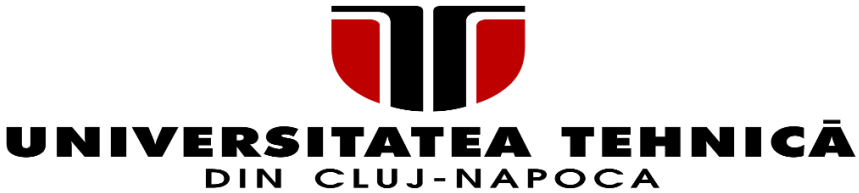



# Sistem de management bancar

Student: Florea Gabriel-Alin

Grupa: 30226

Profesor: Pop Cristina

Profesor laborator: Moldovan Dorin

# Continut documentatie

# 1. Obiectivul temei

Tema curenta pune in discutie implementarea unui sistem de management bancar, ce permite o diversitate de operatii bancare: adaugare perosana in banca, adaugare cont pentru o persoana existenta sau nou venita, retragere de bani, depunere, cat si diverse tipuri de conturi: de economii si de consum. Printre principalele obiective ale temei se numara utilizarea design pattern-ului Design by Contract, implicand utilizarea preconditiilor, post conditiilor, invariantilor, cat si a instructiunilor de tip assert, si de asemenea, un alt obiectiv important este utilizarea design pattern-ului Model-View-Controller, respectiv utilizarea serializarii pentru stocarea datelor intr-un fisier, in mod asemanator stocarii intr-o baza de date.

# 2. Analiza problemei puse in discutie

## a. Asumptii

Aplicatia de fata este o aplicatie construita intr-o maniera destul de defensiva, prin utilizarea preconditiilor, postconditiilor si a invariantilor impreuna cu instructiuni de tip assert. Acest lucru protejeaza aplicatia de utilizarea unor date incorecte, cum ar fi sume de bani negative, sau persoane cu nume nul, sau id-uri egale cu zero sau alte asemenea lucruri. Totusi, in aplicatia de fata se presupune ca utilizatorul cunoaste modul in care aceasta trebuie utilizata, aplicatia neavand mecanisme de protectie impotriva unei utilizari necorespunzatoare, si nici impotriva introducerii unor date aberante. De asemenea in cadrul aplicatiei nu se poate adauga un cont unei persoane care nu exista inregistrata in banca. Pentru a adauga o noua persoana cu un nou cont, trebuie mai intai adaugata persoana, iar abia ulterior se poate adauga contul.

## b. Modelare

In modelarea acestei aplicatii au fost utilizate mai multe structuri de date. Dintre acestea, poate cea mai importanta structura de date utilizata a fost Hashtable<Person, ArrayList<Account>>. Avand o clasa care modeleaza entitatea persoana, respectiv o clasa abstracta care modeleaza entitatea cont, hashtable-ul retine o lista de conturi pentru fiecare persoana. Avantajul utilizarii acestei structuri consta in accesul in timp constant la lista de conturi a fiecarui client, deci

o mare rapiditate. In modelarea aplicatie s-a considerat faptul ca o persoana poate avea zero, unu, sau mai multe conturi la banca.

In alegerea modului de proiectare a aplicatie s-a tinut de asemena cont de cele doua tipuri de conturi posibile la o banca, si anume cont de economii si cont curent. Astfel clasa Account este extinsa de clasele SavingAccount si SpendingAccount. Conturile de economii permit in cadrul aplicatie o singura depunere si respectiv o singura retragere de bani si furnizeaza o dobanda pe baza perioadei pe care este realizat depozitul. Din motive de simplificare a aplicatiei, se presupune ca persoana retrage bani din contul de economii doar in momentul in care numarul de zile pe care a fost realizat depozitul s-a scurs. Contul curent permite mai multe depuneri si retrageri in cont, dar nu furnizeaza nici o dobanda.

### c. Utilizare

Aplicatia de fata doreste sa simuleze o banca, motiv pentru care realizeaza principalele operatiuni pe care o banca le face in mod real. Aplicatia este creata intr-un mod intuitiva si usor de utilizat de catre oricine. La lansarea aplicatie, utilizatorul are un meniu principal din care poate sa aleaga cu ce doreste sa opereze (persoane sau conturi). Pentru fiecare din acestea, utilizatorul poate alege sa vizualizeze ce entitati exista deja salvate in banca, sa adauge, sa sterga sau sa editeze anumite caracterisitici ale acestora. Pentru cont, utilizatorul poate alege de asemenea si operatii de depunere si retragere de bani. Utilizatorul poate efectua oricate operatiuni doreste in aplicatie, cu conditia respecatarii unor minime reguli de utilizare.

Aplicatia curenta isi poate gasi utilitatea practica in zona de cercetare a modului in care operatiunile bancare au loc, deoarece permite simulari diverse ce pot duce la intelegerea si implementarea unor mai bune metode de interactiune persoana-banca in lumea reala.

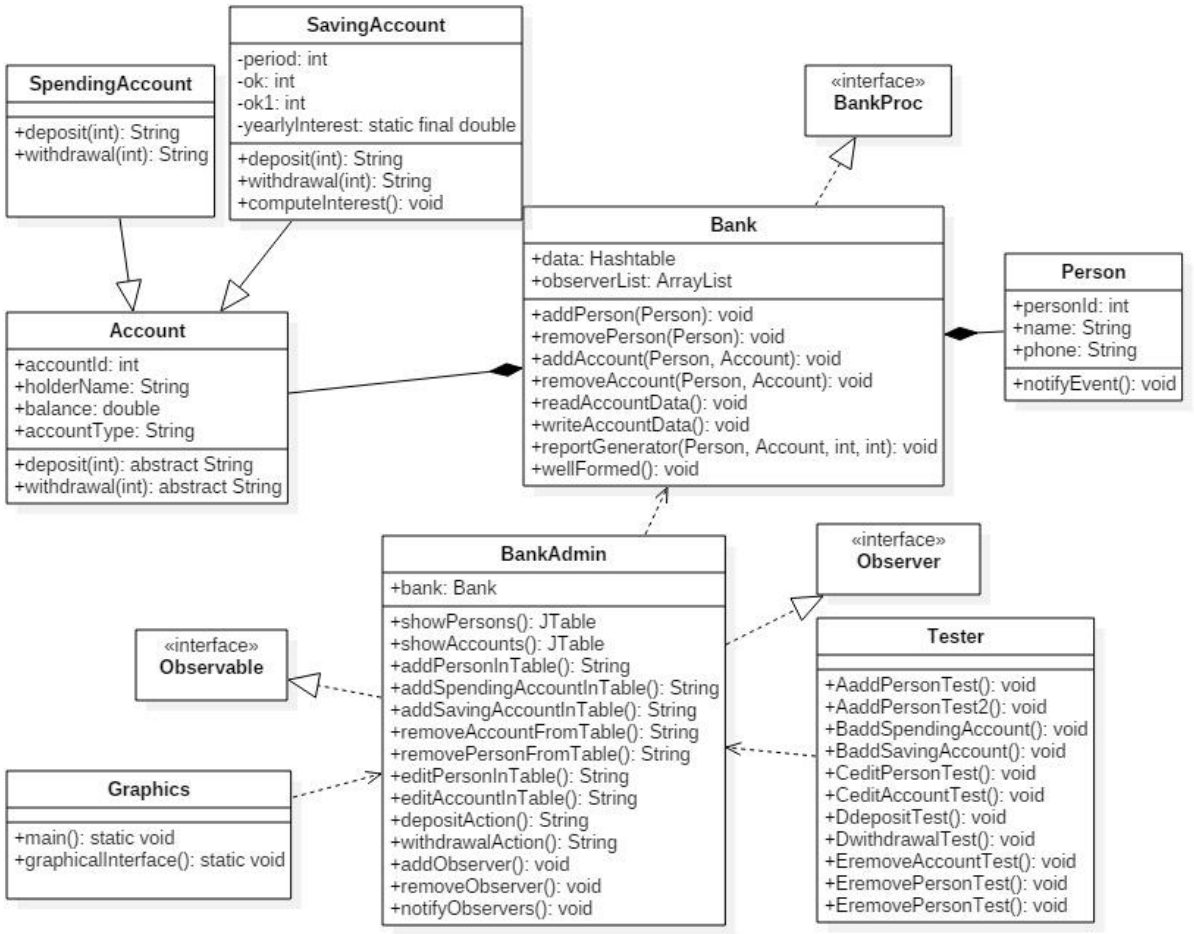
## 3. Detalii de proiectare

### a. Decizii de proiectare

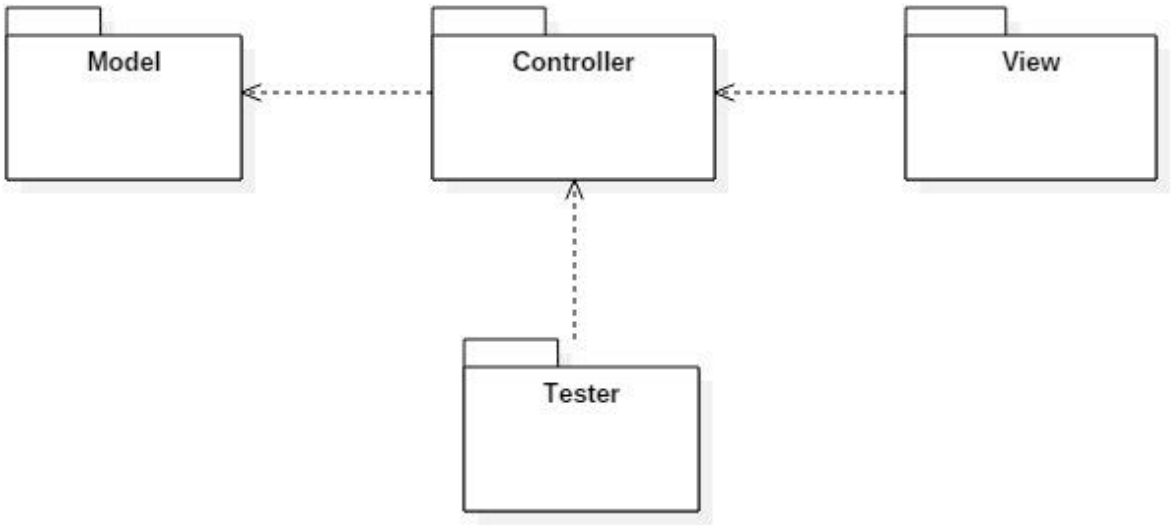
In ceea ce priveste deciziile de proiectare luate, proiectul de fata se constituie din 8 clase si 3 interfete, integrate in 4 pachete. Fiecare din clasele din care este compus proiectul: Person, Account, SavingAccount, SpendingAccout,

Bank, BankAdmin, Graphics si Tester au un rol deosebit de important in cadrul acestuia. Interfetele definite in acest proiect sunt: Observer, Observable si BankProc. Pachetele care compun acest proiect sunt: model, view, controller si tester. Pachetul model cuprinde clasele: Account, Person, SpendingAccount si SavingAccount, respective interfetele Observable si Observer, pachetul view cuprinde clasa Graphics, pachetul controller cuprinde clasele: Bank si BankAdmin, respectiv interfata BankProc, iar pachetul tester contine clasa Tester. Prin intermediul acestui mod de organizare a proiectului si prin deciziile de proiectare luate in cadrul acestui proiect, acesta are o structura clara, omogena si compacta, usor de inteles si de dezvoltat in aplicatii mai complexe.

## b. Diagrame UML



Aceasta diagrama UML prezinta clasele din care este compus proiectul, impreuna cu campurile si cu metodele acestora, cat si modul in care acestea sunt legate intre ele. Printr-o atenta analiza a acestei diagrame se poate observa modul in care programul este structurat si se poate intui functionarea aceastuia.




Aceasta diagrama UML de pachete evidentiaza modul in care pachetele din care este compus programul acesta sunt interconectate, si respectiv modul in care acestea se relationeaza.

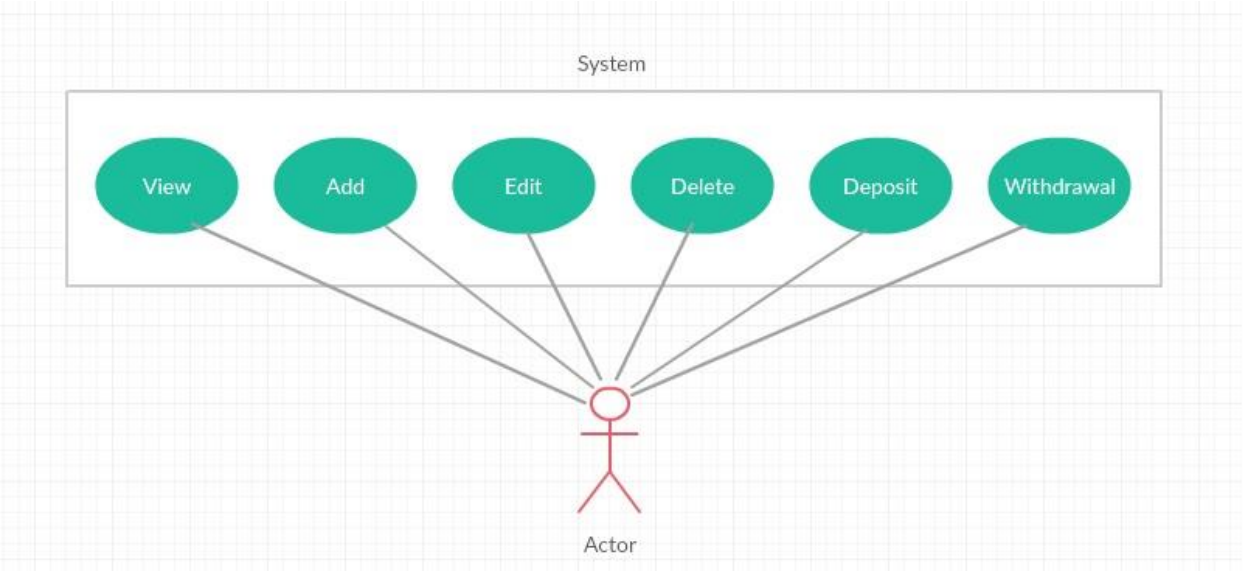


Diagrama cazurilor de utilizare, prezentata mai sus, permite observarea posibilitatilor de intercatiune pe care utilizatorul le are cu sistemul programului. Aceasta diagrama faciliteaza modul de intelegere a functionarii programului.

## c. Structuri de date utilizate

Pentru a putea realiza intr-un mod corespunzator cerinta pusa in discutie, am utilizat un hashtable, mai exact un Hashtable<Person, ArrayList<Account>>, cu ajutorul caruia, pot retine intr-un mod eficient conturile detinute de fiecare din persoane unei banci, avand timp de acces constant la date. Deoarece o persoana poate avea mai multe conturi la banca, pentru fiecare persoana am asociat un ArrayList de conturi. Fiecare din entitatile stocate (persoana si contul) detin anumite campuri care le caracterizeaza (id, nume, etc). De asemenea pentru a putea manipula persoanele care se afla inregistrate in banca am utilizat si o structura de tipul Set. Pentru a putea implementa design pattern-ul observer am utilizat de asemenea un ArrayList in care retin observerii. In realizarea afisarii in JTable, am utilizat tehnici de reflection, utilizand strucuturile necesare implementarii acestor tehnici, clasa Object si metodele asociate, respectiv structuri de tipul Vector. In rest pe parcursul proiectului s-au utilizat doar structuri si tipuri de date clasice.

## d. Proiectarea claselor si interfetelor

Daca e sa luam spre o analiza mai atenta clasele si interfetele din care este constituit acest proiect, atunci se pot afirma urmatoarele:

- Clasa Person defineste structura de persoana cu id, nume si telefon si prezinta o singura metoda pentru notificarea persoanelor necesara in utilizarea design pattern-ului Observer.
- Clasa Account este o clasa abstracta, care defineste notiunea de cont cu principalele caracteristici id, numele detinatorului, suma de bani, respectiv tipul contului. Aceasta clasa prezinta doua metode abstracte pentru depunere si retragere de bani.
- Clasa SavingAccount este o clasa ce extinde clasa Account, avand in plus o variabila pentru perioada depozitului, doua variabile de validare a unor cerinte, respective o constanta ce prezinta dobanda anuala a unui depozit. Aceasta clasa implementeaza metodele de depundere,

respectiv retragere de bani si in plus contine o metoda de calculare a dobanzi pe baza perioadei depozitului, a dobanzi anuale si a sumei de bani din depozit.

- Clasa SpendingAccount este o clasa ce extinde clasa Account, implementand metodele de depunere si retragere de bani.
- Interfata Observer contine o singura definitie de metoda, si anume pentru metoda notify implementata in clasa Person.
- Interfata Observable, adesea intalnita si sub numele de Subject, contine definitia pentru metodele de addObserver, removeObserver si notifyObservers, care sunt implementate in clasa BankAdmin.
- Clasa Graphics este o clasa ce realizeaza interfata grafica a programului, si contine doua metode (metoda main necesara oricarui program Java, respectiv metoda graphicalInterface, care se ocupa propriu-zis de partea de grafica).
- Clasa Tester este o clasa care realizeaza testarea unitara a programului si contine metode pentru testarea diverselor functionalitati ale programului, testarea bazandu-se pe compararea unor mesaje furnizate de program cu anumite mesaje hard-codate (mesajul de “Success!”).
- Interfata BankProc contine definitia pentru metodele necesare functionarii unei aplicatii bancare, si anume: addPerson, removePerson, addAccount, removeAccount, readAccountData, writeAccountData, reportGenerator si wellFormed, care sunt implementate de clasa Bank.
- Clasa Bank este o clasa care realizeaza o parte substantiala din managementul aplicatie bancare, implementand metodele interfetei BankProc, iar ca si atribute continand o lista de observeri si respectiv un Hashtable de perosane si liste de conturi, structuri foarte potrivite pentru retinerea unor asemenea date importante, permitand acces in timp constant la informatiile necesare.
- Calsa BankAdmin este o clasa care mediaza intre clasele de test si de grafica si respectiv clasa Bank, avand si o parte de contributie la managementul aplicatie. Aceasta contine ca atribut un obiect instanta a clasei Bank, iar ca metode implementeaza: showPersons, showAccounts, showingTable, addPersonInTable, addSpendingAccountInTable, addSavingAccountInTable, removeAccountFromTable, removePersonFromTable, editPersonInTable, editAccountInTable,

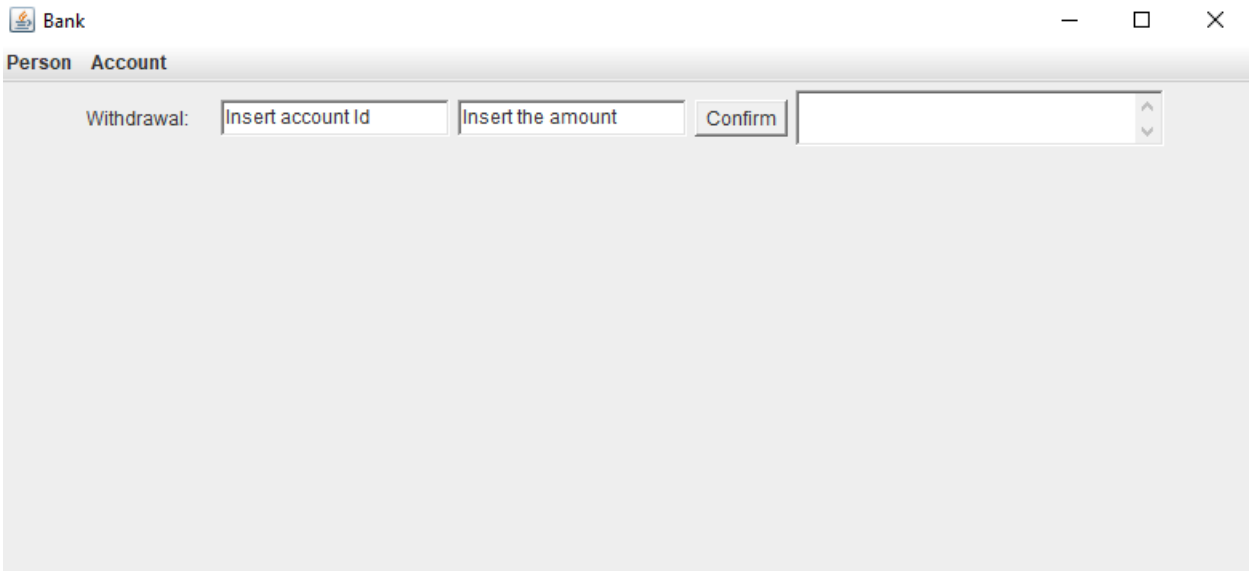
depositAction, withdrawalAction, respectiv implementeaza metodele definite in interfata Observable: addObserver, removeObserver, notifyObservers.

## e. Pachete utilizate

In ceea ce priveste pachetele utilizate in acest proiect, acestea sunt in numar de patru, avand urmatoarea semnificatie:

- Pachetul view prezinta clasa care realizeaza interfata grafica a acestui proiect (Graphics).
- Pachetul tester prezinta clasa care realizeaza testarea unitara a acestui proiect (Tester).
- Pachetul model prezinta clasele si interfetele care modeleaza structurile necesare pentru realizarea acestui proiect (Account, Person, SavingAccount, SpendingAccount, Observable, Observer)
- Pachetul controller prezinta clasele si interfata care controleaza fluxul de executie al aplicatiei (BankProc, Bank, BankAdmin).

## f. Interfata grafica




Interfata grafica a acestui sistem de managemenet bancar este un mediator intre utilizator si partea de cod a programului. Interfata grafica a acestui proiect este realizata cu Java Swing, intr-un mod relativ simplu, dar totusi deosebit de intuitiv si de usor de utilizat pentru utilizatorul final al aplicatiei.

Fereastra care se deschide la rularea aplicatie poarta numele de Bank. Fereastra prezinta in partea superioara un meniu simplu din care se poate alege operatia dorita de utilizator. La apasarea butoanelor ce tin de vizualizarea tabelelor, se afiseaza tabelul solicitat de utilizator. Cand se deschide fereastra de introducere a unui nou obiect in tabel, utilizatorul trebuie sa completeze toate campurile cu valorile solicitate, de tipul solicitat, iar apoi sa apese butonul de confimare. In momentul cand utilizatorul doreste sa editeze un obiect, va trebui mai intai sa introduca o informatie solicitata pentru obiectul supus editarii si sa apese butonul de confirmare a acestei informatii, iar apoi poate sa introduca campul pe care doreste sa il editeze, urmand apoi sa apese butonul de confirmare. La stergerea unui rand dintr-un tabel, utilizatorul nu trebuie decat sa introduca informatia solicitata pentru randul respectiv si sa apese pe butonul de confirmare. La realizarea unei operatiuni de depunere sau retragere, modul in care utilizatorul interactioneaza cu interfata este asemanator adaugarii unui nou rand (se introduc datele solicitate si se apasa butonul de confirmare).

## 4. Implementare

In ansamblul procesului de proiectare, pe langa luarea deciziilor importante de proiectare, alaturi de realizarea diagramelor de clase, de pachete si a cazurilor de utilizare, inca un pas deosebit de important il constituie scrierea codului efectiv.

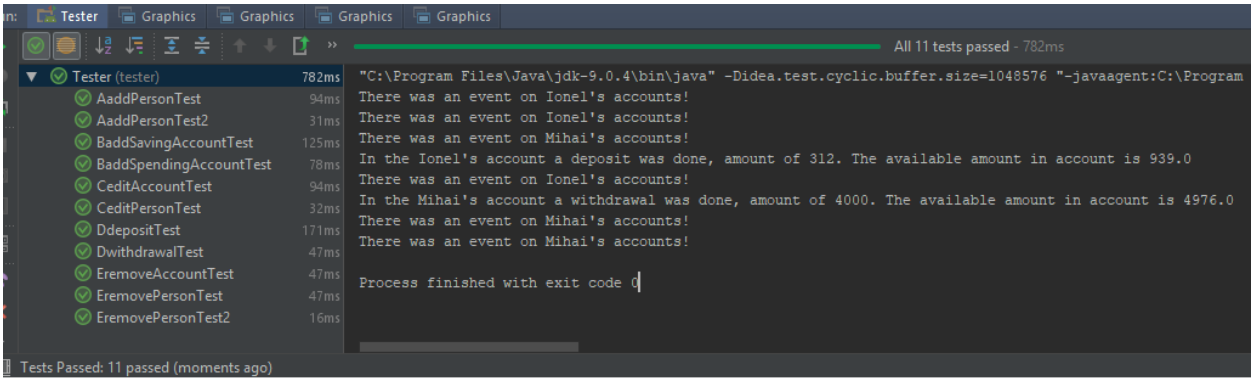
Una din principalele idei care au stat la baza scrierii codului pentru acest proiect a fost realizarea cate unei clase pentru fiecare din tabelele componente ale aplicatiei. Afisarea tabelelor s-a realizat prin utilizarea tehnicilor de reflection. Stocarea datelor s-a realizat intr-un fisier prin utilizarea tehnicilor de serializare a datelor, astfel, acest fisier functionand asemanator unei baze de date.

Pe parcursul codului, verificarea corectitudinii anumitor date s-a realizat prin utilizarea design pattern-ului Design by Contract, iar pentru a putea notifica persoanele inregistrate in banca asupra oricaror operatiuni ce au loc pe conturile acestora s-a utilizat design pattern-ul Observer.

Inglobarea tuturor acestor concepte a condus la aplicatia de fata, care se compune din clase cu cod destul de substantial, dar care datorita organizarii clare, nu prezinta dificultate in intelegere si dezvoltare ulterioara.

## 5. Testare

Acest program este unul care e dependent in mare masura de intrarile furnizate de catre utilizator, fapt care il face mai putin plauzibil pentru o testare unitara, dar acest lucru nu este imposibil. Programul a fost atent testat in fiecare detaliul al sau, iar modul riguros si defensiv de scriere a codului asigura o buna functionare a programului in conditiile anterior specificate. Pentru a verifica corectitudinea programului pentru un set de date de intrare s-a realizat si o clasa specializata in testarea unitara. Testarea a constat in verificarea mesajelor de success sau de esec furnizate de catre metodele care efectueaza operatiunile implicate de acest proiect. Programul trece toate testele impuse, fapt ce se poate vedea in poza de mai jos.




## 6. Rezultate

Rezultatele furnizate de catre program sunt prezentate in interfata utilizator si pot fi vizualizate prin apasarea butoanelor dedicate pentru vizualizarea tabelelor. De asemenea in cazul in care utilizatorul doreste sa realizeze mai multe operatiuni pe un cont de economii, va primi un mesaj prin care va fi instiintat asupra faptului ca nu mai poate face acest lucru. In consola programului se vor afisa orice interventii asupra conturilor, prin utilizarea design pattern-ului observer prin care posesorii conturilor pot afla orice operatiune sau incercare de operatiune care a avut loc pe vreu cont al lor. De asemenea, la efectuarea operatiunilor de deposit si withdrawal, se va afisa in consola programului un mesaj corespunzator unei chitante primite de la banca, prin care utilizatorul este instiintat asupra contului, posesorului contului, sumei retrase/depuse, cat si a celei disponibile. La vizualizarea tabelelor se va afisa in consola un mesaj care va specifica daca banca e corect formata (fiecare persoana

inregistrata in banca sa aiba cel putin un cont). Deoarece s-a folosit design pattern-ul Design by Contract, la introducerea unor valori ale campurilor, care nu sunt corespunzatoare cu cerintele aplicatiei, se va afisa un mesaj de eroare specific in consola aplicatie.

## 7. Concluzii

Personal consider faptul ca acest proiect a fost unul deosebit de util si important, deoarece m-a ajutat sa inteleg, sa asimilez si sa vad utilitatea multor notiuni de programare orientata pe obiecte, de care nu auzisem pana acum. Utilizarea structurii de tip Hashtable, alaturi de modul in care acesta functioneaza si de metodele asociate au constituit pentru mine o adevarata provocare, abordata prin studiu intens si incercari repetate. Utilizarea design pattern-ului Observer, m-a facut sa inteleg mai bine importanta organizarii corecte a structurii aplicatiei si de asemenea modul in care, prin definirea unor interfete potrivite si suprascrierea unor metode predefinite, se poate face o buna comunicare intre diverse parti ale unui proiect. Oportunitatea de a folosi JUnit in cadrul acestui proiect, in care testarea s-a facut intr-un mod mai complex, m-a ajutat sa inteleg eficienta mecanismelor de testare pe aplicatii mai mari, nu doar pe cele triviale. Unul din scopurile principale ale temei curente fiind Design by Contract, am reusit sa observ utilitatea preconditiilor, postconditiilor si a invariantilor, si de asemenea a modului in care, impreuna cu instructiuni de tip assert, acestea pot furniza o structura defensiva programului, care sa fie inteligibila, permitand utilizatorul auto-corectia in cazul efectuarii vreunei greseli in ceea ce priveste introducerea datelor. Utilizarea serializarii pentru salvarea datelor a fost de asemenea o notiune noua pentru mine, iar acest proiect mi-a oferit sansa de a o invata si de a reusi sa o pun in practica, observand deslusit utilitatea acesteia. Utilizarea tehnicilor de reflection in construirea tabelelor de afisat in interfata grafica, m-a facut sa inteleg tot mai mult utilitatea acestor thenici.

Astfel, acest proiect a inglobat o multime de tehnici, structuri, concepte, dintre care unele cunoscute, altele mai putin sau deloc, dar prin complexitatea acestuia, proiectul de fata a constituit o adevarat provocare, o provocare la a inavata tot mai mult si tot mai bine limbajul Java si programarea orientata pe obiecte.

Acest proiect nu este o forma desavarsita, el fiind doar o oglindire a ceea ce o astfel de aplicatie ar putea fi. Printre posibile imbunatatiri ulterioare pe care acest proiect le-ar suporta se numera: realizarea unei interfete utilizator mai eleganta si mai usor de utilizat, adaugarea si altor operatii bancare, care nu au fost obiectul acestui proiect, adaugarea mai multor caracteristici pentru fiecare din entitatile implicate, si chiar adaugarea altor entitati utile, si lista ar putea continua.

## 8. Bibliografie


http://www.tutorialspoint.com/java/java_serialization.htm

https://stackoverflow.com/

http://javarevisited.blogspot.ro/2011/02/how-hashmap-works-in-java.html

https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

https://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise