**Simularea unei aplicatii pentru procesarea unor conturi bancare**

**TEMA 4**

Nume: Adam Cristina-Ioana

Grupa:30224

Facultatea de Automatica si Calculatoare

Specializarea: Calculatoare

Cuprins:

[1.Obiectivul temei 3](#_Toc514327438)

[2. Analiza problemei, asumptitii, scenarii, cazuri de utilizare, erori 3](#_Toc514327439)

[3. Proiectare(decizii de proiectare, diagram UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor) 6](#_Toc514327440)

[4. Implementare 7](#_Toc514327441)

[5. Testare 8](#_Toc514327442)

[6. Rezultate 9](#_Toc514327443)

[7. Cocluzii 9](#_Toc514327444)

[8.Bibliografie 9](#_Toc514327445)

# 1.Obiectivul temei

Obiectivul acestei teme este acela de a omplementa si simula o aplicatie care se bazeaza pe procesarea conturilor unei banci . Implementarea presupune adaugarea de conturi , stergerea unor conturi editarea acestor conturi adica de a adauga sau de a scoate o suma de bani si afisarea conturilor si executarea acestor comenzi in cazul in care nu exista contul cautat atunci sau nu sunt completate toate campuri atunci programul va da o eroare . Un alt obiectiv al acestei teme este acela de a scrie si implementa tehnicile de programare prin contract si anume sa fie scrise pre conditii , post conditii si o metoda de iswellFormed ( ) pentru a vedea daca structura si-a pastrat compozitia .

Acest siste are o raspandire foarte larga aceasta gasinduse peste tot . La toate bancile se gaseste o asemenea interfata pentru editarea conturilor si pentru gasirea conturilor clinetilor . Aceasta aplicatie poate sa fie si in alte domenii si nu numa in domeniul bancar , aceasta poate sa fie folosita pentru logarea si crearea unor conturi pe un anumit server .

Principala problema care se pune este gasirea pre condiilor si post conditiilor pentru metodele din clasele acestui proiect .Se observa din digrama UML ca post conditiile si pre conditiile cele mai importatnte se afla in classa Bank deoarece aici se afla un vector de obiecte de tip Accounts si aici se modifica conturile sau se sterg sau se adauga noi conturi .

# 2. Analiza problemei, asumptitii, scenarii, cazuri de utilizare, erori

## 2.1 Analiza problemei

Dupa citirea problemei se poate observa ca putem avea cel putin 6 clase : o clasa Person care sa aiba ca atribute un nume si un id care sa fie unic pentru fiecare utilizator , o clsa Interfata prin intermediul careia sa se poata simula aplicatia si anume de a adauga noi conturi , sa se stearga conturi sau sa se editeze conturile . O alta clasa este clasa Accounts care este o clasa abstracta aceasta avand ca si atribute Person si un un id de tip string si acesta ar trebui sa fie unic deoarece nu pot exista 2 conturi cu acelasi id , un alt atribut al acestei clasa este suma care reprezinta suma de bani pe care o are utilizatorul acestui cont . Urmatoarele clase sunt SavingAcounts si SpendingAcounts acestea mostenind clasa abstracta Acounts si suprasciu metoda depunere . Diferenta dintre aceste doua clase este ca in clasa SavingAcounts suma nu poate sa fie mai mica decat 0 iar in SpendingAcounts aceasta suma poate ajunge pana la -200 . Urmatoarea clasa este clasa Bank in aceasta am implementat un hastable de Acounts . In aceasta clasa se afla metode pentru verificare conditie de pastrare a structurii de date . Pe langa aceste clasa am mai implementat interfata BankProc aceasta avand numa 3 metode si anume : metoda de adaugare a unui nou cont , de editare sau de stergere a unui cont .

## 2.2 Asumptitii

Cum a fost precizat si anterior, modelul de implementare a aplicatiei este inspirat din viata reala . Unitatea de baza a aplicatiei este Account . Descrierea proiectului ne impune sa folosim structura de Hashtable pentru implementarea bancii si sa rezolvam cazul in care doua obiecte au aceeasi cheie ( doua conturi apartin aceeasi persoane ) . Implementarea exacta a cerintei rezulta intr-o structura de forma : HashMap<Person, ArrayList<Account>> , doar ca aceasta mi s-a parut greoaie atat de gandit , cat si de citit / scris cod . Am cautat pe internet structuri de date care sa-mi permita sa implementez cat mai usor si am dat peste clasa HashMultimap . Aceasta clasa imi permite sa indeplinesc cerintele mult mai usor . Scrierea este mai usoara si , avand metode implementate , totul a devenit mai facil si mai clar , tipul de date al bancii fiind HashMultimap<Person, Account> .

## 2.3. Scenarii

In continuare va voi prezenta intreaga lista de clase implementate si voi prezenta rolul acestora in cadrul aplicatiei . Fata de celelalte proiecte , in acesta am studiat mai mult problema structurarii claselor si am folosit pachete pentru a le pastra cat mai ordonat .

Package **bank**

Clasa **Bank** reprezinta banca propriu-zisa in care sunt tinute conturile oamenilor si care se ocupa de gestiunea acestora . Se pot adauga conturi , sterge conturi , stergerea tuturor conturilor unei anumite persoane , adaugarea unei persoane in sistem , stergerea unei persoane ( se sterg automat toate conturile ) si cele doua functii pentru citirea si scrierea bancii in fisier .

Cum s-a cerut in descrierea problemei , o metoda **isWellFormed** a fost implementata , care are rolul de a verifica integritatea bancii ( prin integritate ma refer la corectitudinea de structura a bancii ) si anume : sa nu existe doua persoane cu acelasi ID , sa nu existe doua conturi cu acelasi ID , persoana din cont sa fie aceeasi cu persoana de la cheie .

Clasa **BankManagement** este clasa in care pornim aplicatia .

In interfata **BankProc** definim metodele pe care le vom implementa in **Bank .**

Package **bank.account**

Clasa **Account** este clasa in care definim functiile pe care le folosim in cadrul unui cont si anume de : suplementare a unei sume dintr-un cont , extragere a unei sume .

Clasa **Person** este clasa care contine datele legate de o persoana .

Clasa **SavingAccount** este clasa care se ocupa de acest tip de cont . Aceasta contine o constanta INTEREST\_RATE ( 0.75% )prin care am definit dobanda „ anuala ” a bancii pentru acel tip de cont . Pentru a putea vedea functionarea corecta a aplicatiei , dobanda nu se va aplica anual , si o data la 10 secunde ( interval de timp definit de student ) .

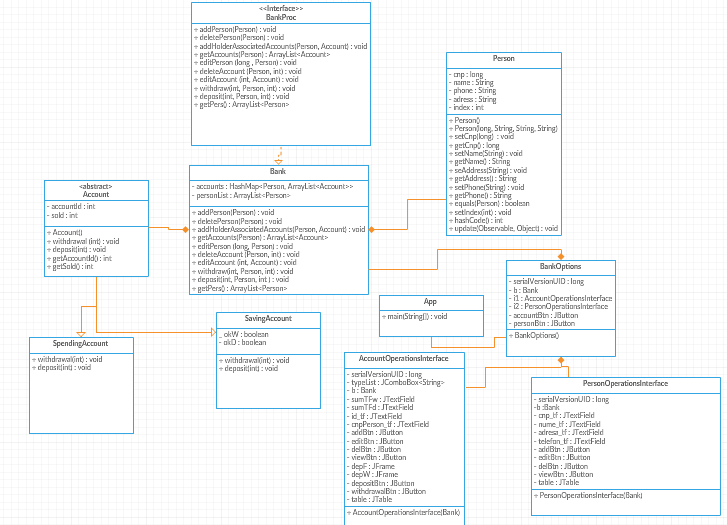
Clasa **SpendingAccount** reprezinta clasa care se ocupa de cel de-al doilea tip de conturi . In aceasta retinem comisionul care se aplica fiecarei tranzactii . Am definit o suma minima care poate fi extrasa si un comision fix de 5$ pentru sume mai mici de 100$ si 5% din suma pentru sume de peste 100$ .

Package **bank.Interface**

In cadrul clasei **Interface** am creat interfata grafica a aplicatiei .

# 3. Proiectare(decizii de proiectare, diagram UML, structure de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor)

Dupa cum am mentionat anterior acest proiect trebuie sa utilizeze cel putin sase clase : : o clasa Person care sa aiba ca atribute un nume si un id , o clasa Interfata prin intermediul careia sa se poata simula aplicatia si anume de a adauga noi conturi , sa se stearga conturi sau sa se editeze conturile si sa se afiseze toate conturile din banca . O alta clasa este clasa Accounts care este o clasa abstracta aceasta avand ca si atribute Person si un un id de tip string si acesta ar trebui sa fie unic deoarece nu pot exista 2 conturi cu acelasi id , un alt atribut al acestei clasa este suma care reprezinta suma de bani pe care o are utilizatorul acestui cont . Urmatoarele clase sunt SavingAcounts si SpendingAcounts acestea mostenind clasa abstracta Acounts si suprasciu metoda depunere . Diferenta dintre aceste doua clase este ca in clasa SavingAcounts suma nu poate sa fie mai mica decat 0 iar in SpendingAcounts aceasta suma poate ajunge pana la -200 . Urmatoarea clasa este clasa Bank in aceasta am implementat un hashtable de Acounts . In aceasta clasa se afla metode pentru verificare conditie de pastrare a structurii de date tot in aceasta clasa se afla metode de incarcare a conturilor dintr-un fisier . Pe langa aceste clasa am mai implementat interfata BankProc aceasta avand numa 3 metode si anume : metoda de adaugare a unui nou cont , de editare sau de stergere a unui cont .

****

In proiectarea acestui sistem de simulare a unei cozi am apelat la tehnica bottom – up . Aceasta tehnica presupune construirea de la inceput a unor elemente primitive ca in exemplu de fata obiectele primitive fiind clasele care au cate un constructor si metode de schimbare a atributelor . Apoi am inceput combinarea acestor clase prin intermediul obiectelor. De exemplu intre clasele Acounts si SavingAcount si SpendingAcounts se alfa relatie de mostenire deoarece aceste doua clase extend clasa Acounts . In programarea aceasta inseamna construirea initiala a unei functii primitive , ce implementeaza elemente atomice de functionalitate , apoi utilizarea lor in construirea de functii mai complexe . Rezultatul este o structura arborescenta care are ca radacina produsul software final .

Pentru contruirea aceste diagram am folosit programul Paint . Din aceasta diagram putem observa relatiile dintre clase . Intrea clasele Interfata si Bank exista relatie de dependent doarece clasa Interfata are ca si atribut un obiect de tip Bank .Tot aceasi relatie se afla intre calseel Person si Accounts si tot din acelasi motiv si anume ca in clasa Acounts se afla un atribut de tip Person . Intre clasele Acounts si Saving Acounts respective SpendingAcounts se afla relatie de mostenire deoarece aceste clase extend clasa Acounts . Clasa Bank implementeaza interfata BankProc , in aceasta interfata avem 3 metode si anume de adaugare cont ,stergere cont si editare cont . Intre clasele Acounts si Bank avem relatie de compozitie deoarece in Bank avem un hashtable de accounts mai bine spus de copii al clasei Acounts deoarece aceasta clasa este abstracta si nu poate sa fie instantiate .

# 4. Implementare

Datorita diagramei UML si a digramei CRC ne este foarte usor de implementat aceasta aplicatie care simuleaza procesarea conturilor la o banca . Implementarea acestei aplicatii am inceput-o cu implementarea clase Interfata . Aceasta clasa are mai multe componente ca JPanel , JButton jLabel JTextarea si JRadioButton . Am folosit un JPanel pe care am pus JButton JTextArea si JRadioButton aceaste component avand rolul de a introduce datele necesare pt crearea unui nou cont sau pentru adaugarea de noi bani .In partea dreapta am un JBapel pe care am pus un JTextArea deoarece aici se afiseaza rezultatele operatiilor effectuate .

In constructorul clasei instantiem componenetele si le adaugam pe JFrame si instantiem si obiectul de tip Bank . Aceasta clasa extinde clasa JFrame cu ajutorul cuvantului extends si implementeaza interfata ActionListener cu ajutorul cuvantului implements . Pentru a putea implementa aceasta interfata clasa Interfata trebuie sa supreascrie metoda actionPerformed (ActionEvent e ) .

Urmatoarea clasa implementata este clasa Bank , aceasta clasa are ca si atribut un vector de tip Nod , aceasta fiind o clasa interioara care are ca si atribute Acounts si pointeri catre urmatorul nod respective precedentul nod . In constructorul acestei clase Bank instantiem vectorul de si apelam metoda de incarcare a conturilor dintru fisier . aceasta metoda se numeste sugestiv incarcare . Aceasta clasa mai are metode pentru a adauga noi conturi , a edita conturi si pentru a sterge conturile . Pentru a putea efectua aceste operatii trebuie sa stim pe ce pozitie se afla fiecare cont , acest lucru putem sa-l aflat cu ajutorul metodei hasFunction care se calculeaza in functie de idContului si apoi se imparte la 11 deoarece acest vector de Noduri are numa dimensiuna de 11 . In aceasta clasa am importat mai multe library deoarece au fost nevoie pentru a citit din fisier , am imporat java.io . Aceasta clasa implementeaza interfata BankProc aceasta interfata avand 3 metode : addAcount editAcount deleteAcount .

Urmatoarea clasa este clasa Acounts aceasta fiind de tip abstract deoarece avem o metoda care nu este implementata si anume metoda depozit . Aceasta clasa are 3 atribute unul de tip Person , unul de tip String care reprezinta id conturlui si unul de tip int care reprezinta suma de bani din acel cont . Aceasta mai are ca si metode de setare a atributelor deoarece trebuie respectate paradigmele POO si anume encapsularea care spune ca atributele sunt de tip private si exista metode publice de set si get pentru aceste atribute .

Clasele SpendingAcounts si SavingAcounts mostenesc clasa Acounts si implementeaza metoda depozit fiecare clasa implementand in felul ei . Una nu lasand ca suma sa fie mai mica de 0 cealalta ne lasand sa fie mai mica de -200 . Pe langa aceasta metoda se afla si constructorii , fiecare constructor apeland constructorul clasei parinte .

# 5. Testare

Implementarea acestui mini-proiect a fost aleasa in felul acesta deoarece este cea mai apropiata de modul in care lucreaza bancile in viata reala . Nu intamplator , acesta este unul dintre cele mai bune moduri de a modela aceasta aplicatie , deoarece , din experienta oamenilor , acestia au dezvoltat in timp un model pentru a rcreste productivitatea si eficienta depozitarii si accesarii conturilor .

Revenind la mini-proiectul prezentat , mi s-a parut o foarte buna aplicatie pentru dobandirea cunostintelor despre crearea bazei unei aplicatii de gestionare a obiectelor ( de orice tip ar fi ele ) .

# 6. Rezultate

In urma simularii aplicatie se observa ca aplicatia decruge fara problem de implementarea in cazul in care utilizatorul aplicatie a introdus correct datele de intrare / Este de observant ca la inceput se afla un numar mic de conturi .

# 7. Cocluzii

Dupa realizarea acestui proiect am invatat conceptual de reflection si totodata mi-am reamintit cum se scrie cod in baze date si cum se realizeaza conectiunea la baze de date.Am invatat sa utilizez JTable in java sis a lucrez cu toate aceste lucruri cu usurinta.

Pentru a vizualiza datele din cadrul bancii am folosit pentru prima data un Jtabel si am fost nevoit sa ma documentez despre utilizarea acestuia .

# 8.Bibliografie

1 . <http://docs.oracle.com/javase/tutorial/>

2 . [www.stackoverflow.com](http://www.stackoverflow.com/)

3. http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/collect/HashMultimap.html