Universitatea Tehnica din Cluj-Napoca

Facultatea de Automatica si Calculatoare

Bank accounts

Documentatie – tema 4 –

Czako Zoltan

Grupa: 30225

Contents

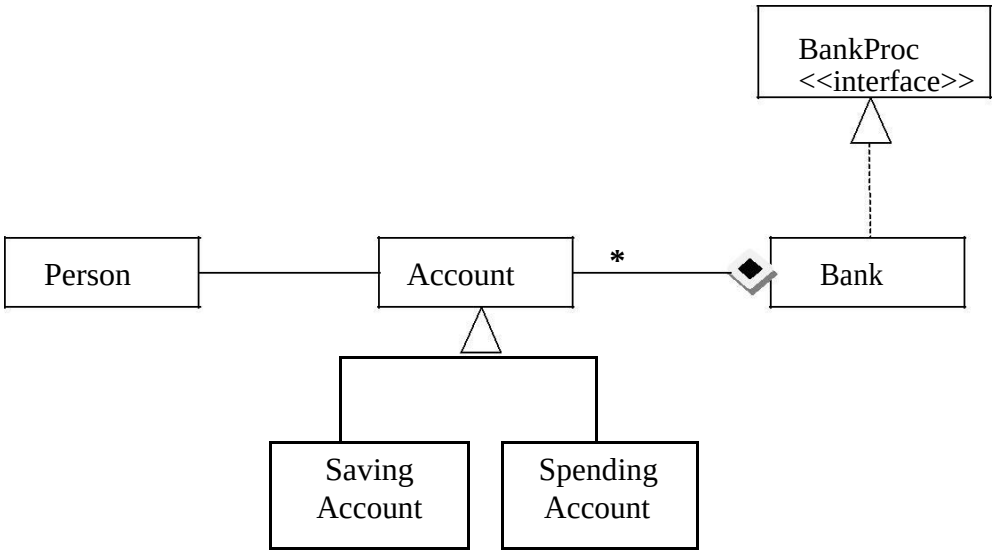
1. Obiectivul temei :

Objective

Design by Contract Programming Techniques

Description

Consider the system of classes in the class diagram below.



1.Define the interface BankProc (add/remove accounts, read/write accounts, report generators, etc). Specify the pre and post conditions for the interface methods.

2. Implement the classes Person, Account, SavingAccount and SpendingAccount. Other classes may be added as needed (give reasons for the new added classes).

3. Implement the class Bank using a hashtable. Chaining will be used in case of collisions. A method toString will be also defined for the class Bank.

3.1 Define a method of type "well formed" for the class Bank.

3.2 Implement the class using Design by Contract method (involving pre, post conditions, invariants, assertions).

4. Implement a user guided test driver for the system. The initial account data for populating a Bank object with Account objects will be taken from a file. Testing evidence (feedback and appropriate information) should be provided to the console as a result of executing the test cases.

2. Analiza problemei, modelare, scenarii, cazuri de utilizare

Aceasta problema apare si in viata reala, e defapt modelarea unui sistem de gestiune a conturilor bancare, adica aceasta aplicatie poate fi folosita de catre o banca, pentru adaugarea conturilor, pentru stergerea conturilor bancare sau folosit pentru implementarea unui automat bancar, care interactioneaza cu clientii. Clientul poate sa retraga o suma de bani din automatul bancar, poate sa vizualizeze contul sau propriu, adica

poate sa vizualizeze cati bani are in contul sau. Un client poate sa mearga la banca, unde are posibilitatea de a creea trei tipuri de cont bancar si anume cont normal, cont pentru economisire si cont de cheltuieli. Fiecare cont are diferite propritetati. Din contul normal putem sa scoatem bani oricand, insa are o crestere foare lenta. Contul de cheltuieli are ca proprietate posibilitatea ca un client sa foloseasca mai multi bani, decat soldul propriu, astfel poate sa faca un imprumut rapid, de suma mica pentru o perioada scurta de timp. Contul pentru economisire are gradul de crestere cel mai mare insa din acest cont utilizatorul nu poate sa extraga bani, numai dupa o perioada de timp bine definita.

Banca are o baza de date care foloseste o tabela de dispersie pentru stocarea conturilor bancare. Acest mod de stocare are ca avantaj un timp foarte scurt de regasire a conturilor, deci ofera o mare viteza de tranzactii intre conturi si o viteza extrem de mari in ceea ce priveste creearea si inserarea conturilor in baza de date.

In realizarea acestui proiect vom folosi tehnica Design by Contract, adica avem deja diagrama UML pentru aplicatia noastra, deci ramane doar implentarea efectiva a programului. Pe de alta parte aceasta tehnica de programare solicita folosirea pre- si post- conditiilor, ceeace vor fi implementate folosind assertion si exceptii definite pentru verificarea corectitudinii datelor de intrare.

3. Proiectare (diagrama UML, structuri de date, proiectare clase , interfete , relații, packages, algoritmi, interfața utilizator)

Pentru rezolvarea cat mai corecta si pentru o aplicatie cat mai avansatsa, am mai adaugat cateva clase la proiectul initial, clase care ne ajuta la pre si post conditii , si clase care realizeaza o interfata intre program si utilizator, facand ca utilizatorul sa poate utiliza acest program foarte usor, fara sa aiba nevoie de cunostinte din domeniul programarii a calculatoarelor.

Clasele existente:

ATM (Automated Teller Machine): clasa pentru interfata grafica intre utilizator si banca. Utilizatorul poate sa retraga bani din contul sau propriu , poate sa vizualizeze banii din contul curent sau poate sa ceara chitanta cu datele contului curent.

Account : clasa care are ca fielduri codul de IBAN pentru contul respectiv, fondul curent exprimat in RON, codul PIN pentru utilizatorul contului si procentajul de crestere a fondurilor, care difera in functie de tipul contului bancar (Account, Saving Account, Spending account)

Bank: clasa care contine toate conturile bancare, este organizat ca o tabela de dispersie, avand cheia reprezentat ca un String, adica cheia este IBAN-ul fiecarui cont. Aceasta clasa implementeaza interfata BankProc, care contine antetul urmatoarelor metode:

- readValue: metoda folosita pentru citirea datelor unui cont
- writeAccount: metoda folosita pentru modificarea datelor unui cont bancar
- addAccount: metoda folosita de banca petru creearea si adaugarea unui cont bancar in baza de date (aici tabela de dispersie)
- removeAccount: metoda folosita pentru stergerea unui cont bancar
- generateReport: metoda folosita pentru generarea unui raport al bazei de date, un raport al conturilor active

Person: clasa care contine datele personale al unui utilizator, adica numele, adresa, e-mailul, numarul de telefon si contul personal. Tot aceasta clasa contine niste metode de verificare a datelor introduse, adica pentru verificarea adresei de mail si pentru verificarea IBAN – ului.

SavingAccount: este o clasa care extinde clasa Account si suprascrie metoda withdrawMoney, metoda care realizeaza extragerea numerarului din contul propriu. In cazul acestui tip de cont bancar avem data de inceput si data de sfrasit, pana ce banii din contul respectiv sunt blocati, adica in acest interval de timp nu se poate extrage bani din acest cont.

SpendingAccount: este o clasa care extinde clasa Account si suprascrie metod withdrawMoney, metoda care realizeaza extragerea numerarului din contul propriu. In cazul acestui tip de cont bancar putem sa scoatem o suma mai mare de bani, decat suma ce avem pe contul nostru, adica putem sa facem un imprumut imediat, de o suma mica de bani (maxim 200 de RON).

Clase pentru exceptii:

1. EmptyFieldException: exceptia care este aruncata daca un field este gol sau contine doar spatii
2. ExistingAccountException: exceptia care este aruncata daca un cont bancar exista deja in tabela de dispersie
3. InexistentAccountException: exceptia care este aruncata daca vrem sa accesam un cont bancar care nu exista in tabela de dispersie
4. InsufficientMoneyException: exceptia care este aruncata daca vrem sa scoatem mai multi bani din contul nostru
5. WrongEmailFormatException: exceptia care este aruncata daca introducem in campul de mail o adresa de mail invalida, adica care nu contine @ sau care dupa @ nu are litere punct si iarasi maxim trei litere (exemplu ceva@yahoo.com)

RetragereNumerar: clasa care reprezinta o interfata grafica intre utilizator si automatul bancar, contine butoate pentru extragere de 10 lej, 20 de lej, 50 de lej, 100 de lej, 200 de lej si 500 de lej si mai are un camp in care se poate specifica alta suma de bani.

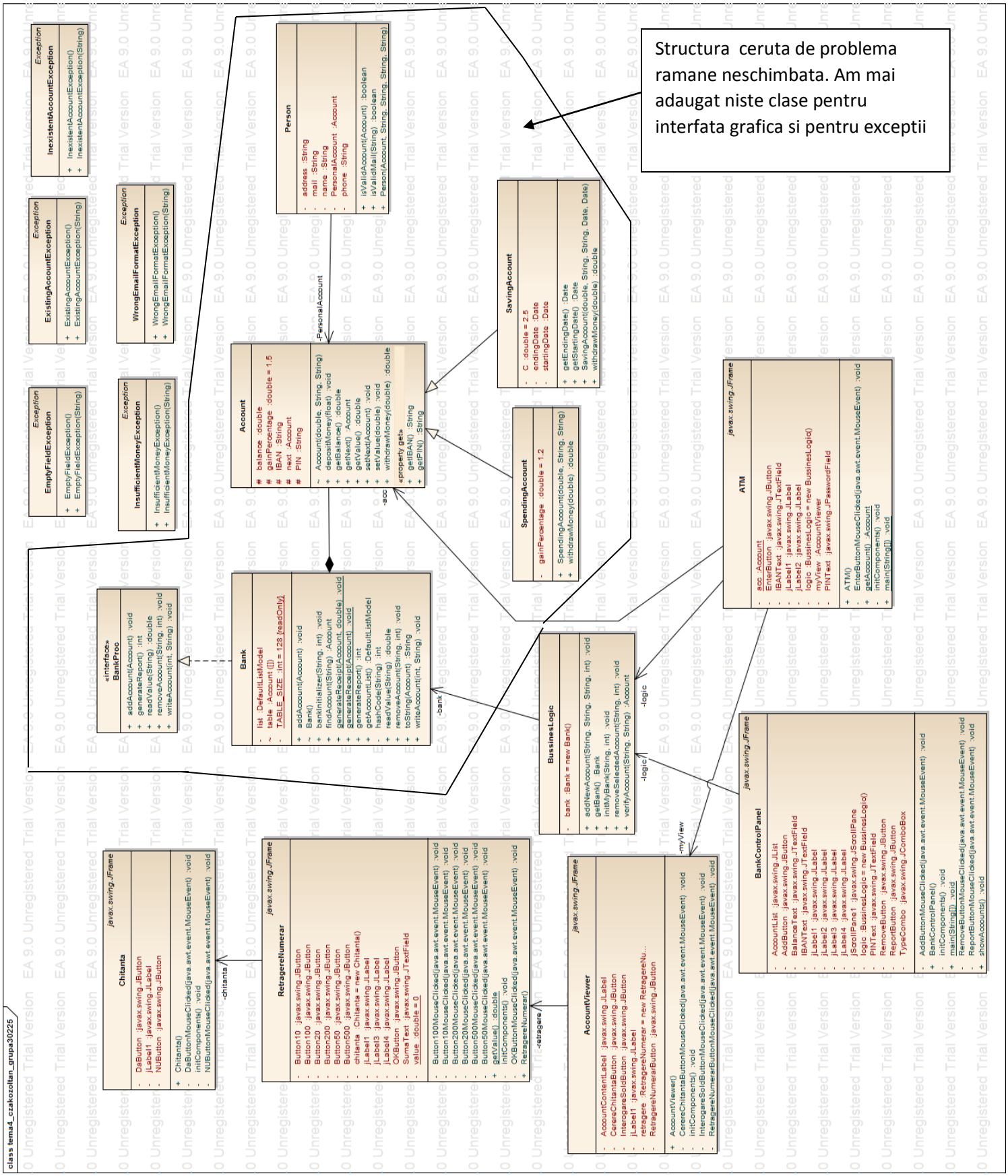
AccountViewer: clasa pentru interfata grafica a bancomatului, dupa introducerea cardului bancar si introducerea PIN-ului corect, aceasta clasa va realiza o interfata prin intermediul careia utilizatorul poate sa aleaga ce vrea sa faca, adica poate sa scoata bani cu butonul Retragere Numrar, poate sa interogheze contul sau propriu, adica poate sa vizualizeze fondul curent de bani sau poate sa ceara o chitanta, pe care va aparea codul IBAN, data de cerere, numarul bancomatului si suma de bani pe care o are pe acest cont.

Chitanta: clasa care contine o fereastra de interogare, daca clientul vrea chitanta, atunci se va genera o chitanta pe care va aparea codul IBAN, data de cerere, numarul bancomatului si suma de bani pe care o are pe acest cont, respectiv va contine suma de bani pe care utilizatorul a scos, adica va contine tranzactia curenta.

BankControlPanel: este on interfata grafica prin care banca poate sa gestioneze conturile active, poate sa adauge alte conturi la cererea unui utilizator, poate sa stearga un cont bancar sau poate sa genereze un raport al conturilor active, care va contine informatii legate de tipul contului, codul IBAN, si suma de bani pentru fiecare cont.

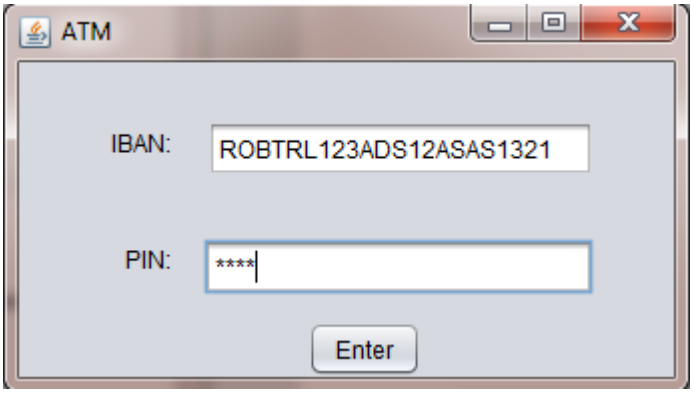
BussinesLogic: clasa care realizeaza legatura intre backend si frontend, adica legatura intre interfata grafica si logica de control si date, creeaza obiecte si verifica corectitudinea datelor

Diagrama UML:

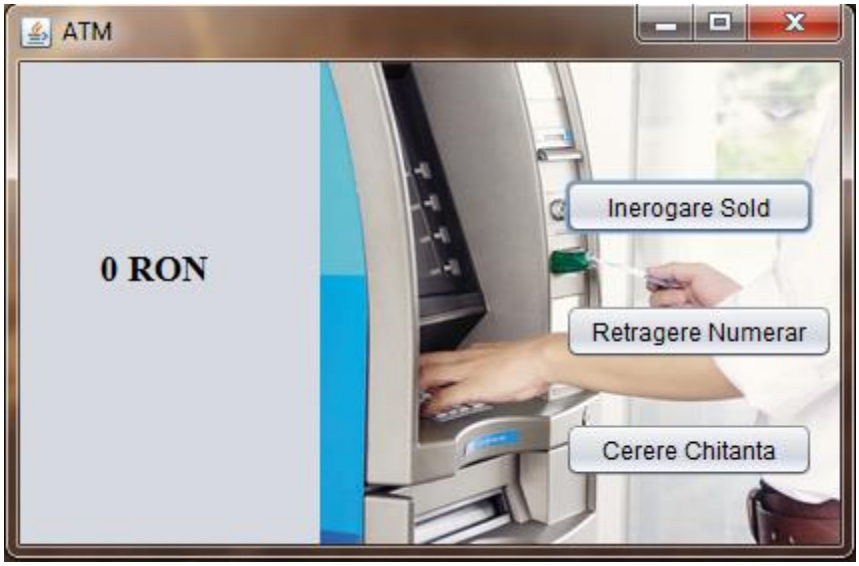


Interfata grafica:

La pornirea programului se va afisa o fereastra de logare in contul propriu, care este folosit ca un card bancar imaginar, adica foloseste IBAN-ul ca si nume de utilizator si PIN-ul cardului pentru autentificare.



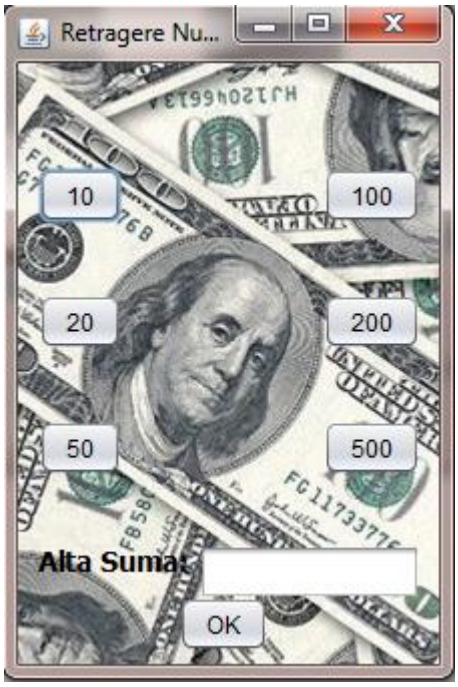
Dupa autentificarea utilizatorului cu succes (adica codul IBAN si PIN – ul corespund), apara fereastra pentru automatul bancar. Utilizatorul poate sa retraga bani din contul sau propriu , poate sa vizualizeze banii din contul curent sau poate sa ceara chitanta cu datele contului curent.



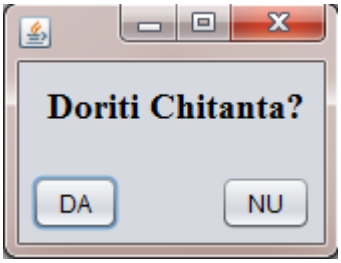
Daca utilizatorul apasa butonul Interogare sold, in loc de 0 RON sa va afisa banii din contul propriu. Apasand butonul Cerere Chitanta, automatul bancar va genera o chitanta care contine IBAN, data de cerere, numarul bancomatului si suma de bani pe care o are pe acest cont.

Daca proprietarul contului apasa butonul Retragere Numerar, se va deschide o noua fereastra pentru introducerea sumei dorite care poate sa fie o suma de 10 lej, 20 de lej, 50 de lej, 100 de lej, 200 de lej si 500 de lej sau se poate specifica o alta suma, prin introducerea sumei dorite in textFieldul din partea de jos a ferestrei si apasand butonul OK.

Fereastra pentru introducerea sumei dorite:



Dupa apasarea unui buton sau dupa introducerea sumei dorite apare o fereastra de interogare, daca utilizatorul doreste chitanta, atunci apasa butonul DA dupa care se va genera o chitanta de forma:

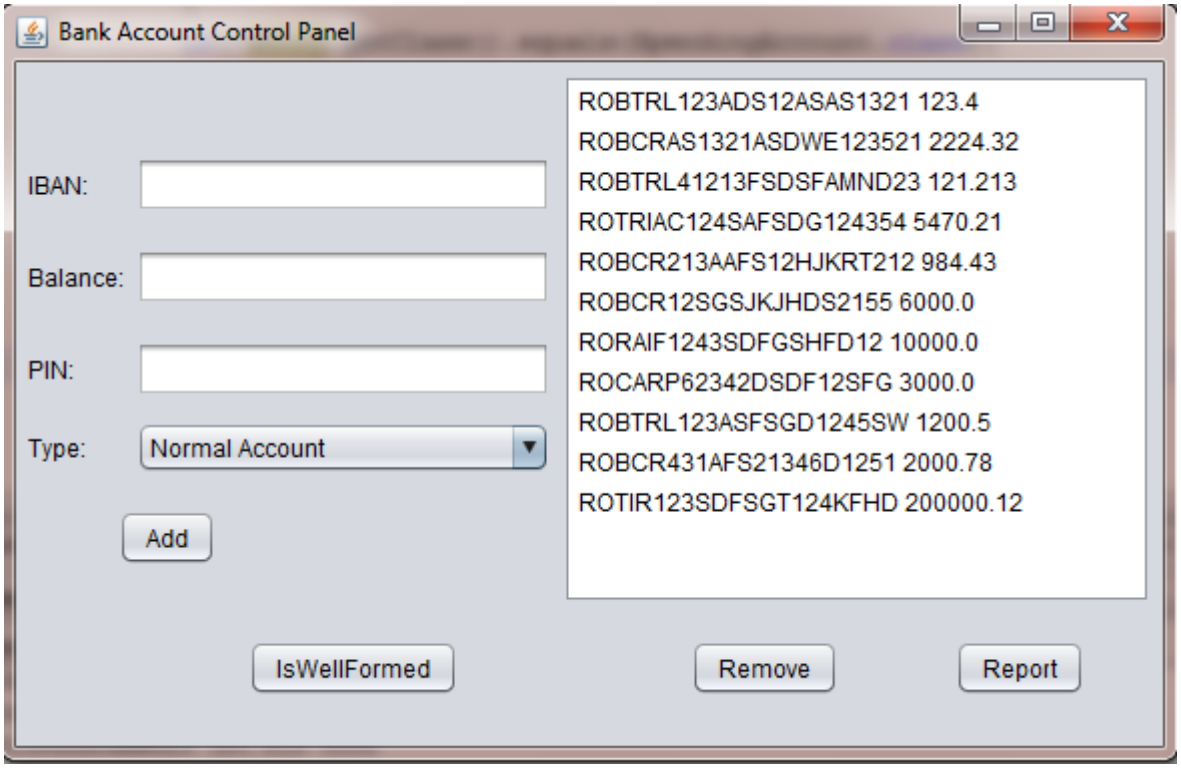


*** BANK RECEIPT ***

DATE: Mon Apr 27 22:00:33 EEST 2015
ATM: 43324882
IBAN ROBTRL123ADS12ASAS1321
NUMERAR RETRAS 10.0
SOLD DISPONIBIL 113.4

Daca contul bancar este de tip Saving Account, atunci utilizatorul nu va putea sa scoata bani din cont pana cenu se expira perioada pentru care banii din cont sunt blocati.

Banca va avea o alta interfata grafica prin care poate sa gestioneze conturile active, poate sa adauge alte conturi la cererea unui utilizator, poate sa stearga un cont bancar sau poate sa genereze un raport al conturilor active, care va contine informatii legate de tipul contului, codul IBAN, si suma de bani pentru fiecare cont.



Daca alegem un cont bancar din lista conturilor si apasam butonul Remove, contul respectiv va fis sters. Daca introducem datele unui cont nou si apasam butonul Add, in cazul in care datele sunt corecte, contul respectiva va fi adaugat la baza de date a bancii (aici la tabela de dispersie). Daca datele nu sunt corecte sau daca codul IBAN exista deja, atunci se va genera o exceptie si va aparea o fereastra de eroare. Daca apasam butonul Report, atucni se va genera un Report cu conturile cancare existente. Acest Report va contine informatii legate de tipul contului, codul IBAN, si suma de bani pentru fiecare cont.

Daca apasam butonul IsWellFormed, atunci programul va verifica daca datele din interiorul bancii sunt corecte sau nu (aduci sa nu avem conturi duplicate sau conturi cu un tip necunoscut sau conturi cu bani mai putini de 0 iar in cazul Spending Account, contul sa nu aiba mai putini bani decat -200 (- inseamna ca am facut imprumut, am cheltuit mai mult decat aveam)

Metoda pentru „well formed”:

```
public boolean wellFormed() throws WrongAccountTypeException, DuplicatedAccountException
   {
      int i;
      Account[] table = bank.getTable();

      for(i=0;i<bank.sizeOfHash();i++)
      {
         Account entry = table[i];
```

```
        while(entry!=null)
        {
           if( !(entry instanceof Account) && !(entry instanceof SavingAccount) && !(entry instanceof
SpendingAccount))
           {
              throw new WrongAccountTypeException("An account has wrong type!");
           }
           if(!entry.getClass().equals(SpendingAccount.class))
           {
              if(entry.getBalance()<-200)
                 return false;
           }
           else
           {
              if(entry.getBalance()<0)
                 return false;
           }
           entry = entry.getNext();
        }

        Account toVerify = table[i];
        Account verificator = table[i];

        while(toVerify!=null)
        {
           while(verificator!=null)
           {
              if(!toVerify.equals(verificator))
                 if(toVerify.getIBAN().equals(verificator.getIBAN()))
                    throw new DuplicatedAccountException("Duplicated IBAN!");
              verificator = verificator.getNext();
           }
           toVerify = toVerify.getNext();
        }
     }

     return true;
  }
```

Aceaste metoda parcurge toata tabela de dispersie, si pentru fiecare intrare care nu este null, adica pentru fiecare cont existent verifica daca contul respectiv e cont normal, cont de economisire sau e cont de cheltuieli. Daca contul nu este de niciun tip de conturi posibile, atunci se va genera o exceptie de Wrong Account Type. Tot aceasta metoda verifica si faptul ca intr-un cont normal sa de economisire nu putem avea bani mai putini de 0 sau intr-un cont de cheltuieli nu putem aveam bani mai putini de -200, adic nu putem imprumuta mai multi bani decat 200 de Ron. Tot aici se verifica daca avem sau nu IBAN-uri duplicate iar in cazul in care avem se va genera o exceptie de tip Duplicated Account Exception.

Report-ul este un pdf care arata astfel:

BANK ACCOUNT REPORT

1. IBAN: ROBTRL123ADS12ASAS1321 BALANCE: 123.4
2. IBAN: ROTRIAC124SAFSDG124354 BALANCE: 5470.21
3. IBAN: ROBTRL123ASFSGD1245SW BALANCE: 1200.5
4. IBAN: ROBCR213AAFS12HJKRT212 BALANCE: 984.43
5. IBAN: ROTIR123SDFSGT124KFHD BALANCE: 200000.12
6. SAVING ACCOUNT IBAN:ROCARP62342DSDF12SFG; BALANCE:3000.0; START DATE:Mon Apr 27 10:31:48 EEST 2015; END DATE:Fri Nov 13 10:31:48 EET 2015
7. SAVING ACCOUNT IBAN:RORAIF1243SDFGSHFD12; BALANCE:10000.0; START DATE:Mon Apr 27 10:31:48 EEST 2015; END DATE:Fri Nov 13 10:31:48 EET 2015
8. IBAN: ROBTRL41213FSDSFAMND23 BALANCE: 121.213
9. SAVING ACCOUNT IBAN:ROBCR12SGSJKJHDS2155; BALANCE:6000.0; START DATE:Mon Apr 27 10:31:48 EEST 2015; END DATE:Fri Nov 13 10:31:48 EET 2015
10. IBAN: QASFDS325FSG346 BALANCE: 12356.0
11. IBAN: ROBCRAS1321ASDWE123521 BALANCE: 2224.32
12. IBAN: ROBCR431AFS21346D1251 BALANCE: 2000.78

4. Testare

Aceasta aplicatie a fost testata de mai multe ori. Testarea am facut prin introducerea manuala a datelor, verificand si cazurile particulare (de exemplu daca am introdus conturi duplicate, atunci mi-a dat eroare, daca am vrut sa retrag mai multi bani decat aveam, am primit eroare, daca am vrut sa scot bani dintr-un cont de tip Saving Account atunci, daca data de sfarsit si data incercarii de retragere bani nu coincid sau data de sfarsit nu este mai mica, atunci nu am putut sa scot bani.) iar rezultatul testarii a fost intotdeauna conform realitatii, fara greseli.

Pentru o testare mai buna am facut aple si la alte persoane, care au incercat utilizarea aplicatiei cautand greseli, iar daca au aparut ceva greseli pe care pana atunci nu am luat in considerare, pe care pana atunci nu am gasit, am reformulat codul pentru corectarea acestor greseli. In prezent aplicatia merge fara greseli.

5. Rezultate

După testarea programului vedem că programul functionează perfect, verifică datele de intrare , afișează rezultatele pe interfața grafică, si efetuează comenzile primite de la utilizatorul programului.

6. Concluzii, ce s-a invatat din tema, dezvoltari ulterioare

Lucrand la acest proiect am invatat am invatat un stil nou de programare, adica stilul Design by Contract Programming. Am invatat cum sa ma gandesc daca am deja diagrama UML si trebuie sa implementez aplicatia conform cerintelor, conform diagramei. Am vazut cum functioneaza Assert-ul in java, inteles si am vazut avantajele folosirii preconditiilor si postconditiilor, care ne ajuta in creearea unei aplicatii cat mai corecte din

punct de vedere a functionalitatii precum si din punct de vedere a logicii si a cerintelor de Bussiness si Management. Am invatat sa creez exceptii proprii, si am inteles mai bine folosirea exceptiilor si avantajele lor. Pe langa acestea am mai invatat folosirea expresiilor regulate, folosirea Regex-ului in java.

7. Bibiliografie

Carti:

Thinking in java – fourth edition Bruce Eckel President, MindView, Inc.

Effective Java™
Second Edition

Internet :

https:// www. youtube . com / - turoriale

http:// stackoverflow . com /

http:// docs. oracle. com / javase / tutorial /