**Assignment 4**

**Programming Techniques**

Bank Application

**Student:**Boian Maria Andreea

**Group**:30423

Teacher: Ioan Salomie

Lab assistant: Claudia Pop

2 nd semester 2017-2018

1.**Assignment objective**

**1. 1. Task description**

Create a graphical interface with a window for performing different operantions on clients of a bank : add new client, update client, delete client, view all clients in a table (JTable), view all accounts of a clients(JTable), a Window for account operations: add new account to a certain client, edit account, delete account , update account. Another window for performing transactions on a client accounts , like deposit or withdraw. User application user will be able to select an existing account, select an existing client,select the type of transaction and perform it . In case that there are not money to perform a transaction, a message will be displayed in console. After the transaction is finalized, the user can see the resulted account or person . Implement a test driver for the system. The account data for populating the Bank object will be loaded/saved from/to a file. .Define the interface BankProc (add/remove persons, add/remove holder associated accounts, read/write accounts data, report generators, etc). Specify the pre and post conditions for the interface methods. 2. Define and implement the classes Person, Account, SavingAccount and SpendingAccount. Other classes may be added as needed (give reasons for the new added classes). 3. An Observer DP will be defined and implemented. It will notify the account main holder about any account related operation. 4. Implement the class Bank using a predefined collection which uses a hashtable. The hashtable key will be generated based on the account main holder (in RO. “titularul contului”). A person may act as main holder for many accounts. Use JTable to display Bank related information. 4.1 Define a method of type “well formed” for the class Bank. 4.2 Implement the class using Design by Contract method (involving pre, post conditions, invariants, and assertions).

**1.2. Application description**

The application allows a user to perform several operations on clients and accounts, like insert, delete update or see a the corresponding results in the form of a JTable. The user can also choose a client and a account and place an perform transactions (deposit,withdraw) money for that client . The result will be loaded/saved from/to a file using serialization.

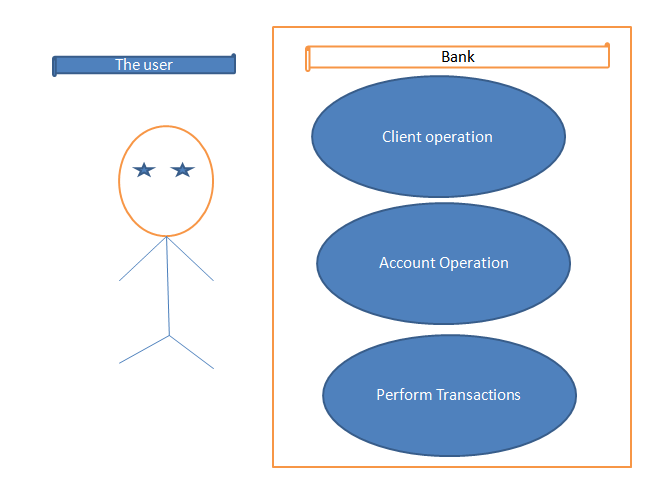
**2.Problem analysis and assumptions**

**2.1. General overview**

This application should be able to fulfill all the requirements in order to display, modify, and keep track of clients and accounts. The accounts ,clients and orders are stored in a HashMap structure. This way, all the data is easier to retrieve and accessed from different computers.

**2.2. Assumptions**

In order for the application to work correctly and to avoid unhandled exceptions, some assumptions have to be made in respect to the input. Before attempting to start the simulation, the user should leave no input field unfilled. Moreover, we assume that each input is a positive, non-zero integer Also, the user should not attempt to input an account or client with an id that is already registered in the bank .The user should also not input a string in the textfield of a integer or the other way around (i.e age in place of id ) .



**3.Use cases:Simulate**

- Summary: This use case allows the user to simulate and observe the results of doing some simple operations over clients and their accounts from the bank.

- Actors: The User

- Preconditions: None

- Main success scenario: 1. The user inputs the necessary data to start the simulation. 2. The user presses one of the buttons. 3. The application displays the Jtable with the data , when View button is pressed. 4. The users observes the resulted records. 5. Simulation ends.

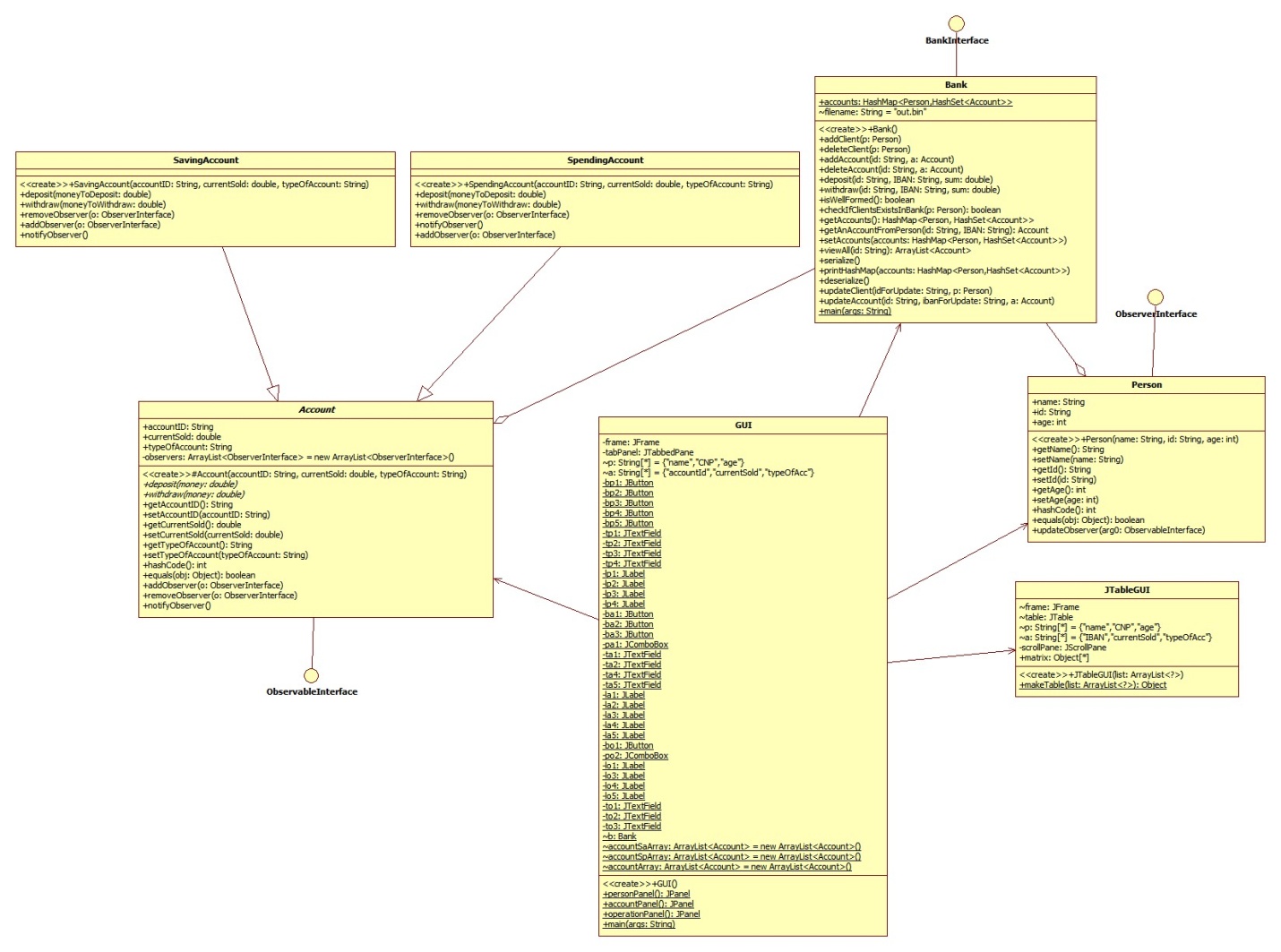
- Alternative sequences: None

- Error sequences: a) Invalid input: appears at step 2, 1.. The program returns an error; the use case fails.

**4.Design**

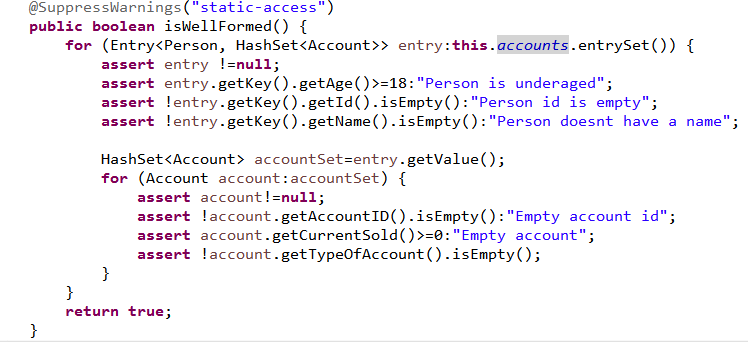
**4.1 Class design**

Class diagram is provided and each class will be analized in the following.I will attach the diagram again in order to be more visible.



**4.1.1 Bank**

This class doess all the operations this application provides , like addAccount or addClient . It uses a HashMap in order to keep the pair client-accounts, and a HashSet inside the HashMap that keeps all the accounts a client can have.It also has methods for serializing and deserializing an object, getting a certain account from a client . The method isWellFormed plays a cruicial role in the class, as it verifies if the structure of the HashMap is respected, by iterating over the map and asserting different condition for both the client and the accounts he/she has.Bank also implements the BankInterface interface that holds all the methods for adding , deleting , updating a client/account, and depozit/withdraw money.This class also implements the Serializable marker interface from Java, as the changes over the clients and their accounts must be serialized and saved to a binary file.



**4.1.2 BankInterface**

This interface contains the methods for adding, deleting a client/account and deposit/withdraw money from bank .It also show the pre and post conditions for these methods which will be tested in Bank class using assert .

**4.1.3 Person**

This class holds the attributs for a client and implements both Serializable and ObserverInterface, which is designed by me and that will be discussed in the following .It also ovverides methods like hashCode() and equals, which are used by the HashMap in order to hash each client after their corresponding id.It has method update from the ObserverInterface which displays a message to console.

**4.1.4 Account**

Is an abstract class which also implements Serializable and ObservableInterface. It has the attributes of an account, like accountID, currentSold and typeOfAccount. It has the abstract methods depozit and withdraw , which will be implemented by SavingAccount and SavingAccount. Methods from the ObservableInterface are also ovveriden.(addObserver,removeObserver and notifyObserver).It contains an ArrayList of observers., which are added/removed each time an account from a certain user is being added/deleted.

**4.1.5 SavingAccount**

This class extends Account and implemets deposit and withdraw method .When money are withdrawed from a saving account , an interest rate of 0.50 is applied. Each time money are deposited or withdrawed from this type of account, the person is notified using notifyObserver().

**4.1.6 SpendingAccount**

This class extends Account and implemets deposit and withdraw method.Each time money are deposited or withdrawed from this type of account, the person is notified using method inherited from ObservableInterface, notifyObserver().

**4.1.7 BuildJtable**

Creates an object of type JTable and displays it on a frame.

**4.1.8 ObserverInterface**

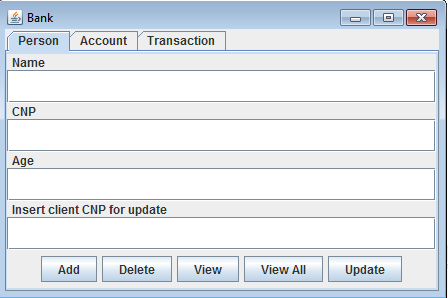
Has an abstract method update() which is ovveriden in Person class.

**4.1.9 ObservableInterface**

Has three abstract methods addObserver(),removeObserver() and notifyObserver() ,which are overriden in abstract Account class.

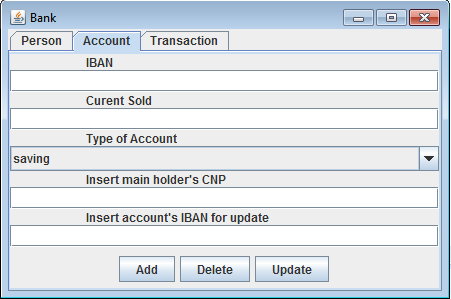
**4.1.10 GUI**

The application's GUI consists from 1 frame , which will provide the user the operation he/she can perform over the data of the bank.The first window which appears describes the operations the user can perform over the clients of the bank.

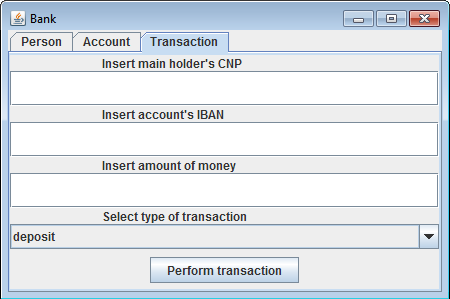


The windows for the client and product operations are very similar, the only difference being the functionality and results obtained from it. Account tab has only 3 buttons which allow the user

to add , delete or update an account by choice .It also has a JComboBx, which allow selection of the desired type of account .



The Transaction tab offers the choice of performing a withdraw or depozit operation over a certain account.For this type of operations , the clients's CNP and account IBAN is requested.



5.Implementation

**5.1. The GUI**

The GUI consists of three windows, each of them providing means for user interaction. JButtons are used in the first window to open the other windows at user choice. In the Client ,Account an Transaction windows input is gathered from JTextFields, and there is no formatting on the JTextFields so input checking will be done at the press of a JButton. In these windows JButtons represent the action desired to perform on the input data, on success data will be cleared from the JTextFields and on fail warning messages will be displayed. In the Client window, two JScrollPanes containing two JTable are used, to display the clients and their accounts of the bank.

**5.2. Class implementations**

**5.2.1** Bank

Bank has as instance variables a HashMap<Person,HashSet<Account>> account variable . This class provides getter and setter methods for all of their fields,a constructor, as well as methods for serializing and deserializing an object of type HashMap<Person,HashSet<Account>>.

**5.2.2** Person

This class represents an entry from a JTable , so the data fields of the class coincide with the columns of the data table. This way the Client class has the fields: id, name, and age . This class provides getter and setter methods for all of their fields,a constructor, as well as overriden methods hashCode() and equals() and also update().

**5.2.3** Account

This class represents an entry from a JTable , so the data fields of the class coincide with the columns of the data table. This way the Account class has the fields: accountId, typeOfAccount and currentSold . This class provides getter and setter methods for all of their fields,a constructor, as well as overriden methods hashCode() and equals().

**5.2.4** SavingAccount

This class inherits all the instance variables from superclass Account and ovverides methods withdraw and depozit.

**5.2.5** SpendingAccount

This class inherits all the instance variables from superclass Account and ovverides methods withdraw and depozit.

**5.2.6** BuildJTable

This class has as instance variables a frame, a JTable object, a scrollPanel object , two strings which contains the names of both Person and Account class and a 2D matrix of type Object. It has one method which builds the matrix that holds the values of the Person or Account type .The constructor builds the actual JTable and places it on a scrollPanel which is lately added to a frame. The JTable has click listeners on its cells which display a message each time a cell is cliked twice.

6. Conclusions

### **6.1 What I learned**

The most important thing which has been learned is the “Design by contract” technique. I think it is very important to not take things for granted, even when things are made by yourself. Sometimes there can occur some events, transparent to you, which can alter the functionality of a well-designed program but always testing and always checking if the program works in normal parameters and if the objects in the project are being modified in the right manner can minimize the risk of losing important data.

Another important thing which has been learned is serialization. The fact that the objects can be stored in one particular state and then later loaded back in order to be reused is a very important thing to know.

Also, another lesson learned is working with data structures already implemented. In this application the clients are stored in a Hash Table using HashMap structure . It is the first time I have used this data structure at such a high level: hash function transparent, iteration through the table transparent and data correctly stored probably in the most efficient way. I have worked with hash tables before but only at a low programming level and working now with them already flawlessly implemented has been a blessing.

### **6.2 Further improvements**

One of the improvements can be upgrading the security, meaning that the application, when started, doesn’t give away all the data it stores but only, at first, a login screen. Here each person can log into their account with a password previously configured or with some other protection mechanism. After log in, the user should be able to see only relevant information about his accounts, the information about other accounts and other account holders should be hidden to him. Another improvement can be, after doing the previous 2, to move it on a server so that the users can connect to their accounts from any computer which has an internet connection. Also, whenever a modification has been made to one of the existing accounts, the holder should be notified with a message on his cellphone.

After all this improvements have been made, it can be just like the applications used by the banks today.

BIBLIOGRAPHY

• Java Serialization

http://www.tutorialspoint.com/java/java\_serialization.htm

• Java HashMap

http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html

• Java assert

http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html o http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html o http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword