Ian Chelaru
Technical University of Cluj-Napoca
2nd year Computer Science
Programming Techniques 2017/2018

Assignment 4 – Specification

1. Assignment's objective

The main objective of the assignment is to design and implement an application which processes data regarding accounts from a bank and their holders. The program is written in java programming language. The integrated development environment (IDE) used is Eclipse.

Secondary objectives:

- implement the model of the program after a given class diagram;
- store and load data from a file in order to maintain the information after the application is closed;
- design by contract;
- use appropriate data structures for storing the data;
- implement a graphical user interface (GUI) in order to make a friendly communication between the user and application.

2. Features

The bank stores some personal information (first name, last name and contact phone) for each person who has or wants to make an account. A person can be the main holder for one or more accounts. Each account has a number, a holder and a type.

The application is meant to be used by the bankers when a specific request is made by a client. The client does not interact with the system.

Use cases

In each use case the client initiates the use of the system. Thus, the client is the primary actor. The banker and the system are secondary actors because they react to the client's request.

Primary actor	Use cases
Client	1. Request a bank profile 2. Edit personal information 3. Delete his/her bank profile 4. Add a new account 5. Edit an account 6. Delete an account 7. View all of his/her account 8. Add money to account 9. Withdraw money from account
Banker	10. Visualize all the account

Use case:	Request a bank profile
Number:	1
Primary actor:	Client
Secondary actors:	Banker, System
Preconditions:	None
Postconditions:	The client has a profile at the bank.
Flow:	The client gives his/her personal information. The banker introduces the information in the system.
Exceptions:	The client already has a profile.

Use case:	Edit personal information
Number:	2
Primary actor:	Client
Secondary actors:	Banker, System
Preconditions:	The client has a profile at the bank.
Postconditions:	The client's profile is updated. The client's accounts are updated.
Flow:	The client gives his/her old personal information. The banker selects the client from the list of persons given by the system. The client gives his/her new personal information. The banker introduces the new information in the system. The system updates all the recorded data regarding the client.

Exceptions:	The client does not appear in the persons list.

Use case:	Delete his/her bank profile
Number:	3
Primary actor:	Client
Secondary actors:	Banker, System
Preconditions:	The client has a profile at the bank.
Postconditions:	The client's profile is deleted. The client's accounts are deleted.
Flow:	The client gives his/her personal information. The banker selects the client from the list of persons given by the system. The banker asks the system to delete all the recorded data regarding the client.
Exceptions:	The client does not appear in the persons list.

Use case:	Add a new account
Number:	4
Primary actor:	Client
Secondary actors:	Banker, System
Preconditions:	The client has a profile at the bank.
Postconditions:	A new account
Flow:	The client gives his/her personal information. The banker selects the client from the list of persons given by the system. The client specifies the type of the account he wants to create. The banker introduces the account number and specifies the type of the account. The holder of the account is generated by the system to be the current client. The system creates a new account and adds it to the client's accounts list.
Exceptions:	The client does not appear in the persons list.
Assumptions:	The banker introduces a unique account number.

Use case:	Edit an account
Number:	5
Primary actor:	Client
Secondary actors:	Banker, System
Preconditions:	The client has a profile at the bank. The client has at least one account.
Postconditions:	The account is updated.
Flow:	The client gives his/her personal information. The banker selects the client from the list of persons given by the system. The client specifies which account he wants to edit. The banker selects the specified account. The client specifies the new information concerning the account. The banker introduces the new information. The system updates the account.
Exceptions:	The client does not appear in the persons list. The account does not appear in the client's accounts list.
Assumptions:	The banker introduces a unique account number.

Use case:	Delete an account
Number:	6
Primary actor:	Client
Secondary actors:	Banker, System
Preconditions:	The client has a profile at the bank. The client has at least one account.
Postconditions:	The account is deleted.
Flow:	The client gives his/her personal information. The banker selects the client from the list of persons given by the system. The client specifies which account he wants to delete. The banker selects the specified account. The system deletes the account.
Exceptions:	The client does not appear in the persons list. The account does not appear in the client's accounts list.

Use case:	View all of his/her account
Number:	7
Primary actor:	Client
Secondary actors:	Banker, System
Preconditions:	The client has a profile at the bank.
Postconditions:	None
Flow:	The client gives his/her personal information. The banker selects the client from the list of persons given by the system. The system displays a list with all the client's accounts.
Exceptions:	The client does not appear in the persons list.

Use case:	Add money to account
Number:	8
Primary actor:	Client
Secondary actors:	Banker, System
Preconditions:	The client has a profile at the bank. The client has at least one account.
Postconditions:	The sum from the account increases.
Flow:	The client gives his/her personal information. The banker selects the client from the list of persons given by the system. The client specifies to which account he wants to add money. The banker selects the specified account. The banker introduces a value representing the deposit money. The system computes the new sum (with interest if there is any) and updates the account.
Exceptions:	The client does not appear in the persons list. The account does not appear in the client's accounts list.
Assumptions:	The value representing the deposit money is a positive number.

Use case:	Withdraw money from account
Number:	9
Primary actor:	Client

Secondary actors:	Banker, System
Preconditions:	The client has a profile at the bank. The client has at least one account.
Postconditions:	The sum from the account decreases.
Flow:	The client gives his/her personal information. The banker selects the client from the list of persons given by the system. The client specifies from which account he wants to withdraw money. The banker selects the specified account. The banker introduces a value representing the money. The system computes the new sum (with interest if there is any) and updates the account.
Exceptions:	The client does not appear in the persons list. The account does not appear in the client's accounts list. There are not enough funds in the account.
Assumptions:	The value representing the money is a positive number.

Use case:	Visualize all the account
Number:	10
Primary actor:	Banker
Secondary actors:	System
Preconditions:	None
Postconditions:	None
Flow:	The banker makes a request to the system. The system displays all the accounts from the bank.

Other assumptions

It is assumed that the user does not move, modify or delete the "bank.ser" file.

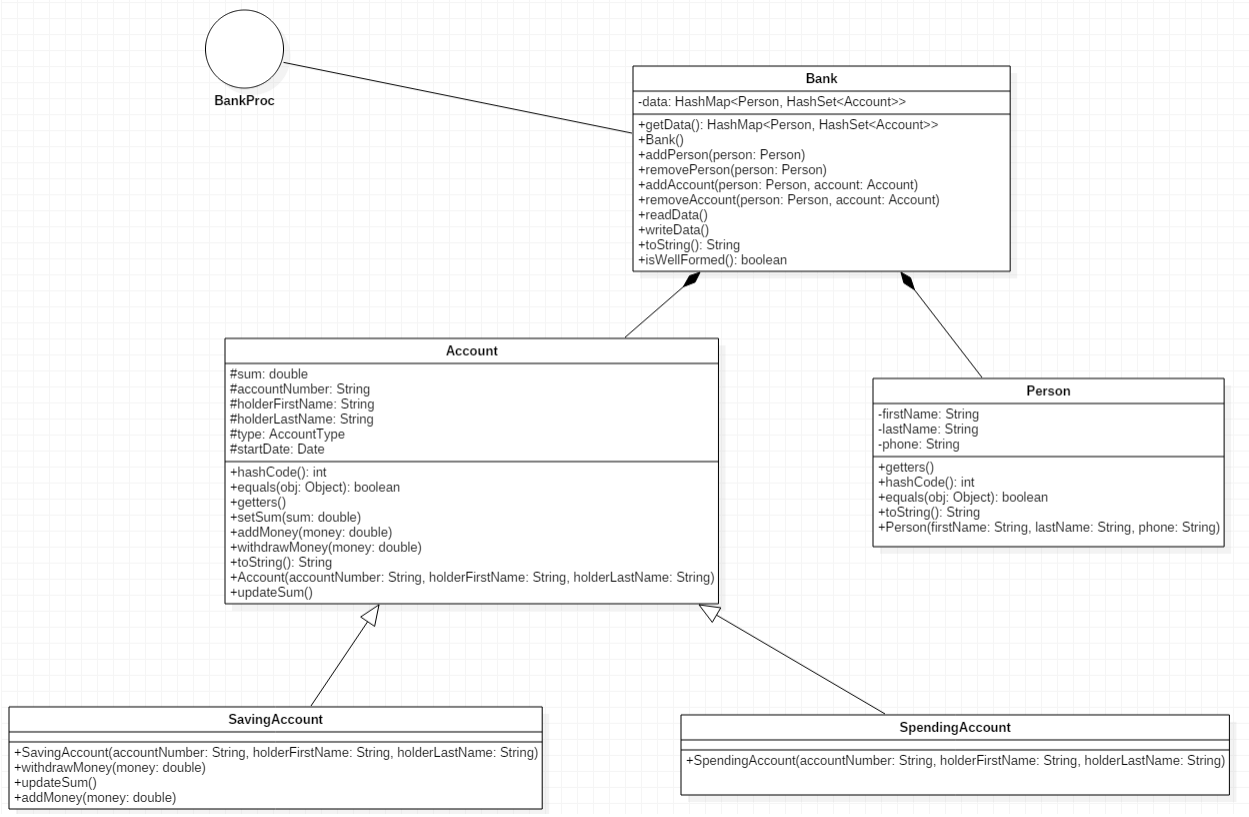
It is assumed that the user introduces a valid phone number when adding or editing a person. No format validation is done by the system.

3. Design

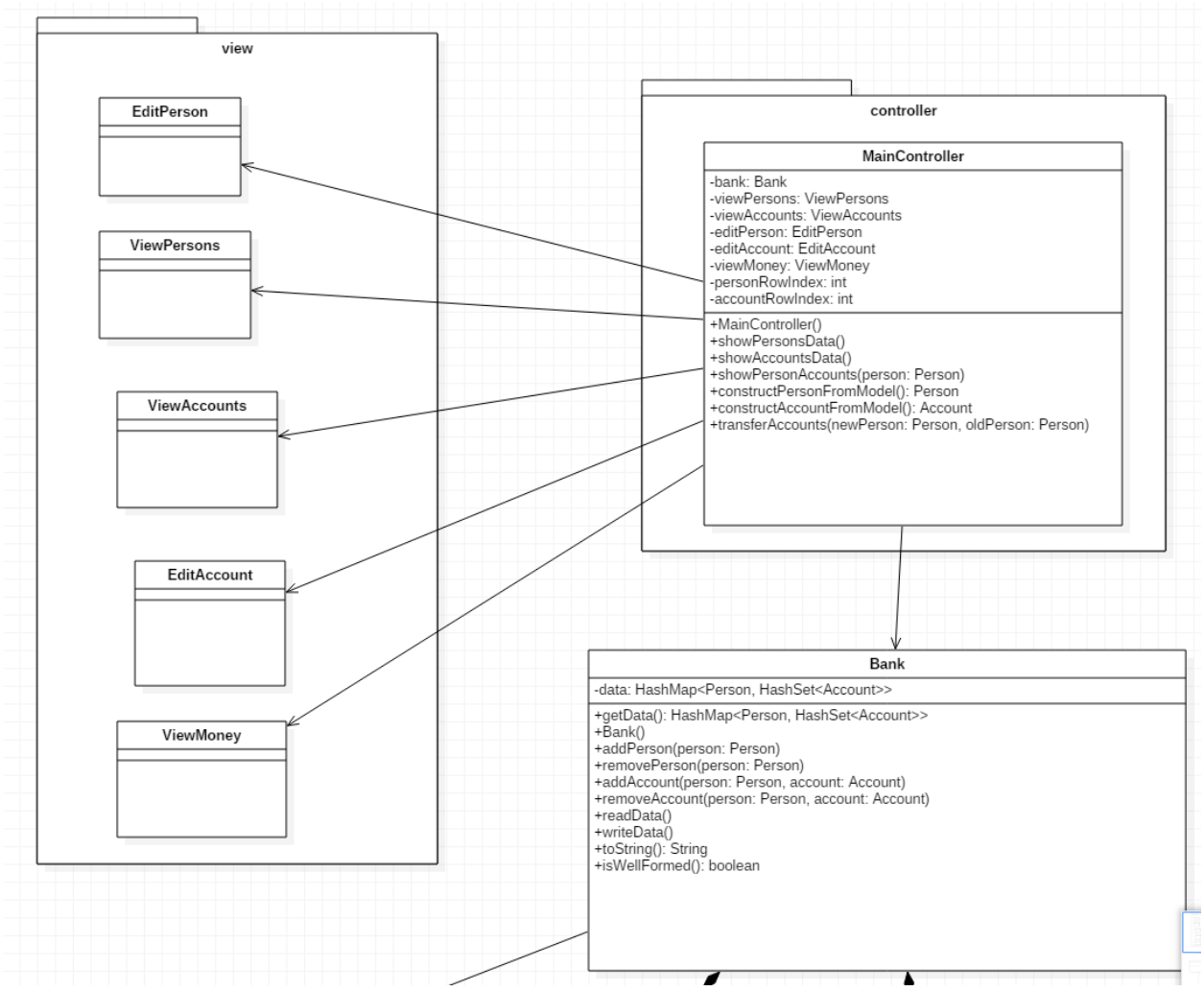
The Model – View – Controller (MVC) pattern was used to design the application. The pattern consists in splitting the problem in three parts, each one having a special purpose. The model consists in implementing classes which abstract the real problem domain. These classes have the purpose to store and manage the data and to implement the methods which solve the application requirements. The view implements the classes which create the graphical user interface (GUI). The controller is the only one which knows about the presence of the other two components. It has the purpose to call the methods from model and view when they are needed. The controller is also responsible on the data flow between the model and view. The model and the view do not communicate directly. They communicate only through the controller.

The project is structured on three packages: model, controller and view. Each one contains the classes which handle the corresponding part of the application.

UML Diagram



Model



UML Diagram

Data structures

In order to store the data in the Bank class, a HashMap data structure is used. A HashMap implements the Map interface. A map is an object that maps keys to values (keys and values can be any object). A map cannot contain duplicate keys; each key can map to at most one value. A HashMap provides constant time performance for the basic operations (get and put), assuming the hash function disperses the elements properly among the buckets. The main reason for using a HashMap is that the accounts can be mapped to their main holder. Due to the fact that a person can be the main holder for more accounts, then another data structure has to be used in order to store all the accounts of a person.

For this problem, the key is a Person object and the value is a HashSet of Account objects. A HashSet implements the Set interface. A set is a collection that contains no duplicate elements.

Thus, the system guarantees that a client or an account do not appear in the data structure more than one time. Using a map also allows easy access to the accounts of a person.

Interfaces

The model consists in an interface called BankProc which is implemented by the Bank class. The interface declares the methods who manipulate the data structure which store the bank data. In the interface implementation, the design by contract technique is used. The contract specifies what that component expects of clients and what clients can expect of it. The technique forces the programmer to specify for each method the preconditions, postconditions and to declare invariants for the data structures. A precondition must hold true for a method at the beginning of the respective method. A postcondition must hold true for a method at the end of the respective method. An invariant is a condition which holds true for a data structure whenever it is in a stable state (when it is not manipulated). An invariant is called as a precondition and postcondition of each method which manipulates the data structure.

4. Implementation

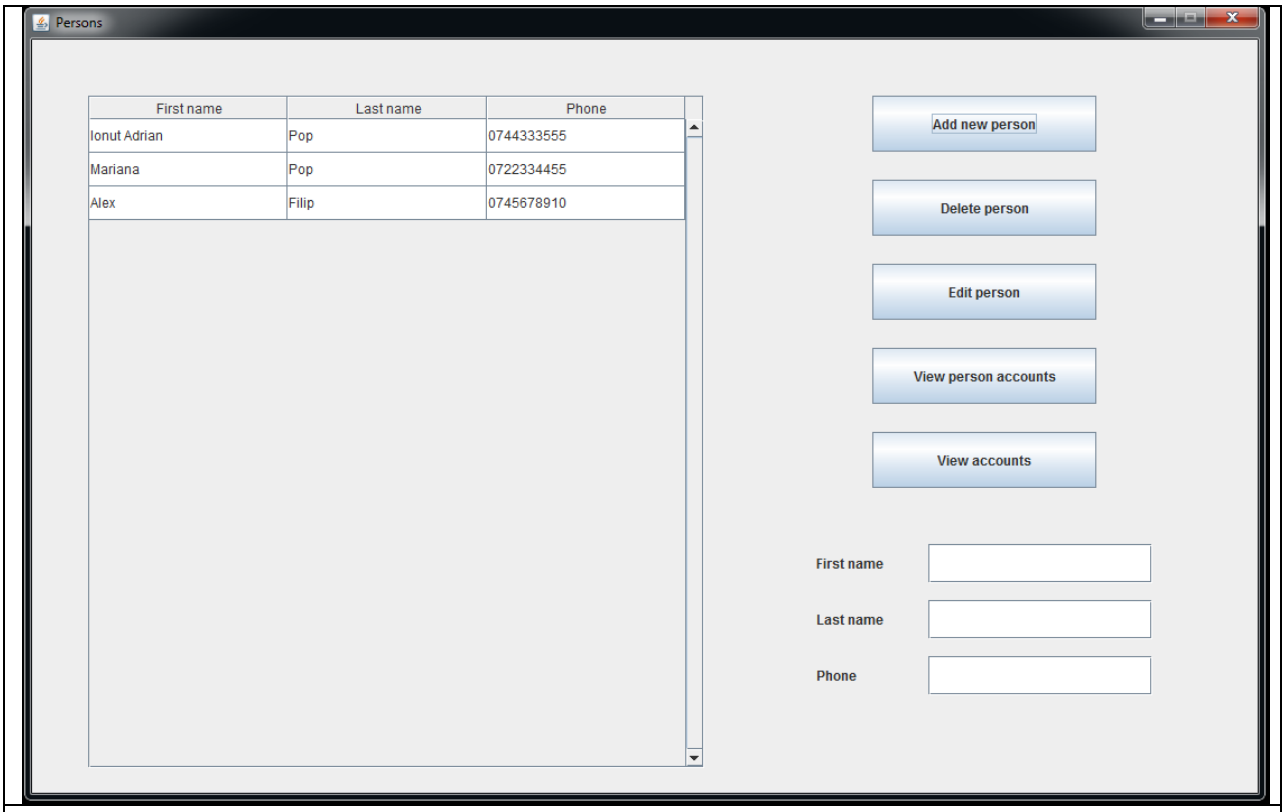
GUI implementation

The GUI consists in five classes (each one extends JFrame): ViewPersons, ViewAccounts, EditPerdon, EditAccount, ViewMoney. Every class uses swing components to implement the GUI. All the components from a class are collected in a panel (JPanel) which is added to the frame. Each class has a corresponding constructor without parameters.

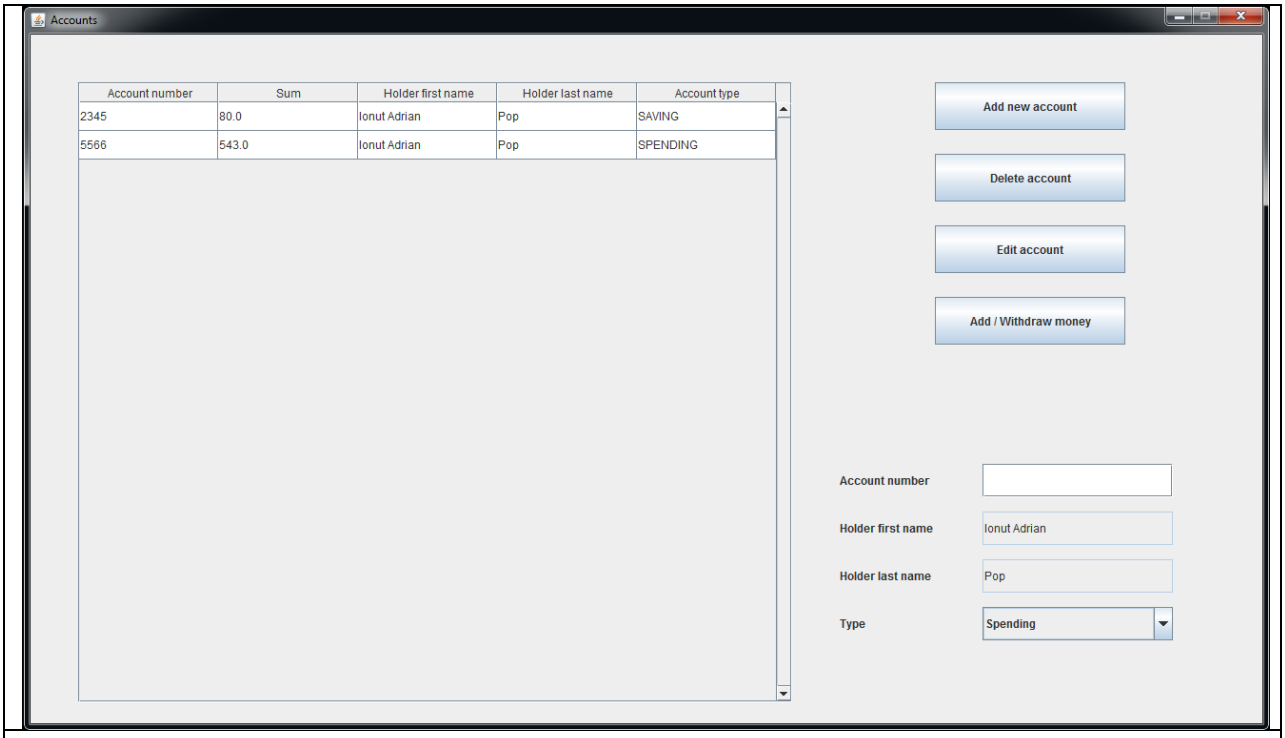
The frames provide buttons (JButton) which do the requested tasks. Each JButton implements an action listener.

The frame has text fields (JTextField) where the user has to input the data. For each text field there is given a label (JLabel) which specify the purpose of the corresponding text field. For every text fields there is a getter implemented in order for the controller to get access to the data written in it.

The ViewPersons frame appears when the application starts. Each person is displayed on a row from a table (JTable). The ViewAccounts frame implements a JTable where the accounts are displayed. Both classes implement three additional methods: one for disabling the buttons, one for enabling them and one for clearing the text fields.



ViewPersons frame



ViewAccounts frame

5. Conclusions

Future developments:

- check validity of the phone number;
- compute interest during the deposit period;
- auto-generator for account number;
- increase the number of attributes for Person and Account classes
- implement Observer Design Pattern.

What I learn from this assignment:

- how HashMap and HashSet work internally;
- why and when is necessary to override the equals() and hashCode() methods;
- what is and how to use serialization;
- how to implement click and window listeners;
- the concept of design by contract.

6. Bibliography

https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html