Assignment nr.3

Lungoci Luca, 30424

1. Assignment objective.

The scope of this is assignment is to create an application that interacts with a database. This application should provide means to insert, edit , delete, view, find data from a data bases. This database stores records describing clients, products and orders. Each record from each one of those tables are described by an id. The table with the orders is dependent of the other two tables because it has to store the id of a costumer and product. These two ids have to actually exist in order to make an order valid. Also, the number of products required in an order cannot be greater than the number of existing products of that type.

This application has to provide the user with tree different windows to interact with, one for order, one for product and one for client. The window for the client and the product are almost identical, but the one destined for the order provides some choice boxes so that the user will select already existing clients and product, there will be no need to guess the name of a client or product.

Data extracted from the data base are displayed in a view table, so that it will be easier to read.

1. Problem analysis, modeling, scenarios, use cases.

* Functional assignments: user can enter values for the fields describing a product and a client, can choose between existing clients and products when making an order, can change the values of an existing client or product record, can delete a client / product record, can search for a client / product with a specific name, can view all the clients and all the products from the data base, will receive a bill after making an order. The user should be able to edit any field he / she wants to.
* Non-functional requirements: the application should be easy to use, provide helpful messages in cases of errors, display the result in a friendly form. It should not crash regardless the inputs received from a user.

Use cases:

Success scenario:

* User opens the order window.
* Selects an existing client and product.
* Enters the amount of products.
* Presses the “Add order” bottom.
* The data base has been updated and a bill (represented by a text file) is available.

Alternative scenario:

* User opens the order window.
* Selects an existing client and product.
* Enters the amount of products.
* Presses the “Add order” bottom.
* There are not enough products for this order.
* The user is informed through the graphical user interface.

Use cases:

* User opens the client window.
* Enters values to add to the database, separated by coma, in this order: id, name, address, phone.
* User presses the “add” button and the text of the label is set to “Ok”.

Alternative scenario:

* User opens the client window.
* Enters values to add to the database, separated by coma, in this order: id, name, address, phone.
* The name has a character that is not a letter, a warning message is set to a label on the user interface.

Alternative scenario:

* User opens the client window.
* Doesn’t enter enough data (there are not enough values for all the fields).
* A warning message is displayed on the graphical user interface.

Some of the messages the user receives when entering data that can’t be processed are:

"Some characters used for names are not allowed"

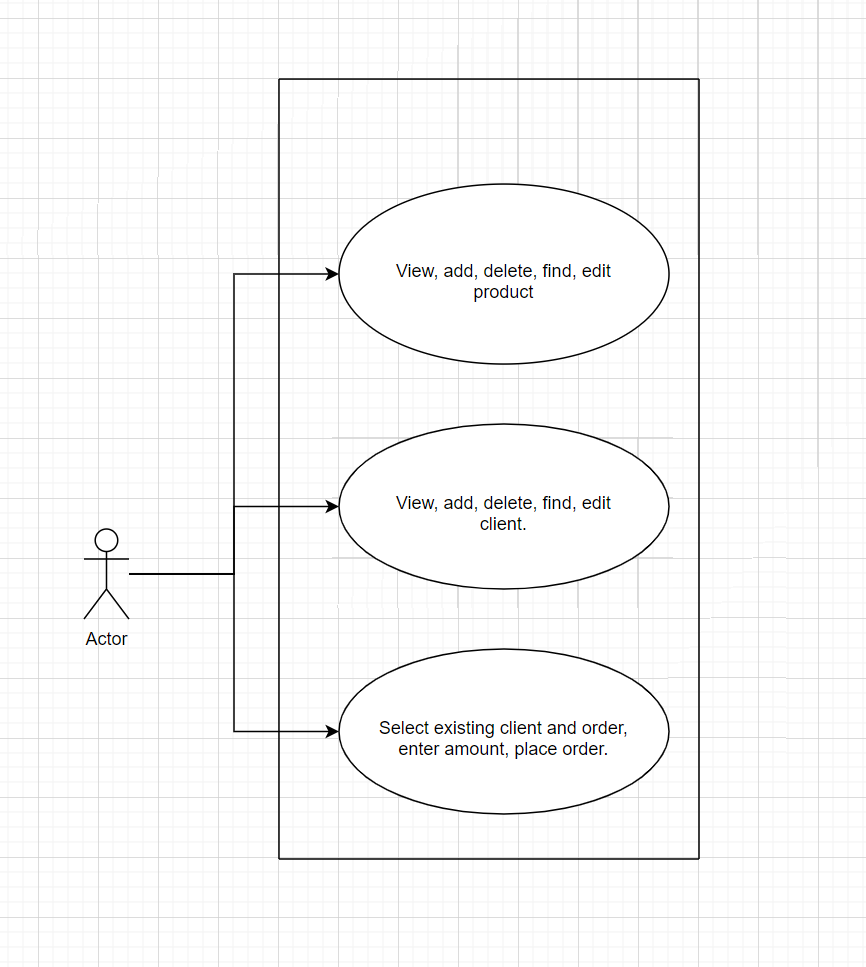
"Maximum number of characters for fields is 45, for phone 10"

"Only numbers allowed for phone number"

"No negative numbers allowed"

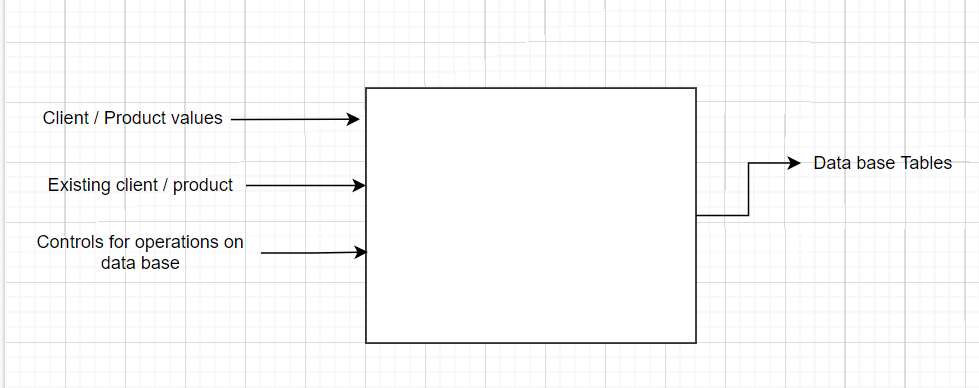
"No such product exists"

"Product with this id already exists"



1. Design.

Application: This application is meant to make interacting with a database very easy. A user that doesn’t know any sql should be able to perform the basic operations on a database.

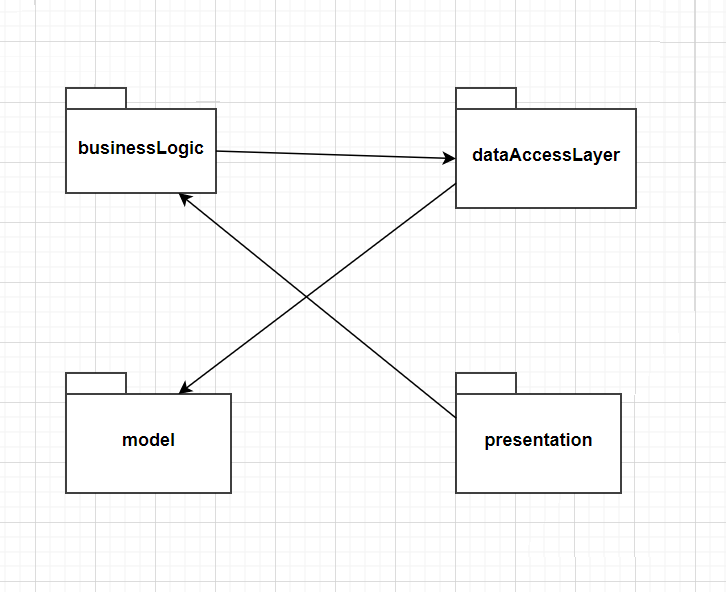


Packages: The classes I considered to be fit for the “businessLogic” package are: ClientBLL (makes the connection between the DAO classes and the classes of the graphical user interface, has methods for adding, deleting, editing, selecting records from the database), ProductBLL, OrderBLL, Validator (performs operations on the data inserted by the user, checks if the data can be used in the application and builds informative messages in case the user inserted data not fit for the application), WriteFile (opens and closes a text file, adds text to a file).

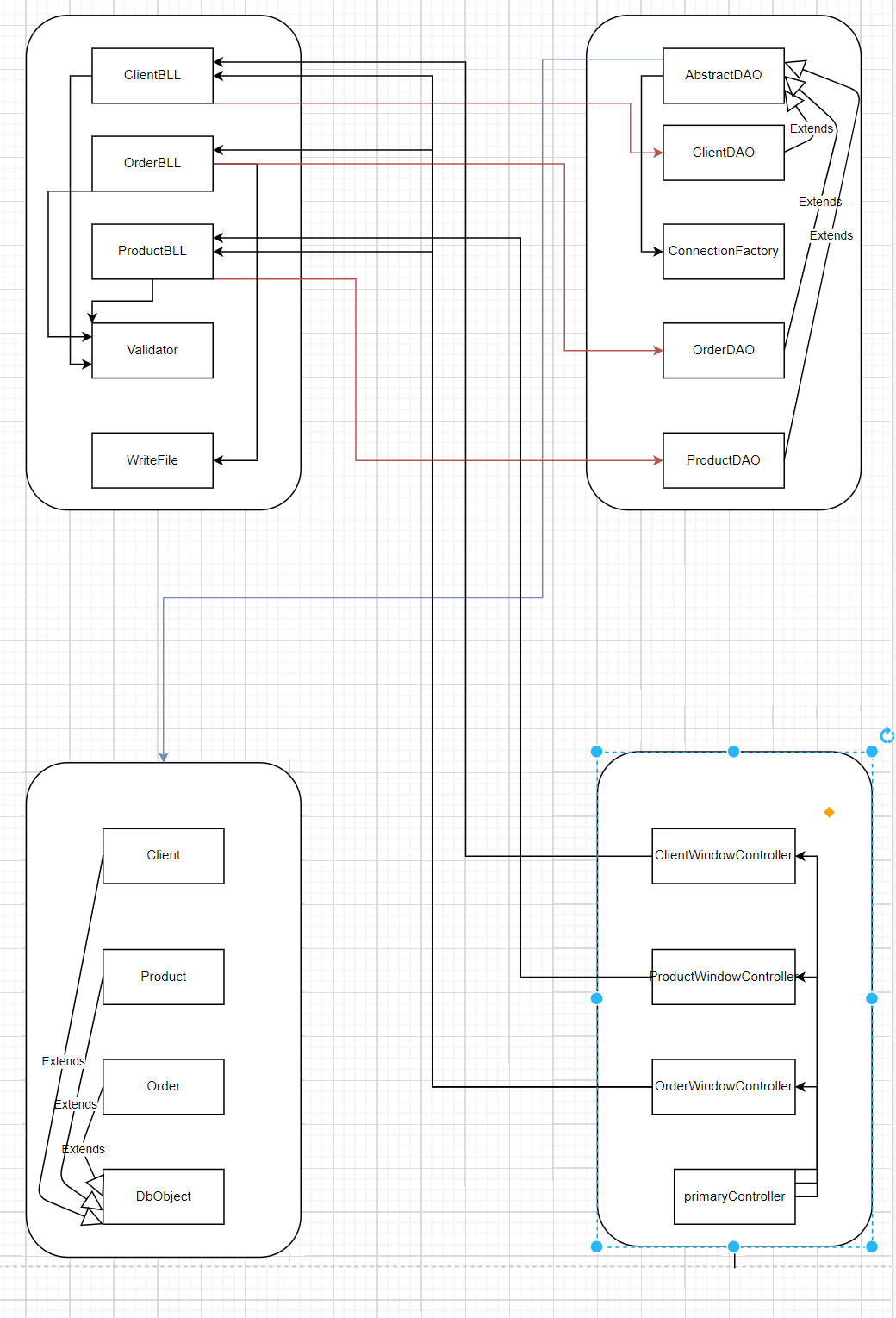
For the “dataAccessLayer” , the classes belonging to this package are: AbstractDAO (builds strings representing sql statements, extracts the result set from the database), ConnectionFactory(makes a connection to the database), ClientDAO, ProductDAO, OrderDAO (these 3 classes extend the AbstractDAO class and provide a type to replace the generic type of the AbstractDAO class).

The “Model” package consists of the classes: DbObject (has only the ID field, common to the following three classes ), Client (describes a database record and has the fields: ID, name, address, phone), Product ( describes a database record with the fields: ID, name, price, number of products), Order(describes a database record with the fields: ID, client ID, product ID, number of products required for this order). Those classes are fit for this package since they provide only setters and getters, they represent means of storage.

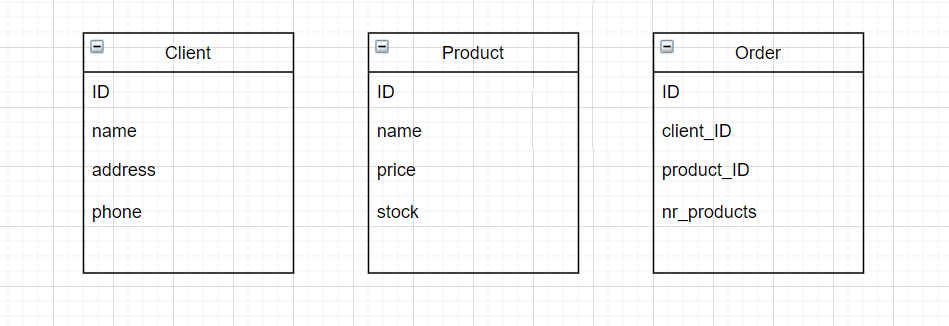
The “Presentation” package includes the classes: ClientWindowController, OrderWindowController, ProductWindowController, primaryController, each corresponding to one of the applications windows. The controller classes for the client and product extend “Application” in order to open a new window with the records from the database.



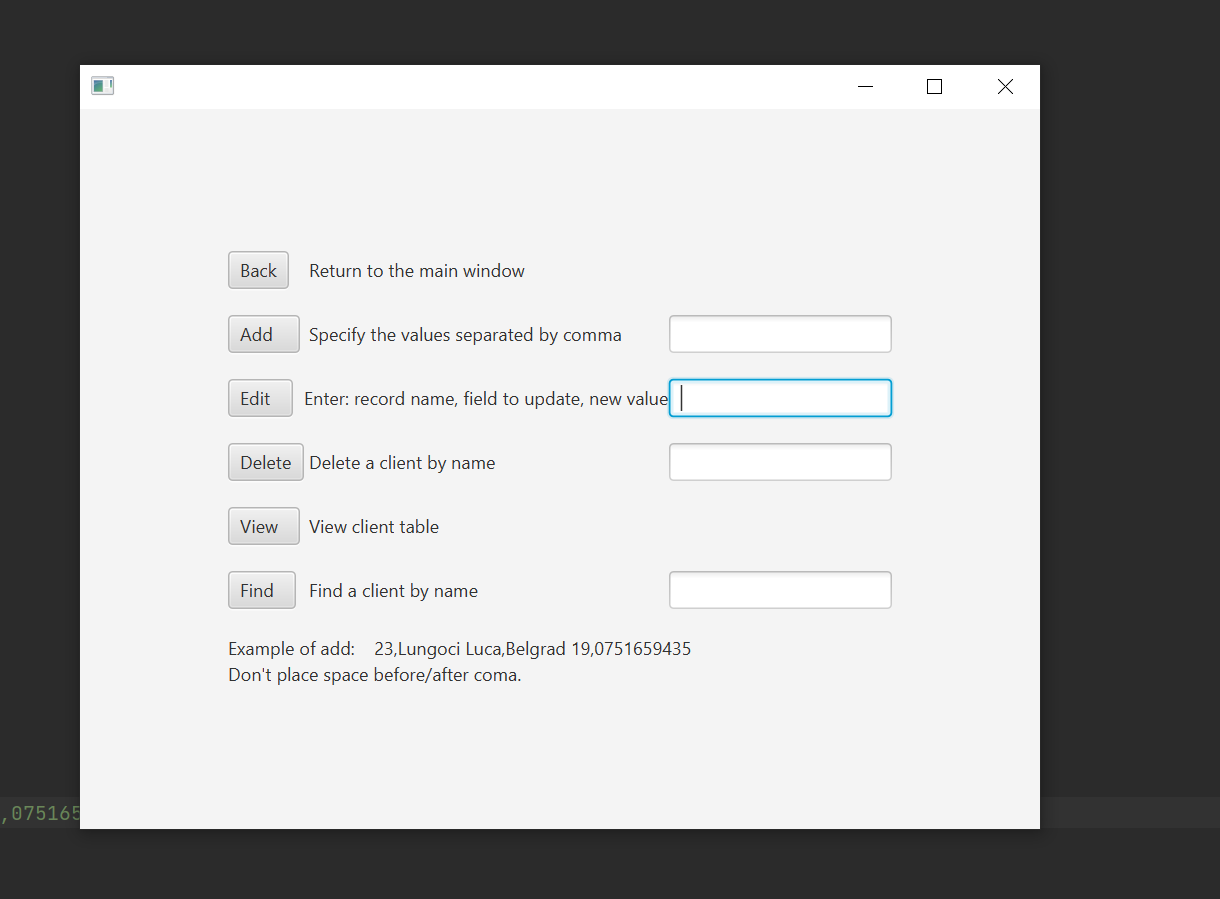
Classes:



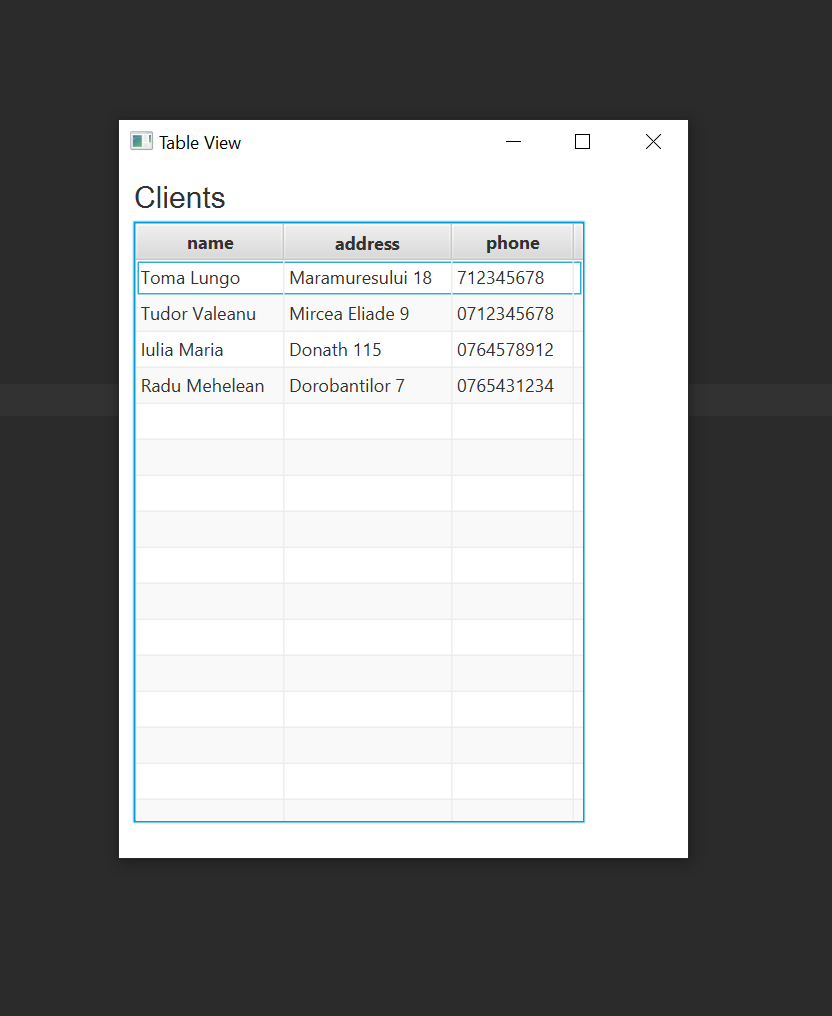
Database:



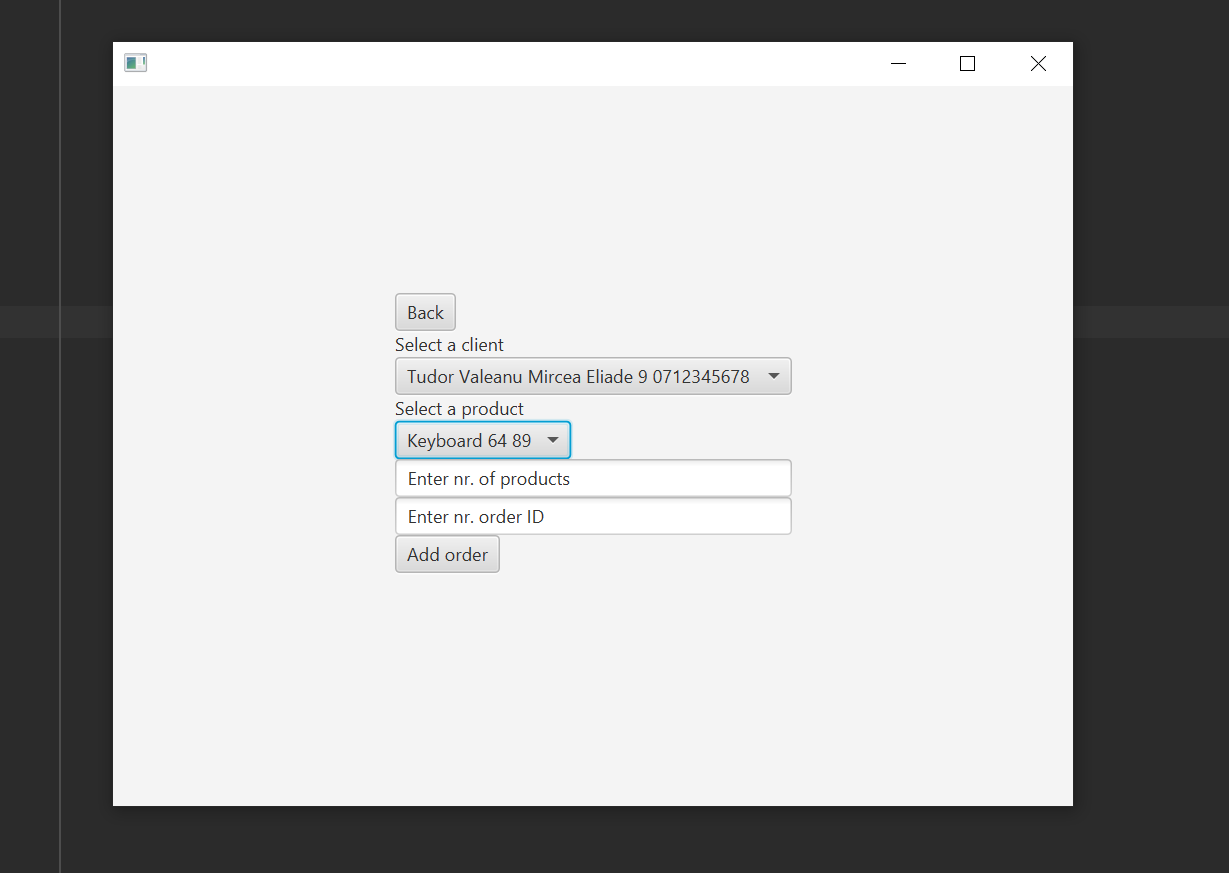
All the fields except ids, price, stock and nr\_products, are of type VARCHAR(). Price, stock and nr\_products need to be numbers because they are used in arithmetic operations when checking if there are enough products for an order and when computing the total price of the order.



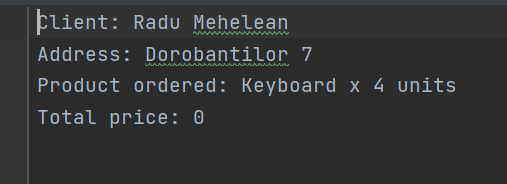
This is the window destined to the clients. The one for products looks the same. It has buttons to return to the main window, “add” is for inserting the content from the text box on its write in the database. There needs to be an id, name, address and phone specified like in the example on the screen. Watch out for spaces before and after comas because it will be taken in consideration as input for the database. Editing needs the text box to be filled in the same way as for inserting. First you need to specify the name of the record, then the field of this record you want to change, then the new value for this field, all of those separated by coma. An example for editing is : “Sunglasses, price, 500”. Here space is allowed. Deleting, viewing and finding are pretty straight forward.



This is the window opened after clicking the “view” button. Looks the same for “find”.



This is the window opened after pressing the “Order” button in the main window. It provides two choice boxes, first for clients and second for products. Two text boxes also need to be filled in order to successfully make an order. In the case of entering an existing id or the requested number of products is too much, a label of warning will be displayed on the screen.

Adding a successful order will generate the following text in the “bill” text file.

After the product name is displayed the quantity the user requested “x 4 units”.

1. Implementation.

The “AbstactDAO” class provides methods to manipulate the database. Because those operation are basically the same for a product, client and order (except for the fields name and types), this class has a generic type. It accepts objects of type “DbObjects” , those being Client, Product, Order. This class provides methods to create objects corresponding to the database records and for a specific record searched by id or name, methods to insert a record in the database, delete record and edit record. One of the instance variable of this class is “type” used to obtain a string with the name of the class that replaces the generic type. This name is used when the sql statements are created. This class provides to the other classes using it a list of objects, when the content of the database is required. Reflection technique is used in this class for providing a string with the fields of a class. This is useful when the user wants to add a new record and needs to know what to add. The “getTableHeader()” method returns a list of objects of type String, each element of the list representing the name of a field. There also exists a variant of this method that also adds the fields of the superclass (the ID field from DbObject). Another use for the names of the fields is when the name of the columns (when displaying the database content) needs to be set. This happens in the classes manipulating the graphical user interface.

The “Validator” class provides means to check the data inserted by the user, but also provide useful messages in case something is wrong with the input provided. Its static methods require as parameters the model objects and uses the fields of type string. The fields of type int are checked in the BLL classes.

The ClientBLL, OrderBLL, ProductBLL classes mainly call the methods of the AbstractDAO class, but also provides some additional functionality when its needed. For example, in the case of clients and products, when one of those is deleted from the database, because of the foreign keys, all the orders corresponding to that client or product must be deleted before. Another additional functionality needed by this application is in case of the products, when an order is performed , the number of products in stock must be updated. Adding clients, products or orders with an id that already exist will cause an error in the methods sending queries to the data base. These errors will be handled in those methods but the user needs to be informed that the update was not performed and why that happened. Because of this, the method calling the “insert” of the AbstractDAO, checks first if a record with that id already exists and if so, builds a string destined for the graphical user interface. The BLL classes are the place where the methods of the “Validator” class are being used.

When adding a new order, an existing client id and product id are required. After calling methods to get an object corresponding to this client and product, in the controller class corresponding to the order window there will be information about the client, product and order. A client object and a product object is returned after executing a “select” sql statement, statement needed to provide the user option for the choice box. This is why in this class I computed the strings for the bill. The bill consists of the client name, client address, product name and the total price the client has to pay for the number of products required. Those string are later on passed to the “WriteFile” and copied in a text file.

“ConnectionFactory” is where the connection to the database is established and closed. It needs a driver, a user, a password. This classed is only used by “AbstractDAO” , where all the database related operations are executed.

The database records information is stored as “model” package objects, in lists, because there are methods for the View table to arrange object fields in columns , after names.

5.Conclusions.

From this assignment I learned how to combine to important concepts in the computer science domain: OOP and databases. This introduction of using java with databases should be essential for my future as a programmer. It also opens up a lot of possibilities for future applications. Working with generic types was another useful experience and developed my skills for a very important OOP concept. This will reduce the amount of code written when classes perform similar things and it will make methods applicable to multiple types. Practicing with the “Layered Architecture” design pattern seemed to be an important skill for the future.

6.Bibliography.

For learning to work with Table view: <https://www.youtube.com/watch?v=mtdlX2NMy4M>