

**FACULTATEA: Automatică şi Calculatoare**

**SPECIALIZAREA: Calculatoare şi tehnologia informaţiei**

**DISCIPLINA: Tehnici de Programare**

**Bank Managment**

**Îndrumător Laborator: Realizator:**

Cristina Bianca Pop Oneţiu Alexandru Lucian

Grupa 30228

***Cuprins***

1. Cerinte Functionale

2. Analiza Problemei

3. Obiective

4. Proiectare

4.1 Diagrama de utilizare

4.2 Diagrama de clase

5. Implementare

6. Concluzii si Dezvoltari Ulterioare

7. Bibliografie

# Cerinte Functionale

**Cerinta:**

Propuneti, proiectati si implementati o apalicatie de gestiune a datelor bancare.

**Consideratii generale**

Problema propusa este una extrem de populara .Aplicatiile pentru managementul bazelor de date sunt realizate in general in Java , fiind folosite pe scara larga : aproape fiecare afacere are o baza de date pentru angajati , clienti , produse , etc. Privind problema din punctul de vedere a logicii folosite in spatele aplicatiei , aceasta este de dificultate medie . Provocarea vine defapt de la nevoia de a face clasele si metodele cat mai reutilizabile cu putinta .

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 .

# Analiza Problemei

Tema are ca obiectiv implementarea unui aplicatii pentru procesarea conturilor bancare. Implementarea presupune crearea de conturi, care pot fi de tipul spending account sau saving account. In functie de optiunea selectata a contului care urmeaza sa fie creat , vom putea adauga sau retrage bani din cont . De asemenea mai exista si posibilitatea stergerii unui cont , precum si o optiune prin care se realizeaza afisarea conturilor existente . 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 si post conditiii . Cerinta acestui proiect consta in implementarea unei aplicatii care sa simuleze o aplicatie bazata pe procesarea conturilor in cadrul unei banci , pornind de la o diagrama de clasa. Pe baza acestei diagrame de clase data sunt date si cateva cerinte: in cadrul interfetei BankProc se vor implementa pre conditiile si post conditiile . De asemenea se cere sa sa implementeze clasele Person , Account , SavingAccount , SpendingAccount si alte clase care raman la decizia celui care realizeaza implementarea . Clasa Bank se cere sa se implementeze prin intermediul unui hashtable si de asemenea se cer sa se realizeze si testele corespunzatoare pentru metodele implementate .

Acest sistem are o raspandire foarte larga in viata de zi cu zi . In cadrul tuturor bancilor se gaseste o asemenea interfata pentru editarea conturilor si pentru gasirea conturilor clientilor . Aceasta aplicatie este cu atat mai utila cu cat cumparaturile online au crescut considerabil in ultimii ani , plata realizandu-se prin intermediul unei banci .

Se impune gasirea preconditiilor si a postconditiilor care trebuie puse . Acestea au fost adaugate in cadrul clasei Bank , intrucat aici se afla metodele care realizeaza procesarea conturilor.

# Obiective

Dupa citirea problemei se poate observa ca putem avea cel putin 6 clase : o clasa Person care sa aiba ca atribute un email,nume si un id care sa fie unic pentru fiecare utilizator. O alta clasa este clasa Accounts care are ca si atribute Person si un un id de tip int 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 Acounts cu noi constructorii. 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 tot pana la 0. Urmatoarea clasa este clasa Bank in aceasta am implementat un hastabble 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 .

Modul de utilizare este urmatorul in cazul in care utilizatorul vrea sa introduca o noua comanda atunci acesta este nevoit sa completeze toate campurile si sa apese butonul de creare . Pentru a putea vizualiza conturile atunci acesta este nevoit sa apese buton de afisare si acestea conturi apar in partea dreapta .

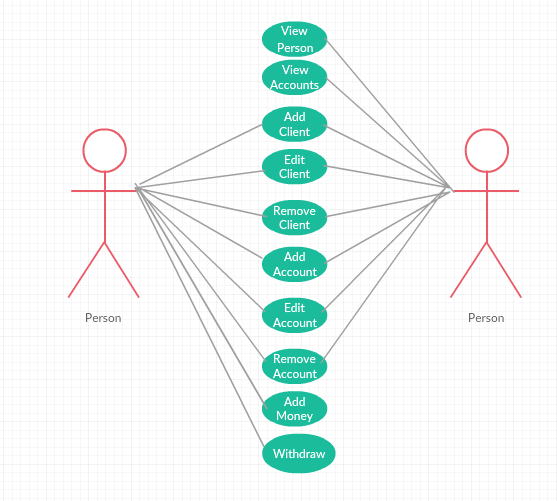
In cazul in care utilizatorul introduce o date gresite sau nu completeaza un camp care trebuie completat atunci programul da un mesaj cu o eroare si nu mai functioneaza.

# Proiectare

**4.1 Diagrama de utilizare**

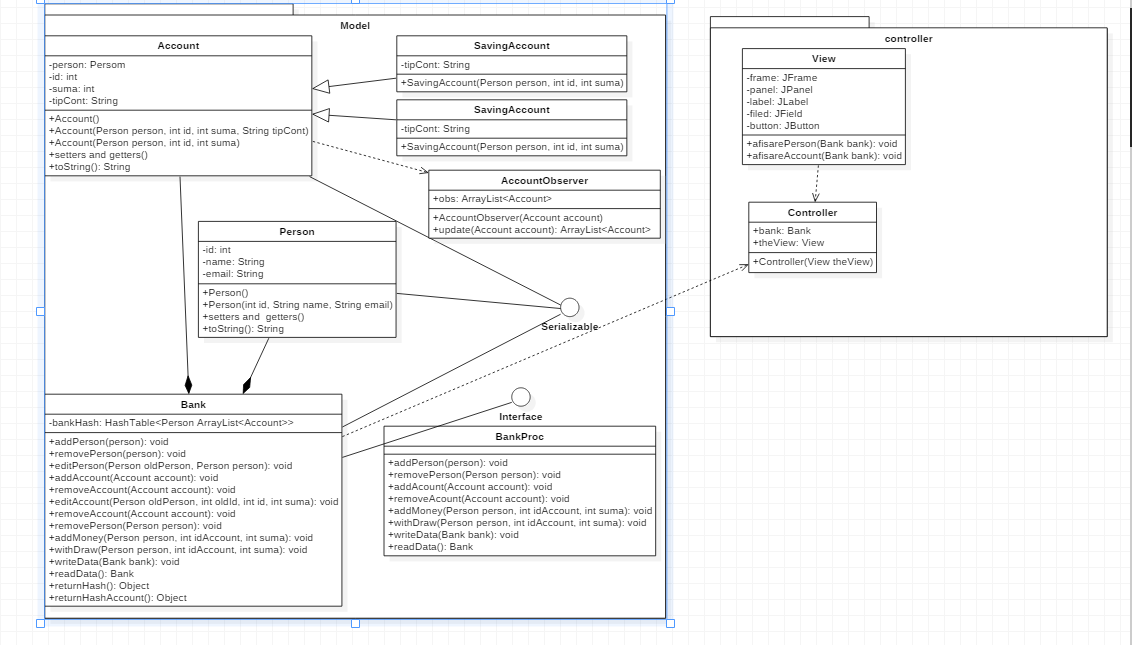
Unified Modeling Language (prescurtat UML) este un limbaj standard pentru descrierea de [modele](https://ro.wikipedia.org/w/index.php?title=Modelare_orientat%C4%83_pe_obiect&action=edit&redlink=1) și [specificații](https://ro.wikipedia.org/w/index.php?title=Specifica%C8%9Bie&action=edit&redlink=1) pentru [software](https://ro.wikipedia.org/wiki/Software) , in programarea POO . Acestea ne ofera o intelegere mai usoara a modelarii problemei , a felului in care clasele se raporteaza unele la altele , si a felului in care principiile POO sunt aplicate . Evident , nu toate metodele si toate atributele sunt reprezentate pentru fiecare clasa . Dimpotriva , am ales sa “ ascund ” unele atribute sau metode , mai putin relevante pentru imagine de ansamblu a problemei , tocmai pentru ca relatiile stabilite intre clase sa fie cele evidentiate .

Diagrama *use case* permite prezentarea posibilităților pe care utilizatorul le are atunci când folosește aplicația. Se distinge o singura operație care poate fi executata de utilizator, deoarece aplicația are un rol de simulare demonstrativa. Diagrama este exemplificată în schema următoare.

****

**4.1 Diagrama de clase**

Diagrama de clasă prezintă principalele clase necesare pentru realizarea aplicației, fiecare element al diagramei având numele clasei, variabilele de clasă, cât și metodele aferente clasei respective. În plus, se conturează relațiile dintre clase (de dependență, de agregare), cât și vizibilitatea datelor și a metodelor (publice, private sau protejate). Pentru a nu supraaglomera schema, în diagramă au fost menționate doar datele definitorii ale claselor. Pentru o perspectivă completă asupra proiectului, se vor vizualiza sursele cu cod. Diagrama de clasă este reprezentată prin schema următoare.

****

# Implementare

**3.3 Proiectare Clase si Pachete**

**Pachete:**

Intregul proiect este impartit intr-o serie de pachete , fiecare continand clase ce implementeaza anumite functionalitati . Dupa tipul claselor continute , se disting urmatoarele pachete :model, controller.

**Clasa Person**

Clasa Person reprezintă modelul unui client din lumea reală, după procesul de abstractizare. Încheiem cu primul nume, și id-ul , apoi adresa . ID-ul este folosit ca un identificator la stocarea intrărilor în tabelul hash. Deoarece am suprasolicitat metoda equals, suprapunem si metoda hashCode pentru a pastra consistenta abordarilor hashing . Mai mult, această clasă implementează serializabil pentru a putea salva prin serializare și să-l încarce după aceea .

**Clasa Account :**

Clasa de cont reprezintă modelarea unui cont real care poate fi deținut de o persoană. Acesta este identificat de un camp ce pastreaza un numar intreg ( id ) pentru identificare unică. De asemenea, deține, evident, suma curentă a clientului. Mai mult, este că vine în două tipuri, date de subclasele sale : Contul de economisire și contul de cheltuieli . Deoarece este o clasă abstractă, ea nu poate fi instanțiată . Deoarece vrem să putem încărca o descărcare dintr-un fișier , implementează serializabil .

Subclasele sale oferă un comportament specializat, în special clasa SavingAccount. Aceasta permite o singură depunere și retragere, în timp ce se calculează o dobanda în pentru contul care este activ.

Clasa SpendingAccount permite multiple retrageri și depozite, dar nu calculează nicio dobanda pentru titular. Este în esență versiunea instanțiată a clasei părinte. Aceste două subclase pot fi , de asemenea , serializate și deserializate folosind mecanismul furnizat de JAVA.

**Clasa Bank :**

Clasa Bank este entitatea centrală a proiectului. Toate operațiunile efectuate de client prin intermediul interfeței sunt orientate spre bancă și conținutul acesteia. Este în relația Compoziție cu clasele Client și Cont. Înregistrările acestor două tipuri sunt stocate în Map despre care am vorbit mai devreme . Această clasă implementează, de asemenea, interfața BankInterface, care oferă anteturile principalelor metode utilizate de bancă. Acestea sunt, de asemenea, implementate prin respectarea specificațiilor furnizate în interfață . Avem următoarele tipuri:

• Adăugarea de clienți și conturi. Acestea se fac fără prea mult efort.

• Eliminați clienții și conturile. Acestea se fac, de asemenea, fără eforturi mari. Cu toate acestea, la eliminarea unui client, eliminăm și conturile acestuia.

• Obțineți toate conturile și clienții. Acestea implică utilizarea hash-ului.

• Retrage și depune bani. Acestea identifică titularul contului care dorește să efectueze operațiunea respectiva si respectivul cont. Apoi vom apela metodele implicite pentru conturile specifice.

• Metode de descărcare si metode de încărcare a datelor , utilizate pentru serializare. Acestea implică utilizarea obiectelor ObjectOutput și Input pentru a codifica și a decoda conținutul obiectelor.

**Clasele UI:**

Clasa CustomTableModel este importantă pentru aspectele legate de GUI. Moșteneste clasa AbstractTableModel, care este utilizată în mod implicit de JTables în Java.

Vorbind despre algoritmi, nu există aspecte provocatoare. Trebuie remarcat că, chiar dacă putem actualiza conturile și clienții, folosim combinații ale metodelor existente .

**4.Pachete:**

Intregul proiect este împărțit în mai multe pachete, care reflectă întinderea aplicației. Avem cel mai mic nivel, care reprezintă entitățile proiectului, și anume banca, clientul și contul și subclasele sale.

Deasupra se află pachetul de controler care deține logica necesară pentru conectarea diferitelor componente și straturi pentru aplicație. În cele din urmă, avem pachetul de vederi care este responsabil pentru toate componentele legate de GUI. Celelalte pachete conțin clase necesare pentru lucruri auxiliare, cum ar fi modelul de masă.

**5. Interfata Grafica**

Proiectul dispune de o interfata grafica care facilitaeaza comunicarea dintre aplicatie si utilizator . Aceasta prevede 2 moduri de utilizare a aplicatiei : ca si client , respectiv ca si administrator . Este deschisa mai intai o fereastra care contine 2 butoane ce permit utilizatorului sa aleaga modul in care doreste sa foloseasca aplicatia printr-un click pe butonul Admin respectiv Client .

Pentru optiunea Client este deschisa o noua fereastra care permite logarea clientului pe baza id-ului , urmand ca o noua fereastra sa se deschida daca id-ul de logare este unul valid . In caz contrar, un mesaj de eroare este afisat . Urmatoarea fereastra deschisa este cea in care se regaseste tabelul pentru produse , din care clientul isi va alege produsul dorit , printr-un dublu click pe produs, in tabel si va introduce cantitatea dorita intr-un text field . Odata stabilita cantitatea, produsul poate sa fieadaugat in cos si mai apoi cumparat . Un mesaj de eroare va fi afisat daca cantitatea selectata este mai mare decat stocul produsului respectiv . O chitanta este apoi creata la terminarea cumparaturilor, ce contine produsele cumparate, cantitatea din fiecare produs si pretul total .

Pentru optiunea Admin este deschisa o fereastra ce contine trei butoane, prin selectarea carora putem sa manageriem tabelul de produse , pe cel de clienti , sau sa vizualizam situatia comenzilor . Tabelul de pentru produse permite editarea, seectarea, stergerea sau inserarea unui produs . Printr-un clikc pe randul corespunzator produsului respectiv acesta este selectat, urmand ca, campurile sale sa fie introduse in 3 campuri de text ce pot sa fie editate . Similar, telelul de clienti poate sa fie editat . O particularitate reprezinta tabelul de comenzi ( OrderManagement ) , care nu suporta operatii directe de editare , ci este actualizat odata cu crearea unei noi comenzi .

**6.Testare si rezultate**

Testarea acestei aplicatii este relativ simpla.

Mai jos, sunt furnizate câteva cazuri de utilizare și instrucțiuni pentru utilizarea interfeței.

Pentru client, este ușor să vizualizați conturile dupa identificare. Ulterior, pur și simplu alegem contul și setam suma dorită pentru a fi plasate sau retrase din cont și apăsaam corespunzător butonul.

De asemenea, clienții sunt avertizați atunci când o schimbare are loc în contul lor prin primirea unui mesaj cu schimbarea și contul în cauză.

Pentru manageri, avem mai multe opțiuni disponibile și, prin urmare, mai multe butoane.

1. **Concluzii si Dezvoltari Ulterioare**

In concluzie, putem spune ca am reusit sa implememtam o aplicatie pentru procesarea de comenzi .Aceasta aplicatie este usor de utilizat datorita interfetei accesibile. In urma realizarii acestui proiect, am invatat o multitudine de lucruri noi : folosirea unui design pattern n-tier, impartirea claselor in pachete in functie de functionalitate, realizarea conexiunii dintre o baza de date si Java, scrierea unor interogari SQL in Java si trimiterea acestora spre baza de date in scopul executarii lor, realizarea unei interfete grafice mai complexa decat cele de pana acum.

**Dezvlotari ulterioare:** S-ar putea adauga implementarea unei interfete de Login cu username si password, precum si o interfata grafica mai complexa, implementarea de oferte, vizulizare profitului magazinului etc.

# Bibliografie

**Java de la 0 la expert ( editia a II -a ) , Stefan Tanasa , Stefan Andrei , Cristian Olaru , Editura : Polirom**

**Cursul de Tehnici de programare al domnului profesor Ioan Salomie**

**Forum-uri.**

**Tutoriale Youtube**

**Cursul domnului profesor Marius Joldos**

**http://docs.oracle.com/javase/7/docs/api/**

**http://stackoverflow.com/**