Homework 3
Order Management

Student: Iepure Ana

Group: 30424

Contents

1.The objective of the homework ……..…… 3
2.Problem analysis, modelling, scenarios, use cases ……...…… 5
3. Design (design decisions, UML diagrams, data structures, class, design, interfaces, relations, packages, algorithms, user interface).....7
4.Implementation…….…..……8
5.Results……...……..6.
6.Conclusions……..……..11
7. Future Implementation…….…….……12
7. Bibliography…….……..…13

1.Objective

The objective of this laboratory homework is the design and implementation of an order management system for a warehouse. The application uses a relational data base, meant for storing information about the customers, orders, products and details about the order, such as the delivery address and the delivery date. The application can be very useful for crowded restaurants or online websites for shopping because keeping track of all the orders becomes easier. It is a flexible program, allowing the user to view, introduce , update and delete anything, extremely easy. The data for the relational data base is updated accordingly to the input from the user.

So, the purpose of this application is minimizing the complexity of such a system, making everything easier.

2.Problem analysis, modeling, scenarios and use cases

Fundamentally, the Reflection API consists of two components: objects that represent the various parts of a class file, and a means for extracting those objects in a safe and secure way. The latter is very important, as Java provides many security safeguards, and it would not make sense to provide a set of classes that invalidated those safeguards.

The first component of the Reflection API is the mechanism used to fetch information about a class. This mechanism is built into the class named Class. The special class Class is the universal type for the meta information that describes objects within the Java system. Class loaders in the Java system return objects of type Class. Up until now the three most interesting methods in this class were:

- forName, which would load a class of a given name, using the current class loader
- getName, which would return the name of the class as a String object,which was useful for identifying object references by their class name
- newInstance, which would invoke the null constructor on the class (if it exists) and return you an object instance of that class of object

To these three useful methods the Reflection API adds some additional methods to class Class. These are as follows:

- getConstructor, getConstructors, getDeclaredConstructor
- getMethod, getMethods, getDeclaredMethods
- getField, getFields, getDeclaredFields
- getSuperclass
- getInterfaces
- getDeclaredClasses

In addition to these methods, many new classes were added to represent the objects that these methods would return. The new classes mostly are part of the java.lang.reflect package, but some of the new basic type classes (Void, Byte, and so on) are in the java.lang package. The decision was made to put the new classes where they are by putting classes that represented meta-data in the reflection package and classes that represented types in the language package.

Example of use case scenario:

Use case: insert a customer

Let’s suppose the user wants to introduce a new customer.

Main success scenario:

1.The user introduces the customer’s id, first name, last name, email and card number.

2. The validator corresponding to the customer class checks if the introduced info corresponds to the default format.

3.If the information is validated, the new customers is introduced in the data base.

Alternative Scenarios:

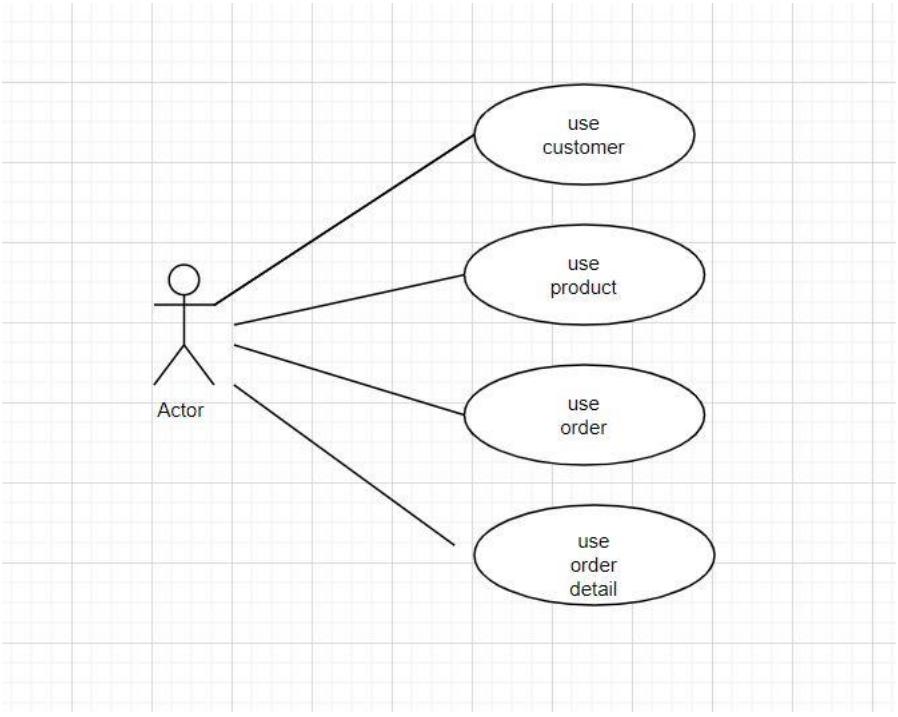
1.Incorrect first name: the names contains special characters.

2.Incorrect last name: the names contains special characters.

3.Incorrect email: the introduced email does not correspond to the default format.

4.Incorrect card id: the introduced id is not 5 digit number.

The use case diagram is the following:



3.Design

A very important first step in implementing the program is choosing how to represent the polynomial: in my case, monomial is the fundamental class which defines the polynomial. These 2 classes are basically the “construction” of the program. Another important part is the “View”, which offers an easy, user-friendly

interface. It contains the 2 text fields where one can introduce the proper format of the two polynomials, 1 text field for the result and buttons for each operation. The controller part connects the internal program to the interface. The project follows the MVC pattern.

Model – Represents the business layer of the application

View – Defines the presentation of the application

Controller – Manages the flow of the application

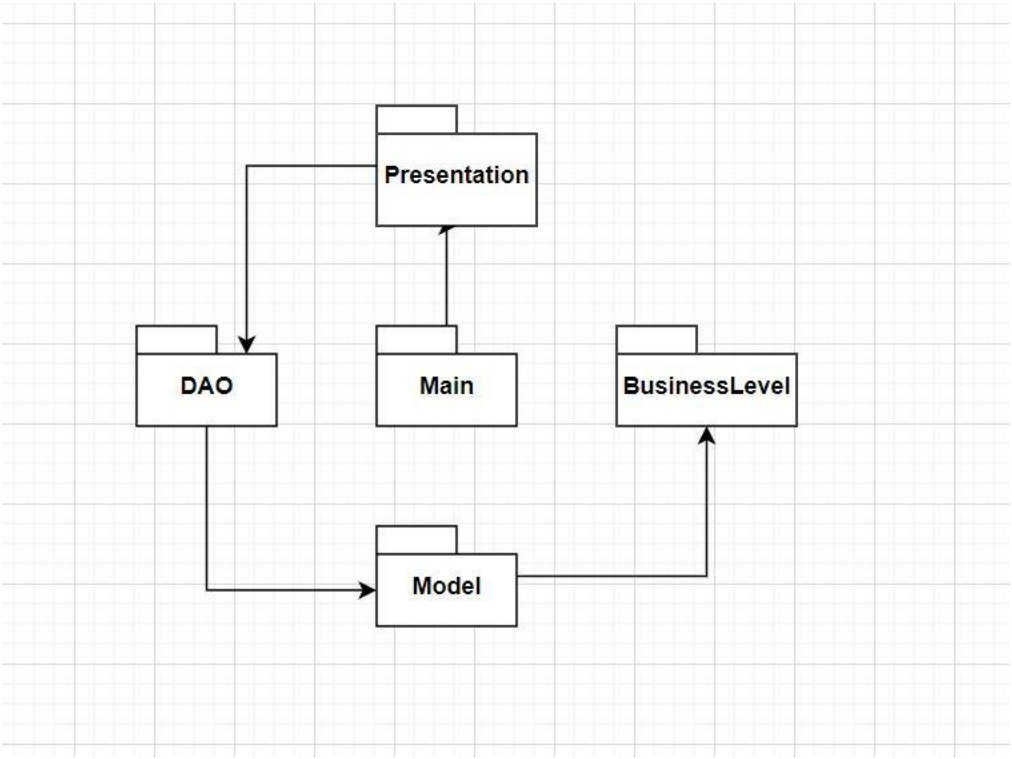
In the MVC design pattern, the model is the data layer which defines the business logic of the system and also represents the state of the application. The model objects retrieve and store the state of the model in a database. Through this layer, we apply rules to data, which eventually represents the concepts our application manages.

The view layer of the MVC design pattern represents the output of the application or the user interface. It displays the data fetched from the model layer by the controller and presents the data to the user whenever asked for. It receives all the information it needs from the controller and it doesn't need to interact with the business layer directly.

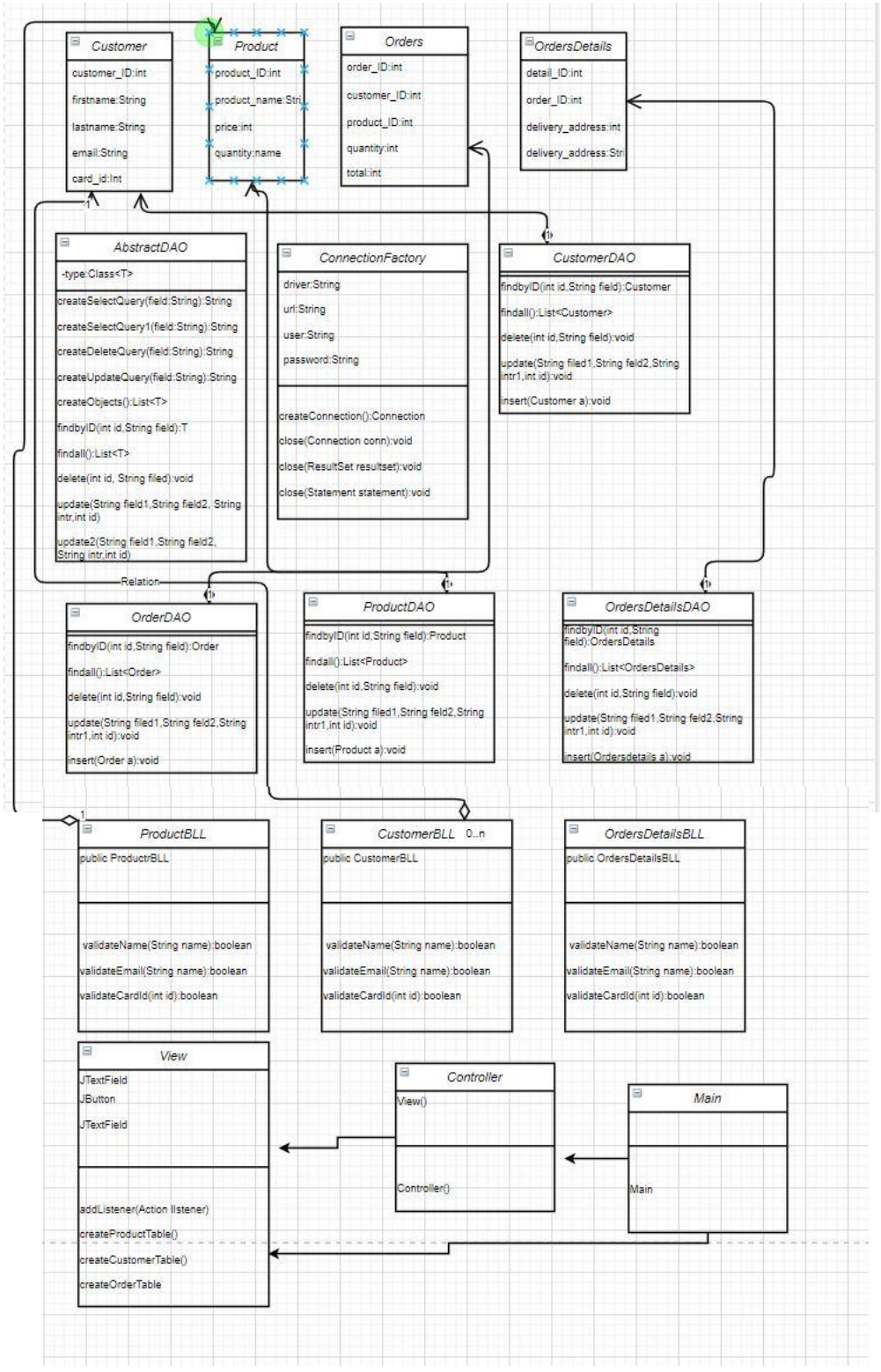
The Controller is like an interface between Model and View. It receives the user requests from the view layer and processes them, including the necessary validations. The requests are then sent to model for data processing. Once they are processed, the data is again sent back to the controller and then displayed on the view.

The MVC Architecture provides an altogether new level of modularity to your code which makes it a lot more readable and maintainable.

Package diagram:



UML diagram:

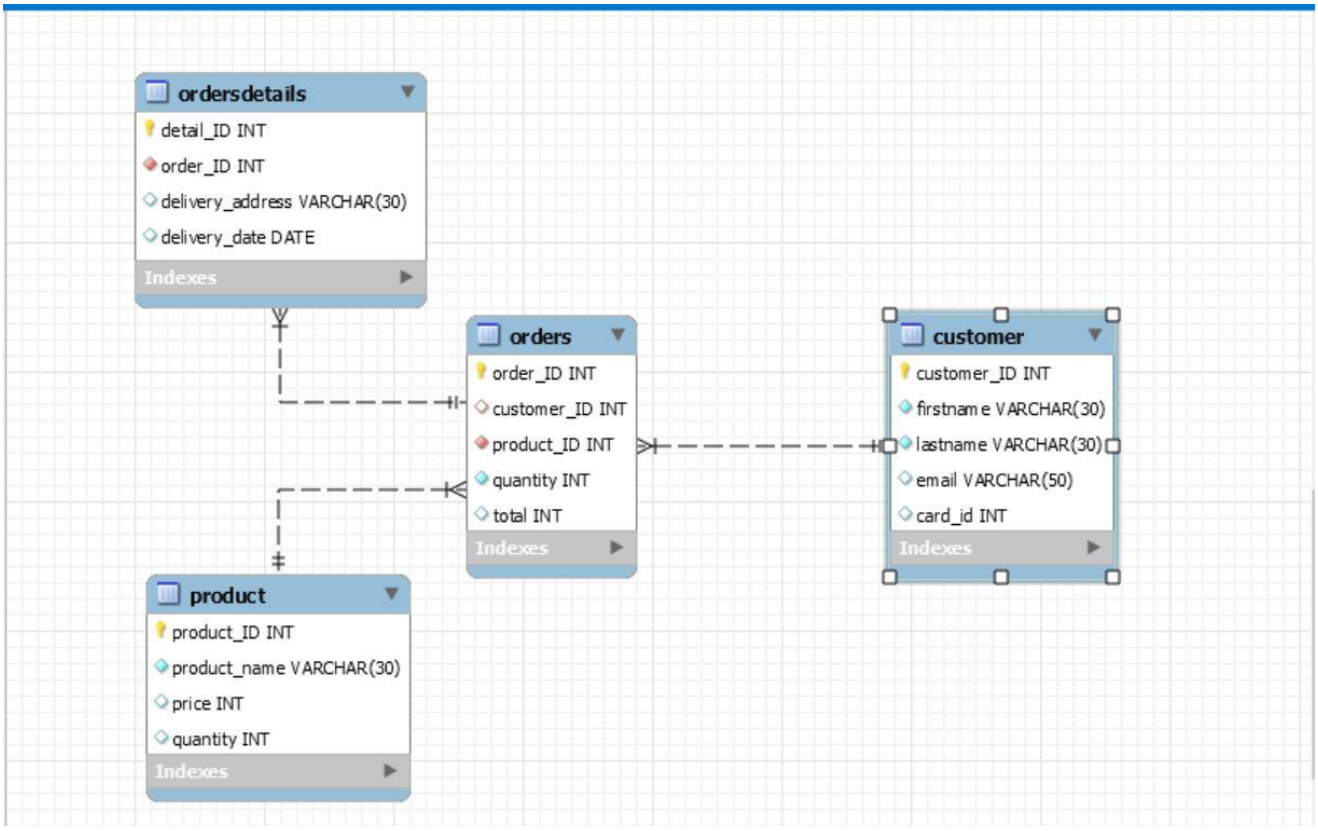


Data base design

I designed a database using mySQL, which consists of four tables, corresponding to the classes in my java application. The customer table has 5 fields: customer id (primary key) of type int, firstname of type varchar, lastname of type varchar, emial of type varchar and a card id of type int. The product table has four fields: product_ID (primary key) of type int, product_name of type varchar, price of type int and quantity of type int. The order table has four fields: order id (primary key) of type int, customer_id (foreign key reffering to the customer table) of type int, product_id (foreign key reffering to the product table) of type int, total of type int.

The orders details table has 4 fields: detail id (primary key of type int), order id (foreign key reffering to the order table) of type int, delivery address of type varchar and delivery date of type date.

The diagram of the data base is the following:



Implementation

Customer Class: this class represents the customer's format. Each object of type customer has as attributes a customer id, firstname, lastname, email, card id.It also contains a contructor, a getter and a setter for each field and a to String method.

Product Class: this class represents the product's format. Each object of type product has as attributes a product id, product name, price, quantity.It also contains a contructor, a getter and a setter for each field and a to String method.

Orders Class: this class represents the order's format. Each object of type order has as attributes a order id, customer id, product id, quantity, total.It also contains a contructor, a getter and a setter for each field and a to String method.

OrdersDetails Class: this class represents the orders details's format. Each object of type order detail has as attributes a detail id, order id, delivery address, delivery date.It also contains a contructor, a getter and a setter for each field and a to String method.

ConnectionFactory Class: this class is responsible for establishing the connection between the data base and the java application. It needs a driver, an url, a user and a password. It also contains a construction, a method getConnection and some methods for closing the connection, the result set and the statement.

AbstractDAO Class: this class implements the generic actions for each table in the data base, such as insert, delete, update.

```
1.private String createSelectQuery(String field)
```

This method creates a select query, which shows the content of a table, according to a specific field.

```
2. private String createSelectQuery1()
```

This method creates another select query, which shows all the content of a specific table.

```
3. private String createDeleteQuery(String field)
```

This method creates a delete query according to a specific field in the table.

```
4. private String createUpdateQuery(String field1,String field2)
```

This method updates a desired field, according to the id of that table.

```
5. private List <T> createObjects(ResultSet resultSet)
```

This method returns a list of objects conatining the content of the table and the names of the columns.

6. `public T findbyID(int id, String field)`

This method can find a specific row in a table according to an id. It uses a select query.

7. `public List<T> findall() throws SQLException`

This method returns a list of generic objects, conating all the data from a table. It uses a select query.

8. `public void delete(int id, String field)`

This method deletes a row in a table, according to a field name and an id. It uses a delete query.

9. `public void update(String field1,String field2,String intr,int id)`

This method updates a coulmn in a rown. It uses an update query.

CustomerDAO Class: this class extends the AbstractDAO class, implementing all of its methods. In addition to this, it contains an insert method, which inserts a new customer in the table.

ProductDAO Class: this class extends the AbstractDAO class, implementing all of its methods. In addition to this, it contains an insert method, which inserts a new product in the table.

OrderDAO Class: this class extends the AbstractDAO class, implementing all of its methods. In addition to this, it contains an insert method, which inserts a new order in the table.

OrderDetailsDAO Class: this class extends the AbstractDAO class, implementing all of its methods. In addition to this, it contains an insert method, which inserts a new order detail in the table.

CustomerBLL Class: this class validates the customer format introduced by the user. It conatins some methods of type boolean which verifies every field.

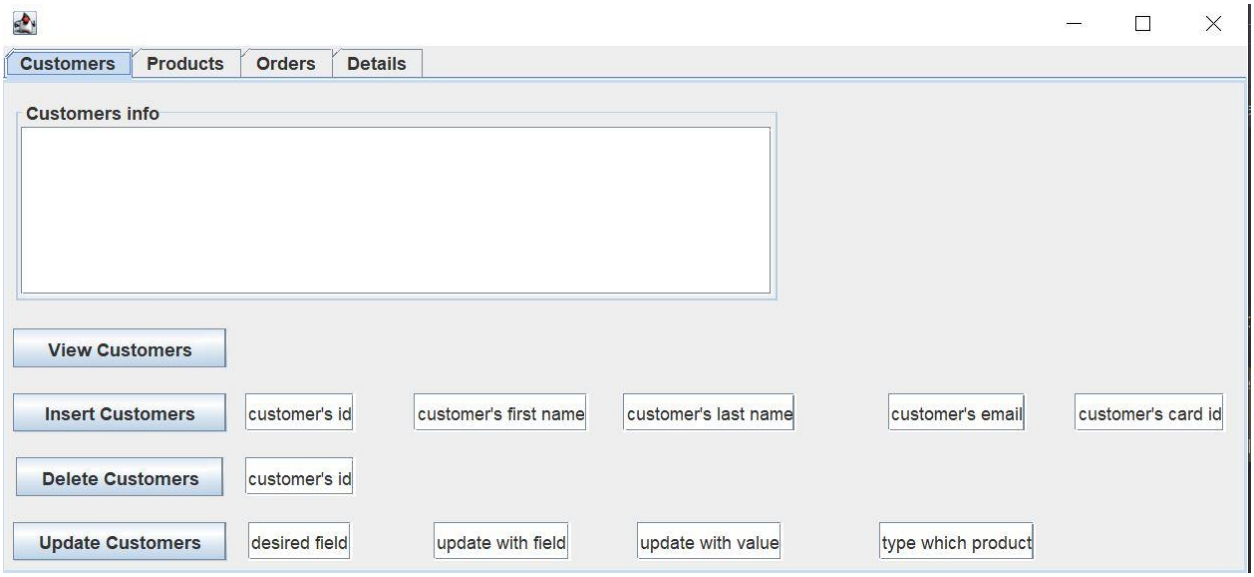
ProductBLL Class: this class validates the product format introduced by the user. It contains some methods of type boolean which verifies every field.

DetailOrderBLL Class: this class validates the order detail format introduced by the user. It contains some methods of type boolean which verifies every field.

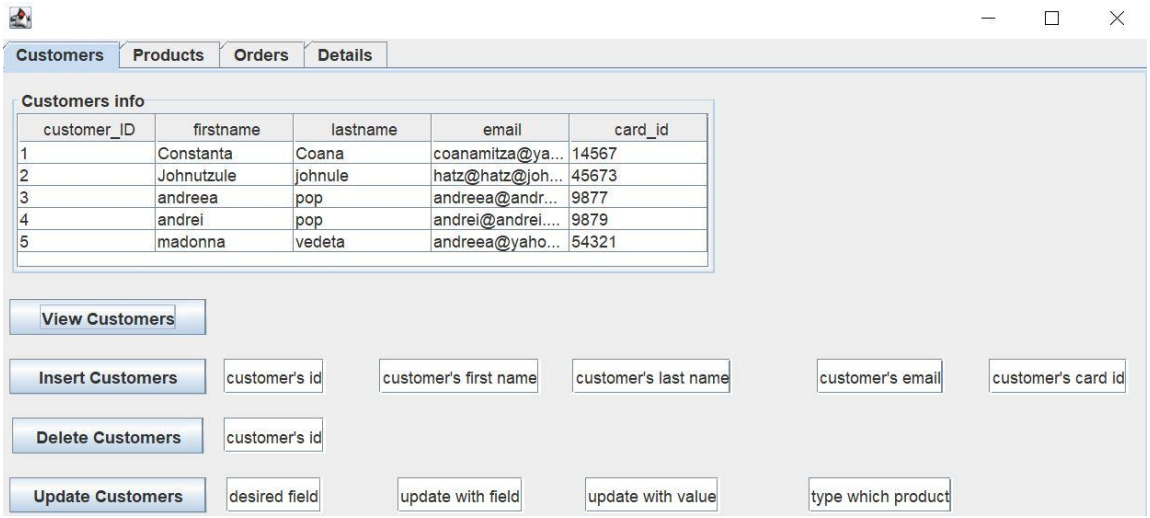
View Class: this class implements the GUI of the program. It contains a frame, panels, buttons, tables and text fields and it uses GridBagLayout. This class is also responsible for creating the tables contents.

Controller Class: this class controls the data flow in the program and it also updates the data base. It adds action listeners for every button. It generates the bill of a specific order, writing the information about that order in a text file.

The GUI:



These are the results after pressing the button "View customers"



Conclusions

The application can be very useful for crowded restaurants or online websites for shopping because keeping track of all the orders becomes easier. It is a flexible program, allowing the user to view, introduce , update and delete anything, extremely easy.

The data for the relational data base is updated accordingly to the input from the user.

So, the purpose of this application is minimizing the complexity of such a system, making everything easier.

In my opinion, this homework was a very good exercise in remembering all the programming concepts that I have learned in these university years. It was pretty challenging because I am new to java so I noticed some things.
First of all, it is very important to organize your project from the beginning and to start with a concept of how you want your project to look like. By doing this, it will help you implement it faster and more practical.
Second of all, I notice that this kind of assignments force you to do a lot of research from which you can learn new concepts. Even if you can find everything on the internet, you still have to know how to manipulate and filter the information so that it works for your code.
The most important thing that I have learnt from this homework is how to work with interfaces and how to link it with a controller.

Bibliolgrafy

1. https://www.infoworld.com/article/2077015/take-an-in-depth-look-at-the-java-reflection-api.html

2. https://stackoverflow.com/questions/4736/learning-regular-expressions

3. https://stackoverflow.com/questions/30656473/how-to-use-gridbaglayout