbstractFrame

## 2.2 Scenarios

In order to perform the polynomial operations there were multiple scenarios that have to be taken into account for performing mathematically correct the computations.

First of all, because of the fact that the division and also the integration of polynomial can give as a result polynomials with real coefficients, instead of casting the result to integer, I decided to allow polynomial to have real coefficients, in order to have realistic results.

Another problem that had to be resolved was having to deal with the “0” terms, or the null polynomial, more precisely, after performing a computation that gives a result equal with 0, there will be terms in the resulting polynomial with zero coefficient that have to be discarded somehow, and the result that it is showed onto the screen to be “0”, and not “0+0+0”. To deal with this problem I choose to check if the polynomial has only zero terms and show only one term, which means only one zero.

When comes to the division operation, a frequently known error has to be taken into account: Division by zero. To handle this error, the division class which holds the method execute, verifies first if the divisor is different from the null polynomial,

If yes, the division is performed, else the method will throw a Runtime exception, with the message “Division by zero”.

For letting the user known that the division cannot be performed, the division action listener, test itself that the divisor is or not a null polynomial, and based on that performs the division or shows a message ( “Division by zero”) to let the users know that the operation could not been done.

## 2.3 Use Cases

Brief Description

The systems waits for the user to select one of the menu buttons, the user can perform operations

- on the client : delete, insert, view, update client;

- on the product : delete, insert, view, update product;

- on the order : insert new order;

For each of this buttons the system opens a new frame, where the client can insert information, or view information.

Actors

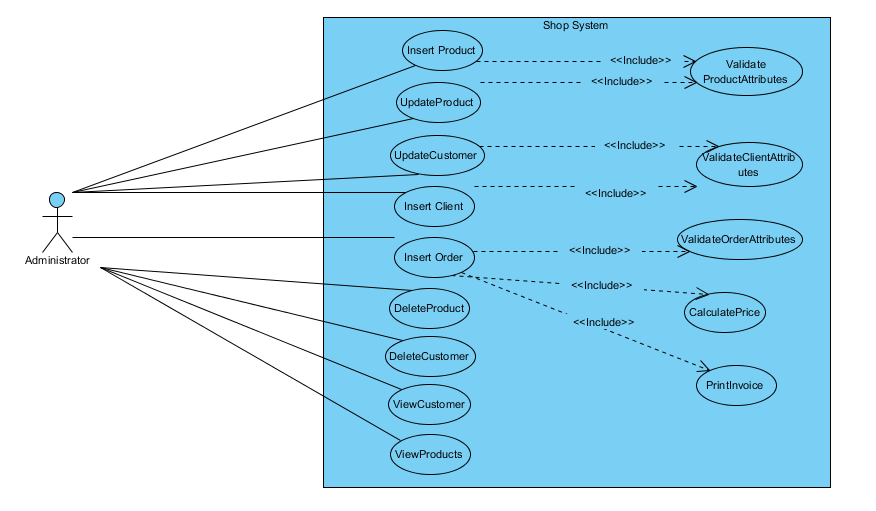
The actors of that can use this application can be an administrator of a shop. Preconditions

At each new run of the application, the text fields will be ready to enter new information on each text field.

Postconditions

Only valid information will be inserted into the database.

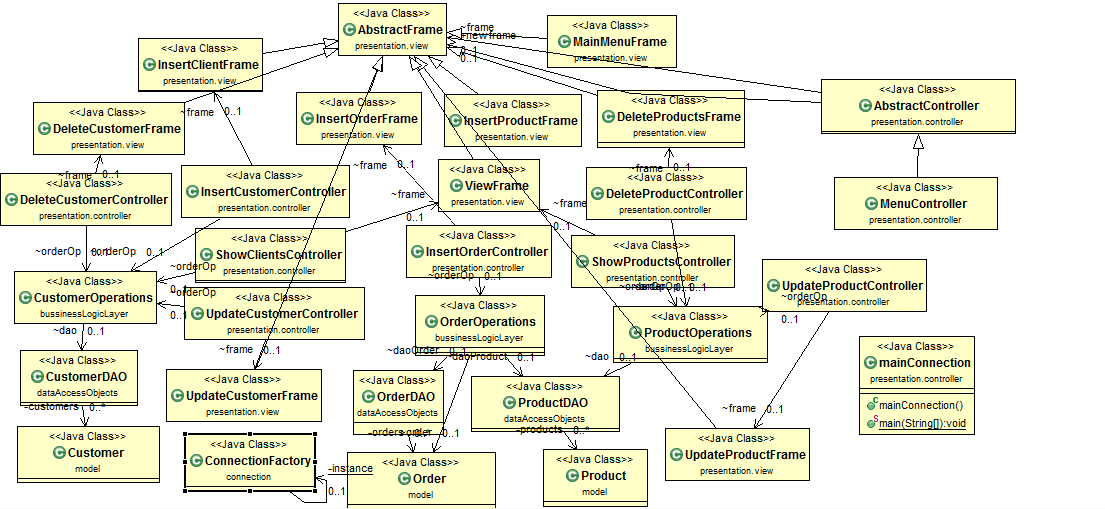
The use case diagram is made in Visual Paradigm and is the following :



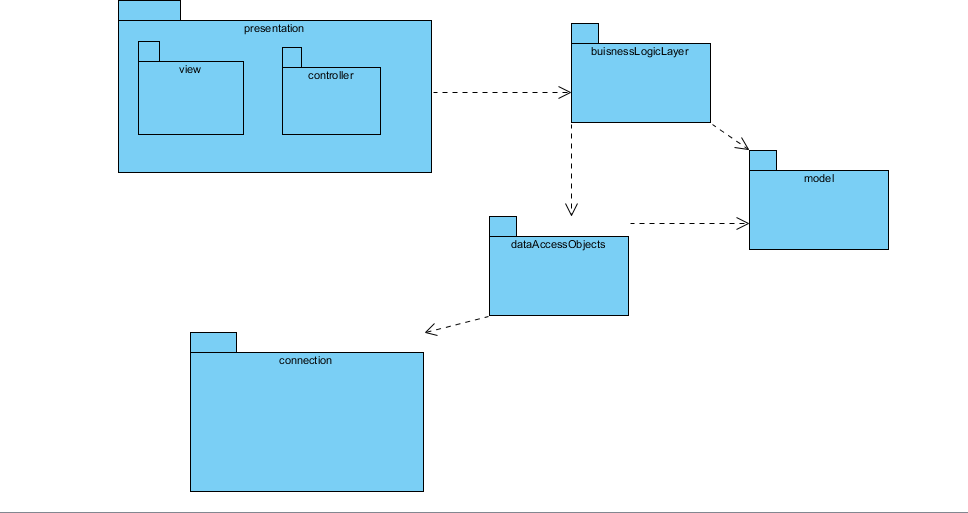
## 3.Design (UML Diagram)

The UML Diagram was build using Eclipse IDE.

This is a more clear view of the diagram, without showing nesting and association multiplicity:



Package Diagram :



## 

## 4.Implementation

The project was implemented in Eclipse IDE, with the help of its features and some predefined classes and interfaces, the more important are : Collections,Array List, Comparable, Swing, Awt, ActionListener , java.Sql;

## 5.Results

The results obtained can be viewed and tested in the user interface.

To verify if the application perfoms well the operations, one can check in the actual database to see if the modification were made, and if yes, to check if they were made correctly.

Here are some print screens which show the design of the user interface :

The menu:

## 6. Conclusions

The assignment help me improve my knowledge regarding Object- Oriented- Programming. I learned a new design pattern used in business, that helps provide security of a database. I also learn to connect a database to a java project and how to execute queries in java.

## 7.Future developments of the project

1. The project can be improved by adding more operations to the menu, for example

View orders, delete order;

1. By creating a login interface that allows only some users to make changes in the database.
2. To improve the user interface, by making it more friendly, and more easy to use.

## 8.Biography

**Books**:

- Joshua Bloch, Effective Java (2nd Edition);

- [Kathy Sierra, Bert Bates](http://shop.oreilly.com/product/9780596009205.do#tab_04_2), Head First Java (2nd Edition), O'Reilly Media;

- Barry Burd, Java for Dummies (5th Edition), Wiley;

**Websites**:

- <http://stackoverflow.com/>

- <https://www.oracle.com/java/>

-http://www.oracle.com/technetwork/articles/javase/index-142890.html

- http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/

- http://christoph-burmeister.eu/?p=1556

- http://theopentutorials.com/tutorials/java/jdbc/jdbc-mysql-create-database-example/