Assignment 4
Administrare banca

Student: Găvenea Radu
grupa 30292

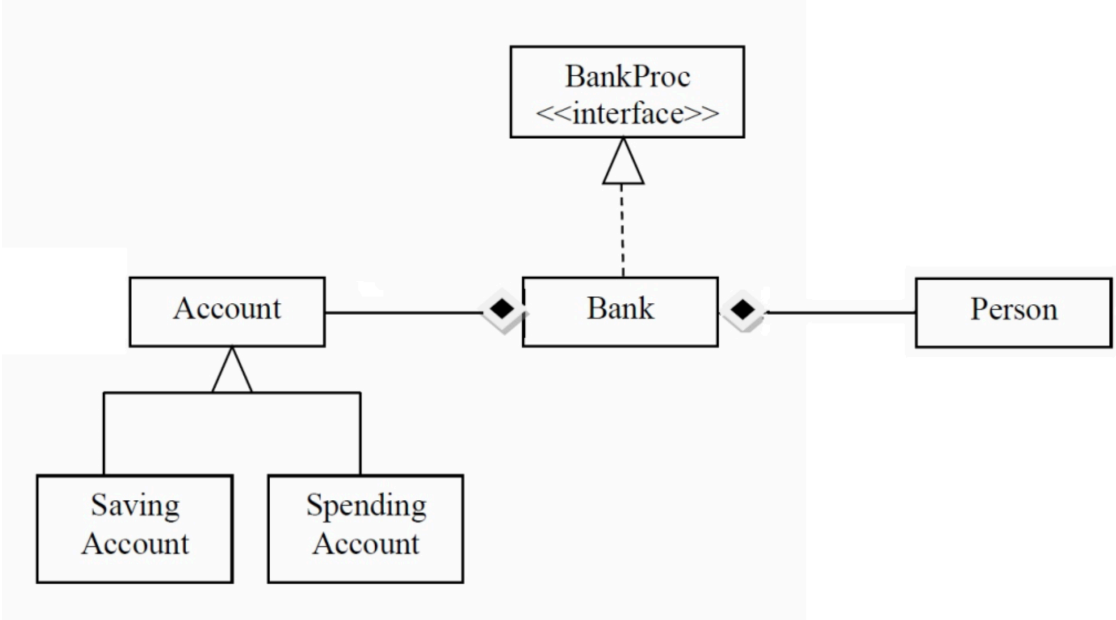
1. Obiectivul temei

1.1. Cerinta

Poriectati si implementati o aplicatie ce are rolul de a administra sistemul de functionare si tranzactionare conturi ale unei bancii. Aplicatia ofera utilizatorului posibilitate de administrare a tuturor clientilor bancii, precum si efectuarea de tranzactii asupra conturilor acestora. Clienti au la dispozitie doua tipuri de conturi, cont de economii si cont de credit, in care pot depune, respectiv retrage bani.

1.2. Descrierea problemei

Aplicatia va fi implementata pornind de la urmatoarea diagrama de clase necesare:



Aplicatia trebuie sa implementeze interfata BankProc, care va contine operatii de citire, editare, aduagare si stergere asupra clientilor si a conturilor, precum si metode de efectuare a tranzactilor asupra conturilor si generare de rapoarte.

Vor fi implementate clasele <Account>, <Person>, <SavingAccount>, <SpendingAccount>, unde se va folosi mostenirea, conform diagramei de mai sus.

Se va folosi pattern-ul Observer pentru a face posibila notificarea unui utilizator in momentul in care se efectueaza modificari asupra unui cont personal.

Aplicatia foloseste o structura de tip hashtable pentru stocarea in cadrul aplicatiei a datelor despre clientii si conturile acestora. Cheia hashtable-ului va fi reprezentata de detinatorul conturilor.

Aplicatia foloseste tablele de tipul JTable pentru afisarea datelor cu privire la informatiile clientilor si conturilor.

Se va implementa o unitate de testare pentru verificare bunei functionari a sistemului.
Datele clientiilor si ale conturilor acestora vor fi serializate si stocate intr-un fisier.

2. Analiza problemei

Aplicatia are rolul de a simula un sistem de administrare a clientilor unei banci si a conturilor pe care acestia le detin. Pentru a face acest lucru posibil aplicatia ofera multiple functionalitati, acestea fiind impartite in trei parti esentiale. In functie de aceste functionalitatii, aplicatia a fost impartita, si din punct de vedere al interactiunii cu utilizatorul, in trei componente de interfata grafice(ferestre) diferite, si anume: Componenta de administrare a clientilor bancii, unde sunt efectuare operatii de creere, vizualizare, editare si stergere asupra entitatilor clienti; componenta de administrare a conturilor, unde sunt posibile efectuarea de operatii de vizualizare, de creare, de editare si de stergere asupra conturilor inregistrate la banca; componenta de efectuare tranzacti asupra conturilor, unde utilizatorul are posibilitatea de a efectua operatii de depunere si retragere de bani.

2.1. Functionalitati principale

2.1.1. Administrarea clientilor

Utilizatorul aplicatiei are la dispozitie posibilitatea de efectuare a operatilor de vizualizare, de adaugare, de editare si de stergere asupra clientilor inregistrati in cadrul sistemului. Aceste operatii sunt disponibile in cadrul unei ferestre separate de restul aplicatiei. Afisarea acestor clienti este realizata in cadrul unui tabel din care se pot selecta clienti, si un formular de adaugare unei noi persoane.

2.1.2. Administrarea conturilor

Utilizatorul aplicatiei poate efectua operatii de vizualizare, de adaugare, de editare si de stergere asupra conturilor aflate in stocul depozitului. In aceasi maniera ca si in cazul clientilor, manipularea conturilor se va face in cadrul unei ferestre diferite in cadrul aplicatiei. Tot ca in cazul clientilor, conturile sunt afisate intr-un tabel, fiind prezente spre vizualizare doar conturile respective persoanei selectate. Se poate introduce cont nou in cadrul unui formular prezent in fereastra.

2.1.3. Efectuarea de tranzactii

Efectuarea tranzactiilor poate fi realizata in cadrul unei ferestre specifice in cadrul aplicatiei, in care clientul poate vizualiza informatii specifice despre contul selectat. Utilizatorul are la dispozitiei posibilitatea de extrage sau de a depune bani in contul curent selectat. In cazul in care contul selectat este un cont de tipul cont de economii, utilizatorul poate efectua o singura operatie de depunere si o singura operatie de extragere de bani. In schimb, in cazul in care contul selectat este unul de tip cont de cheltuieli, utilizatorul poate face cate operatii doreste, atat de depunere de bani, cat si de retragere de bani. Tot in cadrul acestei ferestre se mai afla si un buton folosit pentru reinitializarea fisierului in care sunt salvate informatiile, cat si un buton folosit pentru salvarea starii actuale cu privire la informatiile clietntilor bancii.

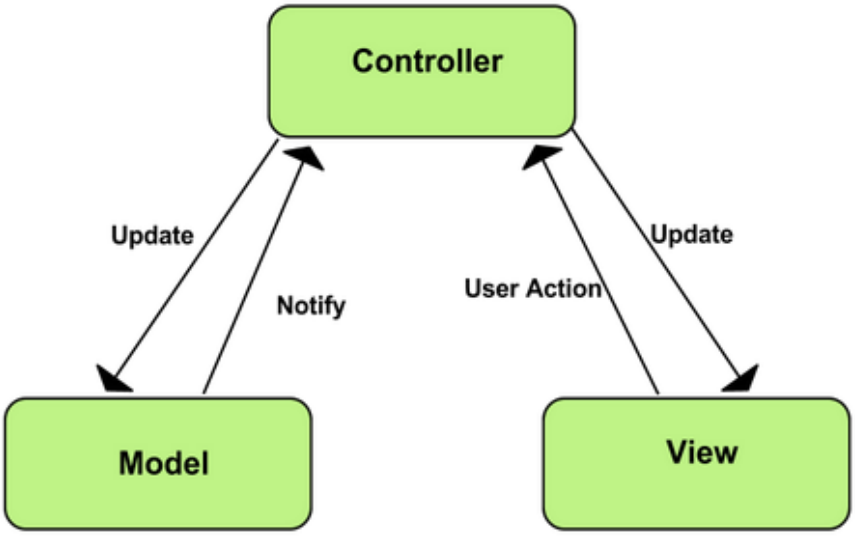
2.1.4. Generarea de rapoarte

O alta functionalitate pe care aplicatia o pune la dispozitie este aceea ca utilizatorul poate genera rapoarte cu privire la stocul produselor din depozit. Acest lucru este posibil prin accesare ferestrei de Rapoarte in care sunt disponibile doua tipuri de rapoarte ce pot fi generate, si anume: raport in care sunt afisate toate produsele din depozit ce sunt in numar mai mic de 10 si un raport in care sunt specificate toate produsele ce au pretul de cumparare cuprins intre 100 si 200 de unitati monetare. Aceste rapoarte vor fi generate in format csv.

3. Proiectarea

3.1. Arhitectura Model-View-Controller(MVC)

Aplicatia foloseste o structura de tipul mvc, care este un model arhitectural raspandit in ingineria software. Acest tip de model are la baza conceptul de izolare a logicii de business fata de partea de interfata cu utilizatorul, astfel incat aplicatia poate fi mai usor de realizat si modificat ulterior, putandu-se lucra doar la nivele dorite.



Model

In cadrul acestui nivel este realizata partea de business logic, aici sunt manipulate datele aplicatiei si tot aici se regasesc toate operatiile folosite pentru manipularea acestora. In cazul aplicatiei de fata, in cadrul efectuarii de tranzactii, in partea de model sunt prezente clasele de reprezentare a entitatilor corespunzatoare tabelelor din baza de date.

View

Acest nivel face posibila interactiunea cu utilizatorul. Aici este realizata reprezentarea grafica a aplicatiei si exprimarea sub forma grafica a datelor. Aici este evidentiata informatia, inainte de a ajunge la controller. In cazul aplicatiei noastre, este folosita o interfata dezvoltata cu ajutorul graficii swing, impartita in 3 ferestre(tab-uri) diferite.

Controller

Acesta este nivelul de unde aplicatia poate fi controlata. Aici sunt manipulate rute, fisiere, metode care se afla in partea de model, si se face legatura intre partea de model si partea de view a sistemului. In cadrul aplicatiei cu polinoame, nivelul controller este implementat sub forma unor listeneri care in urma actiuni de apasare a unui buton de catre utilizator executa toate operatiile prezente in cadrul aplicatiei.

3.2. Cazuri de utilizare

Pentru realizarea unui depozit online de produse este necesar sa consideram cazurile de utilizare pentru un eventual utilizator.

Caz de utilizare: Adaugare bani in cont de economii

Actorul principal:
- utilizatorul

Preconditii:
- aplicatia trebuie pornita

Flow principal:
1. selectare a unui client din lista de clienti a magazinului
2. selectare deschidere fereastra de administrare conturi
3. selectare unui cont de economi din lista de conturi afisata
4. selectare deschidere fereastra tranzactii
5. adaugare suma dorita in campul corespunzator
6. apasare buton <Add>

Flow alternativ:
6.1. sistemul returneaza mesaj eroare ca a mai fost facuta o depunere de bani in cadrul aceluiasi cont.
6.2. se revine la pasul 3.
6.3. se selecteaza un cont diferit din lista de conturi disponibile in tabel

Postconditii:
- sistemul afiseaza mesaj de succes cu privire la finalizarea tranzactiei

Caz de utilizare: Retragere bani din cont de cheltuieli

Actorul principal:
- utilizatorul

Preconditii:
- aplicatia trebuie pornita

Flow principal:
1. selectare a unui client din lista de clienti a magazinului
2. selectare deschidere fereastra de administrare conturi
3. selectare unui cont de cheltuieli din lista de conturi afisata
4. selectare deschidere fereastra tranzactii
5. adaugare suma dorita in campul corespunzator
6. apasare buton <Withdraw>

Flow alternativ:
6.1. sistemul returneaza mesaj eroare ca nu sunt suficienti bani in cont
    6.2.1. se introduce o suma mai mica in campul corespunzator pentru retragere
    6.2.2. se revine la pasul 3 si se selecteaza alt cont

Postconditii:
- sistemul afiseaza mesaj de succes cu privire la finalizarea tranzactiei

Diagrama UML Use-Case

In urmatoarea diagrama sunt prezentate toate cazurile de utilizare pe care un utilizator le are la dispozitie.

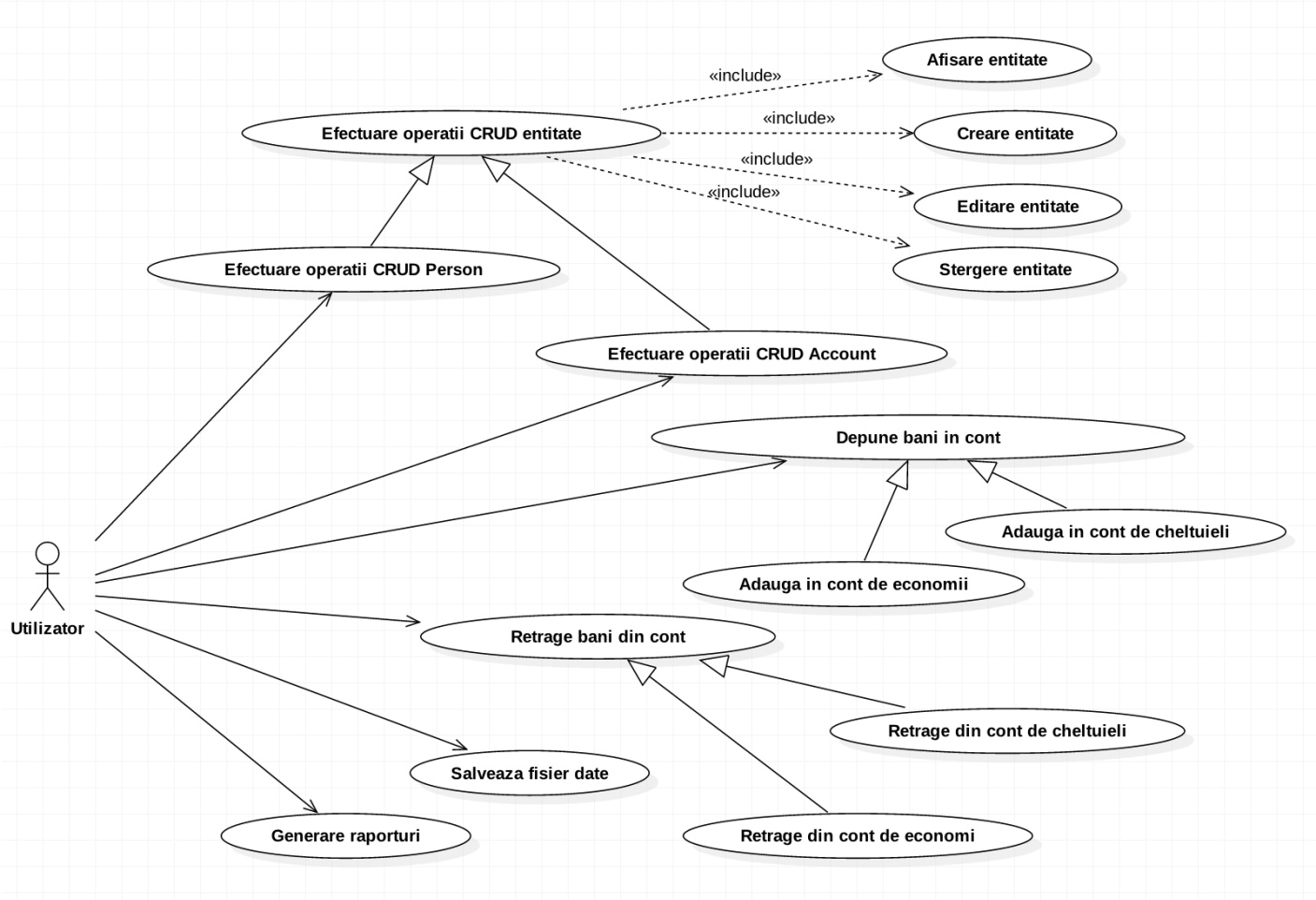


Diagrama use-case Administrare banca

3.3. Stocare datelor

Informatiile despre clienti si conturile asociate pe care le detine banca sunt stocate intr-un fisier local. Acestea sunt citite la pornirea aplicatiei din fisier si salvate la terminarea aplicatiei in cadrul aceluiasi fisier. Avand in vedere ca aplicatia foloseste o structura de date pentru utilizarea acestor informatii, acestea sunt serializate inainte de a fi salvate, respectiv deserializate dupa citirea din fisier.

4. Implementare si testare

4.1. Diagrama de clase

Din diagrama de clase prezentata in figura de mai jos se poate observa destul de usor arhitectura pe nivele folosita de tipul mvc.

Avem astel in partea de sus reprezentat nivelul de presentation.

In parte de mijloc a diagramei parcursa de sus in jos observam pe orizontala nivelul de controllere a aplicatiei, apoi un nivel mai jos este reprezentat nivelul de business logic al aplicatiei.

Iar in partea de jos a diagramei se poate observa nivelul de acces la date.

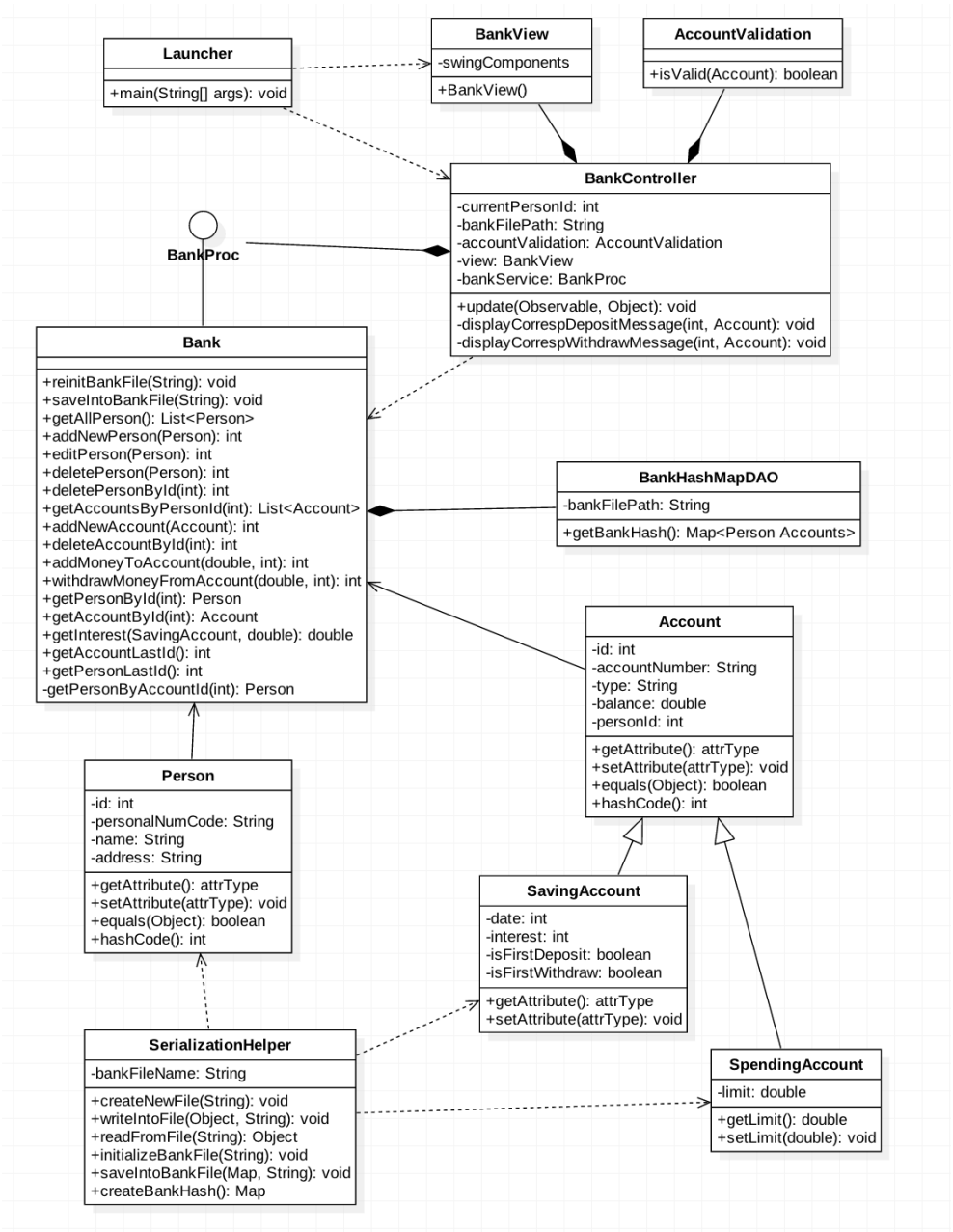


Diagrama de clase U.M.L.

4.2. Structura aplicatiei pe pachete

Dupa o analiza preliminara, am hotarat sa impartim aplicatia in urmatoarele pachete. Aceste pachete sunt alese in asa fel incat sa corespunda cat mai bine unei aplicatii pe nivele de tip MVC:

4.2.1. Pachetul <bank>

Acest pachet are ca obiectiv pornirea aplicatiei si mentinerea sesiunii de cumparaturi pentru un client.
Aici se regaseste clasa Launcher.
Clasa Launcher contine metoda main din care aplicatia porneste prin instantierea controlerului aplicatiei <BankContoller>.

4.2.2. Pachetul <dataAccessLayer>

Acest pachet cuprinde clasele folosite pentru a facilita accesul la date. Toate entitatile dorite in cadrul programului sunt reprezentate printr-o clasa specifica cu nume corespunzator entitatilor folosite in cadrul aplicatiei: Account, Person, BankHashMap. De asemenea, in cadrul acestui pachet sunt implemetate metodele de serializare si deserializare a obiectelor stocate in fisierul local.

Clasa <BankHashMapDAO> este o clasa ce are rolul de accesa obiectele de tipul bankHashMap care sunt serializate si stocate pe disk in cadrul unui fisier local. Metoda getBankHashMap() returneaza o structura de date de tipul Map<Person, List<Account>> folosita pentru stocarea temporara in cadrul aplicatiei a informatiilor cu privire la datele clientilor bancii si a conturilor acestora. Aceasta structura este folosita in cadrul clasei <Bank> pentru procesarea tuturor operatiilor din cadrul aplicatiei.

Clasa <SerializationHelper> este o clasa statice ce are rolul de efectua toate operatiile de serializare si deserializare a obiectelor stocate in fisierul local. De asemenea, tot in cadrul acestei clase sunt prezente si metode de creare a fisierului in cazul in care acesta nu exista.
Alte metode prezente sunt cele utilizate pentru initializarea fisierului cu date noi in cazul in care se doreste acest lucru, precum si metode de salvare a starii curente a structurii de date din cadrul sistemului. Tot aici este creata structura folosita de clasa <Bank> pentru procesarea tuturor operatiilor ce urmeaza a fi efectuate.

4.2.3. Pachetul <model>

In cadrul acestui pachet sunt pastrate toate entitatile folosite de sistem, iar acestea sunt urmatoarele: <Account>, <Person>, <SavingAccount> si <SpendingAccount>. Aceste patru clase contin atributele ce definesc fiecare obiect in parte si metode de get() si set() pentru fiecare dintre aceste atribute. De asemenea, deoarece folosim structuri de date din cadrul Collection Framework, atat pentru clasa <Person>, cat si pentru clasa <Account> s-au definit metode ce suprascriu metodele hashCode() si equals().
Dupa cum se poate observa si din diagrama de clasa mai sus prezentata, atat clasa <SavingAccount>, cat si clasa <SpendingAccount> extind clasa <Account>, mostenind toate atributele acesteia.

Clasa <SavingAccount> extinde <Account> prin atributele necesare procesarii dobanzii disponibile pentru acest tip de cont, precum si doua proprietati care au rolul de a verifica daca s-a efectuat anterior o tranzactie de depunere sau de retragere de bani din contul respectiv.
Clasa <SpendingAccount> extinde <Account> print-o proprietate de limita, cu ajutorul careia se poate stabili limita maxima a sumei care poate fi extrasa din cont.

4.2.4. Pachetul <businessLayer>

In cadrul acestui pachet sunt prezente toate clasele necesare pentru efectuarea tuturor operatiilor de manipulare si procesare a datelor din cadrul aplicatiei. Acesta este practic nivelul de servicii pe care le are sistemul, unde se implementeaza toata logica aplicatiei. Este un nivel intermediar intre nivelul de control (Controllers) care apeleaza metode din cadrul claselor aflate in businessLAyer si nivelul de <dataAccessLayer> care ofera datele ce voi fi procesate in cadrul acestor clase.

Acest pachet are in componenta sa 2 clase de servicii si o interfata folosita pentru accesarea acestor metode din cadrul claselor. Folosirea interfetei este implortanta pentru delimitarea nivelelor de logica si pentru a oferi contractul pe care un nivel trevuie sa il indeplineasca pentru a acesa metodele claselor din nivelul inferior.

Interfata <BankProc> contine toate metodele disponibile si implementate in clasa <Bank>. Aceste metode vor fi detaliate in cadrul prezentarii clasei <Bank>. Aceasta interfata este folosita pentru delimitarea nivelelor de logica si pentru a oferi contractul pe care un nivel trevuie sa il indeplineasca pentru a acesa metodele claselor din celalalt nivel.

Clasa <Bank> implementeaza interfata <BankProc> si contine toate implementarile pentru metodele acesteia. In cadrul acestor metode se regaseste toata logica aplicatei, metode cu ajutorul carora se manipuleaza si proceseaza datele din aplicatie si ofera componenta de servicii necesare. Clasa are definita o structura de tipul Map<Person, List<Account>> care este folosita pentru stocarea datelor obtinute prin deserializarea unui fisier. Toate operatiile care se efectueaza in cadrul acestei clase se fac asupra acestei structurii.
In cadrul acestei clase sunt medodele folosite pentru vizualizarea clientilor, adugarea de clienti noi, editarea unui client, stergerea unui client, precum si de vizualizare a conturilor, de adaugare a unui cont, de modificare a unui cont si stergere a unui cont. Pentru parcurgere si manipularea structurii de date au fost folosite operatii agregate asupra stream-urilor disponibile in Java 8, precum si expresii lambda.
Tot in cadrul acestei clase se regasesc si metodele de procesare a tranzactiilor asupra conturilor aflate in sistemul bancii. Aceste metode permit depunerea si retragerea de bani dintr-un cont si ofera posibilitatea de a diferentia intre tipurile de conturi. In acest sens ofera functionalitati diferite pentru cele doua tipuri diferite de conturi care exista. Acest lucru este posibil verificand tipul de instanta al contului si luarea de decizi diferite in functie de tipul gasit. Adaugarea si retragerea de bani se face in cate o metoda diferita pentru fiecare tip de cont diferit.

Clasa <AccountValidation> este clasa folosita pentru verificarea obiectului <Account> inainte de a fi adaugat in structura de date corespunzator clientului in cauza. Aceasta verificare este necesara pentru a ne asigura ca nu exista inconsitenta in cadrul Obiectelor de tip <Account> prin verificarea corectitudinii datelor introduse de utilizator.
Un exemplu in acest sens este verificarea tipului de cont. Utilizatorul are la dispozitie introducere a tipului de cont pe care un cont nou creat il va avea, iar optiunile disponibile sunt “spending” si “saving”. In cazul in care utilizatorul introduce alt sir de caractere in carul campului <type> din interfata grafica, aplicatia va afisa un mesaj de eroare pentru a notifica utilizatorul cu privire la aceasta eroare.

4.2.5. Pachetul <controllers>

Acest pachet contine logica de control a functionarii aplicatiei din cadrul arhitecturi mvc, facand legatura intre partea de view <BankView> si apelarea servicilor disponibile in clasa <Bank>.

Pachetul contine clasa <BankController> care are rolul de intermedia efectuarea operatiilor pe baza cerintelor provenite de la utilizator, evidentiate cu ajutorul listerner-ilor disponibili in view. Aceasta clasa contine mai multe sub-clase dedicare implementarii acestor listerner-i care au rolul de a specifica operatiile ce trebuie produse. De asemenea, aici sunt prezenti atat listener-i asupra butoanelor pe care utilizatorul le poate apasa, cat si listener-i asupra tabelelor, care au rolul de specifica aplicatiei diferite informatii cu privire la modificarea starii acestora.
Tot in cadrul acestei clase care implementeaza interfata <Observable> avem si metoda de update() corespunzatoare acestei interfete, ce are rolul de a modiifca datele din view cand datele sunt modificate in model, precum si rolul de anunta utilizatorul ca o tranzactie a fost produsa asupra contului sau.

4.2.6. Pachetul <presentation>

Acest pachet ca scop prezentarea sub forma grafica a aplicatiei. Aici este creata interfata grafica disponibila pentru ca utilizatorul sa poata interactiona cu sistemul.

Acest pachet contine o singura clasa <BankView> ce are rolul de furniza interfata utilizator pentru aplicatie. Acest lucru este realizat cu ajutorul framework-ului Swing pus la dispozitie de limbajul Java.

4.2.7. Pachetul <presentationTableUtils>

Acest pachet este folosit pentru realizarea unei functionalitati de generare a tabelelor din cadrul interfetei grafice pe baza numelor atributelor entitatilor prin metoda de reflexie in mod automat. In acest sens toate tabelele din aplicatie vor avea ca si coloane numele atributelor corespunzatoare.

Pentru acest lucru a fost creata o clasa generica <GenericTableModel<T>> care extinde clasa <DefaultTableModel> si suprascrie unele din metodele aceteia. De asemenea, aceasta clasa contine metodele necesare pentru realizarea tabelelor prin metode ce citesc prin reflexie numele coloanelor, si primesc ca parametrii de intrare Vectori generici de tip T.

4.3. Testare

Functionalitatea aplicatiei este testa printr-un bloc de teste de tip unit test. Aceste teste verifica toate operatiile de vizualizare, de creare, de editare si de stergere care se pot efectua asupra entitatilor din cadrul aplicatiei. Astefl au fost create 2 module de teste pentru fiecare entitate in parte, si anume: <AccountTests> si <PersonTests>. Pe langa operatiile de vizualizare, aduagare, editare si stergere asupra unui cont, mai sunt prezente in <AccountTests> si teste de efectuare de depunere si retragere de bani din cont.

Pentru testare este folosit un fisier de stocare a datelor serializate separat ce are ca scop doar rularea acestor teste.
Toate aceste clase de teste contin in faza de initialiare aplelarea unei metode de initializare a fisierului corespunzator testarii. Aceasta metoda are rolul de a re-initializa baza de date cu date corecte pentru procesarea testelor.

5. Rezultate

Aplicatia functioneaza conform asteptarilor/ cerintelor. Utilizatorul acesteia poate efectua o tranzactie prin adaugarea unei sume in campul corespunzator si apasarea butonului corespunzator.
In cazul in care o tranzactie a fost finalizata cu succes, un mesaj corespunzator este afisat de catre sistem pentru a confirma acest lucru.

Iar in cazul in care erori de folosire a aplicatiei apar, de asemenea, sunt generate de sistem mesaje corespunzatoare pentru a informa utilizatorul ce trebuie sa faca sau ce pasi sa urmeze. Un exemplu in acest sens este cazul in care utilizatorul incearca sa depuna o suma de bani intr-un cont de economii, asupra caruia a mai fost efectuata o tranzactie de depunere aterior(acest lucru nu este tolerat datorita specificatiilor aplicatiei).

6. Concluzii

Aplicatia este una pur didactica si nu are aplicabilitate in viata reala, fiind doar un mic prototip pentru o aplicatie ce ar necesita mai multe resurse de dezvoltare.

Dezvoltari ulterioare pentru aceasta aplicatie pot fi reprezentate prin introducerea urmatoarelor functionalitati: adaugarea unui sistem de autentificare pentru fiecare client in parte, posibilitatea de generare a raporturilor cu privire la tranzactiile efectuate de un client si posibilitatea de informare a acetuia prin trimiterea unui e-mail cu privire la aceste modificari.

7. Bibliografie

3. http://www.wikipedia.org/
4. http://stackoverflow.com/