­­­Documentation

Assignment 3. Order Management System

Student: Potra Paul-Antonio

Group: 30421

# Design

This application has a layered architecture with the following packages: *businessLogic*, *connection*, *dataAccess*, *model* and *presentation*.

We’ll start with the *connection* package. It contains only one class, *ConnectionFactory*. Using the factory design pattern, this class ensures that the connection to the database connection is not opened more than once at one time. That is, if there is an active connection and some class wants to create a new one, it will just return the already existing connection. It also closes the *Connection*, *Statement* and *ResultSet* if they are not already closed.

The *model* package contains the classes which correspond to the tables in the database (Client, Product, Order and Bill, which is a record). Every instance variable in those classes represents a field (column) from the database table.

To access the database, I created a separate package, *dataAccess*. This contains only data access objects (DAO). One of them is the *AbstractDAO*, where basic CRUD methods that can be applied to every table in the database are implemented. By creating this generic class don’t need to write the same function for every table separately, but instead we have a generic one in a single place.



The implemented methods are finding an entry in the table by id, finding all entries in the table, insert an item in the corresponding table and deleting an item by id (assuming the id is the primary key, or at least unique).









The update operation was done individually for every class. Thus, I have implemented the following:

* a *ClientDAO* class with a method to update the name of the client given by id.
* a *ProductDAO* class with a method to update the name of the product and one to update the quantity (the stock).
* An *OrderDAO* class which overrides the insert method from the *AbstractDAO*. This was done because the order has auto incremented id in the database table, and thus the user does not need to insert an id when placing the order. This means that the id field must be considered when inserting an order in the database, which happens when using the *AbstractDAO* insert method.

The *businessLogic* package acts as an intermediate between the user (who interacts with the views) and the DAO objects. It is responsible for calling the methods from the DAO classes, and, in addition, it does some exception handling and null checks.

Finally, the presenter package contains the views the user interacts with (*MainView*, *ClientView*, *ProductView* and *OrderView*) and the *Controller* class, which handles the user interaction with the views and performs the necessary logic with the other classes.

One very important technique which was used on this application is reflection. To avoid implementing the basic CRUD operations for every object corresponding to a table in the database, I implemented the *AbstractDAO* class. This is a generic class, so it must work with any objects, no matter what the fields are. To implement this, reflection was used. It allows us to have access to an object of a class and access its fields. The names of those fields correspond with the columns in the database tables, and the name of the table is the name of the class. In this project, reflection wasn’t necessary because the database is small, containing only a few tables. But in a real-world scenario, where databases are large and the tables have a lot of columns, it is practically impossible for one to remember all the fields of all the objects. Thus, reflection can help a lot by creating queries which fit any type of objects, by just accessing the name of their fields. On top of that, when receiving the result set from executing the query, I needed to extract the actual objects from those rows. This was also done through reflection.