Bank

Student:Mocan Ioana

Grupa:30223

Cuprins

1. Obiectivul temei

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

3. Proiectare (diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

4. Implementare si testare

5. Rezultate

6. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare

7. Bibliografie

1. **Obiectivul temei**

Obiectivul temei prezentate este acela de a crea o aplicatie care sa simuleze diferite operatii necesare pentru a crearea si modificarea unor conturi.

Comenzile efectuate asupra conturilor sunt astfel efectuate incat sa scoata in evidenta operatiile care sunt prezente in interiorul unei banci.

**2. Analiza problemei, modelare, scenarii, cazuri de utilizare**

Prin analiza problemei ne dam seama ca avem nevoie de concepte care privesc lucrul cu programare obiectuala.Problema a fost analizata din punct de vedere al unui banci care are clientii retinuti si fiecare poate avea unul sau mai multe conturi.Problema este una care permite diferite abordari si nu este exclusa ideea ca fiecare abordare sa aiba rezultate diferite.In cazul dezvoltari problemei elementele incluse in rezolvarea problemei variaza de la ca la ca ele sunt fie mai bine conturate in ceea ce priveste incapsularea, mostenirea, polimorfismul, abstractizarea.Pentru a putea implementa corect problema este necesar sa fim atenti la datele necesare si la comunicarea cu exteriorul.

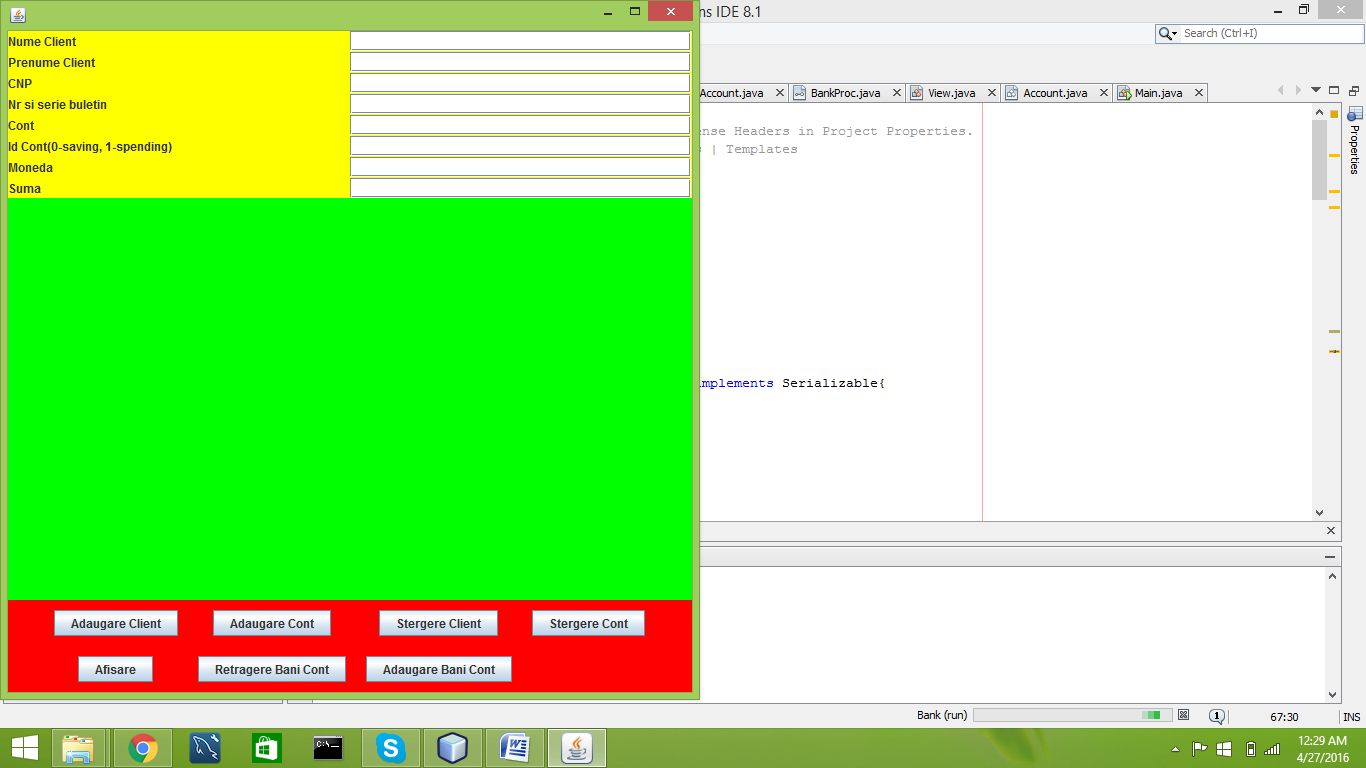
In analiza problemei am luat in considerare faptul ca avem o banca poate avea mai multi clienti si fiecare client poate avea mai multe conturi de diferite tipuri..Fiecare client trebuie sa se afle in baza de date si daca nu exista nu poate sa isi creeze un cont.Daca un client doreste sa depoziteze bani in cont trebuie sa se stie informatiile despre client despre contul in care se depoziteaza si suma dorita.Pentru introducerea de persoane noi avem neoie de nume ,prenume ,cnp,seria si numarul de la buletin si apoi informatiile necesare pentru cont adica numar cont,moneda, sumasi tipul contului adica daca este saving sau de spending.Daca dorim sa stergem un client sau un cont ne folosim de toate informatiile despre client respectiv de toate informatiile despre client si despre cont.Daca se doreste adaugarea de bani in cont sau scoatere este nevoie sa se stie informatii despre persoana si despre contul respectiv.Operatiile specifice acestei probleme sunt : adugare client,adugare cont,afisare ,stergere clienti,stergere cont,retragere bani,depunere bani.

Un aspect care conteaza in relatia utilizator-produs este crearea unei interfete cat mai prietenoase si care sa scoata in evidenta cat mai bine utilizarea codului.

Din punct de vedere a implementari aplicatiei exista mai multe optiuni de implementare ceea pe care am ales-o eu este una care poate fi imbunatatita. Varianta poe care am ales-o sa o implementez poate sa lase putin de dorit doarece administratorul este nevoit sa introduca multe informatii in ceea ce priveste fie o editare fie o adugarea de cont sau de client.Am incercat sa evidentiez cat mai bine ce se intampla intr-o banca si sa se demonstreza ca tot ce se intampla intr-o banca depinde foarte mult pe informatie.

Utilizarea aplicatiei este descrisa in ceea ce urmeaza:

* In functie de operatia care trebuie efectuata se vor lua in considerare unul dintre textfiel-urile care sunt evidentiate in interfata grafica
* O data completate textfielul se apasa pe unul dintre butoane care sunt specifice diferitelor obtiuni prezente.
* Informatia despre client si despre cont se va afisa intr-un tabel care are specificate hedere cu informatia necesara.



**3. Proiectare (diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)**

Penru implementarea aplicatiei este necesara utilizarea a 8 clase, fiecare fiind specializata in solutionarea unor operatii specifice.

**public class Main**

Aceasta clasa are rolul de a apela clasa View care creeaza interfata si utilizeaza operatiile aplicate in interiorul unui depozit.Practic are rolul de a rula aplicatia si preluand tema sistemului de operare pentru a crea casetele cu interfata grafica si pentru a o expune pentru utilizator.In acesata clasa se mai realizeaza si deserializarea.

**public class Person implements Serializable,Observer**

In aceasta functie sunt doar constructorul si functiile de get si set care preiau id-ul ,numele si prenumele,cnp si seria si numarul de la buletin la fiecare client.In aceasta clasa mai sunt suprascrise clasele de equles,hashcode,updat.

**public abstract class Account extends Observable implements Serializable**

In acesata functie sunt declarate mai multe metode abstracte care se vor utiliza mai apoi la clasele de savingAccount si spendingAccount.

**public class SavingAccount extends Account implements Serializable**

In acesata clasa pe langa constructor,get si set si inca cateva metode de care au rolu de a notifica un client in functie de ce schimbari se realizeaza asupra contului lui. Mai avem metoda de equals si de hashcode care practic ne rezolva coliziunile.

**public class SpendingAccount extends Account implements Serializable**

In acesata clasa pe langa constructor,get si set si inca cateva metode de care au roulu de a notifica un client in functie de ce schimbari se realizeaza asupra contului lui. Mai avem metoda de equals si de hashcode care practic ne rezolva coliziunile.

**public interface BankProc**

Aceasta clasa este una de tip interfata si aici sunt declarate antetele metodelor.Aceste metode vor fi mai apoi folosite in clasa Bank care implementeaza HashMap-ul.

**public class Bank implements BankProc,Observer**

In aceasta clasa se utilizeaza un HashMap care ne ajuta sa retinem informatia pe care o avem de retinut adica avem ca si cheie obiectul de tip persoana iar ca si lista avem o lista de Conturi de diferite tipuri fie de saving fie de spending.In aceasta clasa pe langa constructor avem clasa care ne transforma hasul intr-o matrice pentru ca mai apoi sa o putem pune in JTable, avem functiile din BankProc implementate si mai avem si functia de serializare.

**public class View extends JFrame implements ActionListener,ItemListener**

Clasa View are ca rol implementare a interfetei grafice si asigurarea ciclului de informatie dinspre interior sper exterior si invers.Toata implementarea grafica se realizeaza cu ajutorul a trei panel-uri care sunt raspunzatoare de datele de intrare ,de butoane si de datele de iesire.

Interfata grafica este realizata din 7 butoane care se ocupa de realizarea operatiilor.Pe langa butoane se mai gasesc si label-uri care ajuta la intalegerea corecta a semnificatiei datelor introduse si afisate in tabel.

Fiecare buton este responsabil sa indeplineasca o operatie singulara si care sa faca ceea ce ii este cerut.

Butonul „Adaugare Client ” are rolul de a aduga un client in lista de clienti.

Butonul „Afisare” are rolul de a afisa continutul mapului

Butonul „Adaugare Cont” are rolul de a adauga un cont in lista de conturi din map.

Butonul „Stergere Cont” are rolul de a sterge un cont in lista de conturi din map .

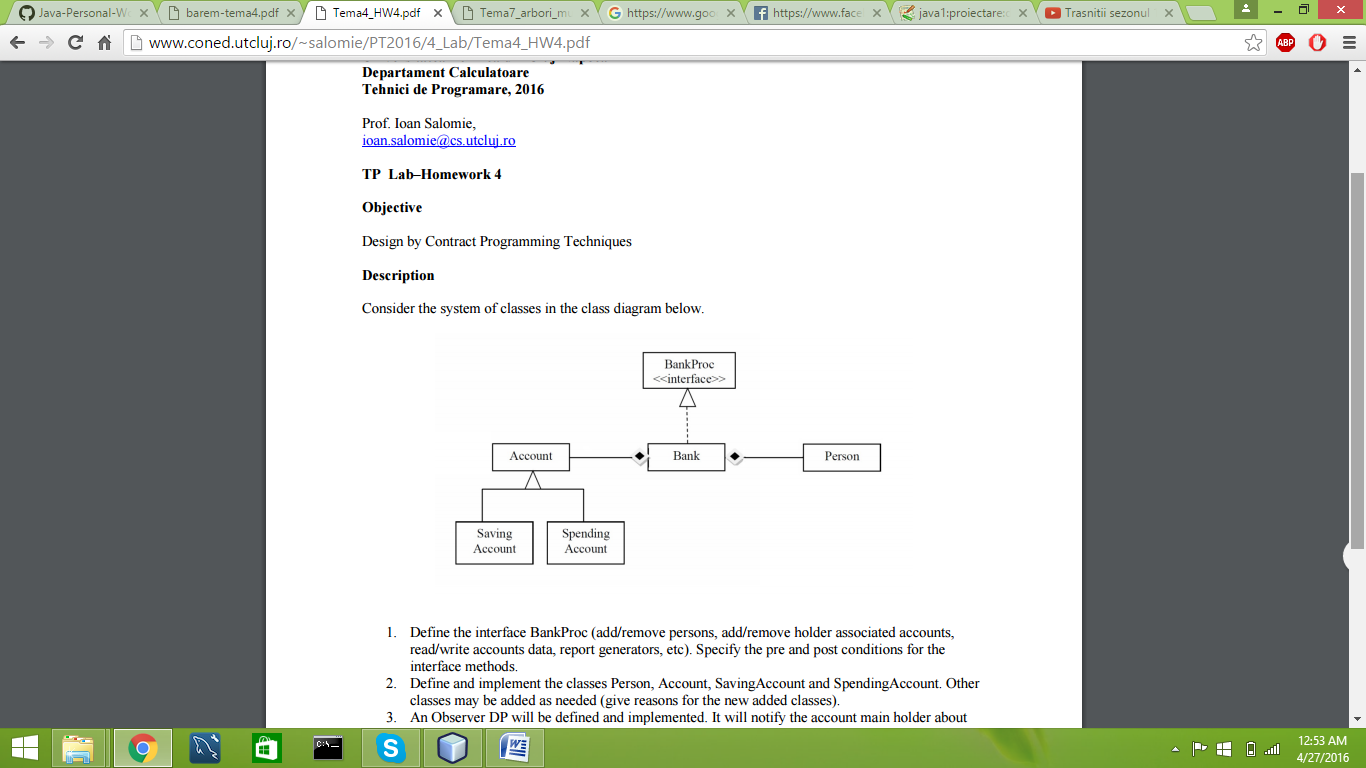
Butonul „Stergere Client” are rolul de a sterge un client din map.

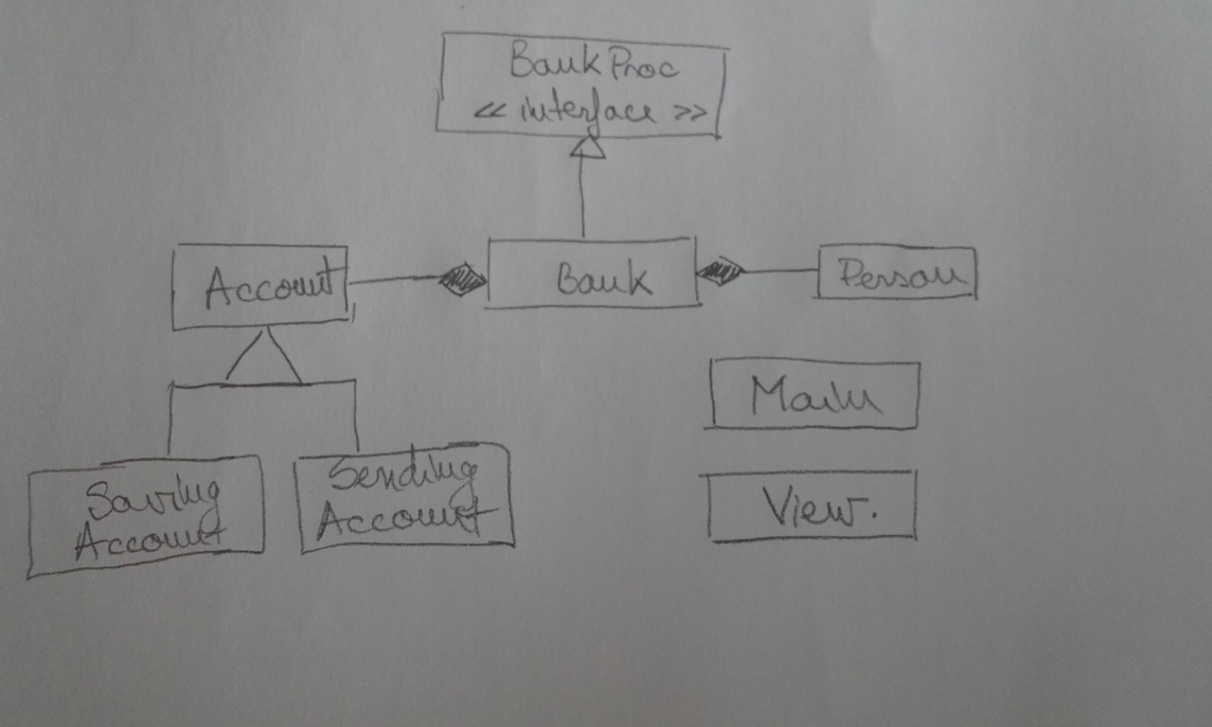
Butonul „Adaugare Bani Cont” are rolu de a aduga o anumita suma intr-un anumit cont.

Butonul „Retragere Bani Cont” are rolu de a retrage o anumita suma intr-un anumit cont

**4. Implementare si testare**

Dupa cum se observa din diagrama UML s-au respectat paradigmele programarii orientate pe obiect. Abstractizarea se poate observa prin evidentierea elementele indispensabile.Conceptul de incapsulare se verifica la declararea private a elementelor din fiecare clasa , acestea putand fi accesate doar cu ajutorul unor metode de tip get si set.





In ceea ce urmeaza o sa explic metodele mai importante din fiecare clasa:

**public class Person implements Serializable,Observer**

Una din metodele importante pe care le implementam aici este aceea de **public boolean equals(Object obj)** care se suprascrie in functie de informatia pe care dorim sa o verificam aceasta functie ne ajuta sa evitam coliziunile la fel si functia **public int hashCode()** care ne creaza o functoe de hash automat care ne pozitioneaza in hashmap in functie de numele si prenumele persoanei respective.

**public abstract class Account extends Observable implements Serializable{**

**abstract public String getCont();**

**abstract public void setCont(String cont);**

**abstract public String getMoneda();**

**abstract public void setMoneda(String moneda);**

**abstract public float getSuma();**

**abstract public void setSuma(float suma);**

**@Override**

**abstract public void notifyObservers();**

**@Override**

**abstract public void setChanged();**

**@Override**

**abstract public boolean equals(Object o);**

**@Override**

**abstract public int hashCode();**

**abstract public int getId();**

**abstract public void setId(int id);**

In aceasta clasa sunt metodele pe care le implemetam in clasele de SavingAccount si SpendingAccount practic majoritatea sunt setere si getere dar in cele doua clase de saving si spending mai avem si alte clase care ne realizeaza operatiile necesare.

**public class SavingAccount extends Account implements Serializable**

**public void addSuma(float suma)** in acesata metoda se realizeaza adunarea sumei la o suma existenta deja in cont.

**public void notifyObservers()**  in aceasta metoda trebuie sa verificam daca ceva s-a schimbat si daca da atunci este anuntat un observator.

In acesata clasa avem **public boolean equals(Object obj)** care se suprascrie in functie de informatia pe care dorim sa o verificam aceasta functie ne ajuta sa evitam coliziunile la fel si functia **public int hashCode()** care ne creaza o functoe de hash automat care ne pozitioneaza in hashmap in functie de numele si prenumele persoanei respective.

**public class SpendingAccount extends Account implements Serializable**

**public void getSumaAccount(float suma)-** in acesata metoda se realizeaza retragerea sumei din interiorul contului.

**public void addSumaAccount(float suma)** in acesata metoda se realizeaza adunarea sumei la o suma existenta deja in cont.

**public void notifyObservers()**  in aceasta metoda trebuie sa verificam daca ceva s-a schimbat si daca da atunci este anuntat un observator.

In acesata clasa avem **public boolean equals(Object obj)** care se suprascrie in functie de informatia pe care dorim sa o verificam aceasta functie ne ajuta sa evitam coliziunile la fel si functia **public int hashCode()** care ne creaza o functoe de hash automat care ne pozitioneaza in hashmap in functie de numele si prenumele persoanei respective.

**public interface BankProc {**

**public void addPerson(Person p,ArrayList<Account> a);**

**public void removePerson(Person p);**

**public void addCont(Person p,Account c);**

**public void removeCont(Person p,Account c);**

**public void adaugareBaniSaving(Person p,Account c,float suma);**

**public void adaugareBaniSpending(Person p,Account c,float suma);**

**public void scoatereBaniSpending(Person p,Account c,float suma);**

**}**

In aceasta clasa scriem antetele metodelor care mai apoi trebuie implementate in clasa de Bank .In principiu sunt metode care aduga sau sterg ceva din map.

**public class Bank implements BankProc,Observer**

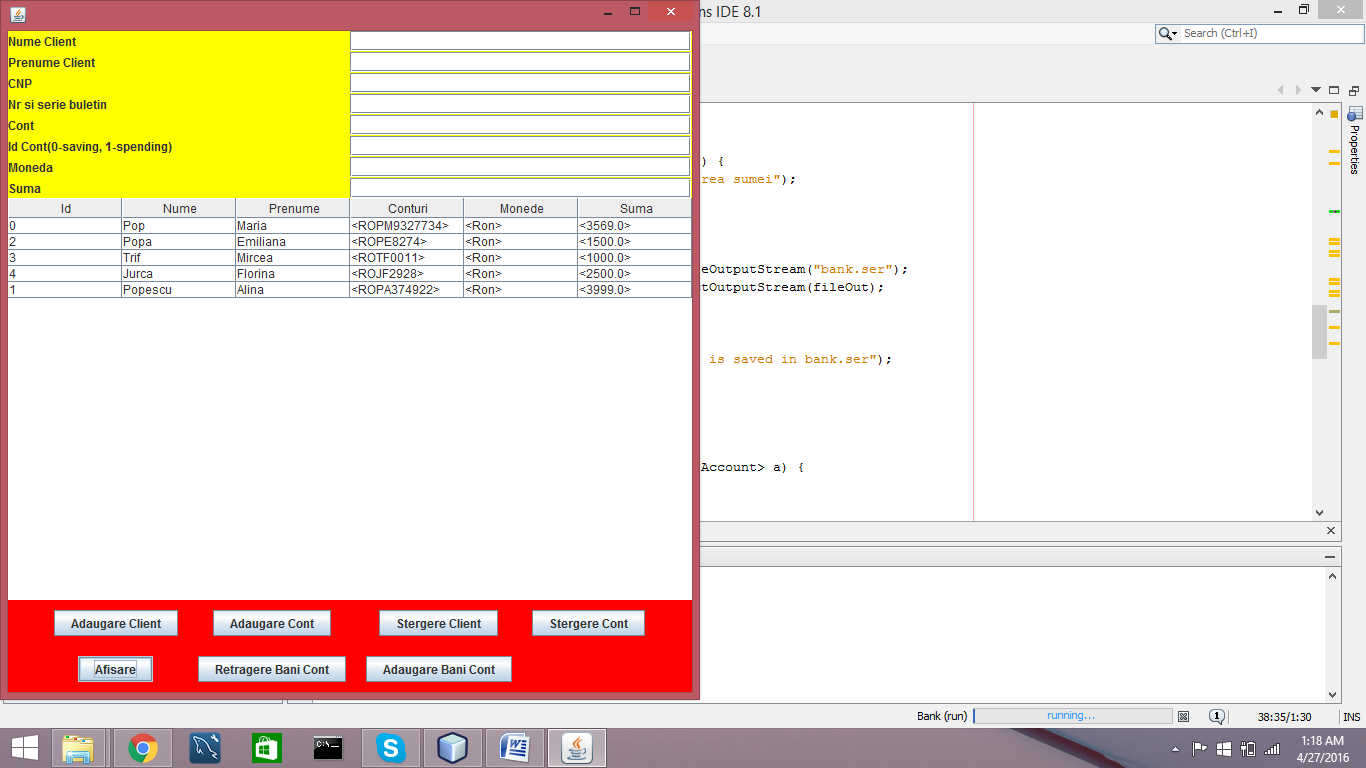
**public boolean isWellFormed()**  acesata functie este special creata pentru a verifica daca map-ul este gol sau nu aceasta conditie este verificata de fiecare data cu ajutorului assertului.

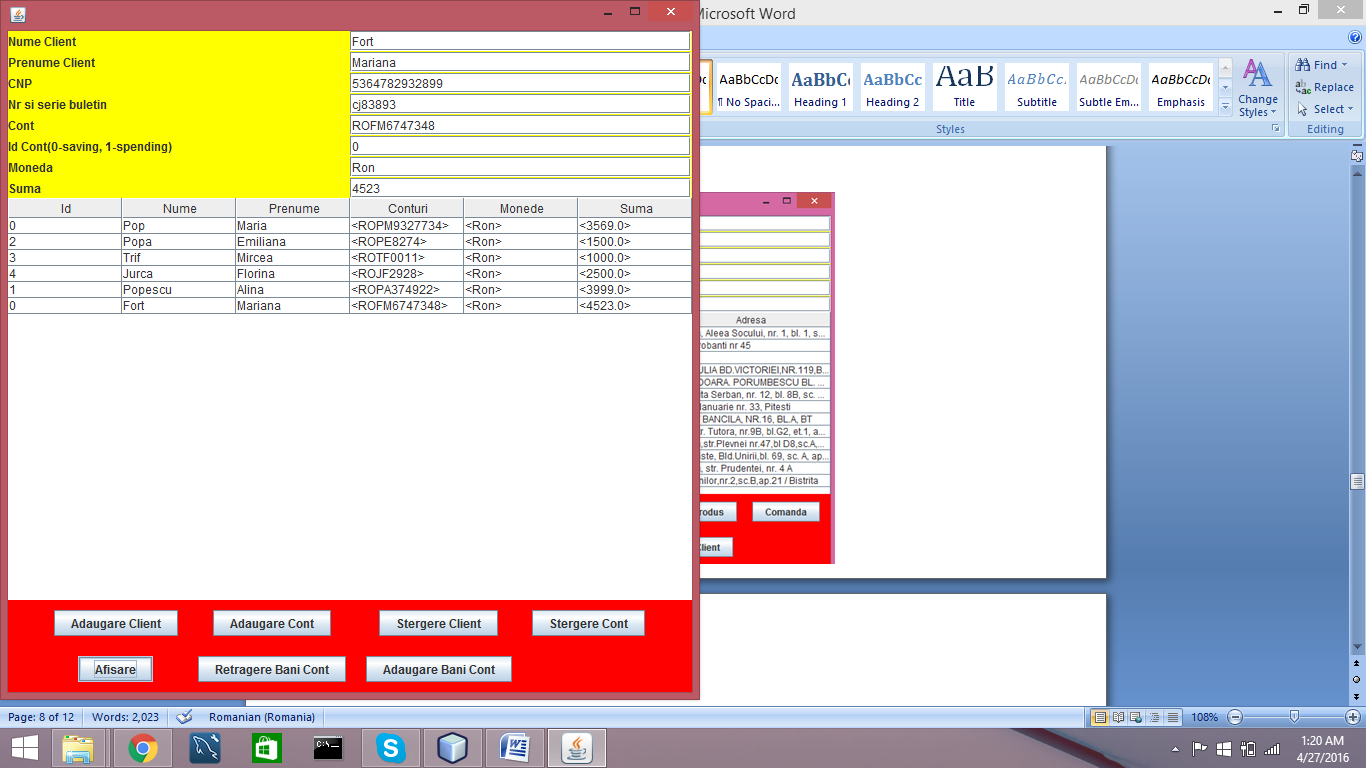
Aici avem functia de convertire din map la o matrice de string-uri care mai apoi este afisata in JTable . Aici avem functia de serializare care realizaza serializarea adica transformarea in biti.Aici sunt implementate functiile de adugare si de stergere cu ajutorul unor functii specifice map-ului.

**5. Rezultate**

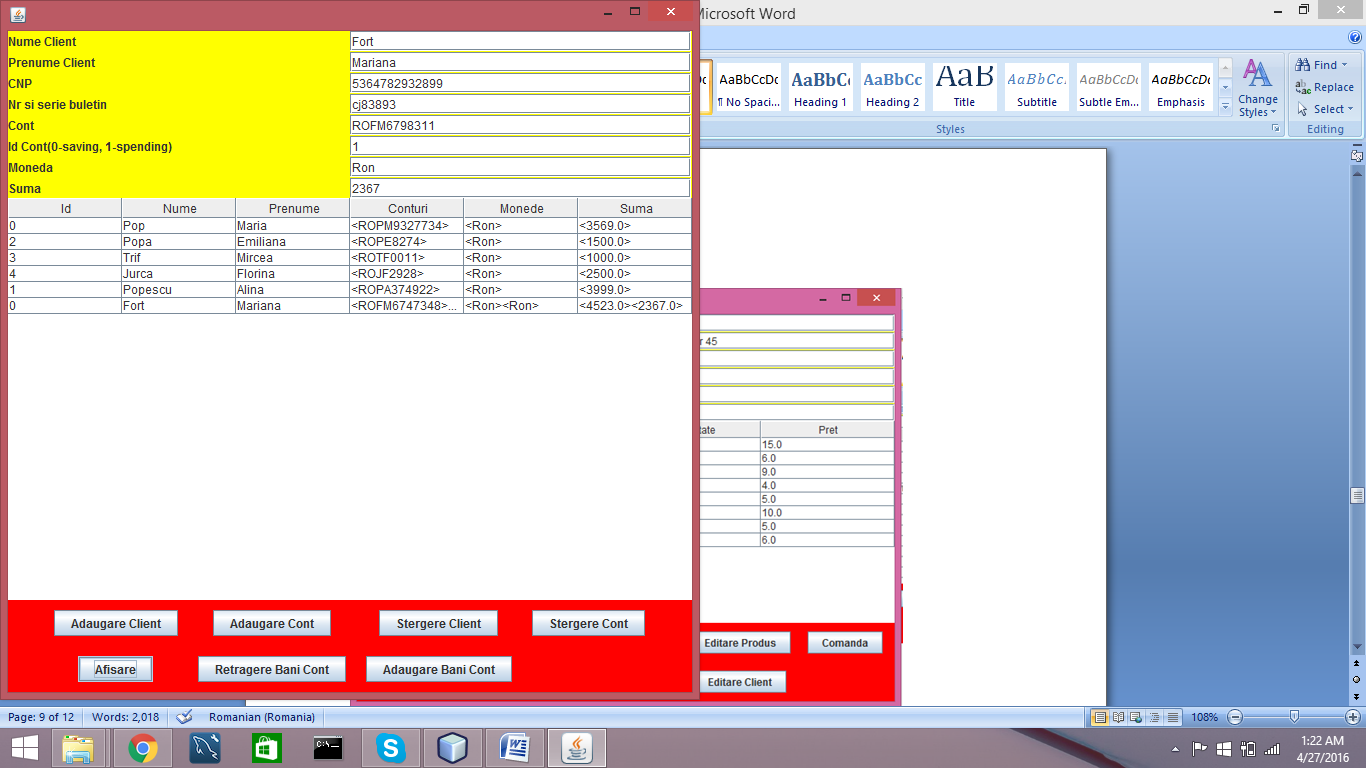
In acesata etapa se va testa functionalitatea aplicatiei pentru functiile pe care acestea trebuie sa le implementeze verificand astfel si corectitudinea algoritmilor implemnentati.

**Afisare fisier**

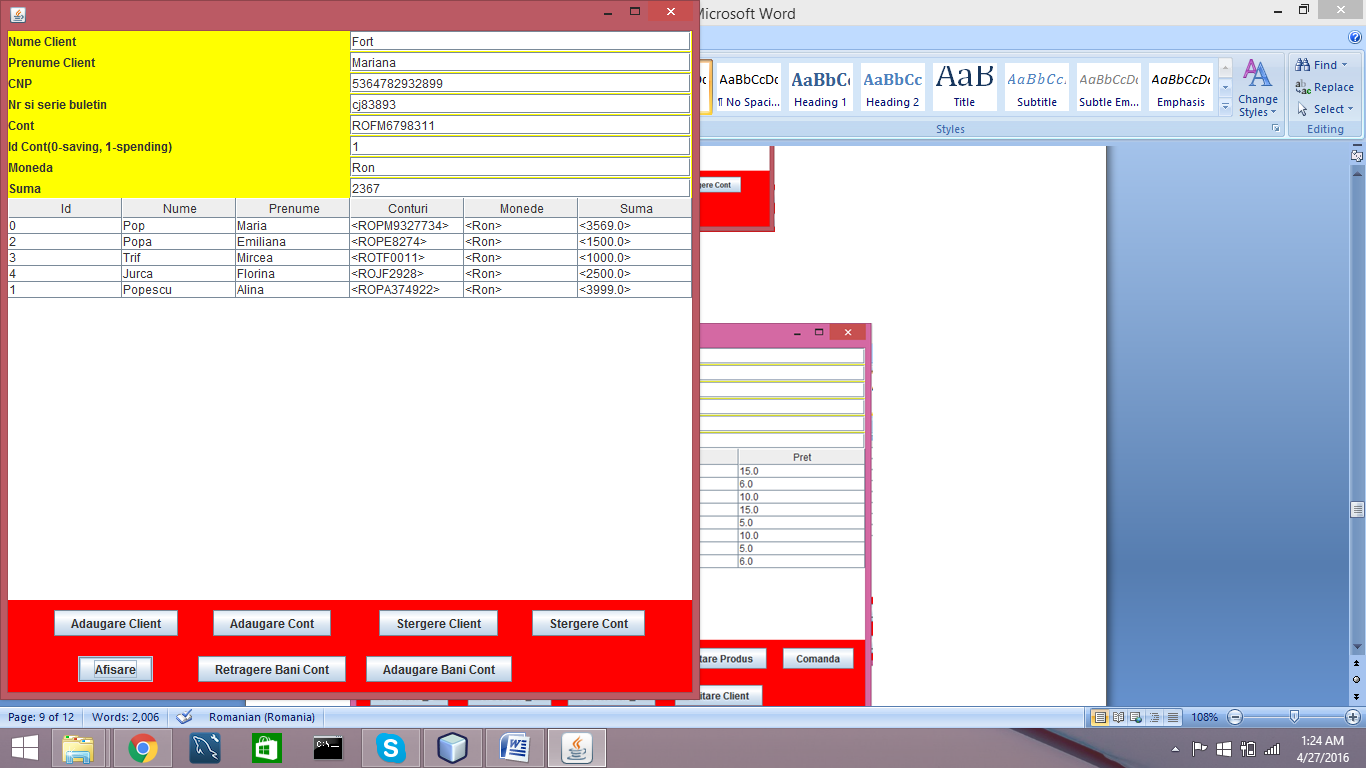


**Adugare de client**

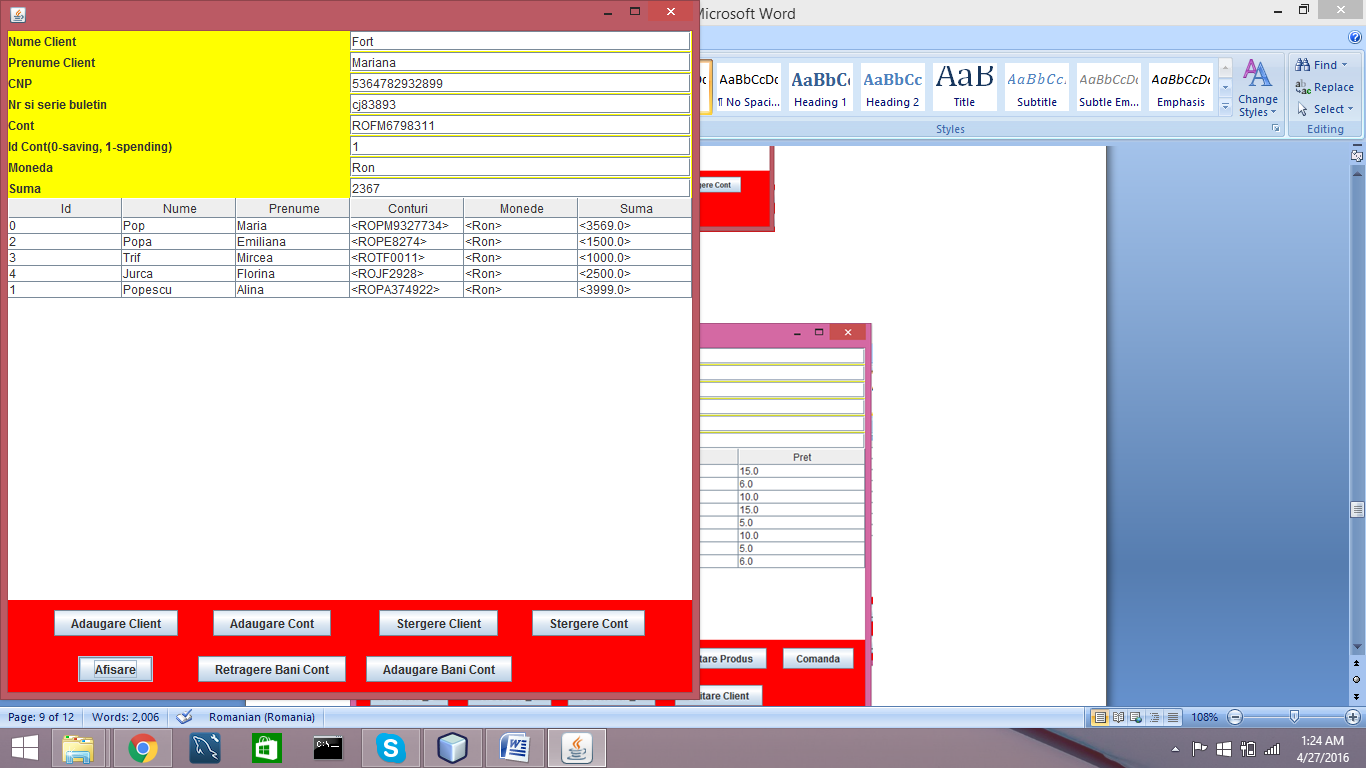
**Adaugare de cont la o persoana**



**Stergere client**



**Stergere cont la o persoana existenta**



Practic ar mai exista si alte operatii de verificat ca aceea se retragere bani din cont si aceea de augare bani in cont . In momentul in care apas butonul de adugare bani in cont se va aduga bani la contul la care este de tipul specificat fie saving fie spending.

**6. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare**

Prin aceasta tema s-au evaluat conceptele de baza ale ale programarii orientate pe obiect concepte care m-au ajutat sa imi insusesc mai bine unele notiuni legate pe programarea orientata pe obiect.

Dezvoltari ulterioare:

* Creare unei ferestre de login si de preferat sa se poata realiza comenzi si din partea de clientului pentru a verifica ce are in conturi .
* Modificarea interfetei pentru a o face mai atragatoare fata de client si adaugarea de obtiuni noi care sa ii satisfac clientului preferintele.

**7. Bibliografie**

[**http://stackoverflow.com/**](http://stackoverflow.com/)

[**http://www.tutorialspoint.com/java/java\_serialization.html**](http://www.tutorialspoint.com/java/java_serialization.html)

[**https://docs.oracle.com/javase/tutorial/uiswing/components/table.html**](https://docs.oracle.com/javase/tutorial/uiswing/components/table.html)

[**https://docs.oracle.com**](https://docs.oracle.com)

[**http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html**](http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html)

[**http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html**](http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html)

[**http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html**](http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html)

[**http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise**](http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise)