



**Facultatea de Automatica și Calculatoare**

**Tema 4:**  
**Sistem de gestiune a tranzacțiilor**  
**în cadrul unei banci**

**Disciplina: Tehnici de programare**

**Student:**

**Coman Vasile**

**An II**

**Grupa 30221**

**Profesor coordonator:**

**Antal Marcel**

**Profesor curs:**

**Ioan Salomie**



## **Cuprins**

<b>1. Obiectivul temei.....</b>	<b>3</b>
<b>2. Analiza problemei, asumptii, modelare, scenarii, cazuri de utilizare, erori.....</b>	<b>3</b>
<b>3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase, interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor).....</b>	<b>4</b>
<b>4. Implementare.....</b>	<b>5</b>
<b>5. Testare.....</b>	<b>6</b>
<b>6. Rezultate.....</b>	<b>8</b>
<b>7. Concluzii.....</b>	<b>11</b>
<b>8. Bibliografie.....</b>	<b>12</b>



## 1. Obiectivul temei

Obiectivul principal al temei este proiectare și implementarea unui sistem de gestiune a tranzacțiilor în cadrul unei bănci folosind tehnica de programare „Design by contract”.

Obiective secundare:

- posibilitatea de a crea conturi clienților acelei bănci, fiecare client putând să își deschidă mai multe conturi stocarea lor într-o structură de date care îi poate identifica unic pe fiecare dintre clienți, structura de tip Map cu key și value unde key oferă unicitate;
- notificarea clienților când are loc o modificare a unuia dintre conturile acestuia;
- oferirea posibilității clienților de a efectua tranzacții asupra unui cont.

Toate aceste obiective duc la implementarea și ajungerea la un sistem complet de gestiune a unei bănci și a operațiilor din cadrul acesteia.

## 2. Analiza problemei, asumptii, modelare, scenarii, cazuri de utilizare, erori

Cerinte și funcționalități:

- adaugare, editarea și ștergerea unei persoane în/din baza de date a băncii;
- adaugare, editarea și ștergerea unui cont asociat unei persoane;
- posibilitatea de a depune sau retrage bani dintr-un cont.

Use-case

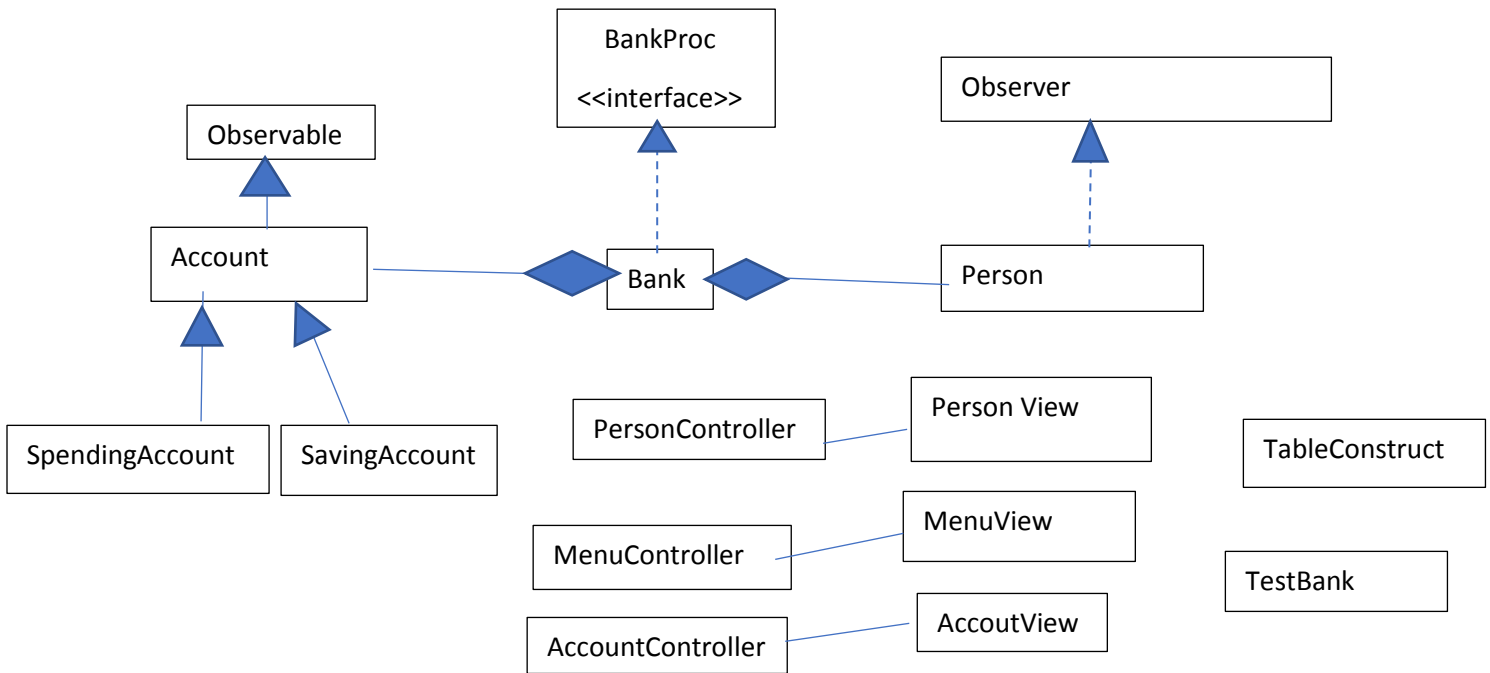
Avem un sistem care permite unui utilizator de a introduce persoane în baza de date a băncii și de a face operații pe acestea de editare și ștergere cât și posibilitatea de creare de conturi de tip saving sau spending acelor persoane. De asemenea utilizatorul poate să introducă sau să retragă bani din cont acest lucru notificându-i-se automat în urma implementării Design Pattern-ului Observer.

Asumptii făcute:

- presupunem ca utilizatorul introduce corect numele, cnp-ul și adresa persoanei;
- presupunem ca utilizatorul introduce dobânda în mod corect, alte erori sau excepții fiind tratate în aplicație.



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



Am ales sa impart proiectul in 13 clase si o interfata:

- interfata **TestBank** unde vom defini operatiile pe care le va putea face aplicatia;
- clasa **Bank** care implementeaza **TestBank** implementeaza operatiile definite de aceasta interfata;
- clasa **Person** defineste un client al bancii;
- clasa **Account** defineste un cont iar clasele **SpendingAccount** si **SavingAccount** extind aceasta clasa si definesc tipuri de conturi diferite;
- clasele pentru interfata grafica, un frame pentru meniu, un frame pentru persoane si unul pentru conturi fiecare de tip controller-view;
- o clasa care genereaza un model pentru tabel;
- si o clasa de test pentru verificarea operatiilor.



## 4. Implementare

Clasa `Person` definește o persoană prin nume, `cnp` și adresă și implementează interfețele `Observer` și `Serializable` pentru ca informațiile vor fi citite și scrise într-un fișier cu extensia `.ser` care are proprietatea de a serializa la scriere și deserializa la citire informațiile și poate să facă `split` după câmpuri. În această clasă avem metode de tip `set` și `get` cât și suprascrierea metodelor `equals` și `hashCode`. Am ales ca `hashCode`-ul să fie primele 8 cifre din `cnp` iar metoda `equals` returnează `true` la comparare dacă cele două persoane au același `hashCode`.

Clasa `Account` extinde `Observable` și implementează `Serializable` la fel ca și clasa `Person`. Conține câmpurile `iban` care este un șir de 24 de caractere generat aleator folosind un alfabet cu toate literele și cifrele, `amount` care reprezintă suma din cont, un câmp de tip `LocalDate` care reprezintă data creării contului pentru conturile de tip `Spending` iar pentru conturile de tip `Saving` reprezintă inițial data creării contului iar mai apoi data primei și singurei depuneri posibile în vederea posibilității calculării dobânzii în momentul în care va dori să retragă acea sumă. Pe lângă metodele de `set` și `get` conține metoda `addMoney` și `withdrawMoney` care modifică soldul din `account`.

Clasa `SpendingAccount` extinde clasa `Account` și nu are nimic în plus față de aceasta utilizatorul putând retrage și depozita de câte ori dorește lucru acesta.

Clasa `SavingAccount` extinde clasa `Account` și conține 3 câmpuri în plus față de `Account`, 2 câmpuri de tip `boolean` care ne arată dacă s-a depozitat sau nu în acel cont și dacă s-a retras sau nu din acel cont deoarece conturile de tip `Saving` permit retragerea și depozitarea unei sume de bani doar o singură dată. Câmpul dobândă modifică soldul contului. La prima și singura depunere data creării contului se înlocuiește cu data depozitării. Această clasă conține metoda `refreshAccount` care calculează cu cât crește soldul persoanei folosind dobânda și data depozitării și modifică `amount`. Astfel că atunci când persoana va dori să își retragă banii din cont acestuia îi va apărea noul sold calculat în urma dobânzii stabilite la început.

Interfața `BankProc` definește metodele de `add`, `remove` și `edit` pentru persoane și conturi cât și operațiile de retragere și depunere în cont.

Clasa `Bank` implementează interfața `BankProc` și `Serializable` și conține un `Hashtable` de persoane și conturi unde cheia este persoana și `value` este un `ArrayList` de conturi din cauza că o persoană poate avea mai multe conturi.

Aici găsim metodele de `read` și `write` scriu respectiv citesc din fișier datele despre conturi și persoane. Când instanțiem un obiect de tip `Bank` în constructor se va apela metoda `read` care va citi din fișier iar de fiecare dată când vom face o modificare asupra unui client sau a unui `account` vom apela metoda `write` de scriere în fișier.

Deoarece în implementare am folosit tehnica de programare `Design by Contract` atunci la fiecare metodă din `Bank` găsim asertiuni pe post de precondiții și postcondiții unde la precondiții verificăm dacă datele care urmează să se proceseze sunt valide și la postcondiții verificăm dacă rezultatul este cel așteptat. Astfel utilizând asertiuni cât mai bine stabilite



putem crește corectitudinea aplicației deoarece nu lasăm să se încalce constrangerile astfel că dacă una din asertiuni este falsă se aruncă excepție.

Implementarea Design Pattern-ului Observer:

-clasa Account extinde Observable conține o listă de observatori, metoda de registerObservers care adaugă observatori la acea listă, removeObservers care șterge observatori din listă și notifyObservers care parcurge fiecare observator din listă și apelează pentru fiecare observator metoda de update definită în clasa Person care îi notifica persoanei când unul dintre conturi s-a modificat, lucru sesizabil în interfața grafică printr-un JOptionPane.

De fiecare dată când adăugăm în Hashtable câte un obiect de tip account adăugăm și câte un observer pentru acel cont. Iar când ștergem contul ștergem și observatorul. Când se execută o operație de retragere sau depunere se apelează metoda notifyObservers care îi va notifica clientului modificarea contului.

Pentru interfața grafică am folosit metoda de construcție model-view și am construit 3 ferestre. Prima fereastră este un meniu unde utilizatorul poate alege să deschidă fereastră de persoane sau cea de conturi.

În fereastră de persoane avem textfield-uri pentru adăugarea de date în cazul în care dorim să adăugăm o nouă persoană, avem butoane pentru afișare tabel cu persoane. Tabelul are adăugați listeneri care permit selectarea unui rând iar dacă apăsăm butonul de delete se va șterge persoana selectată la fel și pentru edit.

În fereastră de conturi alegem persoana careia dorim să îi deschidem un cont, alegem tipul de cont saving sau spending dacă alegem saving va trebui să introducem și dobândă și putem selecta butonul de adăuga cont astfel încât se va crea contul. Avem un textfield pentru suma pe care dorim să o depunem sau retragem. În cazul în care vrem să retragem o sumă mai mare decât este în cont sau dacă vrem să retragem sau să depunem de mai multe ori într-un cont de tip saving aplicația nu ne va permite acest lucru deoarece se vor afișa mesaje cu atenționari.

Tot aici se va sesiza implementarea Design Patternului Observer deoarece la fiecare tranzacție de retragere sau depunere în cont îi se va notifica acest lucru utilizatorului.

O clasă mai specială este cea TableConstruct care extinde DefaultTableModel. Această clasă are metoda createTable care primește o listă de obiecte. Această metoda parcurge fiecare câmp al listei de obiecte și adăugă capurile de tabel. Apoi parcurgem lista de obiecte și salvăm într-un vector valorile iar la final adăugăm acel vector ca și linie la tabel și în final modelul este creat.

Clasa TestBank este o clasă de tip JUnitTestCase care testează anumite cazurile pe care dorim să le testăm și ne arată dacă au trecut sau nu testul așa ne putem da seama dacă aplicația funcționează în mod corect sau nu.

## 5.Testare

Pentru testare am folosit un JUnit Test Case și am ales 4 cazuri de a verifica aplicația.



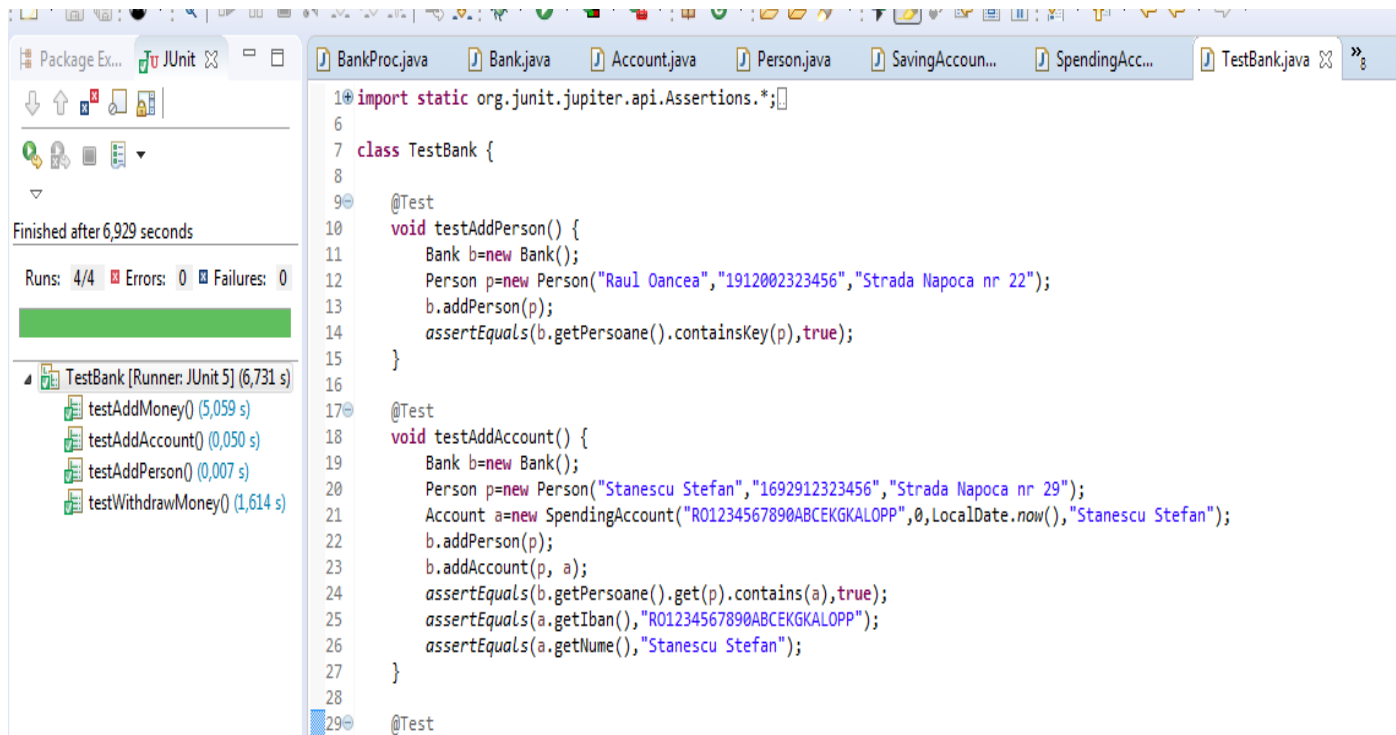
Tip test	Date intrare	Rezultat asteptat	Rezultat obținut	Pass /fail
Add Person	Person p=new Person("Raul Oancea","1972302323456", "Strada Napoca nr 22")	b.getPersoane().containsKey(p)==true	true	Pass
Add Account	Account a=new SpendingAccount("RO1234567890ABCEKGKALOPP", 0,LocalDate.now(),"Stanescu Stefan")	b.getPersoane().get(p).contains(a)==true	true	Pass
Add Money	a.addMoney(200);	a.getAmount()==200	true	Pass
Withdraw Money	a.withdrawMoney(200);	a.getAmount()==0	true	Pass

Aplicatia a trecut cu succes toate cele patru teste sirmatoarele print-screenuri arata acest lucru.

```

29 @Test
30 void testAddMoney() {
31     Bank b=new Bank();
32     Person p=new Person("Podar Tudor","1902907323456","Strada Lunii nr 41");
33     Account a=new SpendingAccount("ROXXX4567890ABCEKGKALOPP",0,LocalDate.now(),"Podar Tudor");
34     b.addPerson(p);
35     b.addAccount(p, a);
36     a.addMoney(200);
37     assertEquals(a.getAmount(),200);
38 }
39
40 @Test
41 void testWithdrawMoney() {
42     Bank b=new Bank();
43     Person p=new Person("Andrioaia Octavian","1922101323456","Calea Floresti nr 45");
44     Account a=new SpendingAccount("ROXXXAF67890ABCEKGKALOPP",500,LocalDate.now(),"Andrioaia Octavian");
45     b.addPerson(p);
46     b.addAccount(p, a);
47     a.withdrawMoney(200);
48     assertEquals(a.getAmount(),300);
49 }

```



## 6.Rezultate

O data ce am rulat aplicatia ne va aparea urmatorul frame:



Apoi daca selectam buutonul de persoane ne va aparea un frame unde putem face add,edit, delete si view de persoane:





Persoane

Nume	Coman Vasile	Add person	Edit person	Delete person	View persons
CNP	1981003123456				
Adresa	Strada Zorilor nr 36				

Vizualizarea datelor intr-un jTable:

Persoane

Nume		Add person	Edit person	Delete person	View persons
CNP					
Adresa					

nume	cnp	adresa
Coman Vasile	1981003123456	Strada Zorilor nr 36
Petrisor Mihaela	1960509123456	Aleea Baita nr 35
Petre Iuliana	1970407123456	Calea Baciului nr 56
Todea Daniel	1951112981098	Calea Florestii nr 68

Deschiderea frameului de account-uri si posibilitatea de a adauga, sterge, vedea si edita conturi cati si de a retrage si depozita bani:

MINISTERUL EDUCAȚIEI NAȚIONALE



**UNIVERSITATEA TEHNICĂ**  
DIN CLUJ-NAPOCA

---

Titularul contului  
Tip  
Dobanda

Coman Vasile  
Spending Account

Add account Edit account Delete account View accounts  
Suma Add money Withdraw money

Vizualizarea datelor despre conturi intr-un jTable:

Titularul contului  
Tip  
Dobanda


Todea Daniel  
Spending Account

Add account Edit account Delete account View accounts  
Suma Add money Withdraw money

observers	iban	amount	date	nume	dobanda	depus	retras
	RO324LJO66...	0.0	2018-05-16	Coman Vasile	0.5	false	false
	ROIXUGIMPD...	0.0	2018-05-16	Petrisor Miha...	0.9	false	false
	ROC2P1EOM...	0.0	2018-05-16	Coman Vasile			
	ROFUS3XO97...	0.0	2018-05-16	Petre Iuliana			
	ROKMQ4YAJJ...	0.0	2018-05-16	Todea Daniel			

Notificarea data de observatori cand contul s-a schimbat:

Message



Contul a fost modificat, noua suma este 123.0

OK



Modificarea JTable-ului dupa efectuarea de tranzactii:

Titularul contului: Todea Daniel

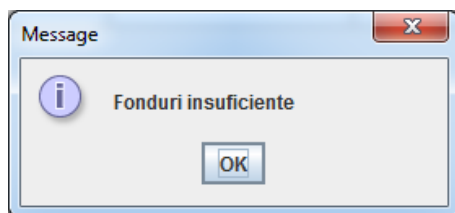
Tip: Spending Account

Dobanda: 0.5

observers	iban	amount	date	nume	dobanda	depus	retras
	RO32410068...	150.0	2018-05-16	Coman Vasile	0.5	true	false
	ROXUGIMPDI...	10.0	2018-05-16	Petrisor Miha...	0.9	false	false
	ROC2P1EOM...	100.0	2018-05-16	Coman Vasile			
	ROFUS3XO97...	10.0	2018-05-16	Petre Iuliana			
	ROKMO4YAJJ...	10.0	2018-05-16	Todea Daniel			

Buttons: Add account, Edit account, Delete account, View accounts, Suma, Add money, Withdraw money

Afisarea de casute de dialog in cazul in care nu se pot realiza diferite operatiuni:



## 7.Concluzii

In concluzie aceasta aplicatie implementeaza corect si eficient gestiunea tranzactiilor din cadrul unei banci oferind o interfata grafica usor de utilizat simuland ce se intampla in viata reala in orice banca.

Posibilitati de dezvoltare ulterioara:

- adaugarea de noi operatiuni pentru cont: transfer intre conturi, descoperire de card, plate de facturi;
- implementarea unor taxe pentru aceste servicii la fel ca si in viata reala;
- adaugarea de mai multe tipuri de conturi nu doar saving si spending;
- adaugarea unui cod pin pentru card operatiunile viind valide doar in urma introducerii pinului.

In urma rezolvarii acestei teme am invatat o multime de lucruri:



- am invatat sa implementez tehnica de programare Design by Contract prin folosirea de assertiuni ca preconditionii si postconditii care nu permit incalcarea constrangerilor impuse de specificatia aplicatiei;
- am invatat serializarea si deserializarea prin scriere si citirea dintr-un fisier cu extensia .ser;
- am invatat sa folosesc Design Patternul Observer pentru notificarea clientilor daca un cont de-al acestora a fost modificat;
- am aprofundat testarea aplicatiei folosind JUnit si construirea de JTable, interfete de tip controller-view si tratarea de exceptii;

## 8.Bibliografie

[http://www.tutorialspoint.com/java/java\\_serialization.htm](http://www.tutorialspoint.com/java/java_serialization.htm)

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

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

<https://dzone.com/articles/observer-design-pattern-java>

[http://www.coned.utcluj.ro/~salomie/PT\\_Lic/4\\_Lab/](http://www.coned.utcluj.ro/~salomie/PT_Lic/4_Lab/)

<http://www.coned.utcluj.ro/~marcel99/PT/>