**Order Management**

**Student: Flueran Robert-Denis**

**Group: 30423**

**Semigroup: 1**

**1.Homework Objectives**

The main objective of the homework is designing an application for managing orders for a warehouse with products, clients and orders stored in a MySQL database

Steps (objectives) in designing the Order Management app:

* Creating Java classes that describe tables from the warehouse database

Client class has fields for client’s name and city. Product class contains fields for product name, stock and price. OrderItem describes every single item that has been ordered and has fields for client’s name, product’s name and ordered quantity. Order is used for computing the clients’ bills and contains field for client’s name and sum to pay.

* Creating a class through which we establish a connection to the database

ConnectionFactory class has all the methods necessary for starting and closing a connection to the database.

* Creating a generic class for CRUD operations on the database and other classes that extend it

AbstractDAO contains all the methods and queries necessary for inserting, updating, deleting and searching for records in the database. Moreover, a class for each table in the database is created that inherits all methods from AbstractDAO in order to use them in that specific table.

* Creating classes that parse the input text file and call the appropriate operations on the database

View class has a method that reads each line from the input file, identifies the required command and calls the corresponding method from Controller class.

* Creating a class that generates bills for customers

OrderBLL has a method for generating pdf files representing bills based on what is found at Order and OrderItem tables.

* Creating a “Main” Class from which we start the application

Start class has the sole purpose of receiving a text file and pass it to the readFromFile method of a view object. After all required operations have been performed, Start calls generatePDF from an OrderBLL object that generates bills.

* Putting the application to the test with several text files

Two input text files have been created: one that puts all operations on the database to the test, and another one (called cleanup.txt) that has commands for clearing the database.

**2.Problem Analysis**

The problem consists mainly in ensuring that all database operations work properly, and that some operations on some tables influence records from other tables, as intended.

**“Order Management” use-case**

* A text file with commands for the database is given as input
* readFromFile method from View class is called from Start class to start parsing the input file
* Parser reads each line and identifies the required operation and calls a function from Controller class. Each method from controller has an AbstractDAO object declared
* The called controller method identifies what table the operations is about (Client, Product, OrderItem) and instantiates the AbstractDAO object accordingly (as ClientDAO, ProductDAO or OrderItemDAO).
* The corresponding operation is called from generic class AbstractDAO
* A connection to the database is established
* In case a command couldn’t be executed (e.g. trying to order more quantity than actual stock), a pdf with an error message is generated
* Connection to the database is closed

**3.Design**

I have created a host of packages that contain all the classes needed for the order management.

* BLL (Business Layer)

It contains only one class: OrderBLL, with the sole purpose of generating pdf files for customer bills.

* Connection

It contains only one class: ConnectionFactory, which simply has methods for establishing a connection to the desired database.

* DAO (Data Access Layer)

It contains the generic class called AbstractDAO, with methods for inserting, deleting, updating and searching for an element in a table, and methods for each table of the database that extend the generic class (ClientDAO, ProductDAO, OrderDAO, OrderItemDAO).

* Model

It contains the classes that describe all tables from the warehouse database: Client, Product, OrderItem and Order.

* Presentation (Presentation Layer)

It contains the classes responsible for parsing the input text file and calling the corresponding database operation for each given command

* Start

It contains only one class: Start, which simply calls readFromFile from View class and createPDF from OrderBLL, since these two methods do all the work regarding the input commands and calling other methods for different commands.

**Examples of commands that can be given to the application:**

* Insert : insert a client or a product record in the database

**-**client (Insert client: Sandu Vasile, Cluj-Napoca): Client

**-**product (Insert product: apple, 20, 1): Product

* Delete : delete a record from the database

-client (Delete client: Ion Popescu, Bucuresti): Client

-product (Delete Product: peach): Product

-order (Delete order: Sandu Vasile, apple, 100): OrderItem

-bill (Delete bill: Luca George): Order

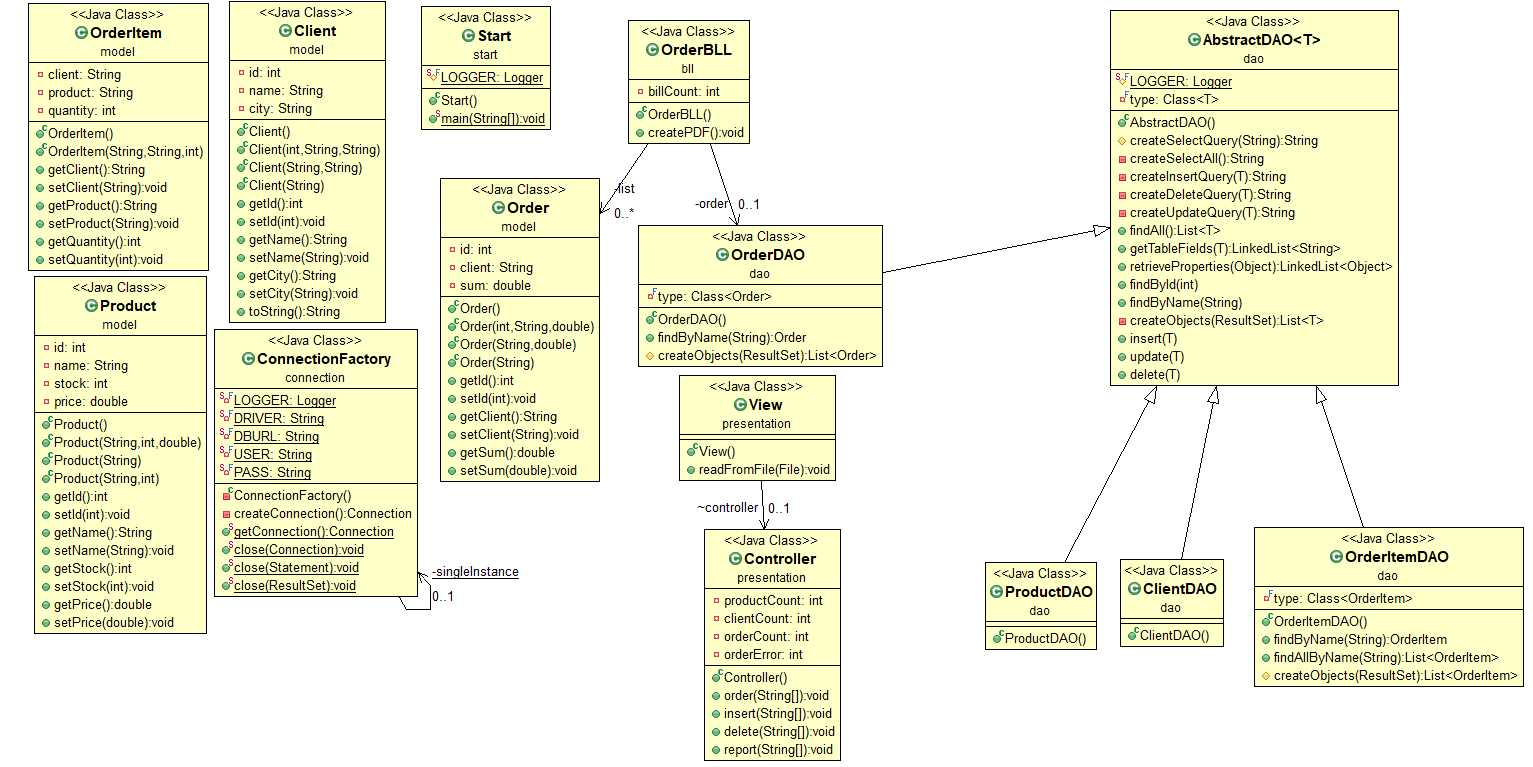
* Report: generate a report in pdf format with a table from the database

-client (Report client): Client

-product (Report product): Product

-order (Report order): OrderItem

**UML Class Diagram**



**4.Implementation**

Let’s start with the BLL package. Here, we’ve one class:

* OrderBLL

OrderBLL class contains an ArrayList of Order objects, an OrderDAO object, called order, for operations on Order table, and an int variable, called billCount, that is incremented each time we find a new bill in the Order table.

Methods:

OrderBLL has only one method, called createPDF, that generates bills in pdf format.

**Most important method from the package:**



Method starts by storing all records from the Order table in the ArrayList<Order> list. Afterwards, it traverses the list, increments billCount for each record, generates a pdf file and stores all records of ordered items (from OrderItem table) with client name equal to the name of the client from the current Order record (list.get(i).getClient()). The PDF is filled with client’s name, a list of things order by the client along with their prie, and total costs of ordered products.

Let’s move on to the Connection package, which also contains only one class:

* ConnectionFactory

ConnectionFactory class contains fields with all the information needed in order to establish a connection to a database.

Methods:

ConnectionFactory has methods for connection and for closing the database connection, the resultSet and the statement required for an operation. This class has been taken from the source code given in the indications for the project.

Let’s describe the DAO package:

* AbstractDAO

AbstractDAO class has one field (private final Class<T> type) which is a generic type variable that can be used in database operations for any table.

Methods:

AbstractDAO has methods for creating queries and methods for executing corresponding queries (insert – createInsertQuery, update – createUpdateQuery, delete – createDeleteQuery, findbyId – createSelectQuery, etc).

**All other classes from this package are simple extensions of AbstractDAO generic class. (Reflection techniques). Only OrderDAO and OrderItemDAO have two overridden methods, since the generic findByName doesn’t work on them (they don’t have a “name” column). All methods are equal when it comes to importance, and they’re quite similar, so I’m going to describe one of the methods.**



The Delete query is going to be stored in a StringBuilder object. The function starts by appending the basic part from the query to sb (DELETE FROM `[table\_name]` WHERE). Afterwards, the function traverses through the fields of the object we give as a parameter (could be Client, Product, Order or OrderItem) and, in the case they have a value other than the default one, they are going to be appended to sb. Here, I thought I should pass an object to the deletion operation and delete from the database a record with the exact fields of the object I give as parameter

Let’s describe the Model package:

* Client

Client class contains fields describing the client table: id (int, which is a primary key), name (String, represents client’s name), city (String, represents the city where the client lives in)

Methods:

Apart from getters and setters, the class has several constructors for different situations

* Product

Product class contains fields describing the product table: id (int, primary key), name(String, represents product’s name), stock (int, which says how many products of a kind are available at the moment), price (double, represents the price for one product).

Methods:

Apart from getters and setters, the class has several constructors for different situations

* OrderItem

OrderItem class contains fields describing the orderitem table: client (String, represents client’s name), product (String, represents product’s name), quantity (int, represents number of products of the same kind that have been ordered)

Methods:

Apart from getters and setters, the class has several constructors for different situations

* Order

Order class contains fields describing the order table: id (int, primary key), client (String, represents client’s name), sum (double, represents total costs for a client)

Methods:

Apart from getters and setters, the class has several constructors for different situations

Let’s describe the Presentation package:

* View

View class has only one field: a controller object, to call corresponding methods on databases from Controller class.

Methods:

View class has a method (readFromLine(File file)) that takes as parameter the input text file and reads each line from the file. Afterwards, it identifies the database from each command and the corresponding function is called by the controller object

* Controller

Controller class has fields that signal the number of pdfs generated for each table. After a “report” command is given from the input, one of the fields (depending on the table we have to generate the report from) is incremented.

Methods:

Controller class contains methods for inserting, deleting records from all four tables and a method for creating pdf reports.

**Important method from the package:**



This represents the method responsible for inserting records into the database. It takes as parameter the arguments received from the last read line of the input file. The method starts by declaring an AbstractDAO object and a string that is going to be equal to the name of the table we want to insert a record in. After the method identifies what table the command is about, the AbstractDAO object is instantiated accordingly (either as ClientDAO or ProductDAO). Afterwards, the other arguments are turned into variables, and a constructor for the object we want to insert is called.

In case we have to insert a product that is already in the database, then no new record is added, but an already existing product’s quantity is updated with the old quantity + new one.

**Important note**

**I’ve decided that, when a client is deleted from the database, his orders won’t be deleted, since, when an order has already been made, it shouldn’t be canceled by the simple deletion of an “account”.**

**5.Results**

I have managed to perform all the requirements for the assignment. I’ve put the application to the test with two input files, one that tests all operations on the warehouse database, and one that has commands for clearing the database, and both tests seem to work just fine.

I have created more than three tables, as required. The way orders are managed is split into two tables: OrderItem, which was every single item ordered by any of the clients; Order, which simply shows how much each client that actually ordered something has to pay.

Reflection techniques have been used in order to create a generic class that contains the methods for accessing the DB: create object, edit object, delete object and find object. The queries for accessing the database for a specific object that corresponds to a table are generated dynamically through reflection.

The input files the application was tested with are as follows:

test.txt:

Insert client: Ion Popescu, Bucuresti

Insert client: Luca George, Bucuresti

Report client

Insert client: Sandu Vasile, Cluj-Napoca

Report client

Delete client: Ion Popescu, Bucuresti

Report client

Insert product: apple, 20, 1

Insert product: peach, 50, 2

Insert product: apple, 20, 1

Report product

Delete Product: peach

Insert product: orange, 40, 1.5

Insert product: lemon, 70, 2

Report product

Order: Luca George, apple, 5

Order: Luca George, lemon, 5

Order: Sandu Vasile, apple, 100

Order: Sandu Vasile, orange, 7

Report client

Report order

Report product

cleanup.txt:

Delete client: Luca George, Bucuresti

Delete client: Sandu Vasile, Cluj-Napoca

Delete client: Ion Popescu, Bucuresti

Delete Product: peach

Delete Product: apple

Delete Product: orange

Delete Product: lemon

Delete order: Luca George, apple, 5

Delete order: Luca George, lemon, 5

Delete order: Sandu Vasile, apple, 100

Delete bill: Luca George

Delete bill: Sandu Vasile

**6.Conclusions**

This has been a truly interesting and useful assignment for me, since I learned a lot from it and I would also like to become a Java programmer. I already did a project on connecting to a MySQL database using Java at OOP, but the method presented in the indications for this project is a much easier and more efficient way than the one I’ve learnt at OOP. Also, at OOP, I only did insertion and deletion. Here, I got to work with all CRUD operations, to think about handling real-life situations (adding more products of the same kind; ordering some products, which would decrease the stocks for those products; generating error messages if the order couldn’t be fulfilled).

I also learned how to work with PDF files in Java and how to generate messages and even tables in PDF format.

**7.Bibliography**

* Three-tier architecture

<http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf>

* PDF format file creation

<https://www.baeldung.com/javadoc>

* SQL Dump

<https://dev.mysql.com/doc/workbench/en/wb-admin-export-import-management.html>

* Javadoc

<https://www.codejava.net/ides/eclipse/how-to-generate-javadoc-in-eclipse>