Assignment 4

Bank

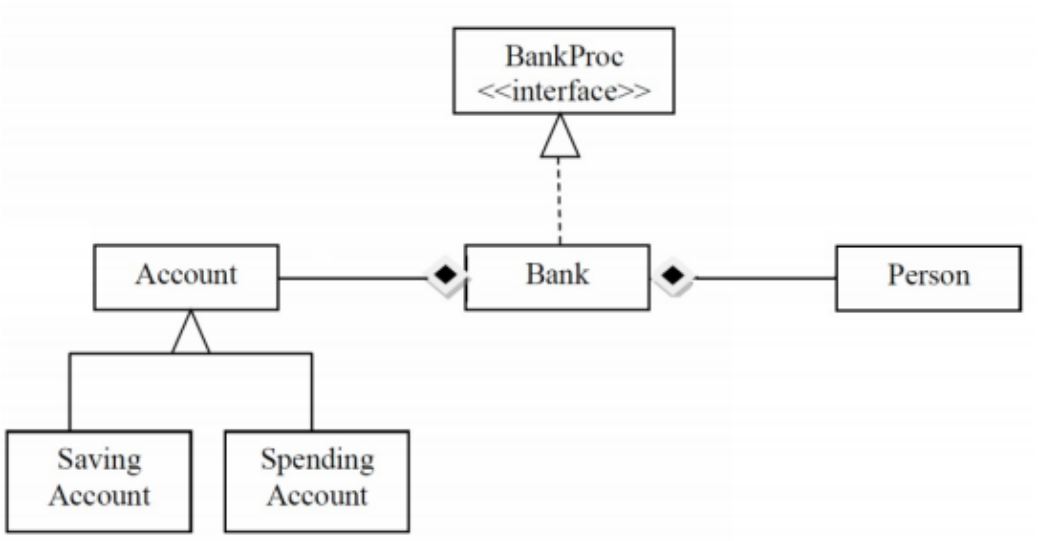
- Documentation –

15.05.2018

Linca Paul Tudor

Group: 30422

1. Task



Implement the class diagram from the homework specification. Choose wisely the appropriate data structures for saving the Persons and the Accounts.

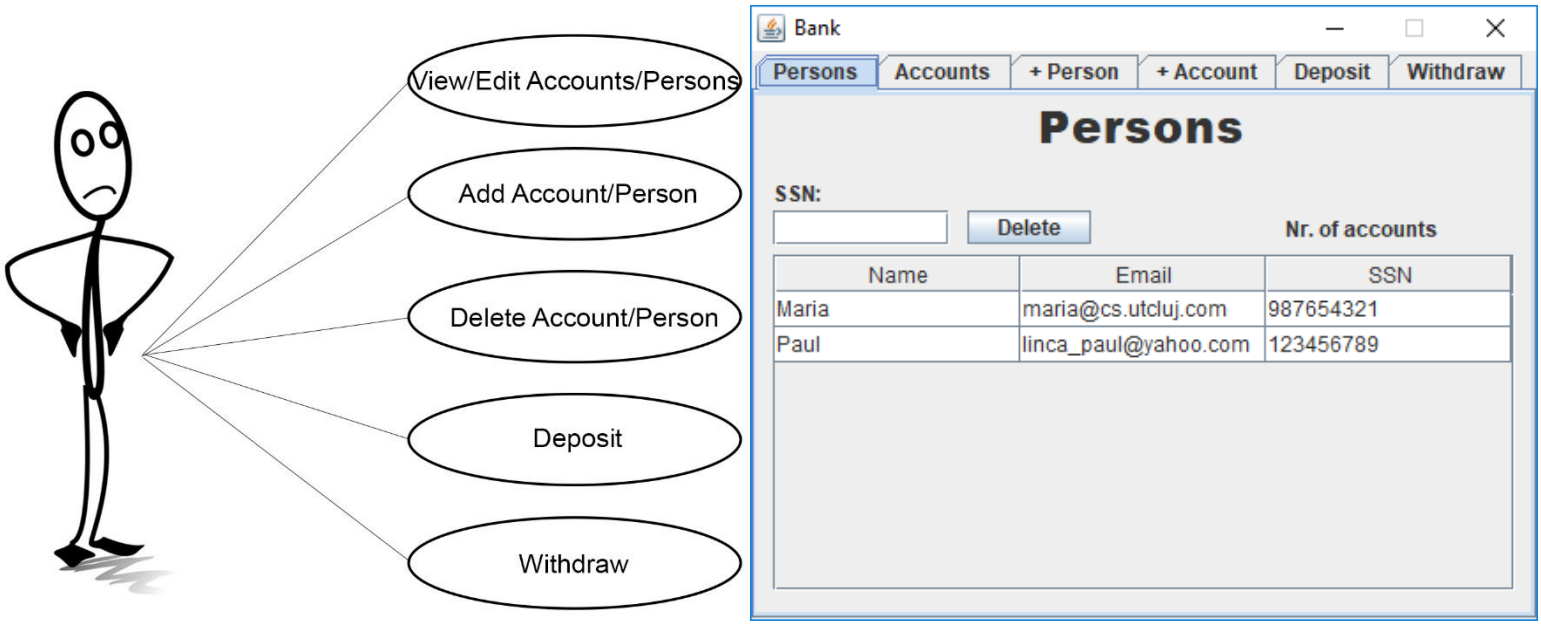
Thus, the first objective of this assignment is implementing the given diagram. Secondly, the data structures which the application will work with must be chosen wisely. All the other steps will be described later in this document.

2. Brief description:

The assignment consists of developing an application which simulates a bank. The bank must be able to provide its customers 2 types of accounts: Saving accounts and Spending accounts. The saving account allows a single large sum deposit and withdrawal and computes an interest during the deposit period. The spending account allows multiple deposits and withdrawals, but does not compute any interest.

The user or client must be presented with a smooth Graphical User Interface to be able to add a new person or account, edit an existing person or account, delete an existing person or account and view all the persons and accounts of the bank.

3. User Diagram



The user is presented with a simple and smooth graphical interface. It consists of 6 tabs, each containing a panel on which one or more actions can be performed. On the "persons" tab, the user can view all the persons and delete one based on the SSN (Social security number). Similarly, on the accounts tab the user can view all the accounts and delete one based on its id. The "+Person" and "+Account" tabs are used to add a new person or account respectively. And the last two tabs are have a name that's pretty self explanatory, and are used to perform operations on accounts.

Thus, for example, a user can go to the "+Person" tab, fill the fields with his/her data and press the "add" button and then go to the "Persons" tab and see that the data is correct. After that, in the "+Account" that the user will be able to add a new account and view it in the "Accounts" tab. Lastly, the user can deposit or withdraw money on and from the created account on the last 2 tabs.

In my opinion, the graphical user interface is pretty straight forward in how the user is supposed to use the system.

4. Assumptions

It is assumed that the user is wise and will not give wrong inputs, such as providing a string of characters in the “SSN” fields instead of providing a field of integers as well as not making it way too long. The same goes for the “ID” field used for deleting an account.

Another assumption is that the user will not put letters in the text fields that are used to communicate the amount of money to deposit or withdraw from an account.

5. Data structures

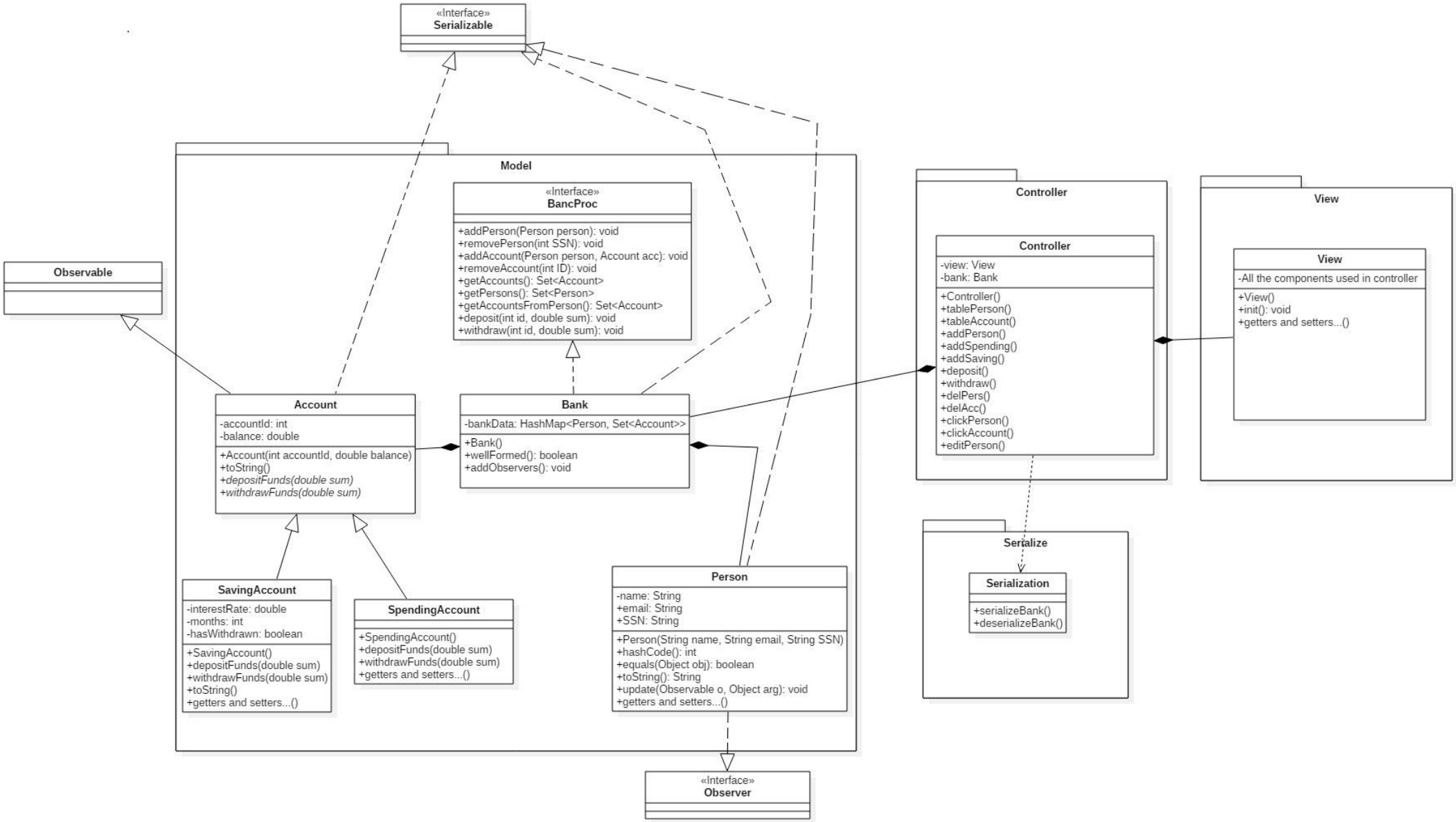
It is mandatory to choose an appropriate data structure for saving the Persons and the Accounts.

The hash table is a data structure used to implement an associative array (by mapping keys to values) with constant access time to its elements. According to the theory, an associative array/ map contains key-value pairs. When implementing a hash table, the key is used to compute an index.

For this assignment, I went with the HashMap and the Set from the Collection interface.

I used the HashMap to save the couples made up of a person and a Set of accounts. It is easy to work with a HashMap since it contains values based on keys. Thus, in my case, the values are the sets of accounts and the keys are the person. This way, each set of accounts corresponds to a person.

6. Class diagram



A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

I designed the problem based on the Model – Controller – View (MVC) architectural pattern. It is commonly used for developing user interfaces that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. The MVC design pattern decouples these major components allowing for efficient code reuse and parallel development.

- The model is the central component of the pattern. It expresses the application's behavior in terms of the problem domain, independent of the user interface. It directly manages the data, logic and rules of the application. Thus, my model consists of the classes presented in the diagram provided by the problem.
- The view consists of the Graphical User Interface (GUI) which will be described later in this document.
- he controller accepts input and converts it to commands for the model or view. So it basically links the model and the view.

Other than that I also used another package called “Serialize” which contains a class that enables serialization.

I chose the MVC architectural pattern because it enables the logical grouping of related actions on a controller together, it creates components that are independent of one another, thus enabling the reusability of the code, it has low coupling meaning that its components has, or makes use of, little or no knowledge of the definitions of other separate components and it is good practice for future group projects I will be a part of because multiple developers will be able to work simultaneously on the mode, controller and views.

7. Class description

- View (GUI)

This class represents the GUI (Graphical User Interface). The private attributes are the GUI's elements which are needed for the correct execution of the simulation. They also have the getters and setters needed to perform the required task.

The init() method is a pretty straight forward window building method. It uses JTextFields, JButtons, JLabels, a JFrame, JScrollPanes, JPanels and JTables. The GUI has a tabbed view for simple navigation.

- The "Persons" tab displays in a JTable that is contained inside a JScrollPane, all the persons. The JTextField expects to receive a SSN of a person and upon pressing the JButton the person to which the SSN inputted corresponds will be deleted from the table. Also, the table has click listeners implemented, thus, by clicking a row, on the label above the table, it will be displayed how many accounts the person has;
- The "Accounts" tab works similarly to the "Persons" tab but for accounts, obviously. The click listeners of the table are implemented in such a way that upon clicking a row, the type of the account will be displayed on the JLabel;
- The "+Person" tab has 3 text fields which expect to receive information about the person to be added or edited and the action will be completed by pressing one of the two buttons labeled "Add" and "Edit";
- The "+Account" has the same three text fields as the previous tab and expect similar input but upon pressing a button a corresponding account will be created and linked to the person described in the text fields. For

Saving type accounts the extra text field labeled “Period:” must be filled in order for the program to work correctly;

- The “Deposit” tab consists of two text fields that expect as input the ID of an account and a sum to be deposited to said account. The action is completed by pressing the “deposit” button.
- Similarly to the previous tab, the “Withdraw” tab consists of two text fields that expect as input the ID of an account and a sum to be withdrawn from said account. The action is completed by pressing the “withdraw” button.

All the tabs have labels all over them to provide help to the user.

The constructor of the View class simply calls the init() method.

This class is contained in the package of the same name.

- Person

This class, as the name would suggest, represents each client of the bank. It has three private attributes: name, email and SSN. All three are of String type. The names of the attributes are pretty descriptive so a detailed explanation is not necessary for them.

In the class constructor the attributes are initialized with the values of the arguments.

The method toString() prints the person in the following format “P(name, email, SSN)”.

Taking into account the mechanisms for get and put of HashMap and that person is the key of the HashMap, this class must override the methods hashCode() and equals(Object obj).

The method hashCode() generates a hash code based on the attribute values of the class instance.

The method determines whether the Person object that invokes the method is equal to the object that is passed as an argument.

This method is called whenever the observed object is changed.

The rest of the methods are the getters and the setters for the attributes of the class.

- Account

The Account class represents each account of the persons. Its attributes are accountId and balance that have very suggestive names.

In the class constructor the attributes are initialized with the values of the arguments.

The method toString() prints the person in the following format “A(id, balance)”.

The methods depositFunds() and withdrawFunds() are both abstract and are meant to be implemented in the subclasses of Account that will be described next.

The rest of the methods are the getters and the setters for the attributes of the class.

This class also extends the observable class, it being the observer and implements the Serializable interface.

- SpendingAccount

This class is a subclass of Account that represents a spending account. It needs no supplementary attributes.

This class’ constructor invokes the parent class instructor through the keyword ‘super’.

The method depositFunds(double sum) adds ‘sum’ from the balance of the account.

The method withdrawFunds(double sum) subtracts sum from the balance of the account only if it is possible.

- SavingAccount

  This class is a subclass of the Account class. It has as supplementary attributes interestRate, months and hasWithdrawn.

  In the class constructor the attributes are initialized with the values of the arguments.

  The methods depositFunds() and withdrawFunds() are similar to the methods from the previous class except that the sum deposited is multiplied by the interestRate and the months and that they can only work once.

  The toString() methos is Overriden to provide a more in depth display of the account.

  The rest of the methods are the getters and the setters for the attributes of the class.

- Bank

  This class represents the bank itself. It has as its only attribute a HashMap that has as key a Person and as value a Set of accounts.

  The class constructor initializes the HashMap.

  The method wellFormed() checks whether the HashMap has been constructed correctly.

  The method addPerson(Person person) is used to add the person in the arguments to the HashMap and puts as values a new Set of accounts.

  The method removePerson(int ssn) removes the person whose ssn matches the argument from the hashMap.

The methods addAccount() and removeAccount() have the same functionality as the previous two methods.

The deposit(int id, double sum) deposits ‘sum’ money to the account with id ‘id’.

The withdraw(int id, double sum) withdraws ‘sum’ money from the account with id ‘id’.

The others are kind of setters and getters.

All these methods are part of the model package.

- Serialization

This class is used to implement the serialization of the bank. The methods of this class are used to serialize and deserialize an object of type class. The serialization will be described more in depth further in the document.

This class is contained in the package ‘serializer’.

- Controller

This class, as the name would suggest, represents the controller of the application. This class accepts input and converts it to commands for the model or view.

It has as attributes: bank and view, each of the type of the same name.

The constructor of the class initialize the two attributes and deserialize the Bank object in the .bin file in which it has been serialized and also calls the other methods.

The methods tablePerson() and tableAccount() refresh the JTables in which the persons and accounts are displayed by removing every row and adding them again to the table. This

refreshes the JTable because the addRow() method refreshes the table.

The methods clickAccount() and clickPerson() implement clickListeners to the JTables. What they do has been described in the View class description.

The rest of the methods have names that are self explanatory and implement actionListeners on the JButtons. They work based on the information in the JTextFields.

This class is part of the controller package.

8. Serialization

Serialization is the process of translating data structures or object state into a format that can be stored or transmitted and reconstructed later. When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object.

The classes ObjectInputStream and ObjectOutputStream are high-level streams that contain the methods for serializing and deserializing an object.

The writeObject(Object obj) method serializes an Object and sends it to the output stream.

This readObject() retrieves the next Object out of the stream and deserializes it.

9. Testing

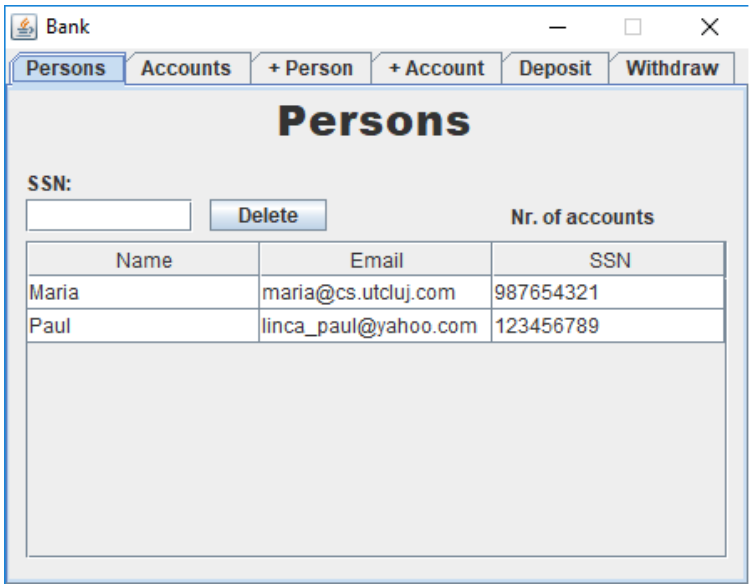
For the testing I designed a test driver using Junit. JUnit is a unit testing framework for the Java programming language. The testing methods all test the most important actions that a user can make when using the application: adding a person, removing a person, adding an account, removing an account, depositing money to an account and withdrawing money from an account.

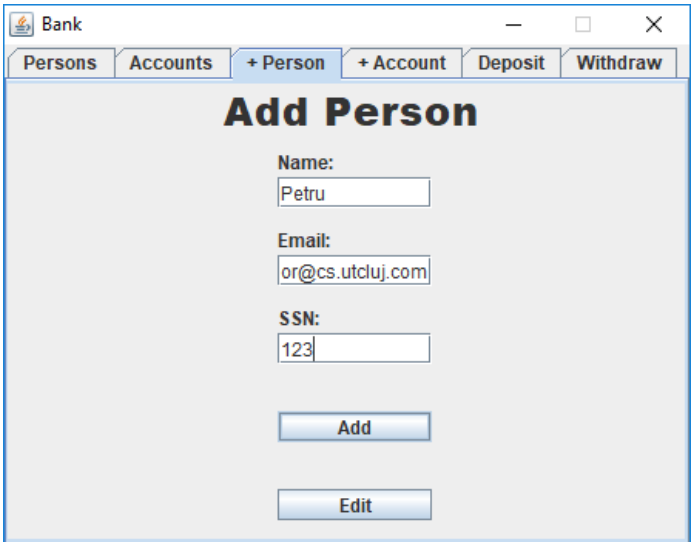
As it can be seen when running the test driver, all the tests have resulted positive meaning that my implementation of these action are correct.

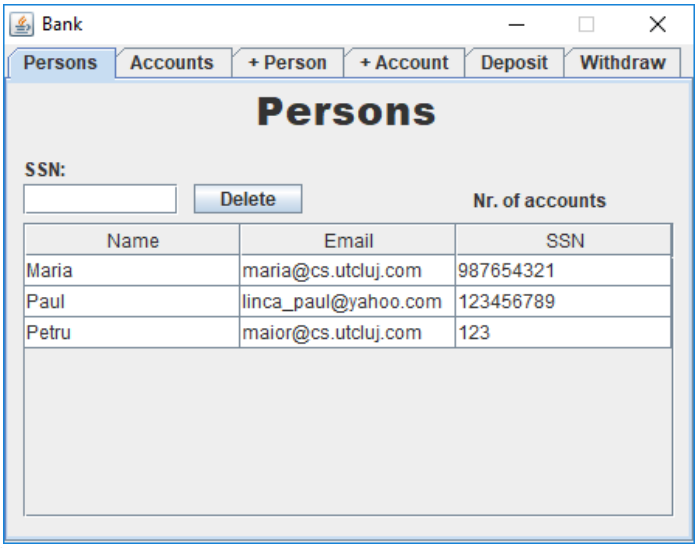
```
public void testAddPerson()
{
    Bank bank = new Bank();
    Person person1 = new Person("Paul", "linca_paul", "123456789");

    bank.addPerson(person1);

    assertTrue(bank.getBankData().containsKey(person1));
}
```

This is the test for the addition of a person. First, a bank is created, then a person is created. After that the person is added into the bank data and finally it is checked if the bank data contains the person. The assertTrue() method checks whether the expected value is true or not.

10. Results







These screenshots show that the process of adding a person is intuitive and correctly handled by the application. Thus, the implementation of this action is done correctly. The first screenshot shows all the persons in the bank. The second one shows that the fields are filled with data and the button add is pressed. The third and final screenshot shows that the person with the data in the fields has been successfully added to the "database".

11. Conclusion

In my opinion, this fourth project has been chosen very wisely. It is a very good way of polishing our OOP skills and of learning new things. Personally I have learned Serialization, how to use HashMaps, how to implement an Observer Design Pattern and how to implement a design by contract.

Further improvements to the programs would be:

- Implement the Observer Design Pattern correctly.
- Getting rid of some unexpected Exceptions
- Giving the two types of accounts more distinctive attributes and methods

12. Bibliography

https://en.wikipedia.org

http://www.coned.utcluj.ro/~salomie/PT_Lic/

https://softwareengineering.stackexchange.com

https://www.google.ro