Technical University of Cluj-Napoca

Bank

-project documentation-

Pop Andrei

Group 30424

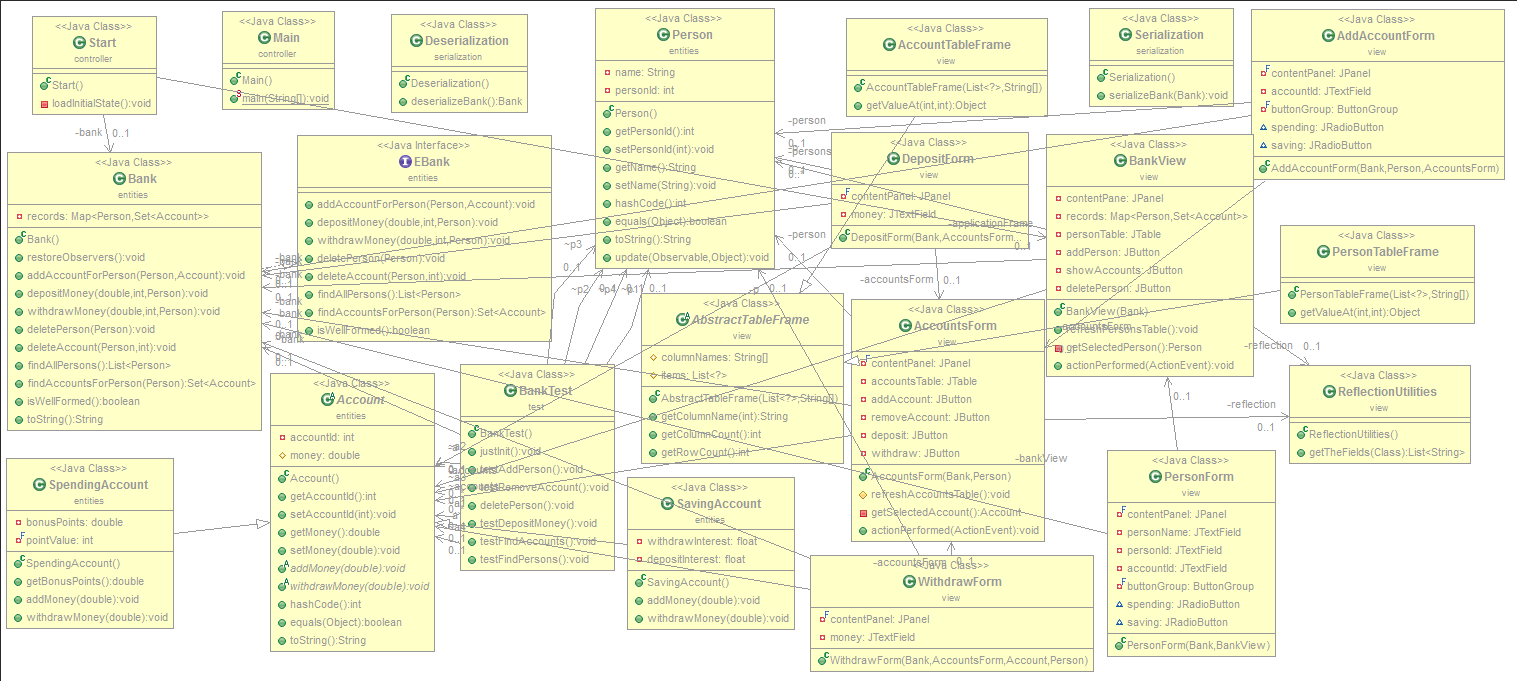
1. Objective

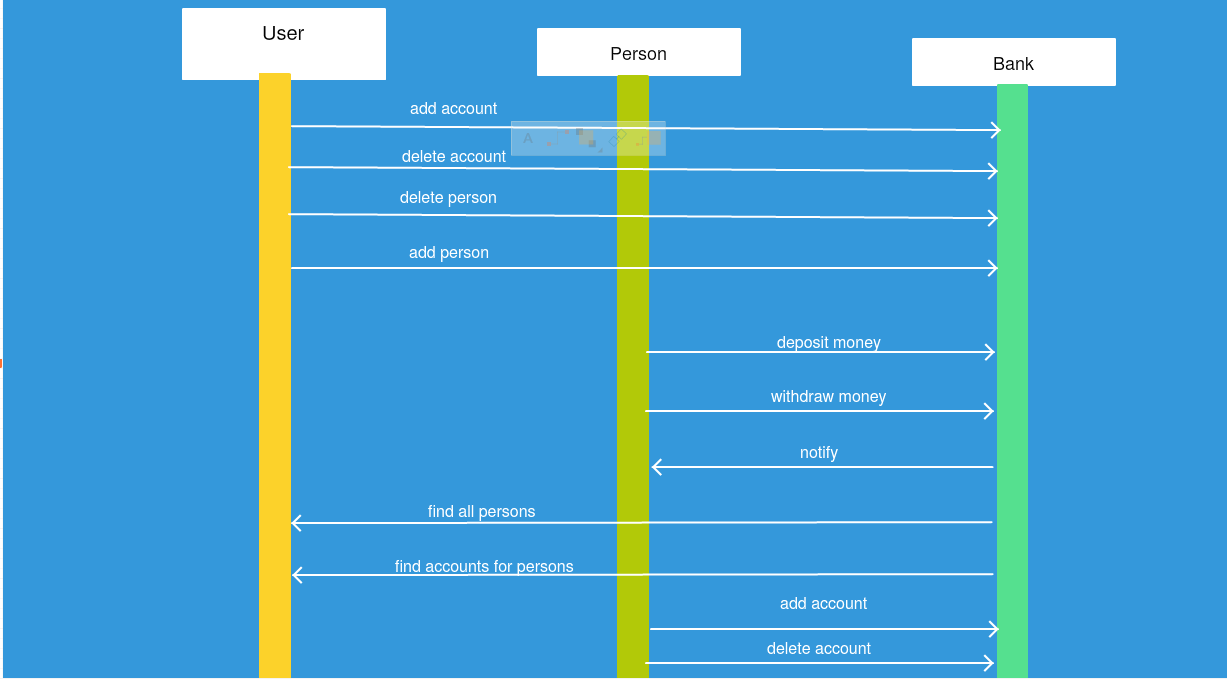
The objective of this project was to implement a bank simulation. Besides this objective, probably the main objective was to learn how to use the Observer design pattern. The Observer pattern is used when there are multiple objects which are independent of each other, and they all need to be updated when the status of a common objects (the subject is changing). In order to do this, implemented classes are used. Each object that needs to be notified when a change occurs in the subject have to extend the class Observer. The Object that holds the status, which is a subject of interest for all the other objects, has to extend the class Observable. When something changes in the status of the subject, in our case the bank, all the persons which are associated with the modified account are notified.

1. Problem analysis, modeling, scenarios, use cases

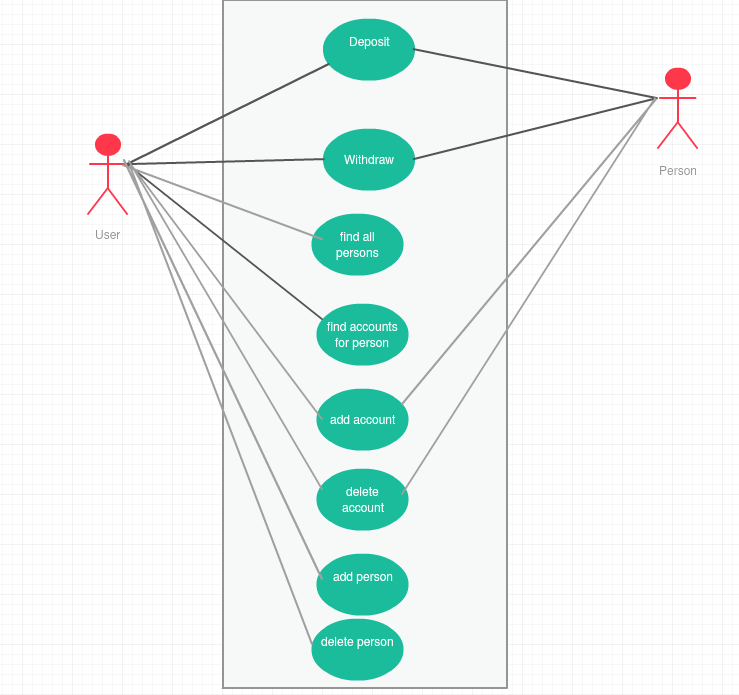
In order to implement a bank simulation, I have created multiple entities which correspond with real life objects. Each person can do certain actions like deposit, withdraw, delete account, add account. Also, there are more options like creating a new person, delete an existing person, find all persons, find all accounts for a person…. This functions can be accesses only by the admins. Each person has an unique identifier and a name as in real world. Each person can own multiple accounts and each person is notified when the status of the account is changed by depositing or withdrawing money. In the bank class, all the information about the persons and the associated accounts is stored. The account can be of two types: spending account and saving account. The spending account class was implemented such that each time an operation is done, there’s no interest. Each operation performed on the saving account involves a small interest imposed by the bank. In order to save the data from the bank between different and independent execution of the application, the serialization was used. Each time the application is closed, the information from the bank class is serialized and saved into a file. When the application is started the next time, the old status is loaded from the file. All the operation are done using a user interface. The information about the bank status is displayed in JTables. In order to reuse the code, the JTables were generated using reflection. This means that the field were obtained directly from the objects displayed in the JTable and not hardcoded in the class. The application was designed only for the administrators. Each of the operation can be performed by any user. In order to keep a consistent bank status, after each operation, the status of the bank is checked in order to see if a valid operation was performed. Also, at the beginning and at the end of each function that was implemented in the bank class there are some constraints that must be met in order to continue the execution of the application. If one of the conditions from the beginning or from the end isn’t met, then an exception is thrown.

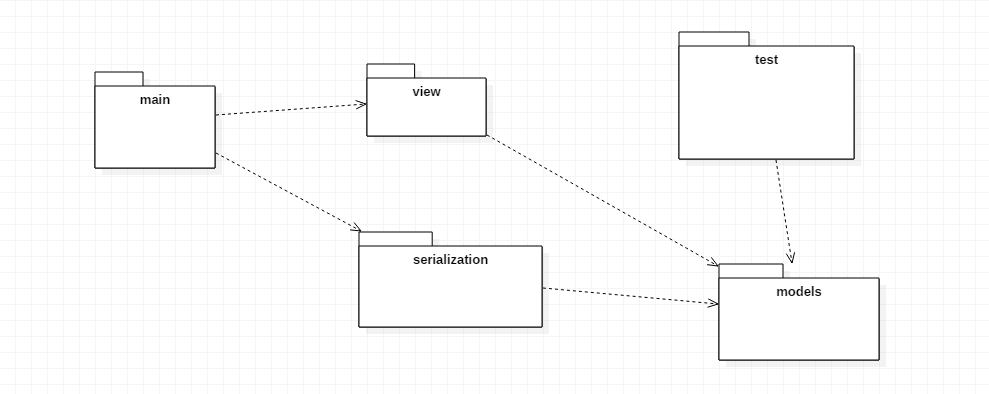
1. Design

3.1 UML Diagram

3.2 Sequence Diagram

3.3 Use Case Diagram

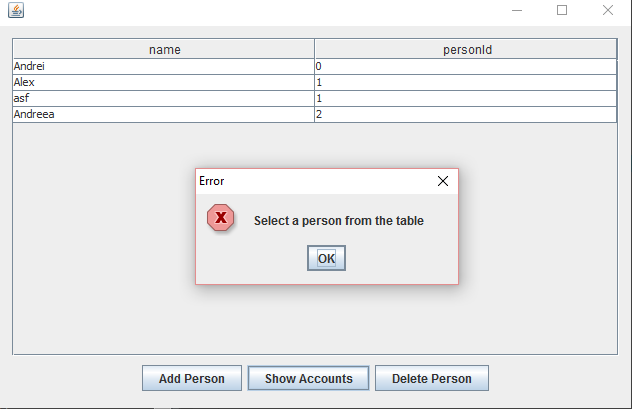


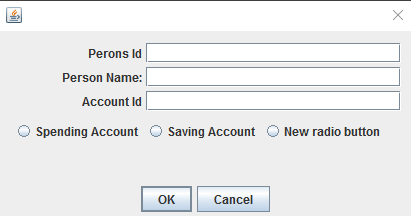
3.4 Use Case Diagram

3.5 Class design

The application is composed of 21 classes and 5 packages. Each of the packages contains classes which fulfill a certain requirement. The main packages of the application are:

* **Main package**: this package was designed in order to launch the application. When the application is first started, the class Start is instantiated.
* **Models package**: this package contains all the models from the application. The models are: Account, Bank, Person, SavingAccount and SpendingAccount.
* Account class: contains all the details about an account which is associated to a person. The account contains an id and another attribute, money. This is the abstract class which is extended by both: SpendingAccount class and the SavingAccount class.
* Person class: holds information about each of the bank’s customer. Each Person has a unique identifier and a name. Each time an operation is performed into an associated account, the person which is associated with the account is notified.
* EBank: is an interface which specifies the behavior of the Bank class.
* **Serialization package**: this package contains specific classes which perform the serialization and deserialization. The serialization is done each time the application is closed, by the Serialization class. The deserialization is done by another specialized class called deserialization.
* **View package**: this package contains all the classes which are part of the graphical user interface. It contains the JTables which display the persons and their associated accounts. In order to add a new account, each time an add operation is performed, a new form is opened, which will be completed by the user in order to create a new account for a certain person. All the possible operations that are implemented by the bank can be simulated due to this package. In fact, the view package is offering the possibility of visually simulating the code.
* **The test package**: this package contains only the test class. This class was created in order to test if each of the operations was correctly performed and has brought the bank from one valid status to another valid status. In the testing class, for each of operation from the bank class, a testing method was implemented. The role of the testing methods was to check if the bank status would still be valid between operations.

3.6 Graphical user interface

The graphical user interface was created in order to offer the possibility of simulating the bank operations. Each of the operations can be executed by pressing a button. The information is stored mainly in two JTables. One JTable is representing the persons and the other one the associated accounts. In order to add a new person into the bank database or to add a new account for an already existing person, special forms are used. The forms fields are completed by the users and the saved into the bank object. After each operation, the status of the bank is checked in order to see if it is still valid after the performed operation. The graphical user interface wasn’t created in a professional way because this wasn’t the purpose of the application development. The graphical user interface is just a way of simulating the implemented operation that can be performed by the bank. Also, it provides a real time interaction with the user.

1. Future improvements

The application was designed only to exemplify how the Observer design pattern can be used an how to adapt it to a specific context. In order to have a real world application, more classes need to be added. Also another problem is that there isn’t a differentiation between the users and the administrators or the employees of the bank. An improvement that can be achieved would be to restrict the ordinary users to a certain set of actions that they can perform in the application. Another improvement would be to create multiple threads that can work concurrently. In this way, more persons can perform operations on their associated accounts without influencing each other and having the application running at the same speed and offering the same performance. Also, in order to be used in the real world, the application would need another design. The design is one of the most important aspects when designing an application which will be available on the market because the non-specialized user will evaluate the application depending on how it looks and not how it works. Another improvement would be the run the application on a web platform such that it wouldn’t be accessible only locally. In order to protect the unauthorized access in the application, another future development of the application would be to offer accounts with different access options. In order to do this only a few administrators would have full rights. The other users would have regular accounts which can be used whenever they want to perform an operation.

5.What I have learned -- conclusions

During the implementation of this application I have learned how to use and how to integrate the Observer design pattern. I have also learned that most of the design patterns are already implemented in Java so I don’t have to implement them when I’m using them. Another reason to use the already implemented classes is that they have been tested many times before they were accessible to the Java programmers. During the development I have also learned how to use reflection in order to generate the JTables and also how to store information into the tables. I think that this was one of the most challenging project that I have developed and I have really learned a lot of things.

6.Bibliography

Books:

- Joshua Bloch, Effective Java (2nd Edition);

- Kathy Sierra, Bert Bates, Head First Java (2nd Edition), O'Reilly Media;

- Barry Burd, Java for Dummies (5th Edition), Wiley;

Websites:

- http:// stackoverflow.com/

- https:// www.oracle.com/java/

-http:// www.oracle.com/technetwork/articles/javase/index-142890.html

- http:// www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/

- http:// christoph-burmeister.eu/?p=1556

- http:// tutorials.jenkov.com /