Order  
Manager

Programming Techniques – Homework 3

2ND Year, 2ND Semester, Group 30422

Student Name: Pop Elisa Teea

Technical University of Cluj-Napoca

Computer Science and Automation Department

Problem Definition

**Task: The application is meant to simulate the processing of orders for a warehouse using relational databases to store the products, the customers and the orders. The user will be able to choose a customer, a product and create a new order.**

The stated problem can be reached by solving the following sub-problems:

* Implementing the generic CRUD operations that will be extended. ( DAO )
* Implementing the functionality of the given data. ( DLL )
* Preparing the data so it can be shown and interacted with in the UI ( Controller + View)

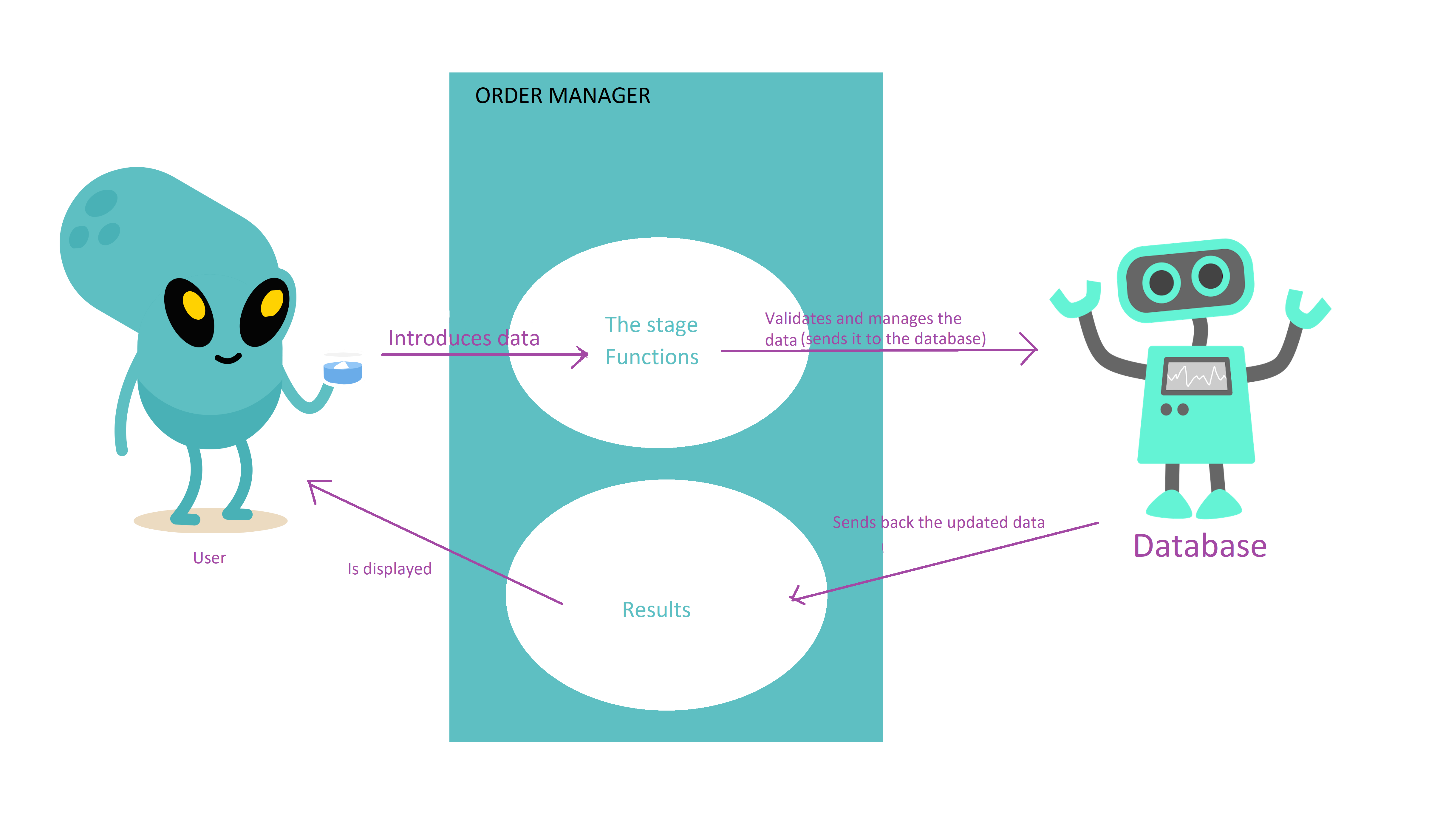
Project specifications:

* Programming language used: Java
* Project SDK: 1.8 ( java version “1.8.0\_191”)
* Project language level: 8 – Lambdas, type annotations, etc.
* GUI: JavaFX
* Program used: IntelliJ Idea
* Git : Bitbucket
* Database: MySQL

Problem Analysis

Use-case diagrams

This application has the intention to allow the user to add, edit, delete and see the data available.



Actors: the User   
Use case title: Start the simulation  
Preconditions: The user has to introduce valid data for all the given fields specific to the function that is about to be used. This varies depending on the current scene.

Success scenario:  
- The user introduces the correct input for each field  
- The user chooses the right operation for the fields that were introduced  
- The change can be noticed in the displayed table.  
  
Alternative scenario:  
- The user introduces an empty, invalid id or field  
- An error box notifying the user about the current problem is shows.  
- After the user reads and closes the box, the activities can be continued normally.

Design and Programming

The project is structured using packages, according to the Layered Architectire: bll is the business logic layer, dao is the data access layer, database contains the connection, gui is the presentation, model contains the architecture of the data and utility provides useful methods that don not fit in other packages.

The database

It plays a very important role in the design. Its implementation is fairly simple, containing only 4 tables:

* Customers: id is the primary key. It mainly contains personal data that is meaningful towards creating an order: name, surname, phone number, email.
* Products: id is the primary key. It mainly contains details about the product: compid ( foreign key for company.id), name, stock and value (price).
* Orders: id is the primary key. It mainly contains foreign keys – customerid for customers.id and productid for products.id. Besides that, productamount for the number of products the client is about to order.
* Company: id is the primary key. Besides id, it contains a column with the company’s name.

The relationships between the tables are the following:

One to many: between customers and orders (a customer can place many orders, an order has only a customer); products and orders (a product can be placed in many orders, an order has only a product); company and products (a company can have many products, a product can only have a company).

The packages

model

* It contains classes that reflect the database’s tables. Those classes are: Customers, Products, Company and Orders. All these classes contain the exact same fields as the database does, and those fields, of course, have the same name. This is necessary for using reflection to extract the data fields from the database.

gui

* As the name suggests, this package covers all the UI-related elements.
* Due to the fact that the MVC architecture is difficult and counter-intuitive to implement with JavaFX because usually we’re working with the Scene Builder so the UI contains a Controller and a .fxml file which is the view. This is the reason why why the application only has a Controller and a View.
* For this implementation, I used multiple views due to the fact that the user can and should be able to swap between the scenes in order to see the data from all the tables. Each of these views has a Controller, which manages each specific operation there is for every different view.

dao

* DAO stands for Data Access Object, and it contains all the methods that directly communicate with the database.

bll

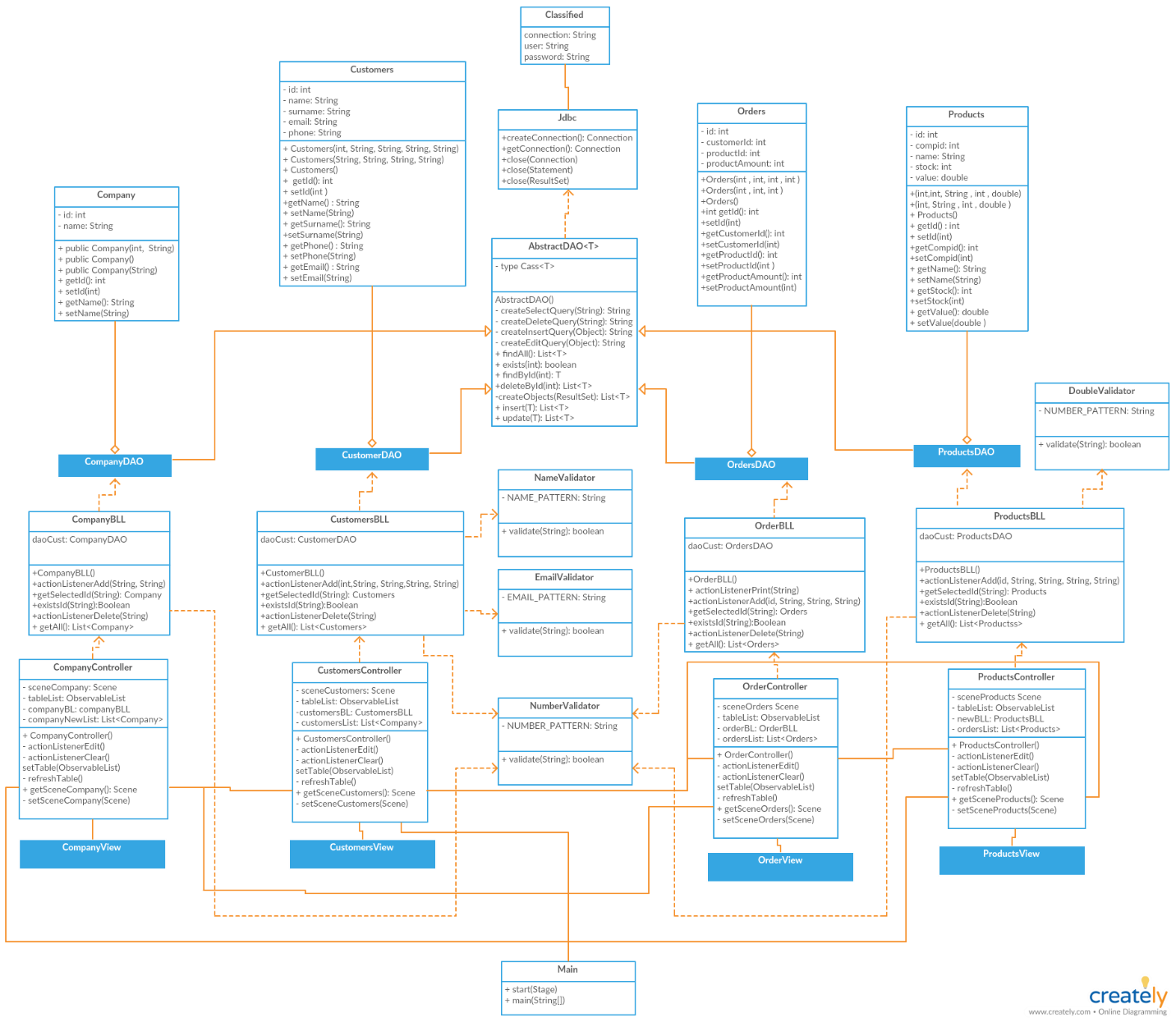
* BLL stands for business logic layer and it contains methods that help coordinate data between the UI and DAO.

database

* This package contains the connection to the used database.

main

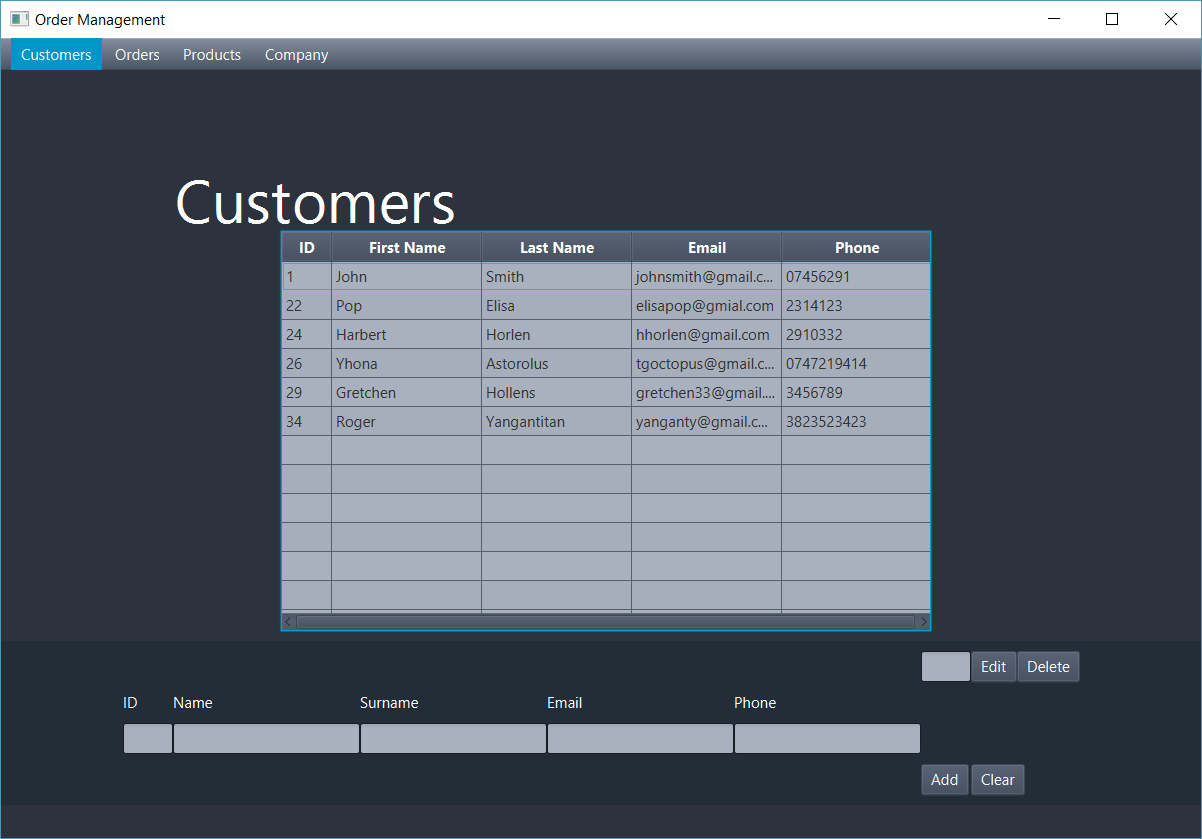
* This package holds a sole class: Main class. This is used for setting the stage and starting the application.

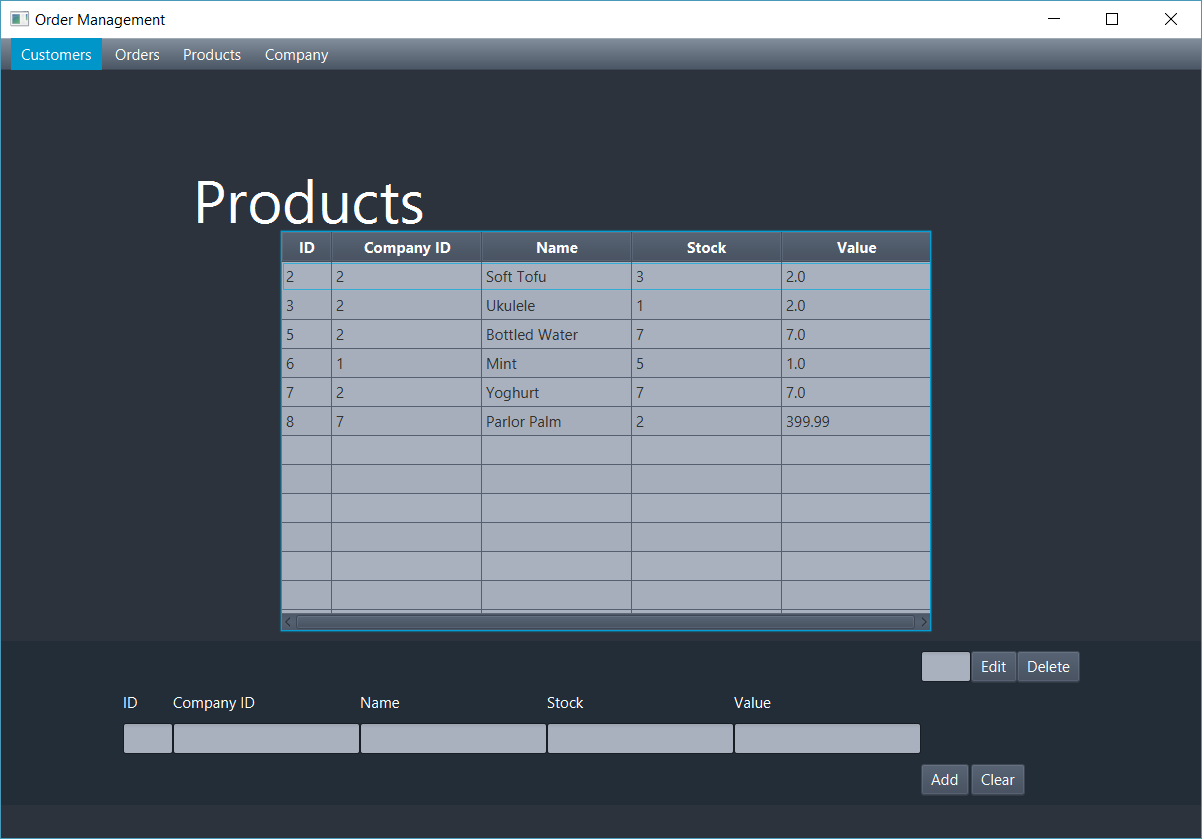


Implementation

The user interface

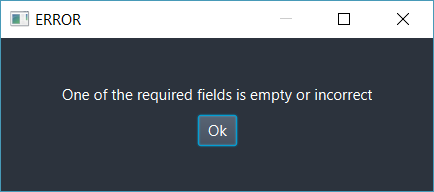
It consists in 4 more packages, each containing a View and a Controller. The Scenes are available to be accessed from one another, because of a ManuBar. Each scene supports the CRUD operations ( Create: The “Add” button, Read: (all) the table that is always shown and updated in real time; (one element) the edit button loads the fields with the element of the given ID, Update: you have to choose and ID and click on edit. After editing the fields, click on “Add” to update the table, Delete: an element, by choosing its id).



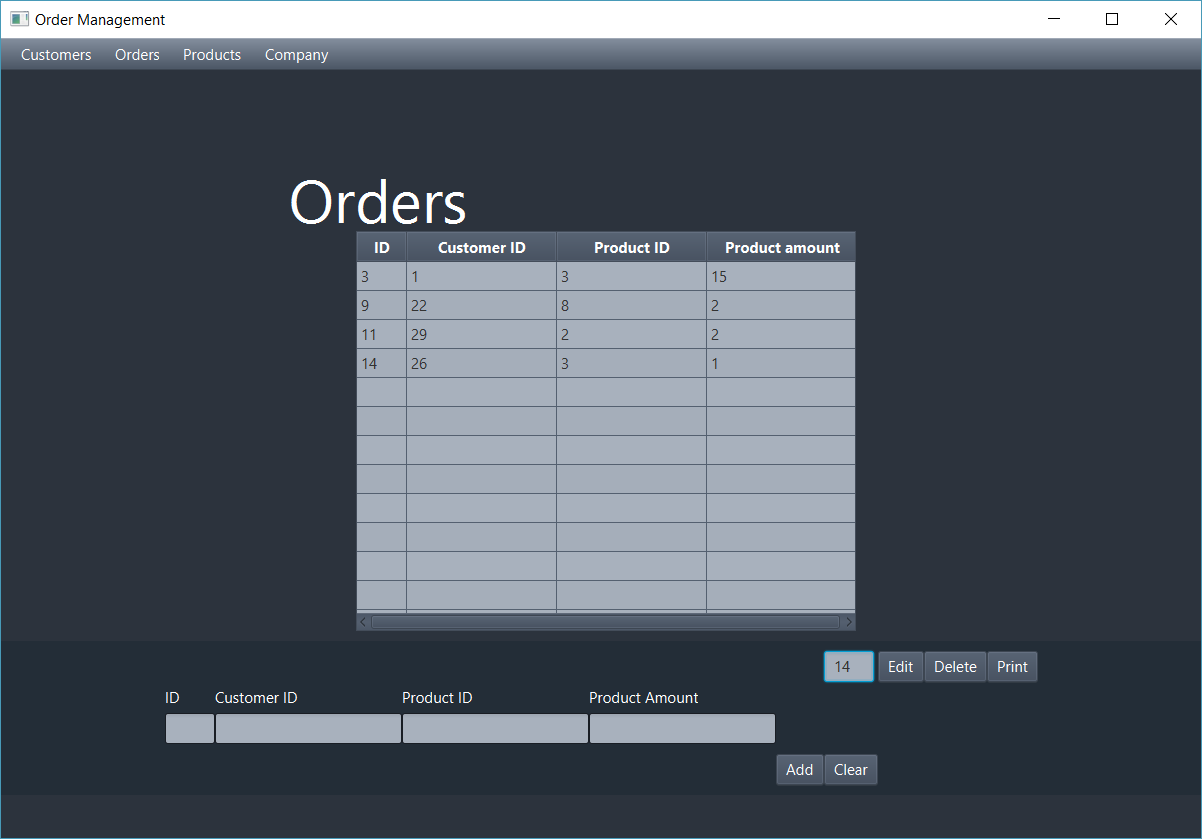


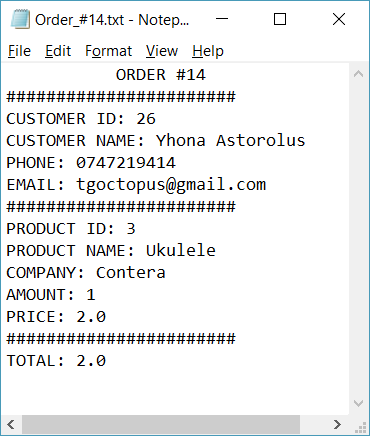
It’s a relatively simple ui, having used a ManuBar with 4 menu items, two GridPanes in which on GridPane has the table with the title label and the other one has 6 TextFields with 5 Labels and 4 Buttons. The latter GridPane may have a different number of those specified elements for other scenes.  
 You can actually change between the scenes in order to see all the data available.

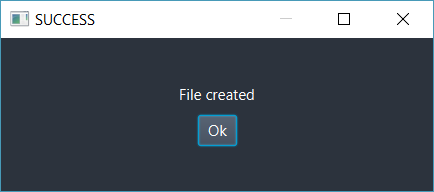
The ID label is always uneditable. The only way it can be edited is by adding a valid ID in the TextField from the upper right corner and clicking the Edit button. By doing that, all the fields are filled with all the current fields from the specified ID. You can change any field and click on the Add button: This will update the item in the database. However, each time you want to edit an item, you have to click on the Clear button afterwards because as long as the ID field is not null, the item at the current ID will keep getting edited.

 As specified before, the fields change from one table to another.

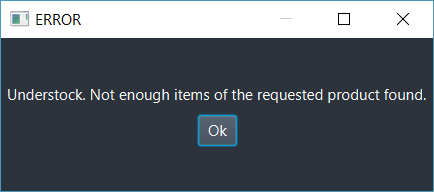
Therefore, depending on the field we have different data input standards. If the field is empty or it does not meet the data inputstandards of that field, this error message will be displayed:



For the Orders view, however, we have an additional function: printing the order receipt. Once the print button is clicked, a success message will appear.



Also in this scene, when we want to add in an order more items than there actually are (in the product list) , an error message will be displayed.



The Controller

Simply sets each action for every UI element present: updates the table, adds action listeners and has getters and setters for the View it represents.

The AbstractDAO

This is a very important method regarding the back-end of this application. DAOs are the only classes able to communicate with the database, therefore, their whole functionality revolves around that. The following functions are implemented by this class:

* createSelectQuery(String field), createDeleteQuery(String field), createInsertQuery(Object t) and createEditQuery(Object t) are all queries for the CRUD operations
* findAll() uses the createSelectQuery in order to get all the fields from the database and the it calls createObjects in order to return the wanted List of objects.
* exists(int id) checks if the given ID can be found in the table. It returns a boolean: true if it can be found, false otherwise.
* findById(int id), similarly to exists, it checks if the given ID can be found in the table. The onfy difference from the previous function is that It returns the found object.
* deleteById(int id) this method removes from the table the row corresponding to the given ID. It also returns the List of updated elements.
* createObjects(ResultSet resultSet) creates the Object List from the results we get immediately after executing the query.
* insert(T t) this method inserts a new entry in the table based on the given argument. This is possible by updating the values in the prepared statement with the attribute values of the given object.
* update(T t) ) this method updates an already existing entry in the table based on the given argument. The approach is similar to the previous method.

The DAO Classes

They are all implementing the abstract class. No class specific function was required for any of them.

The jdbc

This class has several methods for getting and creating the connection to the database. Also, it has methods for closing several database related elements.

* createConnection() creates the connection to the database and returns it if everything goes right. In case of failure, null is returned.
* getConnection() returns the already created connection. If that doesn’t exist, null is returned
* close(Connection connection) is a method that closes the connection given as parameter
* close(Statement statement) is a method that closes the statement given as parameter
* close(ResultSet resultSet) is a method that closes the resultSet given as parameter

The Customers

* This class is used to create and store the data belonging to the cusromers table in the database. The attributes of this class are the following: id, name, surname, email, phone. Besides getters and setter, this function only has 3 constructors: One of them is empty so reflection can be used, one has all the fields and the last one has all the fields excluding ID mostly because that one is going to be automatically generated in most of the cases.

The CustomersBLL

* This class implements several validators in order to verify the data before attempting to add it to the database. This class is able to add, delete and return all the elements from the database to the controller.

The Products

* This class is used to create and store the data belonging to the products table in the database. The attributes of this class are the following: id, compid (representing company id, where the company name can be looked up in the Company table), name, stock, value. Besides getters and setter, this function only has 3 constructors: One of them is empty so reflection can be used, one has all the fields and the last one has all the fields excluding ID mostly because that one is going to be automatically generated in most of the cases.

The ProductsBLL

* This class is similar to every other BLL: it implements several validators in order to verify the data before attempting to add it to the database. It is also able to add, delete and return all the elements from the database to the controller.

The Company

* This class is used to create and store the data belonging to the company table in the database. The attributes of this class are the following: id and name. Just like the classes mentioned above, this function also has 3 constructors.

The CompanyBLL

* This class is the same as the BLLs mentioned before.

The Orders

* This class is used to create and store the data belonging to the orders table in the database. This class has only ID attributes due to the fact that it has one to many relationships with two tables. The IDs are: id, customerId, productid and another important attribute is productAmount.

The OrdersBLL

* This class has the most complex BLL out of all the BLLs presented until now. This is mostly because besides printing a receipt for the order, checking for understock is also a required step. The method only specific to this BLL is the print BLL, which simply computes the text and puts it in a new file. The equation for understock is also calculated:
  + Add: currentStock -= ProductAmount);
  + Edit: currentStock -= (ProductAmount - oldStock);

Results

Since there isn’t much to compute and the application relies heavily on the user interface, it can easily be tested and debugged directly by adding data to the interface.

Conclusions

After finishing this project, I can say I’ve learned a very important aspect about Java and OOP overall, and that is the importance or reflection and how easy it is to extract data from a database. Another important aspect was learning how to set up a layered architecture for such a situation. I’m sure it can be further improved because DAO can be sloppy if it’s badly implemented ()

Further improvements for this program can be:   
- Logs that will help more with debugging   
- Selecting queues to calculate the data for average value at will. This application only computer the values for the first three threads.  
- Computed data showing in real time on the UI and a “Pause” and “Resume” button to check the data needed in the exact second we pause.

References

<https://creately.com>

<https://stackoverflow.com/>

<https://www.baeldung.com/java-dao-pattern>

<https://www.jetbrains.com/help/idea/connecting-to-a-database.html>

<https://www.vogella.com/tutorials/MySQLJava/article.html>

Appendix

1. Cover …………………………………………………………………….. Page 1
2. Problem Definition ……………………………………………………... Page 2
3. Problem Analysis... ……………………………………………………... Page 2
4. Design and Programming ... ……………………………………………. Page 3

* The database …….….………………………………………….... Page 3
* The packages ..……………………………………………….…... Page 3
  + model ….…………………………………………….…... Page 4
  + gui ….……………………………………….……….…... Page 4
  + dao …….……………………………………….………... Page 4
  + bll ….………..…………………………………….….…... Page 5
  + database ….…..………………………………….…..…... Page 5
  + main ….…….…………………………………….…..…... Page 5
  + UML ….………..………………………………….…..…... Page 5

1. Implementation…………………………………………………………… Page 6
   * The user interface ……………………………………………...… Page 6
   * The Controller ……………………………………………………. Page 8
   * The AbstractDAO …………………………………………..…..... Page 8
   * The DAO classes …………………………………………………. Page 9
   * The jdbc ……………………………………………………..……. Page 9
   * The Customers …………………………………….……………... Page 9
   * The CustomersBLL ……………………………...………………... Page 9
   * The Products ……………………………………………………... Page 9
   * The ProductsBLL …………………………………………..……... Page 10
   * The Company ……………………………………………………... Page 10
   * The CompanyBLL ………………………………………………..... Page 10
   * The Orders ………………………………………………….....…... Page 10
   * The OrdersBLL …………………………………………..………... Page 10
2. Results ………………………………………………………………....….. Page 10
3. Conclusions ………………………………………………..…….….…….. Page 11
4. References ……………………………………………………….………... Page 11
5. Appendix ………………………………………………………….…...….. Page 12