DOCUMENTATIE TEMA 4

Aplicatie bancara

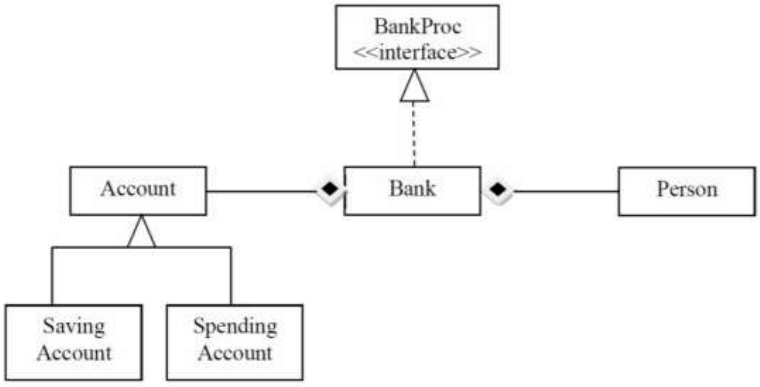
Student: Moldovan Teodora
Grupa: 30225
Facultatea de Automatica si Calculatoare
Profesor Laborator : Antal Marcel

Cuprins

1. Obiective

1.1. Obiectiv Principal

Aceasta tema presupune crearea unei aplicatii bancare pornind de la diagrama de clase:



Aplicatia trebuie sa indelineasca urmatoarele cerinte:

- Sa contina interfata BankProc cu urmatoarele operatii posibile: adaugare / stergere de persone, adaugare / stergere conturi associate unei personae, citire / scriere a datelor conturilor, generatoare de rapoarte. Pentru fiecare operatie din interfata se vor specifica pre si post conditiile.
- Sa contina clasele Person, Account, SavingAccount, SpendingAccount si alte clase care ajuta la realizarea aplicatiei.
- Se va defini un Design Pattern Observer care va instiinta detinatorul unui cont despre operatiile efectuate asupra contului.
- Clasa Bank va fi implementata astfel incat sa contina o colectie predefinita care foloseste tabele de dispersie (hashtables) . Cheia din hashtable va fi generate pe baza informatiilor personale ale titularului de cont.
- O persoana poate avea mai multe conturi. Pentru afisarea informatiilor legate de banca se vor folosi JTables. Pentru clasa Bank se va define o metoda “well formed” si va fi implementata folosind metoda Design by Contract (pre, post conditi, assertions)
- Se va implementa un driver pentru a putea testa proiectul (JUnit)
- Datele conturilor pentru popularea obiectului de tip Bank vor fi incarcate si salvate intr-un fisier.

1.2. Obective Secundare

Obiectiv Secundar	Descriere	Capitol
Analiza problemei si presupuneri	Se analizeaza modul de abordare a problemei propuse.	2
Alegerea structurilor de date	Prezentarea structurii de date alese si a argumentelor	3
Impartirea pe clase	Diagrama de clase si descriere a functionalitatii claselor	3,4
Implementarea solutiei	Detalii referitoare la implementare in urma analizei realizate	4
Testare	Testarea functionarii corecte a aplicatiei si modurile de realizare a acesteia	5

2. Analiza Problemei

In zilele noastre, pentru managementul economiei dar si managementul veniturilor personale, un rol foarte important il joaca bancile. Datorita acestui fapt aplicatia pe care o vom implementa trebuie sa simuleze in mod minimal activitatea unei banci si sa faciliteze operatorilor bancari gestiunea clientiilor si a conturilor. Angajatii bancii care vor utiliza aplicatia trebuie sa aiba posibilitatea sa adauge sau sa stearga clienti, sa adauge sau sa stearga conturi si sa efectueze operatiile de depunere sau retragere a unei sume de bani pentru un anumit client la solicitarea acestora.

Asadar sistemul trebuie sa usureze, impunand in acelasi timp un anumit nivel de Securitate al lucrului cu clientii si conturilor acestora si diminuand timpul efectiv de lucru asupra acestor operatii. Diminuarea timplui de lucru va fi realizata prin incercarea de a minimiza pe cat de mult posibil introducerea datelor in aplicatie (de exemplu pentru a selecta id-ul unui client se va da click pe id-ul acestuia din tabelul cu toti clientii) .

3. Proiectare

3.1. Structuri de date

Una dintre primele probleme aparute in faza de proiectare presupune alegerea unor structuri de date corespunzatoare modului de lucru cu clasele si stocarii anumitor valori pe perioada efectuarii operatiilor dorite.

HashMap este una dintre colectiile predefinite in Java care se bazeaza pe tabele de dispersie. Interfata Map a fost create in Java pentru a putea implementa structure de siruri

asociative. Alte sturcturi care se bazeaza pet abele de dispersie sunt: HashTable, LinkedHashMap, HashSet.

In java Map contine o colectie de perechi de tipul cheie – valoare definite astfel:

Entry < K , V >.

Pentru a putea intelege hashing-ul in java trebuie intelese urmatoarele notiuni: functie de hash, valoare de hash, bucket. Deoarece Java este un limbaj de programare orietat pe obiecte, cheile vor fi obiecte.

Pentru a determina un cod dintr-un obiect, obiectul respective va continte functia public int hashCode() pe care a va suprascrie astfel incat aceasta sa returneze un intreg pe baza valorilor campurilor obiectului respective. Functia de hash trebuie sa returneze valori diferite pentru obiecte diferite si valori egale pentru obiecte identice. Functia de hash se va aplica codului pentru a determina indexul astfel INDEX=hash (hashCode (K)). Coliziunile sunt rezolvate prin chaining, adica fiecare bucket contine o lista inlantuita (LinkedList).

Un ArrayList este o colectie simpla care poate stoca orice tip de obiect, fapt ce permite utilizarea ArrayList-ului pentru definirea listei de conturi a unui client. ArrayList este una dintre cele mai utilizate colectii predefinite in Java, avand multe functii predefinite care sunt usor de utilizat.

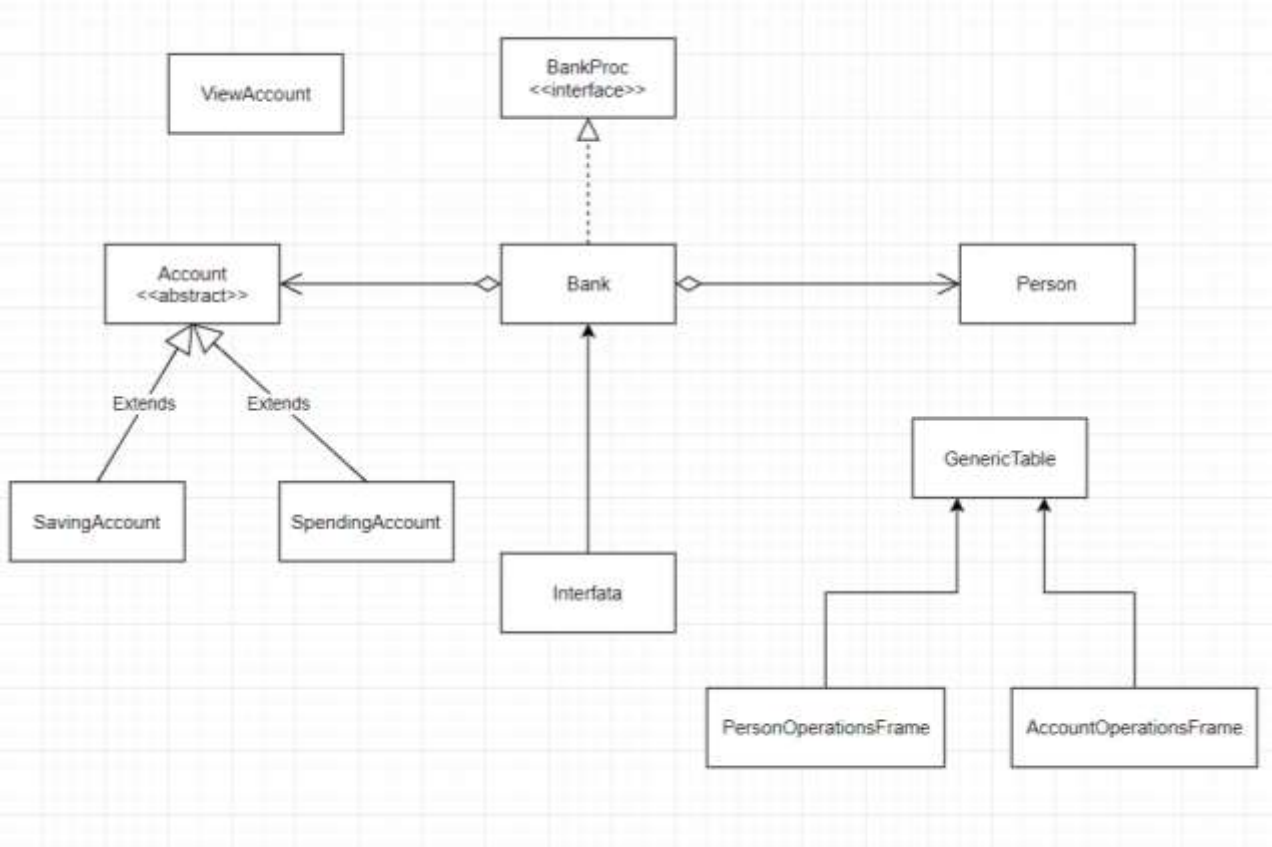
Asadar, se va folsi un HashMap<Person, List <Account> > , pentru a stoca inforamtiile din banca si pentru a le utiliza, astfel fiecare persoana putand avea mai multe conturi.

3.2. Diagrama de clase

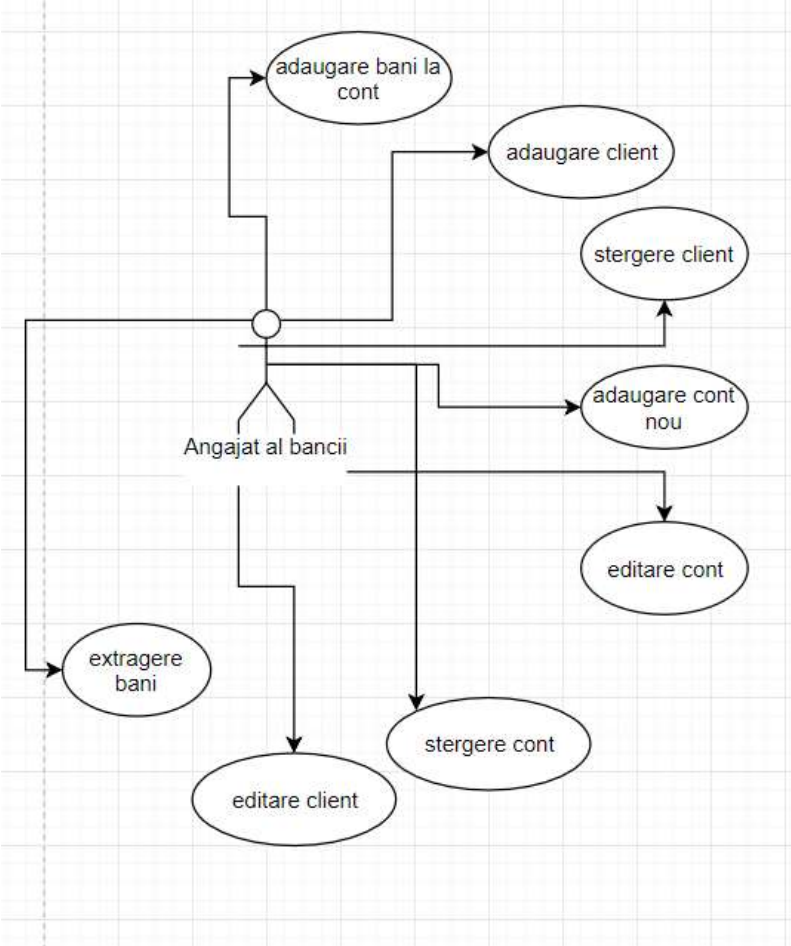
Pentru a descrie modul in care am proiectat acest proiect am ales sa utiizez mai multe tipuri de dagrame UML. Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de modele și specificații pentru software. Este folosita pentru reprezentarea vizuală a claselor și a interdependențelor, taxonomiei și a relațiilor de multiplicitate dintre ele. Diagramele de clasă sunt folosite și pentru reprezentarea concretă a unor instanțe de clasă, așadar obiecte, și a legăturilor concrete dintre acestea. UML oferă o largă gamă de diagrame pentru modelarea diferitelor situații în cadrul unui proiect de dezvoltare software.

- Diagrama de clasa:o diagrama de clasa reprezentand multiplicitatea dintre doua clase.Este folosita pentru reprezentarea vizuala a claselar si a interdependentelor, taxonomiei si a relatiilor de multiplicitate dintre ele. Diagramele de clasa sunt folosite și pentru

reprezentarea concreta a unor instanțe de clasa, asadar obiecte, si a legaturilor concrete dintre acestea.



- Diagrama use case: Datorita simplitatii ei este utilizaaă în special în discutiile dintre firma dezvoltatoare de software și clienti sau utilizatori. Diagramele use case reprezinta interactiunea dintre elementele exterioare unui sistem (numite si actori) și sistem. In cazul acestor diagrame se prezinta actiunea desfăsurata de sistem la interactiunea actorului, insa modul în care sistemul desfășoare acea actiune nu trebuie sa fie reprezentat intr-o astfel de diagrama (conceptul blackBox).



4. Implementare

Metoda de implementare aleasa contine 4 pachete:

- BankAccounts
- BankClients
- BankPackage
- GUI

4.1 GUI

Pachetul GUI contine patru clase: Interfata, GenericTable, AccountOperationsFrame, PersonOperationsFrame.

Clasa GenericTable contine metoda generica createTable, care primeste ca argument un ArrayList de obiecte de tipul T si creeaza un model de tabel (Default Table Model). Metoda ia campurile obiectului transmis si le introducere in headerul tabelului, iar apoi ia valorile campurilor pentru fiecare obiect. Metoda este apelata din clasele AccountOperationsFrame si PersonOperationsFrame. Acestea folosesc JTable create cu ajutorul metodei createTable din clasa GenericTable pentru a afisa informatii legate de banca.

Clasa Interfata contine aranjarea componentelor in frame la rularea aplicatiei si actionListeneri pentru butoane. Prima fereastra a aplicatiei contine doua butoane : Person Operations si Account Operations. La apasarea unuia dintre aceste doua butoane, se va crea o nou instanta a clasei PersonOperationsFrame, respectiv AccountOperationsFrame. Astfel, pentru fiecare dintre cele doua se va deschide o noua fereastra.

Ambele ferestre care se pot deschide au ca si structura de baza un JTabbedPane, care lasa posibilitatea de a selecta una dintre operatiile dorite din acea categorie.

Fereastra cu operatii cu personae (daca se da click pe butonul cu Person Operations), are 3 posibilitati: add, edit, delete. Pentru a adauga o persoana se completeaza toate campurile data si se apasa pe butonul “Add”. Editarea unei personae presupune corectarea numelui sau al prenumelui in cazul in care acestea au fost introduse gresit. Pentru a putea edita o persoana deja existent in banca, se da click pe id-ul persoanei din tabelul cu acestea si se completeaza campul cu nume, campul cu prenume sau ambele, iar apoi se apasa butonul “Edit”. Pentru stergerea unei persoane se selecteaza id-ul acesteia din tabela cu persoanele si se apasa pe butonul delete.

Pentru a nu fi posibila introducerea unui id care nu există in banca, s-a setat campul cu id-ul sa nu poata fi editat, cee ace inseamna ca acesta poate fi completat numai prin selectarea id-ului dorit din tabela.

Toate modificariile care vor fi efectuate asupra unei persoane in una dintre cele trei tab-uri va fi vizibila si in tabelurile din celelalte tab-uri.

Fereastra cu operatii cu conturi (daca se da click oe butonul cu Account Operations), are 5 posibilitati: add, edit, delete, deposit si withdraw. Pentru a adauga un cont, din tabela de persone se selecteaza id-ul titularului de cont (al persoanei care va detine contul nou creat), apoi se selecteaza una dintre cele doua optiuni “Savings Account ” sau “Spending Account”, iar in functie de alegerea facuta, utilizatorul va trebui sa completeze inca un camp: pentru Saving Account cu perioada pentru care va fi deschis contul, iar pentru Spending Account cu suma initiala. Conturile de economii vor fi create initial cu suma 0, iar apoi va fi posibila o singura depunere si retragere de bani. Editarea unui cont presupune schimbarea titularului acestuia si se va realiza prin selectarea din tabela cu conturi a id-ului contului de transferat si din tabela cu persoane a id-ul noului titular.

Stergerea unui con teste identica cu stergerea unei persoane. Pentru depunerea sau retragerea unei sume de bani se va selecta din tabela id-ului contului cu care se opereaza si apoi se va introduce suma dorita, cu mentiunea ca la conturile de economii, trebuie retrasa toata suma existent in acesta.

4.2 BankClients

Pachetul BusinessClients contine 1 clasa: Person.

Clasa Person contine atributele id, nume, prenume, birthday si cnp, gettere si settere pentru aceste atribute si suprascrie metodele hashCode() si equals().

4.3 BankPackage

Pachetul BankPackage contine interfata BankProc si clasa Bank.

Interfata BankProc contine defintiile de metode care vor fi implementate in clasa Bank, iar fiecare metoda este precedata de pre si post conditii, care in clasa bank vor fi verificate cu assert. Metodele din interfata BankProc sunt: addPerson, editPerson, removePerson, addAccount, editAccount, removeAccount, addMoneyToAccount, withdrawMoneyFromAccount, readData, writeData.

Clasa Bank implementeaza interfata BankProc, metodele definite in aceasta si alte metode auxiliare. Practic, clasa Bank contine cea mai mare parte din logica aplicatiei si toate operatiile pe care aceasta trebuie sa le indeplineasca.

4.4 BankAccounts

Pachetul BankAccounts contine 4 clasa: Account, SavingAccount, SpendingAccount, ViewAccount.

Clasa abstracta Account are ca si atribute id-ul contului si suma,are gettere si settere pentru acestea si metodele abstracte addMoney si withdrawMoney.

Clasele SpendingAccount si SavingAccount extend clasa Account.

Pentru SavingAccount se considera ca se poate efectua o singura depunere si o singura retragere de bani, dar banca ofera dobanda pe perioada pe care banii stau in banca. De aceea, clasa SavingAccount are ca atribut in plus fata de clasa Account pe care o extinde atributul interest care este fixat la 0.25 si atributul period care reprezinta durata perioadei cat banii stau in banca.

Pentru SpendingAccount se poat efectua oricate depuneri si retrageri de bani, cu specificatia ca suma care se doreste a fi retrasa nu poate sa depaseasca valoarea sumei existente in cont.

Clasa ViewAccount are patru atribute: holderName, idCont, suma, accountType si un contructor, fiind folosita pentru a putea afisa informatiile despre conturi intr-un JTable, folosind metoda createTable din clasa GenericTable, astfel afisand informatiile commune celor doua si avand un camp special pentru tipul de cont.

5. Testare si Rezultate

Primul pas in testarea corectitudinii aplicatiei este verificarea daca datele de intrare sunt introduse corect de catre utilizator si apoi preluate corect pentru folosire. In cazul in care datele introduse de utilizator nu sunt corecte (nu au formatul numeric sau sir de caractere, in functie de necesitate sau nu există in baza de date), acesta va fi instiitat printr-un mesaj de eroare ca nu a introdus corect datele.

Testarea se realizeaza cu ajutorul JUnit si al driverelor de test definite. Pentru aceasta există pachetul test care contine 5 clase: 4 clase de test si una care ruleaza simultan toate testele.

Cele patru clase de test verifica daca depunerea si retrafgerea baniilor din cele doua tipuri de conturi functioneaza in mod corect.

Operatie	Date de intrare	Rezultat asteptat	Rezultat obtinut	PASS/FAIL
Depunere SavingAccount	Un cont, in care se depun 400. Fiind cont de economii acesta are initial 0.	400	400	PASS
Retragere SavingAccount	Un cont in care se depun 400 si apoi sunt reatrasi.	0	0	PASS
Depunere SpendingAccount	Un cont care are initial 100 de lei si se depun 400.	500	500	PASS

Retragere SpendingAccount	Un cont care are initial 1000 si se retrag 400	600	600	PASS

6. Concluzii si Dezvoltari Ulterioare

Acest proiect este unul care lasa programatorului o posibilitate crescuta de a interpreta problema si modalitatiile de rezolvare, de aceea din punct de vedere al dezvoltariilor ulterioare există numeroase posibilitati.

- Interfata grafica mai atractiva
- Posibilitati de logare atat ca administrator cat si ca client si impartirea functionalitatilor aplicatiei in functie de utilizator (de exemplu clientul sa poata realiza singur operatiile de depunere si retragere sis a nu aiba acces la celelalte operatii)
- Update sincron intre tabelele din Person Operations si cele din Account Operations
- Dezvoltare aplicatiei prin adaugarea mai multor functionalitati: schimbare valuta (de exemplu din lei in euro) , generarea unor fisiere text sau pdf cu datele unui cont (similar unui extras de cont)

7. Bibliografie

http://www.coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW4_Tema4/HW4_Tema4.pdf

http://www.coned.utcluj.ro/~salomie/PT_Lic/4_Lab/HW4_Tema4/HW4_Tema4_Hashing.pdf

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

https://www.javaworld.com/article/2077258/learn-java/observer-and-observable.html

https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html

https://www.javaworld.com/article/2074956/learn-java/icontract--design-by-contract-in-java.html

https://www.draw.io/

https://stackoverflow.com/