

Sistem de gestionare a unei bănci

Halasz Lorand Daniel

Grupa 30226

Cuprins

[1. Obiectivul temei 3](#_Toc513183835)

[2. Analiza si modelarea problemei 4](#_Toc513183836)

[2.1. Cazuri de utilizare si scenarii 4](#_Toc513183837)

[2.2. Asumptii 5](#_Toc513183838)

[3. Proiectare 6](#_Toc513183839)

[3.1. Decizii de proiectare 6](#_Toc513183840)

[3.2. Diagrama UML 7](#_Toc513183841)

[3.3. Structuri de date 8](#_Toc513183842)

[3.4. Proiectare clase 8](#_Toc513183843)

[3.5. Pachete 9](#_Toc513183844)

[3.6. Interfata utilizator 10](#_Toc513183845)

[3.7. Modul de tratare a erorilor 11](#_Toc513183846)

[4. Implementare 12](#_Toc513183847)

[1. Clasa **Person** 12](#_Toc513183848)

[2. Clasa **Account** 13](#_Toc513183849)

[3. Clasa **SavingAccount** 13](#_Toc513183850)

[4. Clasa **SpendingAccount** 14](#_Toc513183851)

[5. Clasa **Bank** 15](#_Toc513183852)

[6. Clasa **UnitTest** 16](#_Toc513183853)

[7. Clasa **MainFrame** 17](#_Toc513183854)

[5. Testare 18](#_Toc513183855)

[6. Rezultate 19](#_Toc513183856)

[7. Concluzii 20](#_Toc513183857)

[8. Bibliografie 21](#_Toc513183858)

1. Obiectivul temei

In ziua de azi, domeniul financiar si comertul sunt doua dintre cele mai dezvoltate domenii din intreaga lume, fie ca vorbim de comert online sau de comert traditional. Strans legate de acestea, si fara de care procesul de dezvoltare era mult incetinit, este domeniul bancar. In ziua de astazi banca reprezinta mai mult decat o institutie publica sau privata, in care omul merge sa depuna si sa retraga bani. Banca reprezinta motorul intregului domeniu financiar, intrucat prin intermediul bancilor se realizeaza tranzictiile din intreaga lume. Obiectivul principal al temei consta in propunerea, proiectarea si implementarea unui sistem de gestionare a unei banci. Obiectivul principal poate fi descompus in mai multe obiective secundare, si anume:

* Implementarea diferitelor operatii pe tabele, precum:
  + adaugare
  + modificare
  + stergere
  + vizualizare continului tabelului
* implementarea unei interfete grafice
* deschiderea unor noi conturi
* utilizarea tabelei de dispersie
* implementarea operatiilor de depunere si retragere numerar
* utilizarea pattern-ului Observer
* separarea conturilor in doua mari subtipuri, cont de economi si cont de cheltuieli
* salvarea datelor intr-un fisier prin serializare

1. Analiza si modelarea problemei

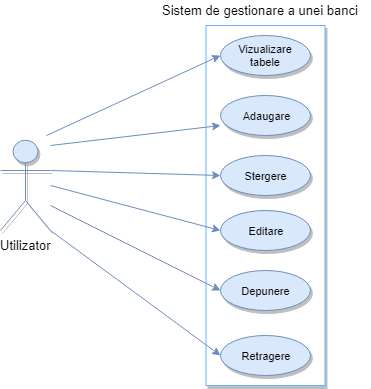
Analiza problemei consta in identificarea domeniului problemei. De cele mai multe ori, problema fiind una complexa urmeaza a fi descompusa in subprobleme pentru a ne usura munca. Aceste subprobleme pot fi, la randul lor, descompuse in alte subprobleme, acest pas repetandu-se pana in momentul in care se ajunge la un nivel de dificultate minim, care poate fi rezolvat. Urmatorul pas consta in identificarea domeniului solutiei. Acest lucru este realizabil prin aplicarea in repetate randuri a unui proces de abstractizare, pana cand se obtine ceea ce se doreste.

Odata analizate aceste lucruri, se formeaza o idee generala asupra problemei. Urmatorul pas consta in descrierea cadrului de cerinte functionale, ceea ce se reduce la prezentarea modului in care sistemul reactioneaza in anumite situatii si la anumite intrari. In cazul de fata, pentru sistemul de gestionare a unei banci, se cunoaste faptul ca trebuie sa poata efectua anumite operatii ( de adaugare, de stergere, de modificare si de vizualizare a campurilor din tabele) si sa ofere o interactiune mai usoara pentru utilizator, prin intermediul unei interfete grafice. Tot in cadrul analizei problemei se considera faptul ca in momentul in care se realizeaza o operatie de depunere sau retrage, este necesara o actualizare a soldului curent cat si atribuirea dobanzii aferente contului de economi, iar in cazul in care nu se dispune de soldul necesar pentru a putea retrage suma dorita, sa nu se efectueze nicio operatie.

Procesul de dezvoltare software se realizeaza pornind de la anumite nevoi. Acestea sunt translatate in cerinte, care urmeaza a fi validate de un client. Pe urma se concepe design-ul prin proiectarea de ansamblu si de detaliu. Se implementeaza componentele si se testeaza separat, dupa care acestea se integreaza in sistem, care la randul sau trebuie validat si testat ca intreg.

* 1. Cazuri de utilizare si scenarii

Cazurile de utilizare reprezinta un set de scenarii legate de modul in care este utilizat sistemul. Acestea ilustreaza modul in care un actor (in cazul de fata un utilizator) interactioneaza cu sistemul. Diagrama de utilizare (diagrama use-case) ne ajuta sa descoperim entitatile sitemului, atributele, actorii si comportamentul sistemului.

 Pentru inceput, daca utilizatorul doreste sa vizualizeze tabelele, este necesar sa apese pe butonele din dreptul fiecarui tabel, care poarta denumirea de “Vizualizare”, urmata de numele tabelului pe care doreste sa il vada. Apoi, in functie de ceea ce doreste utilizatorul sa faca, acesta trebuie sa introduca datele necesara dupa care trebuie sa apese pe butonul corespunzator comenzii alese.

In cazul in care se doreste modificarea unui client, dupa introducerea datelor necesare si apasarea butonului de editare este, de asemenea, necesara, aparea butoanelor de reset si vizualizare personae pentru a vedea modificarile datelor clientului.

Introducerea datelor in mod corect reprezinta un best scenario, de aceea utilizarea sitemului in mod corect presupune introducerea datelor respectand semnificatia campurilor. Totusi, acest lucru nu este sigur si ne putem astepta la diferite lucruri, dintre care:

* Introducere unui text in locul unei valori
* Incercarea de a retrage o suma mai mare decat cea disponibila in cont
* Introducerea unui nou client, cu un id identic cu id-ul altui client deja existent. Acest lucru este valabil pentru toate tabelele.
* Neintroducere unor date indispensabile, precum id-ul titularului unui cont sau a id-ul contului
  1. Asumptii

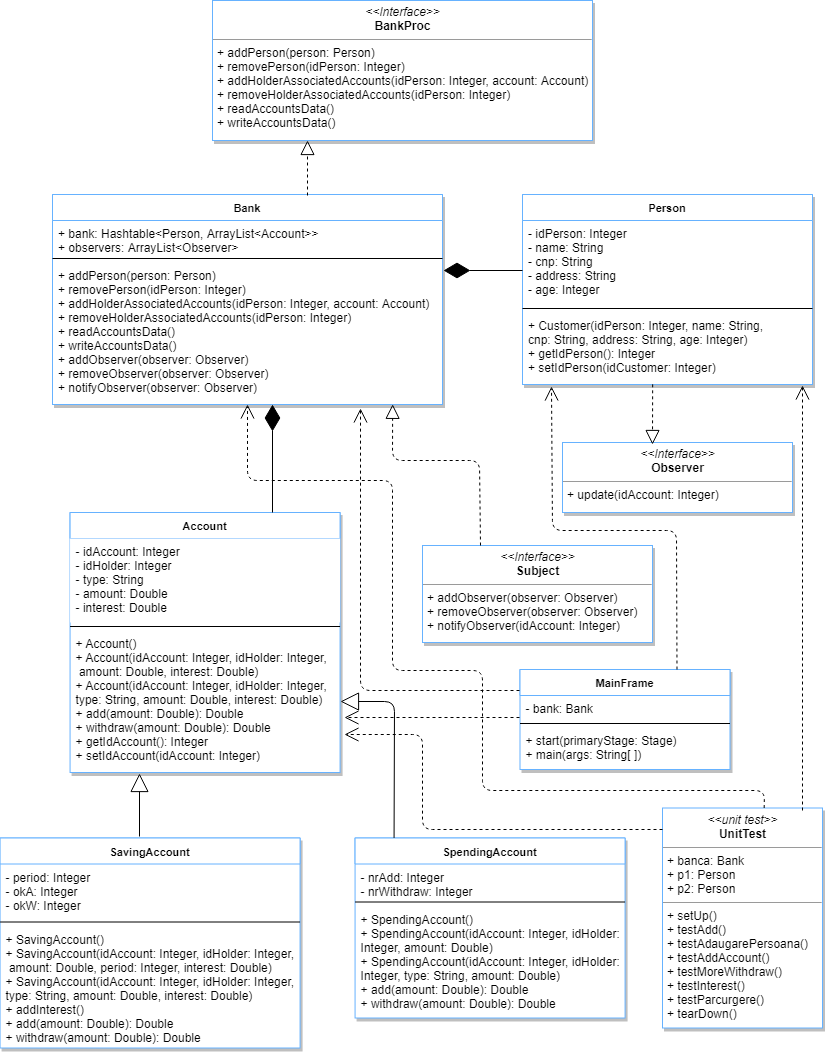
Asumptiile stabilesc cadrul in care se dezvolta aplicatia. In cadrul proiectului s-a stabilit urmatoarea asumptie: presupunem ca datele de intrare sunt introduse in mod corect, conform specificatiilor anterioare.

1. Proiectare
   1. Decizii de proiectare

Pornind de la specificatia proiectului, am luat decizia descompunerii problemei in mai multe clase: clasa MainFrame ocupandu-se de lansarea aplicatiei si de implementarea interfetei grafice, clasele Person, Account, SavingAccount si SpendingAccount reprezentand clasele reprezentative fiecarui tabel, clasa Bank, care implementeaza interfata BankProc si care se ocupa de principalele operatii dintr-o banca, clasa UnitTest pentru testarea sistemului si interfetele Observer si Subject pentru utilizarea pattern-ului Observer.

* 1. Diagrama UML

Urmatoarea diagrama UML descrie clasele si obiectele sistemului si relatiile dintre acestea:



* 1. Structuri de date

Pentru definirea campurilor din tabele se utilizeaza o liste obiecte. Acest lucru este realizabil utilizand colectii, implicit clasa FXCollections care implementeaza toate metodele din interfata ObservableList<T>. De asemenea mai sunt necesare utilizarea unei Hashtable care stocheaza date bancii, a unui ArrayList care reprezinta lista de conturi pentru fiecare utilizator in parte si un List care stocheaza toti observerii pe care mai apoi ii notifica in cazul in care se efectueaza o operatie pe contul respectiv.

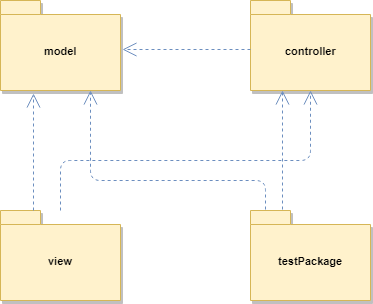
* 1. Proiectare clase

Sistemul de gestionare a comenzilor contine 7 clase:

1. Clasa **Person** – se foloseste pentru a stoca toti clientii. Aceasta implementeaza java.io.Serializable si contine un constructor, iar pentru fiecare atribut al clasei, cate un getter si un setter. De asemenea mai contine si metoda hashCode, care este suprascrisa astfel incat maparea sa se realizeze dupa numele clientilor
2. Clasa **Account** – este o clasa abstracta, care implementeaza java.io.Serializable. Se foloseste pentru a stoca toate conturile bancii. Clasa Account contine mai multi constructori si, de asemenea, cate un getter si un setter pentru atributele clasei.
3. Clasa **SavingAccount** – extinde clasa Account. Aceasta stocheaza toate conturile de tip „cont de economii”, deschise la banca respectiva. Clasa contine un constructor, iar de asemenea sunt implementate metoda de retragere, respectiv depunere numerar, prin adaugarea unei dobanzi la adaugarea unei sume in cont pe o anumita perioada de timp.
4. Clasa **SpendingAccount** – extinde clasa Account. De asemenea memoreaza conturile de tip cont de cheltuieli. Clasa contine un constructor, iar de asemenea sunt implementate metoda de retragere, respectiv depunere numerar.
5. Clasa **Bank** – clasa utilizata pentru administrarea bancii si a tuturor operatiilor necesare pentru buna functionare a unei banci.
6. Clasa **UnitTest** – clasa utilizata pentru testarea sistemului de gestiune a unei banci
7. Clasa **MainFrame** – clasa care sta la baza interfetei grafice si care practic construieste un mod mai usor de interactionare a utilizatorului cu sistemul.
   1. Pachete

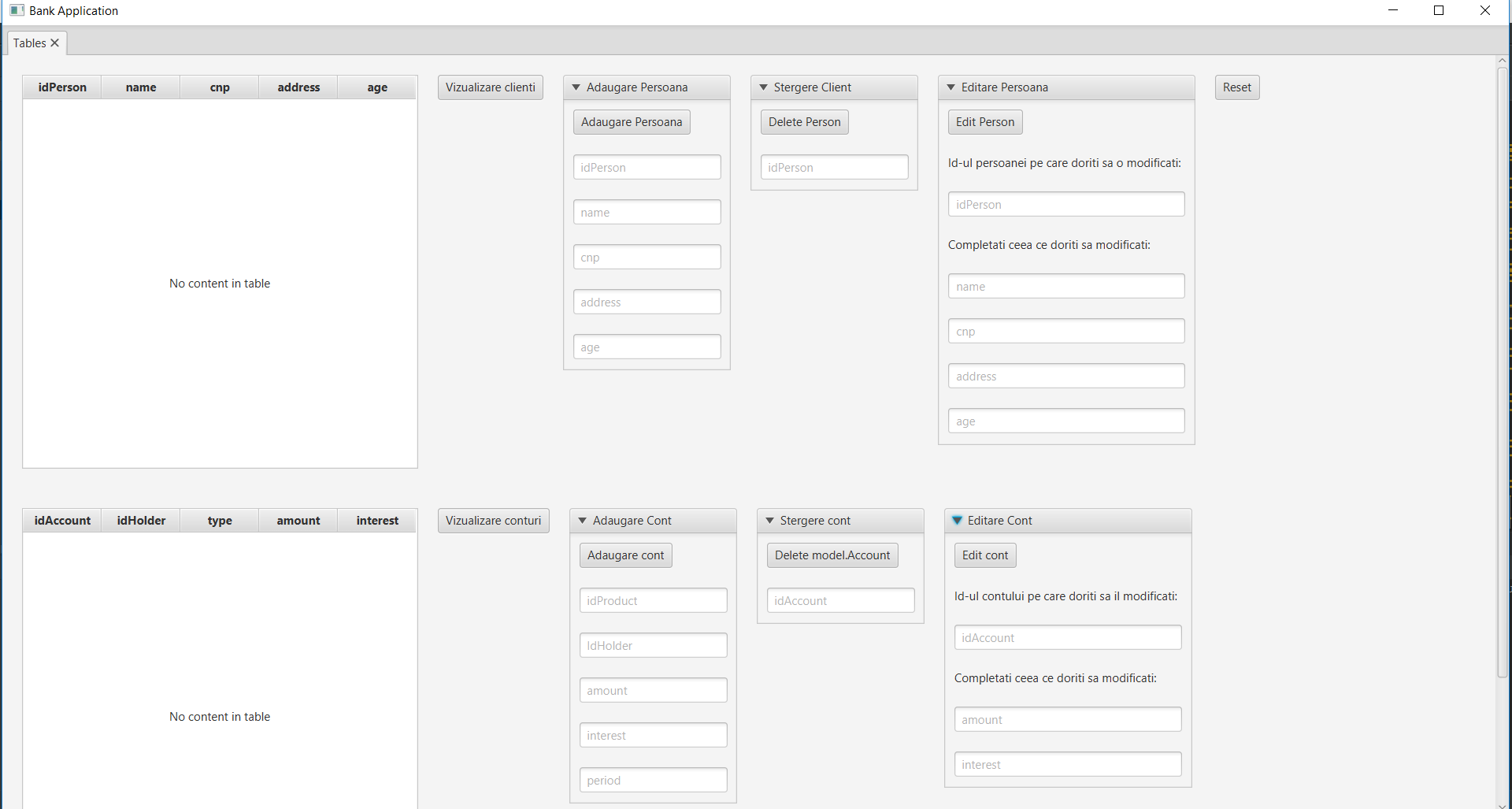
Diagramele de pachet ne ajuta la descompunerea sistemelor mari in subsisteme, lucru necesar pentru a putea realiza o implementare sistematica. Sistemul de gestionare a comenzilor este impartit in 4 pachete:

1. Pachetul model – reprezinta modelele de date ale aplicatiei, mai exact clasele Person, Account, SavingAccount si SpendingAccount
2. Pachetul controller – contine logica aplicatiei, mai exact clasa care administreaza operatiile disponibile intr-o banca, clasa Bank.
3. Pachetul view – contine interfata grafica a proiectului, mai exact clasa MainFrame
4. Pachetul testPackage – contine clasa care prin care se realizeaza testarea intregului sistem, mai exact clasa UnitTest



* 1. Interfata utilizator

Interfata grafica reprezinta un mod mai usor de interactionare dintre utilizator si program. Fara aceasta, utilizarea aplicatiei de gestionare a unei banci devine mai greoaie si destul de dificila pentru un utilizator nespecializat. Interfata este realizata cu ajutorul clasei MainFrame care extinde clasa Application.



Interfata este compusa dintr-o fereastra cu titlul „Bank Application” care este impartita in 3 4 zone, astfel:

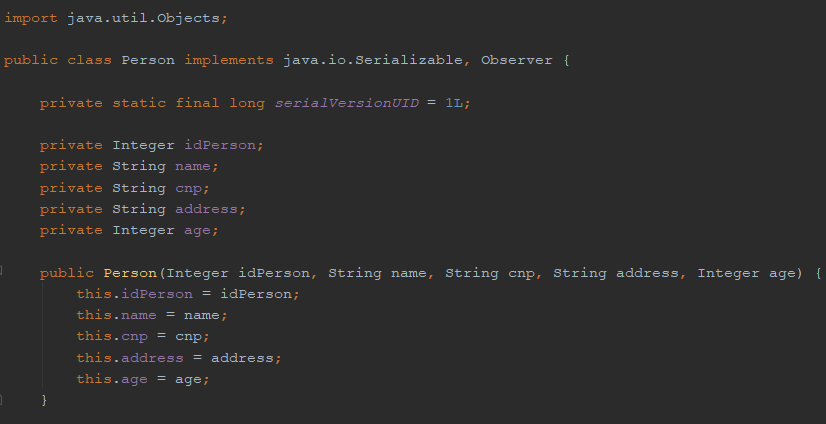
1. Zona 1 – zona rezervata pentru tabelul Person, care contine butoanele si textField-urile necesare operatiilor specificate
2. Zona 2 – zona utilizata pentru vizualizarea si producerea operatiilor asupra tabelului Account, care contine butoanele si textField-urile necesare
3. Zona 3 – zona in care se pot realiza noi retrageri si noi depuneri, pentru un anumit cont
   1. Modul de tratare a erorilor

Ca in orice sistem, aparitia unor erori este foarte posibila si trebuie luata in calcul. In cazul de fata, o eroare destul de posibila este incercarea de a retrage o suma mai mare decat suma pe care titularul o are disponibila in cont. In acest caz, operatia de retragere nu va produce niciun efect asupra contului.

1. Implementare

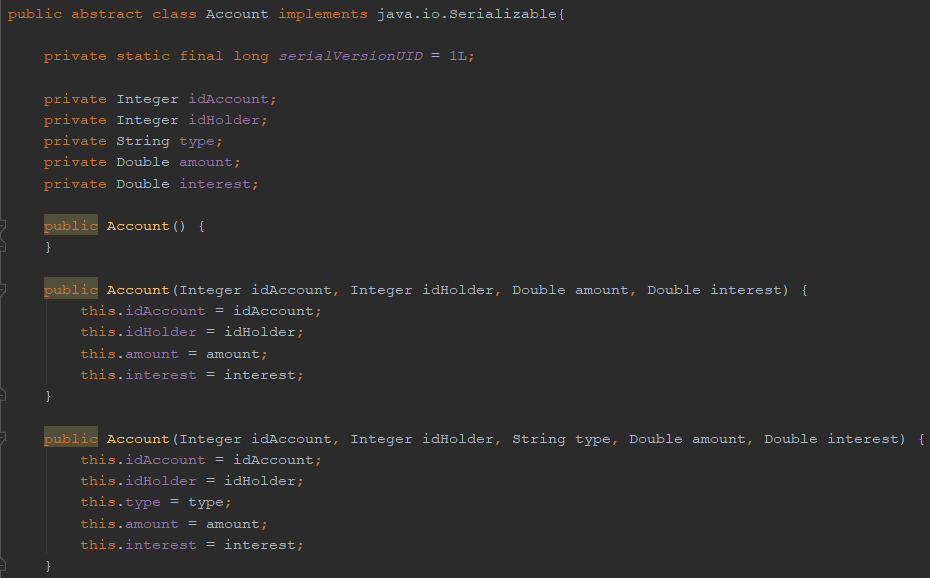
Dupa stabilirea obiectivului principal si a obiectivelor secundare si dupa analiza, modelarea si proiectarea proiectului urmeaza implementarea lui.

1. Clasa **Person**

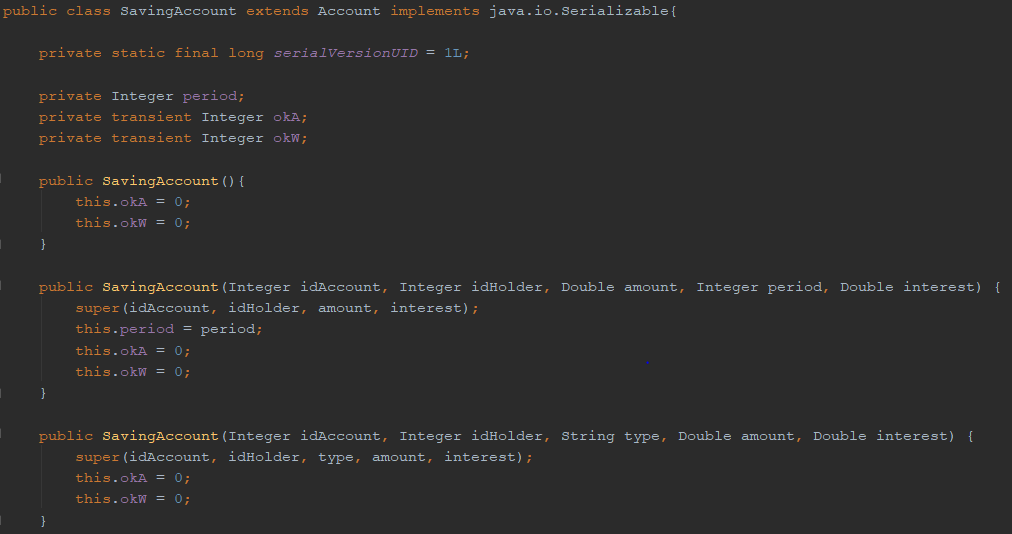


Clasa Person, are ca si atribute id-ul persoanei, numele, cnp-ul, adresa si varsta clientului. Aceasta contine un constructor, iar pentru fiecare atribut al clasei, cate un getter si un setter, metode necesare pentru procesul de incapsulare.

1. Clasa **Account**

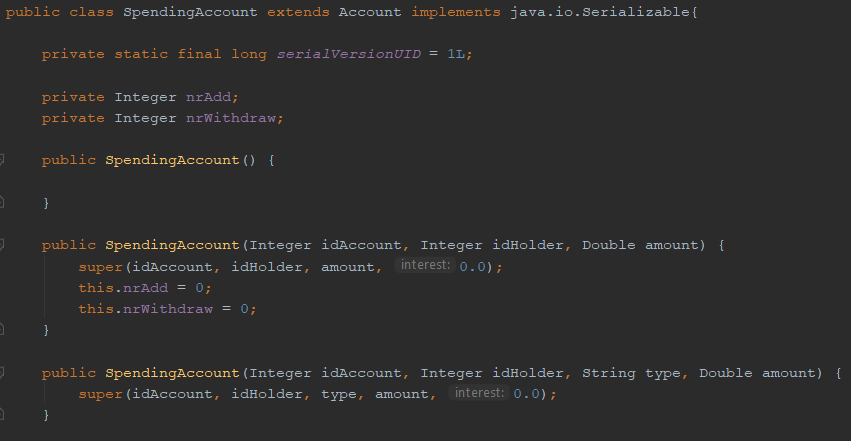


Clasa Account este o clasa abstracta. Aceasta are ca si atribute id-u contului, id-ul titularului, tipul contului, suma si dobanda acordata (in cazul in care este cont de cheltuieli dobanda va fi 0). Aceasta contine un constructor, iar pentru fiecare atribut al clasei, cate un getter si un setter. De asemenea mai contine doua metode abstracte de depunere si retragere.

1. Clasa **SavingAccount**

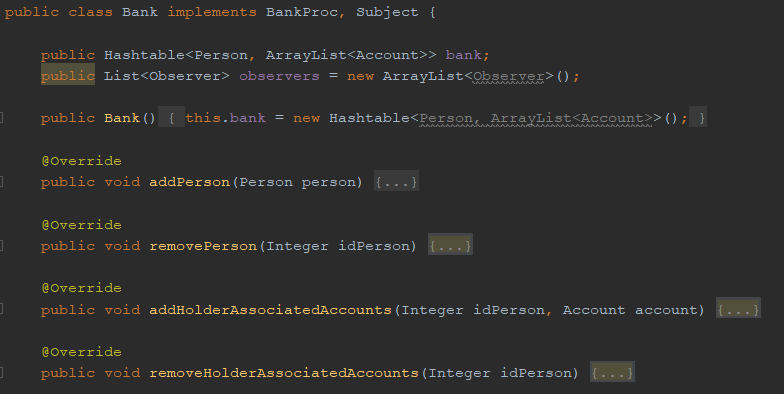
Clasa SavingAccount extinde clasa Account si implementeaza metodele de depunere si retragere a unei sume din cont, si de asemenea metoda de acordare a dobanzii.

1. Clasa **SpendingAccount**



Clasa SpendingAccount extinde clasa Account si implementeaza metodele de depunere si retragere a unei sume din cont.

1. Clasa **Bank**



Clasa Bank implementeaza interfata BankProc si are ca argumente un HashTable si un ArrayList de Observers. De asemenea contine implementarea metodelor din interfata BankProc si Subject.

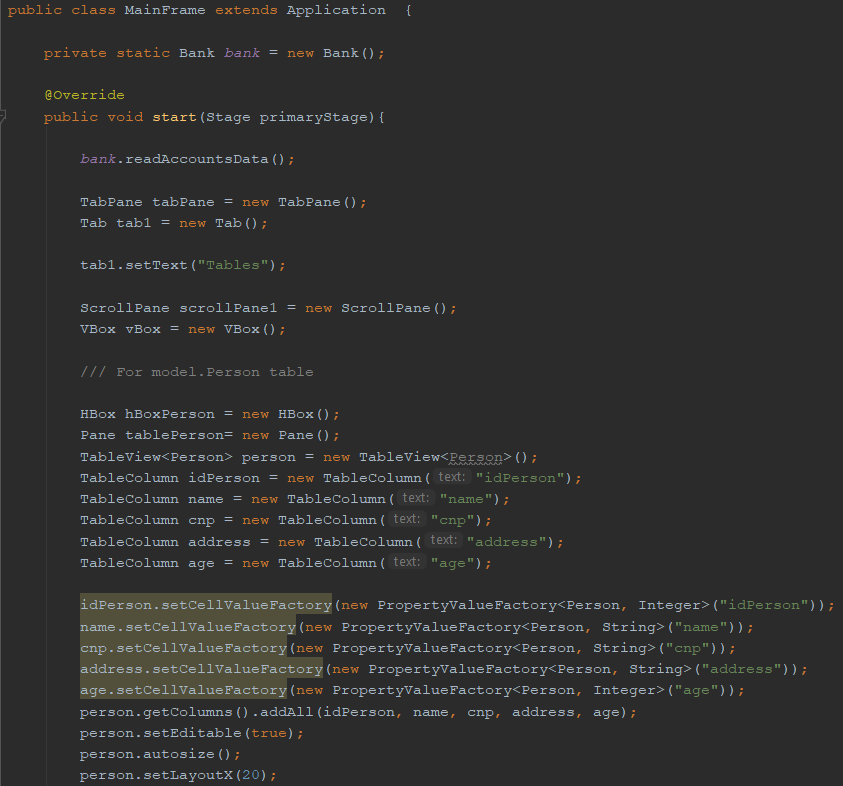
1. Clasa **UnitTest**





Pentru partea de testare s-a realizat o clasa separat, si anume clasa UnitTest, in interiorul careia fiecare test are pregatit ca punct de plecare in momentul rularii o valoare prestabilita. Acest lucru s-a realizat cu ajutorul metodelor setUp(), metoda ce este apelata automat inainte de fiecare test, si tearDown(), metoda ce este apelata automat dupa fiecare test. In afara celor 2 metode s-au mai creat 6 metode de test, care verifica diferite lucruri. De exemplu daca adaugarea unei persoane s-a facut in mod corect.

1. Clasa **MainFrame**



Clasa MainFrame realizeaza practic toata interfata grafica. Aceasta clasa extinde clasa Application si contine metoda main, in interiorul careia se utilizeaza metoda launch. Dupa care urmeaza suprascrierea metodei start care creeaza interfata grafica. Mai apoi, se suprascrie metoda handle pentru fiecare buton in parte.

1. Testare

Partea de testare este analizata mai amanuntit in capitolul anterior la implementare. Pentru fiecare test se considera urmatoarele date de intrare:

p1 = new Person(1, „Marin”, „1910508257841”, „Str.Campina”, 27)

p2 = new Person(2, „Carmen”, „2950204284125”, „Str. Bucegi”, 23)

a1 = new SavingAccount(1, 1, 2500.0, 24, 0.015)

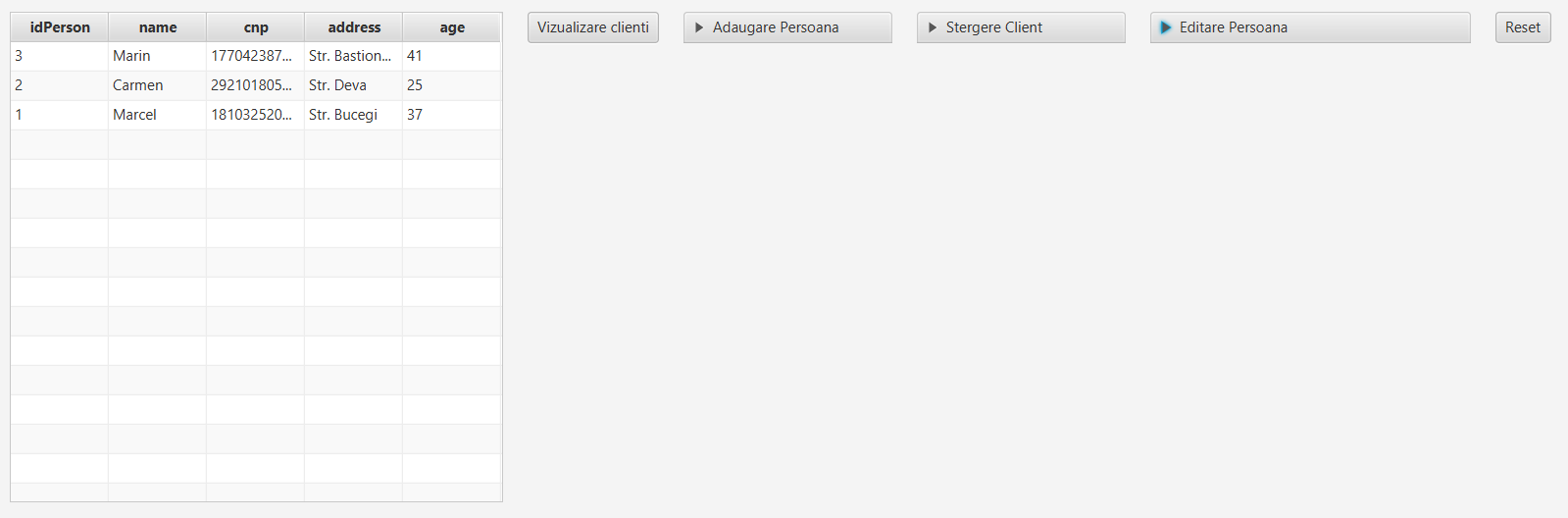
a2 = new SavingAccount(2, 1, 3400.0, 10, 0.02)

a3 = new SpendingAccount(3, 2, 2500.0)

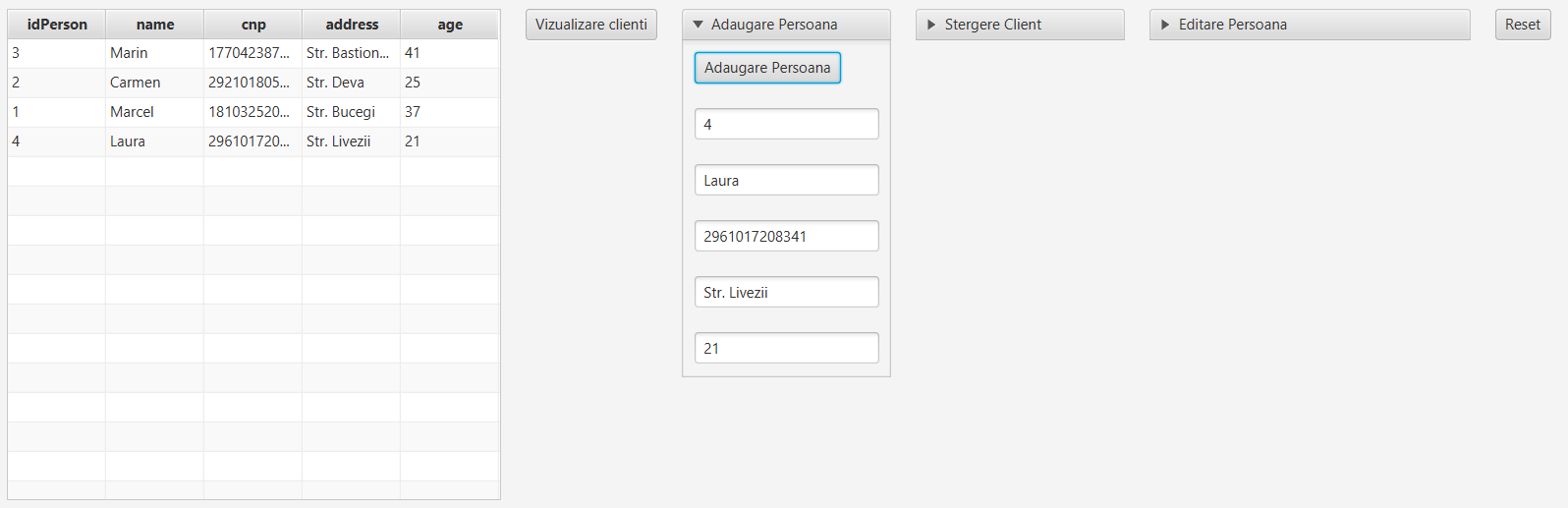
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ce se testeaza | Date de intrare pentru test | Rezultat asteptat | Rezultat efectiv obtinut | Pass/fail |
| Adaugare  Persoana | p3 = new Person(3, „Gica”, „1791012872930”, „Str. Viorilor”, 38) | Gica | Gica | Pass |
| Parcurgere | Conturile persoanei p1 | 2500.0 | 2500.0 | Pass |
| Acordarea Dobanzii | a4 = new SavingAccount(4, 1, 3400.0, 2, 0.02) | 3537.36 | 3537.36 | Pass |
| Posibilitatea de a efectua mai multe retrageri de numerar dintr-un cont de economii | Contul a1  withdraw(100.0)  withdraw(100.0) | 2400.0 | 2400.0 | Pass |
| Depunere Numerar | Contul a3  add(100.0)  add(150.0) | 2750.0 | 2750.0 | Pass |
| Adaugare Cont | a4 = new SpendingAccount(5, 2, 4870.2) | 5 | 5 | Pass |

1. Rezultate

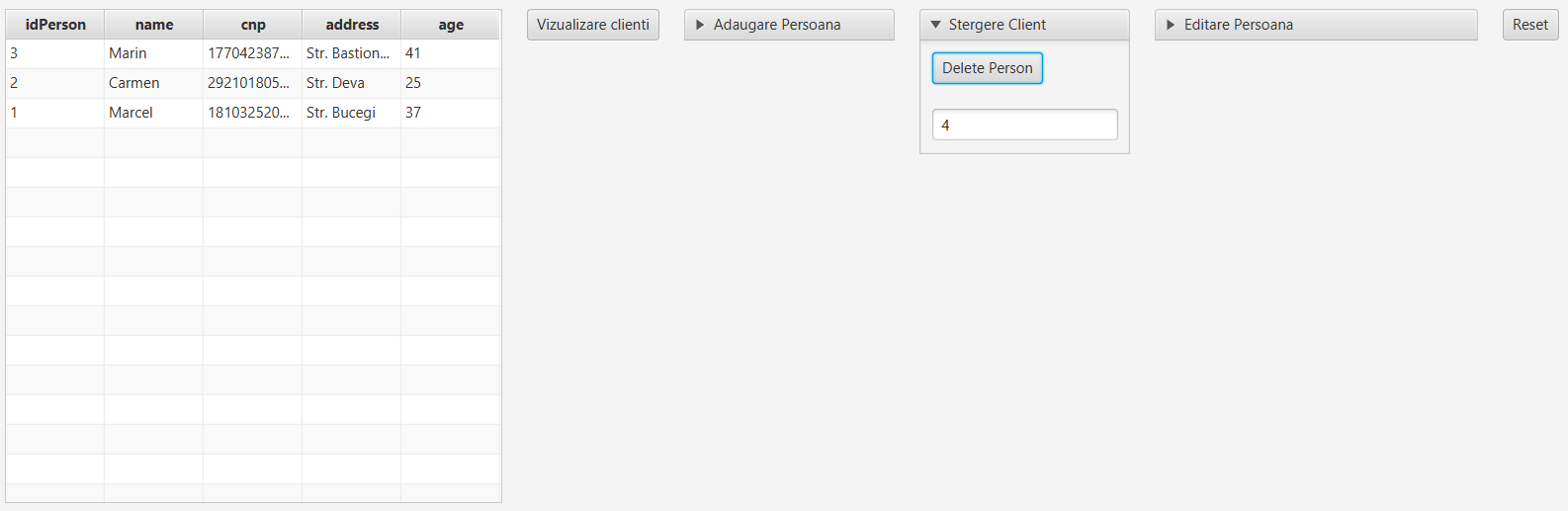
Dupa introducerea datelor dorite se pot apasa butoanele corespunzatoare operatiei alese sau pe butonul de vizualizare date, in cazul in care se doreste doar vizualizarea tabelelor.



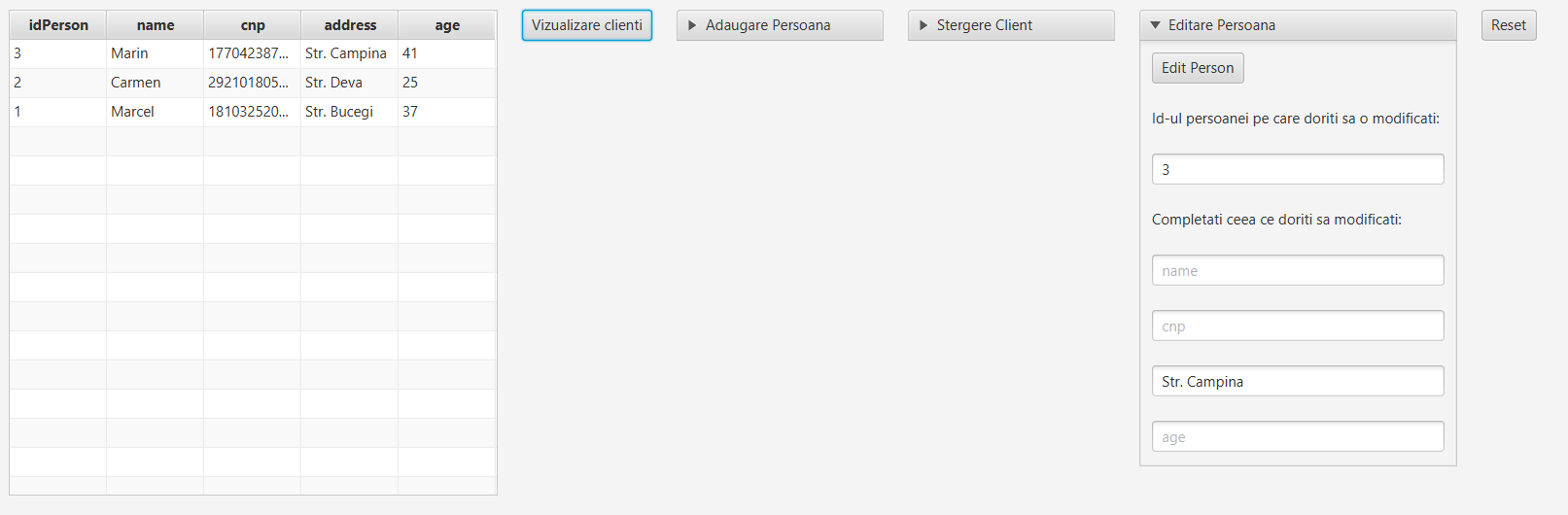
* Adaugare



* Stergere



* Editare



1. Concluzii

O concluzie foarte importanta, a acestui proiect, consta in importanta descompunerii problemei in subprobleme si usurinta cu care o problema poate fi rezolvata in acest mod. De asemenea este foarte importanta realizarea unei analize amanuntite si formarea unei idei generale asupra problemei inaintea de a incepe implementarea ei.

Din acest proiect am invatat lucruri precum:

* Imbogatirea cunostintelor in materie de interfata grafica
* Utilizarea colectiilor intr-un cadru mai amanuntit
* Importanta utilizarii proceselor de modularizare, abstractizare si incapsulare
* Complexitatea realizarii unui proiect de la 0 care, pe langa partea de rezolvarea a problemei, necesita de asemenea o interfata grafica, o unitate de testare si o documentatie laborioasa a intregului proiect
* Scrierea intr-un fisier prin serializare, respectiv citirea datelor dintr-un fisier prin deserializare
* Utilizarea pattern-ului Observer
* La ce se refera termenul de „Design by Contract”

Sistemul de gestionare a unei banci are o gama larga de posibilitati de dezvoltare ulterioara, dintre care amintim:

* Crearea unei baze de date mai complexe
* Dezvolarea sistemului la o scara mai larga
* Posibilitatea de a putea efectua plati/tranzactii
* Adaugarea unui sistem de filtrare a clientilor
* Dezvoltarea interfetei grafice

1. Bibliografie
2. <http://www.coned.utcluj.ro/~salomie/PT_Lic/>
3. <http://users.utcluj.ro/~igiosan/teaching_poo.html>
4. <https://www.draw.io/>
5. <http://www.tutorialspoint.com/java/java_serialization.htm>
6. <http://www.coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW4_Tema4/HW4_Tema4_Hashing.pdf>
7. <https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>