Technical University of Cluj-Napoca

Laboratory Work – Assignment 4
Bank Application

Name: Florea Razvan

Group ID: 30425

Table of Contents

1. Introduction

1.1 Task objective

The objective of this task is rather permissive and concerns design by contract programming techniques. We are given a class diagram to start with and we have to answer the following requirements:

1. 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 (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).
5. Implement a test driver for the system.
6. The account data for populating the Bank object will be loaded/saved from/to a file.

The application should simulate a bank application where an administrator can create accounts for the clients, delete accounts, add accounts for an existing client, remove an account from an existing client, or other applications like generating reports. A client can see the state of his or her accounts, can generate reports, can add money to an account, can withdraw money from his or her account and can have as many accounts as he or she wants.
Due to this reason, the students have full flexibility concerning the implementation and use of resources offered by the Java programming language. It is up to everyone's decision rather to develop the algorithms in a particular way or another. Same for using a Graphic User Interface or not, or modeling in a specific way or not.
Moreover an Observer Design Pattern must be implemented.

1.2 Personal approach

I personally developed a Java application for processing bank related operations depending on the state of the user. The user may be a bank administrator or a client of the bank and each of them have the ability to perform certain operations.

2. Problem Description

2.1 Problem analysis

The problem domain can be divided in smaller problems. In this way we only have to deal with problems of a smaller complexity that together form the main problem, the main objective of our application.

The problem can be seen from different perspectives and different points of view. The modeling of the orders, products and users to be recognized as structures, as independent classes, is very important because it is the core of almost every operation and every function that we want out application to perform. The user must not feel neglected and therefore the graphic user interfaces must not be forgotten and even more, they should be user friendly and efficient. The functionality is the back end "thinking process" of our application and is the most important aspect that we have to take into consideration.

2.2 Modeling

Problem modeling is done by means of object oriented programming and establishing classes, objects and relationships between the components of our problem.

2.3 Scenarios

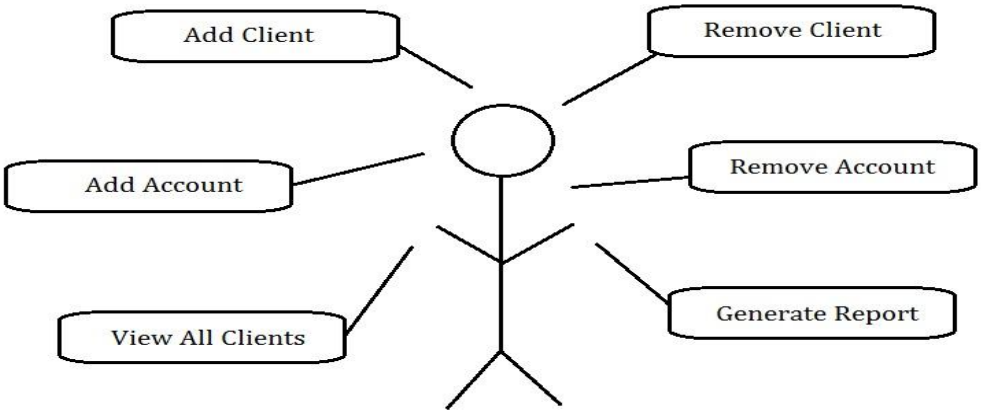
A variety of scenarios have to be taken into consideration. That is, we must be able to predict what the task is ready to be used. That is, the administrator might want to perform operations for a single client or for more clients depending on their account types. The User interface is very friendly and lets the user perform all the implemented operations.

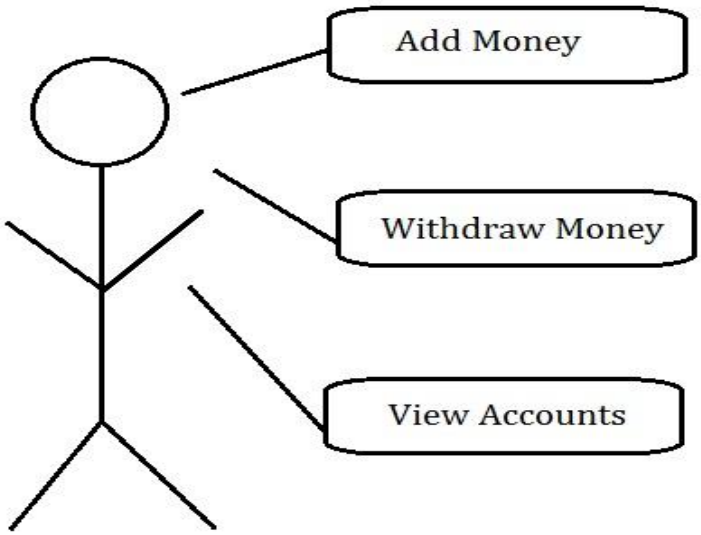
This is why I tried to take into considerations all the possible scenarios that may occur. Therefore I implemented the following functionalities: add client, remove a client, add an account, remove an account from an existing client, add money, withdraw money, generate report, see all the data in the bank at one time, or see all the data for a specific client (i.e. all of his or her accounts).

3. Projection

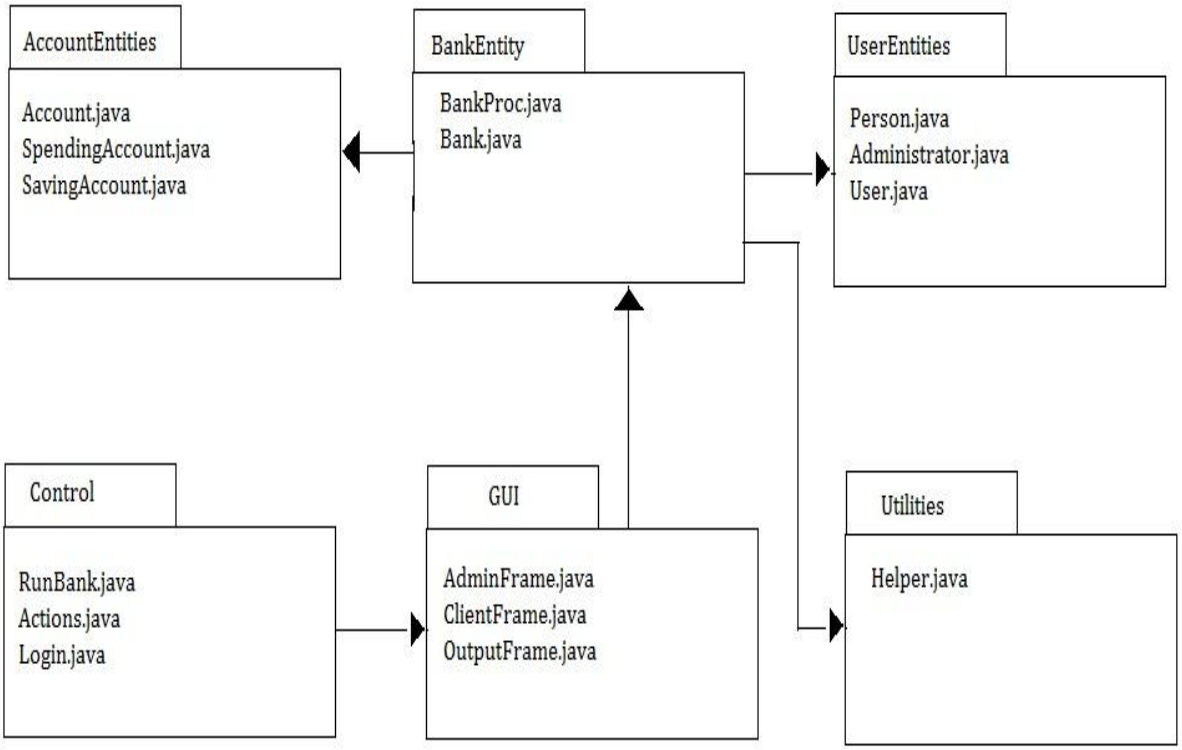
3.1 UML Diagrams

3.1.1 Use Case Diagrams

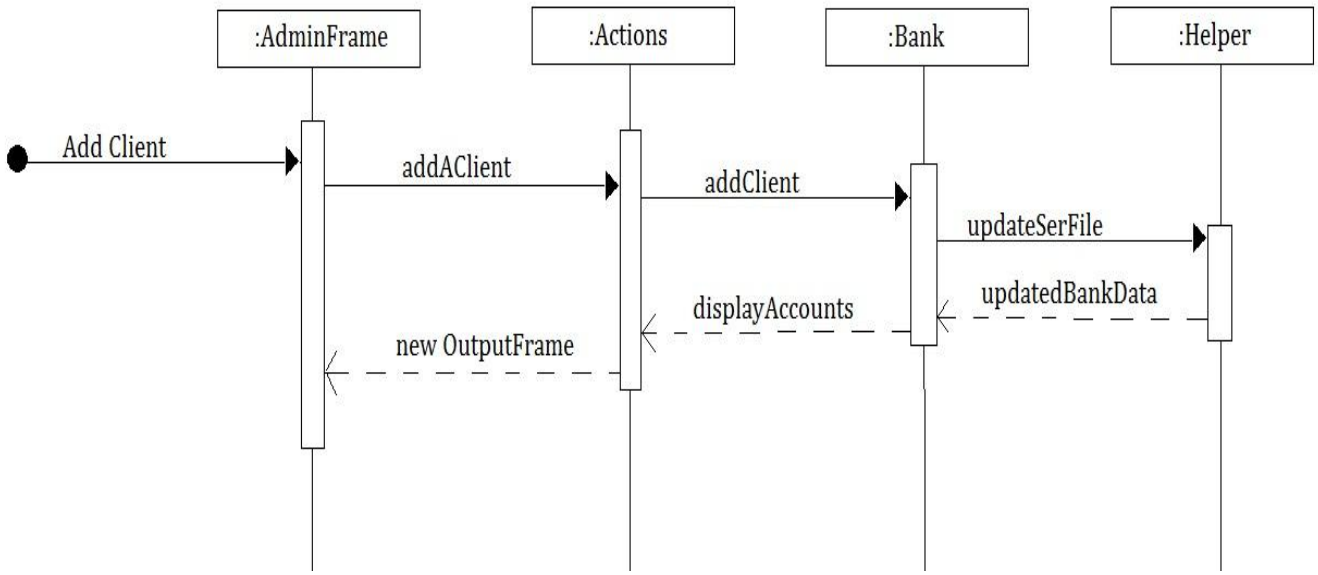




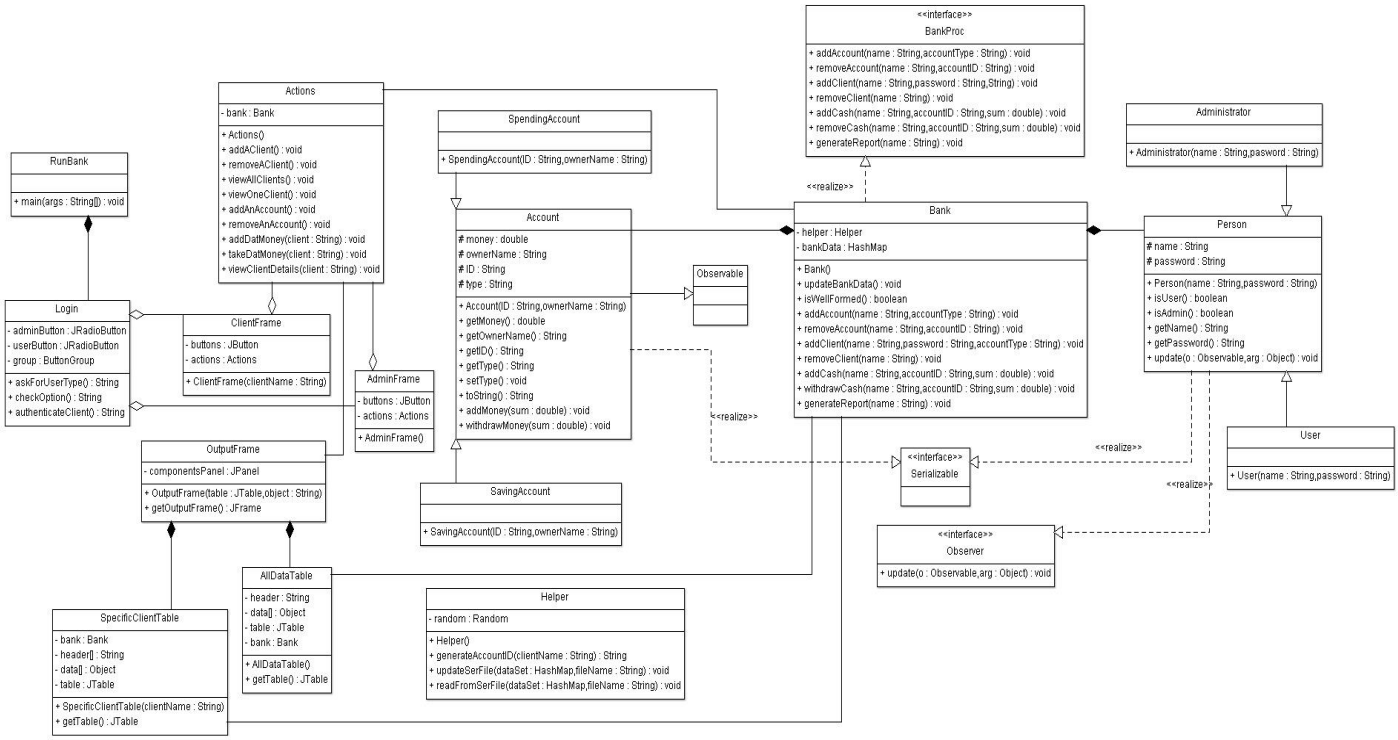
3.1.2 Package Diagram



3.1.3 Sequence Diagram



3.1.4 Class Diagram



The diagrams show how the problem is divided in smaller problems and how the modeling was done. The classes communicate between each other and therefore different types of relationships are established.

The class Bank implements the BankProc interface, and has a composition relationship relative to the classes Account and Person which are abstract classes and are extended by the classes SpendingAccount, SavingAccount and Administrator, User, respectively. The classes Account and Person also implement the Serializable marker interface. The class Person implements the Observer interface and the class account extends the class Observable. Thus we have implemented the Observer Design Pattern. We also have a Helper class which contains methods for serializing and deserializing objects when we save and update the state of the bank data. Also the client and the administrator have representative and distinct user interfaces.

For the class diagrams I have used the ArgoUML environment.

3.2 Data Structures

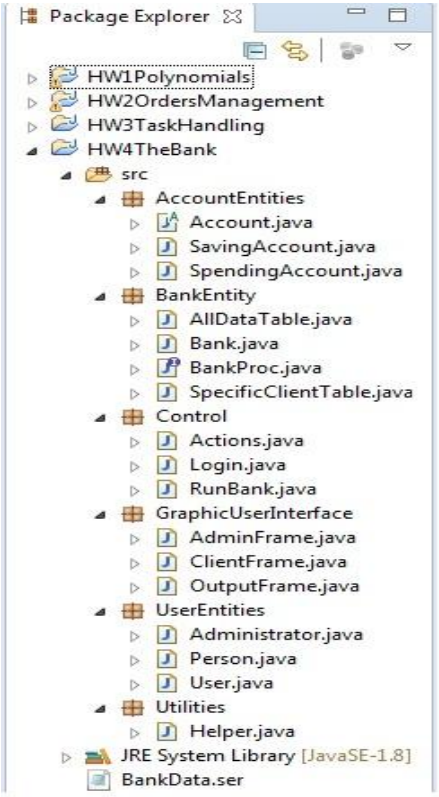
The data structures used are the primitive data types provided by the Java language such as String, integer, double, boolean, font, or other data types such as the customized button, JLabels, JTextFields, JTextAreas, fonts, JFrames.

I also used HashMap to implement the bank data structure which has as keys the unique clients and as values a list of all of the client's accounts.

3.3 Class Projection

The modeling and partitioning of the problem domain is done by means of dividing the problem and data types needed into several classes from which I instantiate objects of that type and use is it in another class. This connection between object of one class and methods of another class is done by means of relationships.

These classes are separated regarding their purpose into different packages. These packages are as follows:



1. Control Package:

The Control package contains the "thinking" done by the applications. That is the functionality of the application and the management of the user input.

It contains the following classes:

- Login.java;
- Actions.java;
- RunBank.java.

2. AccountEntities:

This package contains the classes responsible with representing the account entities. These classes are as follows:

- Account.java;

- SavingAccount.java;

- SpendingAccount.java.

3. UserEntities:

This package contains the classes responsible with representing the user entities. These classes are as follows:

- Person.java;

- User.java;

- Administrator.java.

4. BankEntity:

This package contains the classes responsible with representing the bank entities. These classes are as follows:

- Bank.java;

- BankProc.java (an interface);

- AllDataTable.java;

- SpecificClientTable.java.

5. Utilities:

This package contains the classes responsible with checking the user input and the atomic operations needed for thread synchronization. This package only contains one class:

- Helper.java.

6. GraphicUserInterface:

This package contains the classes responsible with representing the entities onto the output panel. These classes are as follows:

- AdminFrame.java;

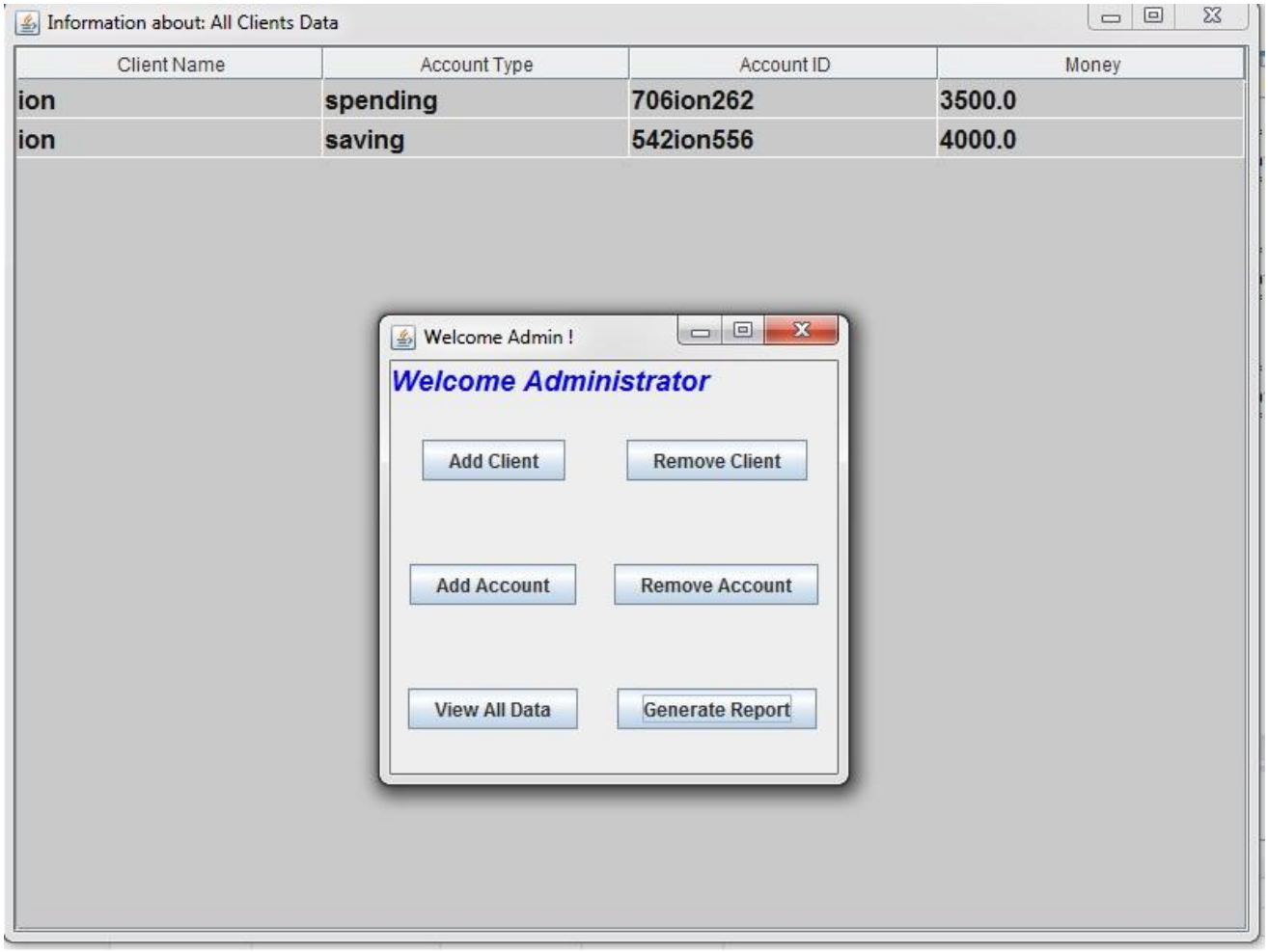
- ClientFrame.java;

- OutputFrame.java.

The packages communicate and everything work together in order to satisfy the user and all of his or her requirements. The algorithms are quite simple due to the good managements of resources, data available and modeling the problem domain.

3.4 User Interface

The user interface that I created for this application is the following one:



4. Implementing and Testing

Implementation of the algorithms that I have used and modeling the problem was done by means of the Java programming language and the Integrated Development Environment (IDE) that I used is Eclipse. Even though I have not tested the application on another IDE, it should maintain its portability.

I have covered all the possible cases for user input for the dialogs with the one using the application so that no errors should occur to stop the running time of the applications. For this I used a lot of try - catch blocks to handle different exceptions that may occur.

I have tested all the possible cases that may appear when someone is using the application, and it was proven to be safe for everyone and is easy to understand how to interact with the application and the user interface.

5. Results

The Application is user friendly and very helpful in simulating bank related operations. Being developed in the Java programming language the final product is highly portable and is able to run on different operating systems as long a java development kit is installed. The application is very straightforward and instructions are always given to the user whenever some wrong data is send as an input and is considered as not being valid.

6. Conclusions

This application meets all of its requirements but it can also be enhanced and developed even more. I have learned some new things and also I increased my java knowledge and experience with this programming language.

7. Bibliography

http://stackoverflow.com/

https://docs.oracle.com/javase/tutorial/uiswing/components/table.html

http://www.color-hex.com/color-names.html

https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html