Warehouse management

UTCN CTI ENGLISH

Muresan Daniel, 30422

Contents

1. Goal of the application

Using a database to manage the flow of products, customers and orders in a warehouse may be a good idea for an owner of such a warehouse, in order to keep track of his business.

This application is meant to simulate a system which would help an owner of a warehouse to efficiently manage the business. The application together with a database developed in MySQL Workbench forms a relative easy way to keep track of the products, the flow of customers and the others. This application should also be able to generate a bill in PDF format for every new order placed and introduced in the database.

The application will give the possibility to manage separately the customers, products and orders respectively. The application will also have an easy-understandable graphic user interface with intuitive buttons, labels and text fields. All those will make it easy to use for any user regardless the experience and the knowledge about databases.

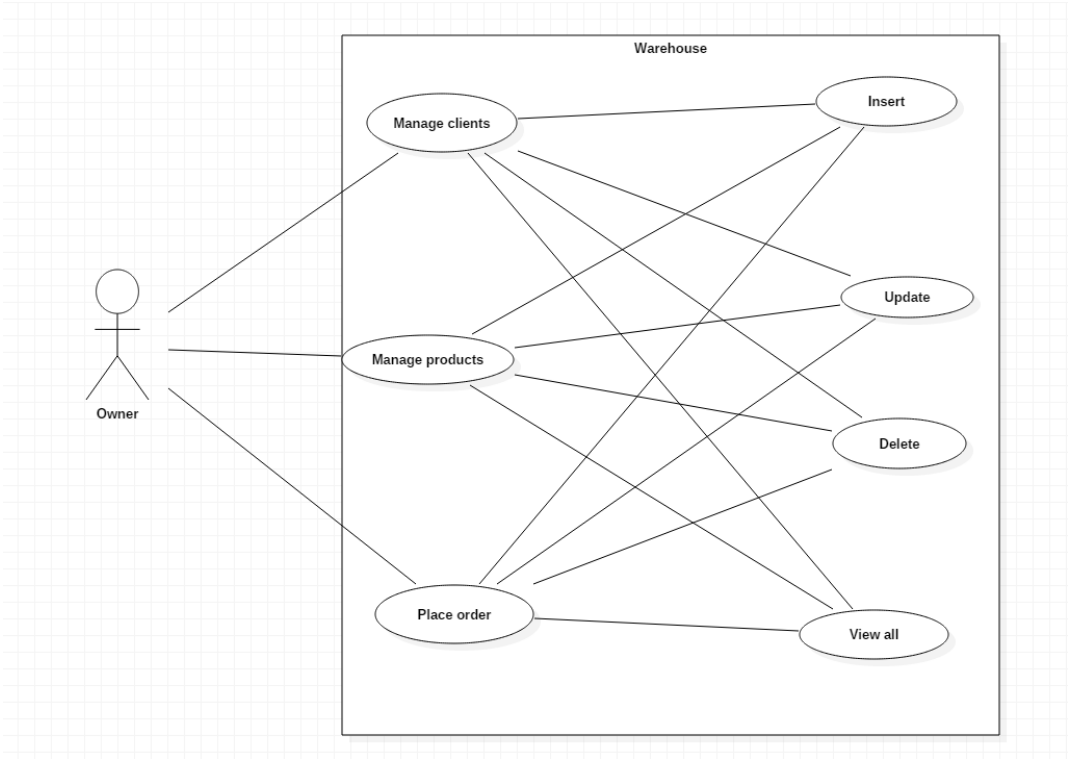
2. Problem analysis

Depending on the data it wants to manage, the application must be able to receive some information about a product, customer or order and then give the possibility for the user to choose an operation to apply on that specific data like adding into the database, updating some entries from the database, deleting some other entries form the database or getting all the entries from a specific table.

The assumptions that I made for this application are the following ones: every product, customer and order respectively are uniquely identified by an ID which can't have duplicates and the user must introduce

the data for every product, customer and order with respect to the unique ID which cannot be modified. Because of this assumption I will use only the ID for searching, deleting, inserting and updating the entries from the database.

The use cases of the application are presented in the next use case diagram.



- **Use case : Manage clients**
- **Primary actor: Owner**
- **Main success scenario:**
    - The user presses the button for managing clients
    - A new frame for managing clients appear and from there the user choses one of the options : insert, update, delete, view all which are also use cases

- **Use case : Manage products**
- **Primary actor: Owner**
- **Main success scenario:**
    - The user presses the button for managing products
    - A new frame for managing products appear and from there the user choses one of the options : insert, update, delete, view all which are also use cases

- **Use case : Manage orders**
- **Primary actor: Owner**
- **Main success scenario:**
    - The user presses the button for managing orders
    - A new frame for managing order appear and from there the user choses one of the options : insert, update, delete, view all which are also use cases

- **Use case : Insert**
- **Primary actor: Owner**
- **Main success scenario:**
    - The user introduces data about a certain product, customer or order, depending on the case, and the presses the button for insert.
    - The application takes data from the text fields and checks if the input is valid and all fields are filled
    - The application checks if the entry is not already in the table and then introduces it into the database
    - The application provides a success message if there were no errors during the process.

- **Use case : Update**
- **Primary actor: Owner**
- **Main success scenario:**
    - The user introduces data about a certain product, customer or order, depending on the case, and the presses the button for update.
    - The application takes data from the text fields and checks if the input is valid and all fields are filled.
    - The application checks if the entry is in the table and then updates the entry from the database with the new values provided by the user.
    - The application provides a success message if there were no errors during the process.

- **Use case : Delete**
- **Primary actor: Owner**
- **Main success scenario:**
    - The user introduces data about a certain product, customer or order, depending on the case, and the presses the button for delete.
    - The application takes data from the text fields and checks if the input is valid and all fields are filled.
    - The application checks if the entry is in the table and then deletes the entry from the database.
    - The application provides a success message if there were no errors during the process.

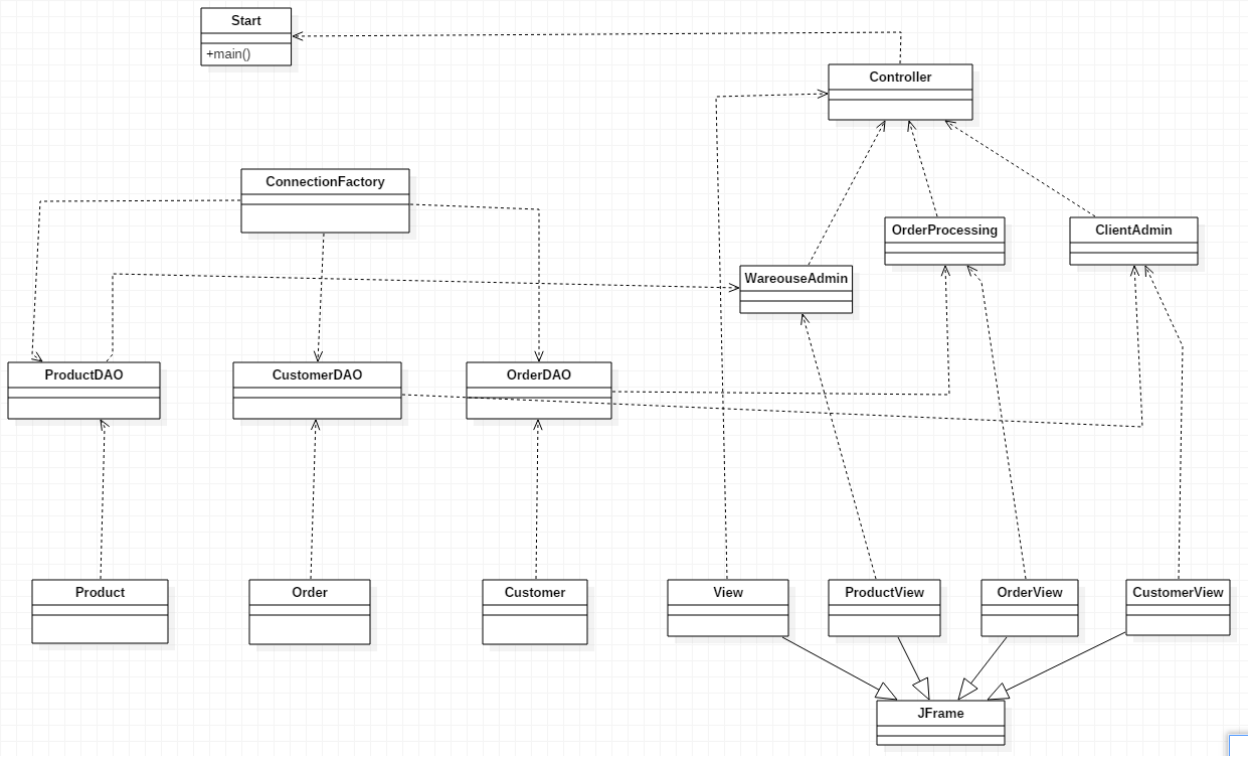
- **Use case : View all**
- **Primary actor: Owner**
- **Main success scenario:**

- The user presses the button for getting all data from a table from the database
- The application gets all the entries from the table and places them also in table form on the graphic user interface.

3. **Projecting**

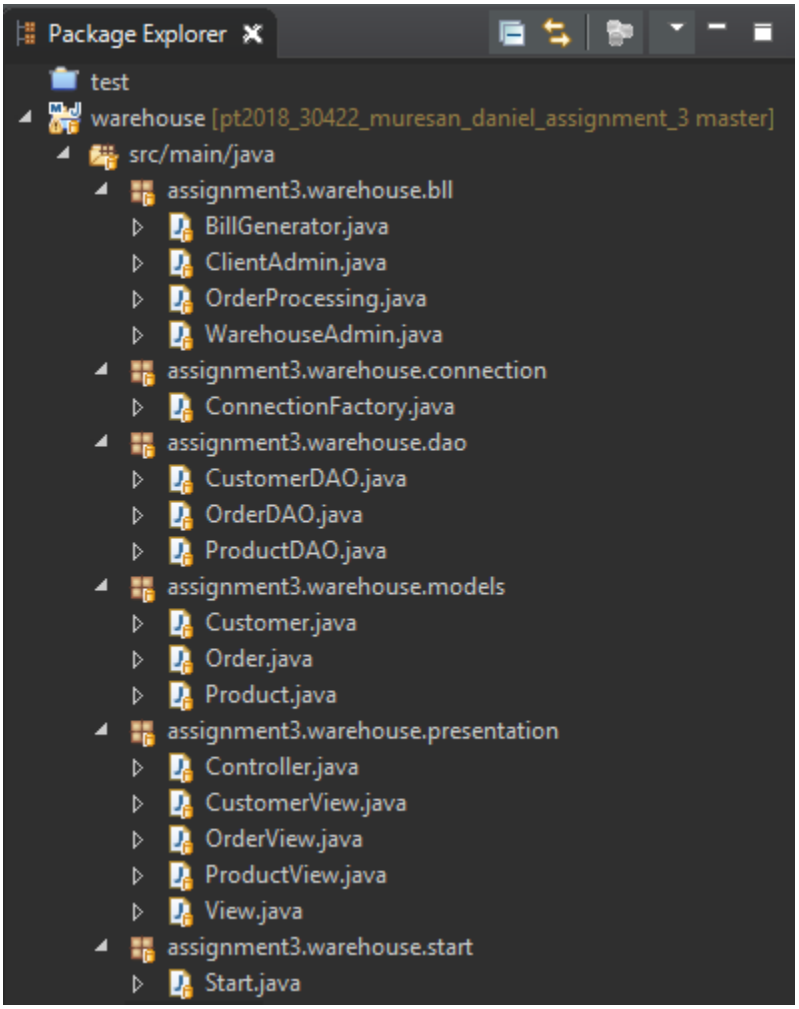
The classes I used for developing this application are the following ones: Product, Customer and Order as models; ConnectionFactory; ProductDAO, CustomerDAO and OrderDAO as data access classes; View, Controlle, ProductView, OrderView, CustomerView as presenation classes; ClientAdmin, WarehouseAdmin, OrderProcessing and BillGenerator as business logic classes and the class Start used for launching the application.

All the classes and the relationships between them are displayed in the next class diagram.



I used a layered pattern to organize the classes in packages. I used the next packages as follows: assignment3.warehouse.bll for the business layer, assignment3.warehouse.dao for the data access classes, assignment3.warehouse.models for model classes, assignment3.warehouse.presentation for GUI classes, assignment3.warehouse.connection and assignment3.warehouse.start.

The structure of packages is presented in the next image.



4. Implementation

Product class

Is the class that represents the products from the real world stored in a warehouse and meant to be sold. The class has few attributes: productID which is the integer which uniquely identifies an object from this class, name as a String representing the name of the product, quantity as an integer representing the number of available products to be sold, and price as an integer representing the price per item. The class has a constructor will all attributes given as parameters and also getters and setters for all attributes.

Customer class

Is the class that represents the customers from the real world which want to buy products. This class has also few attributes: customerID as the integer which uniquely identifies an instance of this class, firstName as a string representing the first name of the customer, lastName also a string representing the last name of the customer, address which is the string with the address of the customer and email which is also a string containg the email of the customer. The class has a constructor with all attributes given as parameters and also getters and setters for all attributes.

Order class

Is the class that represents the orders placed by the customers when they want to buy certain products. As attributes it has the following ones: orderID, the integer which uniquely identifies an order, productID and customerID corresponding to the ones in Customer and Product classes, quantity as the integer representing the quantity to be sold and total as the integer representing the total amount to be paid for that order.The class also has a constructor with all attributes given as parameters and getters and setters for all attributes.

ConnectionFactory

This class is used for creating and closing connections with the database. It has few static attributes specific for creating the connection with the database and an attribute which is an instance of the class Connection. It has a constructor, a method for creating a new connection, a method for returning the connection which is attribute for the class, a method for closing a connection, a method for closing a statement and a method for closing a result set.

ProductDAO

In this class there are methods for actually getting data from the table with products from the database. The first method has the purpose of finding a product by its ID in the database. The next method can insert an object of the class Product in the corresponding table in the database if there is no product with its ID. Also there is a method for updating an entry from the table from database with new information given as an instance of the class Product. The next method is good for deleting from the database an entry corresponding to the product given as a parameter. The last method called viewAll gets all the entries from the table in database and places them in a similar table which is returned.

CustomerDAO

This class is bulit in a similar manner with the one for products. It has the same method for finding, inserting, deleting, updating and getting all the data from the customer table in this case.

OrderDAO

This class is also similar to the ones for products and customers and is used to access the fields of the table with orders in the database. All the methods for finding, inserting, deleting, updating and getting all the data from the orders table are present in this class too.

View class

Is the a component of the graphic user interface and its instances will be the first that the user sees when the application is launched. It has only three buttons which will start new frames corresponding to the use cases whenever the user wants to: manage products, manage clients or place orders.

Controller class

Is the class which manages the behavior of the frame instances of the class View. It has as attributes one such instance of the class View and few subclasses, each one meant to handle a button form the view frame.

ProductView class

Is also a component of the graphic user interface and is the frame that the user will see when he wants to manage the products from the database. It has labels and text fields for introducing data about the products; buttons for selecting operations like insert, delete, update and view all and a table which becomes visible only when the user wants to see all the entries from the table in the database.

CustomerView class

Is built in the same manner with the one for products, having similar layout with labels and text fields for introducing data, buttons for selecting the operation and a table for displaying all the entries form the database when needed

OrderView class

Is the last component of the graphic user interface and also has the same layout with the views for products and customers. All labels, text fields and buttons are present here too. As for the previous two it also has a table where the entries from the database are presented.

ClientAdmin class

Is a sort of controller for the CustomerView class. It basically manages the functionality of the frame by calling the methods from the CustomerDAO class when a specific button is pressed. It also uses methods from ConnectionFactory because every time an operation is performed a new connection is established and then closed back.

WarehouseAdmin class

Is the controller for the ProductView class. In a similar manner with the ClientAdmin class it handles the buttons from the ProductView with some inner classes. The operations are made by calling methods from the ProductDAO class and also there a new connection is established and then closed for every operation apart.

OrderProcessing class

In the controller for the OrderView class. It manages operations on orders also with inner classes, one for each button from the view. The operations are the same but in this case before an insertion is made, the application must check if there is the required quantity for a product, after that place the order and finally update the quantity of the remained products. Every time an order is successfully placed a bill is created for that order with the method from BillGenerator class.

BillGenerator class

Has a method which saves in a PDF file a bill with information about a specific order, information given as parameters for that method.

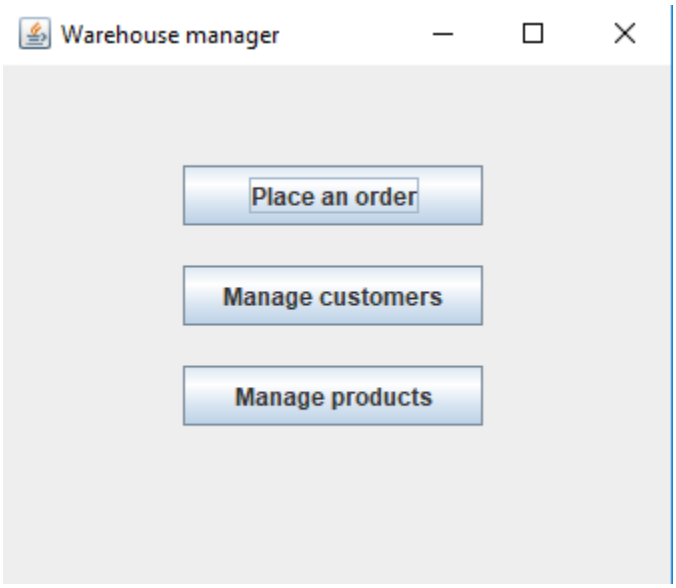
Start class

Contains the method main() which is run in order to launch the application by creating an instance of the class Controller.

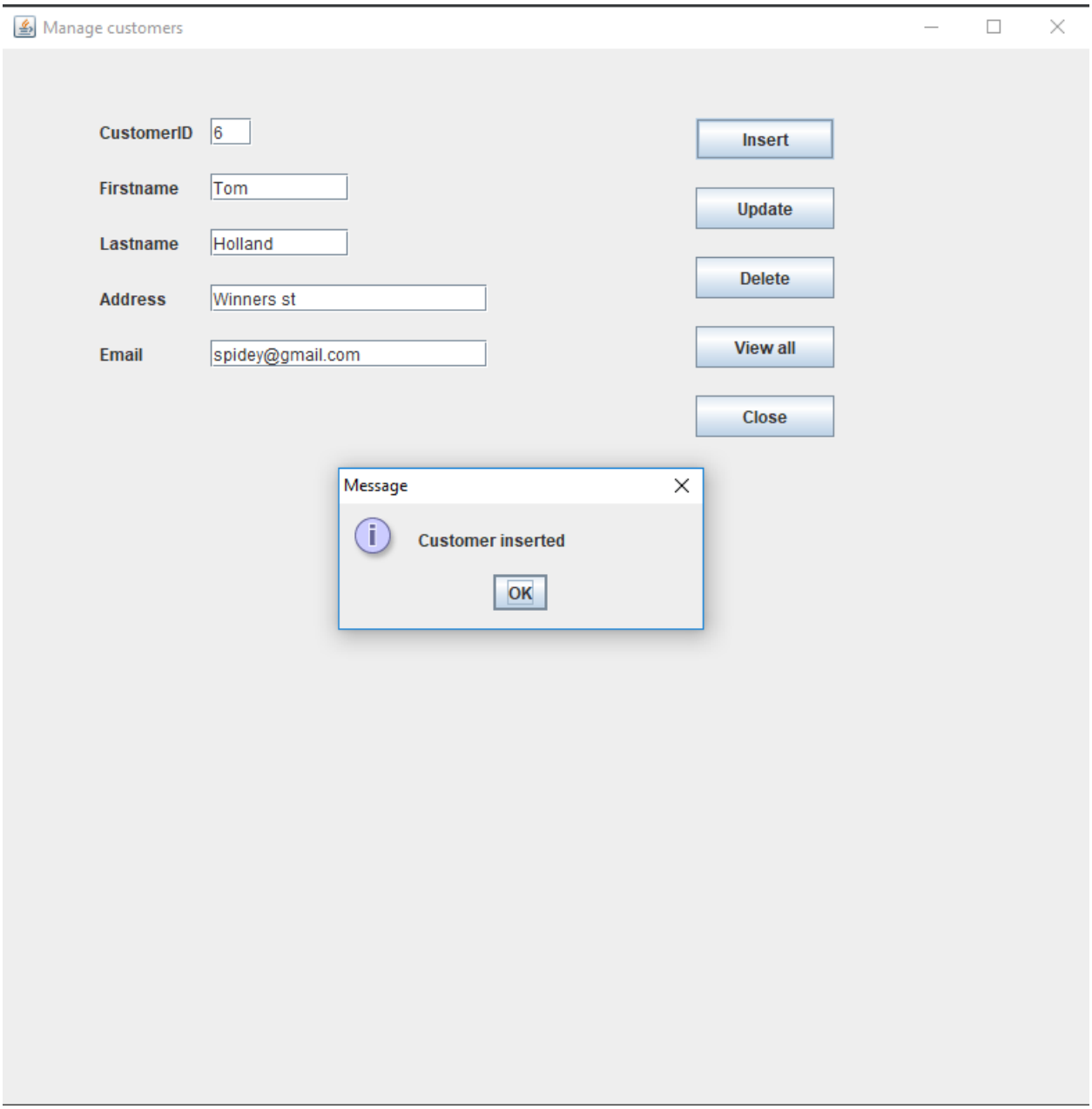
5. Results

In this section I will present a few screenshots with the final application.

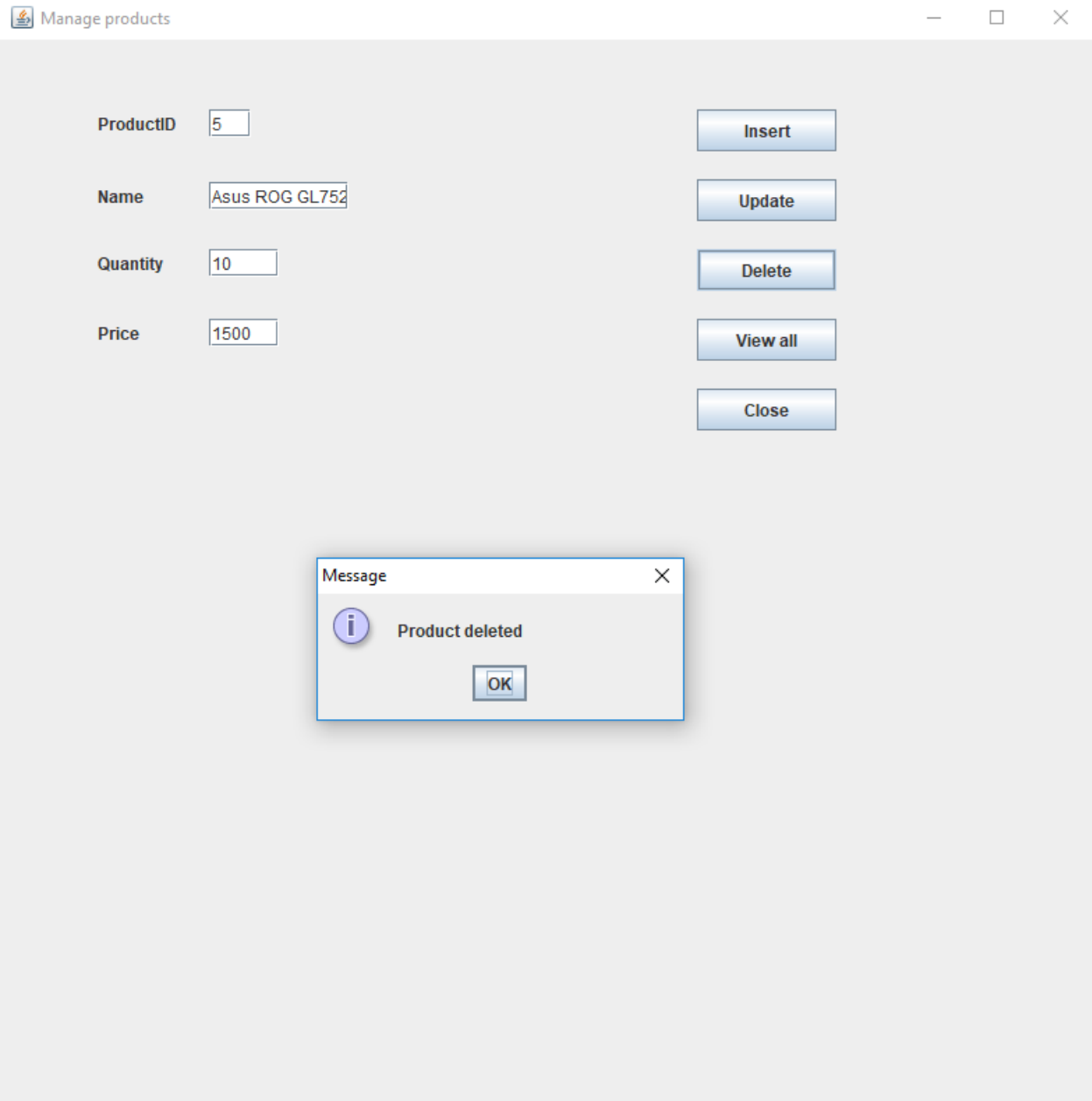
The first image is with the frame the user sees when the application is launched.



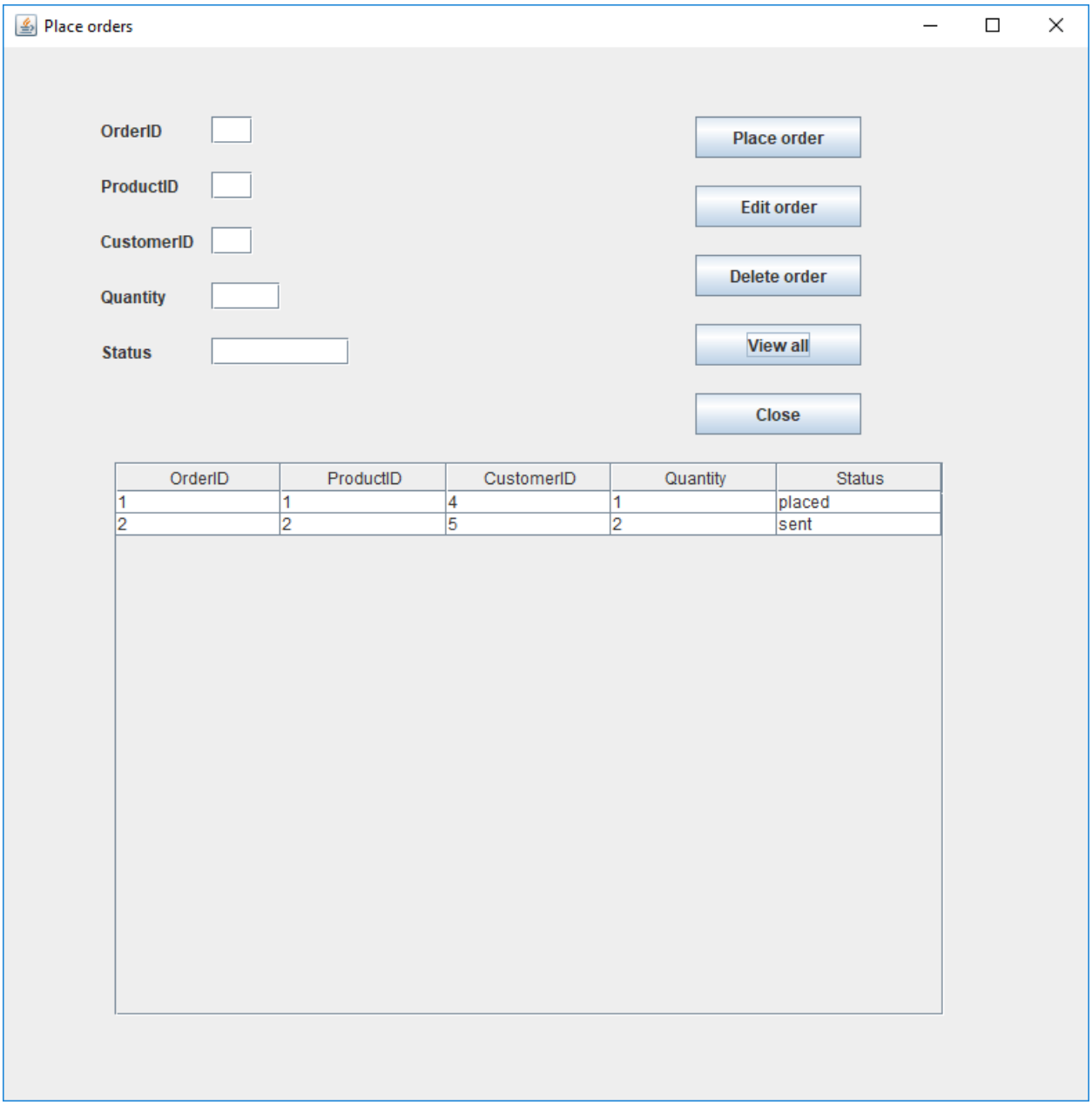
The next image is the result of the operation insert for a customer.



The next image is the result of deleting a product.



And the last image is the result after the operation view all is made on the orders.



6. Conclusions

By developing this application I leant do create connections between a database and a java application and how to handle the exceptions may appear because of that connection. As further development some

interactive tables would be nice so the user should not introduce manually the data for already existing items in tables.

7. Bibliography

http://www.vogella.com/tutorials/JavaPDF/article.html

https://www.youtube.com/watch?v=BCqW5XwtJxY

http://www.homeandlearn.co.uk/java/connect_to_a_database_using_java_code.html

https://www.youtube.com/watch?v=6XoVf4x-tag