Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Year of study: 2017-2018

Order management system

- Assignment 3 -

Ionuț Matiș

Group: 30421

Table of contents

1. Objective …………………………………………………………………. 3
2. Problem analysis, modelling, use cases ………………………………….. 3
3. Analysis ………………………………………………………………. 3
4. Assumptions ………………………………………………………….. 3
5. Use case diagram ……………………………………………………... 4
6. Scenarios ……………………………………………………………… 4
7. Errors …………………………………………………………………. 5
8. Design ……………………………………………………………………. 5
9. Decisions …………………………………………………………….. 5
10. Class Diagram ……………………………………………………….. 5
11. Data structures ………………………………………………………. 6
12. Class design …………………………………………………………..6
13. Relationships ………………………………………………………… 7
14. User interface …………………………………………………………7
15. Database diagram ……………………………………………………..8
16. Error handling …………………………………………………………8
17. Implementation ……………………………………………………………9
18. Results ……………………………………………………………………14
19. Conclusions and future developments ……………………………………15
20. Bibliography ………………………………………………………………15

# 1.Objective

The main objective of this project is to create an application which does process customer orders for a warehouse. **Relational databases** are used to store the products, the clients and the orders.

The secondary objectives are:

1. Create the required tables in a database program (MySQL Workbench will be used)
2. Create the model which contains classes mapped to the database
3. Build the data access layer which contains the classes containing the queries and the
4. database connection
5. Build the business logic layer which contains the classes that encapsulate the application logic
6. Build the presentation layer which contains the classes defining the user interface

# 2.Problem analysis, assumptions, use cases

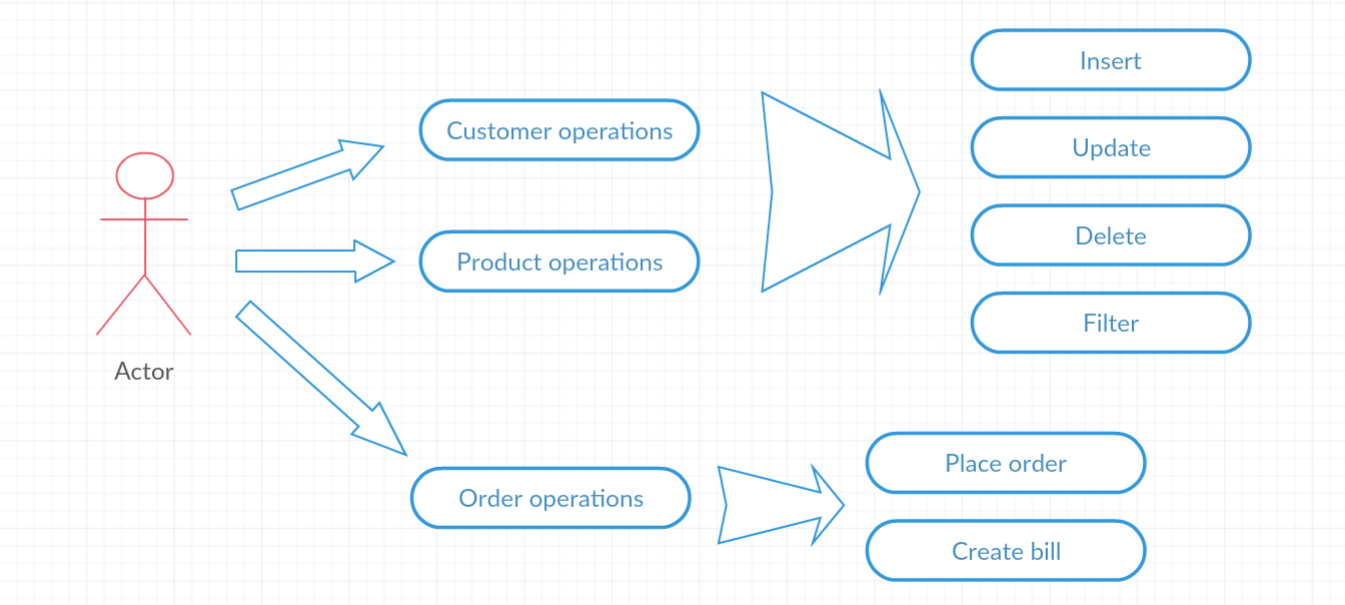
2.1 Analysis

When we think about databases, we think about a way to store items with similar characteristics. These items are stored inside a table which has various fields. One of them is the PK (primary key), which is a unique identifier for each one of the items stored. Large databases have a huge number of tables, but in this assignment I will use only 3 (Customer, Order, Product). The “Order” table is used here to resolve the many to many relationships between the “Customer” table and “Product table” (one customer can buy many products and the same product can be bought by various customers.)

2.2 Assumptions

We assume that the input inserted by the user is correct. This means that the primary keys are integers greater than 0, the names of the customers / products contain letters, the email of the customer has a specific structure (ex: [name@email.com](mailto:name@email.com)), the amount and the price of a product are integers greater than 0.

2.3 Use Case diagram



From the use case diagram, we can see that the user can select 3 types of operations (on customers, on products and on orders). If he chooses to work on the first two, he is allowed to perform various actions on the database tables (insert, update or delete a client / product). If he chooses to filter, he can do it based on the fields from the table (ID, Name of the product / client etc).

If he chooses to work on orders, he can create a new one (by selecting a customer, a product and the amount) or he can generate a bill for each order.

2.4 Scenarios

First scenario (**success**):

1. the user selects both customer and product operations.
2. two windows will open and he chooses to insert a new customer by inserting valid values for it (ID, Name, Email, Age).
3. he also goes in the product window and deletes the first product.
4. after that, he desires to filter the products based on their price. He introduces the filtering options and he will be given a list with the filtered products that match the filtering selection.

Second scenario (**success**):

1. the user selects the order operations.
2. a window opens, and he can now choose which customer orders which product.
3. after this, the user inserts a valid amount of product (greater that 0, but less than the total amount of product)
4. the users can see all the orders of the application.
5. after doing this, he decides to create bills for each order. He succeeds this action
6. Now he has a text file with information about the customer, the product and the price for each order in the application.

Third scenario (**fail**):

1. the user selects the order operations.
2. A window opens, and he can now choose which customer orders which product.
3. After this, the user inserts a valid id of order (an id which was not taken by anyone)
4. He inserts a negative amount of product, which will cause an error and he will be warned

2.5 Errors

In my implementation, there are 2 types of errors. The first type of errors are the ones which appear when the user inserts invalid data (the user fails to insert integers or does not insert anything at all in places such as ID, Age, Amount or Price).

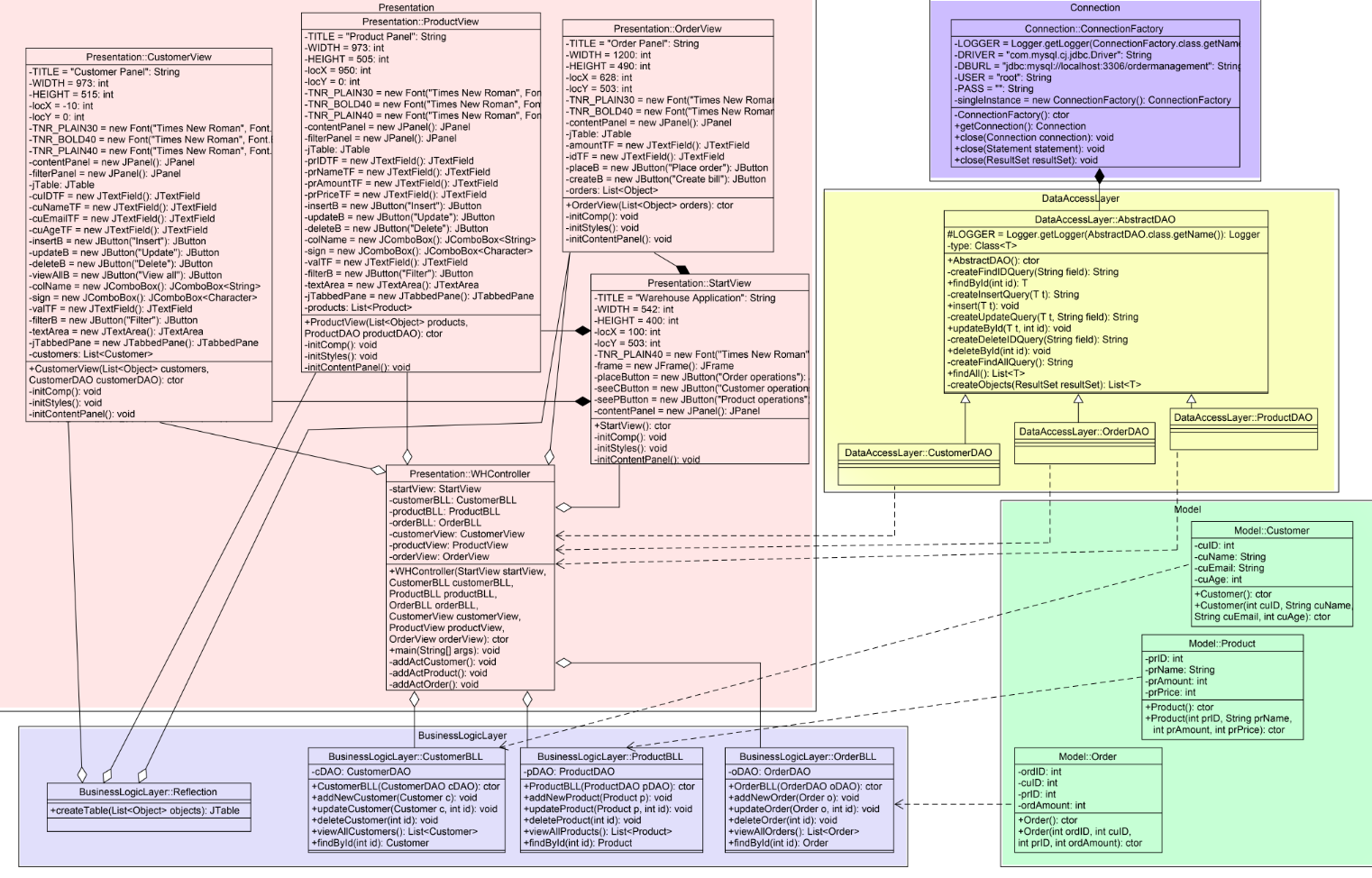
The second type of errors appear when the user inserts values that are not allowed by the database system. For example, the user tries to insert an element with an ID that already exists in the database or tries to delete or update an element that does not exist in the database at all.

# 3.Design

3.1 Decisions

The application has a layered architecture, each layer having a special purpose and calls functions of the layers below it. Model – contains classes mapped to the database table Presentation Layer - contains the classes defining the user interface. Business Layer – contains the classes that encapsulate the application logic. Data Access Layer – contains the classes containing the queries and the database connection.

3.2 Class Diagram



3.3 Data structures

As data structures, the List interface is highly used. It is an ordered Collection (sometimes called a sequence). Lists may contain duplicate elements. In addition to the operations inherited from Collection, the List interface includes various operations such as Positional access, Search, Iteration, Range-view.

Besides List interface, I have also used the ArrayList to store various type of elements. This class is a resizable-array implementation of the List interface. Implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list.

3.4 Class design

For the design of the implementation, an OOP design was chosen. Moreover, a layered architecture was built so that the project is well structured.

**Model package:** contains the 3 classes (Customer, Product, Order) that are also mapped on the database. Each of the class contains specific fields, 2 constructors (one empty so that it can be used in the reflection and one which initializes all the fields), getters and setters.

**Connection Package:** contains the Connection factory. This class contains the name of the driver (initialized through reflection), the database location (DBURL), and the user and the password for accessing the MySQL Server

**Data Access Layer Package:** contains the generic repository which can be mapped on any table from a database. It is implemented by 3 other classes (CustomerDAO, ProductDAO, OrderDAO), but these are empty classes (the generic parameter T is given as Customer, Product and Order respectively, so all the work is done in the superclass.)

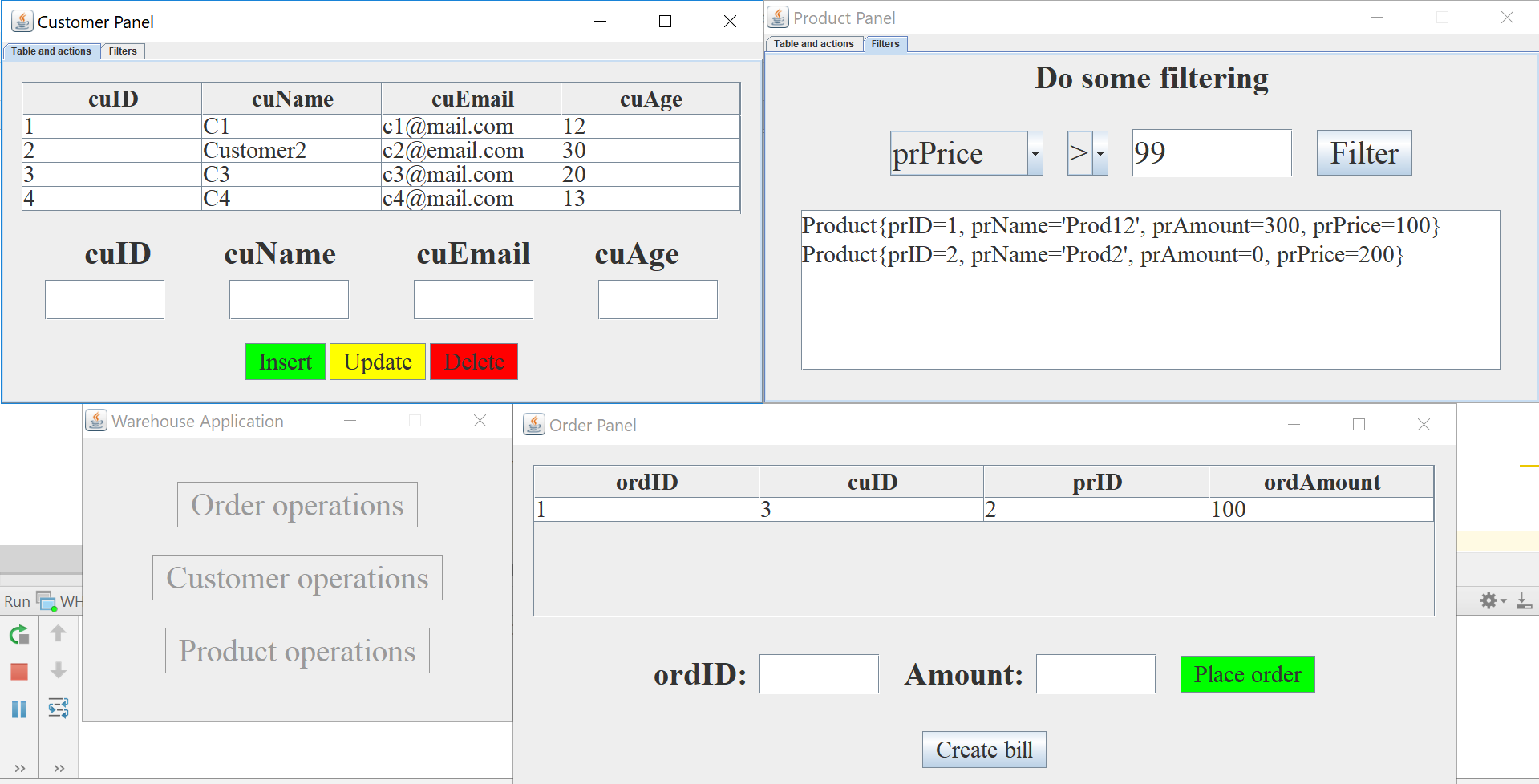
**Business Logic Layer Package:** contains the classes that encapsulates the logic of the application. These are: ProductBLL, CustomerBLL, OrderBLL and also a class that uses reflection techniques to create a method that receives a list of objects and generates the header of the table by extracting through reflection the object properties and then populates the table with the values of the elements from the list.

**Presentation Package:** contains the graphical user interface classes (4 in total) and the controller. The main GUI class is the StartView, which opens as the user runs the program. From it, the user can select 3 different operations (customer operations, product operations and order operations). Based on what he chose, a new window will pop up and the user can perform various actions on the tables (insertion, update, deletion, filtering and even creating bills for the orders.)

3.5 Relationships si de ce

* CustomerDAO, ProductDAO, OrderDAO extend AbstractDAO
* AbstractDAO – ConnectionFactory: composition
* Customer – CustomerBLL: dependency (same for Model – BLL)
* ProductView – StartView: composition
* CustomerView – StartView: composition
* OrderView – StartView: composition
* ProductView – Reflection: aggregation
* CustomerView – Reflection: aggregation
* OrderView – Reflection: aggregation
* WHController has aggregation with all presentation classes, all business logic classes and dependency with all model classes and data access layer classes

3.6 User Interface



This pictures how the user interface looks. The top windows have almost the same structure. The first tab contains one JTable at the top, 4 label, 4 text fields and 3 buttons, whereas the second one contains a label, 2 combo boxes, one text field, one button and one text area. The order panel does not differ much either. It also has a JTable, 2 labels, 2 text fields and 2 buttons (the “Create bill” button creates the bills for all orders.)

3.7 Database diagram

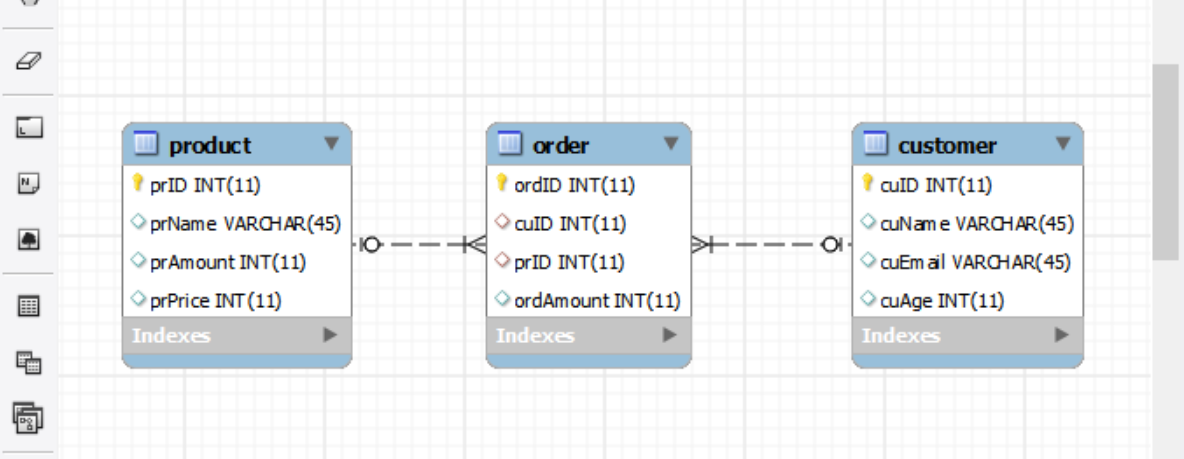
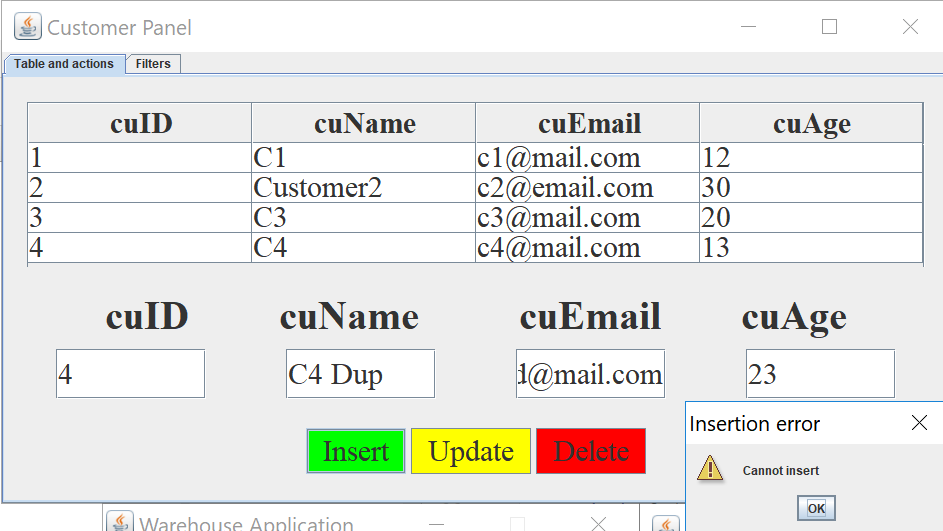


Diagram that shows how I resolved the many to many relationship.

3.8 Error handling

Error handling is done by using JOptionPanes which display a message in case the user wants to insert invalid data or does not respect the database rules. This is very intuitive and resolves any input problem a user might encounter.

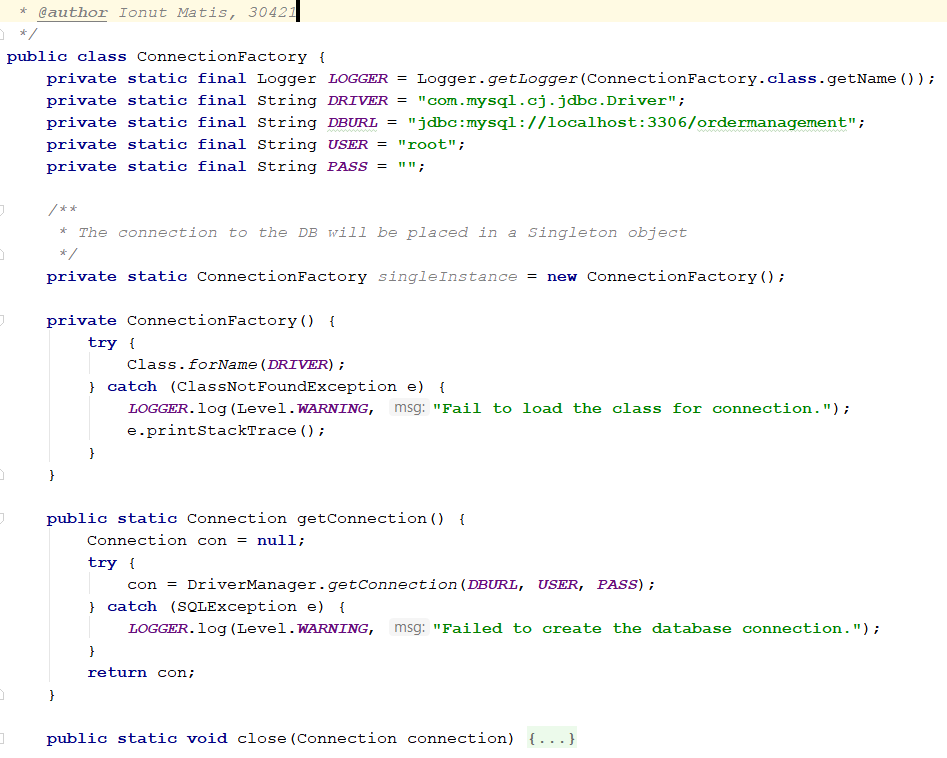


# 4.Implementation

1. **Customer, Product, Order**

The classes from the model package are just basic classes with fields and 2 constructors (one of them is empty.) They also provide setters, getters and a toString() method.

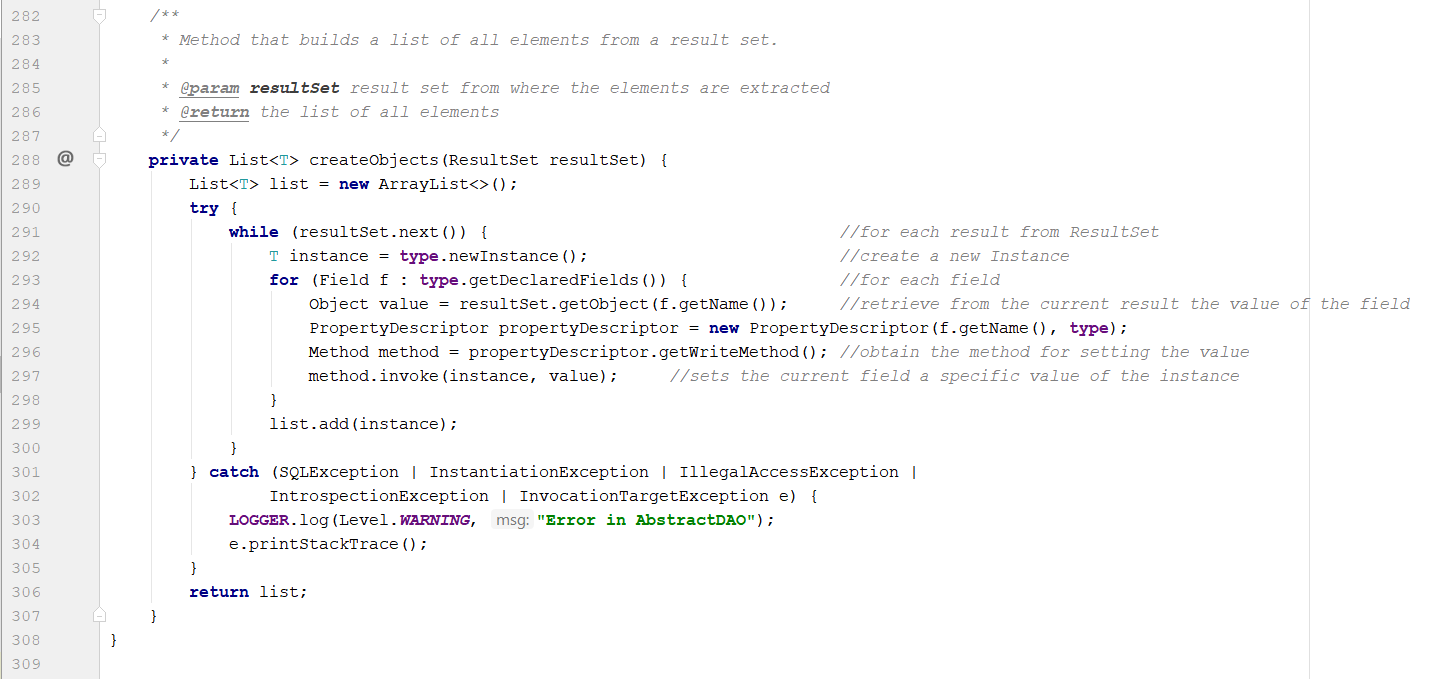
**2.ConnectionFactory**



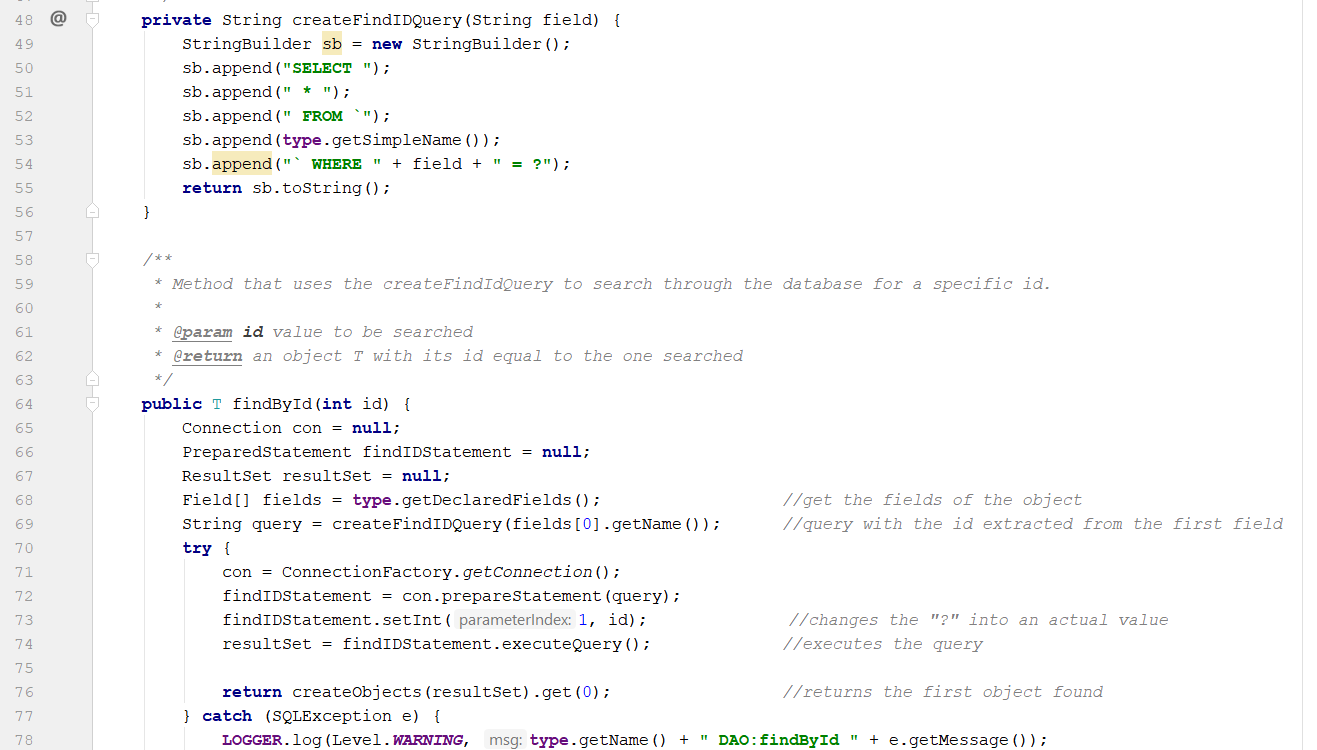
This class contains the name of the driver (initialized through reflection), the database location (DBURL), and the user and the password for accessing the MySQL Server. The connection to the DB will be placed in a Singleton object. The class contains methods for creating a connection, getting an active connection and closing a connection, a Statement or a ResultSet.

**3.AbstractDAO<T>**

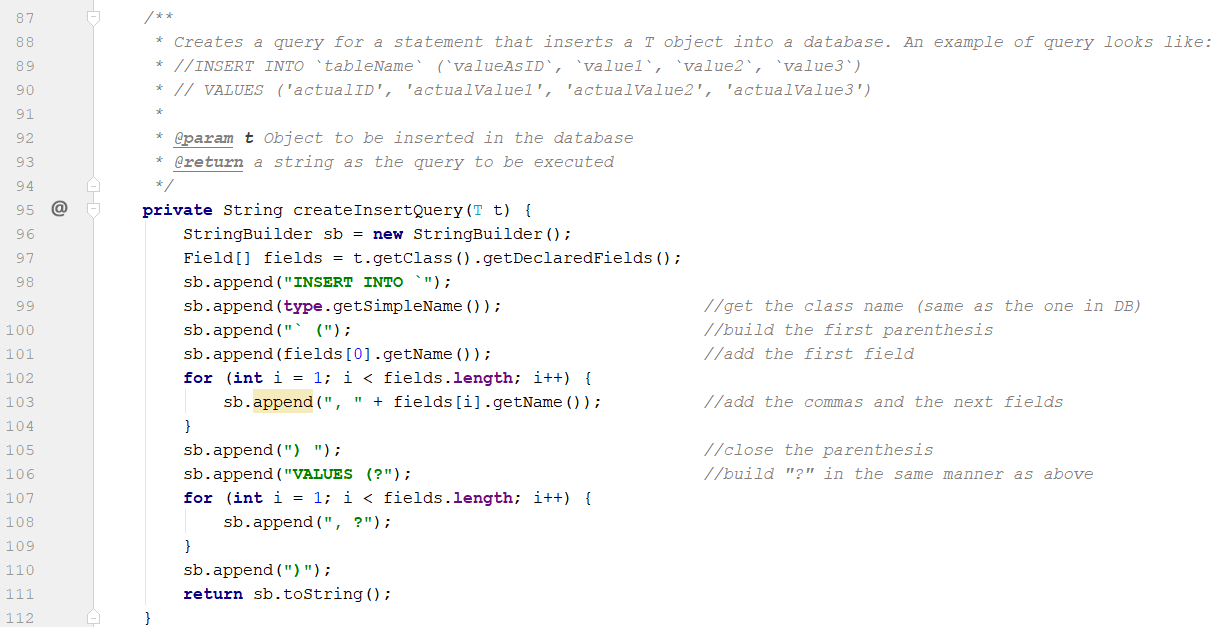
This class is one of the most important ones in this project. It is so because it provides methods for querying any type of table from a database (using a generic T and reflection methods). It provides methods for Inserting, Update, Deletion, Finding by a specific ID and Finding all values. It does every action in 2 steps. Firstly, it creates a statement and places “?” where it needs to insert a specific value or a set of values and then it executes it.



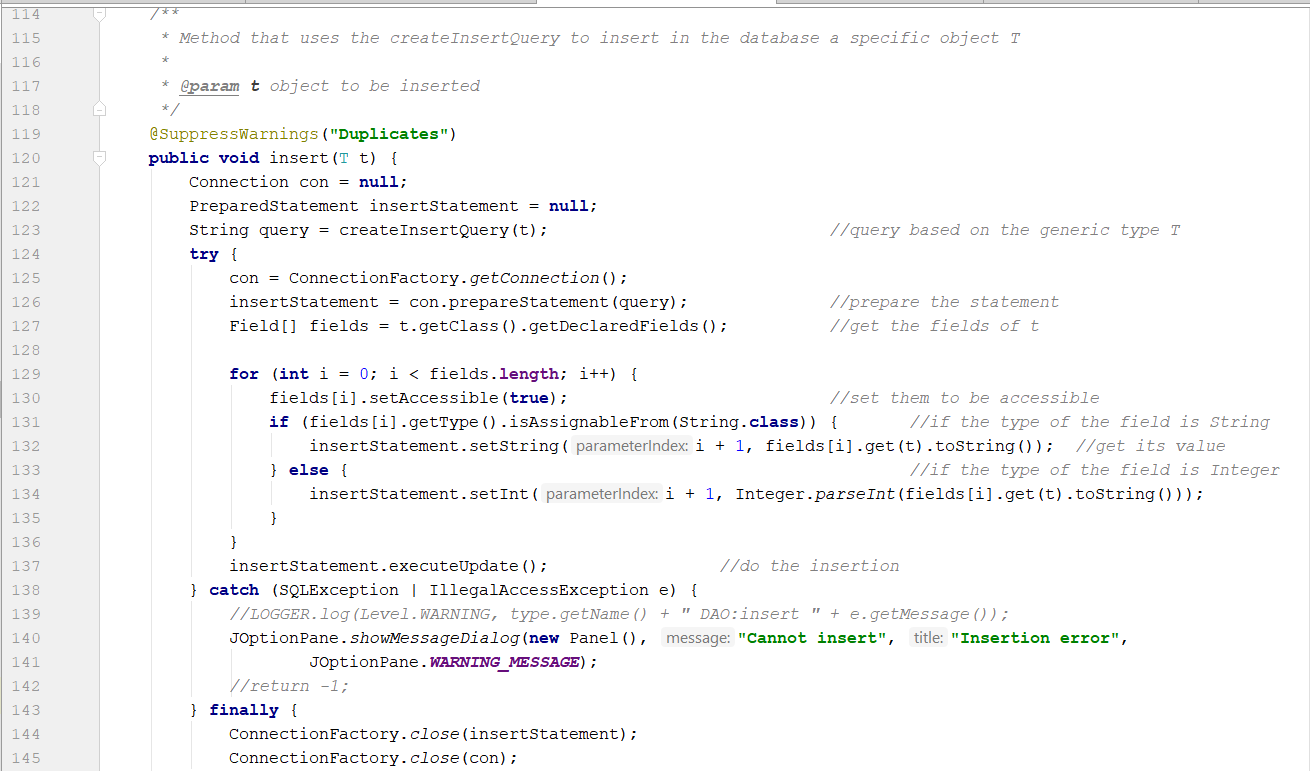
One important method here is this one. It creates through reflection, based on a result set, a list of objects. In the line 292, it is invoked a new empty instance on the type and, without an empty constructor, the method would have failed. For each field of the type, it gets its name and then uses the invoke method to write the value of that specific field. It basically creates an empty object and then builds it up using setters provided by the property descriptor and invoke methods.



In the above picture, it creates a selection query (for example: “SELECT \* FROM `Customer` WHERE ID = ?”) using createFindIDQuery method and then it calls it (line 69). It changes the “?” into an actual value using the predefined setInt method and then it returns the first object created by the createObjects(method)

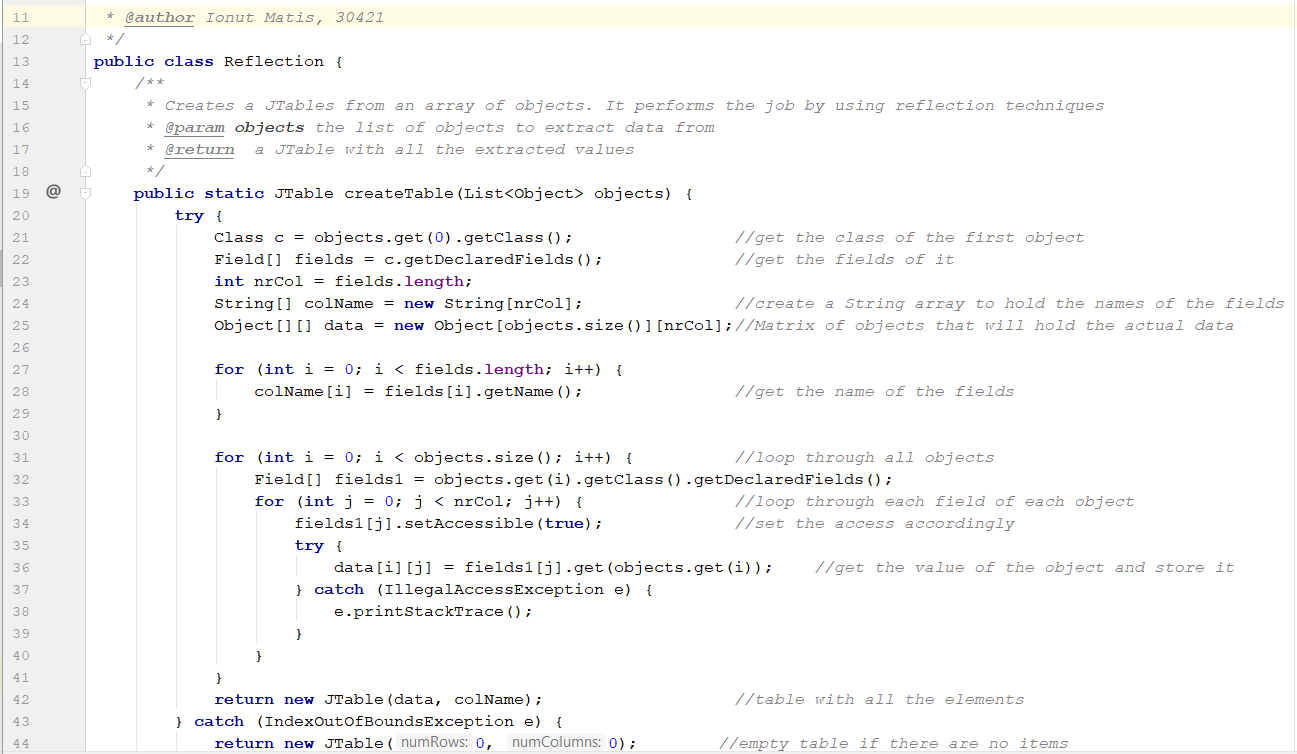


The above method creates an insert query. Based on the reflection techniques, it builds a string as the one in the lines 89, 90. Even though the example has 4 fields, it works for any type of table.



The above method I found the most challenging one, because I had to loop through all fields, extract the value from the object (it checks whether the value is a String or an Integer) and then sets the specific “?” to that extracted value.

**4. Reflection**



The reflection class consists only of a static method 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. If the list has no elements, an IndexOutOfBounds exception will be thrown and an empty table will be created (in the catch block, line 44).

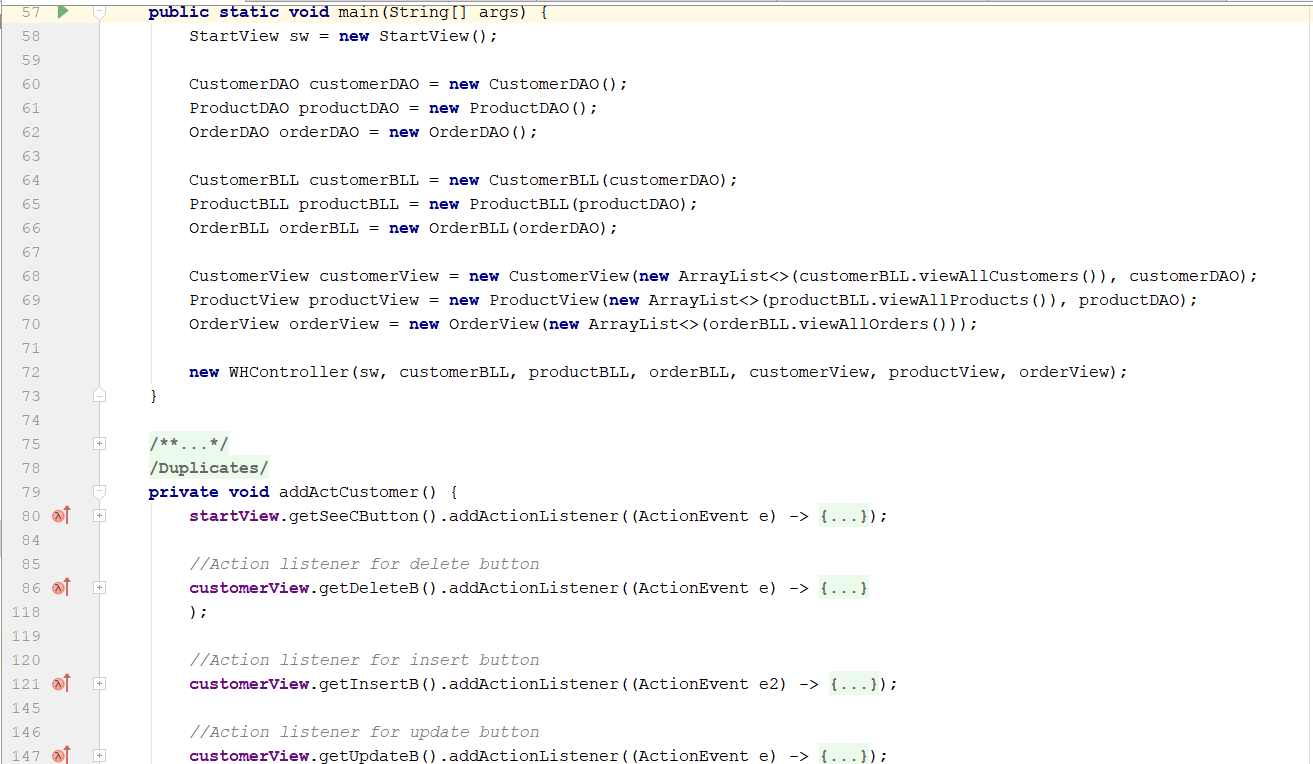
**5. ProductBLL, CustomerBLL, OrderBLL**

They offer an additional layer on top of the DAO classes (just call methods from them accordingly)

**6. StartView, ProductView, CustomerView, Orderview**

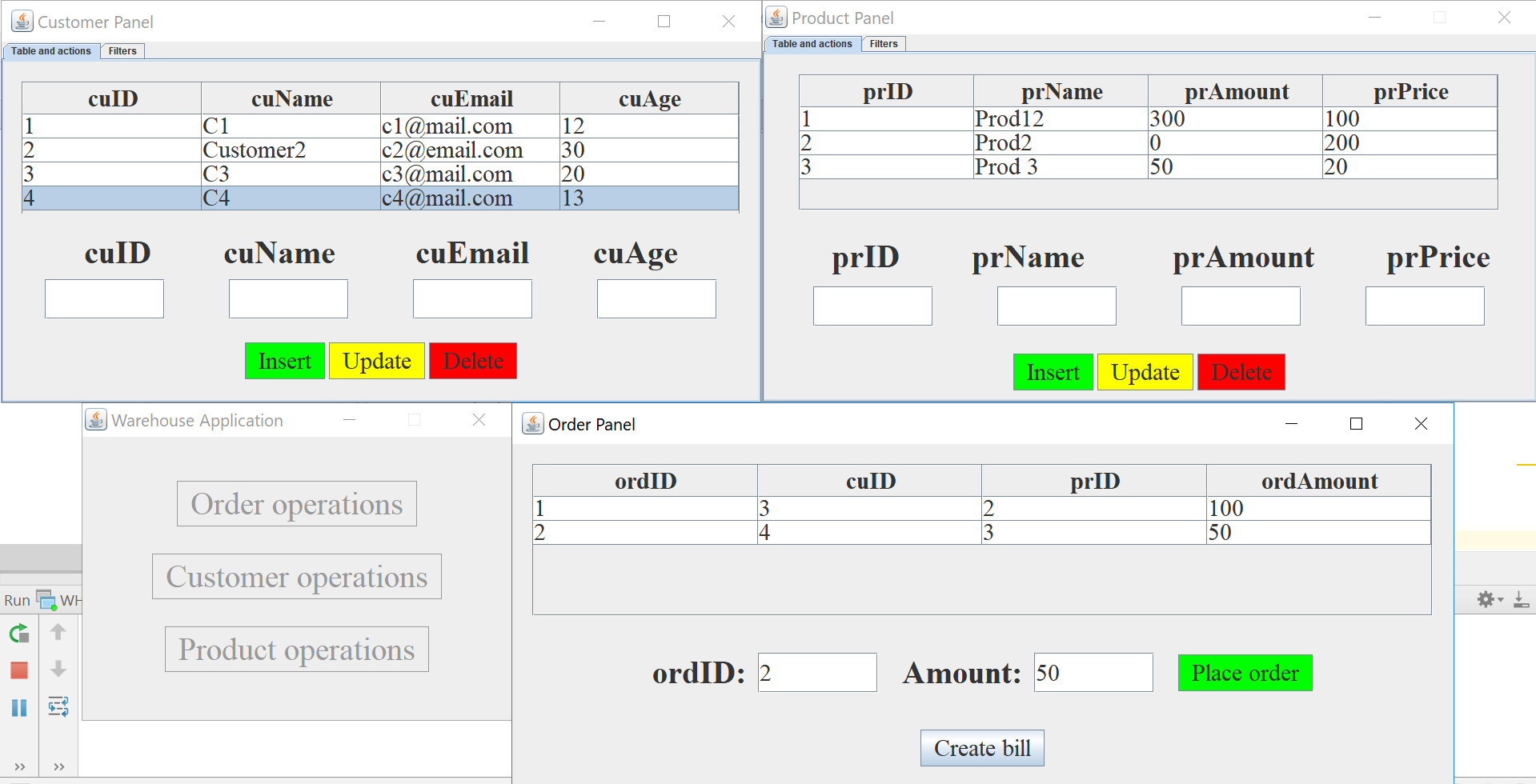
All those classes have methods for setting the JTable, the styles and the specific components on the panel. The last 3 of them have a method that removes all elements from the frame, reloads the components and then repaints and revalidates. This methods is very important if you want to see the tables updating in real time.

**7.WHController**

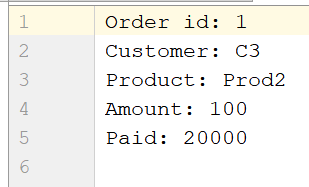


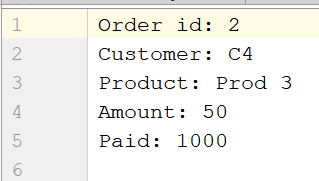
This class contains the main method and adds action listeners to all the buttons of the application. The application starts at the line 72 by invoking a new WHController on specific arguments.

# 5.Results



Insert a new order and then create bills for them.





# 6.Conclusions and further developments

In conclusion, the main and secondary objectives were met. The application works as intended. I have learnt of new things working on this assignment: how to connect to a database and execute queries on it, how to use reflection techniques, how to use generics and how to create a JTable.

Further developments: create the GUI using reflection techniques and implement a correct database structure (more than 3 tables)

# 7.Bibliography

1. <https://stackoverflow.com/questions/2262322/howto-find-out-if-a-field-is-instanceof-a-type-via-reflection>
2. <https://stackoverflow.com/questions/3949260/java-class-isinstance-vs-class-isassignablefrom>
3. <https://www.youtube.com/watch?v=wniqpx8OQxo>
4. <https://docs.oracle.com/javase/tutorial/uiswing/components/tabbedpane.html>
5. <https://stackoverflow.com/questions/5153496/how-can-i-compare-two-strings-in-java-and-define-which-of-them-is-smaller-than-t>
6. <https://stackoverflow.com/questions/2885173/how-do-i-create-a-file-and-write-to-it-in-java>