Technical University of Cluj-Napoca

Computer Science and Automation Faculty

Computer Science Department

2nd semester 2015-2016

Programming Techniques

Homework 4

Pascaru Cosmina –Roxana

Group: 30425

CONTENT

Assignment Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

Problem Analysis. Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .4

Usage Scenarios . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ..4

Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6

* UML Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .6
* Package Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
* Sequence Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .8
* UseCase Diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
* Classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

Conclusions. Further implementations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11

1. Assignment Objectives

**Objective**

Design by Contract Programming Techniques

**Description**

Consider the system of classes in the class diagram below.

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.

1. Problem Analysis. Modelling

The application simulates a bank, and it offers 2 perspectives: a client one and an admin one.

For the client, the user interface of the application is similar to one of an ATM. For the admin, the user interface offers multiple options that include adding new clients, adding new accounts, deleting accounts and deleting clients.

At the beginning, the user chooses which role they want to play, the admin or the customer. In order to access a client account, you need to enter the pin of the account. The you can check the sold of the account, deposit money in the account or withdraw money. At each operation, a message is displayed, noticing the user that a change has occurred in their account.

The admin has the possibility to view all the clients of the bank, and to check the all the accounts of a client.

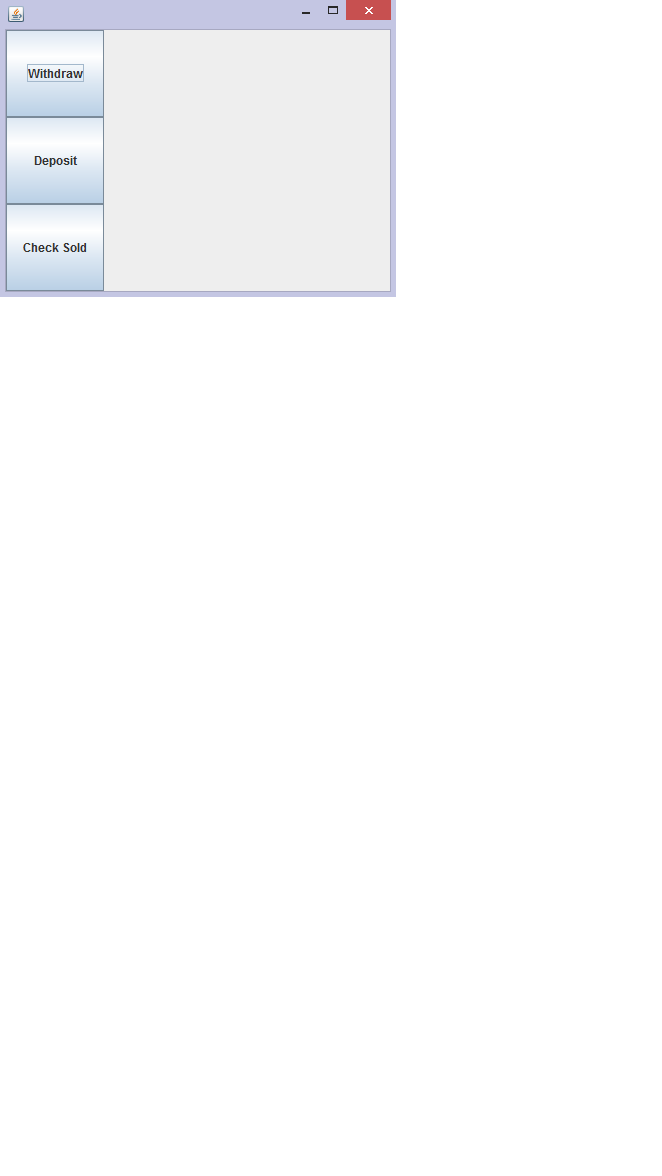
The admin can also select an account of a client to delete, or can add a new account for a client.

1. Usage scenarios

At the start of the application, a frame with two buttons is displayed. This frame gives the user the possibility to choose who they are: a client or the admin.



Client’s view



The client can perform 3 operations : check sold, deposit and withdraw.

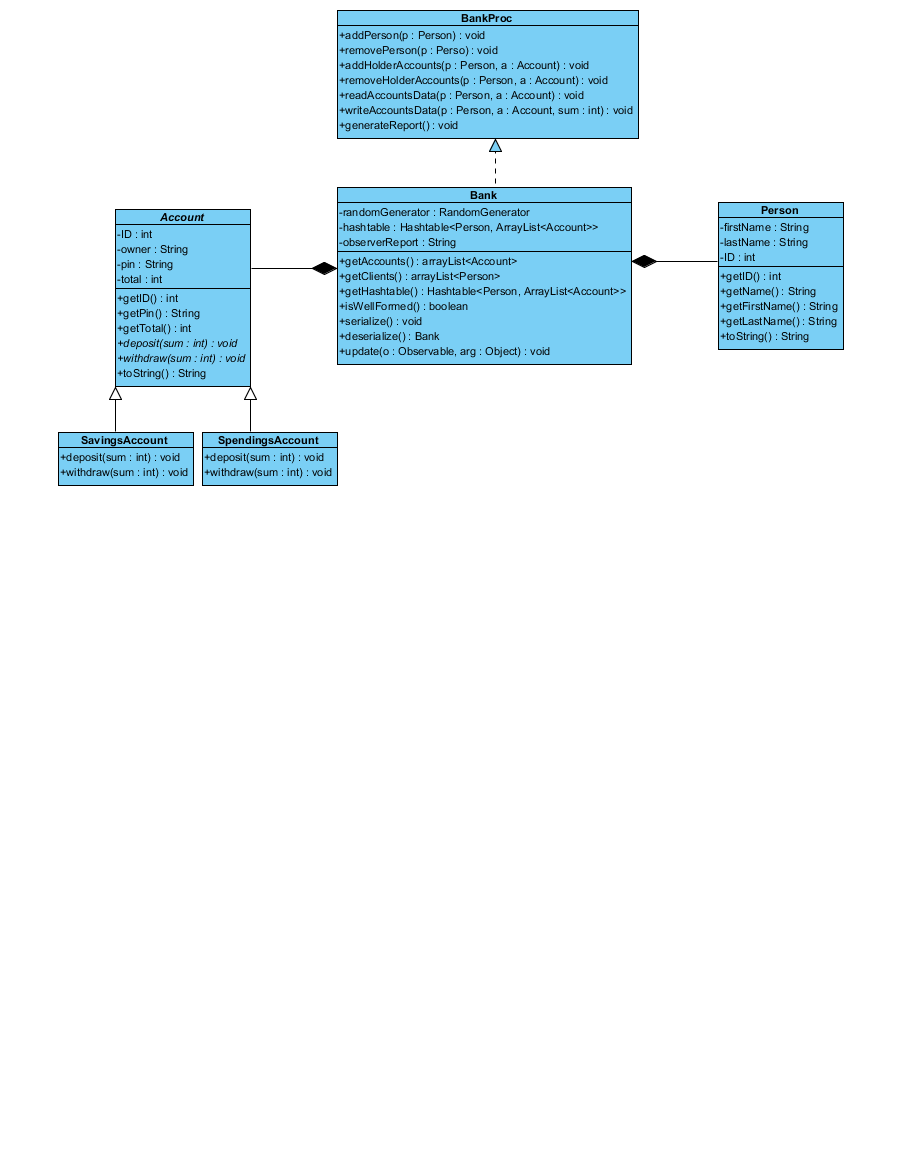
Admin’s view

The admin can view all the clients in a JTable, check all the accounts of a client by the client’s name, add or remove a client, add or remove accounts of a client.



1. Design

4.1 UML Diagram



The UML diagram shows the relationships between classes. The interface BankProc is implemented by the class Bank. It has the methods getAccounts() which returns the array list of accounts in the class Bank, getClients() which returns the array list of clients. The method getHashtable() returns th hashtable whichholds an array list of accounts for each client, as a client can have more accounts open. The class Bank also implements the Serializable interface and the Observer interface.

The class Account is an abstract class which is extended by the classes SavingsAccount and SpendingsAccount. The difference between a savings account and a spending account is that the sold in a savings account cannot be lower than 30000, while a spending account can have the sold zero.

The class Person holds information about a client, such as their first name, last name and an ID

4.1 Package Diagram

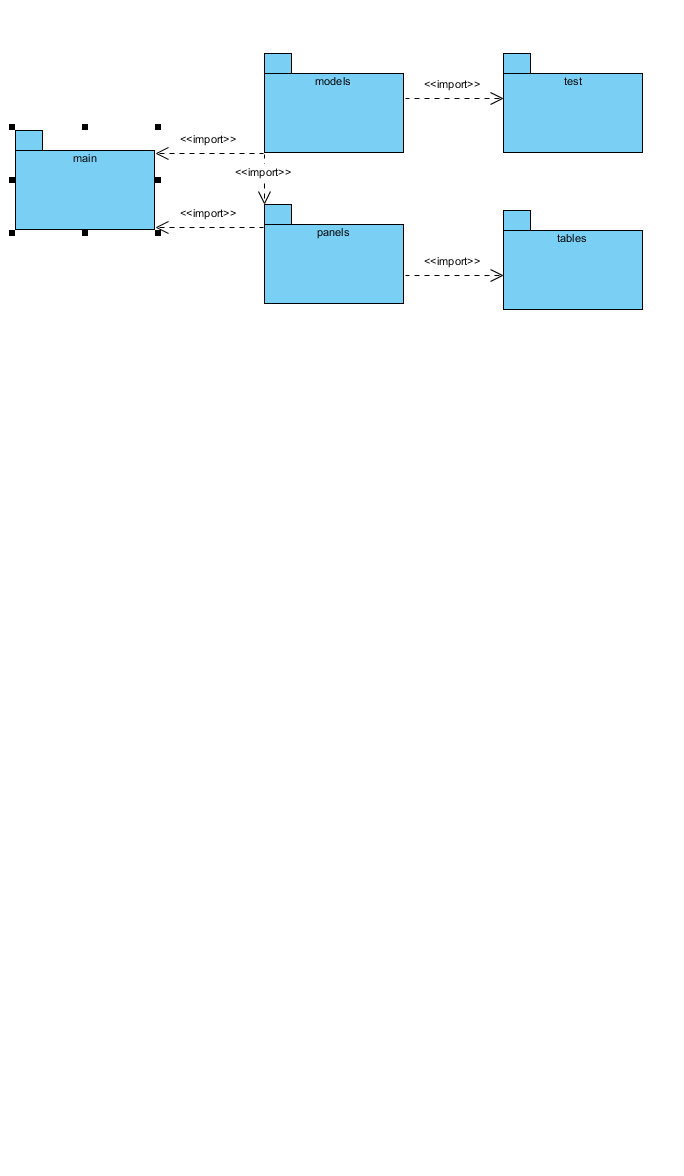
The project is divided into 5 packages. The package main contains the MainClass which starts the application and the LoginFrame class which creates the main frame of the user interface.

The package models contains the classes and the interface presented in the UML class diagram: Account, Bank, BankProc,Person,SavingsAccount and SpendingsAccount.

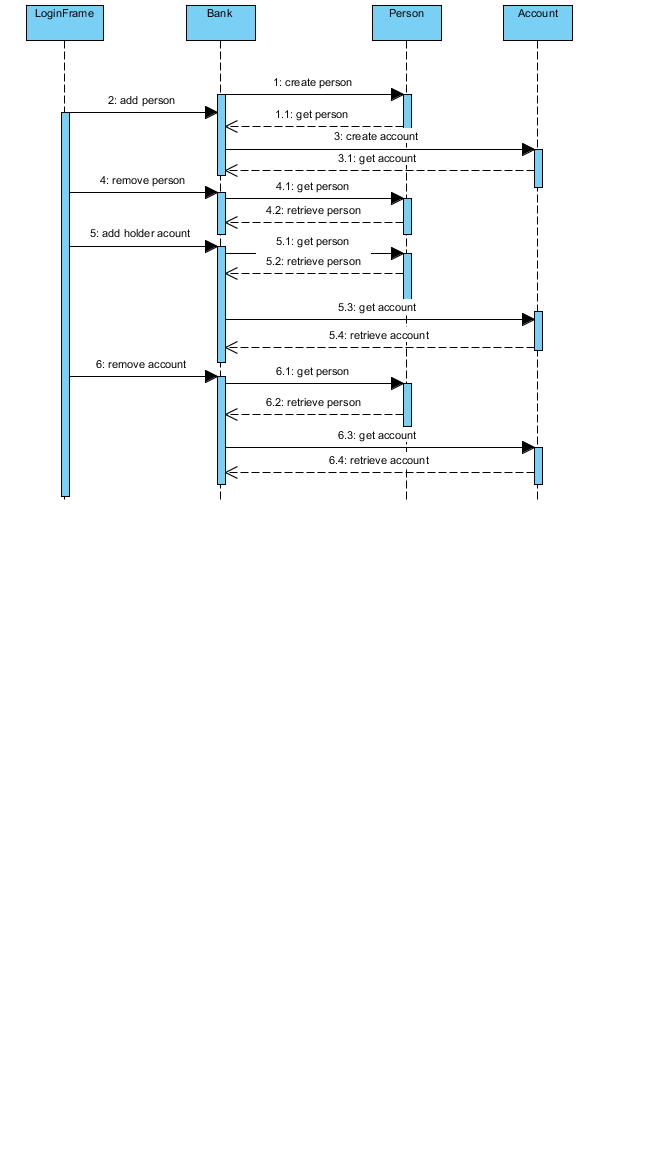
The package panels contains the panel which are later added to the main frame of the user interface: AdminPanel and CustomerPanel. These represent the two different views: client view and admin view.

The package tables contains the tables and tables models used to display information about the bank’s clients and the client’s accounts: ClientsTable, ClientsTableModel, InfoTable, InfoTableModel.

The package test contains the class TestBank1 which contains test methods for the methods of the class Bank.



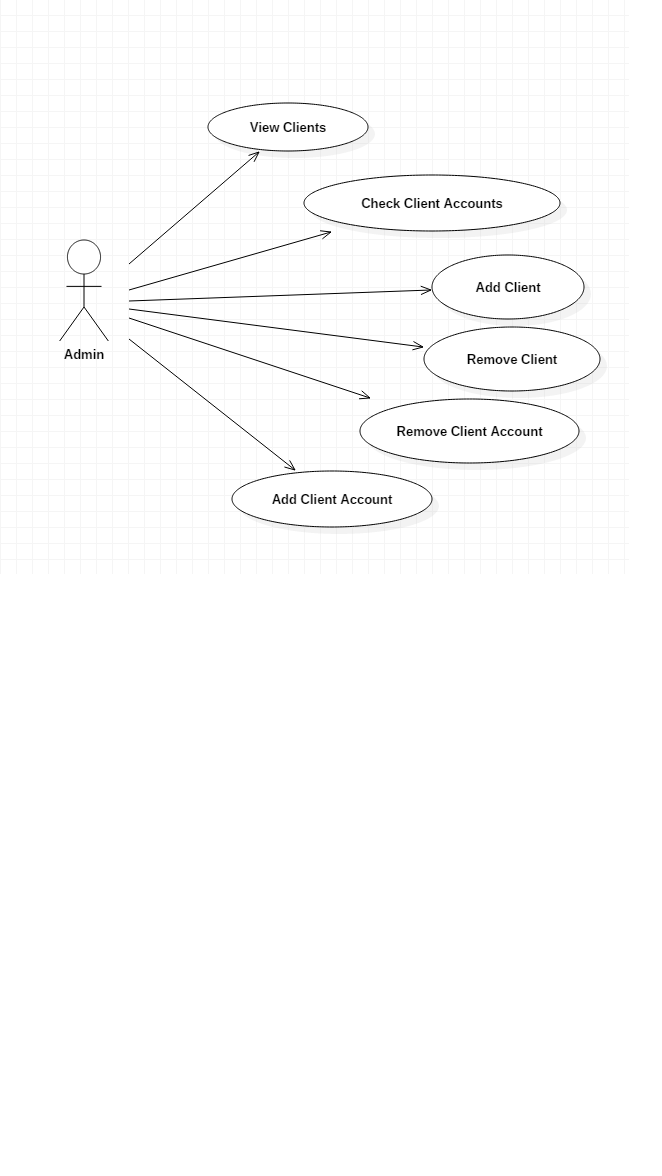
4.3 Sequence Diagram

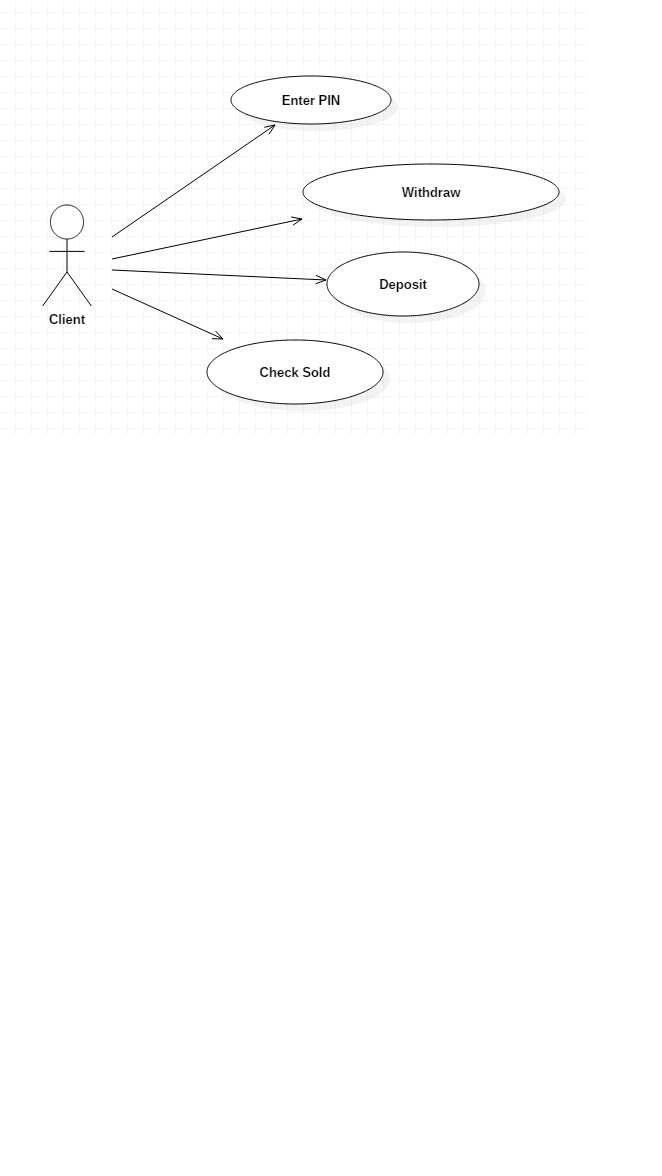


The sequence diagram shows the evolution of the classes of the project. It shows the commands which trigger the activation of the classes.

The add person command will create a new instane of the Bank class which will later create new instances of the classes Person and Account.

The command remove person only accesses an instance of the class Bank and Person.

4.4 UseCase Diagrams



4.5 Classes

Account

* is an abstract class
* is extended by the classes SpendingsAccount and SavingsAccount
* has as attributes: ID, ownerFirstName, ownerLastName, pin and total
* has as methods: getID(), getTotal(), changeTotal(int total), getOwnerFirstName(), getOwnerLastName(), the abstract methods deposit() and withdraw() and the method cannotWithdraw()

SavingsAccount

* extends Account
* cannot have a total less than 30000

SpendingsAccount

* extends Account
* cannot have a total less than 0

Interface BankProc

* methods: addPerson(), removePerson(), addHolderAccounts(), removeHolderAccounts(), readAccountData(), writeAccountData(), generateReport()

Bank

* implements the BankProc interface, Serializable interface and Observable interface
* has as attributes: an array list of persons, an array list of accounts, a hashtable of accounts of persons and a string observerReport which is updated when an operation is done on an account

Person

* implements Serializable interface
* has as attributes: firstName, lastName and ID
* has as methods: getID(), getName(), getFirstName(), getLastName() and toString()

1. Results

The application provides commands specific to the client and the admin. It also saves new entered data by serializing it.

Every transaction is monitored and the information about it is displayed in the console. This way the clients can see the events that occurred and that changed the sold of their accounts.

1. Conclusions. Further implementations

Although it seemed difficult to work with pre, post conditions and with the observable interface, in the end it all came together. Some further implementations for the application would be to generate a report of an account in which all transactions are visible, to add new specifications to the accounts, such as annual interest, an expiration date for the account and so on.

The graphical user interface could also be improved. Some more constraints can be added so that the admin account has a password, or the deleting of a client requires more data (a number of days until the account is no longer usable).

1. Bibliography
2. <https://docs.oracle.com/javase/7/docs/api/java/util/Observable.html>
3. <http://www.javaworld.com/article/2077258/learn-java/observer-and-observable.html>
4. <http://www.javaworld.com/article/2077258/learn-java/observer-and-observable.html>
5. <http://stackoverflow.com/questions/13744450/when-should-we-use-observer-and-observable>
6. <http://www.tutorialspoint.com/java/java_serialization.htm>
7. <http://stackoverflow.com/questions/447898/what-is-object-serialization>
8. <http://stackoverflow.com/questions/447898/what-is-object-serialization>
9. <https://docs.oracle.com/javase/7/docs/technotes/guides/language/assert.html>
10. <http://stackoverflow.com/questions/2758224/what-does-the-java-assert-keyword-do-and-when-should-it-be-used>