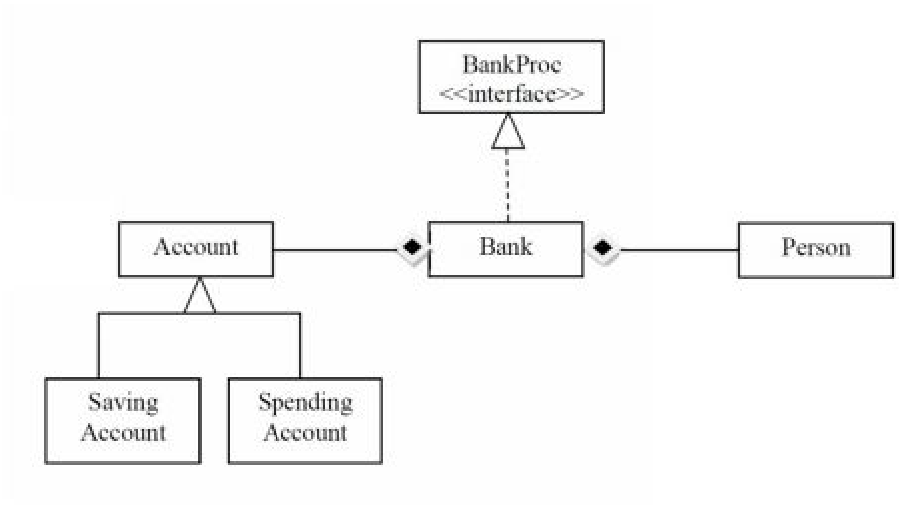
Programming Techniques
HOMEWORK 4

Student: Tudorica Andrei
Group: 30421

0. Contents

1. Objective of the homework

Design by Contract Programming Techniques, Design Patterns
Consider the system of classes in the class diagram below.



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.

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).

5. Design and implement a test driver for the system.

6. The account data for populating the Bank object will be loaded/saved from/to a file.

2. Problem Analysis

In our times the bankar system has the biggest influence in the management of economy, from the big level, in what concerns companies or even countries, to the small level of personal money management, using accounts. This application covers the basic level of account management, making it easier for an account administrator to manage his clients and their accounts. This system makes it way faster and safer to work with the accounts, and minimizing the danger of losing money by mistake.

3. Modelling

The process of modelling is defined as the process of modularising a big problem into smaller ones, that are easier to understand and debug. This also helps making an abstract idea more clear. In software development, modelling is essential in order to build an application that has a strong background structure.

The system divides the problem into 3 + 1 parts. One of them manages the clients. One of them manages the accounts. This part uses the downcasting technique to differentiate between the two types of accounts we have, that is saving account and spending accounts. The saving account allows a single large sum deposit and withdrawal and computes an interest during the deposit period. The spending account allows several deposits and withdrawals, but does not compute any interest. The 2 described parts join in the third one, the Bank part. In thes part I use an array list to store the persons and a hash map to store the accounts, as a client can have multiple accounts. The hash key is the id of the person that holds the accounts and for each key the value is an array list of accounts. The Bank part implements the BankProc interface. In this interface the pre conditions (applies for methods, they are conditions that always have to be fulfilled before the method is executed),post conditions (applies for methods, they are conditions that always have to be fulfilled after the method is executed) and the invariant of the data structure (applies for the data structure, this checks if the essential properties of the data structure are valid. It is called an invariant because this condition set never changes) are defined. These elements are the properties of the Design by Contract model. These elements are implemented in the Bank part, to ensure the manager that every operation occurs safely and no errors in the defined rules occur. The “+1” part is the graphical user interface that contains two panels: one for managing persons and one for managing accounts. Window for Person operations: add new Person, edit Person (this uses the click listener in the JTable to select the person to be edited), delete Person (this uses the click listener in the JTable to select the person to be deleted), view all Persons in a table (JTable is used for this procedure), Window for Account operations: add new Account, edit Account (this uses the click listener in the JTable to select the account to be edited), delete Account (this uses the click listener in the JTable to select the account to be deleted), deposit money to an Account (this uses the click listener in the JTable to select the account to which the money should be deposited), withdraw money from an Account (this uses the click listener in the JTable to select the account from which the money should be withdrawn),view all Accounts in a table (JTable is used for this procedure).

4. Use cases

The developed software application is a basic one, therefore it only simulates the bank management in simple condition, without optimizing any operation. The app is useful though for organizing simple bank operations, like, for example, the ones of a small bank, giving the manager the possibility to easily add, edit or delete persons or accounts in the tables he needs to.

This app could be useful , for instance , in a small bank that has a couple of clients. It would have the table of clients that own accounts in the bank. Also there is the table showing the the accounts stored in the bank. Accounts of 2 types, spending accounts and saving accounts, can be assigned to clients, with the possibility of depositing and withdrawing money from them.

4.1. Scenarios

Let's consider an example of how the application works;

IMPORTANT NOTES:

- The sum input text boxes may only contain integer numbers.
- The ID input text boxes are not editable. Their content can be edited by clicking a line in the JTable, and the ID of the selected account or person is automatically written in the ID text box.
- The edit operation can’t be fulfilled without selecting an element from the JTable (getting an ID in the ID input text box)
- The delete operation can’t be fulfilled without selecting an element from the JTable (getting an ID in the ID input text box)
- The deposit operation can’t be fulfilled without selecting an element from the JTable (getting an ID in the ID input text box)
- The withdraw operation can’t be fulfilled without selecting an element from the JTable (getting an ID in the ID input text box)
- The name and surname input text boxes must not be null when adding a new person.
- The Account Holder ID input text box may only contain integer values.

- The Account Holder ID input text box must contain the ID of an existing person in the bank.
- The sum that can be deposited or withdrawn from a Saving Account can only be 1000.
- The interest of all the Saving account is 2.5%.

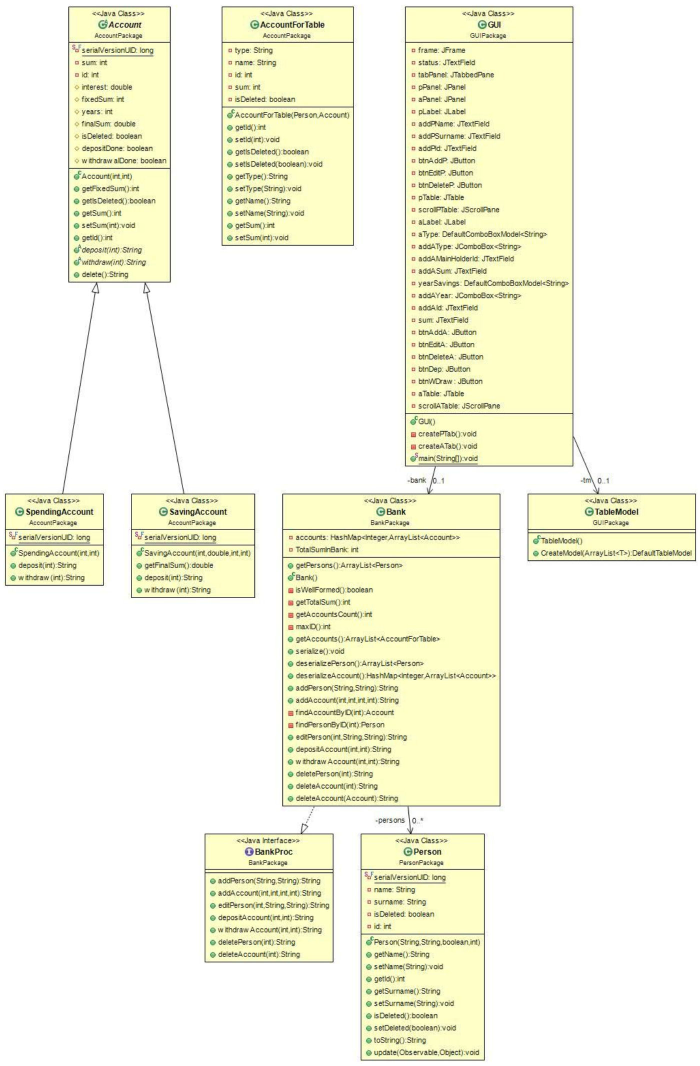
Lets consider a scenario where a user wants to add a client named Pop Cristian, add a spending account for that person with an initial sum of 100, then deposit another 50 and withdraw 30.

Steps:

- The user successfully launches the application.
- The application loads from files the last state of the persons and accounts in the bank.
- In the Person tab, in the lower section the user introduces “Pop” in the name text box.
- In the Person tab, in the lower section the user introduces “Cristian” in the surname text box.
- The user presses the button that had “Add person” written on it.
- The new person is automatically added in the JTable above, with an automatically generated ID (let's say it is 5), that we keep in mind.
- The user switches to the “Account” tab.
- In the Account tab, in the lower section, the user selects the Type of Account as Spending in the dropdown list.
- In the Account tab, in the lower section, the user writes the ID of the main holder (in this case 5) in the “Main Holder ID” text box.
- In the Account tab, in the lower section, the user writes the Initial sum to be deposited in the account (in this case 100) in the “Initial sum” text box.
- The user doesn’t need to select any number of years for the account, because that number is only used for interest computation in saving accounts.
- The user clicks the “Add Account” button.
- The account is automatically generated in the JTable above.
- The user clicks the account in the list.
- The ID of the account is printed in the locked ID field in the lower right side of the window.
- The user writes a sum in the lower left side of the window (in our case 50).
- The user presses the Deposit button.
- In the lower status bar the result of the operation is displayed.
- If the deposit was successful the user is notified. The notification message can be seen in the console of the application.
- The user writes a sum in the lower left side of the window (in our case 30).
- The user presses the Withdraw button.
- In the lower status bar the result of the operation is displayed.

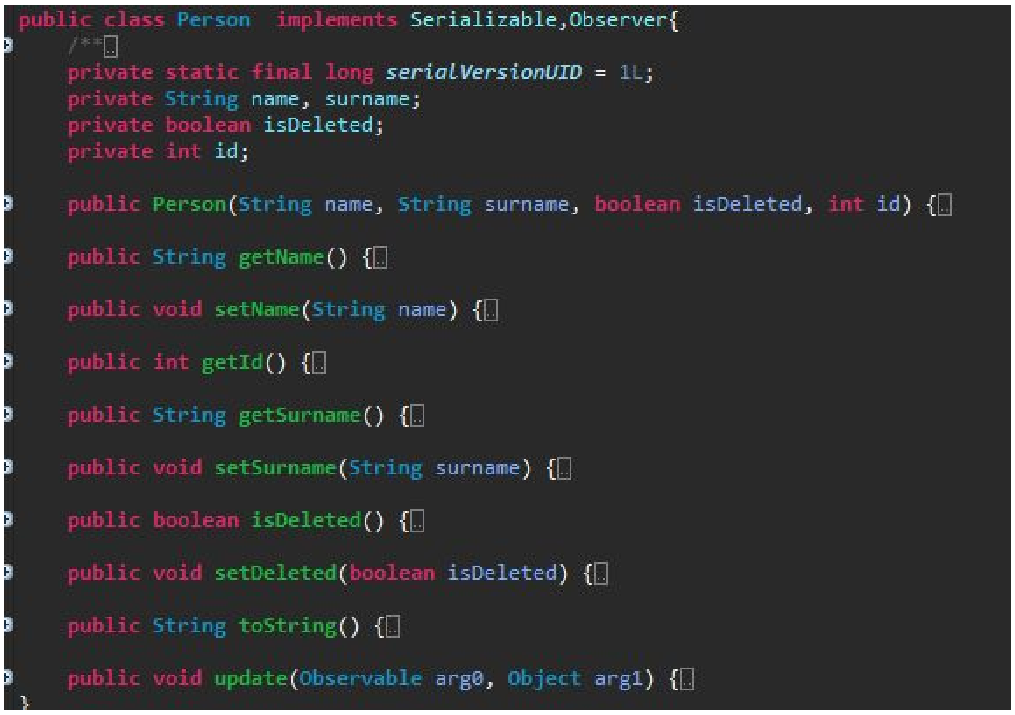
- If the withdrawal was successful the user is notified. The notification message can be seen in the console of the application.
- The user closes the application.
- Before the application is closed, the bank saves into files the actual state of the persons and accounts.

5. Design



5.0. The Person Package

The person package contains only the person class.

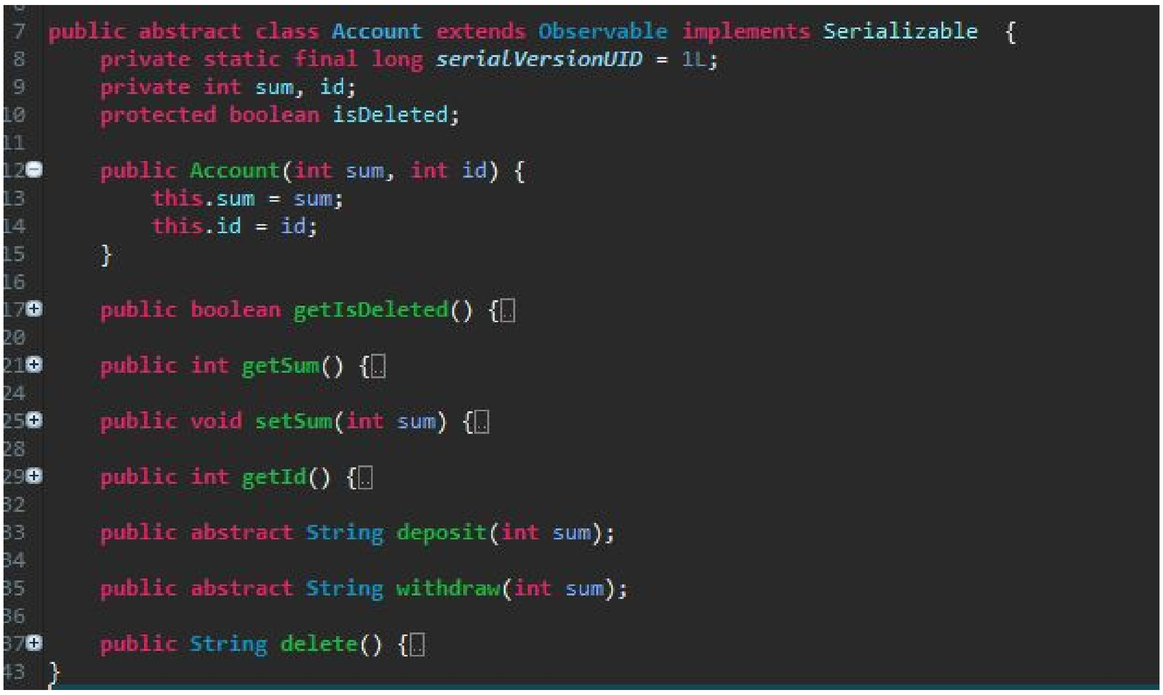


```
public class Person  implements Serializable,Observer{
    /**
    private static final long serialVersionUID = 1L;
    private String name, surname;
    private boolean isDeleted;
    private int id;

    public Person(String name, String surname, boolean isDeleted, int id) {

    public String getName() {

    public void setName(String name) {

    public int getId() {

    public String getSurname() {

    public void setSurname(String surname) {

    public boolean isDeleted() {

    public void setDeleted(boolean isDeleted) {

    public String toString() {

    public void update(Observable arg0, Object arg1) {
}
```

This class contains the definition for a person. Every person has a name, surname, id and a deleted state. The person class has a constructor with all the parameters, getters and setters for all the variables. The class also contains a toString method that returns a printable version of the object, that is “name + surname”. The last method of the class is update. This is a method implied by the fact that the person class implements Observer. In the update method a person is notified when one of it’s account has been modified and a message is printed on the console.

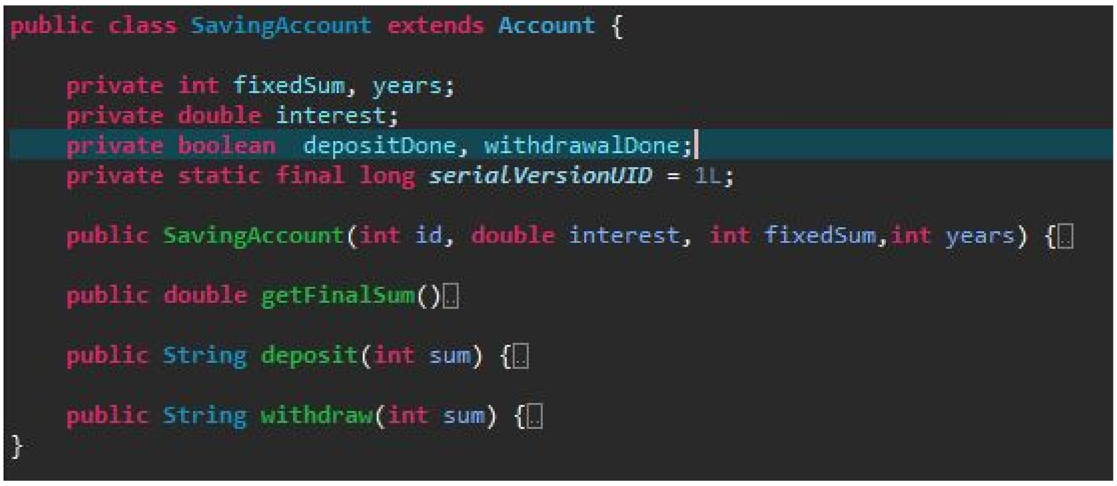
5.1. The Account package

The account package contains the definition of the super class Account, the definition of the subclasses Saving account and Spending account and the definition for the Account for table class that has a role only in the GUI for feeding a table with the relevant information about an account only.



```java
public abstract class Account extends Observable implements Serializable  {
    private static final long serialVersionUID = 1L;
    private int sum, id;
    protected boolean isDeleted;

    public Account(int sum, int id) {
        this.sum = sum;
        this.id = id;
    }

    public boolean getIsDeleted() {

    public int getSum() {

    public void setSum(int sum) {

    public int getId() {

    public abstract String deposit(int sum);

    public abstract String withdraw(int sum);

    public String delete() {
}
```

This class contains the definition for an account. Every account has a sum, ID and a deleted state.The account class has a constructor with sum and id as a parameter, getters and setters for the sum and only getters for the deleted state and the ID of an account. The class also contains two abstract class definitions: one for money withdrawal and one for deposit. The last method in the class is the delete method which changes the deleted state of an account to true, and signals the account holder (the observer) that it has been changed and in what way. This is possible because the Account class extends Observable.

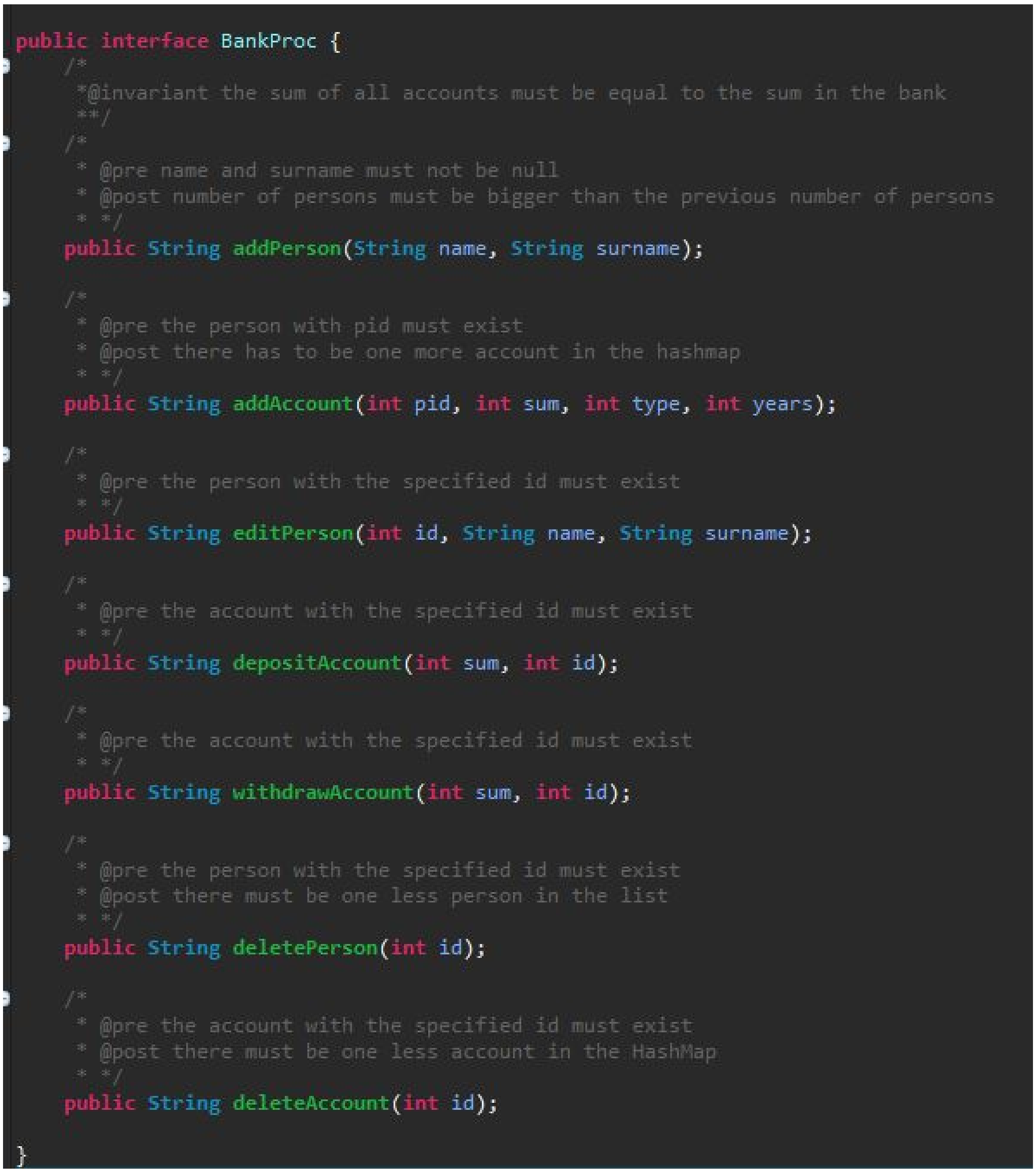


```java
public class SavingAccount extends Account {

    private int fixedSum, years;
    private double interest;
    private boolean  depositDone, withdrawalDone;
    private static final long serialVersionUID = 1L;

    public SavingAccount(int id, double interest, int fixedSum,int years) {

    public double getFinalSum()

    public String deposit(int sum) {

    public String withdraw(int sum) {
}
```

This class contains the definition for a saving account. It extends the Account class but it has some extra variables: the fixedSum that can only be deposited and withdrawn from this account, the number of years for which a deposit is made in this account, the interest per year and some state variables. The methods are: a constructor which receives all the variables, calls the constructi in the super class for id and sum and instantiates the other variable locally. The get final sum method computes the sum with the influence of the interest over the number of years of the length of the accounts life. The deposit and withdraw function are the implementation of the abstract classes defined in the super class Account. They check the conditions of the required action that is performed and return a printable string with the message of success or the reason of the failure. In case of success, the account also notifies the account holder about the changes that have been made. All the messages coming from these functions are printed in the status bar in the GUI, so for any operation, not only a user can check the Jtable for the changes, but he can also read the result in the status bar.

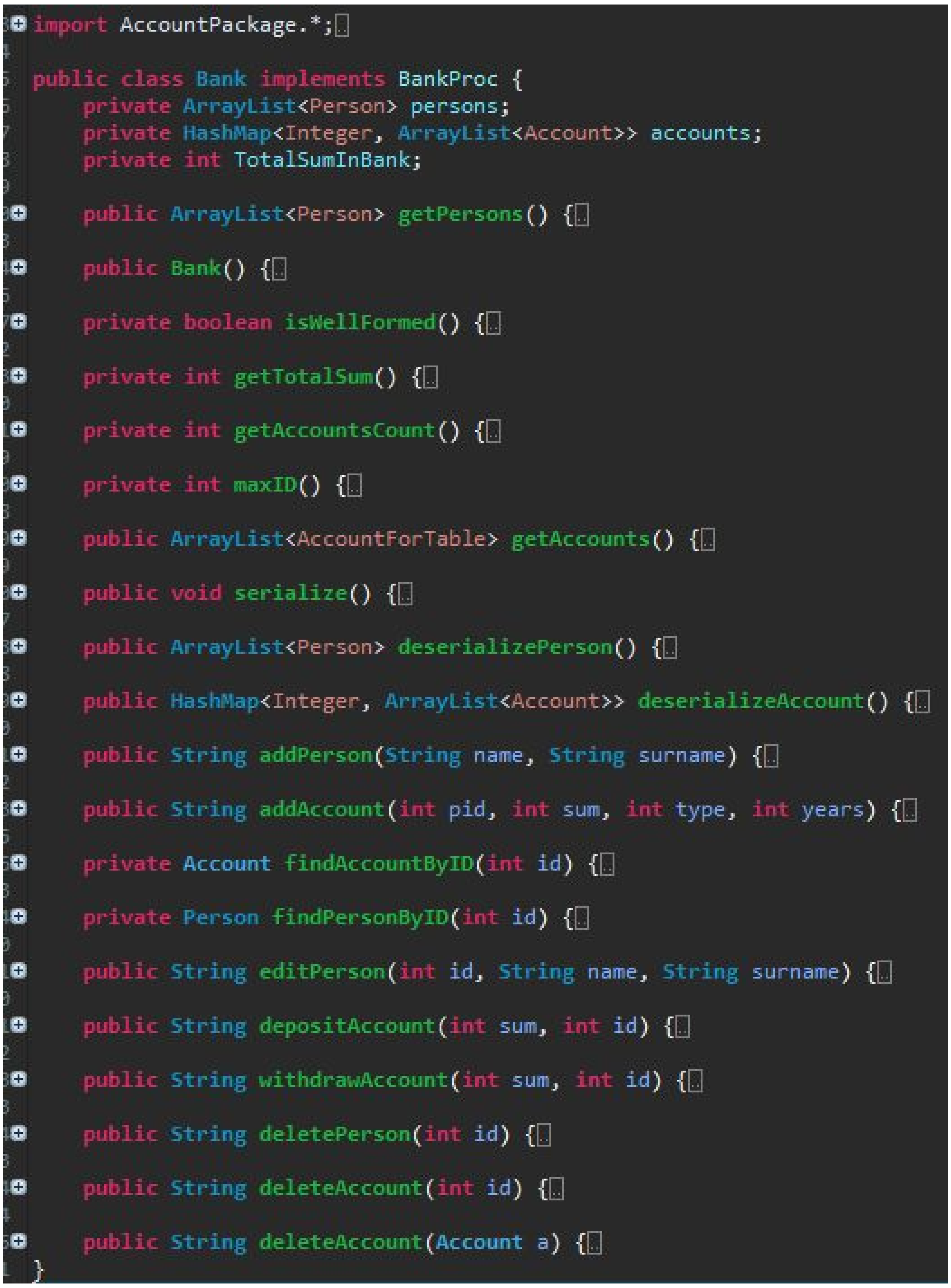
5.2. The Bank package

The bank package contains the definition of the class Bank and an interface, BankProc.



```java
public interface BankProc {
    /*
     *@invariant the sum of all accounts must be equal to the sum in the bank
     **/
    /*
     * @pre name and surname must not be null
     * @post number of persons must be bigger than the previous number of persons
     * */
    public String addPerson(String name, String surname);

    /*
     * @pre the person with pid must exist
     * @post there has to be one more account in the hashmap
     * */
    public String addAccount(int pid, int sum, int type, int years);

    /*
     * @pre the person with the specified id must exist
     * */
    public String editPerson(int id, String name, String surname);

    /*
     * @pre the account with the specified id must exist
     * */
    public String depositAccount(int sum, int id);

    /*
     * @pre the account with the specified id must exist
     * */
    public String withdrawAccount(int sum, int id);

    /*
     * @pre the person with the specified id must exist
     * @post there must be one less person in the list
     * */
    public String deletePerson(int id);

    /*
     * @pre the account with the specified id must exist
     * @post there must be one less account in the HashMap
     * */
    public String deleteAccount(int id);

}
```

The interface contains the definition of the necessary methods in a bank, alongside with the invariant of the class and preconditions and postconditions for each method. The invariant is checked before and after the execution of every method using the isWellFormed() function. If the invariant is not met, the execution is stopped by the assert method. The precondition of every method is checked at the beginning of that method and the post conditions are checked at the ending of every method.



```
import AccountPackage.*;

public class Bank implements BankProc {
    private ArrayList<Person> persons;
    private HashMap<Integer, ArrayList<Account>> accounts;
    private int TotalSumInBank;

    public ArrayList<Person> getPersons() {

    public Bank() {

    private boolean isWellFormed() {

    private int getTotalSum() {

    private int getAccountsCount() {

    private int maxID() {

    public ArrayList<AccountForTable> getAccounts() {

    public void serialize() {

    public ArrayList<Person> deserializePerson() {

    public HashMap<Integer, ArrayList<Account>> deserializeAccount() {

    public String addPerson(String name, String surname) {

    public String addAccount(int pid, int sum, int type, int years) {

    private Account findAccountByID(int id) {

    private Person findPersonByID(int id) {

    public String editPerson(int id, String name, String surname) {

    public String depositAccount(int sum, int id) {

    public String withdrawAccount(int sum, int id) {

    public String deletePerson(int id) {

    public String deleteAccount(int id) {

    public String deleteAccount(Account a) {
}
```

The bank class implements the BankProc interface. It contains an array list of persons and a Hashmap with the keys the ids of the persons and the values array lists of accounts. In the constructor the app tries to deserialize the previous version of the array list and hash map and computes the total money amount in the bank. If it fails or previous serializations do not exist the array list and hashmap are initialized with empty ones and the total sum is zero. The get total sum method computes the total sum of all the accounts in the bank. The get accounts count method returns the number of active accounts in the bank. The max id method returns the maximum id of a person. The add person method creates a new person and adds it to the array list of persons. Precondition: name and surname must not be null, Postcondition: number of persons must be bigger than the previous number of persons. The add account creates a new account of the required type and maps it to the person with the id equal to pid. Precondition: the person with pid must exist. Postcondition: there has to be one more account in the hashmap. The find person by id method returns the person object having the required ID. The find account by id method returns the account object having the required ID. The deposit account function deposits the requested sum in the account with id equal to the id parameter. Precondition: the account with the specified id must exist. The withdraw account function withdraws the requested sum from the account with id equal to the id parameter. Precondition: the account with the specified id must exist; The delete person method sets the person with the requested id as deleted. Precondition: the person with the specified id must exist. Postcondition: there must be one less person in the list.The delete account method sets the account with the requested id as deleted. Precondition: the account with the specified id must exist. Postcondition: there must be one less active account in the bank. The serialize method writes the current state of both the array list of persons and the mapping of accounts to person ids to files before the application is closed in order to preserve the states for the next run of the application. The deserialize methods try to bring the old states of the array list of persons and the mappings of the accounts to the ids of the clients from the files where the were previously serialized.

6. Results

The result of the development of this application, that lasted for two weeks together with writing the documentation, is a software easy to use that makes it easy for the user to manage a couple of bank clients and accounts, depositing and withdrawing money from those accounts.

7. Further development

- Adding different user levels on the application.
- Adding more details on persons.
- Adding more details on accounts;
- Printing financial states of different clients.
- Building a client side of the application.
- Adding to the manager side of the application an area for companies, nut only personal clients.

8.Conclusions

During the development of this project i learned and improved of several skills:

- A better use of Eclipse
- A better use of the Java coding language
- A better understanding and usage of the object oriented programming paradigms
- More organized and clear code
- A better understanding of the graphical user interface elements like Jtable and Tabbed pane.
- Learning about design pattern observer.
- Learning about design by contract.
- Learning about serialization.

9. Bibliography

1. Java general documentation https://docs.oracle.com/javase/7/docs/api/
2. Lecture and laboratory guidelines http://www.coned.utcluj.ro/~salomie/PT_Lic/3_Lab/
3. Word counter https://wordcounter.net/