Design by Contract Programming Techniques

**Dondas Bianca - Elena**

**Grupa 30228**

**Profesor Dr. Ing. Cristina Pop**

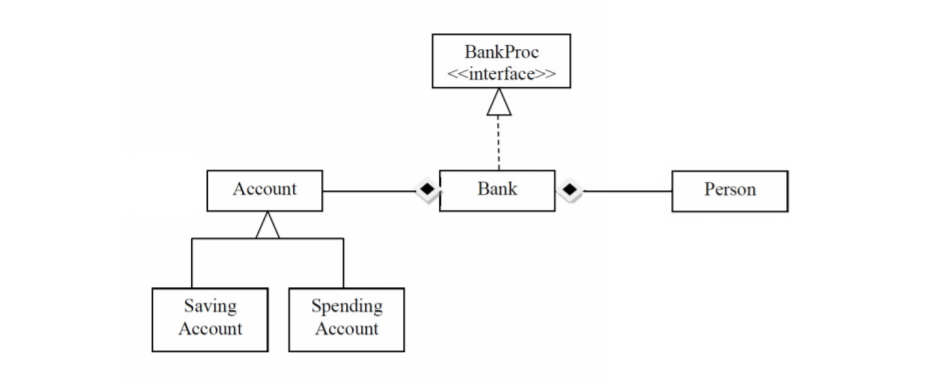
**Cuprins**

# Obiectivul temei

1. Analiza problemei
   1. Analiza
   2. Diagrama use case
2. Proiectare
   1. Diagrama UML
   2. Pachete
      1. Account
      2. Bank
      3. Person
      4. Gui
   3. Clase
3. Implementare
4. Rezultate
5. Concluzii
6. Bibliografie

# Obiectivul temei

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. Define 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 (in 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. Implement a test driver for the system.

6. The account data for populating the Bank object will be loaded/saved from/to a file.

Aceasta tema are ca obiectiv realizarea unei aplicatii care simuleaza managementul unei banci. Actiunile posibile sunt :

1.adaugarea unor persoane noi la baza de date a bancii, editarea lor, sau stergerea unor persoane

2.crearea de conturi ,care pot fi de doua tipuri : de cheltuieli sau de economii

3.stergerea sau editarea unor conturi

4.adaugarea de bani in cont

5.retragerea de bani din cont

1. **Analiza problemei**

Pentru a rezolva aceasta problema, avem nevoie de cunostinte legate de serializare/deserializare, design by contract si utilizarea structurii de Hashtable.

Serializarea obiectelor inseamna salvarea si restaurarea starii obiectelor. Toate datele din aplicatie trebuie retinute intr-un fisier , iar la fiecare deschidere a aplicatiei toate datele din fisier vor fi preluate in aplicatia noastra , la fel si la iesirea din aplicatie toate datele vor fi scrise inapoi in fisier.

Utilizand fluxurile putem scrie aplicatii care salveaza si incarca datele in fisiere. Java permite si un mecanism mai avansat si anume serializarea obiectelor. In forma cea mai simpla serializarea obiectelor inseamna salvarea si restaurarea starii obiectelor. Obiectele oricarei clase care implementeaza interfata Serializable, pot fi salvate intr-un stream (fluxde date) si restaurate din acesta. Pachetul java.io continedoua clase specialeObjectInputStream respectiv ObjectOutputStream pentru serializarea tipurilor primitive. Subclasele claselor serializabile vor fi si ele serializabile. Mecanismul de serializare salveaza variabilele membri nestatice si netransiente. Daca un obiect este serializat, atunci orice alt obiect care contine o referinta la obiectul serializat va fi si el insusi serializat. Se poate serializa orice multime de obiecte interconectate intr-o structura de graf.

Proiectarea pe bază de contract (DbC), cunoscută și sub denumirea de programare prin contract și programare pe bază de contract, este o abordare pentru proiectarea software-ului. Acesta prevede ca designerii de software să definească specificațiile de interfață formale, precise și verificabile pentru componentele software care extind definiția obișnuită a tipurilor de date abstracte cu precondiții, postcondiții și invariante. Aceste specificații sunt denumite "contracte", în conformitate cu o metaforă conceptuală cu condițiile și obligațiile contractelor de afaceri.

Abordarea DbC presupune că toate componentele clientului care invocă o operație pe o componentă de server vor îndeplini condițiile prealabile specificate pentru operațiunea respectivă. În cazul în care această ipoteză este considerată prea riscantă (așa cum se întâmplă în serverul client-server multiplu sau calculul distribuit) se adoptă abordarea "defensivă", ceea ce înseamnă că o componentă de server testează (înainte sau în timp ce procesează cererea unui client) adevărat și răspunde cu un mesaj de eroare adecvat dacă nu.

1-***preconditii***: reprezinta obligatiile pe care datele de intrare ale unei metode trebuie sa le respecte pentru ca metoda sa functioneze corect

2- ***postconditii***: reprezinta garantiile pe care datele de iesire ale unei metode le ofera

3-***invarianti***: reprezinta conditii impuse starilor in care programul se poate afla la un moment dat

Pentru a retine obiectele se folosesc Hashtable sau HashMap, astfel utilizand procedeul de hashing.

Pentru realizarea temei avem nevoie de cunsotinte deste HashMap.

Un HashMap este o structură de date care implementează un tip de date abstract array asociativ, o structură care poate hartă chei 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 majoritatea modelelor de tabele hash folosesc o funcție hash imperfectă, ceea ce ar putea provoca coliziunile hash în cazul în care funcția hash generează același index pentru mai multe chei. Astfel de coliziuni trebuie să fie adăpostite într-un fel.

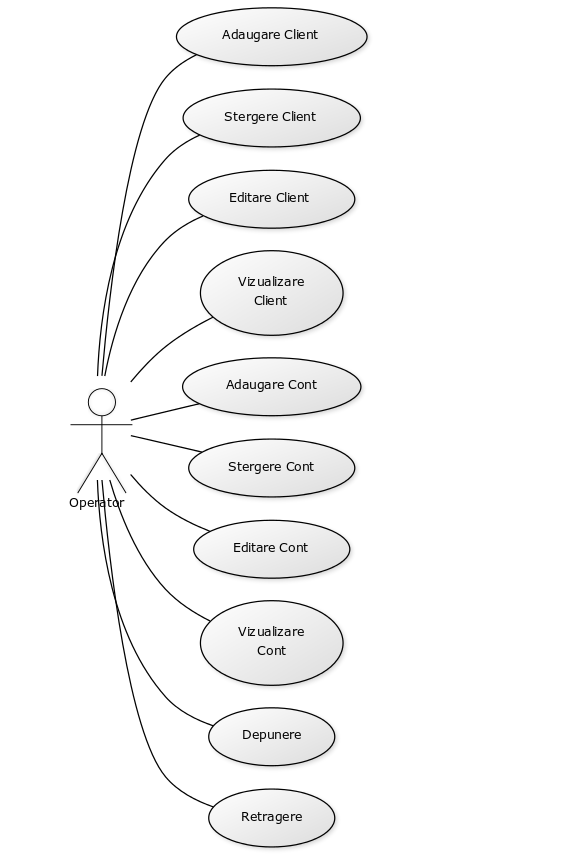
Într-o tabelă hash bine dimensionată, costul mediu (numărul de instrucțiuni) pentru fiecare căutare este independent de numărul de elemente stocate în tabel. Multe modele de tabele hash permit, de asemenea, introducerea și ștergerea arbitrară a perechilor cheie-valoare.

În multe situații, tabelele hash se dovedesc a fi în medie mai eficiente decât arborii de căutare sau orice altă structură de căutare a tabelului. Din acest motiv, ele sunt utilizate pe scară largă în multe tipuri de programe de calculator, în special în cazul matricelor asociative, indexarea bazelor de date, cache-uri și seturi.

HashMap este o clasă de colectare bazată pe hărți, care este utilizată pentru stocarea perechilor de chei și valoare, este marcată ca HashMap <Key, Value> sau HashMap <K, V>. Această clasă nu oferă nici o garanție cu privire la ordinea hărții. Este similar cu clasa Hashtable, cu excepția faptului că este nesincronizată și permite nuluri (valori nula și cheie nulă).

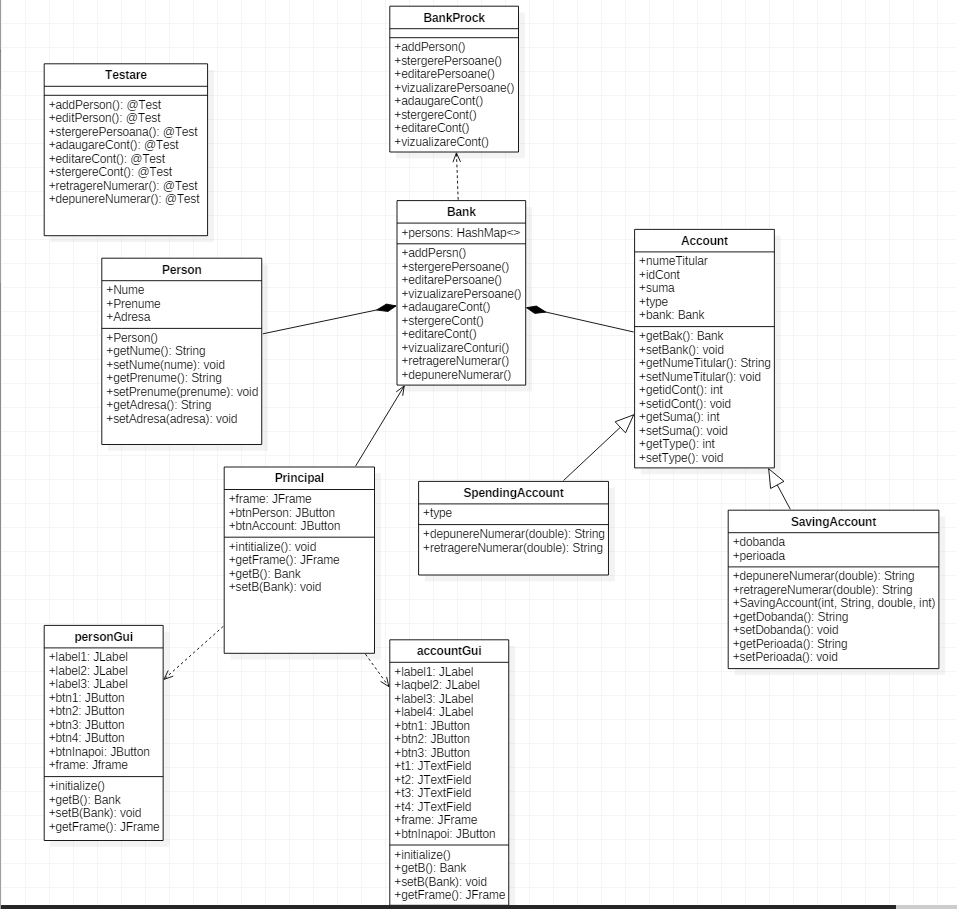
Nu este o colecție ordonată, ceea ce înseamnă că nu returnează cheile și valorile în aceeași ordine în care au fost inserate în HashMap. Nu sortează cheile stocate și Valorile. Trebuie să importați java.util.HashMap sau super clasa pentru a utiliza clasa și metodele HashMap.

**Diagrama use case a proiectului:**



1. **Proiectare**

**Diagrama UML de clase**



*Pachetul „account”*

Pachetul account contine trei clase: Accout, SavingAccount si SpendingAccount.

Conform schemei de la punctul 1 „Obiective” clasa Account este o clasa abstracta iar clase savingAccount si SpendingAccount extind aceasta clasa.

Clasa Account are ca parametrii: Id Cont, suma, si type. Daca type este egal cu 1 atunci se vor apela metodele din Spending Account, iar daca type este egal cu 0 se vor apela metodele din Saving Account. Respectiv clasa Account impelmenteaza constructorul respectiv clasei, gettere si setter.

Clasa SavingAccout se refera la un cont de economii. In plus fata de un cont obisnuit se retine dobanda si perioada depozitului. Se implementeaza settere, gettere si metodele de depunere si retragere numerar. In acest cont se poate face o singura depunere (adica crearea depozitului) si o singura retragere (desfiintarea depozitului). La desfiintarea depozitului se retrage o suma mai mare decat cea depusa, aceasta este calculate astfel: produsul dintre valoarii dobanzii / luna si perioada depozitului divizata cu 100.

Clasa Spending Account se refera la un cont obisnuit. In care nu ai restrictii la cate retrageri sai depuneri ai. Metoda de depunere numerar si retragere numera este prezentata mai jos:

**public** String depunereNumerar(**double** suma)

{

**this**.suma+=suma;

**return** "Operatiunea s-a finalizat cu succes! ";

}

@Override

**public** String retragereNumerar(**double** suma)

{

**this**.suma-=suma;

**return** "Operatiunea s-a finalizat cu succes!";

}

Pachetul „Bank”

Pachetul Bank contine 2 clase: Bank si Bank Proc.

Bank Proc este o interfata. in aceasta interfata se definesc actiunile posibile care se realizeaza asupra clientilor sau asupra conturilor acestora.

Operatii precum : adaugare persoana, stergere persoana, editare persoana, vizualizare persoane,

Adaugare cont, stergere cont, editare cont, vizularizare conturi, extragere de numerar sau adaugare de bani in cont.

Clasa Bank implementeaza operatiile enumerate in mai sus.

Mai jos vor fi prezentate cateva operatii:

Inainte de asta este mentionat ca datele se retin intr-un HashMap de tipul:

HashMap<Person, ArrayList<Account>> persons = new HashMap<Person, ArrayList<Account>>();

Adaugarea unei persoane:

**public** **void** addPerson(String nume, String prenume, String adresa) {

Person p = **new** Person(nume, prenume, adresa);

ArrayList<Account> c = **new** ArrayList<Account>();

**if** (persons.containsKey(p) == **false**) {

persons.put(p, c);

JOptionPane.*showMessageDialog*(**null**, "Person has been added!");

} **else**

JOptionPane.*showMessageDialog*(**null**, "Persoana este deja adaugata!");

}

Adaugarea unui cont:

**public** **void** adaugareCont(**int** type, String numeTitular, **double** suma, **int** idCont) {

Person p2 = **new** Person();

**if** (type == 0) // saving

{

Account c1 = **new** SavingAccount(type, numeTitular, suma, idCont);

**for** (Person p1 : persons.keySet()) {

**if** (p1.getNume().equals(numeTitular))

p2 = p1;

}

persons.get(p2).add(c1);

} **else** {

Account c2 = **new** SpendingAccount(type, numeTitular, suma, idCont);

**for** (Person p1 : persons.keySet())

p2 = p1;

persons.get(p2).add(c2);

}

}

Pentru ca sunt luate in considerare doua tipuri de conturi, inainte de a adauga un cont trebuie sa verificam de ce tip este, deoarece in functie de ce cont este operatiile se vor face diferit. Daca variabila type este egala cu 1 inseamna ca ne referim la un cont de tipul Spending Account, unde nu avem restrictii la numarul de retrageri sau depuneri. Dar daca type este egal cu 0 atunci ne referim la un cont de tipul Saving Account, adica ne referim la un depozit. La un depozit avem voie sa facem o singura depunere ( la crearea depozitului) si o singura retragere( la desfiinatarea depozitului).

Functia de retragere numerar:

**public** **void** retragereNumerar(String numeTitular, **double** sumaExtrasa, **int** type) {

Person p2 = **new** Person();

**for** (Person p1 : persons.keySet()) {

**if** (p1.getNume().equals(numeTitular))

p2 = p1;

}

**for** (Account c1 : persons.get(p2) ) {

**if** (c1.getNumeTitular().equals(numeTitular) && (c1 **instanceof** SpendingAccount)) {

((SpendingAccount) c1).retragereNumerar(sumaExtrasa);

}

}

**for** (Account c1 : persons.get(p2) ) {

**if** (c1.getNumeTitular().equals(numeTitular) && (c1 **instanceof** SavingAccount)) {

((SavingAccount) c1).retragereNumerar(sumaExtrasa);

}

}

}

Functia de depunere numerar:

**public** **void** depunereNumerar(String numeTitular, **double** suma, **int** type) {

Person p2 = **new** Person();

**for** (Person p1 : persons.keySet()) {

**if** (p1.getNume().equals(numeTitular))

p2 = p1;

}

**for** (Account c1 : persons.get(p2) ) {

**if** (c1.getNumeTitular().equals(numeTitular) && (c1 **instanceof** SpendingAccount)) {

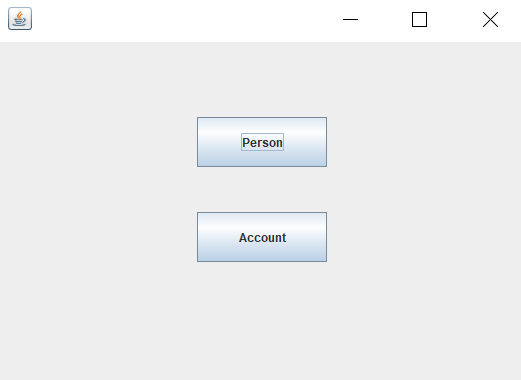
((SpendingAccount) c1).depunereNumerar(suma);

}

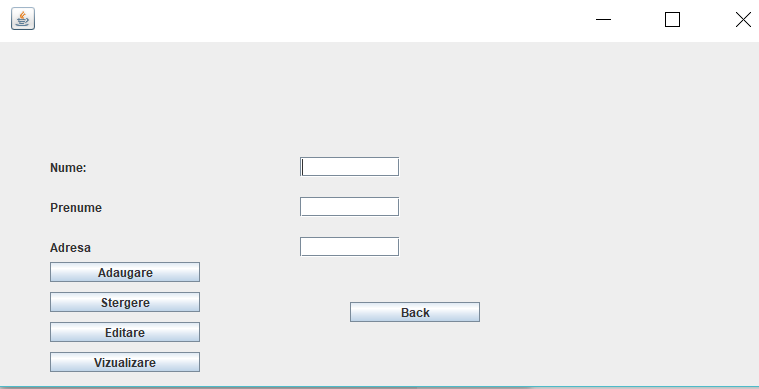
}

1. **Rezultate**

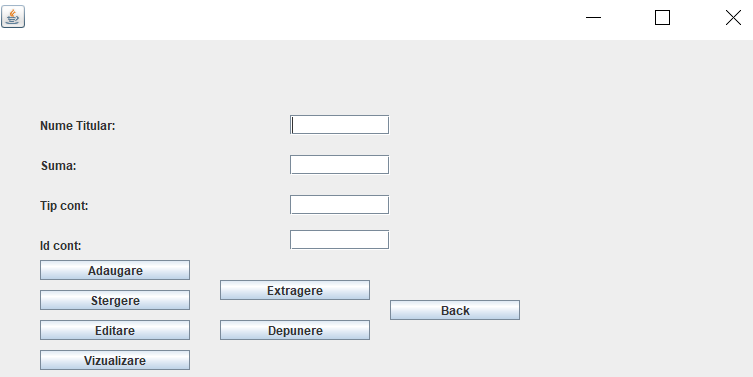
La rularea aplicatiei se va deschide frameul principal



Prima data se va da click pe butonul person, se va deschde o fereastra pentru operatiile ca se pot aplica unei personae: adaugare, stergere, editare, vizualizare.



Pentru a putea folosi orperatiile pentru account trebuie sa avem cel putin o persoana, deoarece un cont se asociaza unei persoane. Dupa ce am adaugat o persoana putem apasa pe butonul “Back” si sa intram in fereastra pentru Conturi care se va deschide dup ace apasam pe butonul “Account” din fereastra principala.



In aceasta fereastra se pot efectua operatiile pentru un cont, cum ar fi: adaugare, stergere, editare, vizualizare, extragere si depunere.

1. **Concluzii**

Aceasta tema a fost benefica prin simplul fapt ca am invatat concepte noi, pe care le-am aplicat. Concepte precum Observe/Subject, serializare/deserializare, au fost pentru prima data intalnite de mine, iar acest exercitiu a fost un prim pas in a le cunoaste mai bine.

Totodata e benefic faptul ca implementam diferite probleme in Java, pentru a ne dezvolta cunostiintele legate de tehnicile de programare orientata pe obiect.

1. **Bibliografie**

* <https://beginnersbook.com/2013/12/hashmap-in-java-with-example/>
* <https://en.wikipedia.org/wiki/Hash_table>
* <https://stackoverflow.com/questions/2257309/how-to-use-hashmap-with-jtable>
* <https://www.tutorialspoint.com/java/java_serialization.htm>
* <http://www.ms.sapientia.ro/~manyi/teaching/oop/oop_romanian/curs15/curs15.html>