***Programing Techniques***

*Homework 3:*

*Database Management System*

* *Rus Rares Tudor*
* *Group 30422*

*Contents:*

1. *Requirements . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..3*
2. *Problem Analysis. . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4*
3. *Use Cases . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5*
4. *Projection . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6*
5. *Implementation . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7*
6. *Conclusions . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8*
7. *Bibliography . . . . . . . . . . .. . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . 9*

*--Requirements :*

The main objective of this assignment is to design and implement a system for processing customer orders and execute them based on the disponibility of the warehouse. To achieve such a goal, we have to implement a database system in such a order to process the tasks given by an user. In order to successfully fulfil this task we have to understand what a queue is, so here is the definition:

Queues are commonly used to model real world domains. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue based systems is interested in minimizing the time amount their "clients" are waiting in queues before they are served. One way to minimize the waiting time is to add more servers, i.e. more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the service supplier. When a new server is added the waiting customers will be evenly distributed to all current available queues. The application should simulate a series of clients arriving for service, entering queues, waiting, being served and finally leaving the queue. It tracks the time the customers spend waiting in queues and outputs the average waiting time. To calculate waiting time we need to know the arrival time, finish time and service time. The arrival time and the service time depend on the individual clients – when they show up and how much service they need. The finish time depends on the number of queues, the number of clients in the queue and their service needs. . Furthermore, the application uses (minimally) the following classes:

• Model classes - represent the data models of the application

• Business Logic classes - contain the application logic

• Presentation classes – classes that contain the graphical user interface

• Data access classes - classes that contain the access to the database.

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

a. Analyze the application domain, determine the structure and behavior of its classes and draw an extended UML class diagram.

b. Implement the application classes. Use javadoc for documenting classes.

c. Use reflection techniques to create a method createTable that receives a list of objects and generates the header of the table by extracting through reflection the object properties and then populates the table with the values of the elements from the list

d. Implement a system of utility programs for reporting such as: under-stock, totals, filters, etc.

*--Problem Analysis:*

Abstraction in its main sense is a conceptual process where general rules and concepts are derived from the usage and classification of specific examples, literal signifiers, first principles, or other methods.

Conceptual abstractions may be formed by filtering the information content of a concept or an observable phenomenon, selecting only the aspects which are relevant for a particular subjectively valued purpose. For example, abstracting a leather soccer ball to the more general idea of a ball selects only the information on general ball attributes and behavior, excluding, but not eliminating, the other phenomenal and cognitive characteristics of that particular ball. In a type–token distinction, a type (e.g., a 'ball') is more abstract than its tokens.

In order to design an application which fulfills the above mentioned description we have to firstly understand what a database is. A database is a structured set of data held in a computer, especially one that is accessible in various ways. So basically a database is a collection of connected table, with various data, on which a set of particularly operations can be done. A table is made of his header with the column specification and of the data it contains. For this application we will made use of the following on a single table operations.

* Insert – this operation will create a new row in the table containing the information we give as its parameters.
* Delete – this operation will delete a row from the table depending on the row number we specify.
* Update – this operation will update a specified row with some changed values we specify.
* Find – this operation will return a specific row depending on the row number we specify
* Select all – this operation will return the entire table with its header and data.

A valid database has to implement correctly all this operations we will try to do so with our tables.

The database we will be working on is the well known MySql Workbench. So one of the goals of this application is to make the MySql database resemble our graphical user interface part of the application. In order to be able to write MySql queries a minimum amount of database knowledge about the MySql statements will be necessary. The other goal which does not depend on the MySql database is the part where we will have to create a bill for every approved order..

The approach used by me is the following:

Normally if we want to do some stuff in our application and see the result in the database then the logical thing to do would be to make a connection between the java project and the database. We do this by downloading some Java Database Connector provided by the MySql environment and inserting it in our workspace. Next we have to make a connection in MySql with a database server. After all this work is done we will try to connect to the database from the java project by using the java library provided by MySql java.sql and the information we received when making the connection to the server(host, root, password).

When making the database operations described above we will make use of the MySql statements which are some predefined queries which we will have to use in our program.

The java classes will be grouped into packages in order to maintain an order and to make the code more easy to understand and more importantly to allow the reusability of the code with perhaps different tables or different parts of the code.

Every table of the relational database will correspond to a class in our workspace, having as attributes the same name as the header description of the columns.

We will have three database tables which we will work with therefore three table classes.

The description of the problem requires us to make use of reflection techniques. The reflection approach was indeed very useful to the design of the application because without it we would have been required to create a lot of methods and to write a lot of cod but what reflection does is that it allow us to create a method which can take as parameter a class or an object and when used in a class we simply just set the parameter to the class we are in. So other said, instead of creating 3 similar methods, for the Client, Product and Order tables, which do almost the same thing, for example the insert into table operation, we design just once applicable for all of the three tables.

*--Use Cases:*

The user has a lot of options from which he can choose freely. On both Client and Product table panels he can introduce the values he wants to introduce in the table, the id of the row he wants to delete, or to find and the values of the row he wants to update. For each action a button was made. On the last panel the user is required to check from a box the name of the client that wants to make the order, the product he wants to buy and its size. After this the order button should be pressed.

- *User Main SUCCES scenario*:

-User launches application successfully.

-User provides relative inputs based on the task.

-User presses the running button so the application starts.

-The date introduced by the user are correctly processed.

-The running is performed successfully with no exceptions or errors.

-The result is displayed in a label provided by the application.

*- User Main FAILED scenario*:

-User can’t launches application successfully.

-User provides wrong inputs for the requirements.

-The inputs are unsuccessfully transformed into the Task parameters.

-The result is not displayed in a label provided by the application.

Here is an example of how the user should give the commands:

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

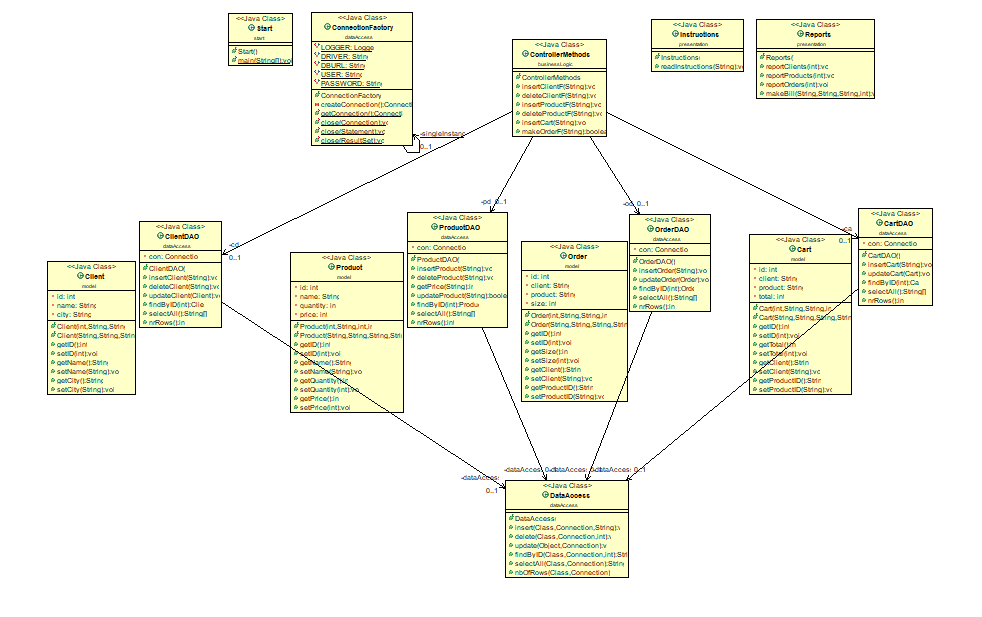
Order: Sandu Vasile, apple, 100

Report client

Report order

Report product

*--Projection:*



*--Implementation:*

In the following, we will take a closer look at the role and the implementation of each package, class with its attributes, constructors and methods.

~ *businessLogic*

- *ControllerMethods.java*

~ *dataAccess*

- *CartDAO.java*

- *ClientDAO.java*

- *OrederDAO.java*

- *ProductDAO.java*

*- DataAccess.java*

This class is the most complex class of all because here are made the general methods using the above presented reflection technique, which will be used by each table class in order to do all of the presented operations. The first method is the insertion method, having as parameters the connection we are working with, the Class representing the table in which we want to do the insertion and a string containing the values we want to insert. We use some predefined reflection methods in order to find the header of the table we are working with, and its name and together with this information and with the string given as parameter we create a SQL statement which covers the insertion of our values and execute it. The delete method takes a parameter the connection and the table we are working with and an integer which indicates the id of the row we want to delete. We again create an SQL query with fields obtained using reflection and the integer passed as parameter to the function and execute it in order to delete the desired row. The update function takes as parameter the connection we are working with and an object. The given object will be compared with an existing object in the table by having the id as common field and the values of the old row object will be updated to the values of the new row object, given as parameter. The findByID function takes as parameter the class and the connection we are working with and the id of the row we want to find and its data is returned into a string from the function. The selectAll function will return a matrix of strings equivalent to the table given as parameter, matrix which will later be used to create the actual table. The nbOfRows function will return and integer representing the number of rows of the class table we need.

*- ConnectionFactory.java*

The attributes of this class are the same as the header of the Order table: id, ClientID, ProductID and size. The size defines how many types of the product the client has bought. When calling the constructor of this class which creates a new Order object with the given parameters, so basically a new row in our table, we have to keep in mind that in the database the id field is the primary key of the table which means that its value of the id on each row must be unique so there won’t be two rows with the same id. The table representing this class was made in order to map the relation between the Client and Product table. The attributes of this class were declared private for security purposes so in order to access their data getter and setters methods have been made.

~ *model*

- *Cart.java*

- *Client.java*

The attributes of this class are the same as the header of the Client table: id, name and city. When calling the constructor of this class, which creates a new Client object with the given parameters, so basically a new row in our table, we have to keep in mind that in the database the id field is the primary key of the table which means that its value of the id on each row must be unique so there won’t be two rows with the same id.

- *Oreder.java*

The attributes of this class are the same as the header of the Order table: id, ClientID, ProductID and size. The size defines how many types of the product the client has bought. When calling the constructor of this class which creates a new Order object with the given parameters, so basically a new row in our table, we have to keep in mind that in the database the id field is the primary key of the table which means that its value of the id on each row must be unique so there won’t be two rows with the same id. The table representing this class was made in order to map the relation between the Client and Product table. The attributes of this class were declared private for security purposes so in order to access their data getter and setters methods have been made.

- *Product.java*

The attributes of this class are the same as the header of the Client table: id, name and age. When calling the constructor of this class, which creates a new Client object with the given parameters, so basically a new row in our table, we have to keep in mind that in the database the id field is the primary key of the table which means that its value of the id on each row must be unique so there won’t be two rows with the same id. Furthermore a restriction was made on the age of the clients in the validator package and as a consequence the age of each client must be greater than 8 and less than 80 years. The attributes of this class were declared private for security purposes so in order to access their data getters and setters methods have been made.

~ *presentation*

- *Instructions.java*

- *Reports.java*

*~ start*

*-Start.java*

*--Conclusion:*

Java offers the real possibility that most programs can be written in a type-safe language. ... It extends Java with a mechanism for parametric polymorphism, which allows the definition and implementation of generic abstractions. The paper gives a complete design for the extended language

I personally, learned and better understood how to use classes and modifiers (which sets the class as public or private), modifiers are also used for variables and methods and also I have my first interaction with the threads and that’s awesome, but in my opinion are kind hard to understand and to work with them, but I had a lot of fun and I like it, because who doesn’t like an tuff chalange.

What’s concerning the further developments, I have some ideas to adjust the interface to make it more attractive because in our days this is what an user is looking for, he or she has no concerns about how the application works or why it takes so much memory. Regarding to possible updates of the application, I have in mind to implement an algorithm which finds the roots of the polynomial and returns a graphical schema and also to increase performance so the application can work faster.

*--Bibliography:*

- <https://www.geeksforgeeks.org/>

- <https://www.wikipedia.org/>

- <https://stackoverflow.com/>