DOCUMENTATIE
TEMA 4

Gestionarea conturilor bancare

Nume: **Stefan**
Prenume: **Florina Diana**
Grupa: **30227**

Cuprins

1. Cerinte Functionale 3
2. Obiective 3
2.1. Obiectiv Principal: 3
2.2. Obective Secundare: 3
3. Analiza Problemei 4
4. Proiectare 5
4.1. Structuri de date 5
4.2. Diagrama de clase 5
4.3. Algoritmi 7
5. Implementare 7
6. Testare 9
7. Concluzii si Dezvoltari Ulterioare 10
8. Bibliografie 11

1. Cerinte Functionale

Proiectarea și implementarea unui sistem de gestionare a datelor digitale ale unei banci, care includ clienti, conturi, si metode de accesare si gestionare a lor, sistemul având un comportament specific situațiilor din lumea reala.

2. Obiective

2.1. Obiectiv Principal:

Obiectivul principal al acestei teme pune accent pe realizarea aplicatiei prin tehnica Design By Contract care se refera la adaugarea unor pre, post si invariant conditii in interfata, cat si adaugarea unor asertiuni corespunzatoare acestor conditii in clasa care implementeaza interfata respective. De asemenea, aceasta tehnica necesita si adaugarea unei metode care sa reprezinte invariantul clasei, metoda ce asigura structura variabilelor de instanta ale clasei respective. Pe langa aceasta tehnica, se recomanda si folosirea unui Design Pattern Observer, folosit pentru o mai buna comunicare intre clasele care implementeaza acest sablon. De asemenea se pune accentsi pe implementarea propriu-zisă a conceptelor într-un limbaj de programare orientat pe obiecte. În cadrul limbajului, se vor studia interacțiunile dintre obiecte, interacțiunea cu utilizatorul (prin interfața grafică), cât și posibilitățile de gestionare a datelor ce contin datele relevante pentru problema propusa, iar pentru gestionarea bazei de date, se vor folosi elemente ale limbajului Java, alaturi de tehnica de serializare, care va permite persistenta datelor inclusiv dupa inchiderea aplicatiei.

2.2. Obective Secundare:

Obiectiv Secundar	Descriere	apitol
Alegerea structurilor de date	Se va descrie ce structuri de date au fost folosite pentru implementare.	4
Impartirea pe clase	Se va descrie clasele care sunt folosite.	4
Dezvoltarea algoritmilor	Se vor descrie algoritmii folositi pentru diverse metode.	4
Implementarea solutiei	Se va descrie pas cu pas solutia gasita pentru rezolvarea acestei probleme.	5
Testare	Cateva imagini care descriu cum a fost realizata testarea, precum si teste JUnit.	6

3. Analiza Problemei

La o analiză sumară a cerinței temei curente, complexitatea ei pare una redusă, implicând o simulare a unei banci virtuale, in care datele sunt stocate in tabele de dispersie cu adresare deschisa. Totuși, la o analiză mai în detaliu, se poate deduce că este necesară o proiectare riguroasă, corectă, eficientă, cât și ușor de utilizat, când se face referire la interacțiunea utilizatorului cu programul rezultat.

Prima problemă care apare este reprezentarea datelor in calculator. Cerința presupune folosirea unor entități precum Persoana, Cont si Banca, in care sa se stocheze datele corespunzatoare entitatilor mai sus mentionate. In acest sens, se alege o implementare bazata strict pe limbajul Java, in care datele sa fie stocate cu ajutorul colectiilor predefinite. Se pune problema cum poate fi gestionata mai bine pastrarea acestor date astfel incat acestea sa nu se piarda, datele unei persoane sa fie strict confidentiale. De asemenea, aceste date trebuie sa fie disponibile doar unui personal autorizat si persoana in cauza, sau despre care este vorba, adica titularul unui cont. De asemenea, se pune problema la ce va avea acces titularul atunci cand va dori sa efectueze anumite actiuni bancare, adica cum va fi definita si prelucrata interfata utilizatorului astfel inca in acelasi timp, sa permita titularului sa isi acceseze datele si acestea sa fie ascunse de restul utilizatorului. Aceasta problema poate fi rezolvata prin crearea unei aplicatii de gestiune a datelor bancare care ar facilita mult mai mult manipularea acestora, cat si prelucrarea lor in timp real.

O a doua problemă a curentei cerințe este utilizarea unor conditii suplimentare in cadrul metodelor implicite care sunt aplicate asupra bazei de date interne a aplicatiei. Astfel, este imperios necesara utilizarea corecta, completa si coerenta a conceptelor pe care se bazeaza metoda de Design by Contract.

De asemenea, pentru cerinta curenta, se doreste ca datele interne ale aplicatiei sa isi pastreze consistenta chiar si dupa ce aplicatia este inchisa. Acest lucru este pus la dispozitie prin tehnica de serializare a datelor, in care obiecte cu toate atributele aferente sunt salvate in intregime pe disc, putand fi ulterior recuperate in aceeasi stare in care au fost salvate.

4. Proiectare

4.1. Structuri de date

Ca si structuri de date, au fost utilizate structuri de tip HashMap, cu rol de pastrare a unor anumite valori la anumite chei. Aceasta structura de date este utilizata in cadrul clasei Bank cu rol de a potrivi persoanele cu conturile pe care le detin la o banca, unde cheia este reprezentata de catre persoana si valoarea de catre o lista de conturi.

4.2. Diagrame UML

O etapa importanta in procesul de proiectare al unei aplicații este realizarea diagramelor UML (Unified Modelling Language) care permit specificarea structurii proiectului, a variabilelor, metodelor, cat si a interacțiunii dintre clase.

O diagrama a cazurilor de utilizare (use case diagram) prezinta o colecție de cazuri de utilizare si actori care:

• oferă o descriere generala a modului in care va fi utilizat sistemul
• furnizează o privire de ansamblu a funcționalităților ce se doresc a fi oferite de sistem
• arata cum interacționează sistemului cu unul sau mai mulți actori
• asigura faptul ca sistemul va produce ceea ce s-a dorit

Diagrama este exemplificata in schema următoare:

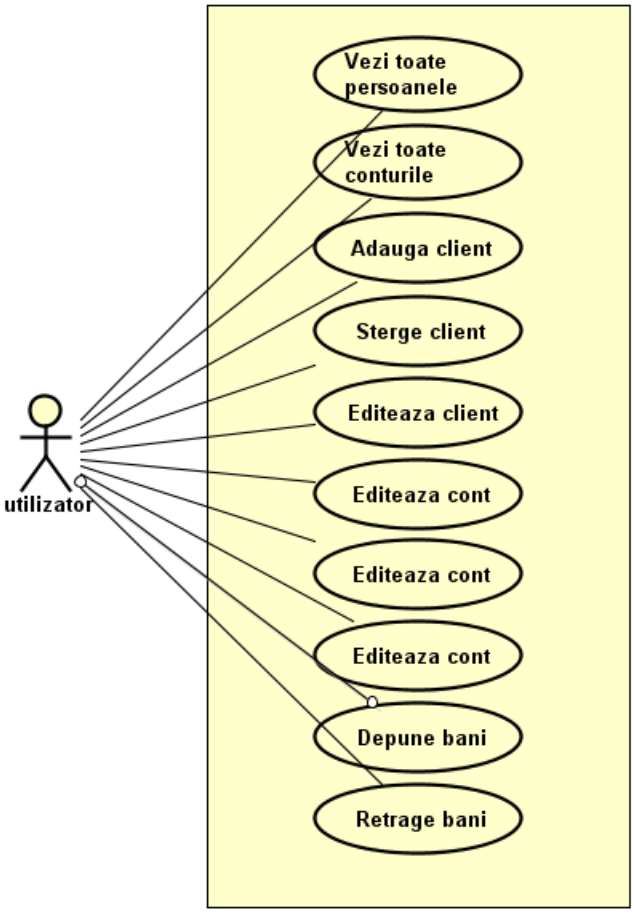
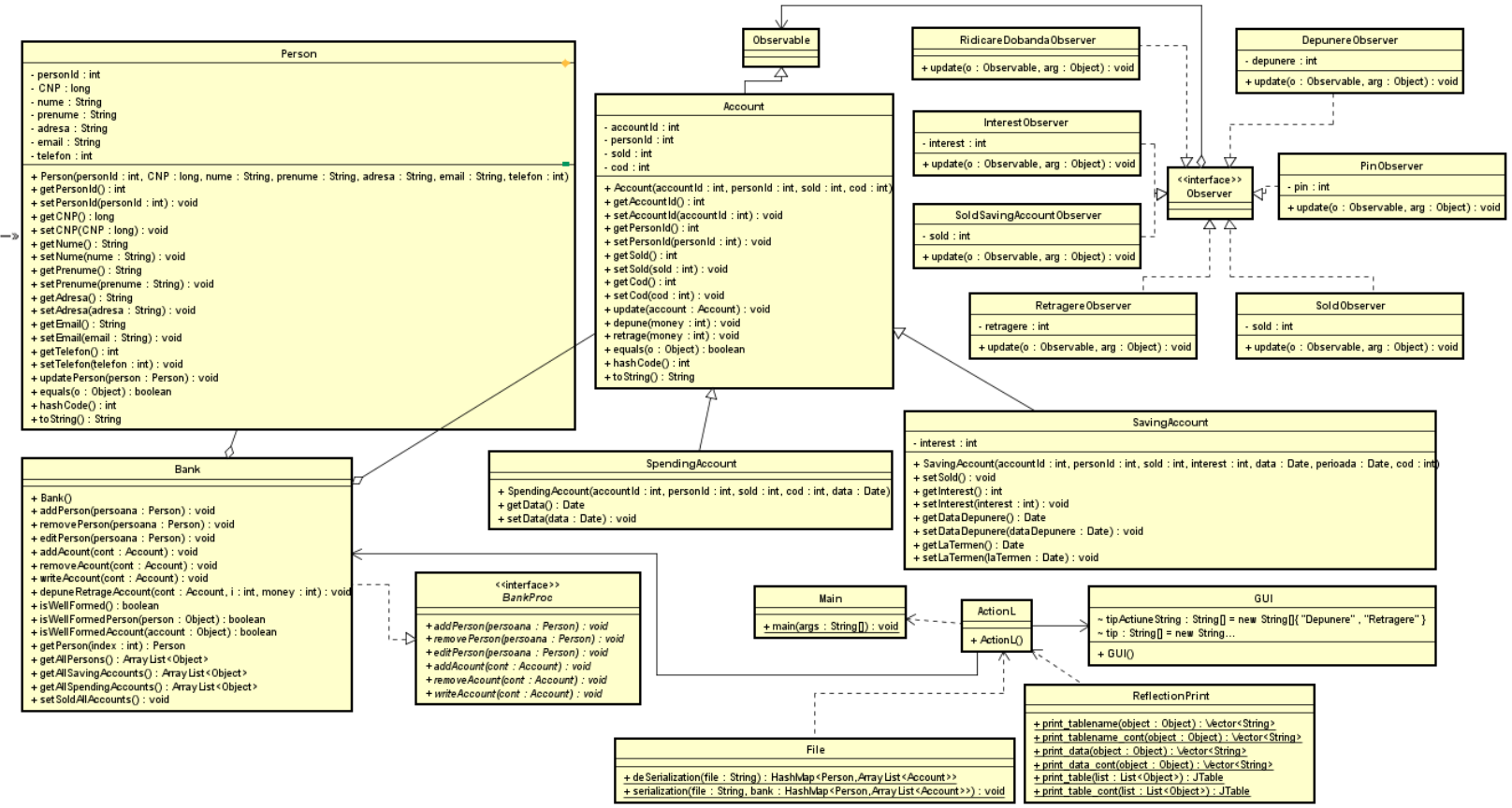


Diagrama de clase

Diagrama de clasa este utilizata in descrierea structurii statice, adică a entităților sau claselor existente intr-un sistem. Diagrama de clase reflecta diferite legături intre entitățile domeniului de obiecte si descriu structura interna si vizibilitatea lor (publice, private, protejate) precum si tipurile de relații. Diagrama de clasa este reprezentata prin următoarea schema:



4.3. Algoritmi

Operațiile sunt implementate prin intermediul unor algoritmi corecți si eficienți (ca timp de execuție si spațiu de memorie).

5. Implementare

Pentru implementarea acestei aplicatii au fost utilizate 17 clase, dintre care 5 mai importante: Bank, Account, Person, SpendingAccount, SavingAccount si o interfata: BankProc.

Interfata **BankProc** defineste semnatura urmatoarelor metode care urmeaza sa fie implementate in clasa Bank: **addPerson** (Person person), **removePerson** (Person person), **editPerson** (Person person), **addAccount** (Account cont), **removeAccount** (Account account), **writeAccount** (Account write).

Clasa **Bank** reprezinta nucleul central al intregii aplicatii, ea fiind cea care reuneste celelalte clase, formand un tot unitar. In aceasta clasa sunt stocate asocierile intre persoane si conturile care le apartin, acest lucru fiind implementat folosind o tabela de dispersie cu adresare deschisa, care permite ca o anumita persoana sa poata avea multiple conturi.

Clasa **Bank** detine ca si variabila instanta o structura de tip HashMap care are ca si cheie o variabila de tip Persoana si ca valoare o variabila de tip ArrayList de tipul Account. Aceasta variabila e folosita pentru a pastra conturile aferente unei personae titulare, care are deschise conturi la banca respective. Clasa Bank, implementeaza metodele din interfata BankProc si anume: addPerson(Person person) – adauga o noua cheie in variabila de tip HashMap, aceasta cheie fiind cnp-ul persoanei respective, acest lucru fiind posibil prin faptul ca metoda de hashcode a fost suprascrisa in clasa Person, removePerson (Person person) – metoda ce sterge o cheie din tabela de hash, stergand o data cu acea persoana toate conturile aferente ei, editPerson (Person person) – metoda ce modifica o cheie din tabela de hash, **addAccount** (Account cont) – metoda ce adauga un nou cont in variabila instanta bank, acest cont fiind adaugat persoanei carei ii este asociat contul respectiv, removeAccount (Account cont) – metoda ce sterge un cont al unei persoane, writeAccount (Account account) – metoda ce modifica datele unui anumit cont.

In cadrul metodelor din clasa Bank, se preteaza validarea datelor de intrare, cat si verificarea rezultatelor si a efectelor realizate de metode. In acest sens, se folosesc tehnici din gama Design by Contract, care se bazeaza pe atentionarea utilizatorului de momentele in care datele introduse sunt eronate, sau partial gresite, dar ofera si o certificare a faptului ca operatiunile s-au incheiat cu succes, afisand catre utilizator, in acest caz, un mesaj relevant. Metoda isWellFormed() reprezinta invariantul clasei si este apelata la inceputul sau sfarsitul metodei pentru a ne asigra ca structura este bine formata, acest lucru inseamna ca nu sunt chei nule la care se pot adauga conturi sau nu sunt chei care nu au asociata o anumita valoare.

Clasa Account detine ca si variabile instanta urmatoarele: accountId – id-ul unic asociat fiecarui cont, personId –id-ul persoanei careia ii este asociat contul respective, sold- care detine suma curenta a contului, cod- care reprezinta pin-ul contului. in aceasta clasa se definesc si se implementeaza urmatoarele metode: retrage (int money) – metoda ce extrage dintr-un cont o anumita suma de bani, aceasta metoda verifica intai daca in contul respective mai sunt sau nu bani, precum si daca suma care se doreste sa fie retrasa este pozitiva si depune (int money) – metoda care depoziteaza o anumita suma intr-un cont, verificand daca suma care se doreste sa fie depozitata este pozitiva. Aceste operatii se fac cu validarea

initiala a datelor, pentru a nu ajunge in situatii in care soldul sa devina negativ, sau sa se faca o depunere sau retragere de numerar cu o suma negativa de bani.

Clasa Account asigura baza pentru constructia unor clase mai specializate, si anume contul de economii si contul curent. Astfel, asupra contului de economii se impun conditii ca soldul sa poata fi extras doar dupa o perioada indelungata de timp, si la extragere sa se acorde o dobanda, dar in schimb, contul curent nu are astfel de restrictii, el putand fi accesat de un numar de ori care sa nu fie constrans de vreo limitare.

Clasa SpendingAccount mosteneste clasa Account si pe langa aceasta defineste ca si varibila instanta data, care ne arata data ultimei accesari a contului.

Clasa SavingAccount mosteneste clasa Account si pe langa aceasta mai defineste ca si variabile instanta interest – dobanda si dataDepunere - data in care suma a fost depusa, laTermen - data la care se doreste ca suma sa fie retrasa. De asemenea, aceasta clasa mai defineste o metoda si anume setSold(), metoda ce adauga sumei dobanda calculate pe o anumita perioada, formula matematica ce a fost folosita fiind SumaFinala = SumaInitiala + SumaInitiala * dobanda * numarulDeZile / 360.

Clasa Person reuneste toate datele necesare identificarii clientilor pe care banca ii are, sau ii poate avea pe viitor. Aici pot fi stocate multiple date de identificare si de comunicare inspre si dinspre client, posiblitatea asignarii mai multor conturi pentru un singur client, si validarea accesului la anumite conturi doar in contextul in care se prezinta date de autentificare si pentru persoana titulara a contului.

De asemenea, persoanele stocate in baza de date interna, vor fi notificate cand la conturile lor intervine vreo modificare, lucru realizat cu sablonul de proiectare Observer.

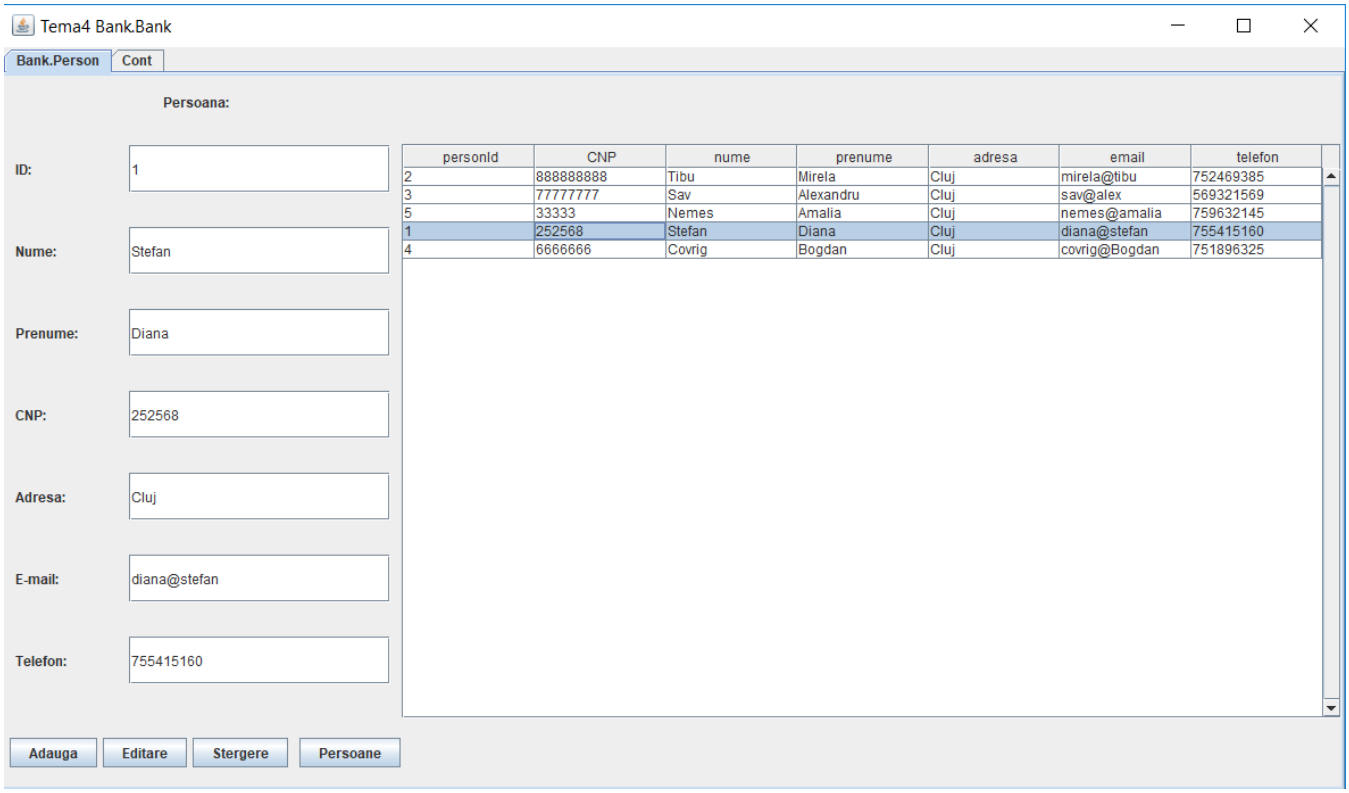
Se poate observa si faptul ca fiecare persoana are nevoie de o modalitate unica de a fi identificata, iar in contextul in care in implementare se folosesc tabelele de dispersie, fiecare persoana va avea o valoare de hash unica fata de oricare alta persoana.

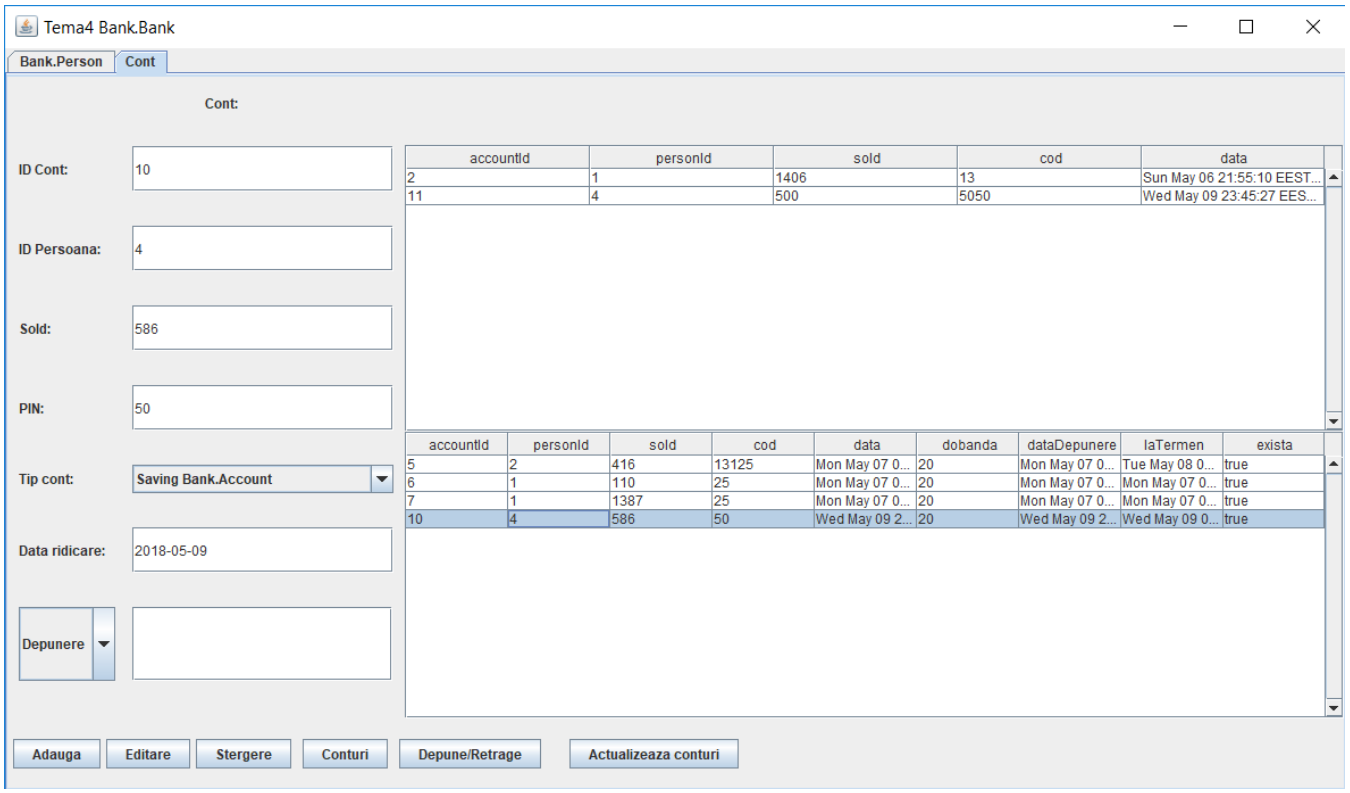
Clasa Person defineste urmatoarele variabile instanta: personId – identificatorul persoanei aferente unei banci, CNP - identificatorul unic al persoanei aferente unei banci, nume – de tip String, numele persoanei respective, prenume - de tip String, prenumele persoanei respective, adresa - adresa persoanei respective, email – de tip String, emailul persoanei respective si telefon –numarul de telefon al persoanei respective. De asemenea, in aceasta clasa a fost suprascrisa metoda hashCode pentru ca acea cheie generate de HashMap sa reprezinte cnp-ul persoanei respective.

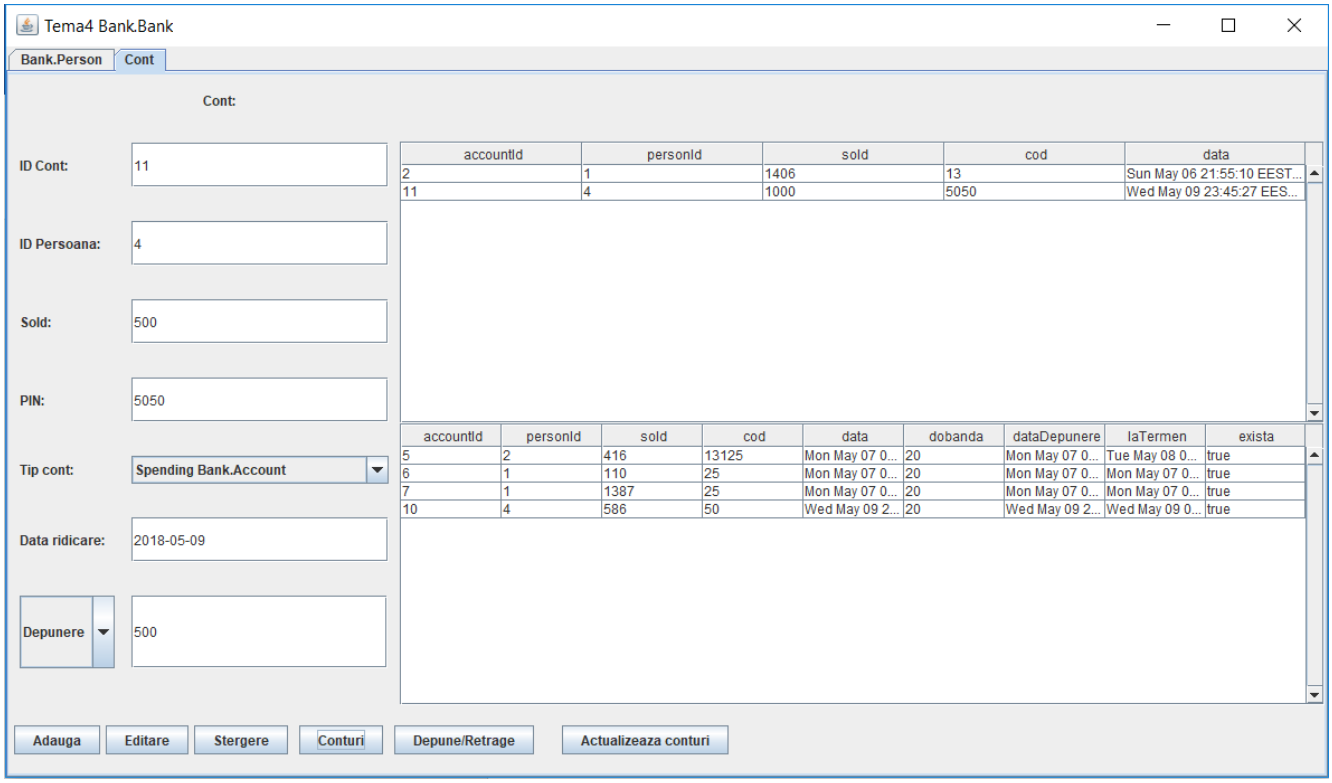
Nivelul de prezentare acopera interacțiunea programului cu utilizatorul, care se face prin intermediul unei interfețe grafice, care folosește elementele puse la dispoziție de limbajul Java.

6. Testare

Pentru testarea acestui proiect a fost folosita o clasa suplimentara BankTest care implementeaza urmatoarele metode cu rolul de a demonstra calitatea programului prin rularea unor teste pe aceasta aplicatie : addPerson ( ) , removePerson ( ) , editPerson ( ) , addAccount ( ), removeAccount ( ) , writeAccount ( ).







7. Concluzii si Dezvoltari Ulterioare

In urma acestui proiect au fost dezvoltate abilitatile de proiectare prin utilizarea tehnicii de Design by Contract si sablonului de proiectare Observer. Utilizarea acestui sablon de proiectare m-a ajutat sa imi dau seama cum functioneaza o aplicatie care comunica cu un intreg sistem bancar si cum este anuntata modificarea unor anumite date. De asemenea, prin utilizarea pre si post conditiilor, am reusit sa vad si un alt mod de tratare a exceptiilor in limbajul de programare Java.

Ca si dezvoltari ulterioare putem aminti urmatoarele idei:

- Crearea mai multor clase, de exemplu sa fie inca o clasa cu personalul bancii.
- Crearea mai multor metode, de exemplu de generare in timp real a unor rapoarte referitoare la tranzactiile efectuate de catre client.
- Creearea unei interfete specifice unui utilizator si unui administrator al bancii
- Un cont de utilizator ar putea conferi clientilor o posibilitate de a stoca tranzactiile incheiate, soldul total, timpii de tranzactionare.
- De asemenea, interfața grafică poate fi îmbunătățită pentru a fi mai intuitivă și mai atractivă pentru orice tip de utilizator.

8. Bibliografie

Giosan Ion, POO curs 6 - Dezvoltarea aplicațiilor OO. Diagrame UML de clase și obiecte, UTCN, 2017

Giosan Ion, POO curs 9 - Interfete utilizator grafice (GUIs), UTCN, 2017

http://www.coned.utcluj.ro/~salomie/PT_Lic/

https://www.tutorialspoint.com/uml/index.htm

http://www.uml-diagrams.org/

https://stackoverflow.com/

http://www.tutorialspoint.com/java/java_serialization.htm

http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html

http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

http://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html

http://stackoverflow.com/questions/11415160/how-to-enable-the-java-keywordassert-in-eclipse-program-wise