Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

2nd Semester 2016-2017

Bank Management System

Programming Techniques

Homework 4

Student Ieremiasș Viorel

Group 30422

Contents

[1. Problem specification 2](#_Toc483472907)

[1.1. Task description 2](#_Toc483472908)

[1.2. Application description 4](#_Toc483472909)

[2. Problem analysis 4](#_Toc483472910)

[2.1. General overview 4](#_Toc483472911)

[2.2. Input & output 4](#_Toc483472912)

[2.3. Use cases 6](#_Toc483472913)

[2.4. Data Structures 8](#_Toc483472914)

[3. Design 10](#_Toc483472915)

[3.1. Class design 10](#_Toc483472916)

[3.2. Used packages 13](#_Toc483472917)

[3.3. Observer pattern 14](#_Toc483472918)

[4. Results 15](#_Toc483472919)

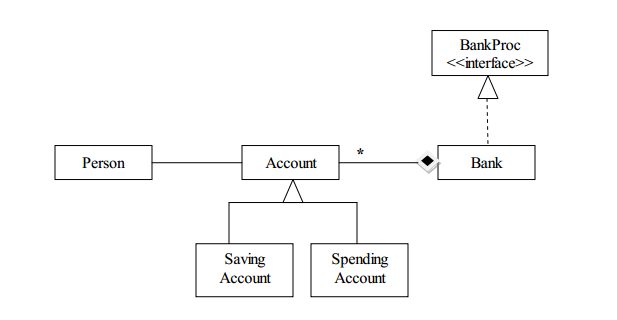
[5. Conclusion 15](#_Toc483472920)

[7. Bibliography 15](#_Toc483472921)

# Problem specification

## Task 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. Design 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. Design and implement a test driver for the system.

6. The account data for populating the Bank object will be loaded/saved from/to a file.

Design and implement an application aimed to simulate bank operations. Considering an application called Bank Management Simulator, design and implement a simulator for bank order processing. The application should contain (at least) following classes: Person, Account, Savings Account and Spending Account. The interface Bank Proc should contain methods for adding/removing accounts, read/write accounts and report generators. The interface methods should contain pre and post conditions. The class Bank should be implemented using Design by contract technique.

## Application description

The task has been implemented in the form of a system designed for bank employed, who need to access accounts belonging to customers in order to open/close accounts, withdraw and deposit money from/into accounts (saving or spending). Also, there is an option to register new persons into the system or delete already existing ones (together with the accounts associated with them).

The given task has been interpreted in the form of a bank management simulator. The Bank Management Simulator provides users the opportunity to manage a commercial bank making decisions with respect to adding new clients, accounts (divided into two categories – saving and spending), the possibility to deposit and withdraw money, generating reports about the bank’s activity. All the information about the bank’s state is saved in order to allow users to preserve their information between more sessions. This simulation program should allow users to get familiar with the banking world of today and the general bank management, although the emphasis isn’t put on commissions for certain operations or other bank policies.

# Problem analysis

## General overview

The application should implement all the requirements given in the task description. The implementation of the bank will consist in a Hash Table that uses Chaining technique in case of collisions. Specific methods for adding/removing and listing information from the Hash Table shall be implemented. In order to allow a better understanding of how this application functions, assertions, contracts and invariants shall be used as well as Java documentation for the project.

## Input & output

All the information that will be registered into the application from the graphical user interface, will be tested in order to comply with validating criteria, and specific dialog messages will appear in case that the user tries to input some wrong data.

Information about clients shall be restricted to their first and last Name, a phone number and the National Identification Number (NIN), which the Bank will use to distinguish between different clients.

A client can have one or more accounts. On all of these accounts, operations like depositing or withdrawing money can be performed. If the withdrawn amount exceeds the available amount in the account, an error message shall be generated. In the case of the saving account, only 1 deposit and 1 withdraw is allowed during the active period of the account.

The user has the possibility to delete client accounts or even delete totally clients from the bank. There is no limit on the amount of money that a user can deposit or withdraw at a given moment of time.

The user has the possibility to output information into the graphical user interface, such that the bank’s state can be checked at any time. Information about the number of clients, the number of accounts, detailed information about the bank’s state and listing of account should be generated and available to the bank manager and employees.

Because the application is meant to serve only educational purposes, it doesn’t require a login step, assuming the person who uses the application is always an authorized bank employee.

In the main window, two buttons are displayed: one that gives access to views and functions destined to the manager and authorized personnel and one that gives access to the application that is destined for bank clients.

The admin window has a table that contains information about the clients. When adding a new person into the system, the administrator will enter the person id, first name, last name and the phone number and then press the “Insert” button. To remove a person, it is enough to select the person in the table and press the “Delete” button. If it is needed to change some information about a person, this can be easily achieved by modifying the person details directly in the table and pressing “Update”. Multiple fields of a client can be modified at one time, except the identification number, which is immutable, but it is important to only update details about a single person and press “Update” before going further to another client.

Clicking a person and then pressing “Account” will open up another window that presents in a table details about all the accounts of a client. The new window also includes buttons for operations like “Open account”, “Close account” and “Update account”, whose meaning and function is obvious.

The user window contains fields that allow selecting the user and the account to perform operations on. The available operations on accounts are deposit and withdraw, although an option to transfer money between two different accounts of a client would be another useful function.

A deposit is made by selecting the person from a list and then choosing in which of the accounts belonging to the person you want to make the deposit. After this, an amount has to be entered, which has to be a valid integer. The process for withdrawing money from an account is similar.

## Use cases

Add person

Description: Add a new person into the bank system, by filling the required fields (id, first name, last name, address), together with the first account of the person, being a new client.

Remove person

Description: The bank employee selects in the table the person that has to be removed from the system. If no client is selected when pressing the button, an error message will be displayed.

Process deposit

Description: The user will see a list of all registered bank clients and he/she will have to select one of the persons. After a person is selected, the "Accounts" list will be populated with all the accounts belonging to the selected person. Next, an account has to be selected and the amount to be withdrawn has to be entered. Error messages will be shown if no person/account was selected or if the amount is not a valid integer number.

Process withdrawal

Description: The process withdrawal is similar to the deposit process. The user selects a person, an account, and enters the amount to be withdrawn.

Add account

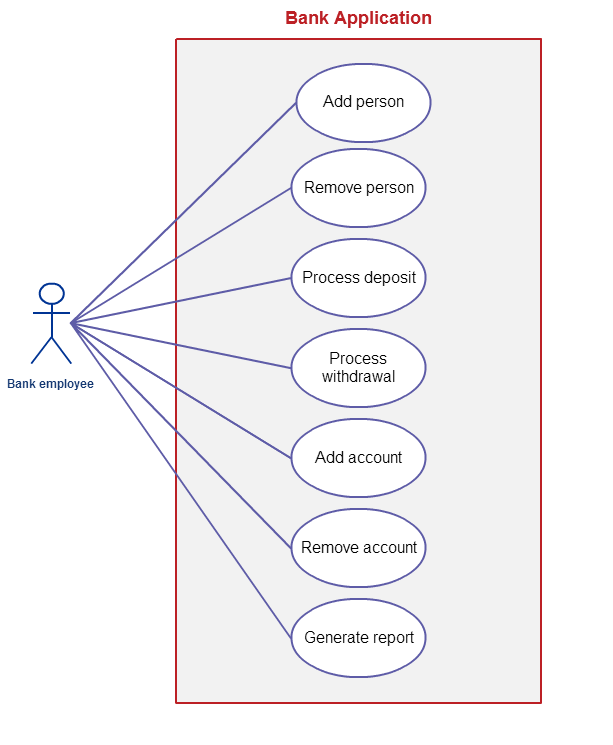
Description: The bank employee selects a person, which is already registered into the system, and then opts for one of the two account types: spending account and saving account.

Remove account

Description: A list with all registered persons will be displayed, and after choosing a person, the person's accounts will be shown in a list. The user will select the account to be closed and proceed to close it.

Generate report

Description: A report containing information about the persons that initiated an operation, the type of the operation, the amount that was deposited or withdrawn and the new balance are sent to the user every time a transaction is performed on his accounts.



## 2.4. Data Structures

The main problem to be taken into account when designing a class is choosing which data structures are needed.

a. Array

This Java class (member of the Java Collection Framework) has many methods for manipulating arrays (such as sorting and searching).

b. ArrayList

The Java class (member of the Java Collection Framework) has resizable implementation of the List interface, implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to Vector, except that it is unsynchronized.)

Advantages: ArrayList can be used to store the list of accounts for each client. Each ArrayList instance has a capacity. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically.

Methods like add or remove (remove an element from a specific position in the array list), size (returns the number of elements), get (returns the element at the specific index from the array) and constructors for the array list from class ArrayList can be used.

Disadvantages: Array List is not synchronized, but it can be synchronized using the Java synchronize.

c. Hash Table

The Java Class Hash Table implements a hashtable, which maps keys to values.In this class, any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method. An instance of Hashtable has two parameters that affect its performance: initial capacity and load factor. The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. The hash table is open: in the case of a "hash collision", a single bucket stores multiple entries, which must be searched sequentially. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased.

When the number of entries in the hashtable exceeds the product of the load factor and the current capacity, the capacity is increased by calling therehash method.

d. Hash Map

Hash Map is actually Hash table based implementation of the Map interface. This implementation provides all of the optional map operations, and permits null values and the null key. (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

Advantage: This implementation provides constant-time performance for the basic operations (get and put), assuming the hash function disperses the elements properly among the buckets. Iteration over collection views requires time proportional to the "capacity" of the HashMap instance (the number of buckets) plus its size (the number of key-value mappings). It's very important not to set the initial capacity too high (or the load factor too low) if iteration performance is important.

Disadvantage: Note that this implementation is not synchronized. If multiple threads access a hash map concurrently, and at least one of the threads modifies the map structurally, it must be synchronized externally. (A structural modification is any operation that adds or deletes one or more mappings; merely changing the value associated with a key that an instance already contains is not a structural modification.)

Chosen data structure: Hash Map

Since this application will not use multithreading, there is no problem using Hash Map. Moreover, we can come across the situation of permitting null entries. Another advantage of using Hash Map is that this collection is collision safe, because the entries of the table act like a linked list. When you put a new entry into the same bucket, it just adds to the linked list. If the hash of the key in the map collides with an existing key, the Map will re-arrange or keep the keys in a list under that hash. No keys will get overwritten by other keys that happen so be sorted in the same bucket.

# Design

## Class design

After succeeding in analyzing the problem, we can go further to developing the classes that are necessary for the implementation of this application.

Interface BankProc defines the operations that can be performed by the bank on manipulating clients and their accounts.

Class Bank implements interface BankProc and thus gives an implementation to all the bodyless methods in the interface. It also defines an object which has the attributes of a bank. This class will contain the implementation of the Hash Table, which shall contain all the information about the bank clients and accounts. The Hash Table maps clients to an array of accounts, as one can have more than one account.

Class Account describes in a generic manner what an account represents. It contains information about the amount deposited in the account, the interest rate of the account, an identification number of the account and a brief history of the operations performed so far and the client which owns the account and the type of the account.

Class SavingAccount extends Account and describes more specific an account for saving money. It, the number of allowed withdrawals and deposits. Its implementations for deposit and withdraw limit the amount of operations that can be performed during the active period of the account and the minimum amount for a transaction.

Class SpendingAccount extends Account and describes an account designed for spending money on a daily basis. It has an interest rate equal to zero by default, but it does not limit in any way the number of transactions that can be performed.

Class Person describes a person which can be a client of the bank. It contains information about the client, like first and last name, phone number, and id – which is unique in the application and represents the person's CNP/Social Security Number and thus is the field by which clients can be easily identified.

The classes used for the graphical user interface are:

- StartView, which is the entry point to the application;

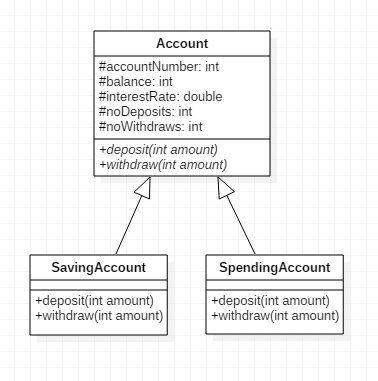
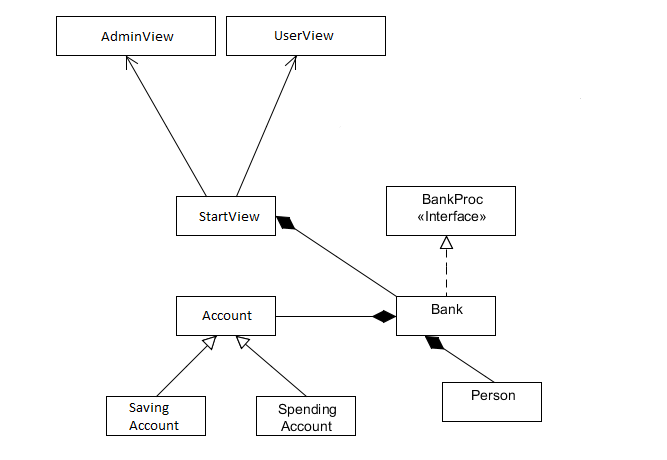
- AdminView, which has the functionality the administrator has access to;

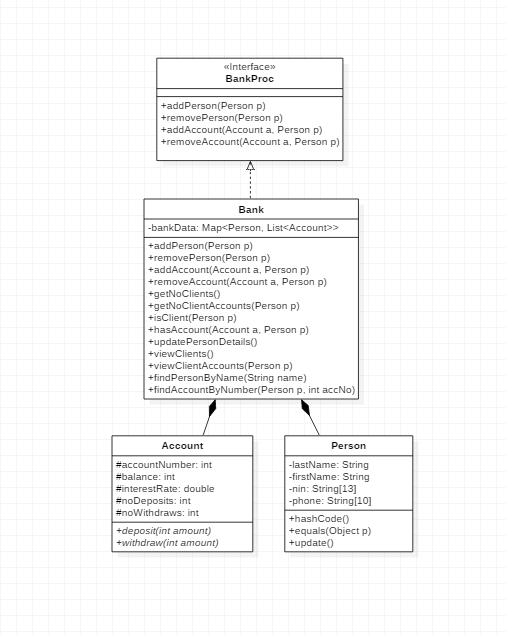
- InsertView and AccountsView, extensions to AdminView, providing additional functionality;

- UserView, which has the functionality the client has access to;

The main class that launches the application is called StartView. It gives access to either the area dedicated to the bank worker, called AdminView, or the area dedicated to the clients of the bank, called UserView.

Class AdminView presents in a table, information about all the client and provides part of the functionality. The administrator can add (InsertView), change information about or remove a new person from the system, or see details about the accounts of that person (AccountsView). Furthermore, AccountsView opens up a separate view, presenting data also in a table, and providing the basic operations on accounts – open, close account.





## Used packages

javax.swing

Provides a set of "lightweight" (all-Java language) components that, to the maximum degree possible, work the same on all platforms.

java.awt

Contains all of the classes for creating user interfaces and for painting graphics and images. A user interface object such as a button or a scrollbar is called, in AWT terminology, a component. The Component class is the root of all AWT components. See Component for a detailed description of properties that all AWT components share.

Some components fire events when a user interacts with the components. The AWTEvent class and its subclasses are used to represent the events that AWT components can fire. See AWTEvent for a description of the AWT event model.

A container is a component that can contain components and other containers. A con tainer can also have a layout manager that controls the visual placement of components in the container. The AWT package contains several layout manager classes and an interface for building your own layout manager. See Container and LayoutManager for more information.

java.util

Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

## Observer pattern

The application does not imply a strong bonding between the client and its accounts, as there is no Person field in any of the account classes. Instead, accounts of the same person are kept in a list and mapped based on the personal identification number of the person.

To provide updates to the client when certain operations took place on one of his accounts, the use of the Observer design pattern was the most appropriate approach.

The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.

It is mainly used to implement distributed event handling systems, in "event driven" software. Most modern languages such as Java and C# have built in "event" constructs which implement the observer pattern components, for easy programming and short code.

The observer pattern is also a key part in the familiar model–view–controller (MVC) architectural pattern. The observer pattern is implemented in numerous programming libraries and systems, including almost all GUI toolkits.

In this case, each account is assigned, after it is created and inserted in the system, an observer, the account main holder. Inside the account, certain methods, like deposit and withdraw, call the update method of the observer, notifying through a message about the changes that occurred.

# Results

The application is able to simulate the way in which a bank operates, by adding/removing accounts, depositing/withdrawing money and reporting the bank’s state at any time during the use of the application.

# Conclusion

By means of using serialization and a friendly graphical user interface, a user-friendly application which simulates the management of a bank has been developed. However, there are further improvements that can be done, in order to adapt the application to real-world situations. It would be useful to generate periodical reports for the clients. Moreover, the application could be further developed to have different types of users, administrator and client user, with more distinct rights and a login step.

# 7. Bibliography

<https://www.tutorialspoint.com/java/java_serialization.htm>

<https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>

<https://sourcemaking.com/design_patterns/observer>

<https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html>