**DOCUMENTAŢIE TEMA 4**

**DESIGN BY CONTRACT PROGRAMMING TECHNIQUES, DESIGN PATTERNS**

**Nume: Prodan Andreea**

**Grupa: 30225**

**Profesor Laborator Assist Antal Marcel**

Contents

[1. Cerinte Functionale 3](#_Toc481485012)

[2. Obiective 3](#_Toc481485013)

[2.1. Obiectiv Principal: 3](#_Toc481485014)

[2.2. Obective Secundare: 3](#_Toc481485015)

[3. Analiza Problemei 4](#_Toc481485016)

[4. Proiectare 6](#_Toc481485017)

[4.1. Structuri de date 6](#_Toc481485018)

[4.2. Diagrama de clase 6](#_Toc481485019)

[4.3. Determinarea precondiţiilor şi a postcondiţiilor 6](#_Toc481485020)

[5. Implementare 7](#_Toc481485021)

[**Interfaţa utilizator** 7](#_Toc481485022)

[6. Concluzii si Dezvoltari Ulterioare 7](#_Toc481485023)

[7. Bibliografie 8](#_Toc481485024)

# Cerinte Functionale

Consideraţi o aplicaţie pentru conturile bancare a unei bănci care să permită extragerea şi depunerea în conturi. Titularii conturilor vor putea să seteze dacă vreau să fie notificaţi asupra orcărei modificări asupra conturilor prsonale. Toate datele băncii vor fi salvate/ încărcate în/ dintr-un fişier.

# Obiective

## Obiectiv Principal:

Dezvoltarea unei aplicaţii utilizând Design by Contract Programming Techniques (postcondiţii şi precondiţii) şi Design Patterns (observator).

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri şi scenarii | Determinarea fiecărei opţiuni pe care utilizatorul o are şi a fiecărei decizii pe care o poate lua. | 3 |
| Alegerea structurilor de date | Utilizarea unei colecţii predefinite care utilizează hashtable | 4 |
| Determinarea interferţelor şi a claselor necesare | Crearea unei interfeţe pentru bancă, pentru observator şi pentru observabil. Determinarea unei superclase pentru conturi. | 4 |
| Determinarea precondiţiilor şi a postcondiţiilor | Definirea lor în interfaţa băncii şi implementarea lor în clasa care implementează interfaţa, utilizânt assertion. Creearea unui invariant şi a unei metode “well formed” pentru clasa băncii. | 4 |
| Implementarea soluţiei | Crearea claselor şi obiectelor necesare. Transpunerea punctelor anterioare în limbaj de programare Java. | 5 |

# Analiza Problemei

Actori:

* Utilizatorul

Trigger:

* Utilizatorul va apăsa unul dintre butoanele afişate pe ferestre după introducerea informaţiilor necesare în căsuţe sau selectarea unui rând din tabel.

Precondiţii:

* La deschiderea aplicaţiei toate datele dinaintea ultimei utilizări a aplicaţiei vor fi încărcate înaipoi în aplicaţie.
* Utilizatorul va putea introduce date despre ceea ce vrea să vadă sau să caute legat de conturi sau titularii conturilor.

Post-condiţii:

* Utilizatorul va putea vizualiza câte un tabel în fiecare fereastră. La selectarea anumitor preferinţe, i.e. date doar pentru o anumită persoană, tabelul va fi reactualzat cu respectivele informaţii.
* La selectarea unui rând din tabel, datele respective vor fi afişate şi în căsuţe. După aceea se va putea selecta o operaţie pe respectivele date. Astfel nu va mai fi necesară introducerea de la tastatură a datelor de către utilizator.
* La închiderea aplicaţiei toate datele vor fi salvate într-un fişier.

**Use Case:**

Sunt afişate două ferestre pentru titulari (Person) şi conturi(Account).

A. Fereastra Person. Tabelul cu toţi titularii este afişat în partea stângă.

1. Utilizatorul doreşte să adauge o nouă persoană
   1. Utilizatorul introduce în fiecare căsuţă din dreptul butonului ‘Add Person’ datele persoanei (id-ul, numele şi numărul de telefon)
   2. Utilizatorul apasă butonul pentru confirmare
      1. Toate datele sunt valide. Tabelul va afişa toţi titularii, având inclusiv noul titular.
      2. Datele nu sunt valide. O eroare va fi aruncată de către assert
2. Utilizatorul doreşte să vizualizeze datele unui anumit titular
   1. Utilizatorul introduce id-ul titularului pe care doreşte să-l caute, în căsuţa din dreptul butonului ‘Show Person’ şi apasă butonul pentru a confirma.
      1. Titularul căutat a fost găsit, prin urmare tabelul va afişa datele clientului/ clinţilor respectivi.
      2. Nu a fost găsit nici un titular cu id-ul introdus. Tabelul returnat va fi gol
3. Utilizatorul doreşte să vizualizeze toţi titularii
   1. Utilizatorul apasă butonul ‘View All’
4. Utilizatorul doreşte să şteargă un titular
   1. Utilizatorul va introduce id-ul persoanei în căsuţa din dreptul butonului ‘Delete’ sau va selecta din tabel persoana respectivă, caz în care în căsuţă va apărea id-ul persoanei.
   2. Dacă titularul există, acesta va fi şters. Tabelul se va reactualiza.

B. Fereastra Account. Este afişat un tabel cu datele conturilor pentru fiecare titular.

1. Utilizatorul doreşte să adauge un nou cont

1.A Utilizatorul doreşte să adauge un nou cont curent

* 1. Utilizatorul introduce în căsuţele din dreptul butonului ‘Add Account’ datele contului. Ȋn câsuţa ‘Type:SP/SA’ va introduce ‘SP’ (spending account), pentru a menţiona tipul contului dorit. Ultima căsuţă va rămâne pe 0.
  2. Utilizatorul apasă butonul pentru a confirma
     1. Datele introduse nu sunt valide. Va fi aruncată o excepţie de către assert
     2. Datele sunt valide. Tabelul va afişa toate conturile, inclusiv noul cont introdus.

2.B Utilizatorul doreşte să adauge un nou depozit

1. Utilizatorul introduce în căsuţele din dreptul butonului ‘Add Account’ datele contului. Ȋn câsuţa ‘Type:SP/SA’ va introduce ‘SA’ (saving account), pentru a menţiona tipul contului dorit.
2. Utilizatorul apasă butonul pentru a confirma
   * 1. Datele introduse nu sunt valide. Va fi aruncată o excepţie de către assert
     2. Datele sunt valide. Tabelul va afişa toate conturile, inclusiv noul cont introdus.
3. Utilizatorul doreşte să vizualizeze datele unui anumit cont
   1. Utilizatorul va introduce id-ul titularului contului şi numărul contului căutat în căsuţa din dreptul butonului ‘Show Account’
   2. Utilizatorul va confirma prin apăsarea butonului
      1. Contul căutat nu există. Tabelul va fi gol.
      2. Contul căutat a fost găsit. Tabelul va afişa datele contului.
4. Utilizatorul doreşte să vizualizeze toate conturile
   * 1. Utilizatorul apasă butonul ‘View All’
     2. Tabelul va afişa toate conturile.
5. Utilizatorul doreşte să şteargă un cont
   1. Utilizatorul introduce id-ul titularului contului şi numărul contului în căsuţa din dreptul butonului ‘Delete’ sau selectează contul din tabel, caz în care datele vor apărea în căsuţă.
   2. Utilizatorul apasă butonul pentru confirmare
   3. Dacă contul există, acesta este şters. Tabelul este reactualizat
6. Titularul doreşte să depună o sumă într-un cont
   1. Utilizatorul va introduce id-ul său, numărul contului în care vrea se efectueze depunerea şi suma pe care doreşte să o depună, în dreptul căsuţei ‘Deposit’.
   2. Utilizatorul apasă butonul pentru confirmare
   3. Datele sunt valide. Suma va fi adăugată în cont. Tabelul va fi reactualizat cu noua sumă din contul respectiv.
   4. Datele nu sunt valide. Suma nu va fi adăugată

# Proiectare

## Structuri de date

Clasa Bank stochează toţi titularii şi conturile acestora într-o colecţie predefiită care utilizează hashtable, HashMap. Conturile sunt stocate într-o altă colecţie predefinită, HashSet. Fiecare titular va avea un set cu conturile personale. Ȋn HashMap, cheile vor fi reprezentate de către titulari (Person), iar seturile de conturi vor vi valorile.

HashMap<Person, HashSet<Account>> map;

S-a ales ales HashMap în favoarea lui HashTable, deoarece spre deosebire de prima, HashTable este mult mai lent fiind sincronizat. Cum în această aplicţie nu se utilizează multithread-uri. Ȋn al doilea rând dacă s-ar dori totuşi o colecţie sincronizată se poate alege ConcurrenHashMap, care este o colecţie mai recentă, utilizată cel mai des în prezent pentru aplicaţii cu mai multe thread-uri.

Pentru stocarea conturilor s-a ales Set deoarece nu permite introducerea duplicatelor. Cum nu pot eixista două conturi cu acelaşi număr de identificare, această colecţie este o alegere bună.

Penru Person, egalitatea se face pe baza id-ului care este unic, iar pentru Account pe baza lui accountNo, care de asemenea este unic.

## Diagrama de clase

Observer

Observable

Bank

Person

Bank

Account

Controller

Spending

Account

Saving

Account

ViewPerson

ViewAccount

Start

## Determinarea precondiţiilor şi a postcondiţiilor

Pentru detectarea de erori sau date nevalide se utilizează tehnici de design by contract.

Ȋn interfaţa Bank sunt definite toate precondiţiile şi postcondiţiile fiecărei metode. Implementarea lor se face în clasa Bank.

Pentru validarea integrităţii colecţiei, s-a creeat o metodă ‘well formed’ isWellFormed() care defineşte invariantul clasei Bank. Aceasta este apelată la finalul oricărei metode care face modificări asupra coecţiei (mutatori). Paşii de verificare sunt parcurgerea colecţiei şi numărarea tuturor cheilor şi a valorilor. Dacă ecestea nu sunt egale cu numărul returnat de apelul size(), atunci înseamnă că colecţia nu mai este validă.

# Implementare

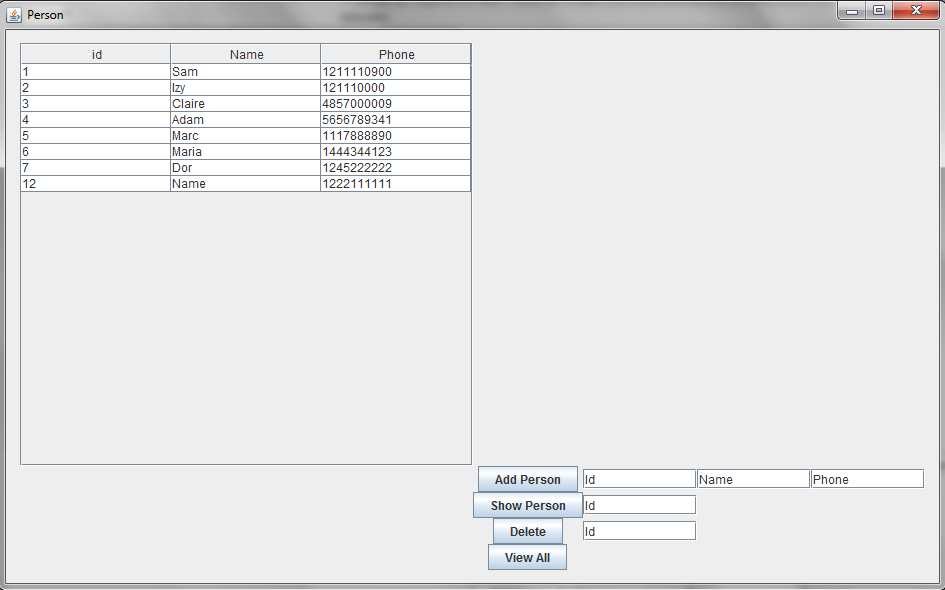
Pentru bancă se utilizează o interfaţă în care se definesc metodele şi postcondiţiile şi precondiţiile.

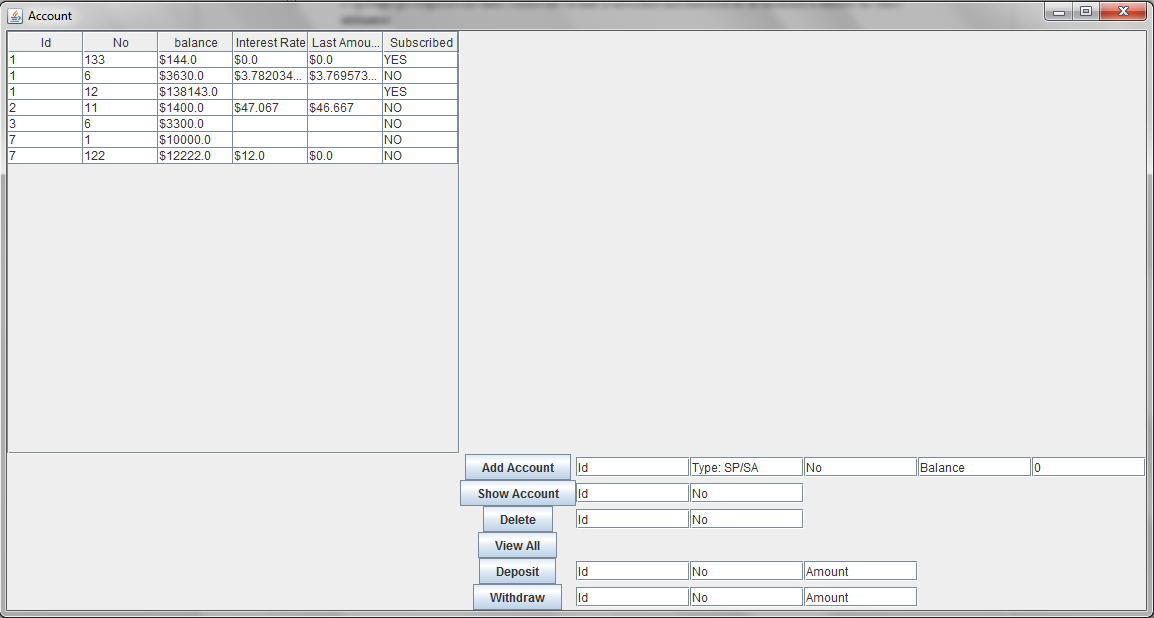
Penru a putea notifica clienţii asupra modificării conturilor, se creează două interfeţe, după Design Pattern Observer, Observer petru Person şi Observable pentru Account. Ȋn momentul în care un titular doreşte să fie notificat, se va apela o metodă subscribe() implementată în Person, care va apela o metodă din cont care îl va memora pe titular într-un câmp privat. Astfel, toate metodele contului care modifică cotul vor apela o metodă update() care va notifica titularul dacă acesta este memorat de către cont. La fiecare modificare, titularul va putea vedea contul care a fost modificat şi modificările care au avut loc.

Pentru conturi se foloseşte o superclasă care conţine metodele şi câmpurile comune între conturi. Sunt valabile două tipuri de conturi: cont curent şi depozit. Spre deosebire de contul curent, depozitul are un interest pentru care conţine două câmpuri care stochează rata interesului annual şi ultima suma obţinută din interes, şi metode care determină aceste sume.

### **Interfaţa utilizator**

Sunt realizate două clase pentru a genera două ferestre: Person şi Account.





Interfaţa utilizator este realizată cu ajutorul clasei GUI care conţine atât definiţia elementelor grafice cât şi definiţia Listenerilor şi tratarea lor într-o singuă metodă. Acest lucru se realizează prin extinderea clasei JFrame şi implementarea clasei ActionListener.

S-a ales implementarea interfaţei ActionListener în loc de tratarea independentă a fiecărui Listener în clase anonime datorită numărului relativ mare de Listeneri. Astfel codul este mai compact şi uşor de citit.

Conţine entităţi care stochează datele pe care le poate introduce utilizatorul. Acestea au o valoare predefinită în caz că utilizatorul decide să nu introducă alte date noi.

Clasele deţin o instanţă controller comună prin care apelează metode din Controller pentru a prelua sau trimite date spre baza de date. Astfel există o barieră între interfaţă şi bancă.

Pentru a face interfaţa mai interactivă s-a adăugat listeneri pe tabel. Astfel prin selectarea unui rând datele acelui rând vor fi trecute în căsuţele în care trebuie introduse datele, iar utilizatorul poate efectua o operaţie mult mai uşor. De asemenea pornirea notificărilor pentru un cont se face uşor prin selectarea căsuţei din coloana ‘Subscribe’ din contul dorit.

Pentru încărcarea datelor, la creearea biectului controller, în constructorul din Controller se apelează o metode de deserializare. Pentru a salva datele la inchiderea aplicaţiei, se utilizează addShutdownHook care apelează metoda de serializare.

# Concluzii si Dezvoltari Ulterioare

Aplicaţia este bună pentru a simula o bancă.

Pentru dezvoltări ulterioare, se poate găsi o altă metodă de a salva datele la închiderea aplicaţiei, metoda curentă având goluri (dacă aplicaţia este omorâtă sau dacâ apar erori).

# Bibliografie

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

<http://stackoverflow.com/>

<http://javahungry.blogspot.com/>

<https://www.tutorialspoint.com>