Technical University of Cluj Napoca

Faculty of Automation and Computer Science

Homework 4 – Bank Account

Lorand Berekmeri

Group: 30425

Date: 13.05.2016

Objective

The objective is to develop a basis for a bank applications. This application must perform all the functions that we can make to the bank. Let's create some accounts put money to extract, or if we are looking for admins to certain clients. In this issue I have never used a database to store all accounts and persons holding accounts because we chose to retain these datalii a file. So we run the app take over all the data in the file and at the end they all application data in the file we will write back. Another II Project of the issue is understanding the concepts serializer and deserializer objects. Another goal for us is to get in touch with Design by Contract programing.

Task

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. 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).

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

Problem analisys, use cases, modelling

*Analisys*

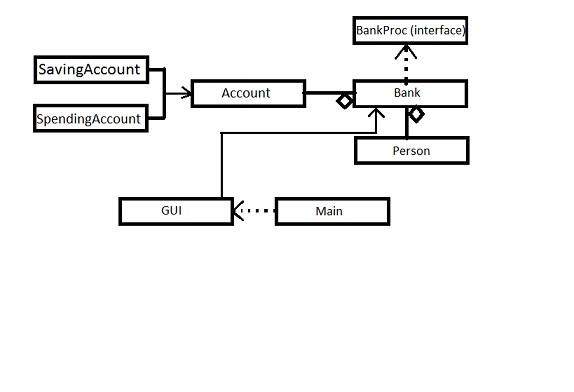
The application must realize the user interface and the bank. The user must be able to quickly create many accounts he wants. He may to choose to create an account on their "Spending Account" or one of a kind "Saving Account". He should be able to introduce or to extract money from a particular account. Admin may have to delete or search and view anumtie people who have an account. Thus all the data from the application must be retained in a file, and each time you open the application all data in the file will automatically be entered in our application, as well as exiting the application all data will be written back into the file.

*Modelling*

In achieving modeling application we used the concept of Desing by Contract, but Serializable and Deserializable. To retain use of user accounts have a Hashtable object type. Thus these Hashtable has the key and CNP each account holder and as object has a list of objects of type Account. Because of that every customer can have multiple accounts. To be able to have different construct without the other hashtable we created abstract class Account and created two classes that extend this class, so all methods of the Account class 2 classes they receive must implement. By using the Inheritance I have removed the possible hashtable declaration for each type of account. I have used conceptual of Desing by Contract by implementing BankProc interface so BankProc class implements the interface Bank and all methods of the interface.

Regarding the OOP paradigm, all the attributes of all classes are private. So I realized encapsulating every class. I used inheritance and Design by Contract. All classes except class GUI and contain getter and setter Account for each attribute of the class.

*Class Diagram*



**GUI**

(-) JFrame Frm

(-) JMenuBar meniu

(-) JMenu file

(-) JMenu Accounts

(-) JMenu Operations

(-) JMenu search

(-) JMenuItem eMenuItem

(-)JMenuItem showAccounts

(-)JMenuItem insertAccounts

(-)JMenuItem deleteAccount

(-)JMenuItem DepositMoney

(-)JMenuItem extractMoney

(-)JMenuItem searchId

(-)JMenuItem searchCNP

(-)JButton insertButton

(-)JButton insertButton

(-)JButton deleteButton

(-)JButton enter

(-)JButton extractButton

(-)JButton depositButton

(-)JButton searchButton

(-)JTextField formNumeText

(-)JTextField formNumeText

(-)JTextField formNumeText

(-)JTextField formNumeText

(-)JTextField formNumeText

(-)JLabel formNume

(-)JLabel formMoneyAmount

(-)JLabel formId

(-)JLabel formPrenume

(-)JLabel formCNP

(-)JScorllPane scrollPane

(-)JTable tableAccounts

(-)JTable tableAc

(-)int iddAccounts

(-)JRadioButton saving radion

(-)JRadioButton spendingRadio

(+)GUI()

(+)clearScreen

#DefaultTableModel buildTableModel

#JTable searchAccount(String id, mesaj)

(-)JTable showAccounts()

(-)JTable searchAccounts()

**Account**

#Person person

#int accountId

#double moneyAmount

#String accountName

#Person getPerson()

#int getAccountId()

#double getMoneyAmount()

#String getAccountName()

*#double depositMoney(double Amount)*

*#double extractMoney(double Amount)*

*#boolean correctParam()*

**BankProc**

(+) readAccounts()

(+) writeAccounts()

(+) addAccount(Account acc)

(+) deleteAccount(Account acc)

(+) Account getAccount(int acc)

(+) double depositMoneyIntoAccount(int AccountId, double money)

(+) double extractMoneyIntoAccount(int accountId, double money)

**Main**

(+)main

**Bank**

# HashTable<String, ArrayList<Account>>bank

(+) Bank()

(+) readAccounts()

(+) writeAccounts()

(+) addAccount(Account acc)

(+) deleteAccount(Account acc)

(+) Account getAcccount(int acc)

(+) double depositMoneyIntoAccount(int Id, double m)

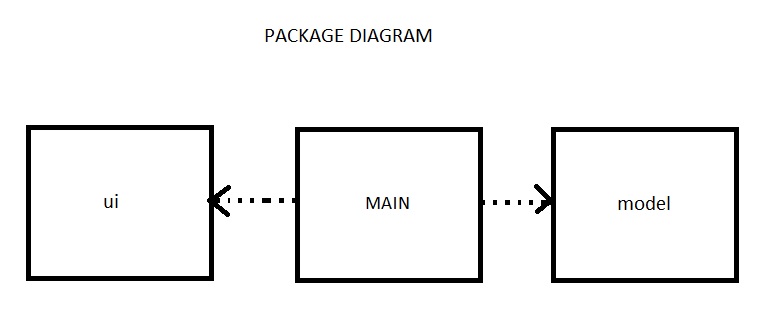
(+) double extractMoneyIntoAccount(int Id, double money)

(+) String toString()

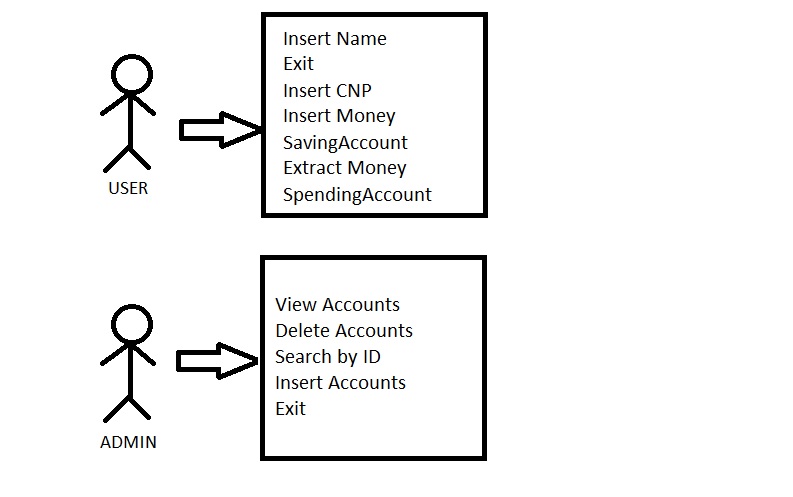
#int nextIdAccount

# Vector<Vector<Object>> searchAccountA (String id, String mesaj)

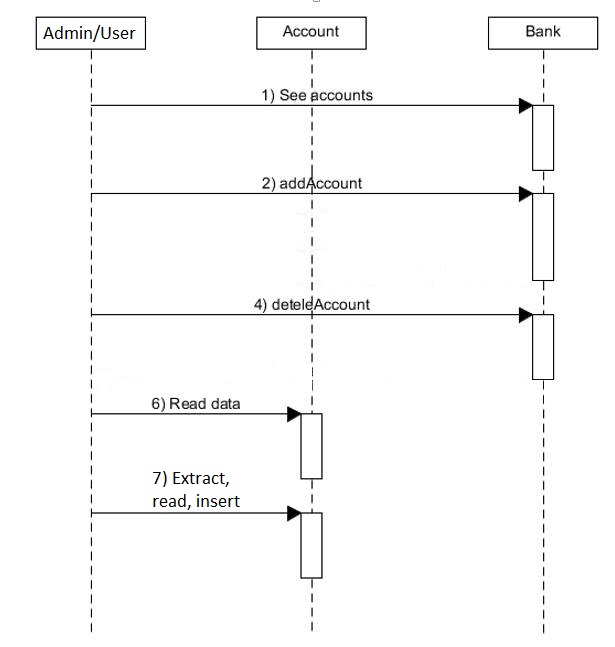
(+)boolean isWellFormed



*Use case Diagram*



*Sequence Diagram*



*Classes*

@**GUI Class** represents the GUI application (user interface). In this class we used buttons, menus, tables, TextPane, TaxeField, ScrollPane, and elements of the controller. This class is very important because in the absence of a Class Controller, it performs all operations here. GUI class extends JFrame and implements interface ActionListener because we need actionEvents buttons and

@**Account Class** is an abstract class that has the attributes has all the necessary attributes an account, namely: account number, amount in the account, account type, interest on the account but also a beneficiary. This class implements java.io.Serializable because we serialize and deserialize an object of type Bank which contains an object of type Account.

@**SavingAccount Class** extends Account class. In this class we initialize all class attributes of the Account class. Thus the beginning each account has the amount of money equal to 0 and the account type for this class is "Spending Account" and the gain is 0.15.

**@SpendingAccount Class** extends Account class. In this class initialize all class attributes Account. Thus the beginning each account has money 0 and account type for this class is "Spending Account" and the interest is 0.05.

**@Bank Class** represents the class in which we store all accounts. This class is implemented with a Hashtable where the key is a String, the numeric code of the account holder, and the node is composed of a list of accounts. In this list we could have accounts of type Spending Account, and also the type of Saving Account. In this class I implement the BankProc interface and the Serializable, because the object of type Bank it will be serializeable.

**@Person Class** is the class which has the attributes all the attributes of a person. Namely: first name, last name and the numeric code. In this class we needed to have only getters and setters and a method is WellFormed.

*GUI Interface*

I made it in Interface swing by several important elements of Java.

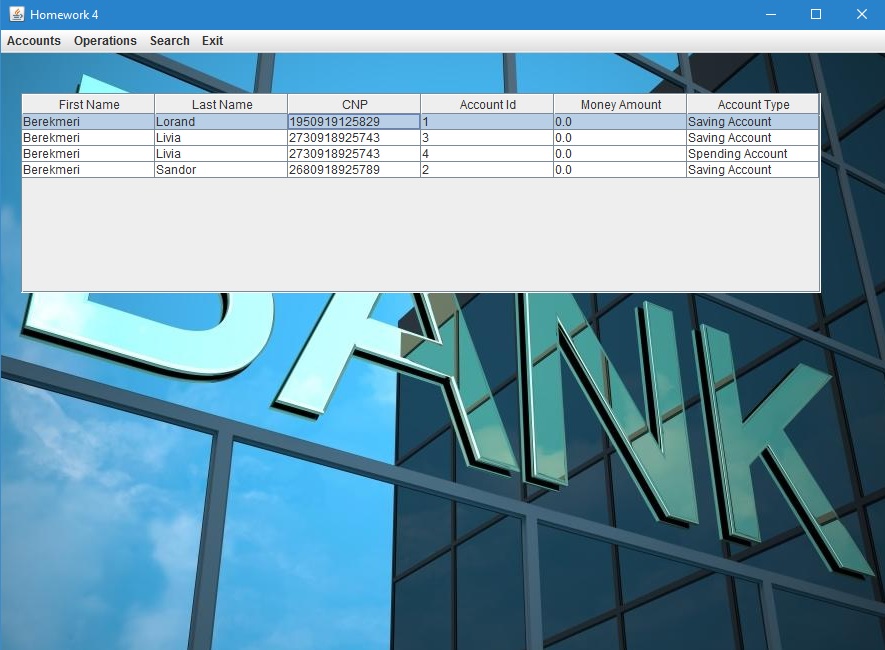


I’ve used the JMenuBar and JMenuItem for achiviement of the Menu. I used to achieving tables JTable and JScrollPane products and orders, and classic JButton, JTextField and JTextPane.

As we see in the picture above, the application interface is very easy to understand and navigate through the application because we tried to make it as visible and not too complicated. I managed this menu in menu bar. Here we introduced all commands that you want to do. As we observe if we are to achieve certain commands that are related to accounts such as viewing accounts, inserting, deleting an account just have to go on "Accounts". If we want to create an account just press the button "Create Account" and then we'll defend all the data that you need to enter the account creation. If we want to achieve our various operations on accounts such as adding or withdrawing a sum of money. If you want to look for an account after account number it is very easy, because we go on the label "search" and then click "search by account number" where we will be required to enter your account number that we want to search and press the button next to him. If you want to search all accounts a single person go again labeled "Search" and then "Search for Person" soon we will see a text where we will be asked to enter personal identification number that we want to look for. If you did not find any person that CNP will appear an error message that we will strictly observe signal, that person has no account. Otherwise, we will see a table of all individual accounts, and details about them such as how much money is in your account or type of account or an acquired for an operation.

*Implementation*

In this subsection of the documentation I will describe the methods they've used since I’ve already explained above all the classes that we used and their role in the application.



So I have to say that each class uses encapsulation because all the attributes of all classes are declared private. This helps when someone wants to change something in our application. Get and set methods will not describe, just to mention them.

Account class

-This Class is abstract so it has a builder, but has different attributes. It contains get and set methods, but some abstract methods that would be implemented in classes that inherit (Spending Account and SavingAccount);

Spending Account class

**@ SpendingAccount Constructor (int acc, Person p)** initializes all the attributes of the parent class "Account" asftel account type is on "Saving Account" money in the account are initialized to 0, and gain for extracting money is initialized with 0.15;

**Parameters:**

**- acc** account number

**-p** is the person who dobanadeste this account

**@AddMoney Method (int money)** is overwritten and is aduagarea amount of money in the account type Spending Account

**Parameters: money** is money which I have added

**Returns:** the amount of money inserted into account

**@ExtractMoneyAccount Method (double money)** is overwritten Class Account and is withdrawing a sum of money with commission

**Parameters** being represented money amount you wish to extract

**Returns:** the amount of money extracted

**@IsWellFormed Method ()** is a method to verify account type Spending Account has all the attributes inserted correctly

**Returns:** true if everything is OK, false

SavingAccount class

**@SavingAccount Constructor(int acc, Person p)** class initializes all the attributes of a parent class "Account" so the account type is on "Spending Account" money in the account are initialized to 0, and the gain for placing money is initialized by 0.05;

**Parameters: account** number -acc

**-p** is the person who earns this account

**@ AddMoney Method (int money**) is overwritten and is aduagarea amount of money in the account type with a small tax Spending Account

**Parameters:** money is money which I have added

**Returns:** the amount of money inserted into account

**@ExtractMoneyAccount Method (double money)** is overwritten Class Account and is withdrawing a sum of money without commission

**Parameters: money** being represented the amount you wish to extract

**Returns:** the amount of money extracted

**@IsWellFormed Method ()** is a Method to verify account type Spending Account has all the attributes inserted correctly

**Returns:** true if everything is OK, false

*Class Person*

Beside the constructor and isWellFormed ()method, class Person contains only getters.

**@ Person Constructor (f String, String L, foreign c)** class initializes class attributes

**Parameters: -f** is FirstName

**-L** represents the LastName

**-c** represents the personalnumeric code

**@IsWellFormed Method ()** for Person class is a method to verify all class attributes were invalid So first and second name must be different VID and PNC must have 13 digits

**Returns:** true if all the attributes meet the requirements and False if one of them meets the condition

*Bank Class*

Bank @Constructor class, which initializes the Hastable

**@ AddAccount Method (the Account)** is implemented because there BankProc interface class that implements Bank. She adds hashtable an account unless any account then it will allocate memory for the list of accounts and will add a new account. If you already have an account just adds another account

**Parameters:** The account is to be added

**@ RemoveAccount Method (the Account)** is implemented because there BankProc interface class that implements Bank. It deletes an account in hashtable, then after deletion an account after account deletion, the person will no longer have any accounts and we delete this person from the database.

**Parameters:** to represent the account will be deleted from the application.

**@AddMoneyIntoAccount Method (idAccount int, int money**) seeking account with account number equal to idAccount. If he finds will insert money in that account amount if will not find such an account will return -1;

**Parameters: - idAccount** account number in which to insert the amount of "money"

**-money** is the amount to be inserted

**@ ExtactMoneyFromAccount Method (idAccount int, int money)** seeking account with account number equal to idAccount. If he finds will extract money in the account if the amount will not find such an account will return -1; If the amount withdrawn is greater than the account will return -1;

**Parameters: idAccount** account number from which to draw.

**money** represents the amount the customer wants to extract from your account.

**@Metoda GetAccount (int AcCN)** seeking account with account number equal to AcCN. If you find it returns, if it does not find an account returns null;

@pre condition check that the account number is as high as 0.

@post condit check if your account has been found, if it was found the method applies isWellFormed () on it to check if it has all the right attributes

**@ReadAccounts Method ()** is the method by which the object type deserializeaza Bank. All information in the file is vo bank.ser introduce the subject bank.

**@WriteAccounts Method ()** is the method that serializes the object type Bank. All information of the object bank will insert file bank.ser

@pre condition is checking if the object bank is different from null

@post condition check that was written to the file bank.serx

**@ ToString Method ()** is a method that creates a string with all the objects in the object bank.

**@ DefaultTableModel buildTableModel Method ()** creates a table of all the data in the type hastTable bank. Thus we have seven columns that contain name, PIN and account data. Each line will contain one account. If a person owns multiple accounts, his name appears several times.

Returns: all accounts HastTable table;

**@ FindAccount Method (String str, int flag)** is a protected method used since ete from another class and help us find specific accounts after PNC after a person or account number, depending on the flag. This function creates a table based on what they want.

**Parameters: str** is the word after that search begins

**flag** is the parameter that will decide after looking after CNP or after the account number

**Returns:** a table with accounts found.

**@ SearchAccountWithNumber Method (int n)** is a private method because it is used only in this class and help us look for that same account account number equal to n, if no such account will return null

**Parameters: n** is the number of account that we want to find

**Returns:** a table with a single line that is the account if there is no such account to return null.

**@ SearchAccountWithCNP Method (String CNP)** is a private method because it is used only in this class and help us all accounts looking person equals CNP function parameter. If not gasete no customer will return null. If it does not create a table of all accounts that person.

**Parameters: CNP** is CNP person to whom we want to display all accounts

**Returns:** a table of all individual accounts, if no person will return null.

*Results*

When the application runs, that too all data from a file. Such application will always keep all data. After I saved all the data already we can perform various operations on bank accounts. After I made the desired operations (deletion, insertion or addition account, extracting money from an account) all data will be written back into the file, so they keeping their eyes safe.

*Conclusions*

In conclusion for this project, I want to bring my conclusions we have drawn following the theme. I learned new things of Java. I learned to use concepts such as: Desing by Contract, Serializable and Deserializable.

This theme was one fun because we made a practical application. Of course application that I realized it is very minimal compared to those that are on the market online. But after realizing this theme I can say that I could do very easily in such an application, but much larger.

The application has many areas that can be greatly improved. We could add a network admin only he had access to inserting / deleting / modifying accounts. You can add several accounts and transfer between different accounts with a certain interest.