ORDER MANAGEMENT

Documentation

**Vlad Lujerdeanu**

Group 30226 | Year 2 | Semester 2

# Contents

1. [The Objective](#_The_Objective) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
2. [Analysis, Modeling and Use Cases](#_Analysis,_Modeling,_Scenarios) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
3. [Designing Phase](#_DesignING_Phase) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
4. [Implementation](#_Implementation) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
5. [Results](#_Results) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
6. [Conclusion](#_Conclusions) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
7. [References](#_References) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

# The Objective

The Objective of this project is to design and implement a Management system for processing customer orders for a warehouse. The management system uses relational databases to store the information about clients, orders and products.

The design and implementation process of this project was as follows:

* Designing and implementing the model classes, namely the Client, Product, Order and Order Items classes.
* Designing and implementing the database connection class.
* Designing and implementing database abstract query class, which is the foundation for building the other database query classes.
* Implementing the other query classes which extend the abstract class. These classes implement specific functionalities for each object: Client, Product, Order and Order Items.
* Implementing the business logic classes, which control the functionalities implemented in the query classes based on conditions described in the Assignment brief.
* Designing and implementing the reports for clients, orders and products.
* Designing and implementing the receipt system and the out of stock report.
* Implementing the Main Class.
* Building the project as a jar file.
* Testing the complete project and solving the bugs.

# [Analysis](#_Analysis), [Modeling](#_Modeling) and [Use Cases](#_Use_Cases)

### Analysis

A management system is a collection of functionalities working together used by an organization to ensure that it can fulfill the tasks required to achieve its objectives. These objectives cover many aspects of the organization's operations including safe operation, product management, client relationships and worker management.

### Modeling

The application will be receiving data from an input text file given as the first argument when running the jar file. The input file should be a document .txt file and the commands should be given in order from top to bottom. The commands available are illustrated below in use cases.

The program executes the commands and outputs success or error messages, depending on the case. If error messages should occur, the command should be checked and re-entered properly.

### Use Cases

1. The program should be started with the following command:

Java -jar PT2020\_30226\_Vlad\_Lujerdeanu\_Assignment\_3.jar <file.txt>

File.txt should be replaced with the path to the file containing the commands to be executed.

1. The commands available are:
   * Insert client: < First Name > < Last Name >, < City >
   * Insert product: < Name >, < Quantity >, < Price >
   * Delete client: < First Name > < Last Name >, < City >
   * Delete product: < Name >
   * Order: < Client’s First Name > < Client’s Last Name >, < Product Name >, < Quantity >
   * Report client
   * Report product
   * Report order

# DesignING Phase

### Design Decisions

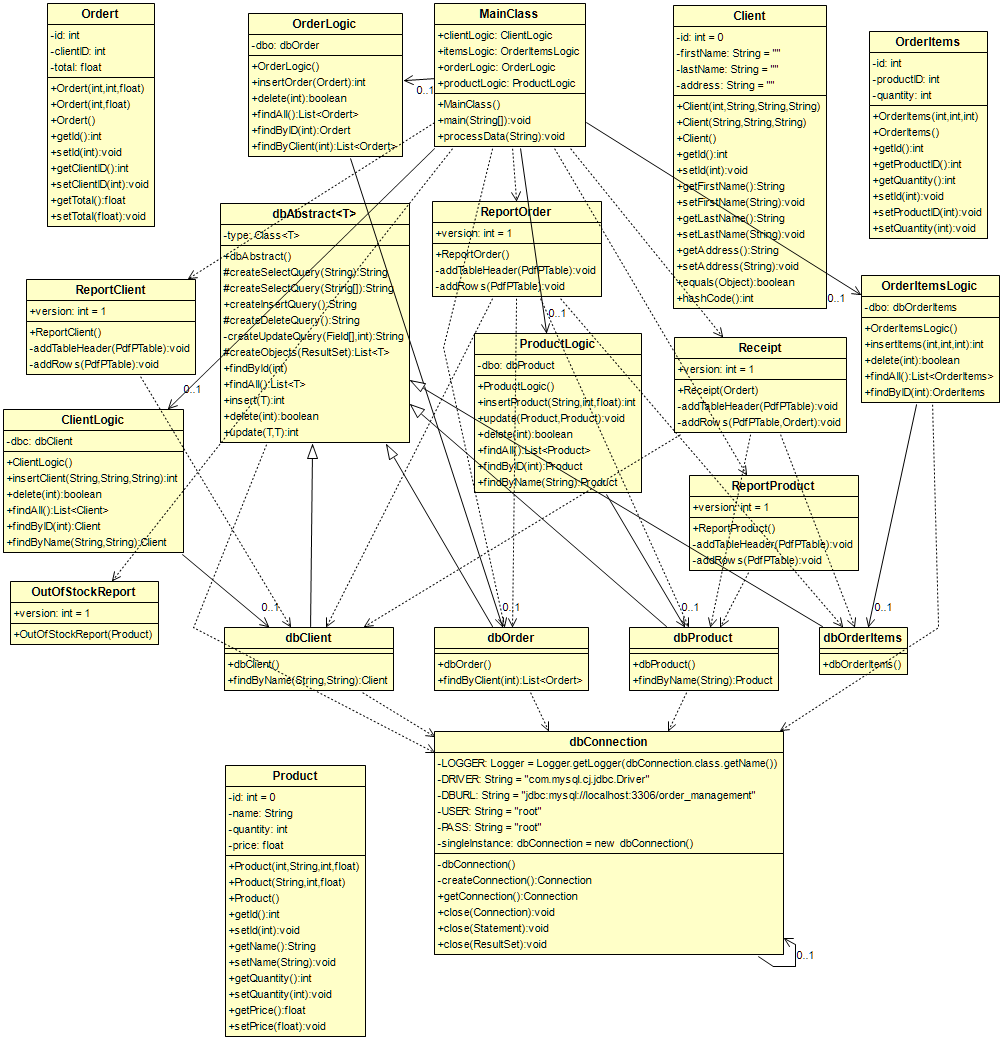
Designing process for this project was a top-down one, and the implementation was bottom-up, firstly implementing the Client, Product, Order and Order Items models and creating the database connection. Secondly, designing and implementing the query abstract class which is the base for the other query classes, and then the other query classes. After that, adding the business logic classes and presentation classes. Finally, using them in the main class where the processing of commands takes place.

Firstly, the application expects the input to be a text file given as the first argument when running it and the results are given as output in the terminal, either by displaying success messages or error messages or by PDF files.

Another design decision is following an layered architecture for separating the output generation, query classes, business logic and the models.

An important design decision is using reflection techniques to create a generic class that contains methods for accessing the database. The methods implemented in the generic class are: create an object, update, delete and find.

### UML Diagram



### Class Design

#### Client, product, order, order items:

These objects store the information extracted from the database.

They have getters and setters for each variable, as well as constructors for different use cases

#### db connection:

This class is implemented using the java.sql library and it creates and manages the connection between application and the database.

It also has close methods for Statements and Result Sets as well as for the Connection itself.

#### db abstract:

This class uses the reflection technique for building a generic class that contains the method for accessing the database.

The functionalities implemented by this generic class are inserting into the database, updating information from the database, finding information in the database by ID, finding all the information in a given table and deleting an object from the database.

#### db client, db product, db order, db order items:

These classes extend the DB ABSTRACT class and implement other functionalities than the ones implemented in the generic class, each functionality being needed only for that particular class.

The DB CLIENT implements a find by name method which returns a Client object if the name appears in the database, else it returns null.

The DB PRODUCT also implements a find by name method but it only uses a single name, not a first name and a last name. This method searches in the Product table for an item corresponding to the name given as argument. It returns a Product object if successful and it returns null if unsuccessful.

The DB ORDER implements a find by client method which searches all the orders for a specified client id and it returns a list of Order objects if successful, else it returns null.

#### Client logic, product logic, order logic, order items logic:

These classes implement the business logic of the query classes, as well as they check for the conditions given in the assignment brief to be fulfilled.

#### Report client, report product, report order:

These classes implement output functionalities.

They create PDF files with all the information from the tables.

#### receipt and out of stock report:

The receipt class implements a confirmation that the user places the order successfully, while the out of stock report is implemented to let the administrators know that the product is running low on supply and an order couldn’t be fulfilled due to insufficient stock.

# Implementation

#### Client:

public class Client {  
 private int id = 0;  
 private String firstName = "";  
 private String lastName = "";  
 private String address = "";

public Client(int id, String firstName, String lastName, String address) { . . . }   
public Client(String firstName, String lastName, String address) { . . . }  
public Client() { . . . }  
public int getId() { . . . }  
public void setId(int id) { . . . }  
public String getFirstName() { . . . }  
public void setFirstName(String firstName) { . . . }  
public String getLastName() { . . . }  
public void setLastName(String lastName) { . . . }  
public String getAddress() { . . . }  
public void setAddress(String address) { . . . }

}

#### PRODUCTS:

public class Product {  
 private int id = 0;  
 private String name;  
 private int quantity;  
 private float price;  
public Product(int id, String name, int quantity, float price) { . . . }

public Product(String name, int quantity, float price) { . . . }  
public Product() { . . . }  
  
 // GETTERS AND SETTERS FOR EVERY VARIABLE  
}

#### ORDER:

public class Ordert {  
 private int id;  
 private int clientID;  
 private float total;  
public Ordert(int id, int clientID, float total) { . . . }  
public Ordert(int clientID, float total) { . . . }  
public Ordert() { . . . }

// GETTERS AND SETTERS FOR EVERY VARIABLE

}

#### ORDER ITEMS:

public class OrderItems {  
 private int id;  
 private int productID;  
 private int quantity;  
public OrderItems(int id, int productID, int quantity) { . . . }

public OrderItems() { . . . }

// GETTERS AND SETTERS FOR EVERY VARIABLE

}

#### db connection:

public class dbConnection {  
 private static final Logger *LOGGER* = Logger.*getLogger*(dbConnection.class.getName());  
 private static final String *DRIVER* = "com.mysql.cj.jdbc.Driver";  
 private static final String *DBURL* = "jdbc:mysql://localhost:3306/order\_management";  
 private static final String *USER* = "root";  
 private static final String *PASS* = "root";  
  
 private static dbConnection *singleInstance* = new dbConnection();  
  
 private dbConnection() { . . . }  
private Connection createConnection() { . . . }  
public static Connection getConnection() { . . . }  
public static void close(Connection conn) { . . . }  
public static void close(Statement statement) { . . . }  
public static void close(ResultSet resultSet) { . . . }  
}

#### db abstract:

public class dbAbstract<T> {  
  
 private final Class<T> type;  
@SuppressWarnings("unchecked")  
 public dbAbstract() { . . . }  
protected String createSelectQuery(String field) { . . . }  
protected String createSelectQuery(String[] fields) { . . . }  
public String createInsertQuery() { . . . }  
protected String createDeleteQuery() { . . . }

private String createUpdateQuery(Field[] fields, int id) { . . . }

protected List<T> createObjects(ResultSet resultSet) { . . . }

public T findById(int id) { . . . }

public List<T> findAll() { . . . }  
public int insert(T object) { . . . }

public boolean delete(int id) { . . . }

public int update(T object, T newObject) { . . . }  
}

#### DB CLIENT:

public class dbClient extends dbAbstract<Client>{  
  
 public dbClient(){  
 super();  
 }  
public Client findByName(String firstName, String lastName) { . . . }  
}

#### DB ORDER:

public class dbOrder extends dbAbstract<Ordert>{  
  
 public dbOrder(){  
 super();  
 }  
public List<Ordert> findByClient(int id) { . . . }

}

#### db products:

public class dbProduct extends dbAbstract<Product>{  
  
 public dbProduct(){  
 super();  
 }  
public Product findByName(String name) { . . . }

}

# Results

The project has been tested using “manually” by testing first each classes’ functionalities. The completed project has been tested using different input test files and checking the output files for logical errors.

# Conclusions

To conclude, the project presented above taught me a lot about analyzing a problem and designing a solution to fit a set of strict requirements and also how to work with a database connected to an application, layered architecture and reflection techniques, how but most importantly, how frustrating it is to build jar files…

# References

1. [Laboratory Guide – Ionel Giosan](http://users.utcluj.ro/~igiosan/teaching_poo.html) ( UML Generating in eclipse and Javadoc creation )
2. [Indications](http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/Assignment_3/Assignment_3_Indications.pdf) ( Layered Architecture and Reflection )
3. [Stackoverflow.com](https://Stackoverflow.com) ( Minor problems with the code )