Technical University of Cluj-Napoca
Faculty of Computer Science
Student: Nagy Hanna
Group: 30421
05th of April, 2016

Programming Techniques
Assignment no. 2

1. Objective of the homework. Requests.

   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 request of the assignment was to implement and test an application based on a real world activity, namely ordering-buying system.
2. Scenes. Models. Analyzing the problem.

   View of a client, customer, buyer.

   We have to imagine tha case when we want to order from a specific website or on an application which is on our phone or etc. And as we enter on the site or application we have to log in, of course, if we have an account there. But that is an other case. Lets have a look on the case when we have an account on the earlier mentioned web site or application and we have to enter our unique username and password. If we did not type it correctly as we settled down at the moment of signing up, then we get an error message and after that, we have several chances to log in again. Supposing, we managed to log in with our unique unsername and password.

   Then we will see another window where there are several buttons, tables, labels. We select in a specific way the product or products we want to order or buy. During these moments of buying but not finalized yet we are able to see our shopping cart and the total sum of our selected products. If we have finished the selecting, buying, ordering and we want to pay and get the bill of what we bought we push a button which have a finalize meaning.

After that bill is sent and the shopping cart is made empty.

Now, lets analyze the case when we do not have an account on the comercial website or application. As we enter the website or application we will see an option where we can create a new profile giving some details and choosing a unigue username. So, we type a username, password, email and country and then push the sign in button. Filling the username, password and email text labels are mandatory, without these informations the program is not able to intruduce you as a new client. And if your username which you chose is not unique, in the sens that it is already exists in the database and it is someone else's unique identifier, you will get an error message that you have to choose a unique username. If you managed all these, you can log in as it was said before.

And now we see the same window which was described earlier. We select products, their quantity, we can see also the shopping cart in real time, and then at the end push the finalyze button and get the bill.

View of an admin/ employer of the firm

Every admin has a unique identifier as a username and a password. They might log using the same buttons and textfields as the clients and customers. The difference is you cannot create an admin if you do no have the proper log in informations. You can enter as a new client, but you cannot enter as a new admin creating earliert an account of an admin. The informations for the logging in to the admin are stored as anything else in the database. The admin informations are encapsulated there. With this order management application you cannot manipulate/ modify /edit or see who are the admins and their usernames and passwords. These datas have to be controlled and manipulated only by the persons in charge of the security and maintaning the system. So, if you are an admin and you entered in the system, you have several opportunities. You can manipulate the products and the clients. You can see all the products , you can delete a product , you can update a product , you can add a new product , you can update the quantity of a product. As manipulating the clients you can delete a specific client. The role of the admin is to maintain the system, to update several time the products and all the data needed to this system to functionate in the proper way.

3. Steps of the project

This application, system needs a lot of data , it has the control of so many information and because of these facts as storing data structure we have chosen to store the data in a database. We chose as Database Management System DBMS MySql Workbanch. The other part of the project is implemented in Ecplise in an object oriented language, namely JAVA.

There were made several tables in the D B M S: Products, Orders, Clients, Admins. For every real world object there is a table too. In table products as it name says we store the actual product with its specifications: its id, its name, its brand, its colour, its size, its RAM size , its price and its quantity. In the table Clients there are the clients who have accounts to the application, they can log in, their informations as username, password, email address and country are stored in this table. When the system verifying whereas the username is in the system or not it has to come to the database and verify it there. In the table Admins there are a few admins, who are responsible for the maintanance of the whole system and production and commerisation of the products. In the table Orders there are the key parts of the system, in this table are stored the clients who have orders. This table consists of the coloumns like id, client id, product id and the quantity which is wanted by the buyer from that specific product.

Layers. Layer Architecture.

For a better structure we used layer architecture. This constists of several packages everyone with its own rule.
This architecture consists of 4 packages, namely: entities , presentation , business logic level (B L L), data access layer (D A L). In Entities package we have the classes 1:1 corresponding to bthe tables from the database. These classes from the Entities package are meant to used by all other classes if they need. Data Access Layer (D A L) package is meant to make the connection between the system and database management system (in our case we used as database management system MySql Workbanch). Only in this package exists classes which make direct connection with the database management system. Above this Data Access Layer package there is the Business Logic Layer (B L L) which is the brain of the system and here happens the main actions of the system. For example, in our case, here is the updating of the quantity after an order or the actual order making, calculating the total price and so on. This Business Logic Layer has access just to the classes in the Data D Access A Layer L package. Above the Business Logic Layer (B L L) there is a package named Presentation. As its name says here are the classes related to the presentation, the interaction of the user and the system, so the gui – graphical user interface. This package has only access to the classes from the B L L package and of course to the entity classes, which can be used by any classes and packages in the system.

Package diagram is shown below in the figure 1 and it reflects the Layer Architecture Structure.

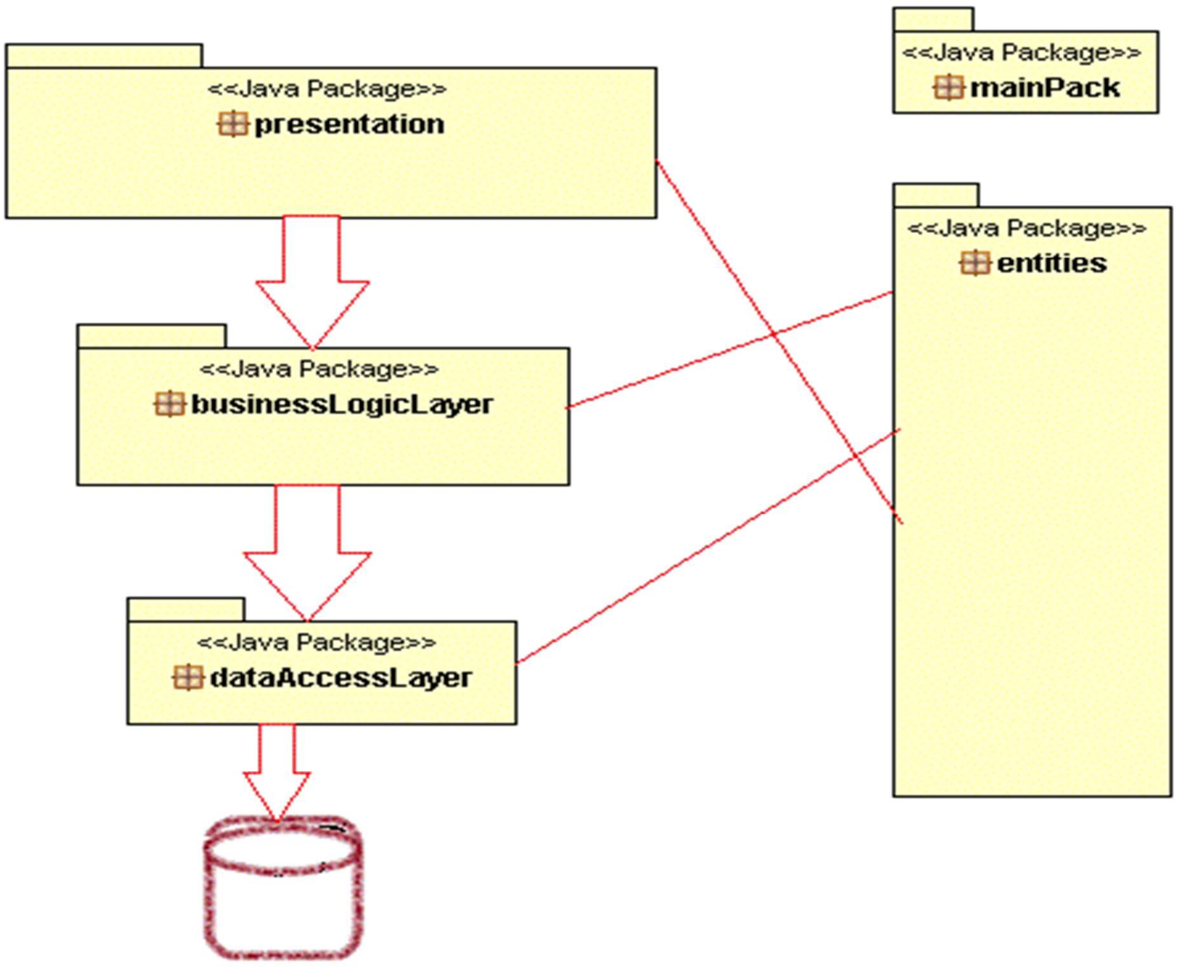


Figure 1

Classes in these packages:

As we said above, there is a package Entites where are 1:1 corresponding classes of the tables from the database.

For example, in the database we have the table Products with the attributes: Product_id, Product_name, Product brand, product colour, product size, product ram, product price, product quantity.

In the entites package we have a class with the same instance variables:

```
package entities;

public class Products {
```

```java
private int id;
private String name;
private String brand;
private String colour;
private String size;
private double price;
private String ram;
private int quantity;


public int getId() {
	return id;
}

public void setId(int id) {
	this.id = id;
}

public String getName() {
	return name;
}

public void setName(String name) {
	this.name = name;
}

public String getBrand() {
	return brand;
}

public void setBrand(String brand) {
	this.brand = brand;
}

public String getColour() {
	return colour;
}

public void setColour(String colour) {
	this.colour = colour;
}

public String getSize() {
	return size;
}

public void setSize(String size) {
	this.size = size;
}

public double getPrice() {
	return price;
}

public void setPrice(double price) {
	this.price = price;
}

public String getRam() {
```

```
            return ram;
        }

        public void setRam(String ram) {
            this.ram = ram;
        }


        @Override
    public String toString() {
        return "Product [productId=" + id + ", name=" + name + ", brand="
                +brand+", colour= "+ colour +", size="+size+", price="+
price+ "]" ;
    }

        public int getQuantity() {
            return quantity;
        }

        public void setQuantity(int quantity) {
            this.quantity = quantity;
        }
```

And so on so an. A class from the Data Access Layer architecture which uses the Products Class from the Entities is for example, the ProductsDAO where are the SQL queries from getting/ reading/ selecting/ updating/ deleting and this kind of perfomations. The class ProductsBLL from the Business Logic Level uses the Products entity class and the ProductsDAO classes too.In the similar way are used the other ones too.
Code snippet example from the B L L package:

```
public class ProductsBLL {


 private ProductsDAO productsDAO;


        public ProductsBLL(){
            productsDAO= new ProductsDAO();

        }


        public void insertNewProduct(int productId, String name, String
brand, String colour, String size, String ram, double price,int quantity){
            productsDAO.insertProduct(productId,name, brand, colour, size,
ram,price, quantity);
        }

        public void deleteProduct(int productId){
            productsDAO.deleteProduct(productId);
        }
	public void updateProduct(int productId, String name, String brand,
String colour, String size, String ram, double price, int quantity){
		productsDAO.updateProduct(productId,name, brand, colour, size,
ram,price,quantity);
	}

	public List<Products> getAllClients(){
		return productsDAO.getAllProducts();
	}

	public ResultSet getResultSet(){
		return this.productsDAO.getAllProductResultSet();
	}
	public ResultSet getFilteredResultSet(String brand){
		return
this.productsDAO.getAllProductFilteredByBrandResultSet(brand);
	}

	public void updateProductQuantity(int productId, int quantity){
		productsDAO.updateProductQuantity(productId, quantity);
	}
	}
```

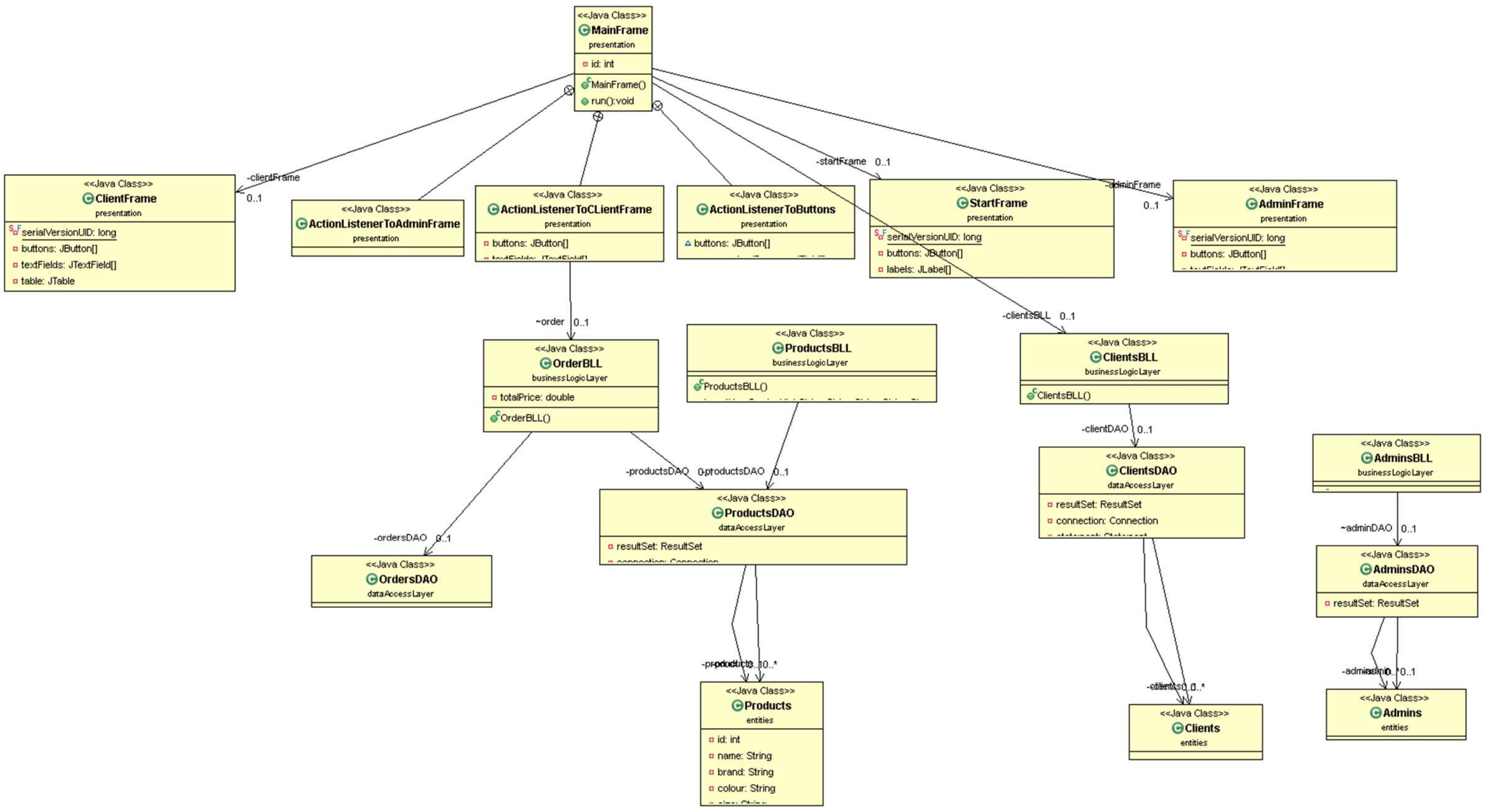
In the Business Logic Level the class which has the main functionality is the Ordering class:

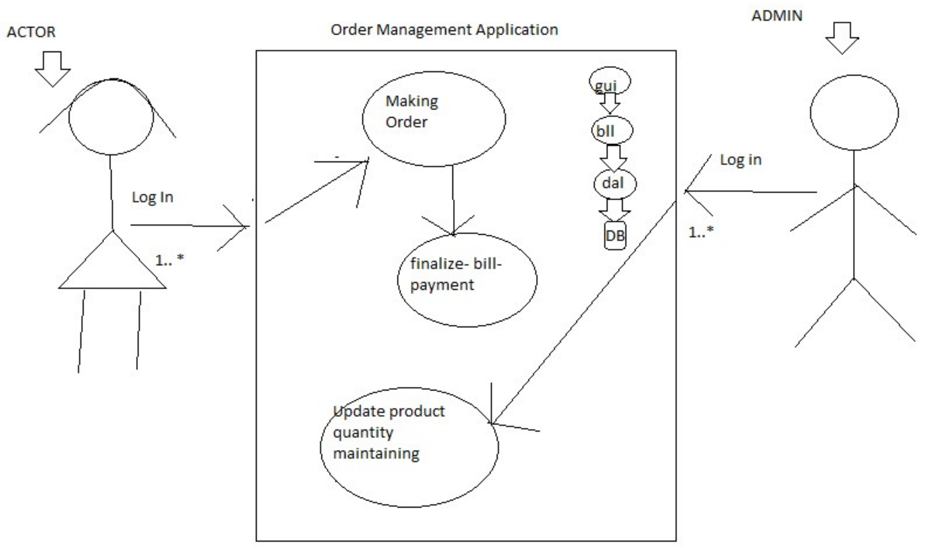
```
public class OrderBLL {

	private OrdersDAO ordersDAO;
	private double totalPrice;
	private ProductsDAO productsDAO;

	public OrderBLL() {
		productsDAO = new ProductsDAO();
		ordersDAO = new OrdersDAO();

	}

	public Boolean checkIsQuantity(int productId, int wantedQuantity) {

		if (productsDAO.getQuantity(productId) >= wantedQuantity) {
			return true;
		}
		return false;

	}

	public void order(int clientId, int productId, int wantedQuantity) {
		if (!this.checkIsQuantity(productId, wantedQuantity)) {
			JOptionPane.showConfirmDialog(null, "There is no enough
product in the warehouse");
		} else {

			ordersDAO.insertAnOrder(clientId, productId,
wantedQuantity);
```

```java
                  this.updateProductQuantity(productId,
productsDAO.getQuantity(productId) - wantedQuantity);

            }
      }

      public Double calculateTotalPrice(int clientId) {
            totalPrice = ordersDAO.calculateTotalPrice(clientId);
            return totalPrice;
      }

      public void updateProductQuantity(int productId, int quantity) {

            productsDAO.updateProductQuantity(productId, quantity);

      }

      public double getTotalPrice() {
            return this.totalPrice;
      }

      public ResultSet getOrderResultSet(int clientId) {
            return ordersDAO.getAllOrderedProduct(clientId);
      }

      public void deleteOrder(int clientId, int productId) {
            ordersDAO.deleteAnOrder(clientId, productId);
      }

      public void deleteAllOrders(int clientId) {
            ordersDAO.deleteAllOrder(clientId);
      }

      }
```

In this class is calculated the total price, it controls the ordering, updates the product quantity after an order and so on.

4. Class Diagrams. Use Case Diagram.





5. Enhancements

For a better perfomance of the system, application it could be extended to have many types of products which are stored in different tables in the database, furthermore, it might extend the structure of the database, in the meaning of that to have more tables, more relationships and better encapsulation of the datas in the tables. For example, it ould be nice if we had a tables just for the warehouse of the products, where we could store just the quantity of them and when there is case of updating the quantity we do not have to access the whole table where all the information about that product is store.

What we have learnt by implementing this order management system:

- How to use database, how to connect a DBMS to the IDE;
- How to apply Layer Architecture and structure our project in a way that it van be seen and analyzed very easily.
- How the order managements works in real life.

  . . . .

6. Testing

Testing is made by testing by the gui the whole system, there is no extra code just for testing the application.

7. Bibliograpy

a. http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
b. http://christoph-burmeister.eu/?p=1556
c. http://theopentutorials.com/tutorials/java/jdbc/jdbc-mysql-create-database-example/