# Assignment 4 Documentation

# Design by Contract

Dreghici Popa Vlad

Group 30422

1. *Problem description*

Consider an application Bank management for clients and accounts of the clients in a real life Bank. Define and implement the required class diagram.

1. *Problem analysis*

Bank management system are usually used in large banks to have a better grip of their clients and accounts. They are conveniently made in such a away that a user can add a new client to the bank by inputting the characteristic info of a person like their name and cnp, and after they add a person they can create an accout for that respective person by selecting the id of the person, the name, and the type of account they want to create like Spending or Saving.

The project should be able to connect to a serialized file which contains all the data about the persons in the bank and all of their accounts. This file is made using two a methold called serialization which when you start the app it will write all the data from the file to the JTable and when any modifications are made it will take them and write them inside the file so that it saves them. The fields of the person table should be id and name. The fields of the accounts jtable are many: Name of the holder, cnp of the holder, the account id of the account, the type ov the account, the period for which it was decided if it is a saving account, the money in the account, balance, and the interest of the account.

The user of the application should be able to input the following data: realize operations from the gui directly in the file, operations for each table are Show all, update, delete, insert and other necessary ones in order to get the data from tables, like adding a new account, modifying it and so on.

The output is shown in a JTable. The table is created for each table from the file, resulting in the end we have 2 tables. One for accounts, one for persons. For each table the user will be able to see all the data in it. And eventually delete a row if needed

Deposit money



Edit account



Withdraw money



Delete account



Add account



Edit person

Delete person

Add person

Run application

Go back

Review Table

User

Select option

1. *Design*

*Classes*

The classes is a blueprint that describes the behavior of the object it supports. The classes are organized in packages, which basically are a collection of classes, interfaces. In our case, the classes are organized in a MVC manner.

An object instantiated from a class has states and behaviors declared in the respective class.

The project will be split in a certain manner. There will be 4 packages: Model, Test, View (gui classes) and finally Controller which controls all the classes that execute operations on the file we access. The model package will contain all the back end processes, like the algorithms, person, accounts. The view package will contain the front end classes. In this case we have for classes: gui, where we have the main options to do on an account like add new one, edit, remove, add money and withdraw. This is the visualization of our application.

**Account class:** The account class will extend Observable in order to be able to implement the observer so a person is notified when there are chages to their account. It will also implement serializable so we will be able to serialize the data from it into the file. It implements all the attributes which a real life account should have like : id, balance, account name, interest and period. It will also have an empty constructor. The methods implemented in account class are the usual ones: getters and setter for all of the attributes defined earlier and also 2 methods to deposit and withdraw money from the account. It will also define a method called isWellFormed which will flag if the account is not good, in order to test it in the jUnit tests.

**SavingAccount class:** Extends the account class, being downcast. Saving account will be a subclass of the Account class, it will acquire all the attributes in the account class and will contain only the deposit and withdraw methods. It will also implement a period, interest system in which the money deposited will be summed with a interest for a certain period given

**SpendingAccount class:** Extends the account class, being downcast. Spending account will be a subclass of the account class. It will acquire all the attributes and also implement the deposit and withdraw methods but no interest or period.

**Person class:** person class will implement observer in order to be able to add it as an observer to their accounts and also implements Serializable. It will contain 2 attributes the cnp and the name of a person and will implement the usual getters and setters. Also it will have a method wellFormed which check if the person is well formed for the assertions. It will also override an update method from the observer interface to be used when notified.

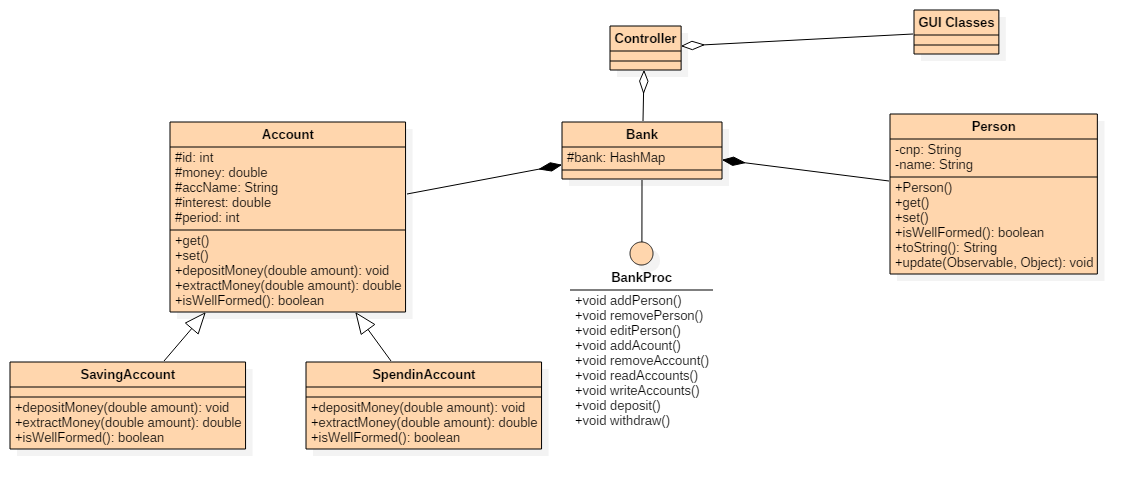
**BankProc interface:** the bankproc interface will be used in order to realize the design by contract. It will contain all the methods implemented in the bank class which implements the bankproc. For each method we will have pre and post conditions in order to assert them at the beginning and at the end of the methods.

**Bank class:** the bank class will implement the bankProc interface and will have in it all the methods described in the interface. For each method we will write the algorithm but before and after we start it we will always assert the pre and post conditions written in the interface

**Gui classes:** those classes are similar between them. All of them implement a JTable which displays the data from the file, and for each case we have certain operations. For accounts and persons we have the options to delete, update, insert and get all data from file.

**GUI Class:** is the main gui class we see when we start the app. It will display the JTable with all the data from the file. You will be able to view the person from this window. Add, edit and remove accounts. Deposit and withdraw money from certain accounts.

**PersonsFrane class:** is the view where we see all the persons in the hashMap. You will be able to add, remove or edit a person from the map.



1. *Implementation*
   1. *Account Class*

In real life, an account usually has a few attributes. In our case the accout number will be the id of it, the balance will be a double variable, the type of the account will be the name of the account (Saving or Spending), the interest will be a double and the period will also be an int. It will have getters and setters but additionally it will have 3 methods. One to deposit money, one to withdraw money from account and one to check I it is well formed.

* 1. *Saving Account Class*

The saving account class will extend the account class and inherit all of it’s attributes. After that it will implement the deposit money method with a twist. It will take the period you want to deposit the money for and a hardcoded interest and will add the amount of money computed in the balance of the account.

* 1. *Spending Account Class*

The spending account class will extend the account class and inherit all of it’s attributes. After that, it will implement the deposit and withdraw method but unlike the saving account there will be no period or interest, the person being free to deposit or withdraw money whenever they want

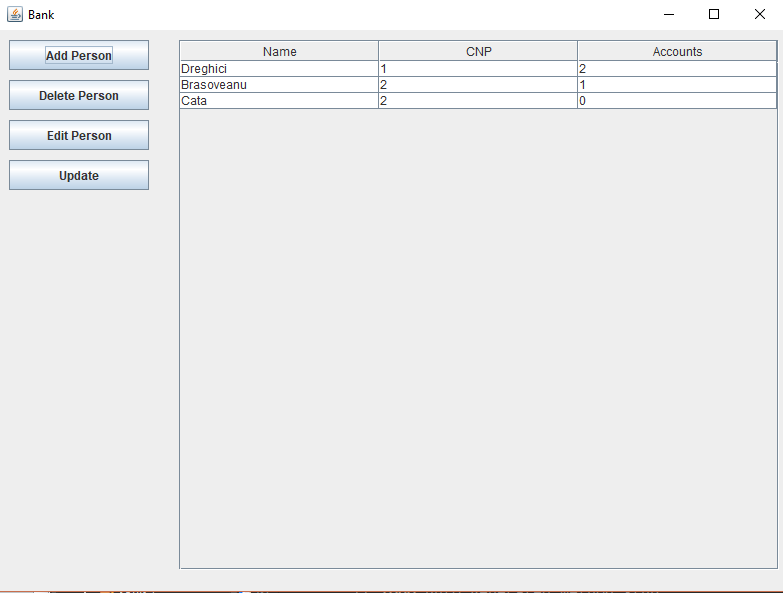
* 1. *Person Class*

The person class will implement the observer interface and define the update method to notify the observer. It will also implement all the usual getters and setters in order to be able to modify the person. The isWellFormed method will be added for helping in the junit test and with the design by contract. It will check if the attributes are not null.

* 1. *Controller Class*

The controller class is exactly what the name says, a controller. It will implement the control of our application. The controller will also link the presentation part with the data access layer. First of all it will connect to the file. It will execute certain reads one a particular button is pressed. For example in the persons menu when the admin wants to add a new person, the admin will have to complete the necessary fields with the data of the person and after that press the add new person button. Once this button is pressed the controller will make the link between the file and the JTable, perform the requested action and insert the client in the table. This is the procedure for most of the algorithms implemented.

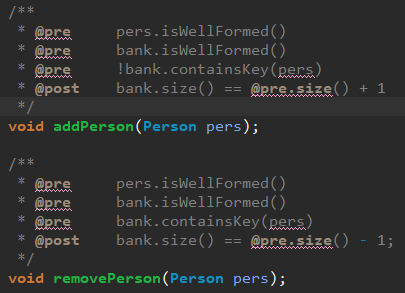
* 1. *Gui Classes*

**

The gui classes are the front end of our applications. They will display all the data taken from the tables and will make the interface user friendly. The gui is separated in 2 parts. The persons page which contains a table with data of the clients from the file, 4 buttons to perform operations. Similar to the persons page, the accounts page also contain a table with acount from the file, with the following elements: a table which contains all the data from the data base, 6 button to perform operations in the table and file.

* 1. *BankProc interface*

The bankProc interface is very important, it will implement all the pre and post conditions for all the methods we will implement in the bank class. All those conditions will be important and will be asserted at the end and start of each method in order to check for bugs. This technique is called design by contract and we will be able to debug the code much easier using those techniques. Depending on what the method does the pre and post condition will be different on each method.



* 1. *Bank class*

The bank class will implement the BankProc interface and Serializable in order to serialize data in the file and from the file. First of all the bank class will override 2 methods for serialization. ReadAccounts will get all the data from the file and will add it into a hashMap which will be displayed inside the GUI with a JTable. Write accounts will fill the file with the new data inserted or modified or removed every time we perform a certain operation on the table. The other method are pretty much self explanatory. The add remove and delete person do what they say, the add account, edit and remove accounts also, and finally the deposit and withdraw money look for the account given and add or withdraw money from it. Before all the methods start, inside them we assert the pre conditions define in the bankproc interface and after the methods finish we assert the post conditions defined inside the bankproc interface.

1. *Testing*

The testing part will be done by creating a new class called BankTest.Java. Inside this class we will define a new Bank object with the hashMap. We will create 5 persons with different cnp and names. We will create 4 new accounts, 2 saving and 2 spending.

We will have a few methods to test the application. First one is testAddAccounts which will check 4 methods to add accounts to different persons. If the assert conditions are correct then the JUnit will not give any exception. If they are false then it will throw a Assertion Error. The next method is to remove accounts. Same idea. The testAddPerson and testRemovePerson will check if we can add and remove a person from the hashMap. Finally the testDeposit and testWithdraw will try to add and remove money from the hardcoded accounts inside the hashMap.

1. *Results*

During the usage of the application the user will be able to execute certain operations on the two tables from the database. First of them is the accounts table, here the user will be able to insert a new account, update an existing account or even delete a account selecting it. Second, the persons table, where the user will be able to execute similar commands on the persons as on the accounts.

1. *Conclusions*

By working on this assignment I acquired more knowledge about how to work with a serialized file and to serialize it. Another thing I learned is how to implement the design by Contract technique in a project by creating an interface and adding pre and post conditions to it. Also a new thing I learned was how to implement an observer and notify all the observers if a certain object has changed.

As a future feature of this application I thought of implementing a way such that the person will be able to decrement the period of their saving account or event extract money from the account only by using a secret pin code like in real life.

1. *Bibliography*

<https://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise>

<http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>

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

<http://stackoverflow.com/>

<https://tutorialspoint.com/>

<https://www.javatpoint.com/>

<https://geeksforgeeks.com/>