Tema 4

Aplicatie bancara

Proiect realizat de

Bursuc Andrei

Seria A, Grupa 30224

Contents

[**1.** **Cerinte functionale** 3](#_Toc481619029)

[**2.** **Obiectivul temei** 3](#_Toc481619030)

[**3.** **Analiza problemei, modelare, scenarii, cazuri de utilizare** 4](#_Toc481619031)

[**4.** **Proiectare** 4](#_Toc481619032)

[I. Decizii de proiectare 4](#_Toc481619033)

[II. Diagrama UML 5](#_Toc481619034)

[III.Proiectarea claselor și relațiile dintre ele 6](#_Toc481619035)

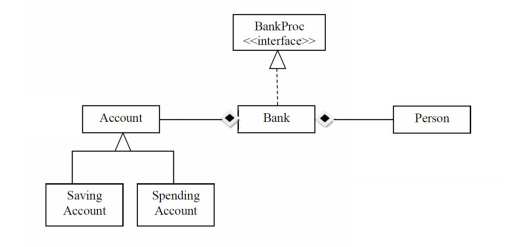
[IV. Interfață utilizator 19](#_Toc481619036)

[**5.** **Concluzii si dezvoltari ulterioare** 20](#_Toc481619037)

[**6.** **Bibliografie** 21](#_Toc481619038)

# **Cerinte functionale**

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.

# **Obiectivul temei**

Acest proiect urmareste realizarea unei aplicatii bancare care are ca scop gestionarea conturilor clientilor existenti ai bancii, dar si posibilitatea adaugarii de clienti noi. Programul trebuie sa ofere bancii o interfata prin care angajati pot: adauga clienti noi, adauga conturi noi pentru clientii deja existenti, aduga bani in conturile clientilor si updata dobanda bancara. De asemenea un client trebuia sa poata vedea conturile pe care le detine la banca, dar si extrage bani din acele conturi, atunci cand un cont este modificat sau adaugat de catre banca el va fi notificat imediat. Clientii dar si angajatii bancii au acces la aceste functii doar dupa ce se vor loga cu username-ul si parola date.

# **Analiza problemei, modelare, scenarii, cazuri de utilizare**

Problema pe care o ridica aceasta aplicatie este data de modul in care permitem angajatilor bancii sa acceseze lista de clientii si conturile aferente acestora intr-un mod cat mai eficient fiind necesar ca ei sa vada in timp real schimbarile. In acelasi timp o alta problematica o ridica modul in care clientii o sa-si vada conturile si o sa poata fi notificati in timp real in legatura cu faptul ca s-a efectat o actiune asupra unui cont sau a fost creat un cont nou.

La deschiderea programului utiizatorul va fi intampinat de o fereastra de logare si de doua optiuni, logare fie ca administrator, fie ca si client.

In cazul in care este vorba de un angajat al bancii acesta se va putea loga cu un username si o parola standard si accesa interfata specifica. De aici el va putea adauga un client nou cu un username unic si un cont initial de un tip ales de clientul ce apeleaza la angajat pentru a isi crea contul. Exista doua tipuri de conturi „Saving Account si „Spending Account”, in cazul primului tip in cont se pot doar depune o suma de bani ce va creste in functie de dobanda curenta, dar nu se poate retrage decat intreaga suma de bani din cont, dupa care acel cont va fi inchis. Pentru al doilea tip de cont se pot adauga oricand adauga bani in limita a unei valori de 1000000 o data, in acelasi timp clientul poate oricand extrage bani, iar acest cont nu se va inchide cand va ramane fara fonduri.

In cazul in care un client se va loga, mai intai daca s-au efectuat modificari asupra contului el va fi notificat in legatura cu acestea ca mai apoi sa poate accesa functiile din interfata. Clientul poate sa-si vada conturile curente intr-un tabel si singura actiune pe care o poate efectua direct este aceea de extragere de fonduri, contul actualizandu-se in tabel atunci cand acesta extrage o suma de bani. De mentionat este faptul ca pot exista mai multi clienti logati in acelasi timp, fara a exista vreun fel de conflict intre acestia.

Datele programului sunt salvate intr-un fisier serializabil, modificarile fiind salvate de fiecare data cand un client sau administratorul se delogheaza. Astfel la deschiderea programului modificarile realizate anterior asupra continutului o sa ramana salvate, astfel evitandu-se utilizarea unei baze de date pentru a stoca obiectele din aplicatie.

# **Proiectare**

## Decizii de proiectare

Pentru realizarea aplicației am optat pentru utilizarea arhitecturii MVC (Model ). Ideea este separarea interfeței utilizator într-un Model, un View (vedere, vizualizare) (creează afişajul, interacționând cu Modelul după nevoi), şi un Controller (răspunde la cererile utilizatorului, interacționând atât cu Vizualizarea cât şi cu Controlorul după nevoi.

Arhitectură:

• **Model** -Această parte a controlatorului manipulează operațiunile logice și de utilizare a informației (trimisă dinainte de către rangul său superior), în cazul acestei probleme modelul este reprezentat de: clasa Person ce defineste clientii, clasa abstracta Account ce defineste conturile clientilor, clasele ce extind Account, SpendingAccount si SavingAccount ( ele definesc fiecare in parte tipurile conturilor), interfata BankProc si clasa Bank ce o implementeaza (ea defineste banca cu clientii si conturile lor) si interfata IEL cu clasa EventListener ce monitorizeaza actiunile din program si notifica clientii.

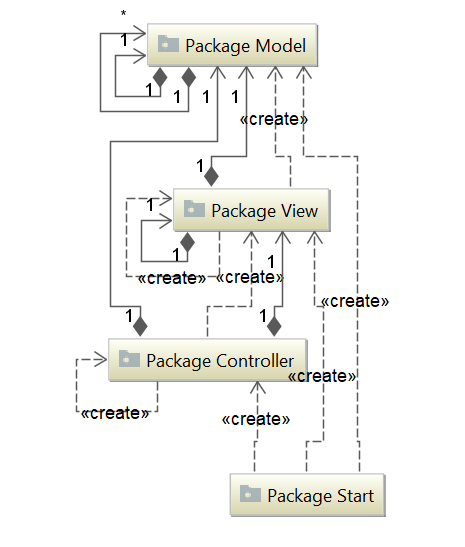
• **View**- Acestui membru al familiei îi corespunde reprezentarea grafică, sau mai bine zis, exprimarea ultimei forme a datelor: interfața grafică ce interacționează cu utilizatorul final. Rolul său este de a evidenția informația obținută până ce ea ajunge la controller. Interfața a fost realizată cu ajutorul bibliotecii Javax.Swing utilizând un plugin special pentru IntelIJ denumit JFormDesigner, ce permite realizarea mai ușoară a unei interfețe atractive pentru utilizator și ușor de folosit. Din View in cazul nostru fac parte clasele: ClientView, BankView si StartingPanel care definesc cele trei ferestre ale programului.

• **Controller** - Cu acest element putem controla accesul la aplicația noastră. Pot fi fișiere, scripts sau programe, in general orice tip de informație permisă de interfață. În acest fel putem diversifica conținutul nostru de o formă dinamică și statică, în același timp. În cazul aplicației de față constructorul controller-ului primeste ca parametrii trei obeicte de tipul: Bank, StartingPanel si listener si are rolul de a lega intre ele ferestrele din program si de a permite listener-ului sa fie transmis la celelalte obiecte din controller.

Codul a fost scris si compilat utilizând IDE-ul IntelIJ IDEA, mediu de dezvoltare software dezvoltat de catre compania JetBrains ( cunoscută anterior sub numele de IntelIJ ). El vine in doua variante ”Community Edition” și ”Ultimate Edition”, aceasta din urmă fiind disponibilă gratis studenților. Prima versiune a IntelIJ a fost lansată in Ianuarie 2001 și a fost unul dintre primele IDE-uri pentru java care dispunea de funcții avansate de ” Code Navigation” și ”Code Refactoring”. Ultima versiune 12.1 dispune de suport pentru Java 8, posibilitatea de dezvoltare de interfață grafică pentru Android, Play 2.0 și Scala.

## Diagrama UML

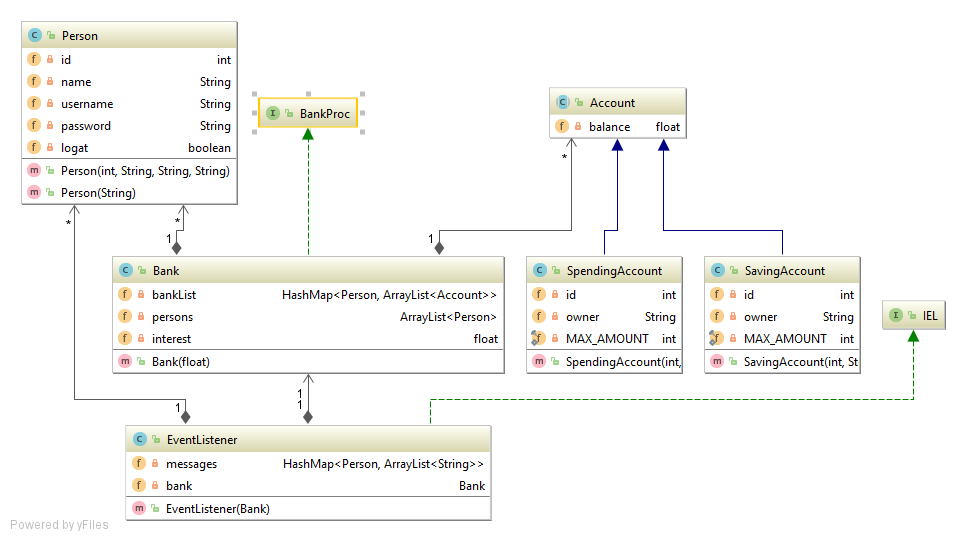
In continuare este prezentata diagrama UML a pachetelor folosite in program iar mai apoi diagrama uml a fiecarui pachet in parte:



## III.Proiectarea claselor și relațiile dintre ele

Proiectul fiind alcatuit din mai multe pachete vom prezenta pe rand fiecare pachet cu diagrama UML corespunzatoare.

**Model**



***Interfete Model***

public interface **BankProc**

Interfata BankProc defineste principalele metode ce vor fi mai departe implementate de clasa Bank.

**Method Detail**

#### addPerson

* void addPerson([Person](file:///C:\doc\Model\Person.html) person,

[Account](file:///C:\doc\Model\Account.html) account)

#### removePerson

void removePerson([Person](file:///C:\doc\Model\Person.html) person)

#### addAccount

* void addAccount([Person](file:///C:\doc\Model\Person.html) person,

[Account](file:///C:\doc\Model\Account.html) account)

public interface **IEL**

Interfata este folosita pentru a defini diveriti listeneri ce sunt folosit pentru a monitoriza modificarile efectuate asupra obiectelor din clasa.

**Method Detail**

#### notifyClient

* void notifyClient([Person](file:///C:\doc\Model\Person.html) person,

java.lang.String message)

#### getMessages

java.util.HashMap<[Person](file:///C:\doc\Model\Person.html),java.util.ArrayList<java.lang.String>> getMessages()

#### addPerson

* void addPerson([Bank](file:///C:\doc\Model\Bank.html) bank,

[Person](file:///C:\doc\Model\Person.html) person)

***Clase Model***

public abstract class **Account**

extends java.lang.Object

implements java.io.Serializable

Clasa abstracta Account este folosita pentru a defini conturile clientilor, ea contine metodele comune celor doua tipuri de conturi ce extind clasa. Principalul atribut al clasei este variabila de tip float balance ce va fi setata prin metoda setBalance de clasele ce extind aceasta clasa abstracta.

**Method Detail**

#### addMoney

public abstract void addMoney(float amount)

#### withdrawMoney

public abstract void withdrawMoney(float amount)

#### getBalance

public float getBalance()

#### setBalance

public void setBalance(float balance)

Seteaza valoarea atributului balance din Account

**Parameters:**

balance- - trebuie sa fie mai mare ca 0.

**Throws:**

IllegalArgumentException- - daca valoarea balance este mai mica decat 0.

public class **SavingAccount**

extends [Account](file:///C:\doc\Model\Account.html)

Aceasta clasa defineste conturile de tip Saving Account ea extinzand clasa de tip Account si implementand metodele de acolo.Clasa SavingAccount este definita de varibailele id de tip int si owner de tip String.

**Method Detail**

#### addMoney

public void addMoney(float amount)

**Specified by:**

[addMoney](file:///C:\doc\Model\Account.html#addMoney-float-) in class [Account](file:///C:\doc\Model\Account.html)

**Parameters:**

amount -

**Throws:**

java.lang.IllegalArgumentException - -pentru obiectele de tip obiectele de tipul SavingAccount metoda addMoney arunca o exceptie pentru ca bani se pot adauga intr-un astfel de cont doar la inceput.

#### withdrawMoney

public void withdrawMoney(float amount)

**Specified by:**

[withdrawMoney](file:///C:\doc\Model\Account.html#withdrawMoney-float-) in class [Account](file:///C:\doc\Model\Account.html)

**Parameters:**

amount - suma ce trebuie extrasa trebuie sa fie egala cu valoarea contnutului contului.

**Throws:**

java.lang.AssertionError - daca suma ce trebuie extrasa nu e egala cu valoarea contnutului contului.

java.lang.AssertionError - daca la final valoarea contului nu este 0.

#### getId

public int getId()

#### getOwner

public java.lang.String getOwner()

public class **SpendingAccount**

extends [Account](file:///C:\doc\Model\Account.html)

Aceasta clasa defineste conturile de tip Spending Account ea extinzand clasa de tip Account si implementand metodele de acolo.Clasa SpendingAccount este definita de varibailele id de tip int si owner de tip String.

**Constructor Detail**

#### SpendingAccount

* + public SpendingAccount(int id,

java.lang.String owner)

Contructorul clasei SpendingAccount primeste ca parametrii varibilele id si owner si le seteaza.

**Parameters:**

id - pozitiv

owner -

**Method Detail**

#### addMoney

public void addMoney(float amount)

Creste valuarea atributului balance cu valoarea data de amount.

**Specified by:**

[addMoney](file:///C:\doc\Model\Account.html#addMoney-float-) in class [Account](file:///C:\doc\Model\Account.html)

**Parameters:**

amount - trebuie sa fie mai mare ca 0

**Throws:**

java.lang.AssertionError - daca amount este mai mic ca 0 sau mai mare decat valoarea maxima data

java.lang.AssertionError - daca valoarea finala a contului nu este egala cu valoarea anterioara + amount

#### withdrawMoney

public void withdrawMoney(float amount)

Scade valoarea balance cu suma data.

**Specified by:**

[withdrawMoney](file:///C:\doc\Model\Account.html#withdrawMoney-float-) in class [Account](file:///C:\doc\Model\Account.html)

**Parameters:**

amount - trebuie sa fie mai mica decat suma curenta din cont si mai mare ca 0

**Throws:**

java.lang.AssertionError - daca valoare este mai mare ca 0 sau negativa

java.lang.AssertionError - daca la final valoarea balance nu este egala cu valoarea anterioara - amount

public class **Person**

extends java.lang.Object

implements java.io.Serializable

Aceasta clasa defineste obiectele de tip Person, practic este modelul unui client ce utilizeaza aplicatia, fiind definita de atributele: id,name,username,password si logat.

**Constructor Detail**

#### Person

* + public Person(int id,
  + java.lang.String name,
  + java.lang.String username,

java.lang.String password)

#### Person

public Person(java.lang.String username)

**Method Detail**

#### hashCode

public int hashCode()

Avem nevoie de aceasta metoda de hashCode deoarece obiectele te tip persoana sunt folosite ca si chei in Map-ul bankList din clasa Bank. Cheia este deifinta ca valoarea primului caracter din username inmultit cu valoarea celui de al doilea inmultit cu lungimea username-ului.

**Overrides:**

hashCode in class java.lang.Object

**Returns:**

#### equals

public boolean equals(java.lang.Object object)

Doua obiecte de tip Person sunt egale daca au acelasi username.

**Overrides:**

equals in class java.lang.Object

public class **EventListener**

extends java.lang.Object

implements [IEL](file:///C:\doc\Model\IEL.html)

Clasa EventListener implementeaza metodele notifyClient si addPerson ea jucand rol de obiect care monitorizeaza toate schimbarile din conturile clientilor. Daca acesta e ogat va primi notifcarea imediat daca nu, o va primi la logare.

**Constructor Detail**

#### EventListener

public EventListener([Bank](file:///C:\doc\Model\Bank.html) bank)

Initializeaza map-ul messages.

**Parameters:**

bank –

**Method Detail**

#### notifyClient

* + public void notifyClient([Person](file:///C:\doc\Model\Person.html) person,

java.lang.String message)

Metoda notifica clientul in timp real daca s-a efectua o modificare asupra contului, in caz contrar va fi notifcat la logare. Aceasta metoda se apeleaza tot timpul cand administratorul efectueaza o modificare asupra contului.

**Specified by:**

[notifyClient](file:///C:\doc\Model\IEL.html#notifyClient-Model.Person-java.lang.String-) in interface [IEL](file:///C:\doc\Model\IEL.html)

**Parameters:**

person -

message -

#### getMessages

public java.util.HashMap<[Person](file:///C:\doc\Model\Person.html),java.util.ArrayList<java.lang.String>> getMessages()

**Specified by:**

[getMessages](file:///C:\doc\Model\IEL.html#getMessages--) in interface [IEL](file:///C:\doc\Model\IEL.html)

#### addPerson

* + public void addPerson([Bank](file:///C:\doc\Model\Bank.html) bank,

[Person](file:///C:\doc\Model\Person.html) person1)

Adauga persoana data in map-ul messages.

**Specified by:**

[addPerson](file:///C:\doc\Model\IEL.html#addPerson-Model.Bank-Model.Person-) in interface [IEL](file:///C:\doc\Model\IEL.html)

**Parameters:**

bank -

person1 –

public class **Bank**

extends java.lang.Object

implements [BankProc](file:///C:\doc\Model\BankProc.html), java.io.Serializable

Aceasta clasa defineste banca si contine un map in care fiecarei persoane ii sunt atribuite o lista de conturi, dar si o lista de persoane si o dobanda.

**Constructor Detail**

#### Bank

public Bank(float interest)

Constructorul initializeaza map-ul, lista de persoane si seteaza interest-ul.

**Parameters:**

interest - pozitiv

**Throws:**

java.lang.AssertionError - daca valoarea interest este negativa

**Method Detail**

#### writeObject

public void writeObject()

Aceasta metoda este folosita pentru a salva in fisierul employee.ser obiectul de tip bank. Astfel evtam folosirea unei baze de date.

#### addPerson

* + public void addPerson([Person](file:///C:\doc\Model\Person.html) person,

[Account](file:///C:\doc\Model\Account.html) account)

Adauga in BankList person si accout si adauga in lista persons obiectul person

**Specified by:**

[addPerson](file:///C:\doc\Model\BankProc.html#addPerson-Model.Person-Model.Account-) in interface [BankProc](file:///C:\doc\Model\BankProc.html)

**Parameters:**

person - persoana nu trebuie sa existe deja in lista de persoane

account - contul ce urmeaza adaugat

**Throws:**

java.lang.AssertionError - daca persoana exista deja

java.lang.AssertionError - daca la final se constata cu metoda contains ca obiectul person nu a fost adaugat

#### removePerson

public void removePerson([Person](file:///C:\doc\Model\Person.html) person)

Sterge persoana din bankList si persons.

**Specified by:**

[removePerson](file:///C:\doc\Model\BankProc.html#removePerson-Model.Person-) in interface [BankProc](file:///C:\doc\Model\BankProc.html)

**Parameters:**

person - persoana trebuie sa existe

**Throws:**

java.lang.AssertionError - daca persoana nu exista

java.lang.AssertionError - daca persoana nu a fost scoasa

#### addAccount

* + public void addAccount([Person](file:///C:\doc\Model\Person.html) person,

[Account](file:///C:\doc\Model\Account.html) account)

Adauga contul in map unde cheia este o persoana data

**Specified by:**

[addAccount](file:///C:\doc\Model\BankProc.html#addAccount-Model.Person-Model.Account-) in interface [BankProc](file:///C:\doc\Model\BankProc.html)

**Parameters:**

person - persoana trebuie sa existe in map-ul bankList

account -

**Throws:**

java.lang.AssertionError - daca persoana nu exista

java.lang.AssertionError - daca contul nu a fost adaugat in lista

#### setInterest

public void setInterest(float interest)

Seteaza parametrul interest.

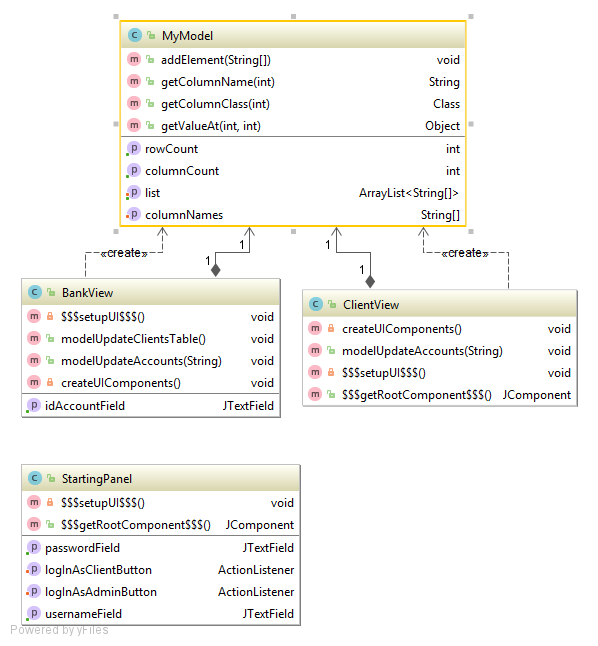
**Parameters:**

interest - trebuie sa fie pozitiv

**Throws:**

java.lang.AssertionError - daca interest-ul nu e pozitiv

**View**



public class **BankView**

extends javax.swing.JFrame

O parte din codul pentru aceasta clasa a fost generat folosindu-se plugin-ul special din IntelIJ IDEA, ea reprezinta interfata prin care adiministratorul poate adauga clienti noi, adauga conturi noi pentru clienti sau adauga bani in conturile clientilor, de asemenea de aici dobanda se poate schimba. La fiecare modificare efectuata asupra contului de catre administrator clientul daca nu este deja logat va fi notificat la prima logare in legatura cu faptul ca o anumita actiune a fost efectuata asupra contului sau.

**Constructor Detail**

#### BankView

* + public BankView([Bank](file:///C:\doc\Model\Bank.html) bank,

[IEL](file:///C:\doc\Model\IEL.html) listener)

Constructorul primeste ca parametrii un obiect de tip Bank si unul de tip Listener, de asemenea defineste listenerii pentru cele doua tabele din interfata si seteaza listenerii pentru butoanele din fereastra.

**Parameters:**

bank -

listener -

**Method Detail**

#### getIdAccountField

public javax.swing.JTextField getIdAccountField()

#### modelUpdateClientsTable

public void modelUpdateClientsTable()

Aceasta metoda preia datele din lista de persoane din obiectul bank si le adauga intr-un ArrayList de tip String[], dupa seteaz in model ArrayList-ul obtinut si astfel se updateza tabelul.

#### modelUpdateAccounts

public void modelUpdateAccounts(java.lang.String username)

Aceasta metoda preia datele din map-ul bankList, din ArrayList-ul de Account din obiectul bank si le adauga intr-un ArrayList de tip String[]. Dupa seteaza in model noul ArrayList obtinut si astfel se updateaza tabelul in functie de clientul selectat in primul tabel.

public class **ClientView**

extends javax.swing.JFrame

O parte din codul pentru aceasta clasa a fost generat folosindu-se plugin-ul special din IntelIJ IDEA, ea reprezinta interfata prin care clientul poate sa-si consulte conturile si sa extraga bani din conturile lui.

**Constructor Detail**

#### ClientView

* + public ClientView([Bank](file:///C:\doc\Model\Bank.html) bank,
  + java.lang.String username,

[IEL](file:///C:\doc\Model\IEL.html) listener)

Constructorul seteaza un listener pe tabelul ce arata conturile, iar apoi seteaza listenerii pe cele doua butoane.

**Parameters:**

bank -

username -

listener -

**Method Detail**

#### modelUpdateAccounts

public void modelUpdateAccounts(java.lang.String username)

Aceasta metoda preia datele din map-ul bankList, din ArrayList-ul de Account din obiectul bank si le adauga intr-un ArrayList de tip String[]. Dupa seteaza in model noul ArrayList obtinut si astfel se updateaza tabelul in functie de situatia conturilor clientului.

**Parameters:**

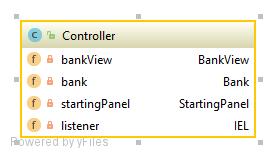
username –

public class **StartingPanel**

extends javax.swing.JFrame

Cea mai mare parte din codul pentru aceasta clasa a fost generat de plogin-ul special din IntelIJ ea definind fereastra care apare la deschiderea programului din care se pot loga atat clientii cat si adminul.

**Controller**



public class **Controller**

extends java.lang.Object

Clasa Controller realizeaza legaturile intre obiectele de tip StartingPanel, Bank si View si permite existena mai multor clienti logati in acelasi timp si posibilitatea lor de a fi notificati cand o actiune se intreprinde asupra unui cont.

**Constructor Detail**

#### Controller

* public Controller([Bank](file:///C:\doc\Model\Bank.html) bank,
* [StartingPanel](file:///C:\doc\View\StartingPanel.html) startingPanel,

[IEL](file:///C:\doc\Model\IEL.html) listener)

Contructorul clientului primeste ca parametrii trei obiecte de tipul: Bank, StartingPanel si IEL, seteaza aceste obiecte in atributele controller-ului, aplica listenerii pe cele doua butoane si face fereastra StartingPanel vizibila

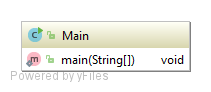
**Parameters:**

bank -

startingPanel -

listener –

**Main**



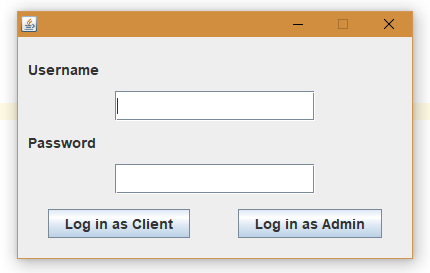
public class **Main**

extends java.lang.Object

Din clasa main citim obiectul de tip Bank din fisierul employee.ser, initializam obiectele de tip StartingPanel si IEL si instantiam Controllerul ce va afisa interfata programului.

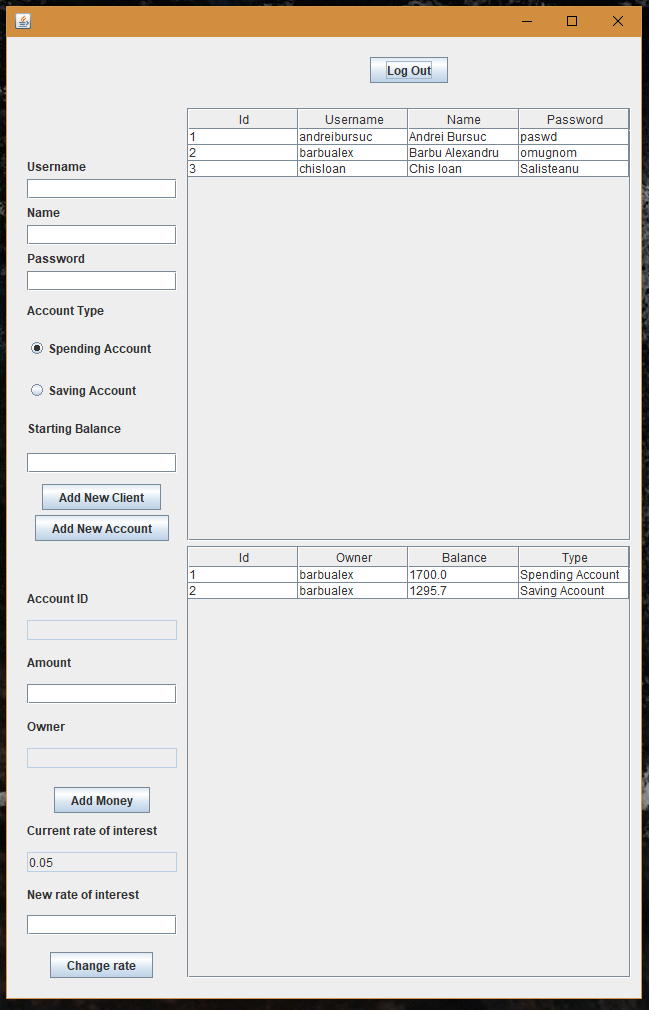
## IV. Interfață utilizator

La deschiderea programului prima fereastra cu care utilizatorul va interactiona va fi cea de logare:



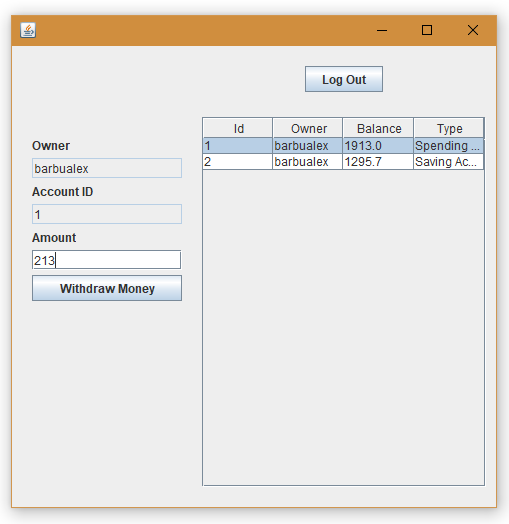
De aici utilizatorul se poate loga ca si client sau administrator, in cazul in care au fost introduse username-ul sau parola gresite utilizatorul va fi intampinat cu mesaje de eroare.

Daca te loghezi ca si administrator se va deschide urmatoarea fereastra:



De aici administratorul poate adauga clienti noi, crea conturi noi pentru clienti, adauga bani in conturile clientilor si modifica dobanda. Atunci cand dai click pe un client din tabelul de sus conturile sale vor fi afisate in tabelul de jos. Daca vrei sa adaugi un cont nou unui client acesta trebuie selectat in tabelul de sus, daca vrei sa adaugi bani intr-un cont el trebuie selectat din tabelul de jos.

Daca te loghezi ca si client urmatoarea interfata va fi afisata:



Daca in perioada in care clientul nu a fost logat s-au efectuat modificari asupra conturilor existente sau au fost adaugate conturi noi inainte de aparitia ferestrei, clientul va fi notificat in legatura cu aceste notificari.

Din aceasta interfata un client isi vede conturile existente si poate retrage bani din acestea.

# **Concluzii si dezvoltari ulterioare**

Aceasta aplicatie poate fi folosita pentru gestionarea activelor si conturilor clinetilor unor banci in asa fel incat sa nu mai fie nevoie de salvarea evidentei acestora in format fizic, iar clientul sa isi poata consulta contul si retrage bani mult mai usor.

Dezvoltarea acestei aplicatii m-a ajutat sa inteleg mai bine modul in care putem salva elementele unui obiect intr-un fisier de tip Serializable, aceasta metoda fiind o metoda mai rapida si mai usoara decat utilizarea unei baze de date. In dezvoltarea programului am avut ocazia sa invat mai multe despre modul in care putem crea o clasa speciala care sa monitorizeze modificarile dintr-o clasa si sa notifice un utilizator al aplicatiei despre aceste modificari, dar si despre listenerii pentru tabele si modul in care ii putem folosi pentru a face interfata mai atractiva si usor de folosit.

Posibile dezvoltari ulterioare ale programului ar fi prezenta unei descreri mai detaliate a unui client ( date legate despre adresa CNP, numar de telefon, email, adresa etc), posibilitatea de a sterge un client, posibilitatea de a modifica datele personale a unui client, posibilitatea administratorului de a transmite notificari personalizate ( cu reclame de exemplu) la unii sau la toti clientii, dar si posibilitatea unui client de a depune personal bani in cont fara sa fie nevoia de interventia unui administrator.

# **Bibliografie**

1. <http://docs.oracle.com/javase/>
2. <http://stackoverflow.com/>
3. <http://www.tutorialspoint.com/java/java_serialization.htm>
4. <http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html>
5. http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html
6. Kathy Sierra, Bert Bates. SCJP Sun Java Certified Programmer for Java 6. McGraw Hill, USA 2008
7. <https://dzone.com/articles/layers-standard-enterprise>