Student : Layis Norbert

Grupa : 30237

Management al conturilor unei banci

-tema4-

Student: Layis Norbert

Grupa : 30237

Cuprins

1.Obiectivul temei

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

3. Proiectare

4. Implementare si Testare.

5.Rezultate.

6. Concluzii si dezvoltari ulterioare .

7. Bibliografie

**1.Obiectivul temei**

In prima faza, obiectivul temei este de a implementa un sistem de gestiune a conturilor dintr-o banca. Se cere definirea interfetei cu numele *BankProc* cat si metodelor acesteia. Aici se vor specifica preconditiile si postconditiile fiecarei metode.

Informatia va fi retinuta intr-o structura de tip hashtable .

In continuare, se cere implementarea unor clase specifice: *Person, Account, SavingAccount* si *SpendingAccount.*

Clasele vor fi implementate utilizand metoda “Design by Contract” ceea ce implica pre, post conditii, invariant si asertii.

Ca mediu de comunicare cu utilizatorul , se va folosi un GUI creat folosind tehnologii SWING.

Ca date de intrare, aplicatia va avea:

* Numele clientului in cauza
* Tipul contului introdus, cat si suma
* Pentru efectuarea tranzactiilor vom avea: suma pentru depozit, pentru retragere, cat si id-urile conturilor aferente.

Ca date de iesire, aplicatia va furniza:

* O structura tabelara cu toti clientii bancii
* O structura tabelara cu toate tranzactiile efectuate

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

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

**2.1. Analiza problemei**

In primul rand, problema cere implementarea unei interfete grafice pentru a putea comunica efficient cu utilizatorul si pentru a afisa evolutia cozii. Pentru acest lucru voi folosi Swing .

Pentru realizarea interfetei grafice voi avea nevoie de o fereastra principala:

* Fereastra caree va contine meniurile

In aceasta fereastra principala, voi avea nevoie de trei meniuri care vor servi la ajutarea utilizatorul pentru introducerea informatiei:

* meniul pentru introducerea informatiei despre clienti si cont
* meniul de tranzactii
* meniul de afisare a bazei de date cu clienti

In al doilea rand, avem nevoie de o serie de entitati ce sa abstractizeze notiunile intalnite in cadrul

unei banci. Inca din speicificatia problemei aceste clase si o interfata sunt definite:

* Clasa ce sa abstractizeze notiunea de banca
* Interfata ce sa defineasca metodele bancii
* Clasa ce sa abstractizeze notiunea de persoana
* Clasa ce sa abstractizeze notiunea de cont
* Clasa ce sa abstractizeze notiunea de Spending account, clasa derivate din cea de cont
* Clasa ce sa abstractizeze notiune de Saving account, clasa derivata din cea de cont

**2.2. Modelare**

Dupa cum reiese din analiza problemei as avea nevoie de un total de 10 clase:

* *MyFrame* : clasa pentru fereastra principala ce va tine toate panourile; extinde JFrame
* *Bank :* clasa ce abstractizeaza notiunea de banca, unde se retin datele aferente unei banci
* *BancProc :* interfata ce defineste toate metodele folosite de clasa *Bank*
* *Person :* clasa ce abstractizeaza notiunea de persoana
* *Account :* clasa ce abstractizeaza notiunea de cont
* *SpendingAccount:* clasa ce abstractizeaza notiunea de cont la vedere, clasa extinsa din *Account*
* *SavingAccount :* clasa ce abstractizeaza notiunea de cont de credit, clasa extinsa din *Account*
* *HashTable : clasa ce implementeaza metodele ce controleaza un hash table*
* *BankTest si NewEmptyJUnitTest : clase de test*

**2.3. Scenarii si cazuri de utilizare**

In aceasta sectiune se vor analiza si urmarii raspunsurile programului la posibile scenarii si cazuri de utilizare ale acestuia.

Odata pornita aplicatia, utilizatorul este introdus direct in panoul de conturi, acesta putand deja introduce noi conturi in banca.

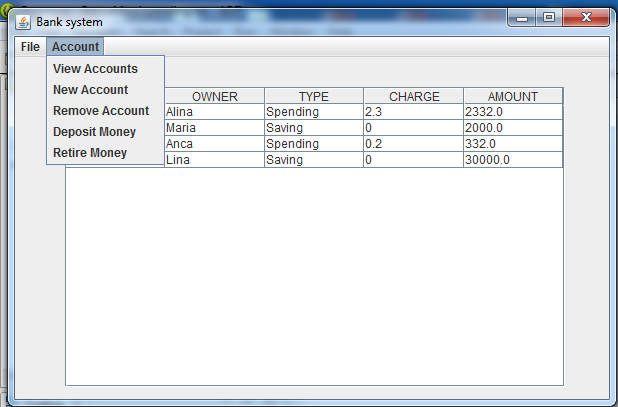


In continuare vor fi prezentate doua cazuri de utilizare: **cazul ideal** si **cazul defavorabil.**

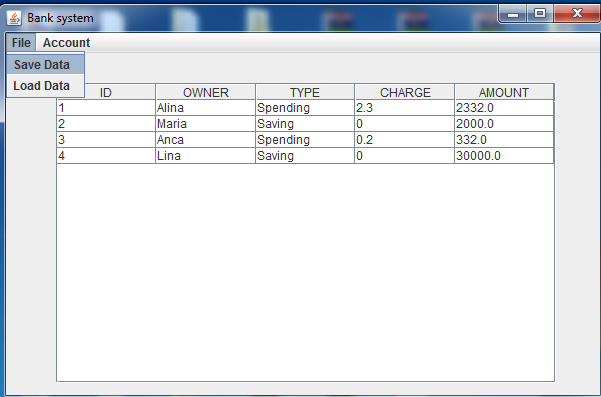
**Cazul ideal.** Cazul ideal este cel in care utilizatorul introduce corect datele de input (Numele si suma ) si foloseste in mod corespounzator aplicatia.

Odata introduse o serie de date initiale, aceste date se pot prelucra. In continuare voi expune metodele prin care putem prelucra un cont.

In continuare, utilizatorul poate efectua o serie de operatii pe conturi, depunere, retragere, stergere de cont.

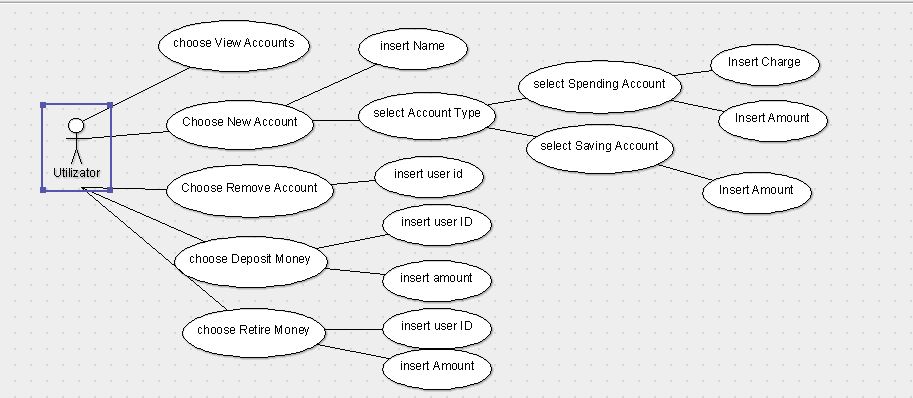


O alta operatie ce o poate executa utilizatorul este de a salva toate datele introduce in fisiere si la revenirea in aplicatie de a da import pentru a nu pierde datele prelucrate anterior.

**

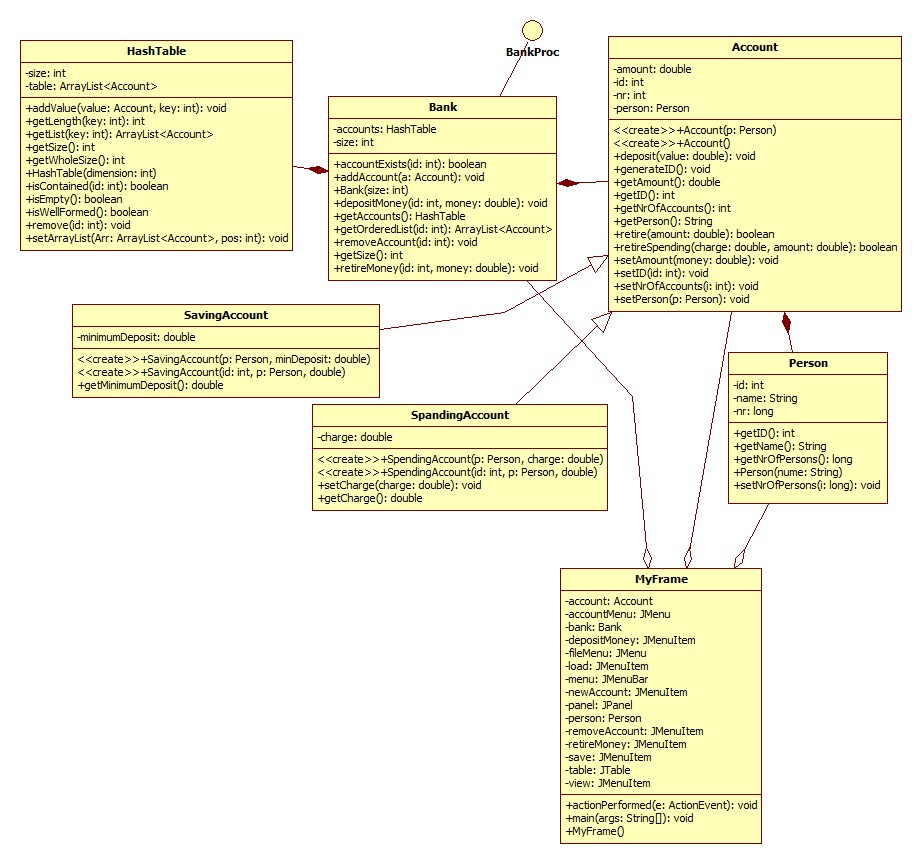
**Cazul defavorabil.** In acest caz vom studia raspunsul aplicatiei la inputuri invalide din partea utilizatorului. Consideram fereastra principala si introducem data de intrare incorecte.

Diagrama USE CASE



**3. Proiectare**

**3.1. Diagrame UML – diagrama de clasa**

****

**3.2. Relatii UML**

In aceasta sectiune se vor explica relatiile diagramei UML a aplicatiei:

* **HashTable – Bank.** Aceasta relatie este o compozitie .
* **MyFrame – Bank.** Doar un obiect de tipul *Bank* se instantiaza per aplicatie, deci per fereastra principala. Rezulta ca relatia dintre cele doua clase este una de agregare.
* **Bank – BankProc.** Intre aceste doua clase relatia este de clasa – interfata. *BankProc* fiind interfata, iar *Bank* implementand aceasta interfata.
* **Bank – Account.** Intre aceste doua clase relatia este de compozitie.
* **SavingAccount – Account.** Relatia este de generalizare. Clasa *Account* este o generalizare a clasei *SavingAccount*, cea din urma mostenind-o.
* **SpendingAccount – Account.** Analog cazului anterior.
* **Account – Person.** Relatia este de compozitie.

**3.3. Proiectarea claselor**

In aceasta sectiune se vor prezenta detaliile fiecarei clase si de asemenea utilitatea ei.

**Account**

Clasa ce abstractizeaza notiunea de cont . Contine toate detaliile referitoare la un cont: numarul acestuia, suma depusa in acesta cat si persoana asociata. Metodele aferente clasei sunt simple settere si gettere plus doua metode de de retragere dintr-un Saving Account si un Spending Account. Aceasta metoda este folosita in scopul de a retrage bani dintr-un cont de tip " Spending Account " sau “ Saving Account ” . Suma de bani care va fi retrasa este reprezentata de parametrul amount . Metoda returneaza true in caz de succes si false in cazul in care suma retrasa este

mai mare decat suma de bani disponibila in cont .

**Bank**

Clasa ce abstractizeaza notiunea de banca. Clasa contine toate detaliile referitoare la o banca: multimea de conturi reprezentate prin hash table. Ca si metode aferente se pot mentiona cea de adaugare cont, stergere cont , depunere in cont, retragere din cont . Aceasta clasa implementeaza metodele definite in interfata *BankProc*.

**BankProc**

Interfata ce contine metodele ce trebuie implementate de clasa *Bank.*

**HashTable**

Clasa hashtable implementeaza metode care fac operatii asupra unui hashtable si contine o metoda de tip isWellFormed() care verifica principalele caracteristici ale hashtable-ului returnand true in caz ca sunt indeplinite sau false in caz contrar.Contine metoda isEmpty() care determina daca hashtable-ul este gol sau metode isContaind() care verifica daca o cheie este continuta sau nu de tabel.In acest HashTable sunt stcate conturile.

**MyFrame**

Clasa ferestrei principale ce contine metoda main a programului. Aici se instantiaza toate obiectele ce compun aplicatia: toate panourile si entitatile aferente.Interfata grafica este sub forma tabelara, fereastra principal continand 2 meniuri fecare cu alte submeniuri.In primul meniu utilizatorul poate salva datele prelucrate sau poate importa baza de date a bancii.Al doilea meniu ofera utilizatorului posibilitatea de a vizualiza tabelul cu clientii bancii sau de a face operatii asupra conturilor, la fiecare operatie asupra conturilor aleasa utilizatorului I se va deschide o noua fereastra de dialog care ii va cere sa introduca anumite date pentru introducerea cat mai clara a clientilor in baza de date.

**Person**

Clasa ce abstractizeaza notiunea de persoana (client). Clasa contine toate detaliile referitoare la o persoana: nume, si id care este obtinut prin incrementarea unei variabile. Ca metode clasa implementeaza settere si gettere pentru atributele specificate.

**SavingAccount**

Clasa ce extinde clasa parinte *Account*. Clasa abstractizeaza notiunea de cont de credit.

**SpendingAccount**

Clasa ce extinde clasa parinte *Account.* Clasa abstractizeaza notiunea de cont la vedere.

**3.4. Algoritmi**

In aceasta sectiune se vor descrie algoritmii folositi pentru implementarea aplicatiei.

3.4.1. **Algoritm de tranzactie (depunere, retragere, lichidare).** Se introduc datele de tranzactie (contul/conturile, cat si suma), se selecteaza din tabelul de hash contul in cauza si se actualizeazaza tranzactia (pentru depunere – se mareste suma, pentru retragere – se decrementeaza suma, pentru pentru lichidare se seteaza la 0 suma contului selectat).

**4. Implementare**

**4.1. Implementarea algoritmilor**

In aceasta sectiune se vor prezenta efectiv metodele care implementeaza algoritmii descrisi in sectiunea anterioara.

**public void depositMoney**(int id, double money) {

assert accountExists(id);

assert money > 0;

ArrayList<Account> current;

current = getOrderedList(id);

int pos = id / size;

current.get(pos).deposit(money);

}

Aceasta metoda este folosita pentru a depune o anumita suma de bani intr-un cont dat de id.

@param **id** - id-ul contului in care banii vor fi depusi

@param **money** - suma de bani care va fi depusa in cont .

**public void retireMoney**(int id, double money) {

assert accountExists(id);

assert money > 0;

ArrayList<Account> current;

current = getOrderedList(id);

int pos = id / size;

if (current.get(pos) instanceof SpendingAccount) {

current.get(pos).retireSpending(

((SpendingAccount) current.get(pos)).getCharge(), money);

} else if (current.get(pos) instanceof SavingAccount) {

current.get(pos).retire(money);

}

}

}

Aceasta metoda este folosita pentru a retrage o anumita suma de bani intr-un cont dat de id.

**@param** id - id-ul contului din care suma de bani va fi retrasa .

**@param** money - suma de bani care va fi retrasa din cont .

Un alt element important este interfata BankProc deoarece ea defineste toate post si preconditiile si invariantii:

/\*\*

\* **@pre** a != null

\* **@pre** !accountExists(a.getID())

\* **@post** !isEmpty()

\* **@post** a.size() = a.@pre.size()+1

\* **@post** accountExists(a.getID())

\*/

**public** **void** addAccount(Account a);

/\*\*

\* **@pre** !isEmpty()

\* **@pre** accountExists(a.getID())

\* **@post** Account.size() = Account@pre.size()-1

\* **@post** !accountExists(a.getID())

\*/

**public** **void** removeAccount(**int** id);

/\*\*

\* **@pre** !isEmpty()

\*

\*/

**public** HashTable getAccounts();

/\*\*

\* **@pre** !isEmpty()

\* **@pre** accountExists(id)

\* **@pre** money > 0

\* **@post** account.getAmount() == account@pre.getAmount() + amount

\*/

**public** **void** depositMoney(**int** id, **double** money);

/\*\*

\* **@pre** !isEmpty()

\* **@pre** accountExists(id)

\* **@pre** money > 0

\* **@post** account.getAmount() == account@pre.getAmount() - amount

\*/

**public** **void** retireMoney(**int** id, **double** money);

**public** **boolean** accountExists(**int** id);

/\*\*

\* **@pre** !isEmpty()

\* **@pre** accountExists(id)

\*

\*/

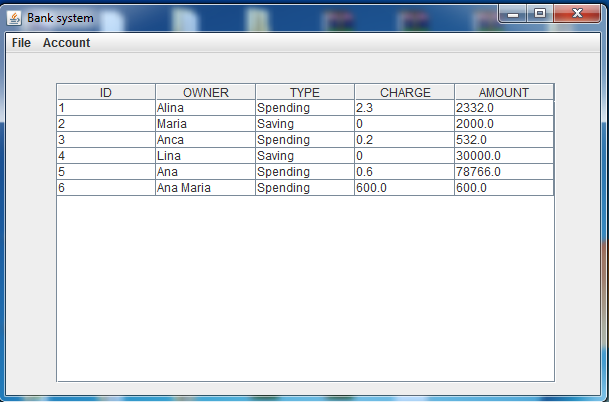
**public** ArrayList<Account> getOrderedList(**int** id);

}

**4. TESTARE**

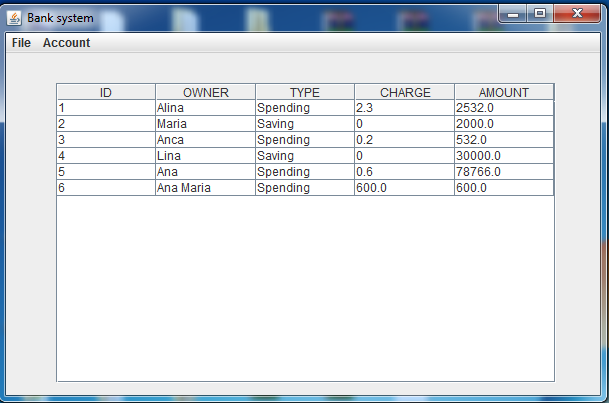
**4.2.1.** **Testarea introducerii unui cont.**

Pentru a testa introducerea unui cont furnizam ca data de input numele “Ana Maria”, iar suma “600”. Se poate observa ca numele este corect introdus in structura tabelara.

****

**4.2.2.** **Testarea efectuarii unei tranzactii.**

Se presupune efectuarea urmatoarei tranzactii: depunerea in contul cu id-ul 1 suma de 200.

****

**4.2.3.** **Testarea serializarii.**

Dupa formarea structurii tabelare dorite, se apasa pe meniul Save Data, apoi la importarea informatiei in interfata se apasa pe Import Data.

**5.Rezultate**

Dupa toate testele effectuate observam urmatoarele rezultate:

* Datele se introduce corect in tabelul de hash
* Datele se iau corect din tabelul de hash
* Datele se afiseaza corect in structura tabelara
* Datele se salveaza corect prin serializare.

**6.Concluzii si dezvoltari ulterioare**

**6.1. Ce am invatat**

Pe parcursul dezvoltarii acestei teme mi-am dezvoltat abilitatile de programator, invatand lucruri noi, atat algorithmic, cat si unele ce tin explicit de limbajul de programare Java. De asemenea am patruns mai adanc in tehnicile de programare orientat obiect, modeland solutia acestei teme.

Punctele cele mai importante invatate pe parcursul rezolvarii temei au fost:

* Folosirea metodei proiectarii prin contract
* Folosirea

**6.2. Dezvoltari ulterioare**

Bineinteles, programul nu este complet finisat, putand suferi unele aditionari, atat algoritmic cat si visual. In cele ce urmeaza voi prezenta update-urile ce se pot aduce rezolvarii problemei:

* Se poate introduce o statistica sup forma grafica a tranzactiilor si balansului fiecarui cont.
* Se poate apela la o interfata mai prietenoasa cu utilizatorul
* Se pot rezolva unele problem de identificare a clientilor
* Adaugarea unor noi campuri afisarii tabelare

**7.Bibliografie**

* Atac la Java, Sierra&Bates
* Thinking in Java, 4th Edition, Bruce Eckel
* <http://download.oracle.com/javase/tutorial/uiswing/layout/spring.html>
* Cursul “Diagrame UML”, Ioan Salomie
* Bruce Eckel – Thinking in Java 4’th edition ;
* Cristian Frasinaru – Curs practic de Java ;
* Tutorials Point , Simply Easy Learning – Tutorialspoint.com
* Java de la 0 la expert (editia a II-a) , Stefan Tanasa ,Stefan Andrei, Cristian Olaru , Editura: Polirom
* Atac la Java (ed. Teora) ,Kathy Sierra &Bert Bates
* JDKTM 6 Documentation
* <http://java.sun.com/j2se/1.4.2/docs/api/>