**Technical University of Cluj-Napoca  
Faculty of Automation and Computer Science  
Computer Science Department**

***Drimbarean Maria Cristina***

***Group:30425***



**Table of contents**

1. **Assignment objectives**

1.1General view

1.2 Personal approach

**2. The analysis**

2.1. Otput computations

2.2 Modeling

2.3. Scenarios

2.4 Use cases analysis

**3. Design**.

3.1. UML diagram

3.2. Data structures

3.3. Class projection

**4. Implementation and testing**

**5.Results**

**7. Conclusions**

**8. Bibliography**

**1.Assignment objective**

**1.1 General view**

*Task was described with the following statement:*

*Description:*

*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.*

*Input data: - The Bank accounts saved in a text file*

*Minimal output: -Basic account operations like withdraw,deposit, transfer and for the admin add/remove accounts, and clients, apply interest and see the clients and their accounts*

**

*1.2Personal approach*

*As the implementation is provided in the scheme, i had to take care not to make the coupling havier, so i had to somehow connect the persons and the account but that is easily done with the hash map method. I mapped each person as a key*

**Application description**

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.

***2.The analysis***

* 1. **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.

* 1. **Input and Output**

All the information that will be registered into the application from the graphical user interface, will be tested to be according to specific 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 Name and an index, which the Bank will use to distinguish clients and shall be randomly generated.

A client can have one or more accounts. In this accounts, some operations can be performed like adding or withdrawing money. If the withdrawn amount exceeds the available amount in the account, an error message shall be generated.

The user has the possibility to delete client accounts or even delete totally clients from the bank. No measure has been taken regarding to the money that a user has at a given moment of time and the user removal at that 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.

***2.1Output computations***

**2.2Modeling**

**2.3Scenarios**

**Title**: Client interaction with bank

**Resume**: Client enters in application and performs operations in the accounts that are available for him: withdraw a sum of money, or deposit a sum of money.

**Actors**: Client

**Scenarios**:

1. Preconditions: application is ready to use
2. Normal scenario:
3. User successfully inserts the input parameters;
4. User pushes the "Withdraw!" button to request the sum;
5. The sum if available is extracted from account and returned;

Alternative scenario:

1. User types wrong data in the input fields;
2. User pushes the "Deposit!" button attempting to start the simulation;
3. An error message is displayed;
4. Another change of inserting/correcting the input data is provided.

**Title**: Administrator takes care of bank database

**Resume**: Admin enters application and perform administrative operations, he cannot extract or deposit money in accounts

**Actors**: Administrator

**Scenarios**:

a)Preconditions: application is ready to use

b) Normal scenario:

1)User successfully inserts the input parameters;

2)User pushes the “Add Account!” button to add a new account;

3)The application creats a new account;

Alternative scenario:

1)User types wrong data in the input fields;

1. User pushes the "Deposit!" button attempting to start the simulation;
2. An error message is displayed;
3. Another change of inserting/correcting the input data is provided.

**2.4 Use cases**

This application simulates operations that clients request to the bank, for accesing their accounts. Therefore money can be extracted and deposited and also reports can be requested.

**2.4.1 Add new client use case**

**Title**: Add new client

**Resume**: The users inserts a new client into the Bank Management application by filling in data like Name, Address, Personal Identification Number.

**Actors**: User, system.

**Scenario**:

1. Preconditions: application is available to the user and he/she can choose data
2. Normal scenario:

B1. User succeeds in inserting data into the desired text field;

B2. User selects the button for saving the properties;

B3. Data is validated and a new client object is added to the bank;

1. Alternative scenario:

C1. User inserts wrong data in the user interface;

C2. A window appears which informs the user what went wrong;

C2. Another chance of inserting correct data is provided.

2.4.2 **Add new account use case**

**Title**: Add new account

**Resume**: The user can add a new account into the bank, wither a spendings account or one for savings. For a savings account, the user must fill in also the amount deposited and the period. In case of a spendings account, only the amount is necessary. Finally, a client shall be selected from the list of clients.

**Actors**: User, system.

**Scenario**:

1. Preconditions: application is available to the user and he/she can choose data
2. Normal scenario:

B1. User succeeds in inserting data into the desired text field;

B2. User selects the button for saving the data;

B3. Data is validated and inserted into the Hash Table;

1. Alternative scenario:

C1. User inserts wrong data in the user interface;

C2. A window appears which informs the user what went wrong;

C2. Another chance of inserting correct data is provided.

2.4.3 **Remove client**

**Title**: Remove client

**Resume**: The user can delete a client from the bank by selecting him from the list of clients displayed in the appropriate form.

**Actors**: User, system.

**Scenario**:

1. Preconditions: application is available to the user and he/she can choose data
2. Normal scenario:

B1. User succeeds in selecting data from the clients list;

B2. User selects the button for deleting the client;

B3. Information about the client is deleted from the application.

1. Alternative scenario:

C1. User doesn’t select a client and presses the delete button;

C2. A window appears which informs the user what went wrong;

C2. Another chance of selecting correct data is provided.

2.4.4 **Remove account**

**Title**: Remove account

**Resume**: Firstly, in the form for removing an account, the user has to select the client form the list of clients for whom he wants to remove the account. Then, another form will appear in which the user shall choose which account shall be removed.

**Actors**: User, system.

**Scenario**:

1. Preconditions: application is available to the user and he/she can choose data
2. Normal scenario:

B1. User succeeds selecting data from the list of clients;

B2. User selects the account that should be removed;

B3. The account is removed from the client accounts and from the bank;

1. Alternative scenario:

C1. User doesn’t select data from the user interface;

C2. A window appears which informs the user what went wrong;

C2. Another chance of selecting correct data is provided.

2.4.5 **Perform operation**

2.4.5.1 **Deposit**

**Title**: Deposit

**Resume**: The user shall select the client name and account to which he wants to deposit money.

**Actors**: User, system.

**Scenario**:

1. Preconditions: application is available to the user and he/she can choose data
2. Normal scenario:

B1. User succeeds in selecting data from the desired field;

B2. User inserts the amount of money that is going to be deposited;

B3. Application diplays a message to the console;

1. Alternative scenario:

C1. User inserts wrong data in the user interface;

C2. A window appears which informs the user what went wrong;

C2. Another chance of inserting correct data is provided.

2.4.5.2 Withdraw

The withdraw operation is very similar to the deposit operation with the difference that a user can withdraw money from a spendings account and only an available amount of money could be withdrawn, otherwise an error message will be displayed.

2.4.6 **List client**

**Title**: List client

**Resume**: The user can see a list of client and details about them into the console by selecting the „List client” button from the menu.

**Actors**: User, system.

**Scenario**:

1. Preconditions: application is available to the user and he/she can choose data
2. Normal scenario:

B1. User presses „List clients”;

B2. The list of clients is displayed in console;

1. Alternative scenario:

C1. User inserts wrong data in the user interface;

C2. A window appears which informs the user what went wrong;

C2. Another chance of inserting correct data is provided.

2.4.7 **List bank**

**Title**: List bank

**Resume**: The user can list into the right-hand side text area of the graphical user interface all the activity that has taken place in the bank.

**Actors**: User, system.

**Scenario**:

1. Preconditions: application is available to the user and he/she can choose data
2. Normal scenario:

B1. User presses „List All”;

B2. The activity log of the bank is displayed onto the console;

1. Alternative scenario:

C1. User inserts wrong data in the user interface;

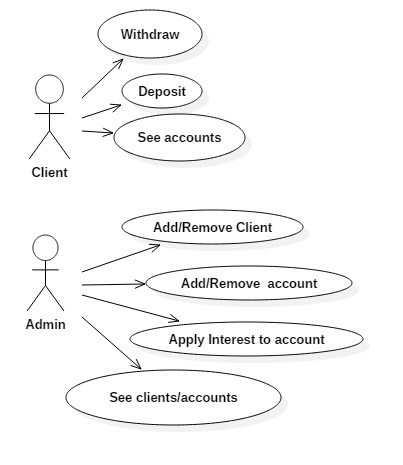
C2. A window appears which informs the user what went wrong;

C2. Another chance of inserting correct data is provided.

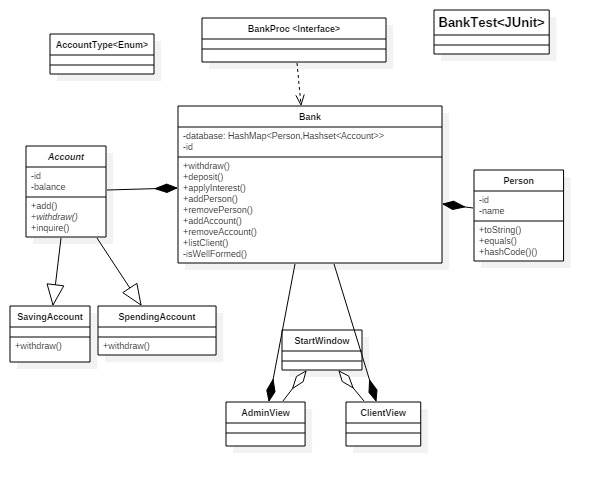
**3.Projection**

3.1 UML Diagrams

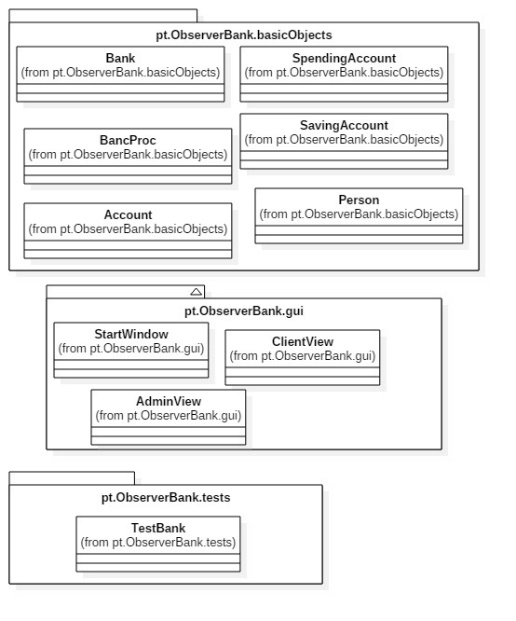
a)Use case



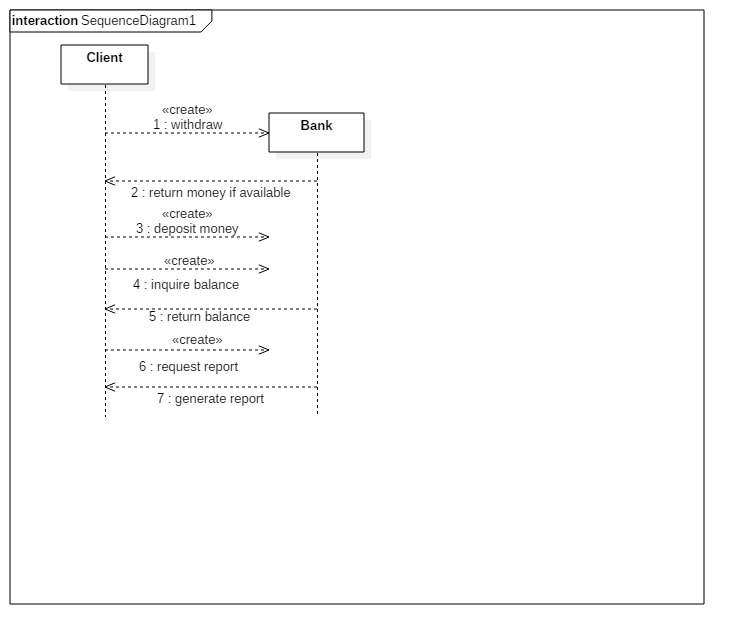
b)Class diagram

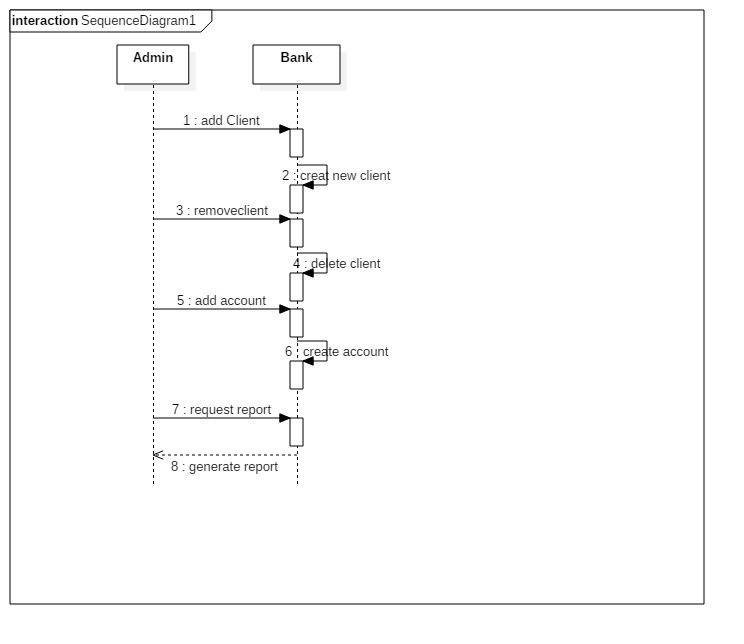


c)Package Diagram



d)Sequence diagram





**3.2Data structures**

HashMap maintains key and value pairs and often denoted as HashMap<Key, Value> or HashMap<K, V>. **HashMap** implements Map interface. HashMap is similar to Hashtable with two exceptions – HashMap methods are unsynchornized and it allows null key and null values unlike Hashtable. It is used for maintaining key and value mapping.

It is not an ordered collection which means it does not return the keys and values in the same order in which they have been inserted into the HashMap. It neither does any kind of sorting to the stored keys and Values. You must need to import java.util.HashMap or its super class in order to use the HashMap class and methods.

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

1. **Arrays**

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

1. **Array List**

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:* Array List can be used to store the list of clients. 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.

1. **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.

1. **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.)

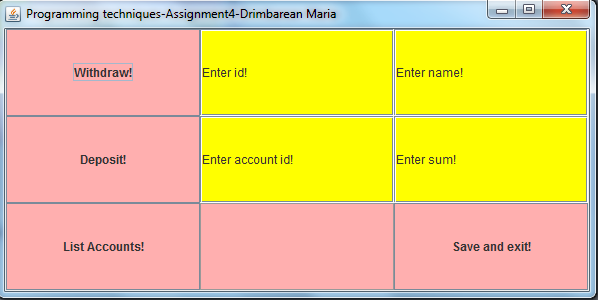
**Chosen data structure: Hash Map**

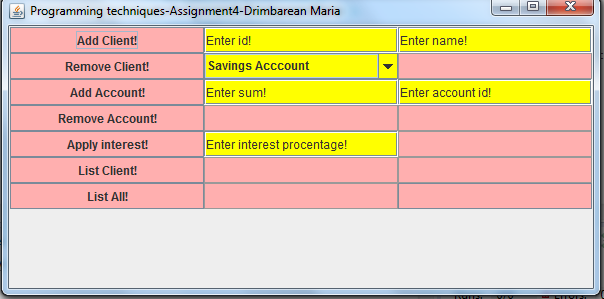
Since this application will not use multithreading, there is no problem unsing Hash Map. Moreover, we can come across the situation of permitting null entries. Another advantage of using Hash Map is that this collection is collision safe, because the entries of the table act like a linked list. When 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.

**3.3 Class projection**

**4.Implementation an testing**

The input windows lets you interact with the bank’s database:





**5.Results**

A database of the accounts is maintained.

**6.Conclusions**

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. One aspect that has not been taken into account when developing the application was the management of transaction commissions. When withdrawing/transferring money, some commissions proportional to the amount processed have to be taken into account. Moreover, the application could be further developed to have different types of users: administrator and client user. These improvements that can be applied are to be taken into account when developing further this application.

**Further development**:

7. *Bibliography*

1.https://docs.oracle.com/javase/tutorial/uiswing/components/combobox.html#uneditable

2.http://www.tutorialspoint.com/junit/junit\_writing\_tests.htm

3.http://www.vogella.com/tutorials/JUnit/article.html

4.http://tutorials.jenkov.com/java-itext/getting-started.html

5.http://www.vogella.com/tutorials/JavaPDF/article.html

6.http://javahungry.blogspot.com/2013/08/hashing-how-hash-map-works-in-java-or.html

7.http://javahungry.blogspot.com/2014/03/hashmap-vs-hashtable-difference-with-example-java-interview-questions.html

8.http://javahungry.blogspot.com/2013/08/how-sets-are-implemented-internally-in.html