***Programming Techniques***

***Simulating an application for processing bank accounts***

***Student : Sporis Ioan Catalin***

***Group: 30421***

**Contents :**

1. **Project Specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . pg**
2. **Problem analysis, modeling, scenarios, using cases . . . . . . . . . . . . . . . pg**
3. **Design (UML diagram, data structures, classes implementations, interfaces, relations, packages, algorithms, graphical user interface ) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . pg**
4. **Implementing and testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .pg**
5. **Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . pg**
6. **Conclusions, further developments. . . . . . . . . . . . . . . . . . . . . . . . . . . . . pg**
7. **Project Specification**

The objective of this homework is to implement and simulate an application that prosseses accounts that can be created for bank customers. The implementation involves creating accounts, that can be of 2 types spending account or saving account. By virtue of the selected option to create an account we are able to deposit or to withdraw money from the account. There is also the option to delete an account, and also an option to see all the existing accounts or seing the accouns according to the selected type of the account spending or saving. Another task of this homework is to write and implement the conditions needed accord to design by contract. The approach of this design involves writing and implementing pre and post conditions. The task of this application is to design and implement the principles of a bank starting from a class diagram. According to this diagram there are also included some tasks: within the interface BankProc the pre and post conditions will be implemented. Also the classes Person, Account, SavingAccount, SpendingAccount and other classes will be implemented by the programmer. The Class Bank will be implemented, using a hashtable and also drive tests must be performed for the implemented methods. In conclusion I have realized an application that simulates the creation and maintance of the accounts within a bank. This system has a great distribution in every day life. Within every bank there is a graphical user interface for editing accounts and for finding all the customers. This application is useful including the fact that online shopping has grown in the last years, these being possible through internet banking. We have to find the pre and post conditions that must be done and these are included within the BankProc interface, because here all the methods that realize account processing.

1. **Analysis of the problem, modeling application, scenarios, using cases**
2. **Analysing and modeling the problem**

After reading the problems specifications we notice that will need the following classes, which are presented in the task. The Class Person has as attribute a lastname, firstname and also an id, in order to identify the person. We have also 2 classes SavingAccount and SpendingAccount extending the class Account.

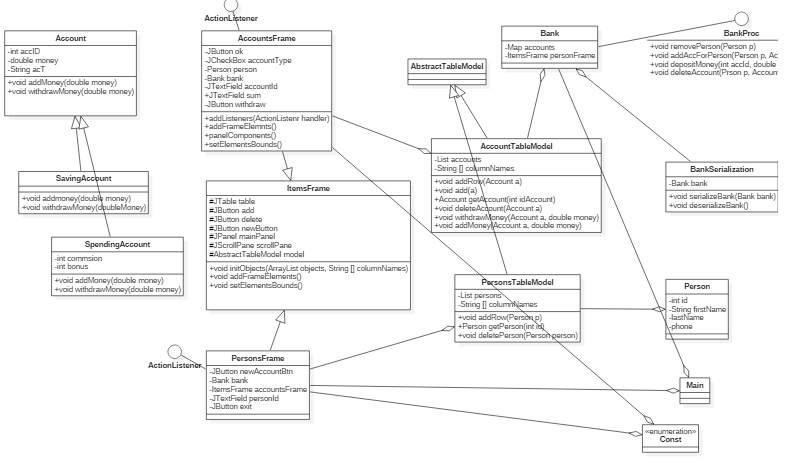
These 2 classes inherit the class Account. They both contain methods for deposit and withdraw money. I have also implemented an interface, BankProc where we can find the methods to create a new account, to delete one.

The next class is Bank, which realizes the implementation from the interface BankProc. Here I have implemented a HashMap<Person,Set<Account>, and also other methods.

1. **Design**

**UML diagram, packages**

The programming language used for building this application is Java, and the developing platform is Eclipse. The interaction with the user is realized in an interactive way using a graphical user interface.



**Classes:**

1. **Account**

**Fields:**

* private int accId;
* private double money;
* private String acT;

**Methods :**

* public abstract void addMoney ( double money ) ;
* public abstract void withdrawMoney ( double money ) ;
* getters and setters ;

1. **AccountsFrame extends ItemsFrame < Account > implements ActionListener**

**Fields :**

* **private** JButton ok;
* **private** JCheckBox accountType ;
* **private** Person person ;
* **private** Bank bank ;
* **private** JTextField accounted ;
* **private** JTextField sum ;
* **private** JButton withdraw ;
* **private** JButton addMoney ;

**Methods :**

* public void addListeners ( ActionListener handler ) ;
* public void addFrameElements ( ) ;
* public void setElementsBounds ( ) ;
* public void initObjects ( ArrayList < Account > accounts , String [ ] columnNames ) ;
* getters and setters

1. **AccountTableModel extends AbstractTableModel**

**Fields :**

* **private** List<Account> accounts;
* **private** String [] columnNames;

**Methods :**

* Public void addRow (Account a) ;
* Public void add ( Account a ) ;
* Public Account getAccount ( int idAccount ) ;
* Public void deletrAccount (Account a ) ;
* Public void withdrawMoney ( Account a, double money ) ;
* Public void addMoney ( Account a, double money ) ;
* Getters and setters

1. **Bank implements BankProc, Serializable**

**Fields :**

* **private** Map<Person,Set<Account>> accounts;
* **private** ItemsFrame personFrame;
* **private** String [] columnNamesAccounts = {"Id","Money"};
* **private** String [] columnNamesPersons = {"Id","FirstName","LastName","Phone"};

**Methods :**

* Methods from interface
* Getters and setters

1. **Interface Bankproc**

**Methods :**

* + public void removePerson(Person p);
* public void addAccForPerson(Person p,Account asocAcc);
* public void depositMoney(int accId,double sum,Person p);
* public void uploadAccountsData(Person p);
* public void loadAccountsData(Person p);
* public void deleteAccount(Person p,Account a);

1. **BankSerialize**

**Fields :**

* Public static Bank bank ;

**Methods :**

* Public static void serializeBank (Bank bank) ;
* Public static Bank deserializeBank() ;

**5. Results**

The results have a designated area in the graphical user interface. By testing the right functioning of the application on various types of data, we can see that these are well performed. So the simulator has a good performance. Also the result can be immediately observed for the most of the operations. For example in case a new account is added or an existing account is deleted the results are available quickly. This fact is happening too when visualizing the accounts and also filtering the accounts after type.

**6. Conclusions, further developments**

To sum up, realizing a program which is able to simulate a bank is not as hard as I thought at the very beginning. This application is easy to use because of the graphical user interface. Realising this homework, I learned how use the design by contract concept, involving pre, post conditions, invariants, assertions. Also I have learned how to use HashMap and the methods add, put, remove.

The program could be developed to make more filters, to display accounts having certain attributes, like name and their amount of money, see the date when they were added and so on.