**DOCUMENTATIE TEMA 4**

**Management-ul conturilor bancare**

**Brincoveanu Vasile Vlad**

**30223**

**Profesor Laborator Assist:Antal Marcel**

Contents

[1. Cerinte Functionale 3](#_Toc476131445)

[2. Obiective 3](#_Toc476131446)

[3. Analiza Problemei 3](#_Toc476131449)

[4. Proiectare 3](#_Toc476131450)

[4.1. Structuri de date 3](#_Toc476131451)

[4.2. Diagrama de clase 3](#_Toc476131452)

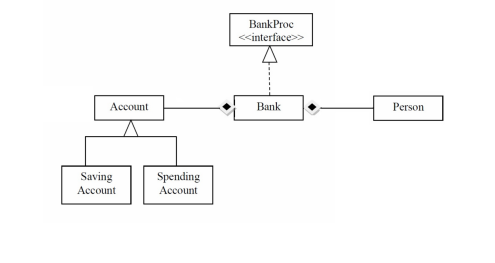
[4.3. Algoritmi 3](#_Toc476131453)

[5. Implementare 4](#_Toc476131454)

[6. Concluzii si Dezvoltari Ulterioare 4](#_Toc476131456)

[7. Bibliografie 4](#_Toc476131457)

# Cerinte Functionale



* Implementam diagrama de mai sus, alegem intelept structura de data pentru a salva persoana si aconturi. Sa diferentiem intre Map si Set .
* Interfata grafica
  + O fereastra pentru operatii pe personae: adauga o noua persoana, editeaza o persoana,vezi toate persoanlele intr-un table(JTable).
  + O fereastra pentru operatii pe account: adauga aount,editeaza account,sterge account,vezi toate countirle(unei personae) intr-un tabel (JTable).
* Sa diferentiam intre cont de cheltuiala si cont de salvari. Contul de salvari va permite o singura suma mare sa fie depozitata si scoasa si va calcula un interest rate in timpul depozitarii. Contul de cheltuiala, va permite mai multe adaugari si retrageri, dar nu va calcula niciun interestRate.
* Adauga sau scaote bani din conturi.
* Documentatie
* Folositi JTable pentru display.Implementam click listener pe JTable.
* Sa facem un o clasa test(Junit).
* Design by contract:folosim preconditit,postconditii in interfata BankProc. Le implementan

In clasa Banca folosind intructiuni assert.

* Design Pattern Observer, pentru a notifica detinatorul acontului cand unul dintre conturi este modificat.
* Salvam informatia din calsa banka intr-un fisier folosind serializarea. Incarcam informatia

cand aplicatia porneste.

# Obiective

1. Definim o interfata BankProc( care va avea adauga/sterge persoana, adauga/sterge cont,

Citeste/scrie date pe cont, etc).Specifica preconditiile si postconditiile metodelor din interfata.

1. Sa implementam clasele persoana, cont,savingcont, spendingcont si altele care vor fi adaugate dupa nevoie.
2. Orice observer va fi definit si implementat. Va notifica detinatorul contului despre operatii pe contul repsectiv.
3. Implementan clasa Bank folosind o colectie predefinita care foloseste un hashtable. Cheia hasttable va fi generata pe baza informatiilor persoanei. O persoana poate fi detinatorul mai multor conturi.Folositit JTable sa aratati informatiile cu privire la Banca.
4. Implementam clasa Banca folosit Design by Contract.
5. Implementant o clasa de test pentru metodele din banca.
6. Datele pentru conturi vor fi incarcate ,salvate dintr-un fisier.

# Analiza Problemei

Se doreste sa simulam, cream o aplicatie bancara , care va crearer si salva persoane intr-o strucutra de date si care va avea diferite conturi ,pe care va putea sa faca incarce bani, sa extraga, sa depuna etc.

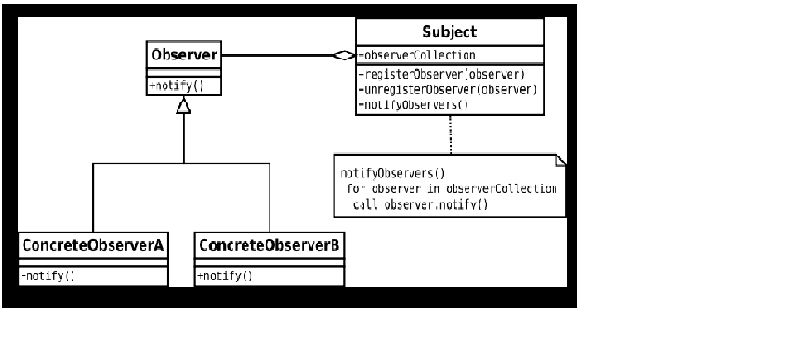
La aceasta aplicatia vom urmarii o diagram de clasa simplista, cea pusa mai sus, la care vom

adauga Design by Contract,Observer , Intefrata, iar mai apoi vom serialzia datele, astfel incat

sa salvam pe fisier fiecare oepratie ce o vom face, si le vom incarca dinou cand pornim aplciatia.

Problema la Design By Contract se pune ca noi vrem sa testam anumite metode, inainte si dupa executarea lor, astfel incat sa fim siguri ca merg bine, dar si mai mult ,nu vom folosi if-uri, deoarece dupa testarile positive vom fi nevoiti sa le scaotem din program, dar daca folosim,assert-uri va fi mult mai usor.Assertu-rile se vor active doar daca vom compila cu argumentul -ea, astfel fara el, compilatorul va sari peste toate asserturile scrise in program, nefiind nevoitit sa mai curatam codul.

Problema la Design Pattern Observer se pune ca noi vrem ca dupa fiecare operatie executata pe cont, cum ar fi adaugare, retragere bani, sa informam detinatorul contului de acestea. Ea se rezolva folosind observeri. Observeri se vor implementa dupa urmatoarea diagrama.



Neintrand in detalii, in cazul nostrum subiectu, in care se vor adauga observeri si din care se vor citii mesaje, va fi contul, iar observatorul, cel care va implementa interfata Observer, va fi clientul. Prin acestea detinatorul contului in care sa efectuat o operatie, va fi informat prompt cu privire la acestea.

Problema icnarcarii si salvarii dintr-un fisier se pune ca noi vrem sa efectuam operatii in aplciatia noastra, dar totodata sa ramana salvate, sau sa avem posibilitatea ca urmatoarea data cand aprindem aplicatia, sa pelcam cu datele salvate anterior. Astfel se va folosi serializarea, prin care clasa banca se va transforma intr-un sir de octeti, cu toate infroamtia din ea si va si scrisa intr-un fisier, de unde la fiecare pornire a aplicatiei vom incarca datele din acel fisier.

**Scenarii si user case-uri**

Programul prevede urmatorii actori: utilizatorul, care este in cazul nostru adminul aplicatiei.

El va comunica cu interfata, si cu ajutorul altui actor controllerul, va efectua oepratii pe persoane sau pe conturile lor.

*Scenariul: Utilizarea normala*

1. Utilizatorul va deschide aplicatia, si va putea vedea un tabel cu toti clientii, si butoane prin care va putea sa adauge client, sa modifice, sa stearga etc.
2. Pentru adaugare, sau editare va umple coloanele necesare din stanga,cu nume,adresa,varsta etc...cu un mic ps(la editare nu va putea sa editeze cnp-ul si varsta deoarce sunt date immovable,declarate final, pentru a se folosi functia de hashing, in mod corect pe ele).
3. Utilizatorul va putea sa dea click pe un client, si apoi click pe butonul showAccounts, unde ii se va deschide o fereastra cu toate conturile persoanei respective + posibilitatea de a adauga alte conturi, de a adauga o suma de bani in ele, de a le edita sau sterge.
4. Pentru creare de cont se va adauga suma, si avme butoane pentru creare cont de salvari sau de cheltuilei, cu posibilitatea ulterioare de a le edita.(interestRate-ul este calculate automat, dar la nevoie, se poate suprascrie fortat, utilziand textFieldul interestRate is butonul edit..)

# Proiectare

## Structuri de date

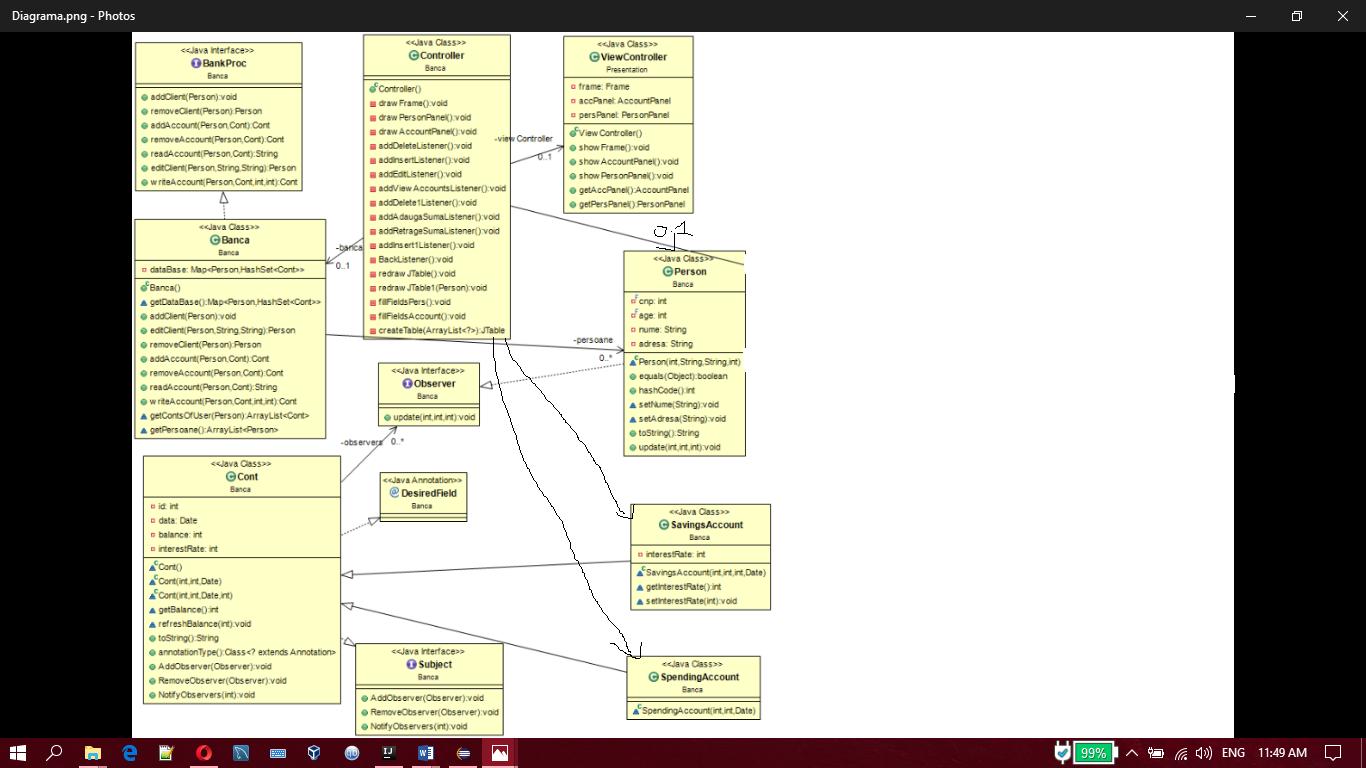
S-au folosit structure de date simple, precum string ,integer , date , pentru memorarea informatiilor despre clienti cat si despre conturile lor.

Pentru a stoca obiecte spre a fi afisate in JTable s-a folosit arrayList(care este un array implementat sub forma de lista, deci fara limita implicita) .

S-au mai folosit matrici de string-uri pentru a crea tabele din JTable.. etc.

S-au folosit si structuri de date mai avansate precum, hashtable(pentru a tine persoane impreuna cu conturile lor).HashSet,folosit impreuna cu hastable astfel -> hashtable tine in ele cheie si valoare, la cheie i-am atribuit hashfunction-ul persoanei, iar la valoare vom avea un HashSet de conturi.

## Diagrama de clase



## Algoritmi

Programul contine simplii algoritmi de aflare a cea mai eficiente cozi, de introducere a clientului in coada respectiva , etc



# Implementare

1. **Pachetul Banca**

In acest pachet se va gasii majoratiatea aplicatiei , impreuna cu clasele:banca,Cont,SavingCont,SpendingCont,Controller,Persoana,cat si interfetele:bankproc,desiredField,Observer,Subject.

**Clasa Persoana**

Contine ,atributele unei persoane(varsta numele,cnp,adresa),contructori, getteri si setteri dar si metoda equals si hash suprascrisa.

public boolean equals(*Object o*) {  
 if (this == *o*) return true;  
 if (*o* == null || getClass() != *o*.getClass()) return false;  
  
 *Person* person = (*Person*) *o*;  
  
 if (cnp != person.cnp) return false;  
 if (age != person.age) return false;  
 return (nume != null ? nume.equals(person.nume) : person.nume == null) && (adresa != null ? adresa.equals(person.adresa) : person.adresa == null);  
}  
  
public int hashCode() {  
 int result = cnp;  
 result = 31 \* result + age;  
 return result;  
}

Camputilre cnp si age sunt de tipul final, astfel date imutable, nu pot fi suprascrise, (editate) , deoarece as ava functia functia de hash cum trebuie.

Daca nu ar fi de tipul final, le vom putea edita, si atunci, cand cautam in hashtable dupa hash-ul unei persoane nu o vom mai gasi, deoarece functia hash va returna alt hash.

Metoda equals pur is simplu verifica toate campurile din clasa sa fie egale cu ce compar.

Clasa implementeaza interfata Observer, si astfel metoda update, care va prelua informatii din clasa cont si le va da persoanei detinatoare(face parte din OBsderver Pattern, unde clasa persoana este observatorul, cel care asteapta informatia). Ea mai implementeaza si interfata Serializable, folosita pentru a transforma obiectele instantiate in acestea intr-un sir secvential de octeti , folositi pentru a salva si incarca starea aplicatiei.

**Clasa Cont**

Clasa abstracta, folosita pentru a fi extinsa de clasele SavingAcount si SpendingAcount.In ea gasim scheletul, in cazul nostru, campurile is metodele generale pe care un cont le poate avea,

Id, data,balance,interestRate..etc). Contine si 2 contructori depinde de ce obiect se va instantia.

Contine getteri sepcifici, cat si un toString supracris. Clasa implementeaza interfata Subject, ce provine din Observer Pattern, ea vine cu metode precum add observer,remove observer si notify.Astfel, vom crea un arraylsit de observeri si vom adauga cate un observer pe fiecare cont care va apela metoda notify cand se vor apela seteri pentru balance etc..si va apela Clasa persoana care va primii informatiile transmise de aceasta clasa.

Ea mai implementeaza si interfata Serializable, folosita pentru a transforma obiectele instantiate in acestea intr-un sir secvential de octeti , folositi pentru a salva si incarca starea aplicatiei.

Mai implementeaza inca o interfata DesiredFields care foloseste Annotations. Am notat cu @DesiredField fiecare camp, ca apoi ,folosind reflexia pentru a contrui JTable, sa stie ce campuri sa scoata din clasa aceasta. Astfel am evitat sa scoata si arraylsit-ul de observeri, care nu ma intereseaza sa il afisez.

@DesiredField  
private int id;  
@DesiredField  
private *Date* data; // data  
@DesiredField  
private int balance;  
@DesiredField  
private int interestRate;  
  
private *ArrayList*<*Observer*> observers = new ArrayList<*Observer*>();

**Clasa SpendingCont**

Clasa extinde clasa Cont de mai sus, astfel preluand toate metodele contructorii is cmapuriel care nu sunt private. Vom folosi super in contructori pentru a initializa aceasta clasa, ea find cu nimic diferit fata de calsa skelet Cont.

**Clasa SavingCont**

Clasa extinde clasa Cont de mai sus, astfel preluand toate metodele contructorii is cmapuriel care nu sunt private. Vom folosi super in contructori pentru a initializa aceasta clasa, dar va mentine balanceul intr-un camp final, si va avea si un interestRate, care calsa spendingcont nu il are, ce vor fi instantiate aici.Impreuna cu getterii si setteri de care vor fi nevoie.

**Clasa Banca**

Clasa banca este modelul din Design Pattern MVC, ea detine toata logica aplicatiei. Este facuta pe Design by Contract, astfel toate metodele sale sunt preverificate si postverificate.Contine campuri precum :

private *Map*<*Person*, *HashSet*<*Cont*>> dataBase;  
private *ArrayList*<*Person*> persoane;

Map=ul este folosit ca si database, unde se vor mentine persoanele si conturile lor, legate intre ele. El mentine o cheie si o valoare, cheia noastra va fi hash pe obiectul de tip person, iar valoarea un hashSet de conturi.

dataBase = new HashMap<*Person*, *HashSet*<*Cont*>>();

Arraylist-ul de persoane ne ajuta sa afisam, cat si sa facem operatii pe un JTable de persoane.

Metodele ei provin din interfata BankProc, pe care trebuie sa le suprascrie. Metode precum:

void addClient(*Person person*);  
  
*Person* removeClient(*Person person*);  
  
*Cont* addAccount(*Person person*, *Cont cont*);  
  
*Cont* removeAccount(*Person person*, *Cont cont*);  
  
*String* readAccount(*Person person*, *Cont cont*);  
  
*Person* editClient(*Person person*, *String nume*, *String adresa*);  
  
*Cont* writeAccount(*Person person*, *Cont cont*, int

Metoda addClient verifca inainte daca clientul exista in hashmap, apoi il insereaza si la valoare instanteaza un hashset de conturi,iar mai apoi verifica sa vada daca sa adaugat in hashmap.

Metoda editClient verifica daca clientul exista in hashmap si apoi il cauta, ca dupa care sa ii seteze numele si adresa, celealte fielduri fiind de tipul final pentru a functiona functia de hash cum trebuie; la final va returna persoana.

Metoda removeClient va verifica daca exista clientul in hashmap iar mai apoi il va sterge si apoi il va verifica din nous a vada daca a fost sters; la final va returna persoana stearsa.

Metoda addAcount va prelua toate conturile persoanei respective din hashmap iar mai apoi va verifica daca contul pe care vrem s ail adauga se afla in acest hashset de conturi, dupa care il va insera si va returna contul inserat.

Metoda removeAcount va prelua toate conturile persoanei respective din hashmap iar mai apoi va verifica daca contul pe care vrem sa il stergem se afla in acest hashset de conturi, dupa care il va sterge,va verifica daca a fost sters si va returna contul inserat.

Metoda readAccount va prelua toate conturile persoanei respective din hashmap iar mai apoi va verifica daca contul pe care vrem sa il citim se afla in acest hashset de conturi, dupa care ii va returna metoda toString. **Evitand sa returnam tot contul ,am sporit securitatea + am evitat editarile cat si scoaterea ilegala a informatiilor din cont.**

Metoda writeAccount va prelua toate conturile persoanei respective din hashmap iar mai apoi va verifica daca contul pe care vrem sa il scriem se afla in acest hashset de conturi, dupa care va verifica ce tip de cont ii, si ii va adauga informatiile necesare, daca nu va arunca o exceptie, daca nu va fi niciun tip stiut de cont va fi assert false, la final returnam contul editat.

Clasa mai implementeaza si interfata Serializable, folosita pentru a transforma obiectele instantiate in acestea intr-un sir secvential de octeti , folositi pentru a salva si incarca starea aplicatiei.

**Clasa Controller**

Clasa care leaga totul, in tip MVC Pattern(Model View Controller). Ea este controllerul acestei aplicatii, si va lega, modelul(banca, persoanele, conturile) de interfata( clasa viewController din pachetul presentation). In ea se va instantia banca, view controller, frameul pe care se vor scrie panelurile de persoane sau de conturi, se vor face metode de scriere a panelelor impreuna si cu implementarea ascultatorilor de pe butoane, tabele pe care le au. Sunt 3 mari implementari de ascultatori folosind inner clases:

Una este deleteMetod, care verifica de unde a venit actiunea, de pe care buton din care panel, si va interpreta aceasta prin a apela metodele necesare stergerii, cat si prin verificarea argumentelor introduse de utilizator prin clasa TestingArgs din pachetul Testings, si afisarea eroriilor daca se intampla ceva neprevazut, ilegal.

Sub aceasi politica se afla si insertMetod, cat is editMetod.

In aceasta clasa avem metoda createTable care foloseste refelxie pentru a crea un JTable cu valoriile din clasa respectiva. Deoarece foloseste refelxie, ea este o metoda generala, va merge pe taote clasele ce respecta tiparele date.

Mai avem si metode de redesenare a tabelelor , care fac ce zice si numele,folosindu-se de metoda createTable.

1. **Pachetul Presentation**

Pachetul este interfata aplicatiei si contine clase precum, accountpanel,personpanel, frame cat si viewController ce le leaga pe toate.Nu voi trece prin toate clasele, idea in mare este ca, noi vom desena un frame pe care se vor sterge si adauga paneluri(contul si persoana) in functie de ce dicteaza controllerul din clasa banca.Deci clasele acountpanel cat si personpanel sunt nsite Panel-uri care au butoane jtextField-uri cat si un jtable care are metoda de set,care va fi apelata de metoda de redrawTable din controller.

1. **Pachetul Serializing**

Pachetul contine o singura clasa numita Serializing, care contine 2 metode de tip statice(Serialize si Deserialize), care vor salva si incarca datele claselor model dintr-un fisier.

1. **Pachetul Start**

Pachetul se ocupa exclusiv cu executarea metodei main, care va apela controllerul aplicatiei.

1. **Pachetul Testings**

Pachetul contine o singura clasa care are metode statice. Metodele primesc variabilele introduse de utilziator din controller, si vor fi testate sa fie valide. Spre exemplu, varsta unui client, nu va putea sa fie mai mcia de 0 dar nici mai mare de , sa zicem,120 de ani. Daca nu se vor indeplinii conditiile, se va arunca o eroare specifica, care va fi prinsa si afisata de controller,astfel utilziatorul va sti clar ce a gresit si ce ar trebui sa faca.

# Concluzii si Dezvoltari Ulterioare

Prin acest proiect, am invatat sa lucrez cu Desing Pattern Observer cat si cu Design by Product. Am invatat sa salvez ca sis a incarca datele folosind serializable (ce mi se pare foarte essential pentru oricare aplicatie). Am invatat sa testez si sa lucrez mai eficient, folosind variabile package or default pentru a pastra informatiile cat de mult securizate. Am invatat sa lucrez in pachete diferite pentru a spori securitatea cat si sa populez JTableuri folosind o metoda generala cu refelxia.

Aplicatia, ca de obicei, poate fi dezvoltata printr-o interfata mai bogata, cu mai multa utilitate, si mai user friendly. Apoi, se poate securiza inforamtia si mai mult, folosind criptari la salvarea informatiei ,decripatri la incarcarea sa. Se poate folosi un log in, in care se logheaza utilizatorul pe contul sau ca sa-si vada conturile si sumele de pe el, si sa opereze actiuni etc.

# Bibliografie

<https://stackoverflow.com>

<http://www.vogella.com/tutorials/JavaAnnotations/article.html>

http://www.coned.utcluj.ro/~salomie/PT\_Lic/3\_Lab/HW4\_Tema4/HW4\_Indications\_Hashing.pdf https://en.wikipedia.org/wiki/Software\_design\_pattern