PT

Bank

Homework 4

Chirodea Mihai – Cristian

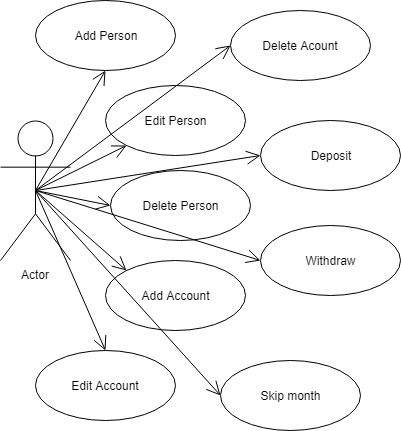
Group 30424

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.

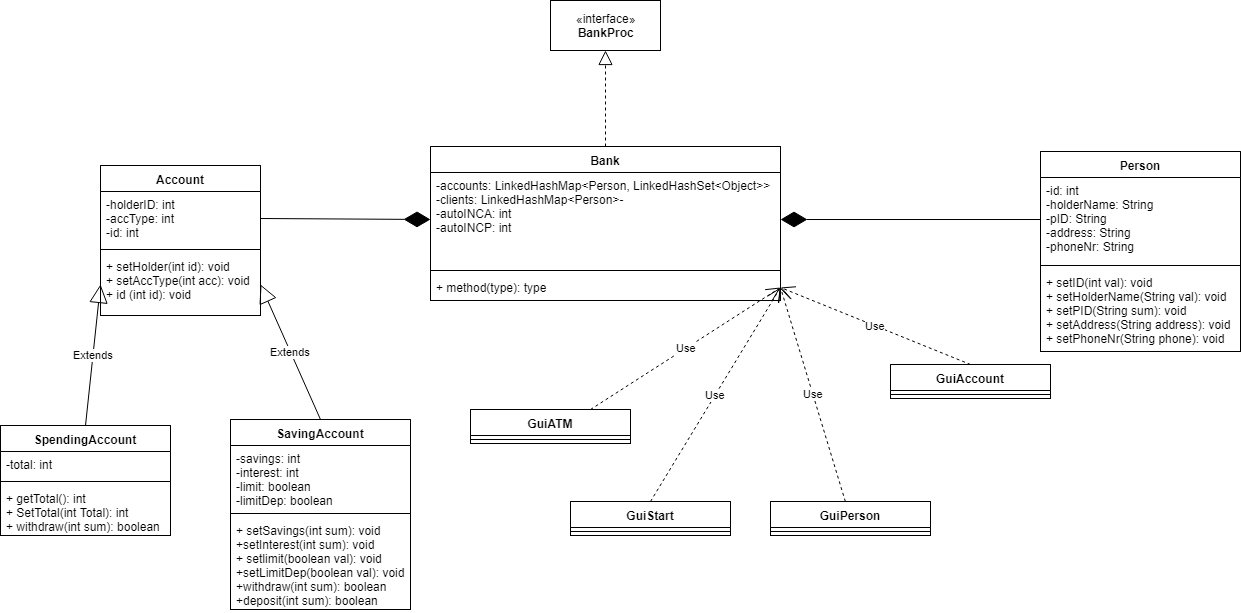
1. **Problem Analysis**

* **Functional Requirements:**
  + In order to create a program that satisfies all the needs for the operations to occur, it’s important to how to use a HashMap. 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 uses the Clients as a key for the HashMap, and in order to be displayed, there needs to be a class that mimics the rows of each client.
  + The list retuned by the method getClients() is of Person type, which I then used to extract data from each field.
* **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, PID, 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 list, in the end I settled with a LinkedHashSet 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.
  + 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.
  + I made a decision to create a separate interface to simulate withdrawals and deposits. This decision wasn’t so hard to implement and didn’t require a lot to do.
  + Lastly I made the decision to use a LinkedHashMap to store user accounts. This decision was a bit confusing to implement until I got a better understanding of how exactly this map worked.

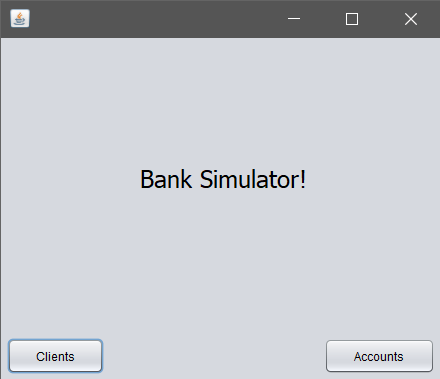
1. **Implementation**

* **Packages:**
  + In order to organize the classes, I split them into several packages:
    - **accounts:** Contains the classes for the Account types. Said class names are Account, SavingAccount, SpendingAccount.
    - **gui:** Contains the classes for the 4 intefaces.
    - **client:** Contains the Class Person that is used as a key in the LinkedHashSet.
    - **bank:** Contains 2 Classes, an interface which is implemented by Bank. In here most of the operations take place, such as editing, deleteing or adding accounts and persons.
    - **test:** Contains the tester class (Junit).
* **Person Class:**
  + Its name is Client, it contains 4 definitions of Strings and 1 of Integer type, named idClient, name, PID, 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.
    - **getpID():** 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.

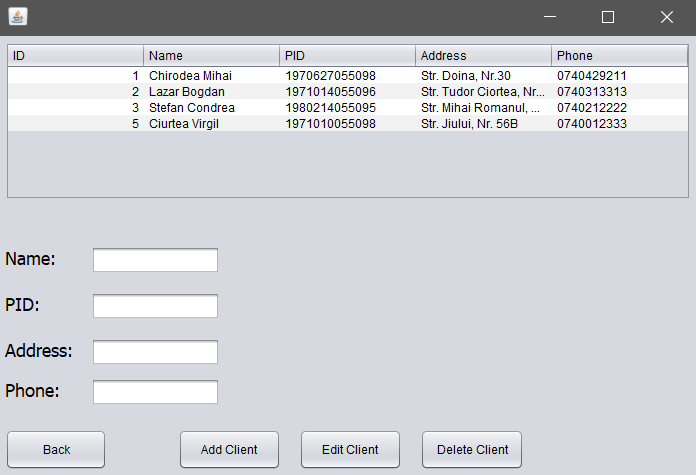
* **Account Class:**
  + Its name is Account, it contains 3 structures. First one is named id, and then, there is accType and lastly holderID
  + id stores the id of the account while accType stores the account type(Saving or Spending) and holderID stores the id of the person.
  + This Class contains the following methods:
    - **The Constructor:** It initializes the declared fields.
    - **setID(int val):** It’s an int method that sets the account id.
    - **sethID(int val):** It’s a void method that sets the holder id.
    - **getaccType ():** It’s an method that returns the account type.
    - **getID():** It’s an int method that returns the ID of the account.
    - **getHolder():** it’s an int method that returns the id of the holder.
* **SavingAccount Class:**
  + Its name is SavingAccount, it contains 3 structures. First one is named
  + limit (a Boolean value that represents if there was a withdrawal), limitDep(A Boolean value that stores if there was a deposit or not), savings( an int value that represents the total amount of money stored).
  + This Class contains the following methods:
    - **The Constructor:** It initializes the declared fields.
    - **setLimit(Boolean val):** It’s a boolean method that sets limit.
    - **setLimitDep(Boolean val):** It’s a boolean method that sets the deposit limit.
    - **getSavings():** It’s a void method that returns the total ammout of money saved .
    - **setSavings(int val):** It’s a void method that sets the vaue of the savings
    - **getLimit()** It’s a boolean method that returns the limit.
    - **getLimitDep():** It’s a Boolean meothod that returns the value of limitDep.
* **Bank Classes:**
  + First class is Bank which contains definitions for the Accounts and Persons.
    - **WriteAccount: Edits the given account with the new values.**
    - **ReadAccount: returns the list of accounts corresponding to the given client.**
    - **AddClient: Adds a new client to the list.**
    - **AddHolderAccount: adds a new account to the list corresponding to the client.**
    - **RemHolderAccount: Removes the given account from the list.**
    - **RemClient: Removes the given client from the list.**
    - **Edit Client: Edits the given client with the given values.**
  + Next class is BankProc which is an interface that is implemented by Bank:

It Contains the definitions the methods listed above.

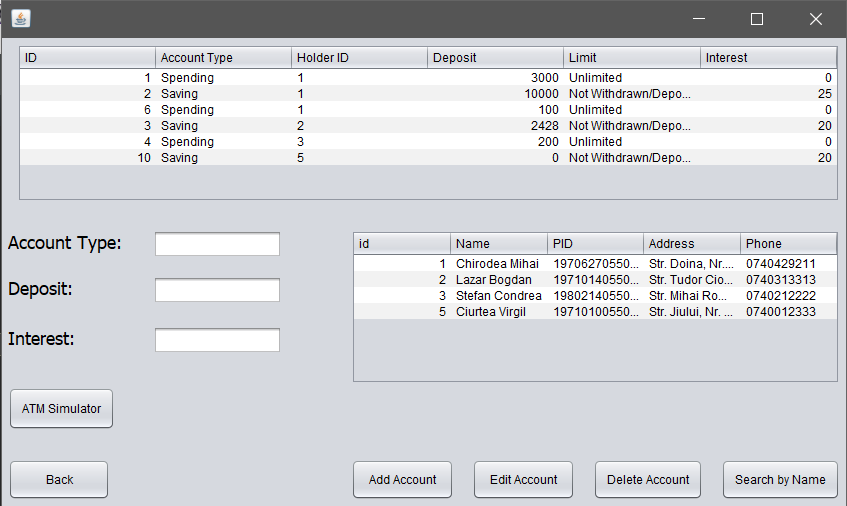
* **GUI Class 1:**
  + Named GuiStart, it contains the definitions of Java.Swing elements such as buttons, labels, textFields, containers and frames.
  + This class is the one responsible for creating the graphical user interface with which the user interacts.
  + As methods it has the constructor that 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 2 buttons (Accounts, Clients) that, after the user presses any of the buttons, opens up the next interface, in which the data 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:

****

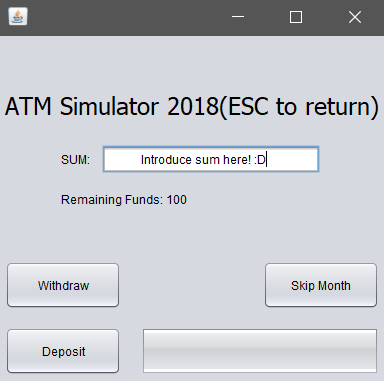
* **The Listener Classes:**
  + Each of those classes is used as a listener for each of the buttons, and when a button is pressed, the class corresponding to each button is called, executing the code in them.
  + As example, when pressing the new Order button, the program jumps to the StartButtonListener class, which in turn calls the second interface.
* **GUI Class 2:**
  + Named GuiPerson, 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 GuiAccount, it’s the interface where the user sees the data coresponding to the Accounts 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 4 buttons (Back, Add Account, Delete, Edit, ATM Simulator) 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:

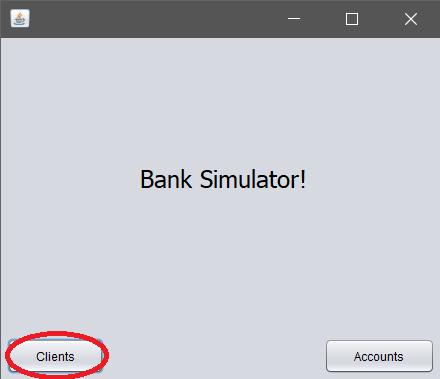
****

* **GUI Class 4:**
  + Named GuiATM, it’s the interface where the user simulates a withdrawal or deposit on the given account.
  + 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 (Deposit, Withdraw, Skip month) 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:

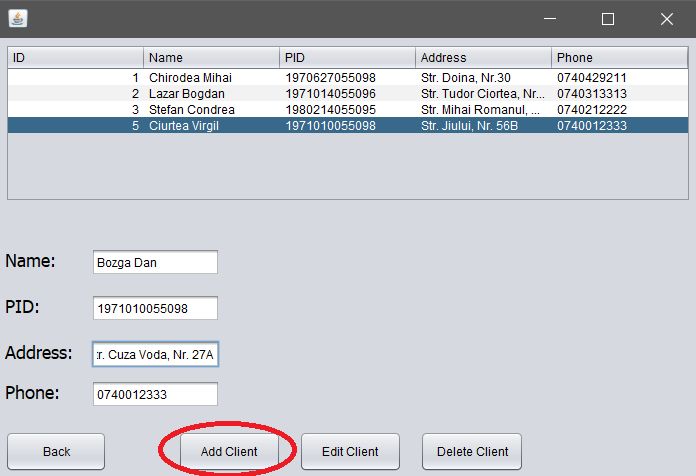
****

1. **Testing & Results**

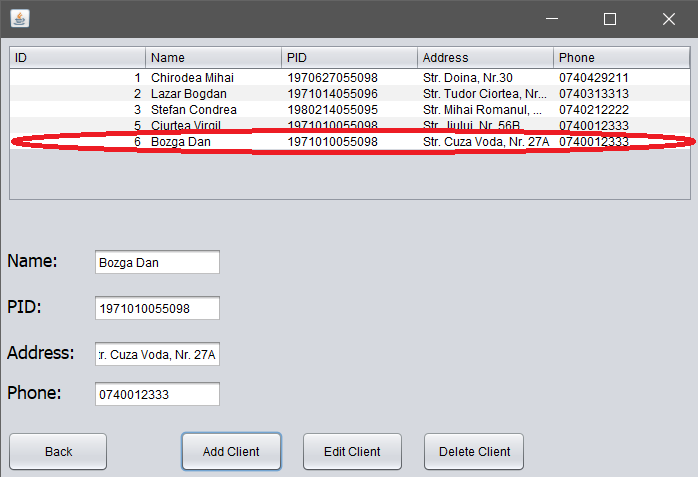
* In order to prove the correctness of the program, I have chosen the following cases and results.
  + **Test Input:**
    - * **The Client table is filled with data**
      * **WE proceed to do operations on said table**
      * **Add**
      * **Edit**
      * **Delete**
  + **Initial phase:**



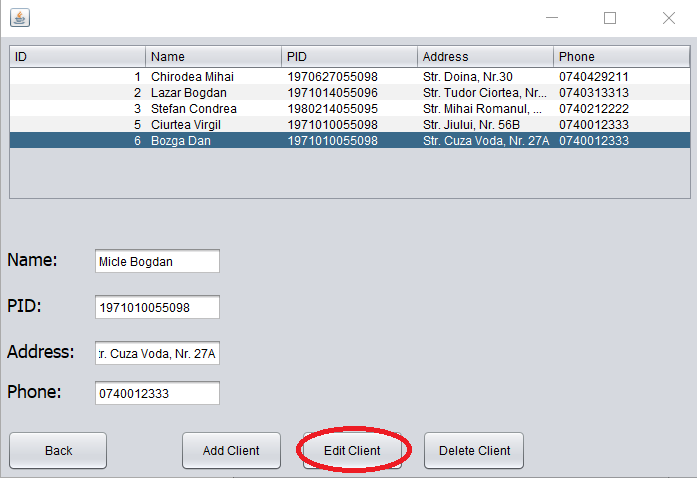
* **Add:** We can see that introducing a wrong data generates an exception.

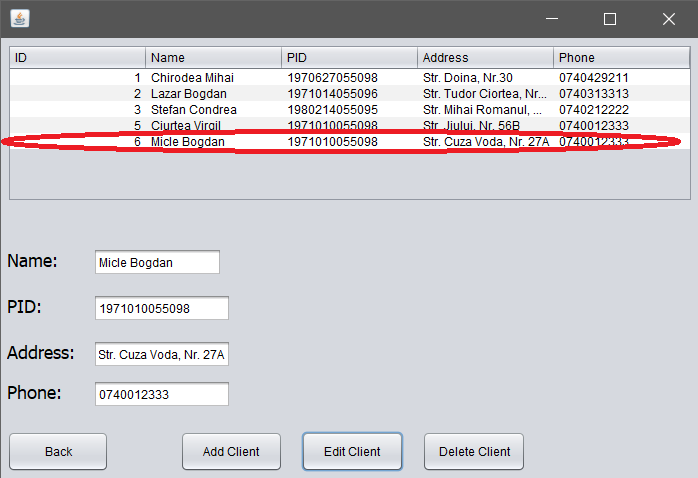


* **Next:** We see that the table is updated and we proceed with edit

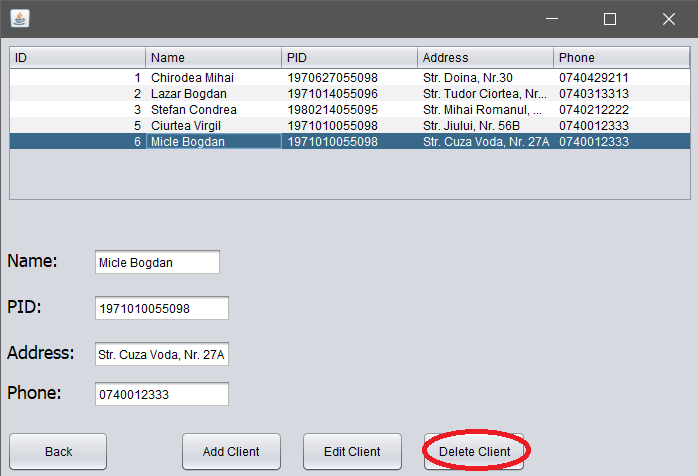


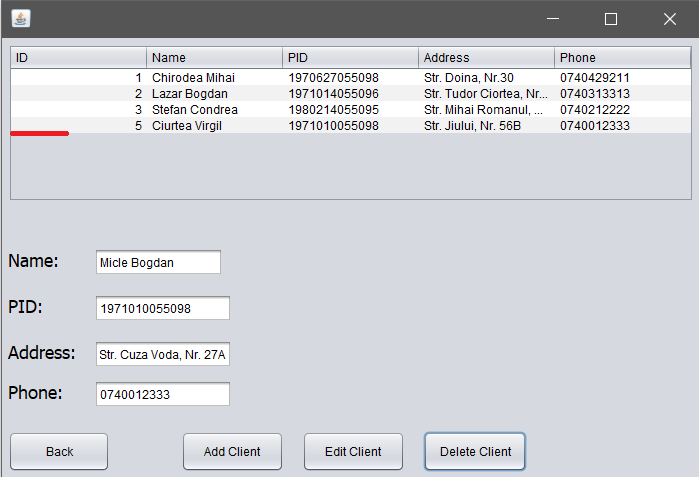
* **Next 2:** After we press edit, the selected row is overridden with what the user has introduced.





* + - **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 the bank class and perform certain operations on it. I had to use serialization, junit and design by contract methods.
* **What I learned:**
  + I learned to create GUI using only code, not drag & drop like I used to do.
  + I better familiarized myself with LinkedHashMaps and OOP use of Objects.
  + I improved on working with the GUI.
  + I learned to create and use Button Listeners.
  + I learned how to link multiple interfaces.
  + I learned how a GUI updates(Hint: It’s a Thread)
  + I learned how to use HashSets
* **Future improvements:**
  + Adding option to save results.
  + Making the program more flexible to user input errors.

1. **Bibliography**

* <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>