**Documentație**

**Banca**

Student: Costea Ovidiu-Bogdan

Grupa:30229

# 

**Contents**

[**1.Obiectivul temei:** 3](#_Toc512479675)

[**2.Analiza problemei ,modelare,cazuri de utilizare** 3](#_Toc512479676)

[**2.1Analiza problemei** 3](#_Toc512479677)

[**2.2. Modelarea** 3](#_Toc512479678)

[**2.3.Cazuri de utilizare** 3](#_Toc512479679)

[**3.Proiectare** 4](#_Toc512479680)

[**3.1. Structuri de date** 4](#_Toc512479681)

[**3.2. Clase** 5](#_Toc512479682)

[**3.3Algoritmi folosiți** 5](#_Toc512479683)

[**3.4. Pachete** 6](#_Toc512479684)

[**3.5. Interfețe** 6](#_Toc512479685)

[**3.6.Interfața cu utilizatorul** 6](#_Toc512479686)

[**4. Implementare și Testare** 8](#_Toc512479687)

[**5.Rezultat** 10](#_Toc512479688)

[**6.Concluzie** 11](#_Toc512479689)

[**7.Bibliografie** 11](#_Toc512479691)

# **1.Obiectivul temei:**

Principalul scop al acestei teme este realizarea unei aplicati care simuleaza operatiile unei banci.Implementarea se bazeaza pe utilizarea serializari pentru salvarea datelor, dar mai mult decat atat am folosit si design patternul Observer pentru notificarea clientilor asupra modificarilor efectuate asupra conturilor.Proiectul a fost realizat in limbajul de programare Java ,iar ca mediu de programare am folosit IDE-ul Eclipse.

Proiectul pe care deja l-am implementat ofera posibilitatea utilizari atat de catre clienti oferind posibilitatea adaugari drepturilor de administrator, activitatile pe care acestia le pot executa vor fi foarte bine conturate

# **2.Analiza problemei ,modelare,cazuri de utilizare**

# **2.1Analiza problemei**

Dupa o scurta analiza a cerintei obiectivele care trebuie indeplinite sunt foarte bine conturate. Astfel aplicatia trebuie sa permita lucrul cu conturi si persoanele care le detin. Astfel un anumit client are posibilitatea de a detine mai multe conturi . In functie de tipul contului se vor putea face anumite operatii care sa schimbe continutul acestora.Contul va putea fi un cont de economii sau un cont de cheltuieli ,diferenta intre cele doua conturi consta in faptul ca un cont de economi va primi in cazul retrageri totale, nu se poate retrage o suma mai mica decat suma totala, o dobanda de 5%,iar nu cazul conturilor de cheltuieli avem posibilitatea de a face mai multe tranzactii succesive.

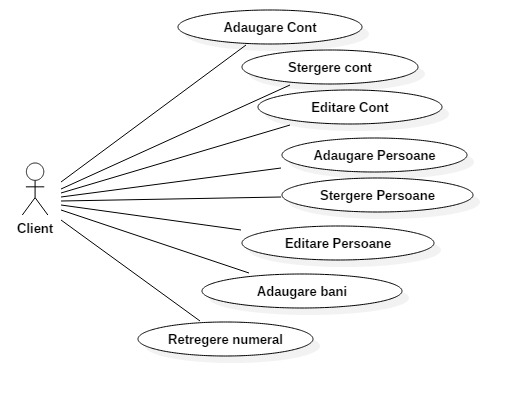
# **2.2. Modelarea**

Pentru implementarea acestui simulator de banca am avut nevoie de mai multe clase si o interfata.In primul rand pentru aveam nevoie de cate o clasa pentru conturi respectiv pentru persoane, deoarece avem doua tipuri de conturi bancare vom avea nevoie de alte doua clase care sa extinda clasa account.Pe langa acesta clase am mai avut folosit clasa bank care se ocupa cu operatiile de baza care pot fi realizate asupra detinatorilor si conturilor detinute de acestia. Si bineinteles de o clasa care va in care se va implementa interfata grafica si alte clase folosite pentru testarea, simularea acestui sistem bancar.

# **2.3.Cazuri de utilizare**

Utilizarea aplicatie este simpla. Odata cu deschiderea aplicatiei vom putea putea selecta una dintre cele trei operatii care sunt disponibile pe primul tab. Respectiv putem visualiza tabelul cu persoane, conturi sau putem creea o noua inregistrare cu un nou client. In cazul in care vizualizam tabelul cu persoane vom putea efectua si anumite operatii asupra acestor persoane respectiv stergerea unei persoane selectate din JTable, asocierea unui nou cont unei persoane, editarea unei persoane, iar in cazul vizualizari tabelei de conturi vom avea si acces asupra unor operatii ( editare ,adaugare bani, retragere de numeral, stergerea unui cont ) .

**Diagrama USE-CASE:**

****

# **3.Proiectare**

Aplicatia a fost dezvoltată printr-o abordare top-down , sistemul fiind proiectat ca un tot, apoi acesta a fost divizat in subsisteme. Clasele au fost realizate intr-un mod abstract, acestea asteptandu-se la un rezultat specific fara a furniza instructiuni de returnare a acelui rezultat. Dupa ce un model acceptabil a fost atins metodele au putut fi definite.

Motivul utilizarii acestei abordari a fost faptul că oferă un foarte bun nivel de abstractizare, acest lucru imbunatățind viteza de dezvoltare a aplicației, un cod mai ușor de întreținut și mai pușin predispus la erori.

# **3.1. Structuri de date**

Datorita numarului de clienti si conturi care poate sa varieze in timp pentru o mai buna utilizare a memoriei si timpului am optat pentru utilizarea unei structuri de date de tip HashMap. Pentru fiecare nou client introdus vom cauta o noua pozitie in tabela in functie de hash codul generat. Odata adaugat in hashMap ii va fi asociat in locul unei valori o lista de conturi destinate retineri acelor conturi detinute de respectiva persoana. Pentru eviatare coliziunilor a fost aplicata metoda inlantuiri, practic atunci cand unei key i se atribuie o noua valoare aceata va fi adaugat in capatul listei ( LIFO – Last In First Out ).

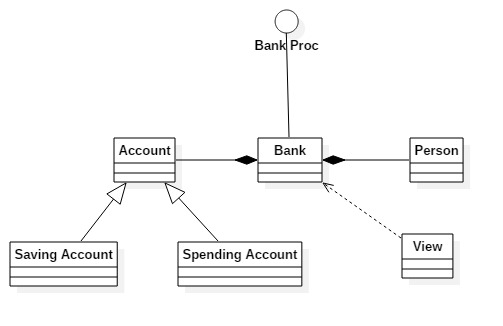
# **3.2. Clase**

Funcționalitatea programului conține următoarele șapte clase. Clasa ”app” reprezintă clasa de unde se poate realiza o scurta simulare, precum si clasa ” TEST\_BANK ” care se va ocupa stric de verificare corectitudinii metodelor implementate in clasa ” Bank ” .

In clasa View se gaseste implementare interfetei grafice pecum si ascultatori pentru fiecare buton in parte.

Clasa ” Account ” folosita in ” Bank ” impreuna cu ” Person ” , este mostenita de catre clasele ” SavingAccount ” si ” SpendingAccount ” va implementa metodele de retragere/adugare de bani. Prin mostenerie se va putea folosi proprietatea de polimorfism asadar ne va usura munca in cadrul proiectului mai ales in utilizare hashMap-ului.

Diagrama UML de clase :

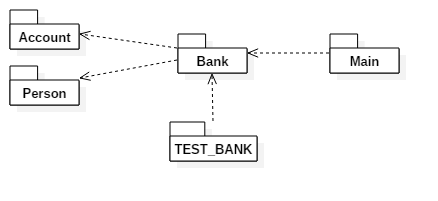


# **3.3Algoritmi folosiți**

In dezvoltarea acestei aplicatii nu am avut nevoie de algoritmi care sa se fie eficienti din punct de vedere al timpului si al memorie ,datorita lucrului cu tabele de dispersie care deja implementeaza metode foarte eficiete din acest punct de vedere. Singurele implementari care au fost facute au fost verificare datelor introduse pentru creearea unei noi persoane sau cont , asigurarea inaintea unei retrageri pentru a fi sigur in privinta sumei extrase .

# **3.4. Pachete**

In acest caz am optat pentru impachetarea claselor in anumite pacheta pentru a oferi o mai buna intelegere a codului . In urma acestei convecti am optat pentru creearea a 5 pachete , dintre care unul destinat doar pentru testare respectiv initializare . Care contin clasele specifice pentru fiecare parte a implementarii. Pachetul Bank contine doar clasa Bank care mapeaza persoanele respectiv conturile , pachetul Account contine clasa Account care va fi extinsa de clasele SavingAccount si SpendingAccount necesare pentru a face diferenta intre tipurile de conturi posibile. Aceste pachete comunica intre ele implentand astfel ideea principala a acestui sistem.



# **3.5. Interfețe**

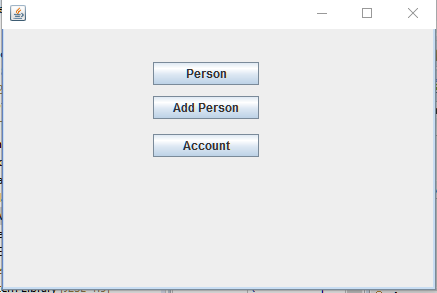
In dezvoltarea acestei aplicati am optat asupra implementarii unei interfata predefinita si uneia definite . Clasa Bank va implementa interfata BankProc in cadrul careia au fost declarate methodele care vor trebui implementate ( adaugar /stergere persoane , adaugare /stergere conturi, editare de conturi si persoane ). De asemenea in cadrul clasei Person a fost implementata interfata Observable pentru a permite folosirea design pathernului Observers. Folosit in pentru a anunta utilizatori asupra operatiilor efectuate in conturi.

# **3.6.Interfața cu utilizatorul**

Interfata grafica este destul de simpla și intuitivă pentru orice utilizator, caruia ii ofera posibilitatea de a creea un nou cont prin simpla selectare a unui titular si introducerea unei cantitati de bani care va fi depusa in momentul creeari. Interfata conține un JtabbedPane unde au fost adaugate mai multe containere Jpanel, utilizarea acestei componente faciliteaza lucrul cu ferestre oferind astfel sansa de a reduce numarul de ferestre utilizate.

Fiecare Jpanel a fost populat la randul lui cu componentele grafice de care aveam nevoie. Tabul pentru creearea unui nou cont a fost populate cu un JtextField, JcheckBox, Jbutton care ii ofera utilizatorul ajutorul necesar pentru adaugarea unui nou cont la o persoana dorita.

Jtabel reprezinta componenta pe care pana acum un am avut oprtunitatea sa o folosec. Acesta piesa mi-a permis prin simplele sale metode sa afisez continutul preluat din structura de date hashMap. Odata ce o noua componenta a fost creata avem posibilitatea sa-i asignam un tablou bidimesional de tipul Object in care sunt stocate informatiile din structura ,precum si numele fiecarei coloane in parte printr-un vector de tipul Object unde fiecare camp a fost asignat cu cate un nume ce reprezinta variabilele claselor.



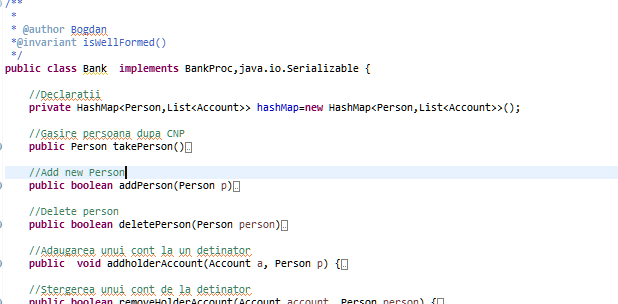
In imagina de mai sus este prezentata pagina principala care prin apasarea unui din cele 3 butoane ne va duce la pagina dorita. In cazul in care nu suntem inregistrat la aceasta banca vom avea sansa de a creea o noua inregistrare prin simpla apasare a Jbutton intitulat ”Add Person” caruia i-a fost assignat un acultator de mouse, iar atunci cand vom da click pe el ne va duce la un alt tab. Asa cum am mentionat si mai sus celelalte doua butoane vor afisa persoanele si conturile inregistrate in fisierul bank.ser.Aici exista si metoda isWellFormed , care verifica daca datele existente in banca au un format corespunzator si verifiva toate conditiile ca datele sa fie valide si logice.

Bank implementeaza interfata BankProc unde au fost definite metodele care ar trebui implementate aici, totodata aici se realizeza si contractul cu clasa care o implementeaza. Sunt descrise in format Java doc. BankProc reprezinta are si rolul de a realiza contracte cu clasa Bank care implementeaza acesta interfata. Sunt descrise doar antetetele functiilor si sunt specificare comentarii care corespund principiilor design by contract in care specificam ce valori ce ar trebui sa fie primite de catre ficare functie , iar rezultatul se va incerca sa fie un rezultat grantat. Clasa isWellFormed reprezinta invariantul acestei interfete in care se va verifica atat la inceputul metodei cat si la sfarsitul acesteie conditiile care atesta ca datele deja introduse sunt corecte din anumite puncta de vedere , cum ar fi lungimea fiecarei liste corespunzatoar fiecarei key in parte.In cazul in care una dintre aceste verificari va esua in consola ne va aparea anumite mesaje de erroare care sugereaza acest lucru.

# **4. Implementare și Testare**

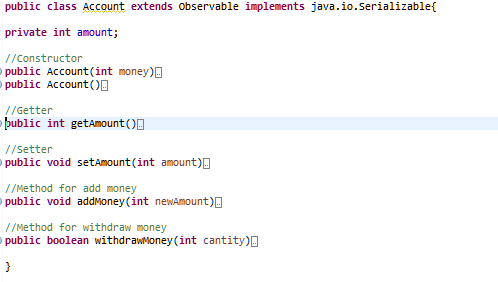
Programul a fost scris prin implementarea claselor deja menționate. În fiecare clasă au fost implementate metode folosind algoritmi deja descriși mai sus, acestea au fost declarate public ,iar campurile private fiind acesibile doar cu ajutorul getterelor si setterelor pentru a aigura permisiunile la aceste date.

**Clasa Bank:**

 Clasa Bank apartine pachetului .Principala utilizare acestei clase a fost maparea tabelei de dispersie ,prin. Cu ajutorul tehnicii de serializare aceste date din tabela de dispersei au fost convertite intr-un fisier de biti in care fiecare obiect este delitat de celalalt. Serializarea este o technical utilizata pentru salvarea datelor dintr-o aplicatie asemeni bazelor de date. Pentru a prelua pachetele de informatii deja salvate a folosit deseliazarea.

**Clasa Account:**

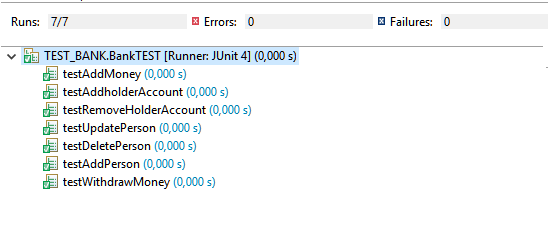
Clasa account este clasa care va fi extinsa de catre SavingAccount si SpendingAccount. Acestea implementea operatiile de baza care se poy executa asupra conturilor : intrpducerea unei noi cantitati de bani , retragerea de numeral , interogarea soldului



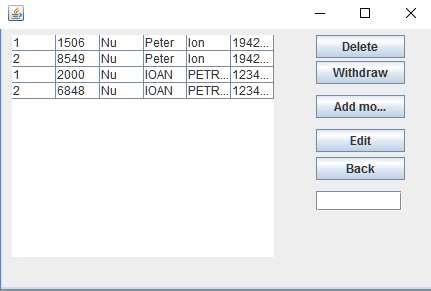
Extendirea clasei Observer ajuta la implemetarea design paternului Observer. Acesta se foloseste in cazul in care mai multe clase(*observatori*) depind de comportamentul unei alte clase(*subiect*), in situatii de tipul: o clasa implementeaza / reprezinta logica, componenta de baza, iar alte clase doar folosesc rezultate ale acesteia (monitorizare) , o clasa efectueaza actiuni care apoi pot fi reprezentate in mai multe feluri de catre alte clase (view-uri ca in figura de mai jos) . Practic in toate aceste situatii clasele Observer  **observa** modificarile/ actiunile clasei Subject . Observarea se implementeaza prin **notificari initiate din metodele clasei Subject**

# **5.Rezultat**

O serie de teste au fost facute pentru a asigura o bună comportare a acestei aplicații. În cadrul clasei Test\_Bank au fost declarate și instantiate variabile de tipul cont si person . Rezultatele metodei assert sunt vizibile, dacă afirmația nu este adevarată testul va fi considerat eșuat altfel va fi considerat un success. După terminarea unui test se reiau ințializare și toate celelalte teste.



Conturile deja create pana la momentul de fata vor fi vizibile prin simpla apasarea a butonului Account care va schimba tabul si va initializa JTable- ul cu informatiile preluate dupa deserializare. Odentic cu aceasta operatie se va face sin in cazul tabelei de personae.



# **6.Concluzie**

# Acest proiect reprezinta daca as putea sa-l numesc machete unei aplicati de care permite realizarea unor operatii traditionale la o banca de catre un client.Astfel vom avea sansa de a interoga soldul, adauga numeral, retrage o parte din suma existent deja intr-un cont . Exista posibilitatea extiderii acesteia prin adaugarea unor poze fiecarei persoane in parte.O alta imbanatatire care ar putea fi adusa este extinderea aplicatiei cu un chat in care clientul sa poata contacta direct pe cei de la banca , avand astfel sansa de-a le adresa intrebari primind raspunsuri in timp real. Am putea oferi utilizatorilor posibilitatea de asi exprima parerea asupra produselor oferite de acesta banca si viitoarele imbunatatiri care ar putea fi aduse in cadrul ei.

# **7.Bibliografie**

https://stack overf low.co m

https://www.y out ube.co m /

htt ps:// www.tut oria lspoint. Co m/