Aplicatie pentru procesarea conturilor dintr-o banca

Universitatea Tehnică Cluj-Napoca

Facultatea de Calculatoare și Automatică

Departamenul de Calculatoare și Tehnologia informației

Nume: Muresan George

Grupa: 30226

Materie: Tehnici de programare

1.Obiectivul temei

Cerinta proiectului este: Procesarea conturilor dintr-o banca.

Obiectivul temei este de a implementa o aplicatie cu interfata grafica pentru a fi folosita ca aplicatie ce foloseste serializarea si hash-map-ul.

Java pune la dispozitie conceptul de serializare. Acesta poate fi definit astfel: reprezentarea unei entitati sub forma unei secvente de octeti, cu ascunderea diverselor detalii, precum:

* reprezentarile particulare ale datelor pe diferitele platforme ([byte ordering](http://en.wikipedia.org/wiki/Endianness) etc.). Acest aspect se dovedeste util mai ales in cazul transmisiei pe retea, cand cele doua capete ale canalului de comunicatie ruleaza pe sisteme de operare sau arhitecturi diferite.
* structurile arbitrar de complexe ale obiectelor: acestea pot contine variabile membru de tip referinta, vectori de orice adancime etc. Cu exceptia cazului in care programatorul doreste sa defineasca explicit o schema particulara de serializare, sistemul poate asigura serializarea automata a intregii componenţe a obiectelor, explorand recursiv structura acestuia. Graful de referinte obtinut poarta numele de web of objects.

Serializarea este utilizata pentru implementarea conceptului de persistenta, vazuta drept capacitate a unui obiect de a supravietui dupa incheierea executiei programului. Acest deziderat se obtine prin depozitarea informatiilor acestuia si recuperarea lor la urmatoarea executie. Desigur, acelasi efect se poate reproduce prin stocarea explicita a componentelor unui obiect in fisiere sau baze de date, dar serializarea reprezinta un mecanism uniform, aplicabil asupra oricarui tip de obiecte.

În computere, o tabelă hash (hartă hash) este o structură de date care implementează un tip de date abstract de matrice asociativă, o structură care poate mapa cheile către valori. O tabelă hash utilizează o funcție hash pentru a calcula un index într-o gamă de găleți sau sloturi, din care se poate găsi valoarea dorită. În mod ideal, funcția hash va atribui fiecărei chei unei găleți unice, dar cele mai multe modele de tabele hash utilizează o funcție hash imperfectă, care ar putea cauza ciocniri de hash în care funcția hash generează același index pentru mai multe chei.

Asemenea coliziuni trebuie să fie adăpostite într-un fel. Într-o tabelă hash bine dimensionată, costul mediu (numărul de instrucțiuni) pentru fiecare căutare este independent de numărul de elemente stocate în tabel. Multe modele de tabele hash permit, de asemenea, introducerea și ștergerea arbitrară a perechilor cheie-valoare, la costul mediu constant [amortizat [2]) pe operație. [3] [4] În multe situații, tabelele hash se dovedesc a fi mai eficiente decât arborii de căutare sau orice altă structură de căutare a tabelului. Din acest motiv, ele sunt utilizate pe scară largă în numeroase tipuri de programe informatice, în special pentru matricea asociativă, indexarea bazei de date, cache-uri și seturi.

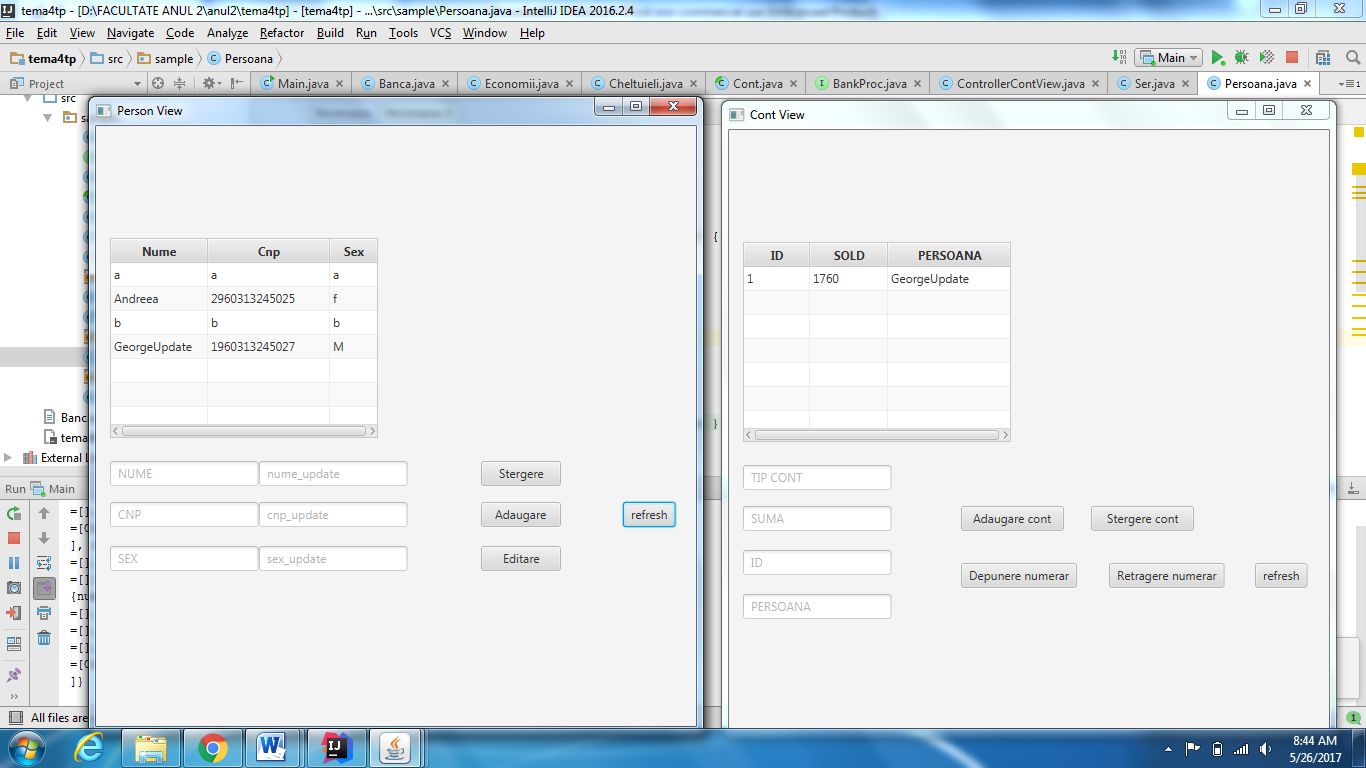
2.a) Analiza problemei

În partea de analiză a problemei trebuie depistate principalele clase, respectiv caracteristicile și funcționlitățile acestora și realizate legăturile dintre ele dar si implementarea opeartiilor. Programarea orientata pe obiecte oferă avantajul de a putea începe dezvoltarea unui proiect folosind doar informațiile de la suprafață, fără a fi nevoie de implementarea efectivă a funcționalităților. Această strategie este cunoscută sub numele “Top-Down”. Ea este foarte avantajoasă din punctul de vedere al găsirii componentelor constituente, deoarece pot fi găsite, relativ ușor, structuri cu o legătură directă în lumea reală( obiecte, acțiuni etc.). Din păcate această versatilitate vine cu prețul complexității, ea crescând spre măsură ce se avansează pe nivelele inferioare.In subiectul de fata trebuie sa analizam rezolvarea procesarii conturilor dintr-o banca.

2.b) Modelarea

In partea de modelare interfața cu utilizatorul este stratul cel mai de deasupra al unei aplicații, de aceea realizarea ei a reprezentat punctul de pornire al acestui proiect. În primul rând, ea trebuie să fie cât mai intuitivă și permisivă în ceea ce privește modul de introducere a datelor.

Dupa cum putem observa, datele de intrare si de iesire reies foarte bine din imaginea de mai jos.



2.c) Scenarii

Scenariile care pot sa apară și sa ducă la oprirea din funcționare a aplicatiei sau eventual la probelme de implementare pot fii:

* Serializarea si deserializarea facuta incorect
* Introducerea incorecta a datelor
* O pana de current

2.d) Cazuri de utilizare

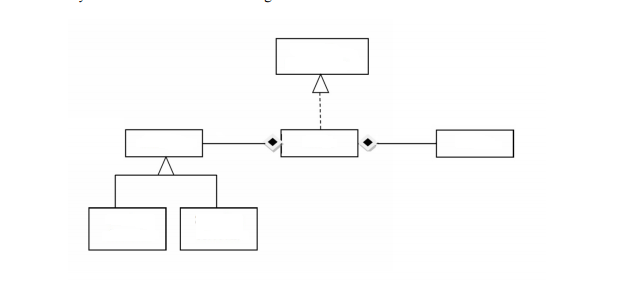
Aplicația se poate folosi pentru:

* Gestionarea conturilor dintr-o banca

3.Proiectare

**Interfața grafică** este realizată cu JavaFX Scene Builder gasit in IntelliJ(una dintre ele), iar cealalta este realizata manual. Acesta generează un fișier .fxml care este introdus în proiect. ActionEvent-urile ale fiecărui buton sunt setate în JavaFX Scene Builder prin introducerea în câmpul “On Action” numele metodei care trebuie apelata și id-ul Text field-urilor, în câmpul “id”.

Dupa cum puteti observa, mai jos este reprezentata **diagrama UML** a claselor .



CHELTUIELI

ECONOMII

BankProc

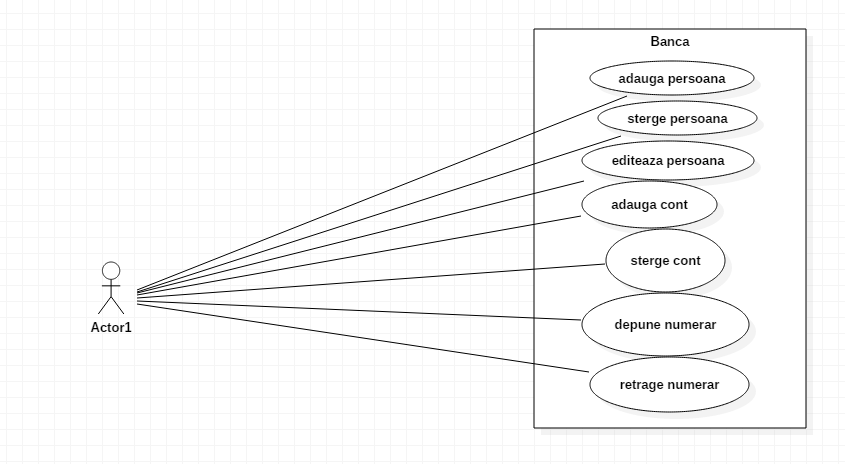
<<interface>>

PERSOANA

BANCA

CONT

Diagrama **use case:**



4. Implementare si testare

Aplicatia mea foloseste 11 clase, fiecare dintre ele findu-mi necesara pentru implementarea gestionarii conturilor si a persoanelor dintr-o banca.

Clasa **Persoana** este clasa ce defineste proprietatile unei persoane.

**public class** Persoana **implements** Serializable {  
 **public** String **nume**;  
 **public** String **cnp**;  
 **public** String **sex**;  
  
 **public** Persoana(String nume, String cnp, String sex) {  
 **this**.**nume** = nume;  
 **this**.**cnp** = cnp;  
 **this**.**sex** = sex;  
 }  
 **public** Persoana() {  
  
 }  
  
 **public** String getNume() {  
 **return nume**;  
 }  
  
 **public void** setNume(String nume) {  
 **this**.**nume** = nume;  
 }  
  
 **public** String getCnp() {  
 **return cnp**;  
 }  
  
 **public void** setCnp(String cnp) {  
 **this**.**cnp** = cnp;  
 }  
  
 **public** String getSex() {  
 **return sex**;  
 }  
  
 **public void** setSex(String sex) {  
 **this**.**sex** = sex;  
 }  
  
 **public** SimpleStringProperty getNumeP(){  
 SimpleStringProperty numeP=**new** SimpleStringProperty(**nume**);  
 **return** numeP;  
 }  
 @Override  
 **public boolean** equals(Object obj) {  
 **if** (**this** == obj) **return true**;  
 **if** (obj == **null**) **return false**;  
 **if** (getClass() != obj.getClass()) **return false**;  
 Persoana other = (Persoana) obj;  
 **if** (**cnp** == **null**) {  
 **if** (other.**cnp** != **null**)  
 **return false**;  
 } **else if** (!**cnp**.equals(other.**cnp**))  
 **return false**;  
 **return true**;  
 }  
  
 **public** String toString(){  
 **return "numele :"**+ **nume** + **" "** + **" CNP: "** + **cnp** +**" sexul: "**+ **sex**+ **".\n"**;  
 }  
}

Clasa **Cont** este o clasa abstract ace implementeaza niste metode simple.

**public abstract class** Cont **implements** Serializable {  
 **public int sold**;  
 **public** Persoana **p1**;  
 **public int id**;  
  
 **public abstract int** getId() ;  
  
  
 **public abstract void** setId(**int** id) ;  
  
  
 **public abstract int** getSold() ;  
  
  
 **public abstract void** setSold(**int** sold) ;  
  
  
 **public abstract** String getP1() ;  
  
 **public abstract** Persoana getTit();  
  
  
 **public abstract void** setP1(Persoana p1) ;  
  
 **public** SimpleStringProperty getNumeP(){  
 **return p1**.getNumeP();  
 }

Clasa **Ser** este folosita doar pentru serializare.

**public class** Ser  
{  
 **public static** Banca importb() **throws** IOException, ClassNotFoundException {*//deseriliazre* Banca b;  
  
 FileInputStream fisierConturi = **new** FileInputStream(**"Banca.ser"**);  
 ObjectInputStream o = **new** ObjectInputStream(fisierConturi);  
 b = (Banca) o.readObject();  
 o.close();  
 fisierConturi.close();  
  
 **return** b;  
 }  
  
  
 **public static void** exportb(Banca b) **throws** IOException {  
  
 FileOutputStream f = **new** FileOutputStream(**"Banca.ser"**);  
 ObjectOutputStream o = **new** ObjectOutputStream(f);  
 o.writeObject(b);  
  
 o.close();  
 f.close();  
  
 }  
}

Clasa **Economii** este o clasa de tipul cont care mai are 2 metode, retragere numerar si depunere numerar.

**public class** Economii **extends** Cont **implements** Serializable {  
  
 **private int dobanda**=10;  
  
  
 **public void** adaugareBani (**int** s)  
 {  
 **try** {  
 **if**(s>1000)  
 {  
 **sold**+=s;  
 **int** intermediar=(**sold**\***dobanda**)/100;  
 **sold**+=intermediar;  
 System.***out***.println(**"au fost adaugati bani in economii "**+**sold**);  
 }  
  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
 System.***out***.println(**"Suma de adaugat este prea mica"**);  
 }  
  
  
 }  
 **public void** retragereBani(**int** s)  
 {  
 **try** {  
 **if**(s<=**sold**)  
 {  
 **sold**-=s;  
 }  
  
 }**catch** (Exception e)  
 {  
 e.printStackTrace();  
 System.***out***.println(**"nu s-au putut retreage bani din contul de economii"**);  
 }  
  
 }  
  
  
 **public** Economii(Persoana p1) {  
 **this**.**p1**=p1;  
 }  
  
 @Override  
 **public int** getId() {  
 **return id**;  
 }  
  
 @Override  
 **public void** setId(**int** id) {  
 **this**.**id**=id;  
  
 }  
  
 @Override  
 **public int** getSold() {  
 **return sold**;  
 }  
  
 @Override  
 **public void** setSold(**int** sold) {  
 **this**.**sold**=sold;  
 }  
  
 @Override  
 **public** String getP1() {  
 **return p1**.getNume();  
 }  
  
 **public** Persoana getTit(){**return p1**;}  
  
 @Override  
 **public void** setP1(Persoana p1) {  
 **this**.**p1**=p1;  
  
 }  
  
 **public** String toString(){  
 **return "Cont Economii, sold: "** + **sold** + **"---idul"**+**id**+ **"\n"**;  
 }  
}

Clasa **Cheltuieli** este tot o clas de tipul Cont si implementeaza la fel doar 2 metode noi .

**public class** Cheltuieli **extends** Cont **implements** Serializable{  
  
 **public** Cheltuieli(Persoana p1) {  
 **this**.**p1**=p1;  
 }  
 *// public Cheltuieli()  
 //{  
  
 // }* **public void** adaugareBani (**int** s)  
 {  
 **try** {  
 **if**(s>0)  
 {  
 **sold**+=s;  
  
 System.***out***.println(**"au fost adaugati bani in cheltuieli"**+**sold**);  
 }  
  
 }**catch**(Exception e)  
 {  
 e.printStackTrace();  
 System.***out***.println(**"Suma de adaugat este prea mica"**);  
 }  
  
  
 }  
 **public void** retragereBani(**int** s)  
 {  
 **try** {  
 **if**(s<=**sold**)  
 {  
 **sold**-=s;  
 }  
  
 }**catch** (Exception e)  
 {  
 e.printStackTrace();  
 System.***out***.println(**"nu s-au putut retreage bani din contul de economii"**);  
 }  
  
 }  
  
 @Override  
 **public int** getId() {  
 **return id**;  
 }  
  
 @Override  
 **public void** setId(**int** id) {  
 **this**.**id**=id;  
 }  
  
 @Override  
 **public int** getSold() {  
 **return sold**;  
 }  
  
 @Override  
 **public void** setSold(**int** sold) {  
 **this**.**sold**=sold;  
 }  
  
 @Override  
 **public** String getP1() {  
 **return p1**.getNume();  
 }  
  
 **public** Persoana getTit(){**return p1**;}  
  
 @Override  
 **public void** setP1(Persoana p1) {  
 **this**.**p1**=p1;  
 }  
  
 **public** String toString(){  
 **return "Cont Cheltuieli, sold: "** + **sold** + **"--idul "**+ **id** + **"\n"**;  
 }  
}

Clasa **Banca** contine un hasmap cu persoane si liste de conturi, dar si metodele de adaugare a unei persona in hash,stergere etc … si adugare de cont, stergere, etc .

**public class** Banca **implements** Serializable,BankProc{  
  
 **private** HashMap<Persoana,List<Cont>> **HBanca**;  
  
 **public** Banca(HashMap<Persoana, List<Cont>> HBanca) {  
 **this**.**HBanca** = HBanca;  
 }  
 **public** Banca() {  
 **HBanca**=**new** HashMap<>();  
 }  
  
 **public** HashMap<Persoana, List<Cont> >getHBanca() {  
 **return HBanca**;  
 }  
  
 **public void** setHBanca(HashMap<Persoana, List<Cont>> HBanca) {  
 **this**.**HBanca** = HBanca;  
 }  
  
 **public void** addPerson(Persoana p) {  
 **if**(equals(p)==**false**)  
 {  
 List<Cont> lc=**new** ArrayList<Cont>();  
 **HBanca**.put(p,lc);  
 }  
 *// System.out.println(getPersoane());* }  
  
 **public void** delPerson(Persoana p)  
 {  
 **if**(**HBanca**.containsKey(p))  
 {  
 **HBanca**.remove(p);  
 }  
 }  
  
 **public void** editPerson(Persoana p,String numen,String cnpn,String sexn)  
 {  
 List<Cont> lc=**new** ArrayList<>();  
  
 **if**(**HBanca**.containsKey(p))  
 {  
 **HBanca**.remove(p);  
 }  
 p.setCnp(cnpn);  
 p.setNume(numen);  
 p.setSex(sexn);  
 **HBanca**.put(p,lc);  
  
 }  
  
 **public** List<Persoana> getPersoane()  
 {  
 List<Persoana> listpers=**new** ArrayList<Persoana>();  
 Set<Persoana>keys=**HBanca**.keySet();  
 **for**(Persoana key:keys)  
 {  
 listpers.add(key);  
 *//System.out.println(key);* }  
 **return** listpers;  
 }  
 *//------------------------------------------------------gata cu persoanele----------------------------------------------------------* **public void** addContEco(Economii ec,Persoana p)  
 {  
 **if**(**HBanca**.containsKey(p))  
 {  
 List<Cont> listacon=**HBanca**.get(p);  
 listacon.add(ec);  
 **HBanca**.put(p,listacon);  
 }  
 }  
 **public void** addContChe(Cont che,Persoana p)  
 { System.***out***.println(**"personae inainte de if"**);  
 System.***out***.println(**HBanca**);  
 **if**(**HBanca**.containsKey(p))  
 {  
 List<Cont> listacon=**HBanca**.get(p);  
 listacon.add(che);  
 **HBanca**.put(p,listacon);  
 System.***out***.println(**"lista conturi din add banca"**);  
 System.***out***.println(listacon);  
 }  
 **else** System.***out***.println(**"nu contine persoana "**+p);  
 }  
 **public void** delCont(Persoana p,Cont c)  
 {  
 **if**(**HBanca**.containsKey(p))  
 {  
 List<Cont> listacon= **HBanca**.get(p);  
 listacon.remove(c);  
 **HBanca**.put(p,listacon);  
 }  
 }  
  
 **public void** editCont(Persoana p,**int** schimb,Cont c)  
 {  
 **if**(**HBanca**.containsKey(p))  
 {  
 List<Cont> listcon=**HBanca**.get(p);  
 **for**(**int** i=0;i<listcon.size();i++)  
 {  
 **if**( listcon.get(i)==c)  
 {  
 listcon.get(i).setId(schimb);  
 }  
 }  
 }  
 }  
  
 **public** List<Cont> getConturi()  
 {  
 List<Cont> listaconturi=**new** ArrayList<Cont>();  
 List<Cont> listcon=**new** ArrayList<Cont>();  
 **for** (Map.Entry<Persoana,List<Cont>> me:**HBanca**.entrySet())  
 {  
  
 listcon=me.getValue();  
 **for**(**int** i=0;i<listcon.size();i++)  
 {  
 listaconturi.add(listcon.get(i));  
 }  
 }  
 **return** listaconturi;  
 }  
 *//-------------------------------------------------------gata cu cu conturile--------------------------------------------------------*}

Clasa **ControllerContView** care se ocupa de fisierul fxml ContView,mai exact de interfata.

**package** sample;  
  
  
**import** javafx.beans.value.ObservableValue;  
**import** javafx.collections.FXCollections;  
**import** javafx.collections.ObservableList;  
**import** javafx.collections.ObservableMap;  
**import** javafx.fxml.FXML;  
**import** javafx.scene.control.TableColumn;  
**import** javafx.scene.control.TableView;  
**import** javafx.scene.control.TextField;  
**import** javafx.scene.control.cell.PropertyValueFactory;  
**import** javafx.util.Callback;  
  
**import** java.io.IOException;  
**import** java.util.HashMap;  
**import** java.util.List;  
  
**public class** ControllerContView {  
  
 **private** Banca **banca**;  
 **private** ObservableList<Cont> **model**;  
 Ser **s**=**new** Ser();  
  
  
  
 @FXML  
 **private** TableView<Cont> **tablecont**=**new** TableView<>();  
 @FXML  
 **private** TableColumn<Cont,Integer> **idCol**=**new** TableColumn<>();  
 @FXML  
 **private** TableColumn<Cont,Integer> **soldCol**=**new** TableColumn<>();  
 @FXML  
 **private** TableColumn<Cont,String> **persCol**=**new** TableColumn<Cont,String>(**"nume persoana"**);  
 @FXML  
 TextField **tipText**=**new** TextField();  
 @FXML  
 TextField **sumaText**=**new** TextField();  
 @FXML  
 TextField **idText**=**new** TextField();  
 @FXML  
 TextField **persText**=**new** TextField();  
  
  
 **public void** setServices() **throws** IOException, ClassNotFoundException {  
  
 **this**.**banca**=**new** Banca();  
 **this**.**model**=FXCollections.*observableList*(**banca**.getConturi());  
  
 **tablecont**.setItems(**model**);  
  
 }  
  
  
 @FXML  
 **public void** initialize(){  
 **idCol**.setCellValueFactory(**new** PropertyValueFactory<Cont, Integer>(**"id"**));  
 **soldCol**.setCellValueFactory(**new** PropertyValueFactory<Cont, Integer>(**"sold"**));  
 *//persCol.setCellValueFactory(new PropertyValueFactory<Cont, String>("persoana"));* **persCol**.setCellValueFactory(**new** Callback<TableColumn.CellDataFeatures<Cont, String>, ObservableValue<String>>() {  
 **public** ObservableValue<String> call(TableColumn.CellDataFeatures<Cont, String> p) {  
 **return** p.getValue().getNumeP();  
 }  
 });  
  
 }  
  
 **public void** handleAddC() **throws** IOException {  
  
 **if**(**tipText**.getText().equalsIgnoreCase(**"cheltuieli"**)) {  
  
 String cnps = **persText**.getText();  
 List<Persoana> listpers = **banca**.getPersoane();  
 System.***out***.println(**"lista persoane"**);  
 System.***out***.println(listpers);  
 **for** (**int** i = 0; i < listpers.size(); i++) {  
 **if** (listpers.get(i).getCnp().equalsIgnoreCase(cnps)) {  
 *//Persoana p = new Persoana(listpers.get(i).getNume(),listpers.get(i).getCnp(),listpers.get(i).getSex() );  
 //p.setCnp(listpers.get(i).getCnp());  
 // p.setNume(listpers.get(i).getNume());  
 // p.setSex(listpers.get(i).getSex());* Cont c = **new** Cheltuieli(listpers.get(i));  
 c.setId(Integer.*parseInt*(**idText**.getText()));  
 **banca**.addContChe(c, listpers.get(i));  
 **s**.*exportb*(**banca**);  
  
 }  
 }  
 }  
 **else if**(**tipText**.getText().equalsIgnoreCase(**"economii"**)) {  
 String cnps = **persText**.getText();  
 List<Persoana> listpers = **banca**.getPersoane();  
  
 System.***out***.println(listpers);  
 **for** (**int** i = 0; i < listpers.size(); i++) {  
 **if** (listpers.get(i).getCnp().equalsIgnoreCase(cnps)) {  
  
 Economii e = **new** Economii(listpers.get(i));  
 e.setId(Integer.*parseInt*(**idText**.getText()));  
 **banca**.addContEco(e, listpers.get(i));  
 **s**.*exportb*(**banca**);  
  
 }  
 }  
  
 }  
 **this**.**model** = FXCollections.*observableArrayList*(**banca**.getConturi());  
  
 **tablecont**.setItems(**model**);  
 System.***out***.println(**"lista conturi"**);  
 System.***out***.println(**banca**.getConturi());  
  
 }  
  
 **public void** refresh() **throws** IOException, ClassNotFoundException {  
  
 **banca**=**s**.*importb*();  
 System.***out***.println(**banca**.getHBanca());  
 **s**.*exportb*(**banca**);  
 **this**.**model** = FXCollections.*observableArrayList*(**banca**.getConturi());  
 **tablecont**.setItems(**model**);  
  
 }  
  
 **public void** handleAdauga() **throws** IOException {  
 Cont c = **tablecont**.getSelectionModel().getSelectedItem();  
 **if**(c **instanceof** Economii) {  
 ((Economii) c).adaugareBani(Integer.*parseInt*(**sumaText**.getText()));  
 **s**.*exportb*(**banca**);  
 }  
 **else** {  
 ((Cheltuieli)c).adaugareBani(Integer.*parseInt*(**sumaText**.getText()));  
 **s**.*exportb*(**banca**);  
 }  
  
 }  
 **public void** handleRetrage() **throws** IOException {  
 Cont c = **tablecont**.getSelectionModel().getSelectedItem();  
 **if**(c **instanceof** Economii) {  
 ((Economii) c).retragereBani(Integer.*parseInt*(**sumaText**.getText()));  
 **s**.*exportb*(**banca**);  
 }  
 **else** {  
 ((Cheltuieli)c).retragereBani(Integer.*parseInt*(**sumaText**.getText()));  
 **s**.*exportb*(**banca**);  
 }  
  
 }  
 **public void** handleSterge() **throws** IOException {  
 Cont c = **tablecont**.getSelectionModel().getSelectedItem();  
 Persoana p=c.getTit();  
 **banca**.delCont(p,c);  
 **s**.*exportb*(**banca**);  
  
 }  
  
  
}

Asemanator sunt si celellate 2 Controllere, singura diferenta ca ControllerMainWindow imi porneste cele 2 interfeete prin 2 metode Init.

5.Concluzii

Ca o concluzie, acest proiect m-a ajutat sa imi consolidez cunostintele de programare orientata pe obiect dobandite in primul semestru si sa imi organizez munca in baza paradigmelor POO de asemenea acest proiect m-a ajutat sa inteleg mult mai bine sistemul bancar.

In cadrul dezvoltarilor ulterioare se pot aminti urmatoarele: imbunatatirea interfetei grafice,adaugarea de teste pentru introducerea datelor, etc.