Bank Processor

Vlad-Cristian Buda 30423

Table of Contents (no numbering)

1. Objective
2. Analysis of the problem
   1. Assumptions
   2. Modeling
   3. Use cases
   4. Errors
3. Design
4. Implementation
   1. Business Layer
   2. Model Layer
   3. Presentation Layer
5. Testing
6. Results
7. Conclusions
8. Biobliography

1. Objective

The main objective of this project is to design and develop a Java program specialized in processing customers and accounts for a bank.

In order to obtain this functionality, the main problem can be decomposed into several steps needed to reach the goal: obtain a proper representation for each type of object being manipulated (person and account); correctly design the methods used for performing different actions involving the objects we represent (add a new person, delete an existing one, add a new bank account associated to a person etc.); design a friendly and intuitive graphical user interface for the user to easily take advantage of the functions provided by the program.

2. Analysis of the problem

The solution of the problem should be able to correctly process bank related operations and keep a record of all persons and their respective accounts. In order to obtain this functionality, several operation must be implemented: insert a new person/account; delete an existing person/account; update the information related to an existing person/account. First of all, in order to do that, a correct representation of the real life objects must be proposed. The fact that the user is responsible with populating the database requires some assumptions to be made.

a) Assumptions

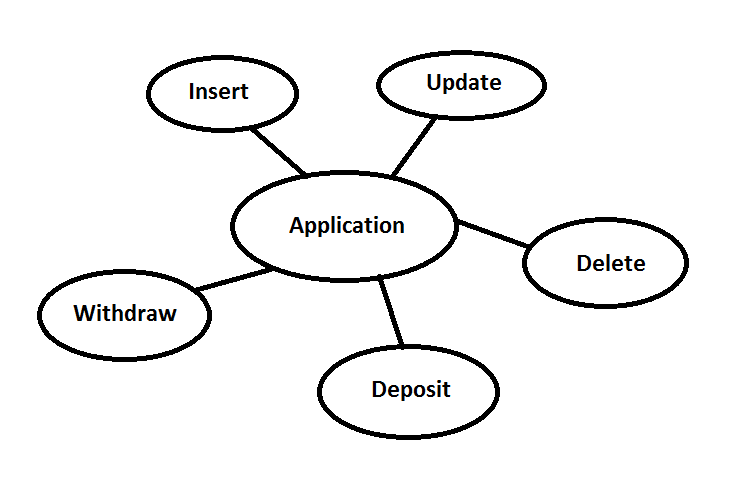
The assumptions that should be made regarding this problem are stricly related to the corectness of the information the user enters. The input for each selected operation should obey the preconditions imoposed by each method implementing the functionality. Once the preconditions are met, the postconditions assures the user the desired outcome. In the case one precondition is not fulfilled, the user will be notified and a proper explanation of the error will be offered.

b) Modeling

As mentioned above, an important aspect in finding a solution is to correctly represent the data. In the case of our application, 2 real life object should be represented. The first one is the client, the person of which one or many accounts are associated to. Each client is uniquely identified in the bank’s database by its CNP. The other relevant information for representing a client is its name and address. The next one is the bank account, the instance used for depositing or withdrawing money. Similar to a client, each account is uniquely identified by its IBAN. In this case each account can be of 2 types: Saving Account and Spending Account. The difference between those two types is the following: the Saving Account allows only one big deposit and withdraw, offering an interest while the Spending Accounts supports unlimited number of deposits and withdraws with no limit imposed. Each time a person or an account is added, or a withdraw/deposit operation si performed, the information being displayed is updated.

c) Use cases

The use-cases are represented by the functions the program offers: the capability of managing a bank’s database with clients and associated accounts. All the available operations, i.e. use cases possible are shown in the following diagram:



For each of the above mentioned use-cases, a button on the graphical user interface will be present. In order for the user to perform an operation, he will first enter the data having the correct syntax and then will push the button corresponding to the desired use case. Depending on the chosen operation, the results will be immediately displayed on the graphical user interface or the user will be notified in case something went wrong. The steps described above are the same for each use-case.

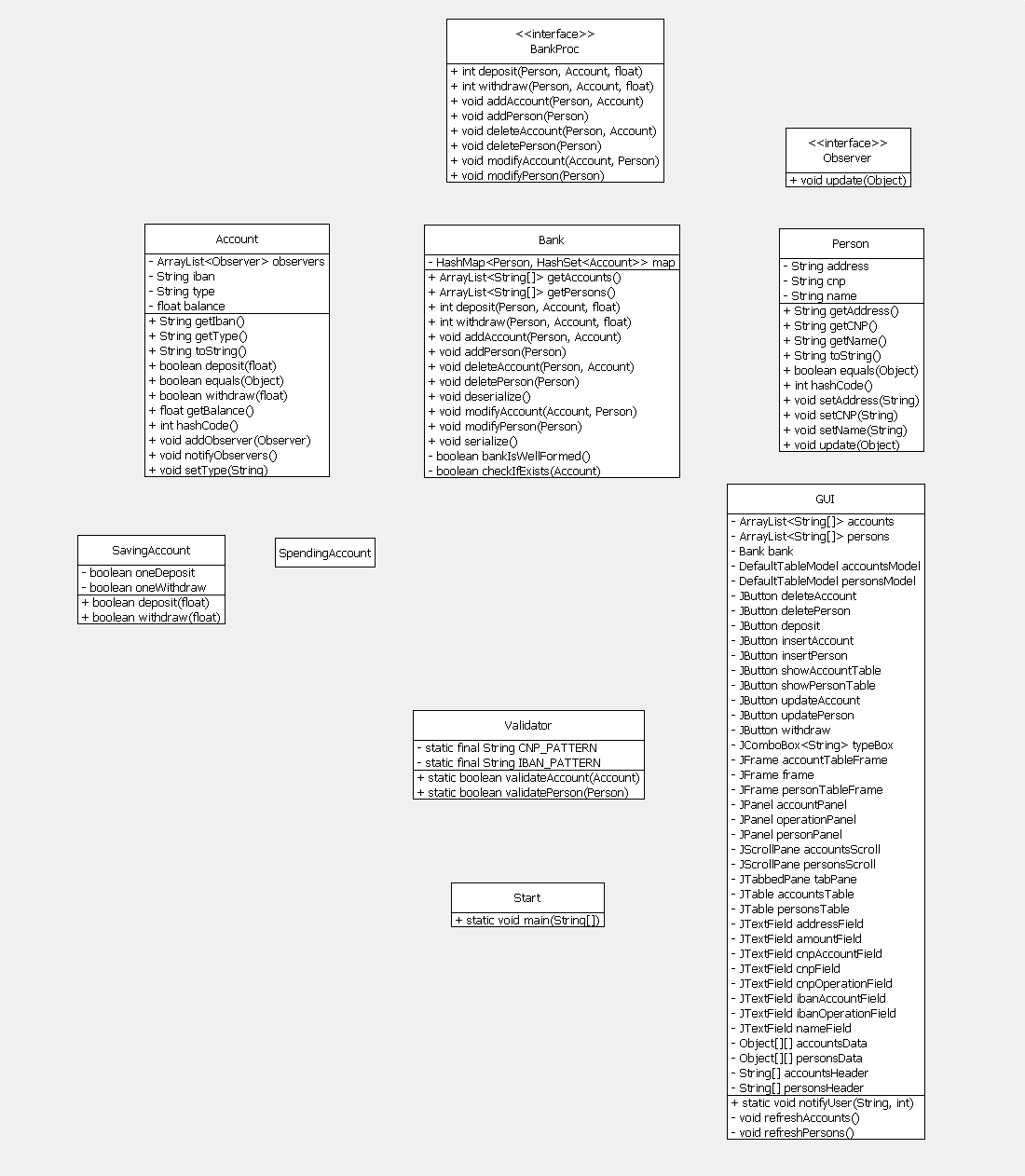
d) Errors

The corectness of the program guarantees no errors in terms of computing the result if the input is correctly entered. Even though the input entered may have a wrong format, the user will be immediately prompted and the program should not encounter any errors. At this point, it is safe to say that all the exceptions that might appear due to invalid input are addressed.

3. Design

The design of the application is made in an object oriented manner. As mentioned earlier, the classes corresponding to the modelled object contains the relevant information for that object.

The following UML diagram shows the entire structure of the program along with all the classes, interfaces and relationships between them.



There are 4 packages containing the main classes: Bank package, Model package, Validator package and GUI package. Business package contains the main logic of the program and is represented by the class Manager. Bank package contains class Bank which implements the BankProc interface where all the operations a bank must perform are declared. Class Bank implements methods for adding new persons, add accounts associated to a person and perfom specific operations such as deposit or withdraw. Model package contains the data being modeled, i.e., Account and Person. Class Validator contains 2 static methods used by the Bank class to check the preconditions of each method. Class GUI contains the graphical user interface for the entire application.

When it comes to data representation, class Bank represent its inner data as an HashMap where each person is a key mapping to a HashSet of accounts associated to him. A HashMap and generally a HashTable offers some obvious advantages over a classic data structure. Adding, finding and removing an element is efficient, taking constant time to perform. Also HashMap offers a natural way of associating a data structure for storing accounts to each added person. HashSet also has the same efficiency as HashMap and implements the mathematical set property (all elements are unique).

All the main classes will be explained in detail in the following section along with their implementation, main methods and also the graphical user interface.

4. Implementation

We already mentioned above the main packages and their corresponding methods. Each class implementation will be thoroughly explained next.

The program is structured in 4 packages: Model, Validator, Bank and GUI.

Model package contains the classes which represents the modeled data. Class Person represents an entity inside the data structure of a bank which can have have multiple accounts. A Person is uniquely identified by his CNP. Other relevant information being stored is the name and address. Besides getters and setter, class Person also overrides methods hashCode() and equals(). Each person acts as a key in the bank’s database, leading to a set of accounts. The value of the key is computed based on the hashcode of the CNP, which uniquely identifies one person. Method equals defines the equality between 2 objects of type Person: they need to have the same CNP. This class also implements the Observer interface and defines the method update() used to notify each person when their accounts has been modified. Class Accounts represents a bank account. The Accounts class is divided in 2 different account types: Spending and Saving account. Each type is represented by its own class which extends Account. The difference between the two is the following: Saving account allows only one big deposit and withdraw with an interest offered while the Spending account allows unlimited number of deposits and withdraws with no other restrictions applied. Each method overrides methods deposit() and withdraw() from Account to implement their specific functionality. Besides that, class Account also overrides methods hashCode() and equals() in order to properly store an account in the HashSet associated to each person. The hash code for an account is computed based on its IBAN, the unique identifier for an account. Besides that, each account holds a list of observers and notifies them when an operation was performed on their account (deposit, withdraw, update etc.).

Both classes, Person and Account, implement interface Serializable to save the content of the HashMap in a binary file.

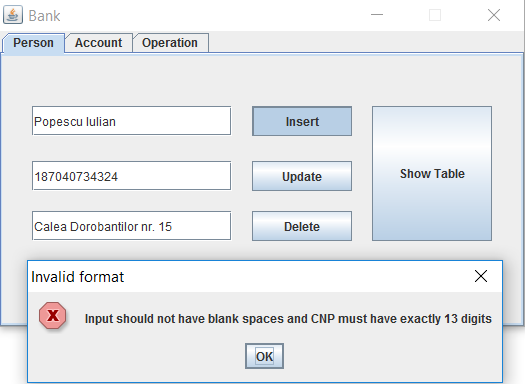
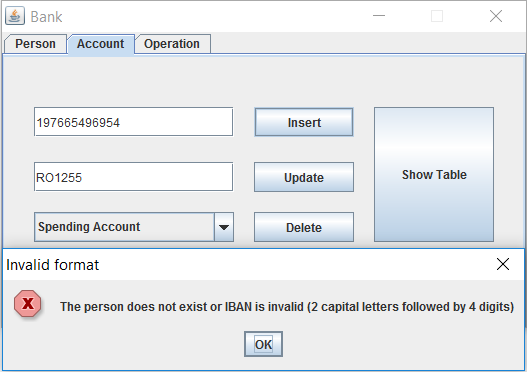
Validator package contains the class Validator used by the bank to validate the input. This class checks a part of preconditions in order for a new person to be added to the database or an account to be created. Method validatePerson(Person) receives an object of type Person and checks whether its fields are empty or the CNP respects the pattern (13 digits). If a field is empty or the CNP does not have the requested form, the method returns false. The other method, validateAccount(Account), validates the input for a new account. The only thing checked is whether the IBAN respects the syntax or not (2 capital letters and 4 digits). Method returns false if the account has invalid syntax.

Bank package implements the bank structure along with the methods defined in BankProc interface. As already mentioned, the data inside the bank is hold in a HashMap where each person acts as a key mapping to a HashSet of accounts. The methods defined by the BankProc interface are the following: addPerson, deletePerson, modifyPerson, addAccount, deleteAccount, modifyAccount, deposit and withdraw. As we can see, these methods are common for a bank. Each method checks the preconditions specified in the interface and if they are met, the postconditions assures the correct functionality of the methods. If the preconditions are not met, each method throws an AssertionError and the caught exception will specify each precondition was invalidated.

Besides this methods which implements the functionality of a bank, this class also contains methods serialize() and deserialize(). Serialize method is used to save the HashMap in a binary file. Deserialize method does the opposite. It reads the HashMap from the specified file and constructs the object saved in the file by serialize method. Methoda bankIsWellFormed() represent the invariant of the Bank class. Each time an operation is performed, this method is called to check that the balance of each account is greater than 0.

GUI package contains the class which implements the graphical user interface for this application. The single frame holds the panels for performing operations on persons, accounts and making deposits and withdrawals. Furthermore, each person and account are separately held in a JTable and can be easily accessed by pressing the corresponding button on the graphical interface. Each JTable has click events applied on them in order to easily access the complete data related to each instance in the table. Type errors are addressed inside this class and the user will be prompted by a message when the sytanx of the input is invalid. Each button handler catches the AssertionError exceptions that may be throws by the Bank class and prompts the user about the error. Some of the warning messages that may appear due to invalid input are shown below.

When the application first launches, the deserialize method from Bank is called, loading all the objects from the file. When the user closes the app, method serialize is automatically called and the data from that session is saved on the disk.

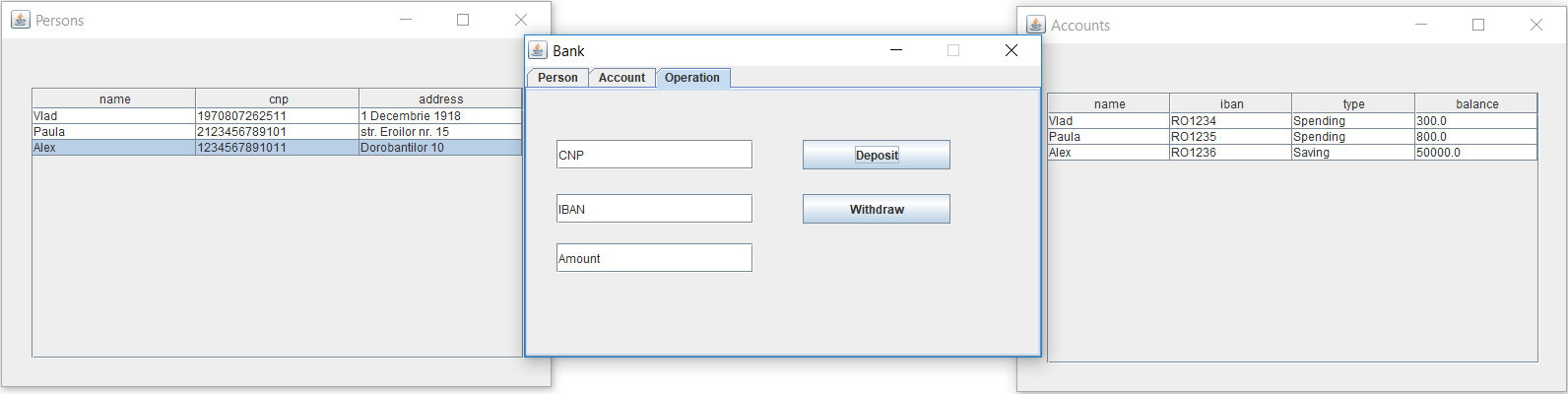
 

5. Testing

JUnit tests were conducting in order to test the functionality of each method which performs an operation on the bank’s database content (add person, remove person, add account, remove account, deposit, withdraw).

6. Results

After all the necessary steps (design, implementation, testing) we managed to obtain a fully functional program for processing orders for a warehouse. Also, besides the main functionality, a simple and intuitive graphical user interface was created to ease the access to the program’s functions. The following screenshots present the application in various use cases:



7. Conclusion

During the design and implementation of this project I was able to learn new Java concepts especially the Design by Constract pattern, which ensures the functionality of a method by establishing preconditions for the user. Besides this, thorough this project I also had the oportunity of getting an hands on experience with Java serialization mechanism for storing objects on the disk and also a new pattern, called Java Observer, used to notify users in case the state of an object has been modified.

For this project, as future improvements I suggest the addition of new functionality and new bank account types, such as credit ones, where the customer can loan money from the bank and also the abbility of paying a debt monthly with an interest added.

8. Biobliography

<https://stackoverflow.com>

http://coned.utcluj.ro/~salomie/PT\_Lic/4\_Lab/HW4\_Tema4/HW4\_Tema4\_Hashing.pdf

<https://www.draw.io/> (for drawing the UML diagram)

Java: How to Program, 9th Edition (Deitel)