Homework 4
Bank

Buzea Vlad-Calin
30422

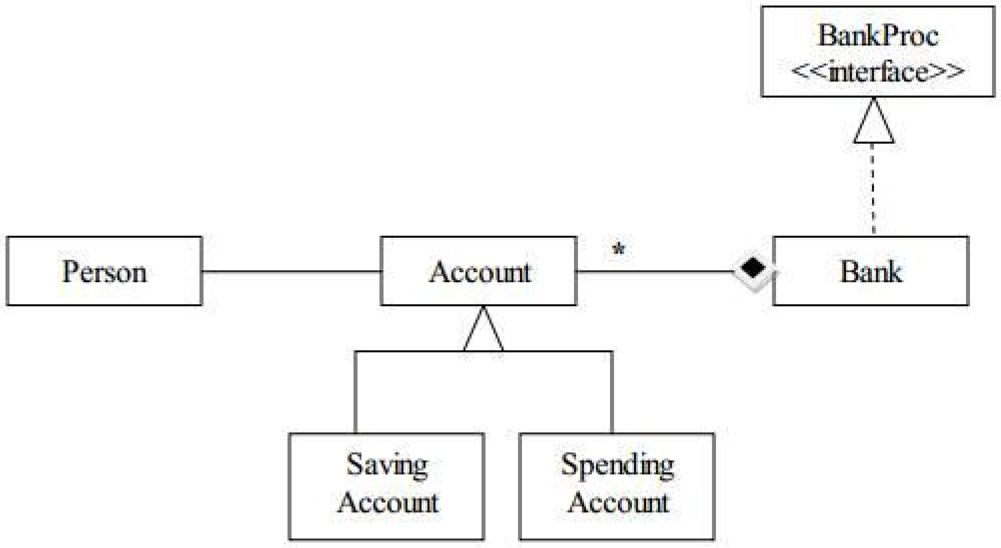
Table of Contents

1. Objective of the assignment

Design by Contract Programming Techniques

Description

Consider the system of classes in the class diagram below.



1.Define the interface BankProc (add/remove accounts, read/write accounts, report generators, etc). Specify the pre and post conditions for the interface methods.
2. Implement the classes Person, Account, SavingAccount and SpendingAccount. Other classes may be added as needed (give reasons for the new added classes).
3. Implement the class Bank using a hashtable. Chaining will be used in case of collisions. A method toString will be also defined for the class Bank.
3.1 Define a method of type "well formed" for the class Bank.
3.2 Implement the class using Design by Contract method (involving pre, post conditions, invariants, assertions).
4. Implement a user guided test driver for the system. The initial account data for populating a Bank object with Account objects will be taken from a file. Testing evidence should be provided to the console for defined methods.

2. Problem analysis

2.1. Modeling

A quick overview over the objective shows that, for this project, a bank is a collection of accounts. On this collection, that takes the form of a hashtable, we may perform various operations: add/remove accounts, read/write accounts, report generators, etc. An interesting aspect is the difference between the saving account and the spending account.

A **bank account** is a financial account between a bank customer and a financial institution. A bank account can be a deposit account, a credit card, or any other type of account offered by a financial institution. The financial transactions which have occurred within a given period of time on a bank account are reported to the customer on a bank statement and the balance of the account at any point in time is the financial position of the customer with the institution. a fund that a customer has entrusted to a bank and from which the customer can make withdrawals.

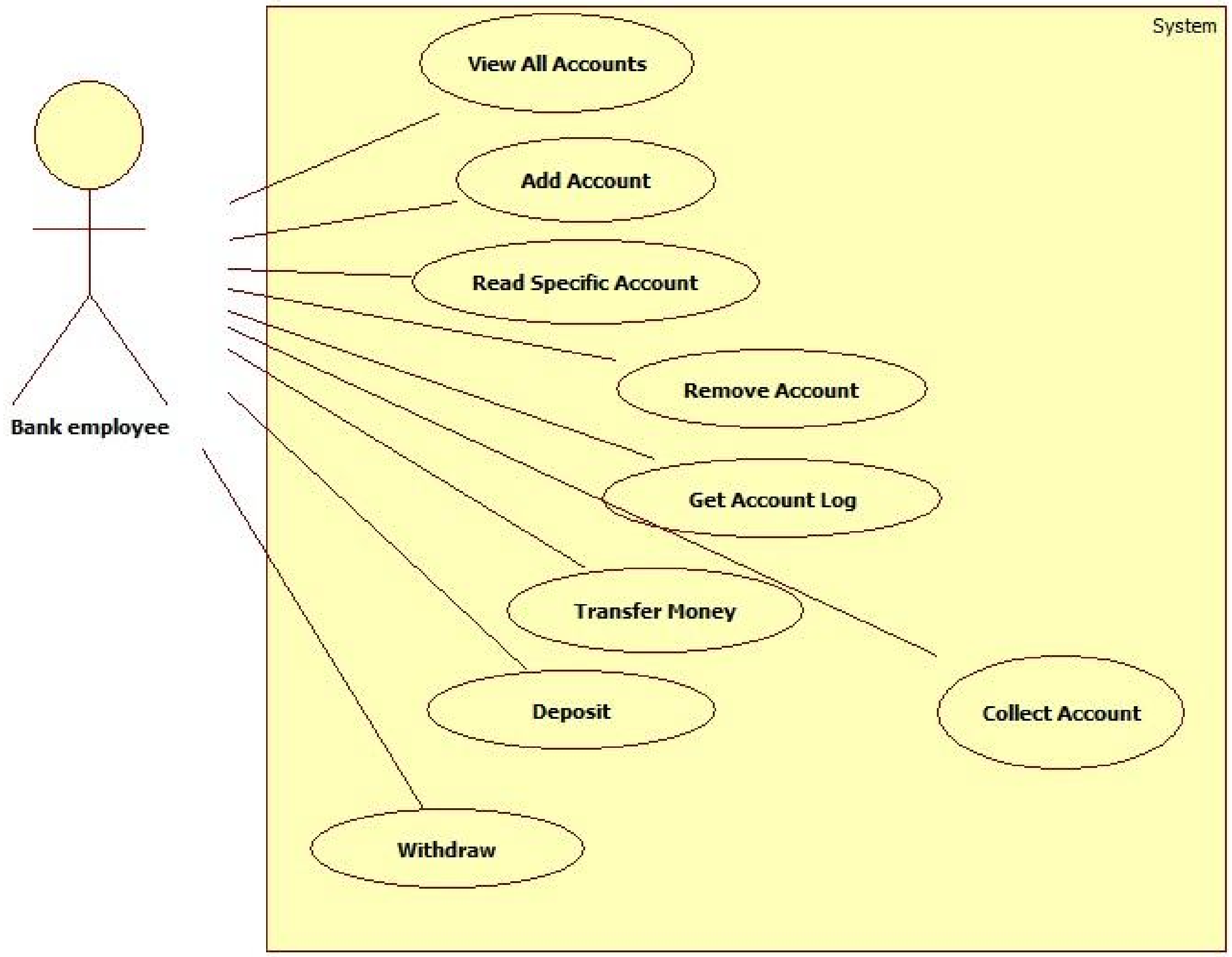
Saving accounts are accounts maintained by retail financial institutions that pay interest but cannot be used directly as money in the narrow sense of a medium of exchange (for example, by writing a cheque). These accounts let customers set aside a portion of their liquid assets while earning a monetary return. For the bank, money in a savings account may not be callable immediately and in some jurisdictions, does not incur a reserve requirement, freeing up cash from the bank's vault to be lent out with interest.

A **transactional account**, known as a **current account** (British English) or **checking account** (American English), is a deposit account held at a bank or other financial institution, for the purpose of securely and quickly providing frequent access to funds on demand, through a variety of different channels.
Transactional accounts are meant neither for the purpose of earning interest nor for the purpose of savings, but for convenience of the business or personal client; hence they tend not to bear interest. Instead, a customer can deposit or withdraw any amount of money any number of times, subject to availability of funds.

One may wonder about the money chain that takes place in a bank, because all these accounts may seem unprofitable. Well, in real life the accounts play the role of sustaining the bank's real source of income: the loans. Although we will not implement them, we will mention for the record their usage. In finance, a loan is a debt provided by one entity (organization or individual) to another entity at an interest rate, and evidenced by a note which specifies, among other things, the principal amount, interest rate, and date of repayment. A loan entails the reallocation of the subject asset(s) for a period of time, between the lender and the borrower. In a loan, the borrower initially receives or borrows an amount of money, called the principal, from the lender, and is obligated to pay back or repay an equal amount of money to the lender at a later time. Typically, the money is paid back in regular installments, or partial repayments; in an annuity, each installment is the same amount. The loan is generally provided at a cost, referred to as interest on the debt, which provides an incentive for the lender to engage in the loan. In a legal loan, each of these obligations and restrictions is enforced by contract, which can also place the borrower under additional restrictions known as loan covenants. Although this article focuses on monetary loans, in practice any material object might be lent.

2.2. Use Cases

Due to time considerations my application has only implemented the part regarding the bank employees' interface. They have access to all operations implemented, thus it is relevant enough if we test it. The client interface would be restricted just to withdraw operations, making it irrelevant for our testing.



2.3. Scenarios

Scenario I

a)Identification Summary

Title: Addition of a new account

Summary: This case allows the bank employee to create a new account referring a Person and a type: Savings or Spending

Actor: bank employee

b)Flow of Events

Preconditions: The user interface did not malfunction and the user input data was correct.

Main success scenario:

1. The employee starts the Bank application.
2. The employee chooses from the list of buttons the button "Add Account" and clicks it.
3. A new window is opened for the creation of an account (Title: "Add Account").
4. The employee completes all the owner information(SSID, First Name, Last Name, Email, Address, Phone Number)
5. The employee selects the account type (Savings or Spending)
6. The employee introduces the initial account balance
7. If Spending account was selected jump to step 9 , else continue
8. The employee introduces the account due date
9. The employee click the "Add Account" button the complete the operation
10. The application shows a message:"ERROR" in case of bad input, "Account successfully added" in case of success

Scenario II

a)Identification Summary

Title: Read Account

Summary: This case has the purpose of showing the account information of an account ID

Actor: bank employee

b)Flow of Events

Preconditions: The user interface did not malfunction.

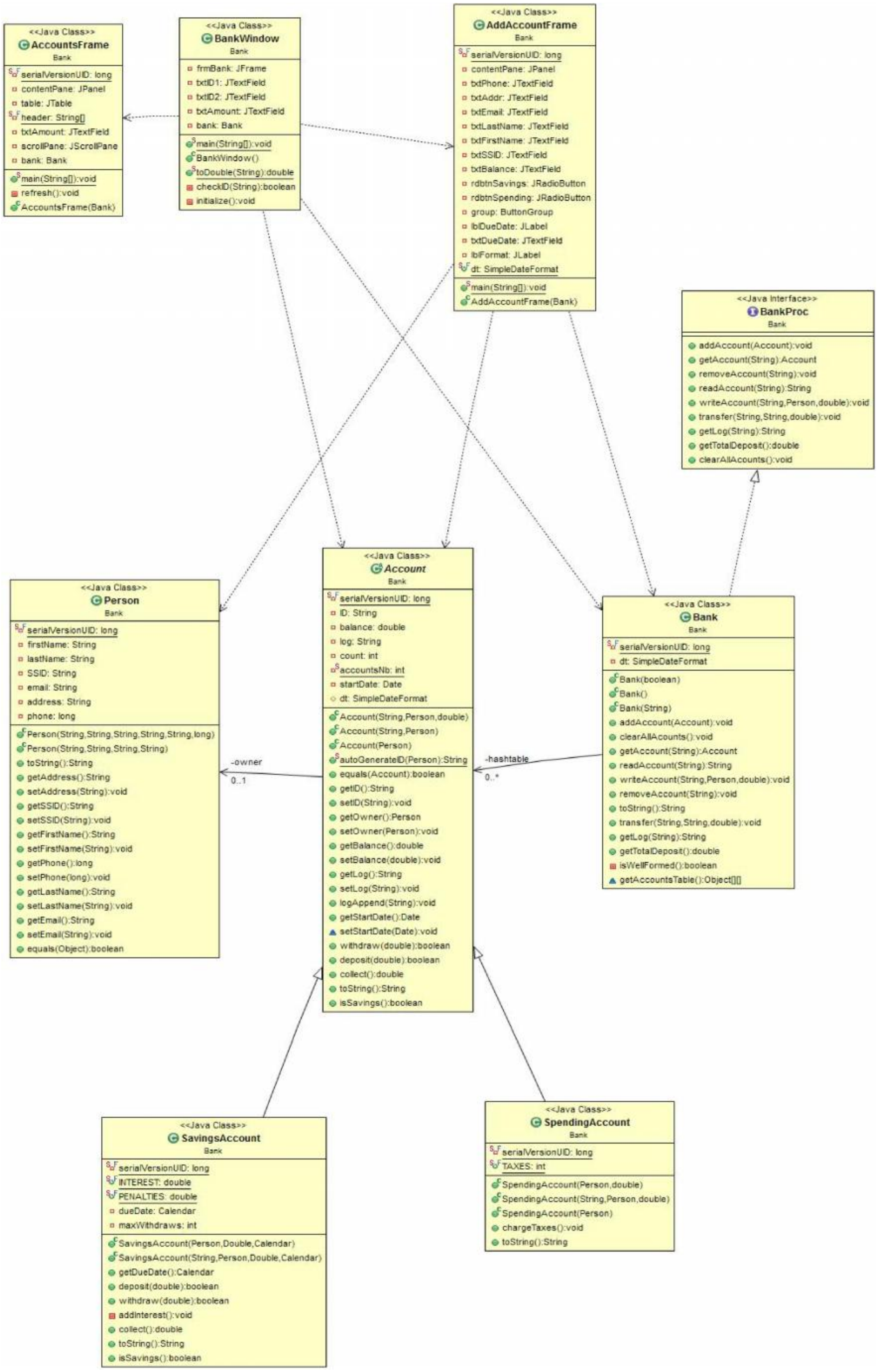
Main success scenario:

1. The employee starts the Bank application.
2. The employee introduces the desired ID in the Account ID 1 text field.
3. The employee clicks the "Read Account" button
4. The System shows a message: "Not a Valid ID!" in case the account ID does not exist in the bank, the Account information otherwise. By Account information we mean a String of the form:
   "
   Account_ID
   Owner_SSID; Owner_first_name Owner_lastname; email:Owner_email; address:Owner_address; phone number:Owner_phone; balance$
   Account_type
   Start_Date [Due_Date]
   "

3. Design

3.1. Class Diagram

The contracted class diagram has been presented in the Objective section. Further on we will present the extended class diagram. Please note that due to software considerations, the relationships could not be fully represented. The real, implemented relationships are those presented in the initial UML Class Diagram. If not visible, please consult the project folder!



<<Java Class>>
AccountsFrame
Bank
serialVersionUID: long
contentPane: JPanel
table: JTable
header: String[]
txtAmount: JTextField
scrollPane: JScrollPane
bank: Bank
main(String[]):void
refresh():void
AccountsFrame(Bank)
<<Java Class>>
BankWindow
Bank
frmBank: JFrame
txtID1: JTextField
txtID2: JTextField
txtAmount: JTextField
bank: Bank
main(String[]):void
BankWindow()
toDouble(String):double
checkID(String):boolean
initialize():void
<<Java Class>>
AddAccountFrame
Bank
serialVersionUID: long
contentPane: JPanel
txtPhone: JTextField
txtAddr: JTextField
txtEmail: JTextField
txtLastName: JTextField
txtFirstName: JTextField
txtSSID: JTextField
txtBalance: JTextField
rdbtnSavings: JRadioButton
rdbtnSpending: JRadioButton
group: ButtonGroup
lblDueDate: JLabel
txtDueDate: JTextField
lblFormat: JLabel
dt: SimpleDateFormat
main(String[]):void
AddAccountFrame(Bank)
<<Java Interface>>
BankProc
Bank
addAccount(Account):void
getAccount(String):Account
removeAccount(String):void
readAccount(String):String
writeAccount(String,Person,double):void
transfer(String,String,double):void
getLog(String):String
getTotalDeposit():double
clearAllAcounts():void
<<Java Class>>
Person
Bank
serialVersionUID: long
firstName: String
lastName: String
SSID: String
email: String
address: String
phone: long
Person(String,String,String,String,String,long)
Person(String,String,String,String,String)
toString():String
getAddress():String
setAddress(String):void
getSSID():String
setSSID(String):void
getFirstName():String
setFirstName(String):void
getPhone():long
setPhone(long):void
getLastName():String
setLastName(String):void
getEmail():String
setEmail(String):void
equals(Object):boolean
-owner
0..1
<<Java Class>>
Account
Bank
serialVersionUID: long
ID: String
balance: double
log: String
count: int
accountsNb: int
startDate: Date
dt: SimpleDateFormat
Account(String,Person,double)
Account(String,Person)
Account(Person)
autoGenerateID(Person):String
equals(Account):boolean
getID():String
setID(String):void
getOwner():Person
setOwner(Person):void
getBalance():double
setBalance(double):void
getLog():String
setLog(String):void
logAppend(String):void
getStartDate():Date
setStartDate(Date):void
withdraw(double):boolean
deposit(double):boolean
collect():double
toString():String
isSavings():boolean
-hashtable
0..*
<<Java Class>>
Bank
Bank
serialVersionUID: long
dt: SimpleDateFormat
Bank(boolean)
Bank()
Bank(String)
addAccount(Account):void
clearAllAcounts():void
getAccount(String):Account
readAccount(String):String
writeAccount(String,Person,double):void
removeAccount(String):void
toString():String
transfer(String,String,double):void
getLog(String):String
getTotalDeposit():double
isWellFormed():boolean
getAccountsTable():Object[][]
<<Java Class>>
SavingsAccount
Bank
serialVersionUID: long
INTEREST: double
PENALTIES: double
dueDate: Calendar
maxWithdraws: int
SavingsAccount(Person,Double,Calendar)
SavingsAccount(String,Person,Double,Calendar)
getDueDate():Calendar
deposit(double):boolean
withdraw(double):boolean
addInterest():void
collect():double
toString():String
isSavings():boolean
<<Java Class>>
SpendingAccount
Bank
serialVersionUID: long
TAXES: int
SpendingAccount(Person,double)
SpendingAccount(String,Person,double)
SpendingAccount(Person)
chargeTaxes():void
toString():String

3.2. Data Structures

The main data structure used in this project, besides the usual arrays, is the Hashtable.

public class **Hashtable**<K,V>
extends Dictionary<K,V>
implements Map<K,V>, Cloneable, Serializable
This class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value.
To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode() method and the equals method.

An instance of Hashtable has two parameters that affect its performance: initial capacity and load factor. The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created. Note that the hash table is open: in the case of a "hash collision", a single bucket stores multiple entries, which must be searched sequentially. The load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased. The initial capacity and load factor parameters are merely hints to the implementation. The exact details as to when and whether the rehash method is invoked are implementation-dependent.

Generally, the default load factor (.75) offers a good tradeoff between time and space costs. Higher values decrease the space overhead but increase the time cost to look up an entry (which is reflected in most Hashtable operations, including get and put).

The initial capacity controls a tradeoff between wasted space and the need for rehash operations, which are time-consuming. No rehash operations will ever occur if the initial capacity is greater than the maximum number of entries the Hashtable will contain divided by its load factor. However, setting the initial capacity too high can waste space.

If many entries are to be made into a Hashtable, creating it with a sufficiently large capacity may allow the entries to be inserted more efficiently than letting it perform automatic rehashing as needed to grow the table.

This example creates a hashtable of numbers. It uses the names of the numbers as keys:

```
  Hashtable<String, Integer> numbers
    = new Hashtable<String, Integer>();
  numbers.put("one", 1);
  numbers.put("two", 2);
  numbers.put("three", 3);
```

To retrieve a number, use the following code:

```
  Integer n = numbers.get("two");
  if (n != null) {
    System.out.println("two = " + n);
  }
```

The iterators returned by the iterator method of the collections returned by all of this class's "collection view methods" are fail-fast: if the Hashtable is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a ConcurrentModificationException. Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future. The Enumerations returned by Hashtable's keys and elements methods are not fail-fast.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: the fail-fast behavior of iterators should be used only to detect bugs.

As of the Java 2 platform v1.2, this class was retrofitted to implement the Map interface, making it a member of the Java Collections Framework. Unlike the new collection implementations, Hashtable is synchronized. If a thread-safe implementation is not needed, it is recommended to use HashMap in place of Hashtable. If a thread-safe highly-concurrent implementation is desired, then it is recommended to use ConcurrentHashMap in place of Hashtable.

3.3 Algorithms

The Algorithms used in this project are those corresponding to hashtables:

- Hashtable insert : compute value of hash function(corresponding to transmitted key) and insert value at the end of the linked list corresponding to that table entry.
- Hashtable retrieve: compute value of hash function(corresponding to transmitted key) and search the value in linked list corresponding to that table entry
- Hashtable delete: compute value of hash function(corresponding to transmitted key) and search the value in linked list corresponding to that table entry; if found, delete it from the linked list.

3.4 Class Design

CRC CARDS

Person	
Attributes, Functions:	Collaborator classes
Name, SSID, Address, phone number, email;	

Account	
Attributes, Functions	Collaborator classes
Has identification number Has owner Has balance Has activity Log Has start date Has interest Owner can check balance Owner can withdraw or deposit money Owner can collect all money in account	Person

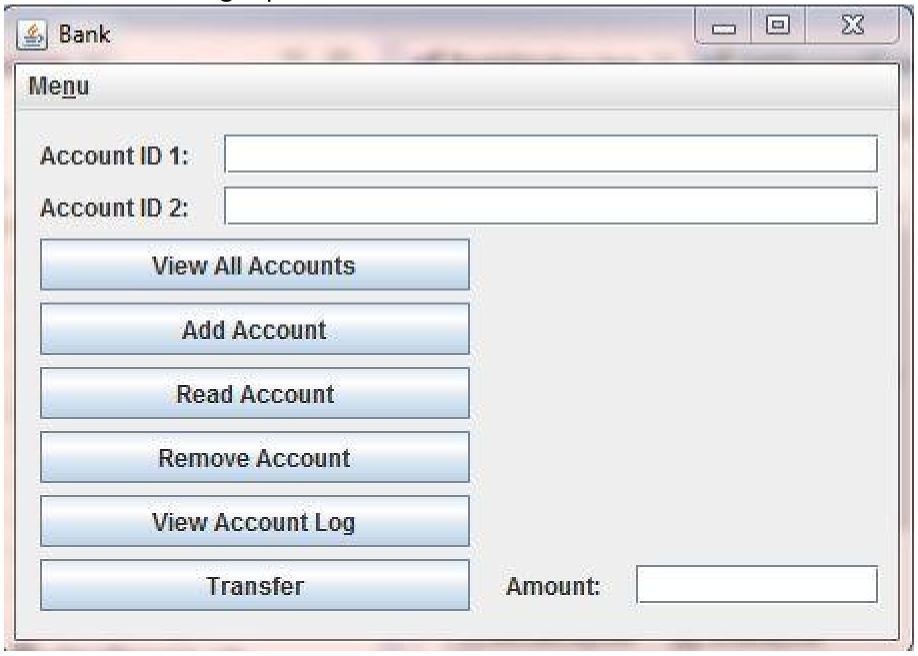
SpendingsAccount extends Account	
Attributes, Functions	Collaborator classes
Small taxes(fees) Owner has unlimited withdraws and deposits	

SavingsAccount extends Account	
Attributes, Functions	Collaborator classes
Customer receives interest after the deposited amount May have penalties in case of withdraw May have limited withdraws May lose cumulated interest in case of deposit Has due date Cannot collect before due date	

Bank	
Attributes, Functions	Collaborator classes
Has many accounts Transfer money between accounts Get total balance Add/remove account Read/write account Get account logs Get monthly log View Accounts	Hashtable Account

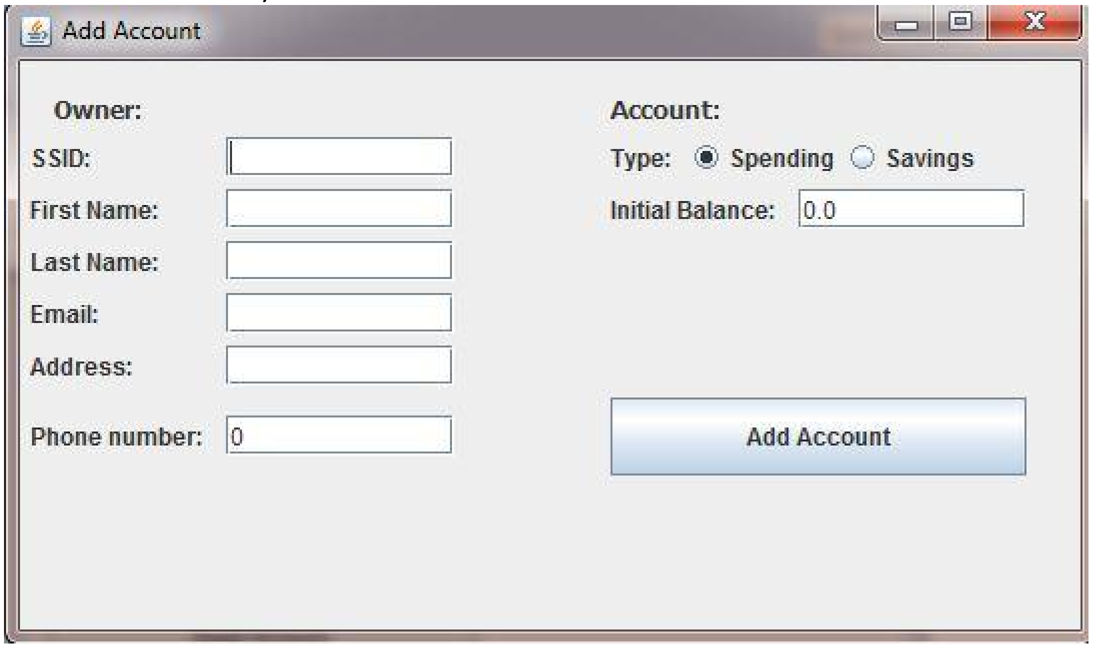
3.5. Interface

Due to the fact that only the employee interface was implemented, the User Interfaces have a smaller number. The main frame has the following aspect:



In this frame the user can introduce two account IDs in the two text fields placed on the top part. He may also choose to View all the Accounts, add a new Account, read information regarding an account (the account with the ID introduced in the Account ID 1 corresponding text field), remove an account (the account with the ID introduced in the Account ID 1 corresponding text field), get the account log (the account with the ID introduced in the Account ID 1 corresponding text field),transfer money from the first account to the second, specifying the amount in the rightmost text field (labeled "Amount:").

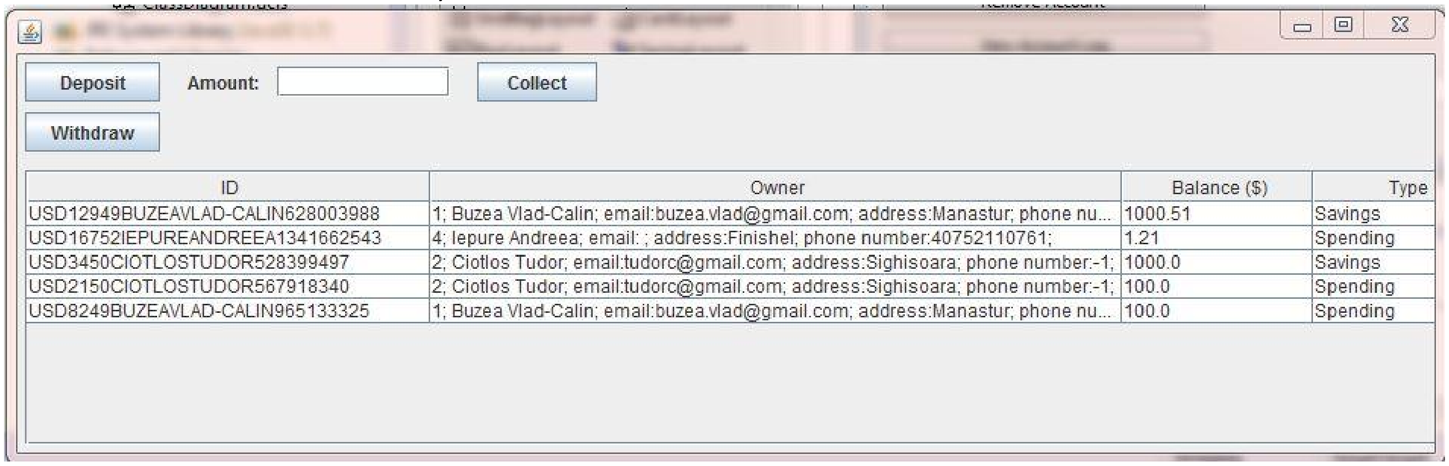
From the main frame we may also access the AddAccountFrame:



Type: [✓ Spending] Savings
Initial Balance: 0.0
Phone number: 0

This frame has the purpose of creating a new account, thus facilitating 6 text boxes for completing information regarding the owner (SSID, First Name, Last Name, Email, Address, Phone number) that must all be completed. In case one of the fields cannot be completed (let's say the phone number) we suggest filling the field with "0". Further on we select the Account type: Spending or Savings. These two values are complementary, thus they take the form of radio buttons in the same ButtonGroup. Further on we complete the balance in the next text field (and the due date in case of SavingsAccount). Finally we press the "Add Account" button and finish the operation.

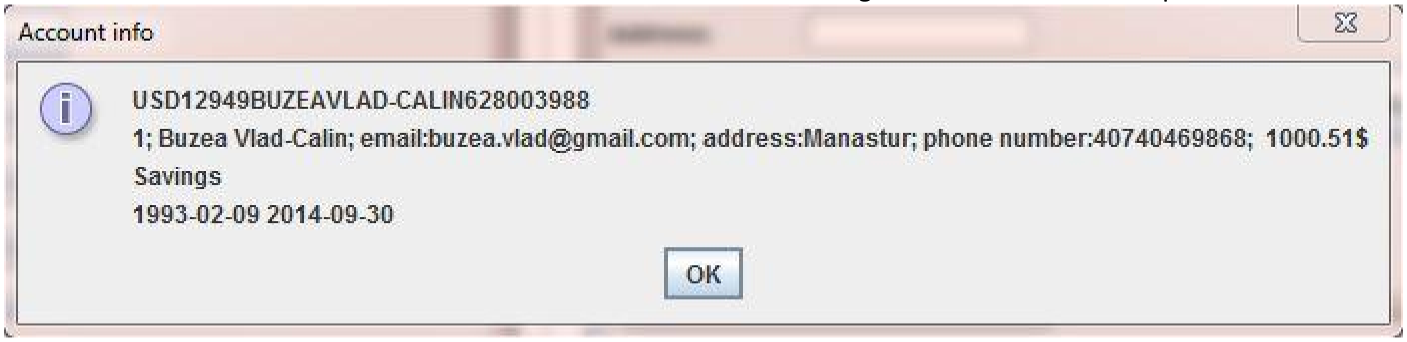
From the main frame we may also access the AllAccountsFrame:



ID	Owner	Balance ($)	Type
USD12949BUZEAVLAD-CALIN628003988	1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone nu...	1000.51	Savings
USD16752IEPUREANDREEA1341662543	4; Iepure Andreea; email: ; address:Finishel; phone number:40752110761;	1.21	Spending
USD3450CIOTLOSTUDOR528399497	2; Ciotlos Tudor; email:tudorc@gmail.com; address:Sighisoara; phone number:-1;	1000.0	Savings
USD2150CIOTLOSTUDOR567918340	2; Ciotlos Tudor; email:tudorc@gmail.com; address:Sighisoara; phone number:-1;	100.0	Spending
USD8249BUZEAVLAD-CALIN965133325	1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone nu...	100.0	Spending

This window is used to display all the accounts in the bank, in a JTable that has as columns the account ID, the account owner, the account balance and the account type. In this window the user can also deposit or withdraw a specified amount, or collect the account (if passed due date).

Another GUI element to be mentioned is the information message that transmits various pieces of information:



USD12949BUZEAVLAD-CALIN628003988
1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone number:40740469868; 1000.51$
Savings
1993-02-09 2014-09-30

4. Implementation and testing

In this section we will specify the most relevant functions for the implementation of the application:

- void addAccount(Account a)

Adds account to bank storage

- void clearAllAcounts()

Empties the bank of all accounts

- Account getAccount(java.lang.String ID)

Returns the account with the specified ID.

- String getLog(java.lang.String ID)

Generates report regarding an Account

- double getTotalDeposit()

Calculates the total amount of money stored in the bank

- String readAccount(java.lang.String ID)

Returns information regarding the requested account

- void removeAccount(java.lang.String ID)

Removes an account from the bank

- void transfer(java.lang.String destID, java.lang.String sourceID, double amount)

Transfers the specified amount from source account to destination account

- void writeAccount(java.lang.String ID, Person newOwner, double balance)

Edits the properties of an account

- static String autoGenerateID(Person owner)

Generates an account ID that is based on the owner's data and should not be duplicate. It is a static method that can be called by the constructor of the Account class, such that the user doesn't have to manually introduce one. The uniqueness of the ID is guaranteed by an account number and the impredictibility by the last digits that are a hash function of the owner.

- double collect()

Empties the account balance and returns the corresponding amount that has to paid to the client. It is overridden in the SavlingsAccount class to return 0 in case the due date has not been reached yet. Otherwise it will also add the corresponding interest.

- boolean deposit(double amount)

Adds the transmitted amount to the account balance. It is overridden in the SavlingsAccount class such that all cumulated interested is discarded in case of deposit.

- boolean isSavings()

Checks if an account is a SavingsAccount.

- boolean withdraw(double amount)

Extracts from the account the requested amount, if enough balance. It is overridden in the SavlingsAccount class to allow maximum 6 withdraws/account and apply a penalty at each withdraw.

Testing has been done in the TestDrivers class for preliminary tests and in the BankTestDriver class for the final proofing. The code sequence executed was the following:

```
Bank bank=new Bank("Accounts.txt");

            System.out.println(bank.toString());   //I prints the initial accounts found in the 
bank

            Person raul=new Person("3","Jantea","Raul","jantearaul@gmail.com","Baia de 
Aries",+40749690691L);
            Account accountRaul=new SpendingAccount(raul);
            bank.addAccount(accountRaul);
            System.out.println(bank.toString());   //II adds raul to the bank an prints

            bank.transfer(accountRaul.getID(), "USD12949BUZEAVLAD-CALIN628003988", 50.0);
            System.out.println(bank.toString());  //III transfers 50$ from vlad to raul and 
prints

            System.out.println(bank.getLog("USD8249BUZEAVLAD-CALIN965133325")); //IV prints the 
log of vlad
            System.out.println("Bank total deposits: "+bank.getTotalDeposit()+"$\r\n");//V prints 
the bank total deposits

            System.out.println("Account info:"+bank.getAccount("USD12949BUZEAVLAD-
CALIN628003988").toString()); //VI prints vlad's account info
            System.out.println("Collected:"+bank.getAccount("USD12949BUZEAVLAD-
CALIN628003988").collect()+"\r\n");// VII prints collected amount from account
            bank.removeAccount("USD12949BUZEAVLAD-CALIN628003988");
            System.out.println(bank.toString()); //VII removes account and prints
```

The results of the test are the following:

```
USD12949BUZEAVLAD-CALIN628003988
1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone number:40740469868;
1000.51$
Savings
1993-02-09 2014-05-06
USD16752IEPUREANDREEA1341662543
4; Iepure Andreea; email: ; address:Finishel; phone number:40752110761;  1.21$
Spending
2014-04-27
USD3450CIOTLOSTUDOR528399497
2; Ciotlos Tudor; email:tudorc@gmail.com; address:Sighisoara; phone number:-1;  1000.0$
Savings
2014-39-24 2015-56-22
USD2150CIOTLOSTUDOR567918340
2; Ciotlos Tudor; email:tudorc@gmail.com; address:Sighisoara; phone number:-1;  100.0$
Spending
2014-34-24
USD8249BUZEAVLAD-CALIN965133325
1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone number:-1;  100.0$
Spending
2014-03-24

USD12949BUZEAVLAD-CALIN628003988
1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone number:40740469868;
1000.51$
Savings
1993-02-09 2014-05-06
USD16752IEPUREANDREEA1341662543
4; Iepure Andreea; email: ; address:Finishel; phone number:40752110761;  1.21$
Spending
2014-04-27
USD3450CIOTLOSTUDOR528399497
2; Ciotlos Tudor; email:tudorc@gmail.com; address:Sighisoara; phone number:-1;  1000.0$
Savings
2014-39-24 2015-56-22
USD1851JANTEARAUL787652646
3; Jantea Raul; email:Baia de Aries; address:jantearaul@gmail.com; phone number:40749690691;  0.0$
Spending
2014-05-06
USD2150CIOTLOSTUDOR567918340
2; Ciotlos Tudor; email:tudorc@gmail.com; address:Sighisoara; phone number:-1;  100.0$
Spending
2014-34-24
USD8249BUZEAVLAD-CALIN965133325
1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone number:-1;  100.0$
Spending
2014-03-24

USD12949BUZEAVLAD-CALIN628003988
1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone number:40740469868;
950.51$
Savings
1993-02-09 2014-05-06
USD16752IEPUREANDREEA1341662543
```

```
4; Iepure Andreea; email: ; address:Finishel; phone number:40752110761;  1.21$
Spending
2014-04-27
USD3450CIOTLOSTUDOR528399497
2; Ciotlos Tudor; email:tudorc@gmail.com; address:Sighisoara; phone number:-1;  1000.0$
Savings
2014-39-24 2015-56-22
USD1851JANTEARAUL787652646
3; Jantea Raul; email:Baia de Aries; address:jantearaul@gmail.com; phone number:40749690691;
50.0$
Spending
2014-05-06
USD2150CIOTLOSTUDOR567918340
2; Ciotlos Tudor; email:tudorc@gmail.com; address:Sighisoara; phone number:-1;  100.0$
Spending
2014-34-24
USD8249BUZEAVLAD-CALIN965133325
1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone number:-1;  100.0$
Spending
2014-03-24

Created: 2014-05-06

Bank total deposits: 2201.7200000000003$

Account info:USD12949BUZEAVLAD-CALIN628003988
1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone number:40740469868;
950.51$
Savings
1993-02-09 2014-05-06
Collected:2727.01319

USD16752IEPUREANDREEA1341662543
4; Iepure Andreea; email: ; address:Finishel; phone number:40752110761;  1.21$
Spending
2014-04-27
USD3450CIOTLOSTUDOR528399497
2; Ciotlos Tudor; email:tudorc@gmail.com; address:Sighisoara; phone number:-1;  1000.0$
Savings
2014-39-24 2015-56-22
USD1851JANTEARAUL787652646
3; Jantea Raul; email:Baia de Aries; address:jantearaul@gmail.com; phone number:40749690691;
50.0$
Spending
2014-05-06
USD2150CIOTLOSTUDOR567918340
2; Ciotlos Tudor; email:tudorc@gmail.com; address:Sighisoara; phone number:-1;  100.0$
Spending
2014-34-24
USD8249BUZEAVLAD-CALIN965133325
1; Buzea Vlad-Calin; email:buzea.vlad@gmail.com; address:Manastur; phone number:-1;  100.0$
Spending
    2014-03-24
```

A quick look over the output show that the program acts as wanted.

5. Results

An overview over the application shows a simple solution to the banking problem where we can:

- Add accounts
- Remove accounts
- Read account information
- Deposit/Withdraw money
- Transfer money
- Get Account log
- Collect money from accounts

SavingsAccounts and SpendingAccounts are clearly distinguished and can be viewed as abstract accounts.

6. Conclusions, what has been learned, further developments

As further developments we can mention the following:

- Client interface
- User login
- Monthly log
- Better graphics

7. Bibliography

http://en.wikipedia.org/wiki/Bank_account
http://en.wikipedia.org/wiki/Savings_account
http://en.wikipedia.org/wiki/Transactional_account
http://www.investopedia.com/terms/s/savingsaccount.asp
http://www.gnu.org/software/mit-scheme/documentation/mit-scheme-ref/Basic-Hash-Table-Operations.html