Dan Mitrea
Group 30421
Year of study : 2016-2017

Technical University of Cluj Napoca

Programming Techniques
- Assignment 4 -
Design by Contract Programming Techniques, Design Patterns

Contents

Assignment Objective

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.
   Define a method of type “well formed” for the class Bank. 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 to / from a file.

Problem Analysis

Banks are spread all across the world and are responsible for the flow, and also the creation, of money. Without banks, the world would be in chaos, and money will not have the meaning it has today. This being said, banks need a good software that stores such things as accounts, people having these accounts, and money, in an efficient way, because it has to deal with endless people and accounts, and even more transactions each day. Banks need to keep track of all the people from their data base, of all the associated accounts, and also of the money that is being held in each account. So, an application that does exactly this is the thing they need.

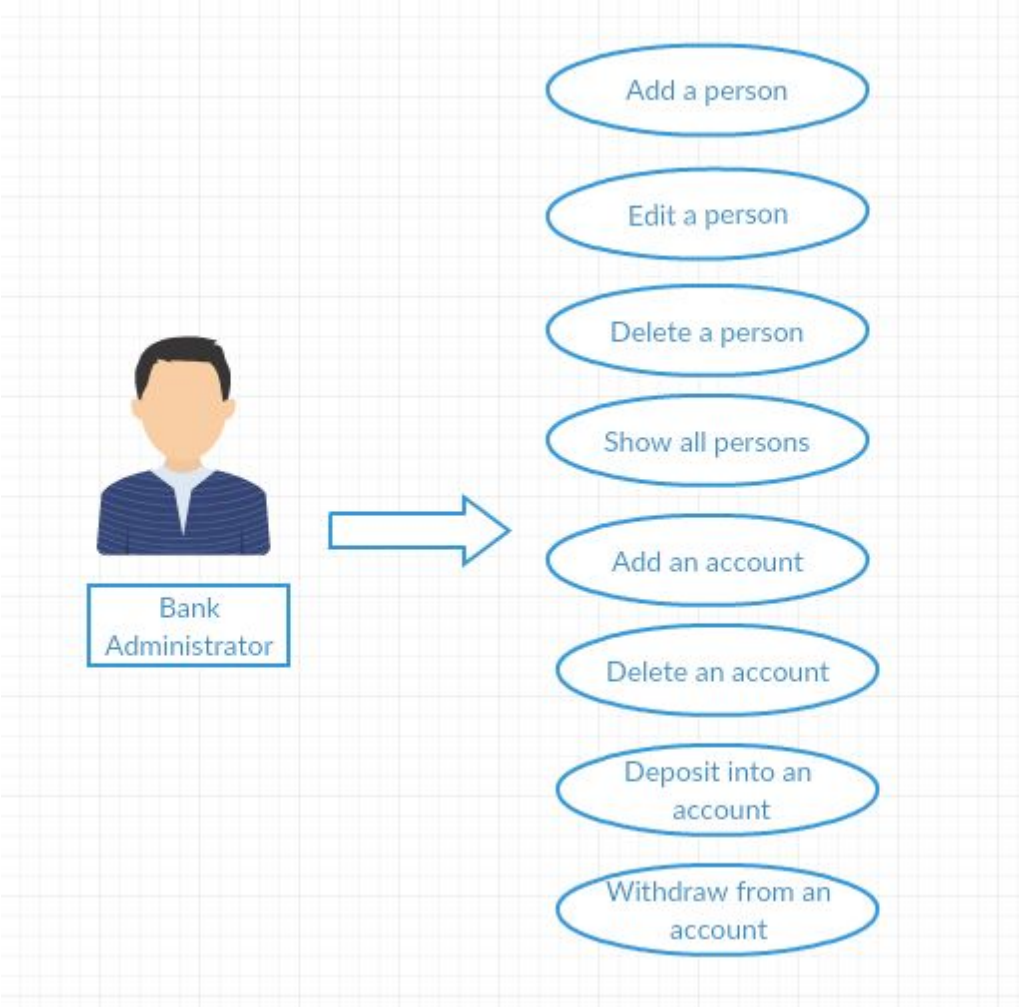
The application can be divided in two parts, the one that has to deal with people, and the one that has to deal with the accounts associated to these people. It supports operations such as: adding a new person or a new

account, deleting a person or an account, editing a person's information, and also depositing of withdrawing money from an account. All these operations are essential for the good flow of a bank's transactions and need to be done in an efficient manner. A good thing to be noted is that the way these operations are done is of the utmost importance, while the graphical user interface does not need to be extremely fancy, or complicated, it just has to be concise and easy to use.

Modelling

In our case, the best way to model our application is by using a HashMap to store our persons and their associated accounts. The reason for which I have chosen this data structure, that hash map, is that it stores elements extremely efficiently, and also, for each object, it can store a list of associated objects to that particular one. In our case, each stored person can have any associated accounts. Its main feature that differentiates it from other data structures is that it can store one null key and any number of null values, and that the iteration of values is done using a fail fast iterator. The way it stores values is by a hash table with buckets. Also, because a person can have a number of different types of accounts, the application uses a class " Account " which is being extended by all the types of accounts a user can have, so that we do not have to be redundant with our methods that are the same for all accounts.

Use cases



Scenarios : user creates an account and adds money into it

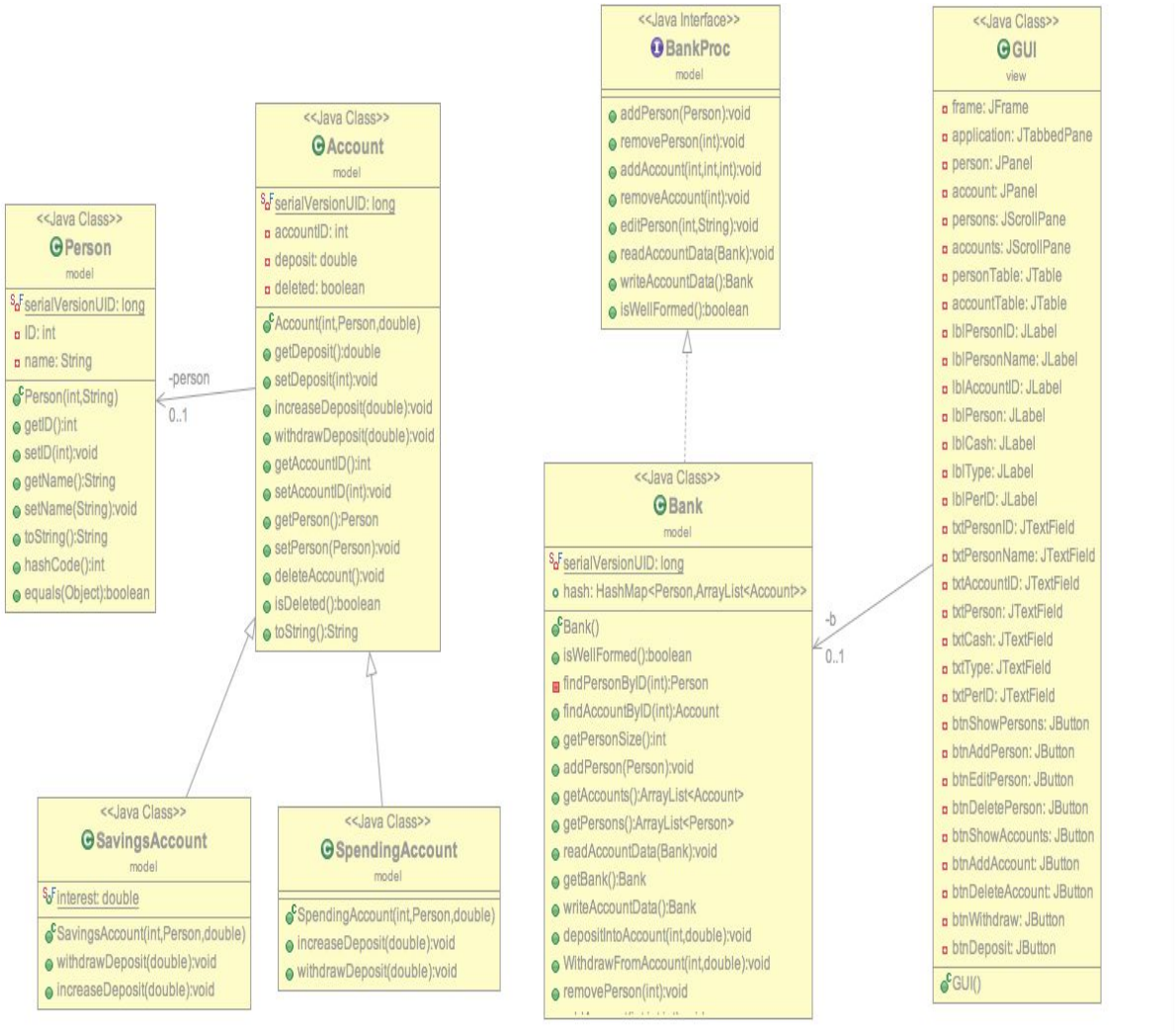
1. The user starts the application.
2. The user checks the person panel, in order to see the id of the person that will be the holder of the account.
3. Then, he / she goes to the account panel, typing in the account id, the account type (0 for a saving account, 1 for a spending account) and the person id who will own the account.
4. The user presses the “ADD ACCOUNT” button.
5. If the required fields of the operation are not filled in, there will be a pop up message telling the user that the input data is wrong.
6. The account has been created, the user can see all the accounts by pressing the “SHOW ALL ACCOUNTS” button.
7. Then, the user can type in an amount of money he / she wants to deposit into the account.

8. By pressing the “SHOW ALL ACCOUNTS” button again, the user will see the updated money field.

Error case:

If the required fields of the operation are not filled in, there will be a pop up message telling the user that the input data is wrong. Every operation has such a feature, that if the required fields are not filled in, a pop up message will appear, informing the user what kind of operation does not have the required fields.

Design



Packages:

I have used the standard MVC design pattern (Model - View - Controller). The packages have the role of separating relevant data and to better encapsulate information. Their design is as follows:

1. The classes contained by the “model” package have the role to model a real object and its behaviours into our application.
2. The class contained by the “view” deals with the graphical user interface, or simpler said, manages what the user can see and do with the application.
3. The class contained by the “controller” controls the data flow of the model classes and updates the view whenever necessary.

Classes:

Person, Account, Savings Account and Spending Account:

These classes have the role to model a person, an account and the specific types of accounts a person can have. Well, not everything they do, but only what is of interest for our application. The “Person” class contains the following attributes: a person’s ID and his / her name. What is more interesting is that both “SavingsAccount” and “SpendingAccount” extend the class “Account”. They inherit all the methods from the super class as they are, the only difference being the way the user can deposit or withdraw money from the accounts. When we are talking about a savings account, the user can only withdraw the whole money stored into it. When the user wants to deposit money into such an account, a 10% interest rate will be perceived of the deposited sum. Regarding the spending account, depositing and withdrawing money from such an account is done simply, with no interest rates or anything else special.

BankProc interface:

I have designed an interface called “BankProc”, which contains all the methods from the class Bank. What is to be noted about this interface is that I mention all the pre and post conditions of each method, since this application uses the Design by Contract technique. Also, the application contains

methods to serialize and deserialize the information from the bank into a file, but these methods will be described shortly.

Bank class:

This is the class that has all the methods of the bank. It has the public methods that the application requests, and also some private methods, having the purpose to aid in the main methods. Every method has been written using the design by contract technique. All the pre and post conditions of each method are verified using the “assert” keyword, which tests if an invariant is correct. This keyword is used to find bugs in the code, since it delivers more information than the exceptions. Also, the method “ isWellFormed() ” represents the invariant of the class, which is, a property that holds true in any instance of that class.

Other methods to be mentioned are the ones that the application uses in order to serialize the information it holds in / from a file.

```
public Bank writeAccountData() { // deserialize
    // This method retrieves the next Object out of the stream and
    // deserializes it.
    // The return value is Object, so you will need to cast it to its
    // appropriate data type.
    try {
        assert (isWellFormed());
        FileInputStream fileIn = new FileInputStream("bank.ser");
        ObjectInputStream in = new ObjectInputStream(fileIn);
        Bank b = (Bank) in.readObject();
        in.close();
        fileIn.close();
        return b;
    } catch (IOException e) {
        e.printStackTrace();
        return new Bank();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
        return new Bank();
    }
}
```

This is the method I used to take the information from the file and load it to the application when it starts. I call it every time I want to display the persons or the accounts, so that the displayed data is as accurate and up to date. This method retrieves the next Object out of the stream and deserializes it. It is

important to be noted that the return value is of the type Object, so that you will need to cast it to the appropiate data type before returning it.

GUI class:

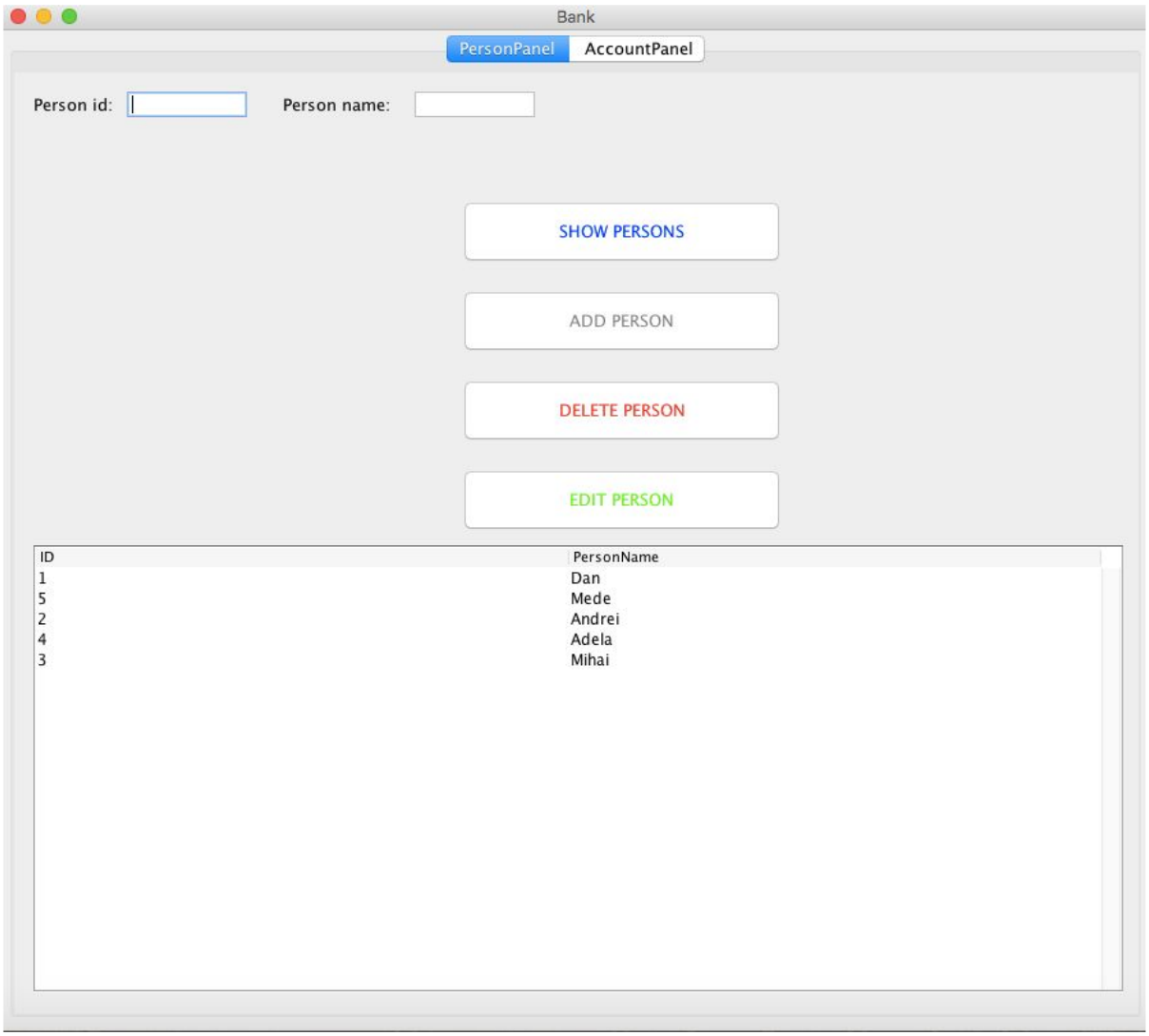
This is the class that implements the graphical user interface. It has the purpose to make it easy for the user to perform the required operations on the information that is held in the bank. Also, it must help the user to his job as easily as possible, and in a friendly way. It might not have the best design, nor the most fancy one, but it gets the job done. Every text field has its own label, telling the user what information should be typed in. Also, the buttons are big and have intuitive names so that the user knows what to press when he / she wants to perform a certain action. It consists of two tabbed panes, one for the operations on persons, and one for the operations on accounts. Every pane, in its second half, the bottom one, has a scroll pane build on a JTable. After creating the names of the columns by the information I want to show, the application iterates through the list of persons and accounts from the bank and displayes the required information in its own specific row and column.

This scroll pane has the purpose to display the required information in a pleasant way. Also, I have chosen this data type so that, if the bank holds a lot of information, i can be displayed nicely, by scrolling down.

Main class:

This is the main class, the one that contains the “ public static void main(String [] args) ” method. Since all the operations are done in the graphical user interface class, this main method just creates a new object of type “ GUI “. Everything is done in the contructor of the class “ GUI ”, so, as a result, just creating an object of this kind is enough for the application to run as it should.

User Interface:



As I have previusly mentioned, the graphical user interface does not have the best design, but it gets the job done. It is intuitive for the user and tell him enough in order for him / her to use the application with no user manual or something similar.

The interface has 2 different panes, having the purpose of differentiating between the operations done on persons and on accounts. For each operation, a pop up window will apear if the input data is not correct (which in our case, can happen only if the user does not type anything in the text fields required for a specific operation).

Since the information is loaded and saved from / to a file, even if the user closes the window, when reopening it, it will contain the same data as before. After every operation, if the user wants to see the actual result, he / she can press the “ SHOW ALL ACCOUNTS ” or “ SHOW PERSONS ” button and see that the information has been modified.

Implementation and testing

For the implementation, I first read the documentation provided on the data structures that we can use for storing the information in the bank in an efficient manned. Also, I watched tutorials and read on the internet as much as I could about the programming techniques we did not know before, used in implementing this specific application. After making sure I understood the concepts, I began writing the easier classes, like “ Person ” and the account classes. Then, I designed the “ Bank ” class and the interface it implements based on what I have learned. It was not easy, but finally i got used to using these new techniques and I made sure everything was ok by creating some objects in the main class. After this, I proceeded with the implementation of the GUI class.

For testing, I have used the design by contract technique and also JUnit testing. For each method in the bank class, I verify the pre, post conditions, and also the invariant, using the assert keyword. Also, for each method in the bank, I created a JUnit test, which has the role of verifying again these pre and post conditions.

Results

The result is, after many hours of coding and debugging, an application that has a great applicability in our society, and also an application that uses some programming techniques that are very helpful and important, that I did not know before. Even though the application has some flaws, I believe that with some minor improvements, it could be used in real life.

Conclusions

What did I learn

After finalizing this application, I can sincerely say that I have learned quite a lot. Besides refining my skill in writing java code, which is a skill I will always want to improve, the main thing that I have learned is these new programming techniques. I learned to use the design by contract technique and to test every method that I implement by some pre and post conditions. I have also learned what an invariant is and how it can be helpful to have such a method in my application. What is more, I had the chance to use the JUnit testing again and become better at using them in my applications.

I am sure that I can use what I learned after writing this application in many future projects, and that I am one step closer to my goal, which is to become the best programmer I can be.

Further possibilities for improvement

Although the application does all the basic requirements it should do, it still has some areas that can be improved. First of all, I have not use the Observer design pattern that I could have implemented in order to notify the account houlder about any change in one of his / hers accounts. Also, the methods in bank could be more complex and take into account some more parameters. What is more, the GUI could have a better design, and maybe more fields for each type of data that the bank holds, persons or accounts.

Bibliography

1. http://stackoverflow.com
2. https://wordcounter.net
3. https://creately.com
4. http://stackoverflow.com/questions/2758224/what-does-the-java-assert-keyword-do-and-when-should-it-be-used