PT

Store

Homework 3

Chirodea Mihai – Cristian

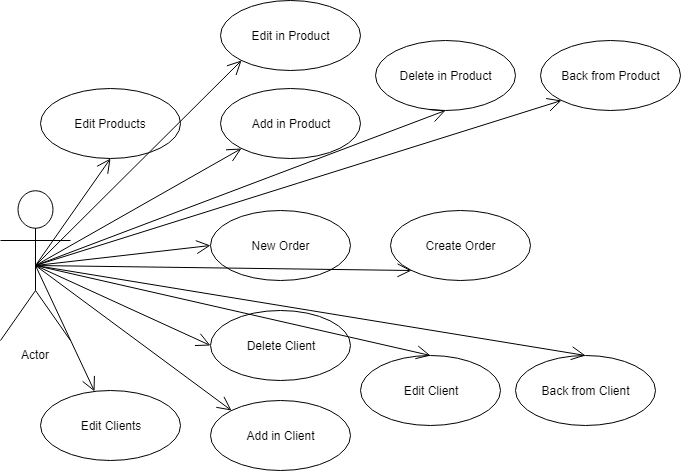
Group 30424

1. **Project Objective**

* The main objective of this project was to create a Java application capable of working with tables. The program had to perform operations such as edit, delete, add and create. The program also required an easy to use interface, created using Java’s Swing package.
* The project was realized in Eclipse and it’s able to perform the operations listed above. In order for the program to give correct results during simulation, a few steps have to be followed:
  + - **First Step**: The user has to choose the correct button in relation to the table he wants to edit.
    - **Second Step**: After pressing a button, the user is presented with a new interface in which a table with all the data is shown. The user can then select from there a row and proceed to edit its contents or delete it. He can also choose to add if he desires to do so.
    - **Third Step:** After choosing an operation, the user should check for errors in the data, for example in the clients table, we have the button to add a new client, but before we do that, we must make sure we introduce a valid email and phone format. If this step is skipped and there are errors in the data, there is a possibility that the program may crash.
* If the steps above have been followed then the program should display the correct result, else it may not display them at all.

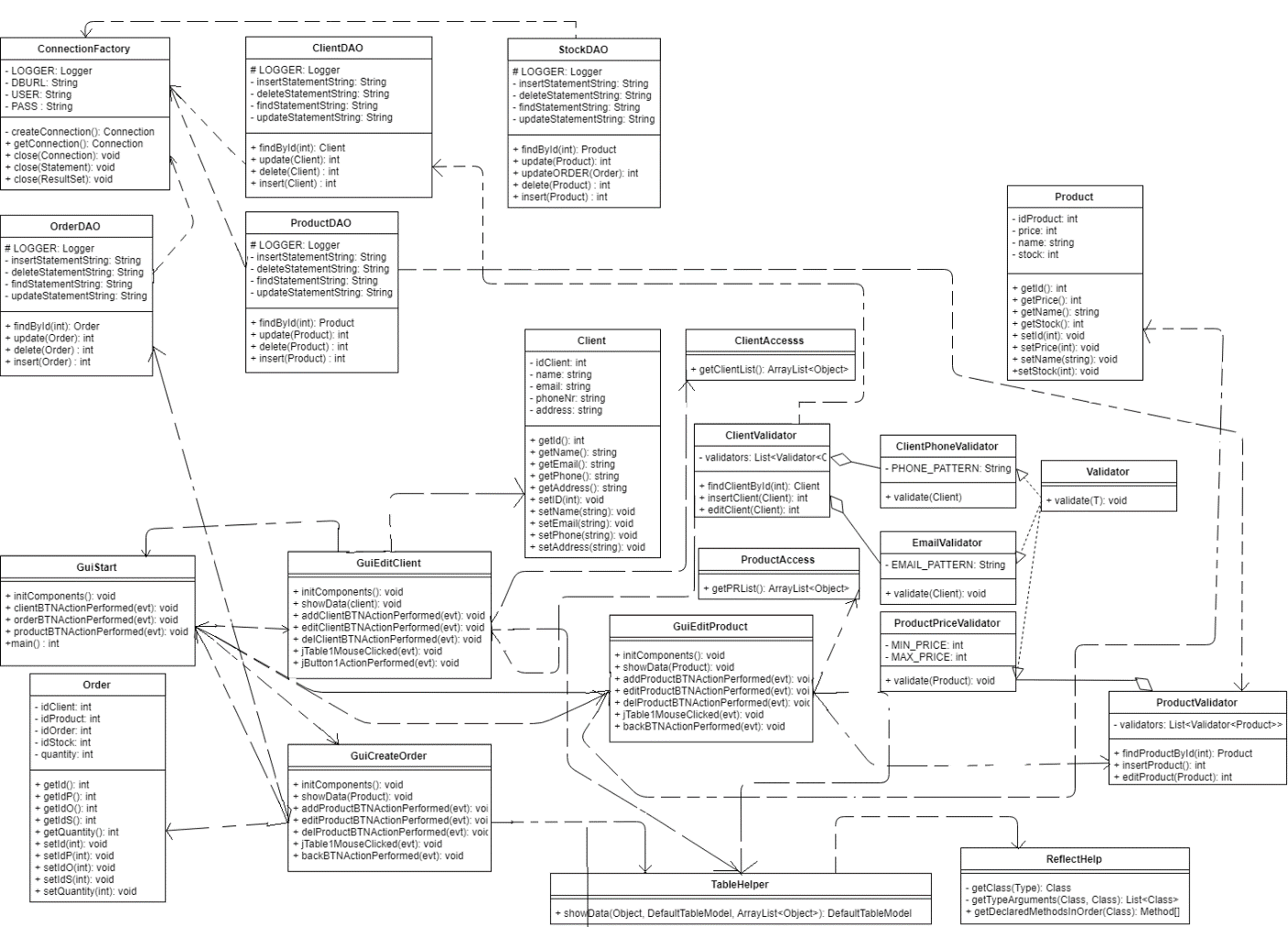
1. **Problem Analysis**

* **Functional Requirements:**
  + In order to create a program that satisfies all the needs for the operations to occur, it’s important to know how to create a connection with a database, so that we edit the data in it. Understanding that, we can decompose the problem into data structures and perform the needed operation by creating methods and classes to deal with said data structure.
  + So, first step is figuring what sort of structure to use, in order to store the clients. As the specification says, the program takes the client list from the database, and in order to be displayed, there needs to be a class that mimics the rows of each client.
  + As the list retuned by the method getClients() is of Object type, I used reflection techniques in order to get the fields and dsplay the contents correctly onto the tables.
* **Use Case:**



1. **Design**

* **UML Diagram:**

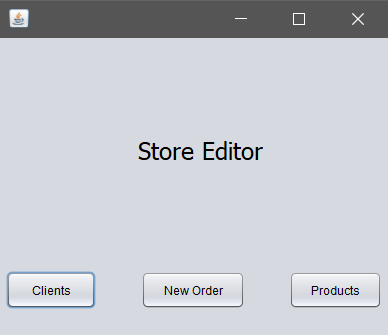
****

* **Data Structures:**
  + In order to best represent the clients, I chose to create a Class, named Client that contains the description of each individual, more precise (idClient, name, email, phone, address).
  + To best represent the 4 windows that the user will interact with I created 4 separate interfaces, one containing the first part where the user chooses on what table to operate and the other containing the second parts that displays the actual data.
* **Design Decisions:**
  + A big decision I had to make was to either use an ArrayList or a HashSet to represent the client/product lists, in the end I settled with an ArrayList as it was easier to implement.
  + This decision made a huge impact on my program as it was a lot easier to do the client operations and it took a lot less lines of code to get it to work. But as a downside I had to pay more attention to the indexes, so it would not go out of bounds.
  + Another decision I made was to add mouse listeners to the tables.This decision was quite easy to implement and didn’t require a lot of coding.
  + Another decision I made was to split the classes into packages. This decision had a huge impact as I ended up with way more classes that I anticipated.
  + Lastly I made the decision to not show anything when a new order is created as in order to show, it would have taken rewriting the Graphical user interface, which would require a lot of time and a lot of lines of code to be rewritten.

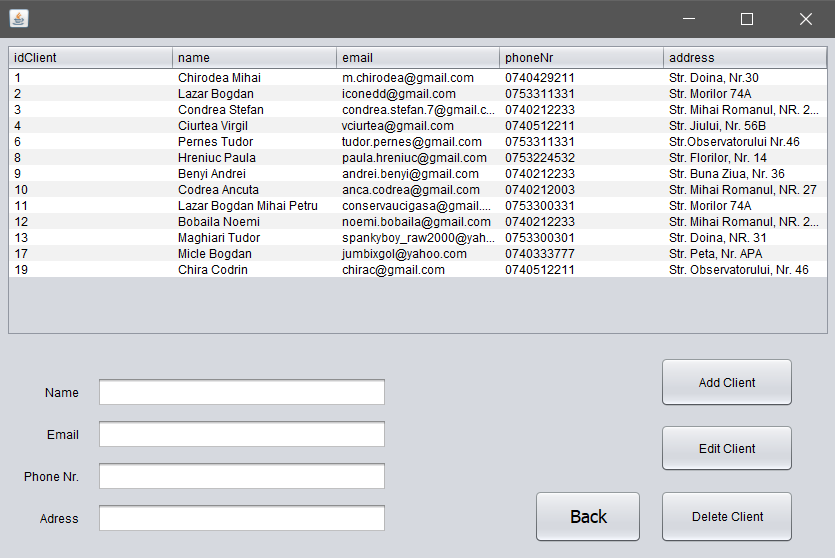
1. **Implementation**

* **Packages:**
  + In order to organize the classes, I split them into seven packages:
    - **reflectionhelper:** Contains the ReflectHelp class that deals with the problem of getting the list of methods by declared order and not by compiled order.
    - **presentation:** Contains the classes for the 4 intefaces.
    - **model:** Contains the basic classes Client, Product, Order and the TableHelper that is used to reflect the objects and show the data onto the given table.
    - **dataAccessLayer:** Contains the Client, Product, Order helpers that contain the queries necessary to extract, edit, update add nad delete data. It also contains an auxiliary class named StockDAO which is used togheter, with ProductDAO and OrderDAO.
    - **connection:** Contains the connector used in creating the link with the database.
    - **businessLayer:** Contains the Validators used in checking if introduced data is ok or not.
    - **bussinessLayer.validators:** Contains the validators used in checking certain fields (i.e. email, phone, price).
* **Clients Class:**
  + Its name is Client, it contains 4 definitions of Strings and 1 of Integer type, named idClient, name, email, phone, address. It contains methods that get and set the fields.
  + This class contains the following methods:
    - **The Constructor:** It initializes the declared fields.
    - **setID(int val):** It’s an int method that sets the arrival time of the client by receiving an int value.
    - **setName(string val):** It’s an string method that sets the service time of the client by receiving an int value.
    - **getEmail():** It’s a void method that returns the arrival time of the client .
    - **getPhone():** It’s a void method that returns the service time of the client.
    - **getAddress():** It’s a void method that returns the id of the client.
    - **setAddress(string val):** It’s a string method that sets the address of the client by receiving an int value.

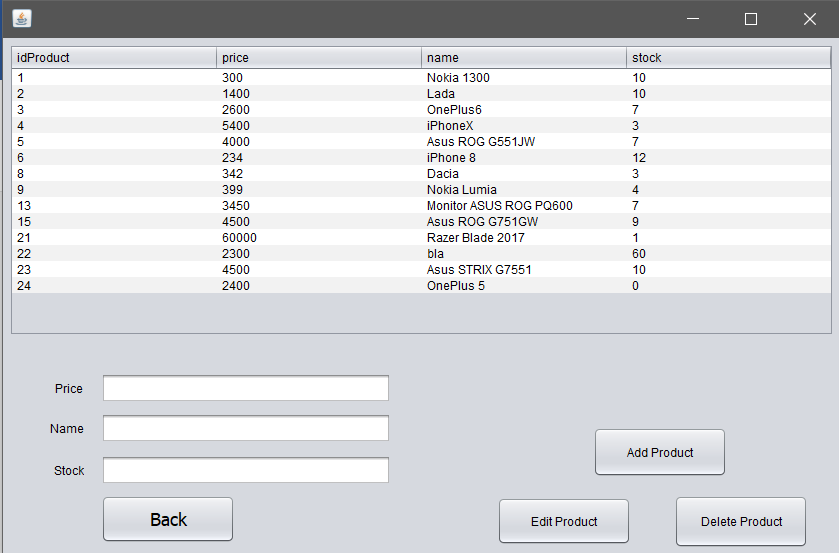
* **Product Class:**
  + Its name is Product, it contains 4 structures. First one is named idProduct, then price, name, stock.
  + idProduct stores the id of the client while name, stock and price get their respective corresponding values.
  + This Class contains the following methods:
    - **The Constructor:** It initializes the declared fields.
    - **setID(int val):** It’s an int method that sets the arrival time of the client by receiving an int value.
    - **setName(string val):** It’s an string method that sets the service time of the client by receiving an int value.
    - **getPrice():** It’s a void method that returns the arrival time of the client .
    - **getQuantity():** It’s a void method that returns the service time of the client.
    - **getStock():** It’s a void method that returns the id of the client.
    - **setStock(string val):** It’s a string method that sets the address of the client by receiving an int value.
* **Order Class:**
  + Its name is Order, it contains 5 structures. First one is named idClient, then idProduct, idOrder, idStock, quanity.
  + idProduct stores the id of the client while name, stock and price get their respective corresponding values.
  + This Class contains the following methods:
    - **The Constructor:** It initializes the declared fields.
    - **setID(int val):** It’s an int method that sets the arrival time of the client by receiving an int value.
    - **setIdP(int val):** It’s an int method that sets the service time of the client by receiving an int value.
    - **getIdS():** It’s a void method that returns stock id. .
    - **getQuantity():** It’s a void method that returns the quantity of the order.
    - **getStock():** It’s a void method that returns the id of the client.
    - **setStock(int val):** It’s a string method that sets the stock id of the order by receiving an int value.
* **TableHelp Class:**
  + Named TableHelper, it contains only one method that helps with showing data through reflection
    - **showData():** returns a model for the filled-up table.
* **ReflectHelp Class:**
  + Named ReflectHelp, it help in getting rid of the problem of getting th method list in declaration order.
* **Data Access Classes:**
  + First class is ClientDAO which contains definitions for queries in the Client Table in the database.
    - **Update: updates the table with the given values.**
    - **Delete: deletes the given entry.**
    - **Insert: inserts a new row into the table.**
    - **findById: find a client by id.**
  + Next class is ProductDAO which contains definitions for queries in the Product Table in the database.
    - **Update: updates the table with the given values.**
    - **Delete: deletes the given entry.**
    - **Insert: inserts a new row into the table.**
    - **findById: find a product by id.**
  + Next class is OrderDAO which contains definitions for queries in the Order Table in the database.
    - **Update: updates the table with the given values.**
    - **Delete: deletes the given entry.**
    - **Insert: inserts a new row into the table.**
    - **findById: find a order by id.**
* **Connection Class:**
  + Named ConnectionFactory, it helps in getting the connection to the database.
* **Business Classes and SubClasses:**
  + Used to validate input data, they provide methods such as email validator, phone validator and price validator.
* **GUI Class 1:**
  + Named GuiStart, it contains the definitions of Java.Swing elements such as buttons, labels, textFields, containers and frames, and lastly, of Jtable.
  + This class is the one responsible for creating the graphical user interface with which the user interacts.
  + As methods it has the constructor that initializes the interface and its elements and that adds listeners to the buttons, if they exist.
  + It uses groupLayout which, because of it’s constraints, can be modified however the programmer wants and can arrange elements in a way that other layouts have a hard time copying.
  + It has 3 buttons (edit Product, edit Clients, new Order) that, after the user presses any of the buttons, opens up the next interface, in which the sidata is shown and ready to be edited.
  + This class is also where the program is executed from.
  + The Interface for this class looks as follows:

****

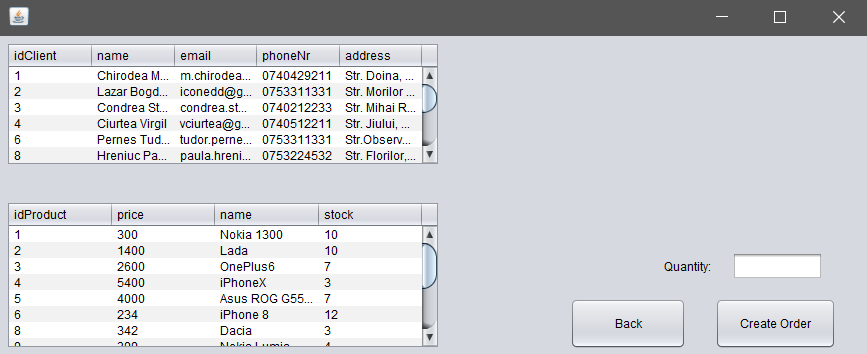
* **The Listener Classes:**
  + Each of those classes is used as a listener for each of the buttons, and when a button is pressed, the class corresponding to each button is called, executing the code in them.
  + As example, when pressing the new Order button, the program jumps to the StartButtonListener class, which in turn calls the second interface.
* **GUI Class 2:**
  + Named GuiEditClient, it’s the interface where the user sees the data coresponding to the Client table and chooses what to do with it.
  + It contains definitions of Java.Swing elements (JButton, JLabel, JFrame, Container and so on).
  + As methods it contains the constructor that initializes the interface and add the listeners to all the buttons.
  + It contains 3 buttons (Back, Add Client, Delete, Edit) which has the following functions:
    - **Back Button:** closes the current interface and re-runs the first one.
    - **Caution! Closing any window with the ‘x’ button will close the program, not the window.**
  + The interface for this class looks as follows:

****

* **GUI Class 3:**
  + Named GuiEditProduct, it’s the interface where the user sees the data coresponding to the Client table and chooses what to do with it.
  + It contains definitions of Java.Swing elements (JButton, JLabel, JFrame, Container and so on).
  + As methods it contains the constructor that initializes the interface and add the listeners to all the buttons.
  + It contains 3 buttons (Back, Add Client, Delete, Edit) which has the following functions:
    - **Back Button:** closes the current interface and re-runs the first one.
    - **Caution! Closing any window with the ‘x’ button will close the program, not the window.**
  + The interface for this class looks as follows:

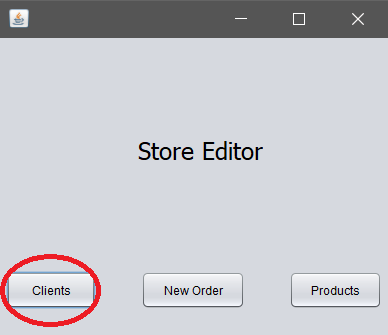
****

* **GUI Class 4:**
  + Named GuiCreateOrder, it’s the interface where the user sees the data coresponding to the Client table and chooses what to do with it.
  + It contains definitions of Java.Swing elements (JButton, JLabel, JFrame, Container and so on).
  + As methods it contains the constructor that initializes the interface and add the listeners to all the buttons.
  + It contains 3 buttons (Back, Add Client, Delete, Edit) which has the following functions:
    - **Back Button:** closes the current interface and re-runs the first one.
    - **Caution! Closing any window with the ‘x’ button will close the program, not the window.**
  + The interface for this class looks as follows:

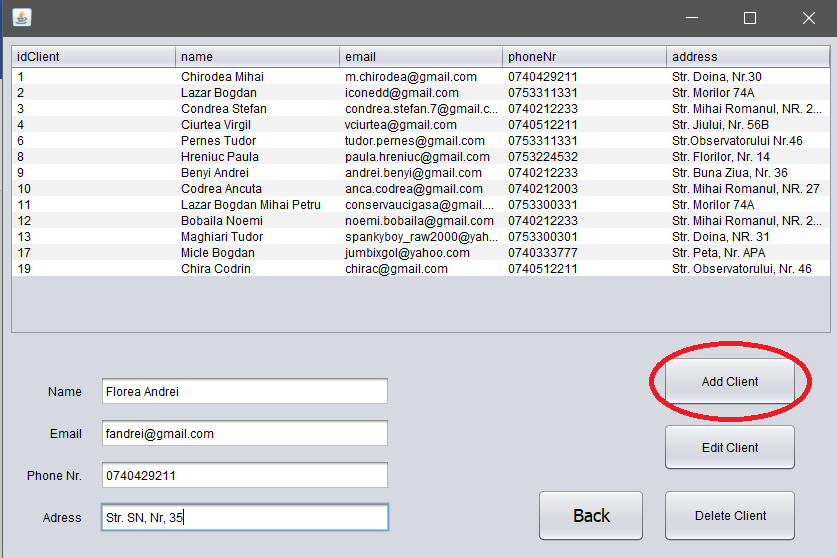
****

1. **Testing & Results**

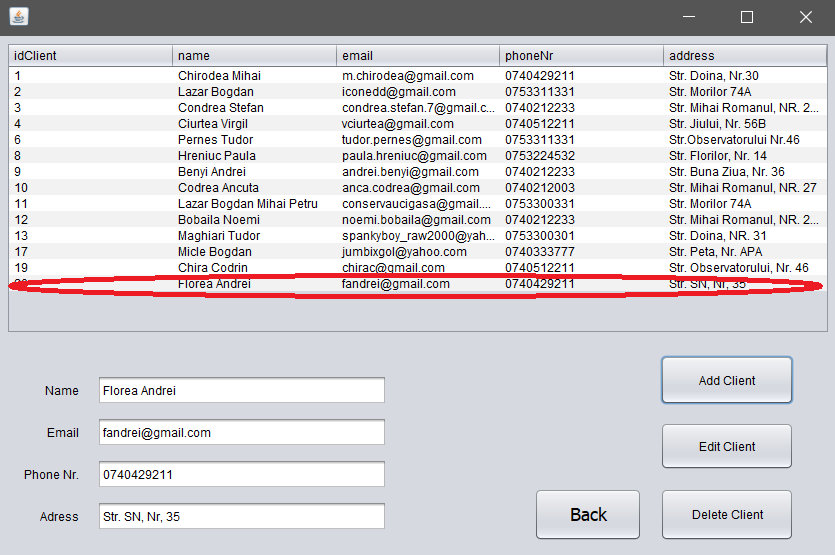
* In order to prove the correctness of the program, I have chosen the following cases and results.
  + **Test Input:**
    - * **The Client table is filled with data**
      * **WE proceed to do operations on said table**
      * **Add**
      * **Edit**
      * **Delete**
  + **Initial phase:**



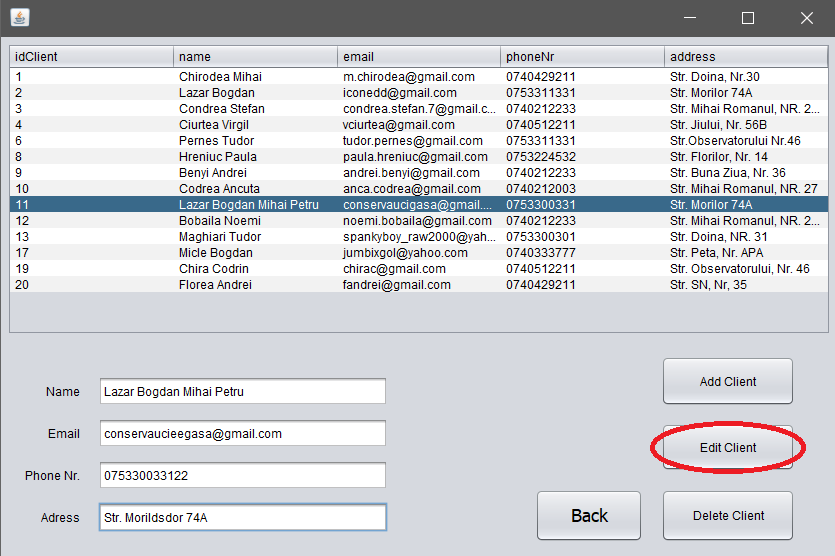
* **Add:** We can see that introducing a wrong data generates an exception.

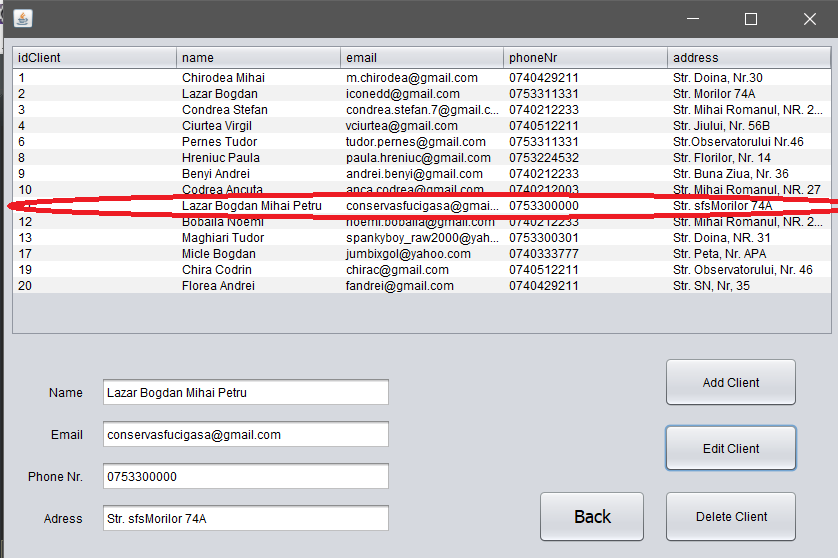


* **Next:** We see that the table is updated and we proceed with edit

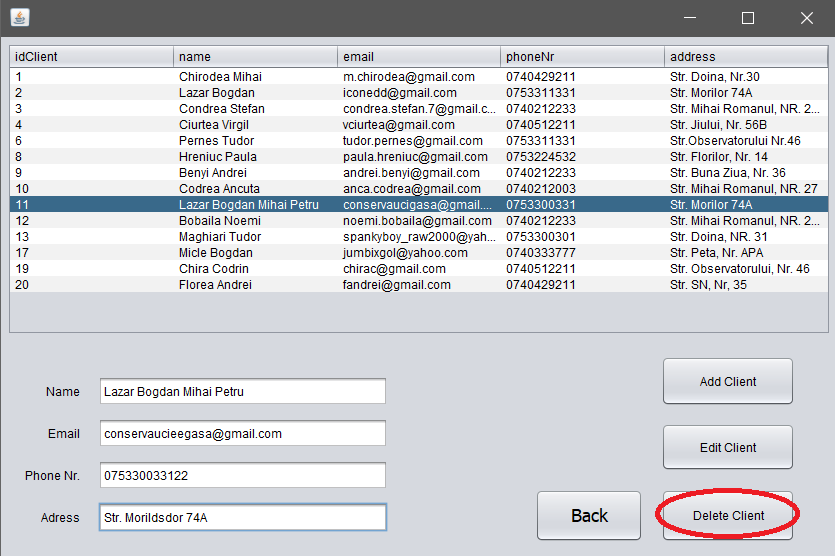


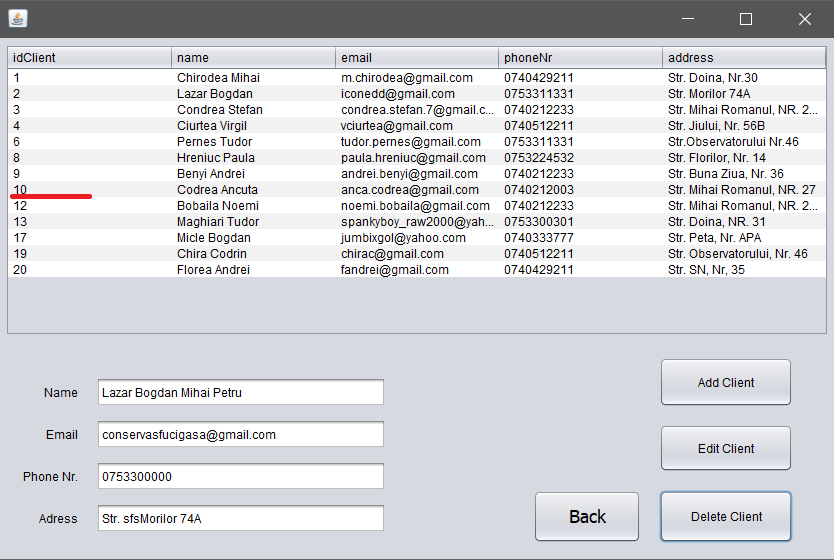
* **Next 2:** After we press edit, the selected row is overridden with what the user has introduced.





* + - **Finally:** We have the delete operation and as we ca see the table gets updated after delete is pressed.





1. **Conclusions**

* To sum it all up, I had to create a program that could communicate with a database and perform certain operations on that database. I had to use reflection connect to a database.
* **What I learned:**
  + I learned to create GUI using only code, not drag & drop like I used to do.
  + I better familiarized myself with Array Lists and OOP use of Objects.
  + I improved on working with the GUI.
  + I learned connect to a database.
  + I learned to create and use Button Listeners.
  + I learned how to link multiple interfaces.
  + I learned how to use reflection techniques in order to get object properties.
  + I learned how a GUI updates(Hint: It’s a Thread)
  + I learned how to use Array Lists
* **Future improvements:**
  + Adding option to save results.
  + Displaying results from Order after New Order is pressed.
  + Making the program more flexible to user input errors.

1. **Bibliography**

* <https://stackoverflow.com/questions/3481828/how-to-split-a-string-in-java>
* <https://docs.oracle.com/javase/tutorial/uiswing/components/panel.html>
* <https://www.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html>
* <https://docs.oracle.com/javase/tutorial/uiswing/layout/spring.html>
* <https://stackoverflow.com/questions/5993779/use-string-split-with-multiple-delimiters>
* <https://stackoverflow.com/questions/22212412/sort-a-set-in-reverse-order>
* <https://stackoverflow.com/questions/37628/what-is-reflection-and-why-is-it-useful>
* <http://www.java2s.com/Tutorial/Java/0125__Reflection/ListmethodsofaclassusingReflection.htm>
* <https://stackoverflow.com/questions/5001172/java-reflection-getting-fields-and-methods-in-declaration-order>
* https://github.com/wmacevoy/kiss/blob/master/src/main/java/kiss/util/Reflect.java