**Universitatea Tehnincă Cluj Napoca**

**Facultatea de Automatică și Calculatoare**

**Secția Calculatoare si Tehnologia Informatiei**

**Gestiunea unei banci**

**Assignment 4**

**Bilc Sergiu Ion**

**30224**

**An: II**

**Cuprins**

1. Obiectivul temei

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

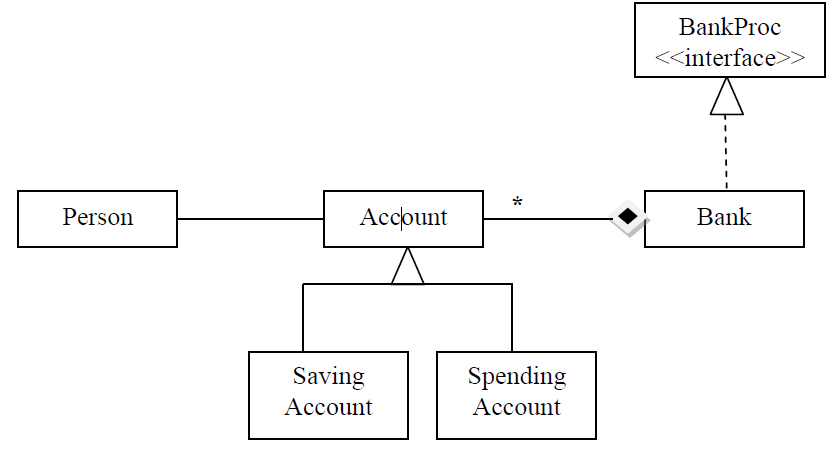
3. Proiectare (diagrame UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator)

4. Implementare si testare

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

6. Bibliografie

1.Obiectiul temei

**Cerinta**

Utilizand schema din figura de mai sus, sa se implementeze un sistem de gestiune a unei banci.

1. Definiti interfata BankProc(add/remove account, read/write account, report). Specifica preconditiile si postconditiile metodelor interfetei.
2. Implementati clasele Person, Account, SavingAccount, SpendingAccount. Se pot adauga si alte clase de care aveti nevoie (argumentati motivele pentru care le-ati adaugat).
3. Implementati clasa Bank utilizand tabel de dispersie. Pentru coliziuni se va folosi inlantuierea. . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
   1. Definiti o metoda de tip “well formed” pentru clasa Bank.
   2. Implementati clasa utilizand metoda “Design by Contract”(impicand pre, post conditii, invarianti, asertii).
4. Implementati drivere de test pentru sistem. Datele initiale pentru popularea obiectului Bank cu obiecte Account se vor lua dintr-un fisier.

**Obiectivul**

Luand in considerare specificatia problemei, ne dorim sa implementam un sistem de gestiune a unei banci, care permite utilizatorului sa aiba doua tipuri de carduri: SpendingAccount si SavingAccount, si care sa permita tranzactionarea intr-un mod rapid si sigur a banilor clientilor.

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

2.1. Analiza problemei

Inca de la inceput, avand in vedere specificatia problemei, putem sa stabilim detaliile cele mai importanta ale implementarii. Clientii vor fi pastrati intr-o tabela de dispersie, cnp-ul lor fiind cheaia tabelei, in timp ce conturile clientilor se vor pastra intr-o alta tabela de dispersie care foloseste cnp-ul clientului respectiv pe post de cheie a tabelei de dispersie.

Conturile bancare sunt de doua tipuri: conturi de economii si conturi de cheltuieli.

Conturile de economii adauga la fiecare depunere de bani, un anumit procent de dobanda. In schimb conturile de cheltuieli percep comision pentru fiecare retragere de bani si pentru fiecare vizualizare a contului.

Aplicatia este conceputa petru angajatii bancii, astfel incat sa le ofere acestora un mod rapid si uzual de gestionare a clientilor bancii, a conturilor acestora si a tranzactilor cu aceste conturi. Angajatii bancii pot sa adauge, sterge sau modifica clientii din si in banca, deasemenea, poate adauga, modifica sau sterge conturile fiecarui client si poate realiza tranzactii cu conturile amintite anterior. Pe partea de tranzactii se pot depune bani, se poate sterge sau se poate lichida contul.

**2.2. Modelare**

Am structura proiectul folosindu-ma de arhitectura MVC (Model - View - Cotroller ). Astfel, aplicatia este aranjata in 3 pachete: Models ( care reprezinta datele, obiectele, folosite de aplicatie: Bank, Account, Person, SavingsAccount,SpendingsAccount), Views(“vederile”, sectiunea grafica) si Controllers( controlorul care preia intrarile de la utilizator si le transpune in modelele cu care lucram, mai apoi ocupandu-se si de operatiile effectuate ) .

Clasele si interfetele pe care le utilizeaza aplicatia, in urma specificatiei problemei sunt urmatoarele:

* **Account**: pentru modelarea conceptului de cont la banca;
* **SpendingAccount**: o specializare a contului, fiind cont de cheltuieli(se percepe comision pentru retragere bani si interogare sold si are o limita de depuneri);
* **SavingAccount**: o specializare a contului, fiind cont de economii(pentru fiecare depunere de bani se adauga un procent de dobanda);
* **BankProc**: interfata care stabileste sablonul in care trebuie sa se incadreze o banca;
* **Bank:** pentru modelarea conceptului de banca de economii;
* **Person**: pentru modelarea unui client care solicita serviciile bancii;
* **View**: interfata grafica cu utilizatorul a aplicatiei
* **Controller**: controller folosit pentru gestiunea aplicatiei, realizeaza conexiunea intre componenta de tip Bank si utilizator, prin interfata grafica

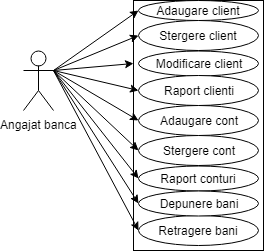
**2.3. Diagrama cazurilor**

**1.Actori**

Analizand modul de functionare dorit al aplicatiei, putem remarca ca pentru aceasta este nevoie de un singur tip de utilizator, si anume, angajatul bancii. Aceasta poate realiza toate operatiile disponibile in cadrul aplicatiei: adaugarea de clienti, stergerea clietilor, modificarea acestora, adaugarea de conturi, stergerea sau modificarea acestora si realizarea tranzactiilor pentur anumite conturi.

**2.Cazuri de utilizare**

Diagrama cazurilor de utilizare:

Pentru a putea utiliza corespunzator aplicatia, angajatul bancii trebuie sa adauge un client in banca, doar mai apoi fiind capabil sa ii adauge un cont bancar corespunzator acestuia. Sistemul ofera posibilitatea introducerii datelor de contact si astfel creaza un obiect Person, care va fi stocat intr-un hashtable in functie de cnp-ul acestuia.

Mai apoi se doreste crearea unor noi conturi pentru un anume client, conturile fiind de economii sau de cheltuieli.

O alta functionalitate importanta este cautarea conturilor si al clientilor al unei persoane i functie de cnp-ul acesteia. Aceasta cautare se face in toate punctele cheie, unde datele privitoare la conturi ar putea fi de ajutor. De asemenea orice cont si orice client poate fi sters prin specificarea cnp-ului clientului si id-ului contului.

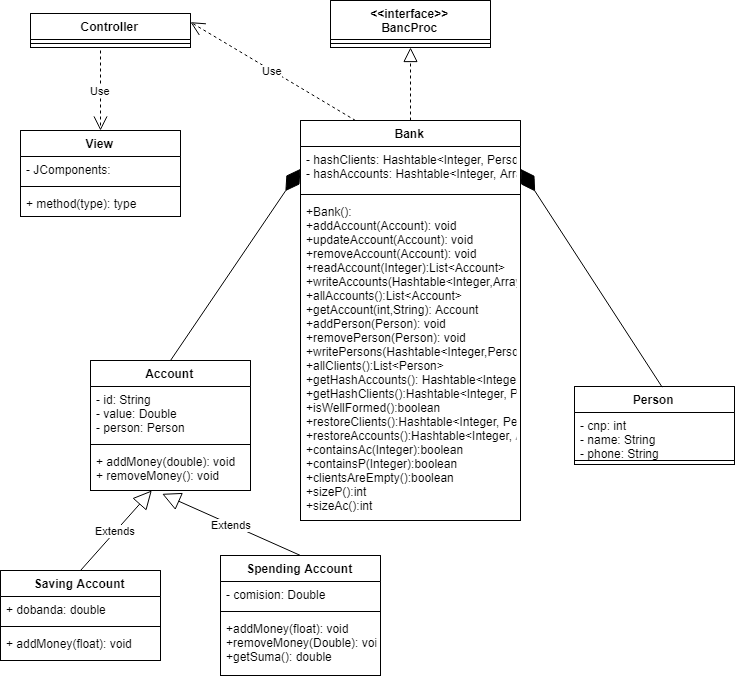
Raportat la tranzactiile necesare pentru card: se pot adauga bani in cont, se pot scoate bani din cont.

3. Proiectare

3.1. Decizii de proiectare

Am decizia sa minimezez pe cat posibil datele pe care utilizatorul este nevoit sa le introduca, astfel incat acesta are de introdus in principal date legate dnoi clienti sau de noi conturi pe care acesta doreste sa ii introduca in sistem, in timp de datele care deja se afla in banca o sa ii fie usor de accesat. De exemplu, in cazul in care utilizatorul doreste sa stearga un client, cnp-urile tuturor clientilor o sa fie afisat intr-un JcomboBox, in dupa ce utilizatorul va alege clientul dorit, acesta o sa poata sters din baza de date fara probleme.

3.2. Diagrama UML



**3.3. Structuri de date**

Pe langa tipurile de date primitive, am folosite si o serie de colectii, precum ArrayList-ul, dar cea mai importanta fiind HashTable-ul in care am decis sa stochez atat informatiile legate de clienti, cat si informatiile legate de conturi, deoarece, aceasta structura de date faciliteaza accesul rapid la informatiile de care utilizator are nevoie.

**3.4. Proiectare clase**

**Pachetul Models:**

* Clasa Person
  + Este o clasa ce formeaza nivelul cel mai de jos al aplicatiei, clasa ce reprezinta modelul din spatele clientului din viata reala.
  + Contine un cnp, un nume si un numar de telefon pentru fiecare client
* Clasa Accout
  + Este o clasa ce formeaza nivelul cel mai de jos al aplicatiei, reprezentand modelul din spatele conturilor din viata reala.
  + Este formata dintr-un id, un nume, si un obiect de tipul Person, clientul care detine contul curent.
* Clasa SavingsAccount
  + Este o clasa ce formeaza nivelul cel mai de jos al aplicatiei, reprezentand modelul din spatele contului de economii din viata reala.
  + Aceasta clasa extidne clasa account, iar in plus are un field pentru dobanda perceputa pe suma de bani existenta in cont
* Clasa SpendingsAccount
  + Este o clasa ce formeaza nivelul cel mai de jos al aplicatiei, reprezentand modelul din spatele contului de cheltuieli din viata reala.
  + Aceasta clasa extidne clasa account, iar in plus are un field pentru comisiunul perceput de banca din suma de bani existenta in cont, atunci cand se executa o extragere.
* Clasa Bank
  + Este o clasa ce formeaza banca propriu zisa din viata reala
  + Este formata dintr-un tabel de clienti si un tabel de conturi
* Interfata BancProc
  + Este o interfata care stabileste functionalitatile existente in banca(clasa Bank)

**Pachetul View:**

* Clasa View:
  + Reprezinta interfata aplicatiei

**Pachetul Controller:**

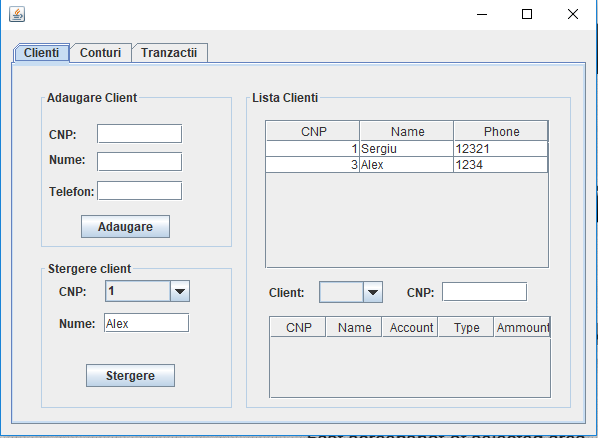
* Clasa Controller:
  + Clasa in care este realizata functionarea aplicatiei, in care sunt utilizate componentele din layerul business logic, informatile extrase din acestea fiind afisate in interfata grafica.

**Interfata cu utilizatorul**

Sistemul pune la indemana utilizatorilor o interfata grafica simpla, usor de utilizat, care acopera toate necesitatile unei banci in gestionarea eficienta si sigura a conturilor unor clienti. Aplicatia este formata din mai multe taburi, fiecare avand o fuctionalitate proprie. La deschiderea aplicatiei, datele existente in banca sunt incarcarcate i interfata, in timp ce este deschis tabul cu datele si operatiile referitoare la clienti, utilizatorul avand posibilitatea sa acceseze celelalte taburi. In pagina principala utilizatorul trebuie sa aleaga operatia pe care doreste s-o faca dintr-un meniu. Celelalte taburi sunt un tab in care se pot efectua operatii cu conturi, iar in ultimul tab se vor efectua tranzactiile pentru conturi.

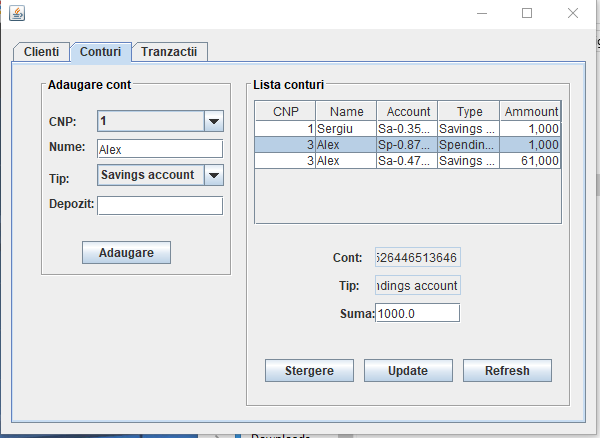
**Tab-ul Clienti**

In acest Tab sunt disponibile optiune psobile pentru un client. Si anume: in partea stanga sus, este posibila adaugarea unui nou client, utilizator fiind nevoit sa introduca datele personale ale acestuia, in partea staga jos, este posibila stergerea unui client, utilizator fiind nevoit sa selectez cnp-ul clientului pe care doreste sa il stearga, in timp ce in partea dreapta a tab-ului sunt afisati toti clientii, utilizatorul avand in acelasi timp posibilitatea de a selecta un anumit client petru care sa vada toate conturile. Acest lucru se va face atunci cand utilizatorul aplicatiei va selecta un anume client din tabelul situat in partea dreapta sus, in timp ce conturile acestuia vor aparea in partea dreapta jos, in tabelul situat in pozitia respectiva.

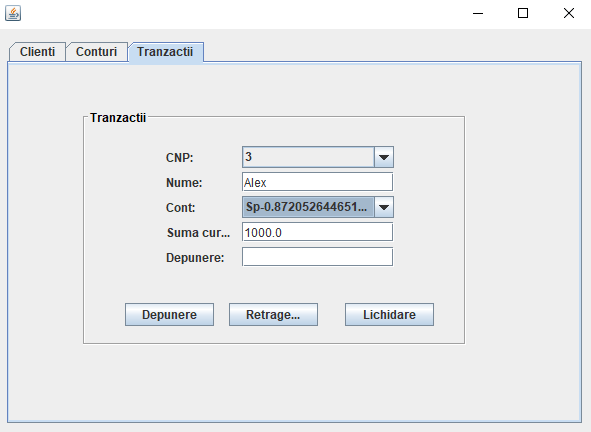


**Tabul Conturi**

In acest Tab sunt disponibile optiune posibile pentru unul sau mai multe conturi. Si anume: in partea stanga putem sa adaugam un nou cont, in timp ce in partea dreapta putem vedea intr-un tabel toate conturile existente in momentul actual in banca, impreuna cu informatiile despre aceasta. Atunci cand utilizatorul va dori introducerea unui nou cont, va fi nevoit sa selecteze cnp-ul clientului caruia doreste sa i se deschida un cont, numele acestuia fiind completat automat in functie de cnp. Apoi este necesara selectarea tipul contului ce urmeaza sa fie deschis, iar in final se va introduce suma initial care va fi introdusa in cont. La apasarea butonului adaugare, contul va fi creat. In ceea ce priveste conturile exisnte la momentul actual in banca, acestea vor fi afisate intr-un tabel, iar utilizatorul poate accesa si modidica informatiile despre acestea selectand un anume client in tabel. Tot astfel se poate realiza si stergerea unui anume cont.



**Tabul Tranzactii**

La accesarea acestei optiuni se deschide o fereastra unde se introduce cnp-ul persoanei pentru care dorim sa-i tranzactionam ceva in cont. Se selecteaza conturile acestei persoane prin alegerea din in interiorul JcomboBoxului din dreptul labelului “Cont”. La schimbarea fiecarui cnp, lista conturilor din comboBoxul amintit anterior este actualizata. Deasemenea, de fiecare data cand selectam un anume cont, suma existenta in momentul actual in cont este afisata pe ecran. De altfel, la fiecare modificare a contului, pe ecran apare suma actualizata a contului. Pentru a realiza o tranzactie, se introduce suma de bani pe care dorim s-o tranzactionam. In functie de ceea ce dorim sa facem: apasam butonul “Depune”, pentru depunerea banilor in cont si butonul “Retrage” pentru retragerea banilor din cont. Daca se incearca retragerea unei sume de bani mai mare decat cea care e in cont, sau daca se incearca depuneri care depasesc limita maxima de depuneri pentru conturile de cheltuieli, se genereaza avertismente, sistemul ramanad neschimbat, asteptand ca utilizatorul sa introduca informatii corecte.

La fiecare modificarea a continutului aplicatiei, acesta este scris in fisierele de backup.

4. Implementare

Pentru implementarea aplicatiei, am folosit limbajul de programare Java, si IDE-ul

Eclipse. In cele ce urmeaza as dori sa prezint mai in detaliu clasele cuprinse in acest proiect, explicand principalii algoritmi folositi.

**Clasa Person:** este o clasa simpla, mai mult conceptuala. Clasa modeleaza conceptul de persoana care solicita serviciile unei banci. Clasa pastreaza si furnizeaza informatii despre o persoana care isi deschide conturi in banca.

**Clasa Account:** modeleaza conceptul de cont deschis intr-o banca. Contul este identificat printr-un id unic, generat random, suma de bani pastrata in cont si persoana care detine respectivul cont. Deoarece la fiecare repornire a aplictiei, aceasta populeaza obiectul Bank cu obiecte de tip Account, clasa Account implementeaza interfata marker Serializable. Astfel obiectele Account sunt restituite din fisierul salvat pe disc.

Metodele mai importante ale clasei sunt:

**\*public void addMoney(double val):** Metoda depune o suma in cont, la suma antecedenta adaugandu-se suma transmisa ca parametru. Metoda verifica pre, post conditii si invariantii. Se verifica daca suma din cont a crescut in urma adaugarii sumei curente.

\***public void removeMoney(double val):** Metoda retrage o suma de bani din cont, daca suma solicitata exista, sau afiseaza un mesaj de avertisment cum ca suma solicitata depaseste suma aflata in cont.

\***public double getValue**():Metoda returneaza suma de bani aflata in cont.

**Clasa SpendingAccount:** extinde clasa Account. Modeleaza conceptul de cont de cheltuieli. Ideea este ca daca o persoana doreste sa aiba un depozit in banca, isi face cont de economii, nu de cheltuieli. In plus in acest tip de cont se percepe comision pentru retrageri si pentru interogare sold. Pentru depuneri, sistemul nu percepe comision deoarece e bina ca oamenii sa puna cat mai multi bani in cont, pentru ca banca sa nu falimenteze. Comisionul este unic pentru intreaga banca, indiferent de client.

Metodele specializate in clasa curenta sunt:

**\*public void addMoney(double val):** Metoda adauga o anumita suma de bani in cont.

**\*public void removeMoney(double val):** retrage o suma de bani din cont,

**\*public float getValue():** metoda returneaza suma de bani pastrata in cont

**Clasa SavingAccount:** extinde clasa Account. Modeleaza conceptul de cont de economii. Cea mai importanta proprietate e ca acest cont este ca la fiecare depunere in cont se primeste ca dobanda un procent din banii pe care tocmai ii adaug.

Metoda specializata in clasa curenta este:

**\*public void addMoney(float val):** Metoda adauga o anumita suma de bani in cont si dobanda aferenta respectivei sume de bani.

**Clasa Bank:** clasa modeleaza conceptul de banca, cu functionalitatea si atributele intalnite in cadrul unei banci, cat si conditiile care trebuie respectate la fiecare pas al executiei. Banca este o compozitie de obiecte de tip Account: care pot fi de tip SpendingAccount sau SavingAccount, impreuna cu obiecte de tipul Person, ce reprezinta clientii bancii. Banca este reprezentata prin doua tabele de dispersie: una de de tip Hashtable<Integer,ArrayList<Account>>, in timp ce cealalta este de tipul Hashtable<Integer,Person>,. Cheia dupa care se face dispersia in ambele tabele este cnp-ul clientului. In caz de coliziune, conturile se leaga prin inlantuire. In acest sens in tabela de dispersie pastram o lista de Account-uri, pentru a folosi inlantuirea in rezolvarea coliziunilor.

Constructorul clasei bank: creaza tabele de dispersie, si incearca reconstituirea lui pe baza obiectelor scrise in doua fisiere de pe disk.

Metodele clasei sunt urmatoarele:

**\*public void addAccount(Account a):** metoda adauga un cont in banc ape pozitia cheii specificate prin cnp-ul clientului, care se gaseste in Account. Mai intai se testeaza preconditiile si invariantul. Daca clientul pentru care se inregistreaza contul, mai are si alte conturi, se acceseaza lista de conturi, se adauga contul curent, apoi se reinsereaza lista actualizata. Altfel se creaza o lista formata doar din contul curent si se insereaza in tabelul de dispersie. La final se verifica daca tabelul a ramas tot bine format, si daca numarul de conturi a crescut cu 1. Daca una din aceste asserturi esueaza, se genereaza exceptie.

**\*public void addPerson(Person a):** metoda adauga un client in banca. Mai intai se testeaza preconditiile si invariantul. Dacă clientul este deja inserat, atunci acesta este suprascris. La final este verificat invariantul și faptul dacă clientul a fost adaugat cu succes.

**\*public void updateAccount(Account a):** metoda updateaza un cont din banca. Mai intai se testeaza preconditiile si invariantul. La final este verificat invariantul și faptul dacă clientul a fost adaugat cu succes.

**\*public void removeAccount(Account a):** metoda sterge un cont din tabelul de dispersie. Initial se verifica tabelul sa fie vid, sis a fie bin format. Se cauta clientul pentru care stergem contul, parcurgem lista conturilor si unde gasim un cont identic cu cel pe care dorim sa-l stergem, stregem respectivul cont. In cazul in care in urma stergerii lista de conturi ramane vida, se sterge si cheia de identificare a clientului, eliberandu-se astfel un spatiu in tabela de dispersie. La final se verifica sa ramana tabelul bine format, si numarul de conturi sa scada cu 1.

**\*public void removeClient(String cnp):** metoda sterge un client din tabelul de dispersie. Initial se verifica sa existe in tabela cnp-ul specificat ca parametru si ca banca sa fie bine formata. Se sterge campul alocat respectivei chei si astfel se elimina orice informatie legata de respectiva persoana si conturile pe care le detinea la banca. In final se verifica sa ramana tabela bine formata, si ca numarul de elemente ale tabelului sa scada cu 1.

**\*public ArrayList<Account> readAccount(String key):** metoda returneaza o lista de conturi pentru o persoana specificata prin cnp-ul transmis ca parametru. Initial se verifica sa nu fie o cheie de identificare unica si ca tabelul sa fie bine format. Se acceseaza lista corespunzatoare cheii si se returneaza.

**\*public ArrayList<Account> allAccounts():** metoda returneaza o lista a tuturor conturilor din banca. Initial se verifica sa nu fie vid tabelul si sa fie bine format. Se genereaza o enumerare a cheilor tabelei, apoi se acceseaza lista de conturi corespunzatoare fiecarei chei. Toate aceste liste de noduri se adauga unei liste declarate in metoda, lista care se va returna, fiind considerata traportul bancii.

**\*public ArrayList<Client> allClients():** metoda returneaza o lista a tuturor clientilor din banca. Initial se verifica sa nu fie vid tabelul si sa fie bine format. Se genereaza o enumerare a cheilor tabelei, apoi se acceseaza lista de conturi corespunzatoare fiecarei chei. Toate aceste liste de noduri se adauga unei liste declarate in metoda, lista care se va returna, fiind considerata traportul bancii.

**\*public int sizeAc():** metoda returneaza numarul de conturi aflate in banca. Se genereaza o enumerare de chei, se acceseaza listele de conturi corespunzatoare fiecarui nod, si dimensiunea lor se aduna, constituind numarul de conturi din banca. Acest numar se returneaza.

**\*public int sizeP():** metoda returneaza numarul de clienti aflati in banca. Se genereaza o enumerare de chei, se acceseaza listele de conturi corespunzatoare fiecarui nod, si dimensiunea lor se aduna, constituind numarul de conturi din banca. Acest numar se returneaza.

**\*public boolean isWellFormed():** metoda verifica daca tabelul de dispersie al bancii e bine format. Se parcurge tot tabelul: se verifica numarul de intrari in tabela returnat de functii sa fie acelasi cu numarul parcurgerilor, si ca dimensiunea fiecarei liste sa fie aceeasi cu dimensiunea obtinuta prin parcurgerea listei.

**\*public Hashtable<Integer,ArrayList<Account>> restoreAccounts():** metoda creeaza un table de dispersie cu date salvate in fisierul de pe disc corespunzător conturiilor. Tabelul este returnat, si prin el se face reconstructia bancii.

**\*public Hashtable<Integer, Product> restoreClients():** metoda creeaza un table de dispersie cu date salvate in fisier pe disc corespunzător clientilor. Tabelul este returnat, si prin el se face reconstructia bancii.

**\*public void writeAccounts(Hashtable<Integer,ArrayList<Account>> h):** metoda scrie in fisier pe disc tabelul de dispersie corespunzător conturilor.

**\*public void writeClients(Hashtable<Integer, Person> h):** metoda scrie in fisier pe disc tabelul de dispersie corespunzător clientilor

**Testarea**

Pentru testare am folosit tehnica Junit pentru cele mai importante metode ale aplicatiei. . Aceasta este o facilitate a limbajului Java, care ofera posibilitatea testarii moderne a codului, pe parcursul evolutiei programului, fara a fi nevoiti sa asteptam pana la finalul proiectului, si fara a depune prea mult efort. Un test facut e bun facut, si mici modificari aduse unei metode nu necesita introducerea de noi teste. Se pot rula cele existente deja in proiect.

Tot pentru o testare buna am implementat preconditii, postconditii si un invariant pentru clasa Bank. Assertiile implementate verifica si ele, ca sistemul sa ruleze si sa-si desfasoare activitatea in limitarile normale.

5. Concluzii si dezvoltari ulterioare

Prin intermediul acestei teme am invatat sa folosesc metoda Design by contracts, prin implementarea unor preconditii, postconditii si a unui invariant al clasei Bank. De asemenea am invatat sa folosec Hashtable. De asemenea am aprofundat unele cunostinte proaspat acumulate, cum ar fi serializarea si lucrul cu colectii. Consider ca aceasta tema m-a ajutat sa aprofundez cunostintele mele generale in domeniul Java si sa invat alte tehnici importante.

Dezvoltari ulterioare:

* sa existe posibilitatea tiparirii pe chitanta a tuturor operatiilor efectuate pe cont;
* sa fie validate datele de intrare;
* cand se sterge un cont sau o persoana cu o lista de mai multe conturi, sa apara un mesaj cu suma care trebuie eliberata respectivei persoane
* sa se extinda datele retinute despre clienti
* introducerea de noi tranzactii posibile . Ex : transfer bancar.
* Sa se pastreze o evidenta e tuturor modificarilor efectuate in cont ; modificarile sa se puna intr-un fisier si sa fie un fel de arhiva a bancaii.

6. Bibliografie

[1] [http://coned.utcluj.ro/~salomie/PT\_Lic/4\_Lab/](http://coned.utcluj.ro/~salomie/PT_Lic/4_Lab/ )