**APLICAȚIE BANCARĂ**

**Nume prenume: Dulău Marius Cristian**

**Grupa: 30225**

**Profesor laborator: Asist. Antal Marcel**

Contents

[1. Cerințe funcționale 3](#_Toc476131445)

[2. Obiective 3](#_Toc476131446)

[2.1. Obiectiv principal 3](#_Toc476131447)

[2.2. Obective secundare 3](#_Toc476131448)

[3. Analiza problemei](#_Toc476131449) 4

[4. Proiectare](#_Toc476131450) 5

[4.1. Structuri de date](#_Toc476131451) 5

[4.2. Diagrama de clase](#_Toc476131452) 5

[4.3. Algoritmi](#_Toc476131453) 6

[5. Implementare](#_Toc476131454) 12

[6. Testare](#_Toc476131455) 14

[7. Concluzii si dezvoltări ulterioare](#_Toc476131456) 18

[8. Bibliografie](#_Toc476131457) 19

# Cerințe funcționale

Dezvoltați o aplicație care să conturile bancare ale unor persoane. Aplicația are următoarele caracteristici:

* Adaugă persoane în sistem
* Adaugă unul sau mai multe conturi pentru o anumită persoana
* Șterge o persoană
* Șterge minim un cont al unei persoane
* Afișează persoanele și conturile într-un tabel
* O persoană poate adauge sau să retragă numerar

# Obiective

## Obiectiv Principal:

1. Definiți interfața BankProc (adăugați / eliminați persoane, adăugați / eliminați conturi asociate titularilor, citiți / scrieți date contabile, generatoare de rapoarte etc.). Specificați pre și post condițiile pentru metodele de interfață.
2. Proiectați și implementați clasele Person, Account, SavingAccount și SpendingAccount. Alte clase pot fi adăugate după cum este necesar (dați motive pentru noile clase adăugate).
3. Va fi definit și implementat un observator DP. Acesta va notifica titularului contului despre orice operațiune legată de cont.
4. Implementați clasa Bank utilizând o colecție predefinită care utilizează un hashtable. Cheia hashtable-ulului va fi generată pe baza titularului contului. O persoană poate acționa în calitate de titular principal pentru multe conturi. Utilizați JTable pentru a afișa informații legate de bancă.
5. Proiectați și implementați un driver de testare pentru sistem.
6. Datele contului pentru popularea obiectului băncii vor fi încărcate / salvate din / într-un fișier.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri și scenarii | Descrierea în limbaj natural a scenariilor de utilizare și funcționare aplicației, însoțite de o diagramă. | 3 |
| Alegerea structurilor de date | Pentru a construi un tabel a fost nevoie de o un HashMap pentru a stoca toate persoanele și conturile asociate lor. | 4 |
| Împărțirea pe clase | Clasele au fost structurate în mai multe pachete. Aceste pachete sunt : model, MyException, presentation, start și testare. | 5 |
| Dezvoltarea algoritmilor | S-au dezvoltat algoritmi pentru selectare, inserare, modificare și ștergere de date. De asemenea au fost implementați algoritmi pentru a genera un tabel cu date. | 5 |
| Implementarea soluției | Implementare interfață grafică și a metodelor specifice operațiilor pentru persoane și conturile asociate. | 5 |
| Testare | Testarea efectuării operațiilor în interfața grafică și cu Junit. | 6 |

# Analiza Problemei

Aplicația are un singur actor : administratorul. Acesta poate insera persoane, să le modifice datele, să le scoată din sistemul de gestiune al conturilor, să creeze sau să șteargă conturi asociate persoanelor titulare. De asemenea, administratorul mai poate efectua tranzacții ale clienților: depunere sau retragere numerar. Acesta mai are la dispoziție un tabel prin care poate vizualiza persoanele și conturile aflate în sistem.

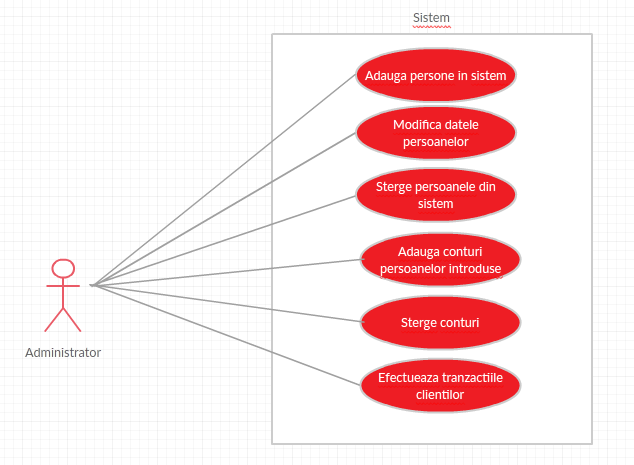


Fig. 1. Diagrama use case

# Proiectare

## Structuri de date

În cadrul aplicației s-a utilizat interfața Map implementată ca HashMap. În Java, un Map este o colecție de chei K cu valori V. Clasa Map are mai multe implementări, în funcție de structura de date utilizată:

* Hash table: HashMap, Hashtable, LinkedHashMap
* Red-black trees: TreeMap

HashMap-ul utilizat în aplicație are ca și cheie persoana, iar ca valoare contul sau conturile aferente persoanei. Pentru a putea utiliza HahMap-ul în aplicație a fost necasar ca metodele equals() și hashCode() să fie suprescrise.

## Diagrama de clase

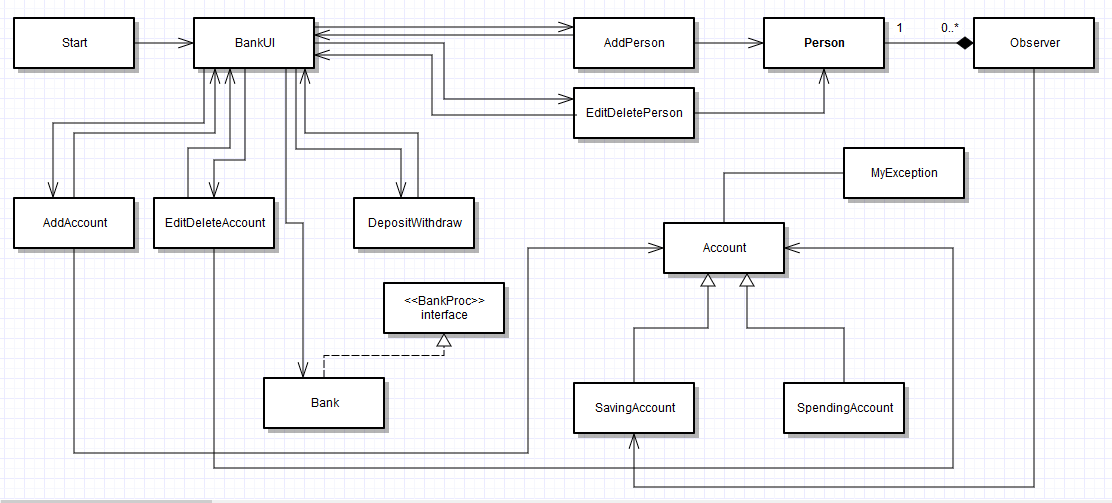
****

Fig. 2. Diagrama UML

## Algoritmi

La HashMap metodele de equals() și hashCode() au trebuit suprascrise.

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Bank other = (Bank) obj;

**if** (*map* == **null**) {

**if** (other.*map* != **null**)

**return** **false**;

} **else** **if** (!*map*.equals(other.*map*))

**return** **false**;

**return** **true**;

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + ((*map* == **null**) ? 0 : *map*.hashCode());

**return** result;

}

În clasa Bank s-au implementat metode pentru a efectua operații cu HashMap. Codul acestor metode este prezentat mai jos.

@Override

**public** **void** addPerson(Person p) {

// **TODO** Auto-generated method stub

**assert** p != **null** : "Parameter is null!";

**assert** p.getPersonCNP().length() == 13 : "CNP must have 13 digits";

*map*.put(p, **null**);

**assert** *map*.containsKey(p) : "Map does not contain your parameter";

}

@Override

**public** **void** addAccount(Account a) {

// **TODO** Auto-generated method stub

**assert** a != **null** : "Your account is null";

**assert** a.getBalance() >= 0 : "Balance must be positive";

Set<Account> mySet = *map*.get(a.getAccountHolder());

**if**(mySet == **null**)

{

mySet = **new** HashSet<Account>();

*map*.put(a.getAccountHolder(), mySet);

}

mySet.add(a);

**assert** mySet.contains(a) : "Account is not inserted!";

}

@Override

**public** **void** removePerson(Person p) {

// **TODO** Auto-generated method stub

**assert** *map*.containsKey(p) : "Map does not contain parameter p";

*map*.remove(p);

**assert** !*map*.containsKey(p) : "Map contain parameter p";

}

@Override

**public** Person findPerson(String cnp) {

// **TODO** Auto-generated method stub

**assert** cnp.length() == 13 : "CNP must have 13 digits";

Person p2 = **null**;

**for**(Person p : *map*.keySet())

{

**if**(p.getPersonCNP().compareTo(cnp) == 0)

{

p2 = p;

**break**;

}

}

**assert** *map*.containsKey(p2) : "Map does not contain person with your given CNP";

**assert** p2 != **null** : "Map contains null object";

**return** p2;

}

@Override

**public** **int** getMaxAccountId() {

// **TODO** Auto-generated method stub

**int** nr = 0;

**for**(Person p : *map*.keySet())

{

HashSet<Account> a = (HashSet<Account>) *map*.get(p);

**if**(a != **null**)

{

**for**(Account i : a)

{

**if**(i.getAccountId() > nr) nr = i.getAccountId();

}

}

}

**assert** nr >= 0 : "Negative id";

**return** nr;

}

@Override

**public** HashSet<Account> readAccounts(Person p) {

// **TODO** Auto-generated method stub

**assert** p != **null** : "Parameter is null!";

**assert** *map*.containsKey(p) : "Map does not contain Person p";

HashSet<Account> a = **null**;

**for**(Person p2 : *map*.keySet())

{

**if**(p2.equals(p)) a = (HashSet<Account>) *map*.get(p);

}

**return** a;

}

@Override

**public** **void** removeAccount(Account a) {

// **TODO** Auto-generated method stub

**assert** a != **null** : "Account is null";

Set<Account> mySet = *map*.get(a.getAccountHolder());

**assert** mySet.contains(a) : "Map contain the parameter account";

mySet.remove(a);

**assert** !mySet.contains(a) : "Map still contain the parameter account";

}

Mai jos este prezentat codul pentru generarea tabelului ce prezintă datele despre persoane și conturi. Acest tabel este creat folosind tehnici de reflection.

**public** **static** JTable createTable(Object obj)

{

**int** nrLinii = 0, nrColoane = 0;

**if**(obj **instanceof** Person) nrColoane = 4;

**else** **if**(obj **instanceof** Account) nrColoane = 5;

**if**(obj **instanceof** Person) nrLinii = *map*.keySet().size();

**else** **if**(obj **instanceof** Account)

{

**for**(Person p : *map*.keySet())

{

**if**(*map*.get(p) != **null**) nrLinii += *map*.get(p).size();

}

}

Object[] columnNames = **new** Object[nrColoane];

Object[][] rowData = **new** Object[nrLinii][nrColoane];

**int** index = 0;

**if**(obj **instanceof** Person)

{

columnNames[index++] = "CNP";

columnNames[index++] = "First Name";

columnNames[index++] = "Last Name";

columnNames[index++] = "Address";

}

**else** **if**(obj **instanceof** Account)

{

columnNames[index++] = "Account Id";

columnNames[index++] = "Holder CNP";

columnNames[index++] = "Balance";

columnNames[index++] = "Account Type";

columnNames[index++] = "Interest Rate";

}

index = 0;

**for**(Person p : *map*.keySet())

{

**if**(obj **instanceof** Person)

{

rowData[index][0] = p.getPersonCNP();

rowData[index][1] = p.getFirstName();

rowData[index][2] = p.getLastName();

rowData[index][3] = p.getAddress();

index++;

}

**else** **if**(obj **instanceof** Account)

{

HashSet<Account> a = (HashSet<Account>)*map*.get(p);

**if**(a != **null**)

{

**for**(Account i : a)

{

rowData[index][0] = i.getAccountId();

rowData[index][1] = i.getAccountHolder().getPersonCNP();

rowData[index][2] = i.getBalance();

**if**(i **instanceof** SavingAccount)

{

rowData[index][3] = "Saving Account";

rowData[index][4] = ((SavingAccount) i).getInterestRate();

}

**else** **if**(i **instanceof** SpendingAccount)

{

rowData[index][3] = "Spending Account";

rowData[index][4] = ((SpendingAccount) i).getInterestRate();

}

index++;

}

}

}

}

JTable table = **new** JTable(rowData, columnNames);

**return** table;

}

Aplicația este construită pe principiul MVC (Model View Controller). A fost necesar crearea de clase ce implementează ActionListener. În continuare este prezentată clasa ce șterge o persoană din Map.

**class** DeleteActionListener **implements** ActionListener

{

@Override

**public** **void** actionPerformed(ActionEvent e) {

// **TODO** Auto-generated method stub

Person p = b.findPerson(editDeletePerson.getCNP());//se preia cnp-ul si se cauta persoana

//daca e exista in Map se sterge si se reimprospateaza tabelul

**if**(p != **null**)

{

b.removePerson(p);

refreshTablePerson();

editDeletePerson.setVisible(**false**);

editDeletePerson.dispose();

}

}

}

# Implementare

Aplicația este împățită în 5 pachete:

* model – clase ce ilustrează principalele obiecte utilizate în aplicație;
* MyException – conține o clasă ce este utilizată pentru a arunca excepții cu un mesaj personalizat
* presentation – pachetul conține clase ce construiesc interfața grafică
* start – acest pachet conține clasa cu metoda main() a proiectului
* testare – conține o clasă destinaată testării

Pachetul model conține 5 clase și o interfață. Interfața BankProc are următoarele metode ce vor fi implementate în clasa Bank:

* **void** addPerson(Person p) – introduce o nouă persoană în colecția Map
* Person findPerson(String cnp) – returnează un obiect de tip Persoană; acest obiect este căutat după cnp-ul (id-ul) persoanei
* **void** removePerson(Person p) – șterge persoana dată ca parametru din colecția Map
* **void** addAccount(Account a) – introduce cont nou în colecția Map, corespunzător cheii ce reprezintă titularul de cont; clasa Account are o metodă ce returnează cine e titularul
* **void** removeAccount(Account a) șterge contul dat ca parametru din colecția Map
* HashSet<Account> readAccounts(Person p) – metoda returnează toate conturile persoanei date ca și parametru
* **int** getMaxAccountId() – returnează cel mai mare id al unui cont din colecția Map.

Clasa Bank are un singur câmp:

**private** **static** Map<Person, Set<Account>> *map* : interfața Map este implementată ca și HashMap. HashMap-ul utilizat în aplicație are ca și cheie persoana, iar ca valoare contul sau conturile aferente persoanei.

Clasa Person are următoarele câmpuri:

* **private** String personCNP – id-ul persoanei este prorpiul cod numeric personal
* **private** String firstName – prenumele persoanei
* **private** String lastName – pnumele persoanei
* **private** String address – adresa persoanei

Clasa Person are ca și metode gettere și settere.

Clasa Account are următoarele câmpuri:

* **private** **int** accountId – id-ul contului
* **private** Person accountHolder – titularul contului este reprezentat printr-un obiect de tip Person
* **private** **double** balance – soldul bancar

Clasa Account are ca și metode gettere și settere.

Clasele SavingAccount și SpendingAccount moștenesc clasa Account. Ele mai au în plus câteva cîmpuri. Clasa SavingAccount are în plus următoarele câmpuri:

* **private** **double** minBalance – suma minimă de bani ce trebuie să rămână în cont
* **private** **double** interestRate – dobânda
* **private** **double** initialBalance – suma inițială; pe baza acestui câmp se calculeaza valoare ce se adaugă la cont ân urma dobînzii

Clasa SpendingAccount are în plus doar câmpul interestRate.

În capitolul următor este prezentată testarea aplicației, precum și interfața grafică rezultată din clasele pachetului **presentation**.

# Testare

Pentru a urmări corectitudinea simulării, testarea a avut loc observând ceea ce se întâmplă în interfața grafică.

Aplicația conține în partea stângă butoane cu ajutorul cărora se efectuează operațiile de inserare/ modificare/ ștergere, iar în partea dreaptă un tabel în care pot fi vizualizate persoanele și conturile lor (Fig. 3.a, Fig3.b).

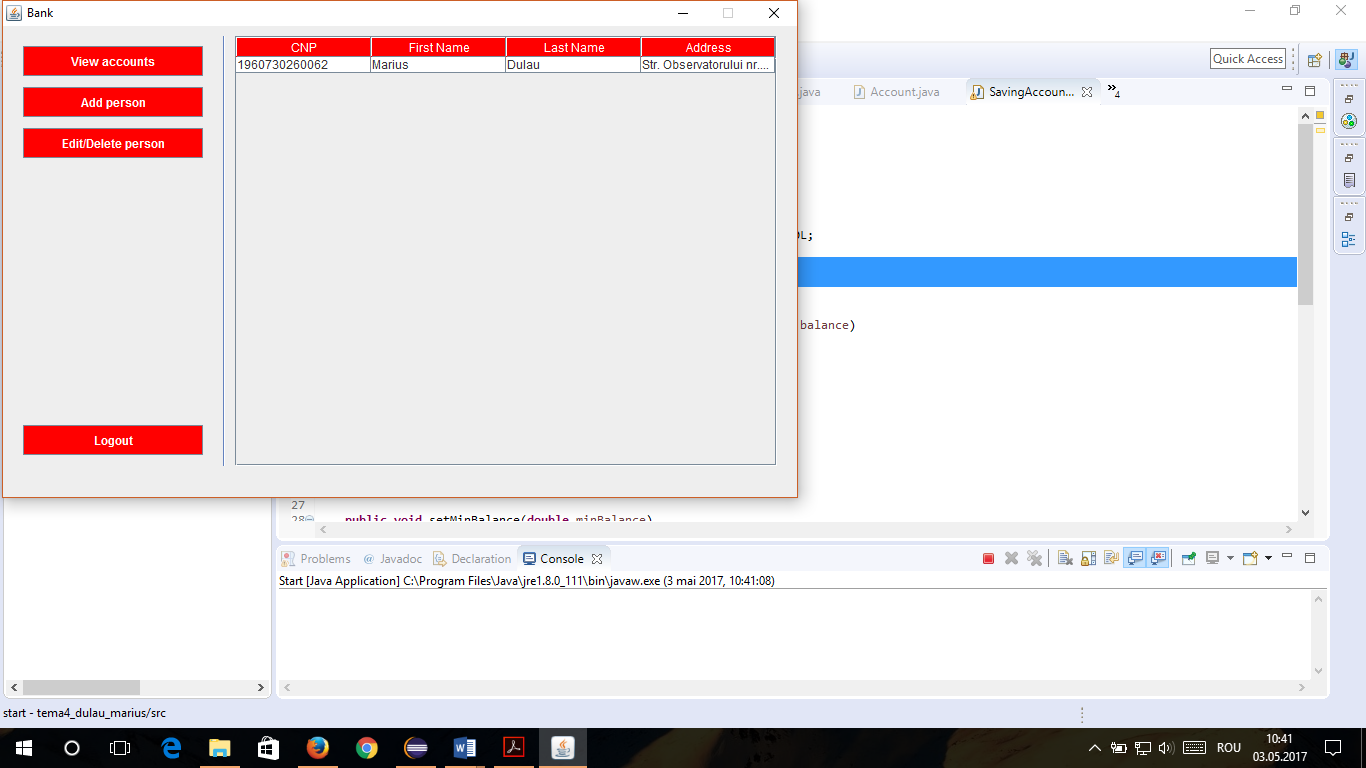


Fig. 3. a. Meniul principal al aplicației în cazul vizualizării persoanelor

Dacă se actionează butonul „View accounts” atunci în table se vor vizualiza conturile(Fig. 3.b), iar meniul se schimbă.

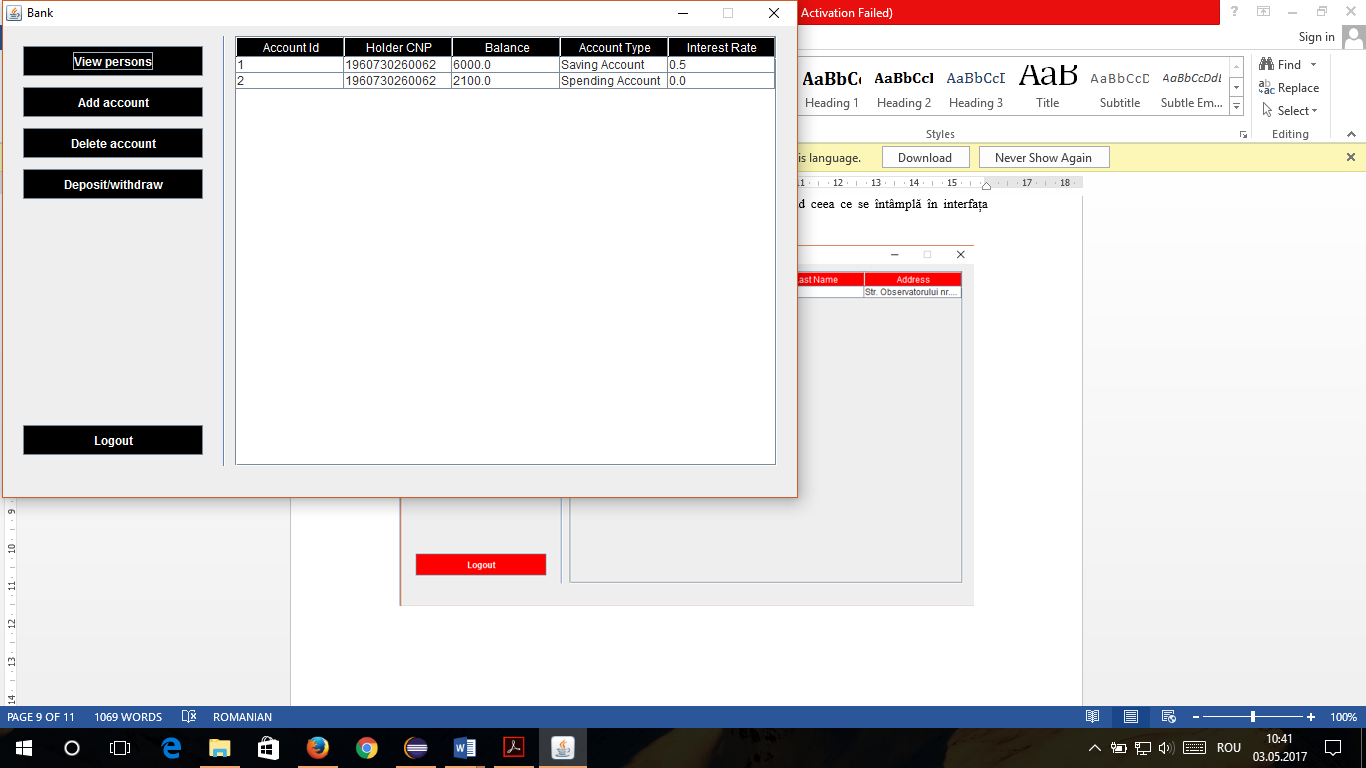


Fig. 3. b. Meniul principal al aplicației în cazul vizualizării conturilor

Dacă se actionează butonul „View persons” atunci în tabel se vor vizualiza persoanele (Fig. 3.a).

Dacă se dă click pe butonul „Add person” va apărea o fereastră de introducere a datelor pentru persoana respectivă(Fig. 4.).

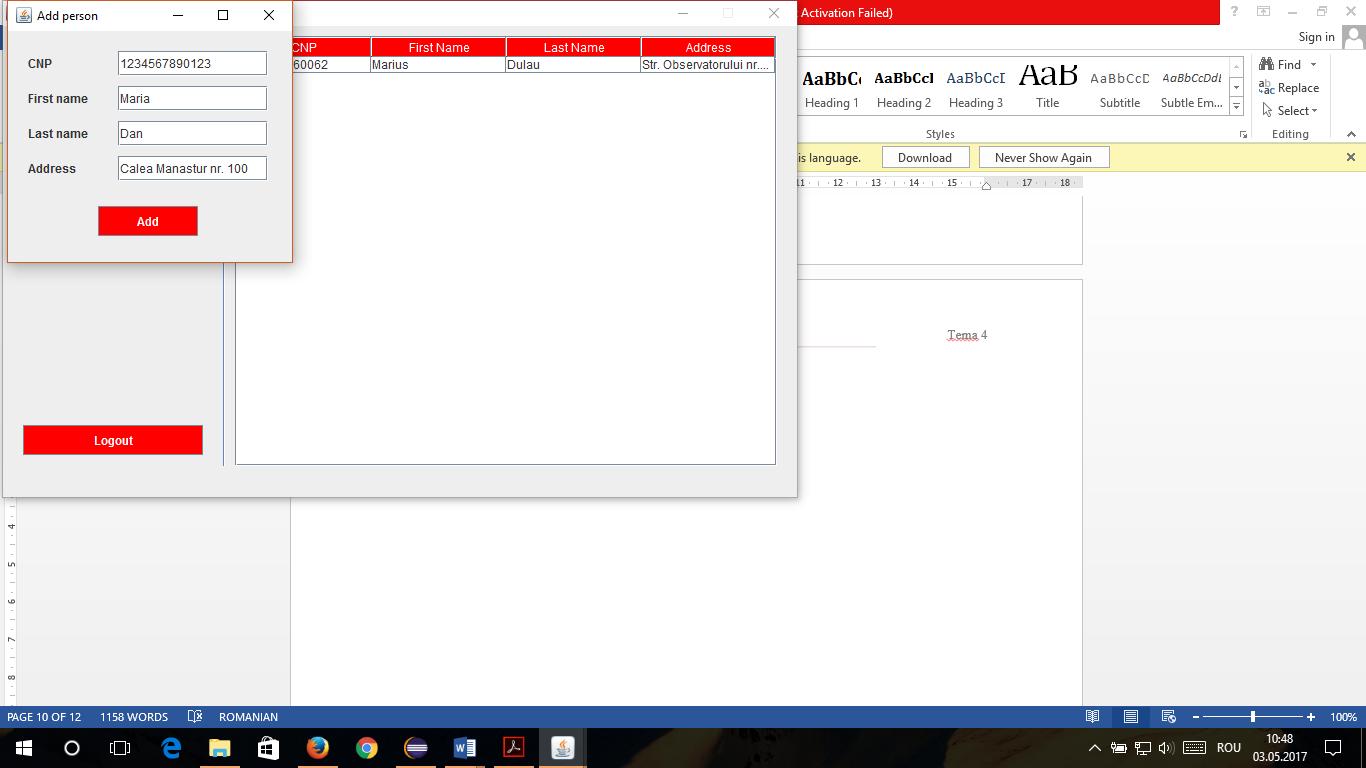


Fig. 4. Fereastra de adăugare a unei persoane

După ce se apasă pe butonul „Add” în tabelul principal vom vedea persoana inserată (Fig. 5).

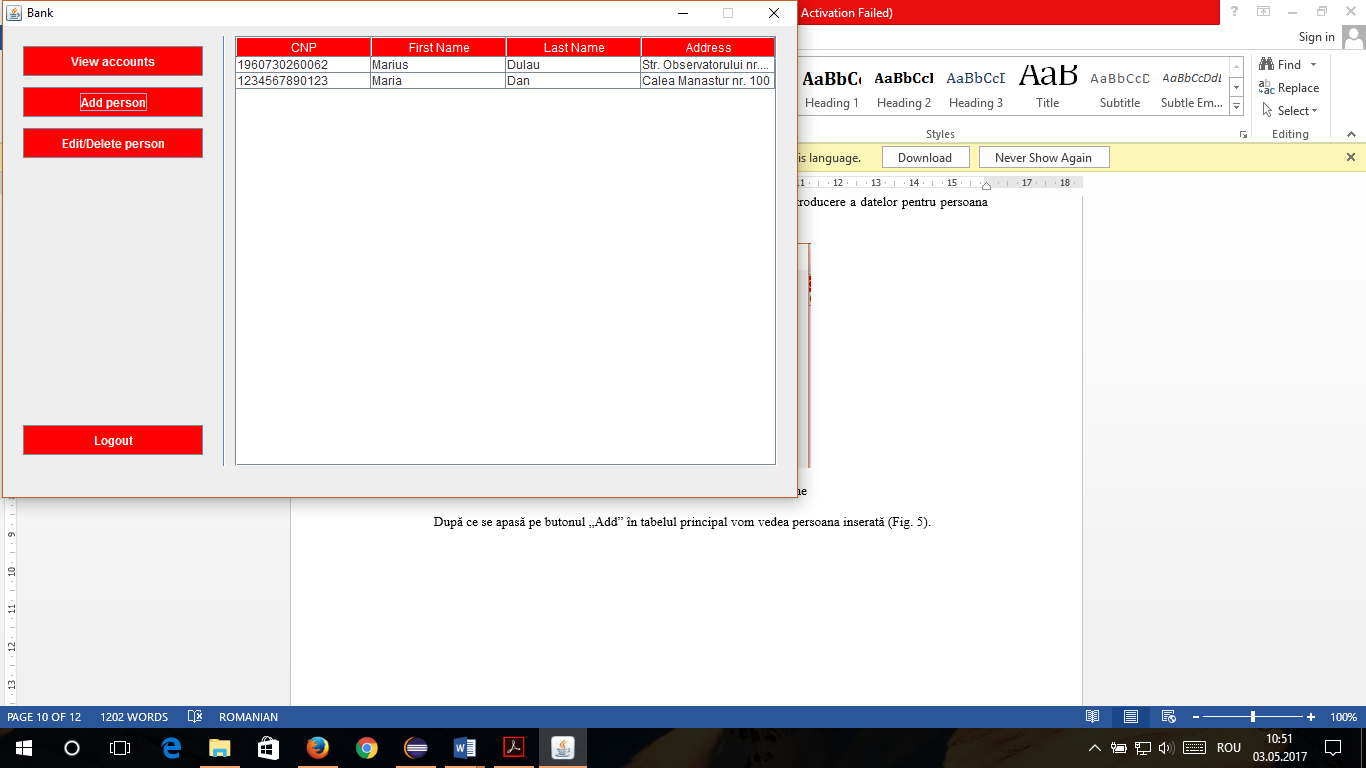


Fig. 5. Persoanele inserate în sistem apar în tabel

Dacă se dă click pe butonul „Edit/Delete person” va apărea o fereastră în care trebuie introdus cnp-ul persoanei. Apoi vor apărea restul datelor ce pot fi modificare (Fig. 6.). Fereastra dispune de opțiunile de modificare a datelor sau ștergerea lor din aplicație.

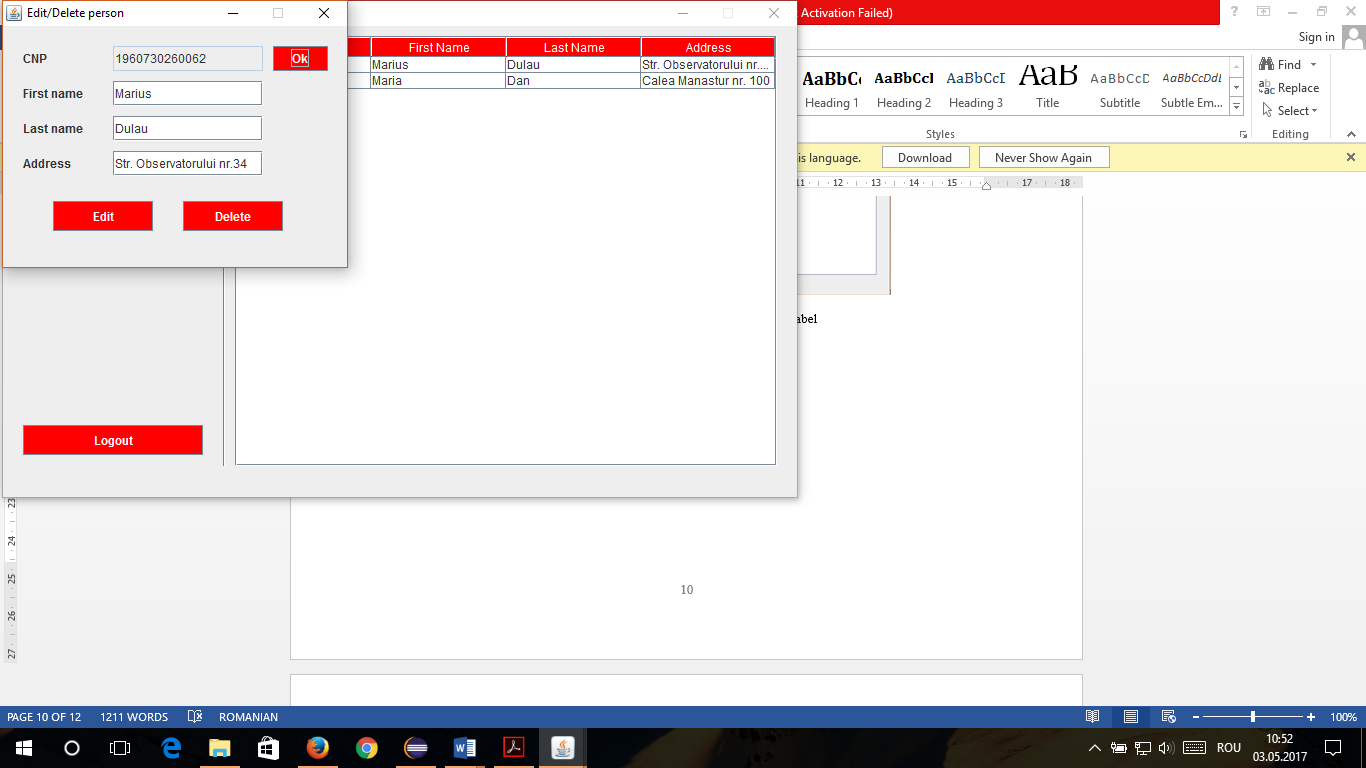


Fig. 6. Fereastra de modificare/ștergere a unei persoane

Analog este și la conturi cu crearea și stergerea. Dacă se dă click pe butonul „Deposit/Withdraw” va apărea o fereastră în care trebuie introdus cnp-ul titularului. Dacă cnp-ul este corest se vor afișa detaliile contului, precum și suma pe care titularul dorește să o depună sau să o retragă (Fig. 7).

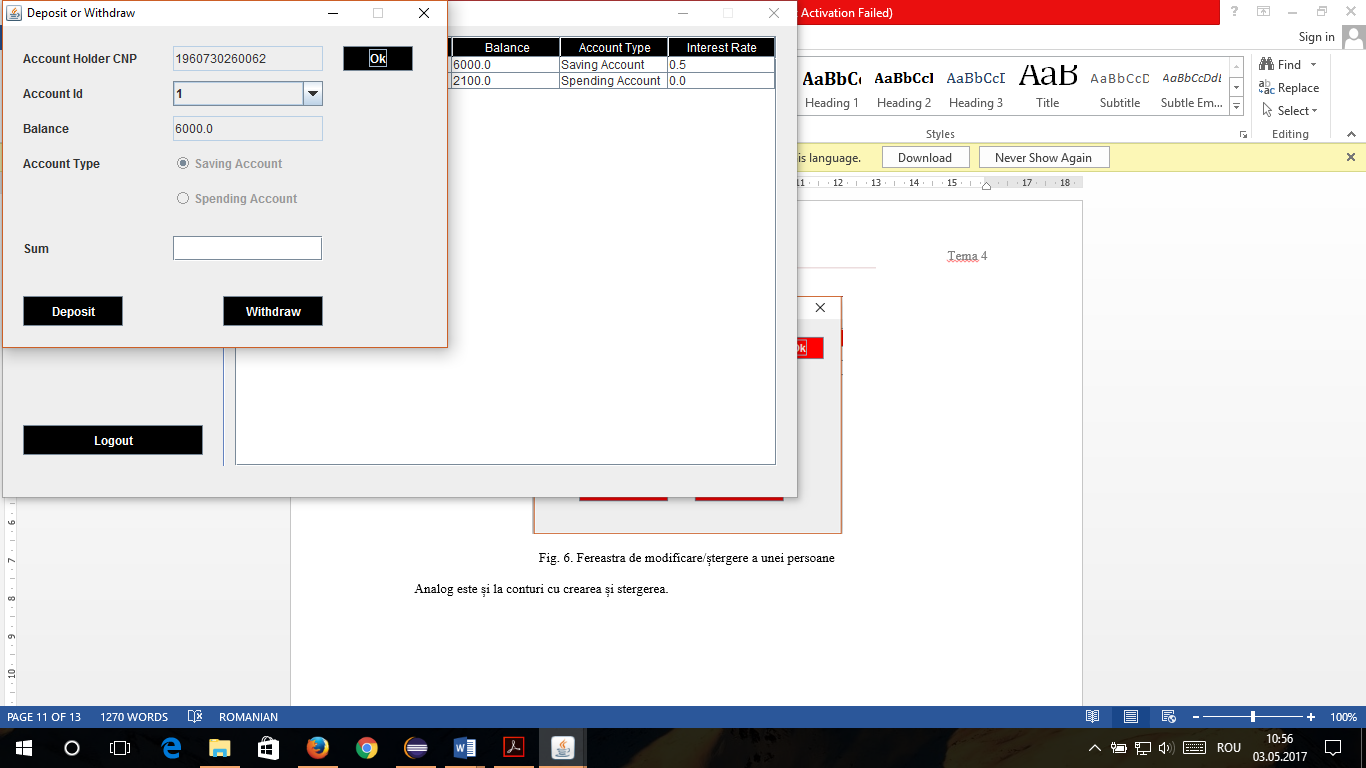


Fig. 7. Fereastra de depunere/retragere numerar

Pentru testare cu Junit s-a folosit o clasă de test. În interiorul acestei clase s-au declarat variabile pentru a efectua operațiile necesare. S-au efectuat teste pentru introducerea unei persoane cu parametrii valizi, obiect persoană null, dacă se găsește o persoană după codul numeric personal sau pentru a detecta dacă o persoană ștearsă din Map mai există. De asemenea s-au mai efectuat teste pentru id-ul max din Map, adăugarea unui cont și numărul de conturi ale unei persoane.

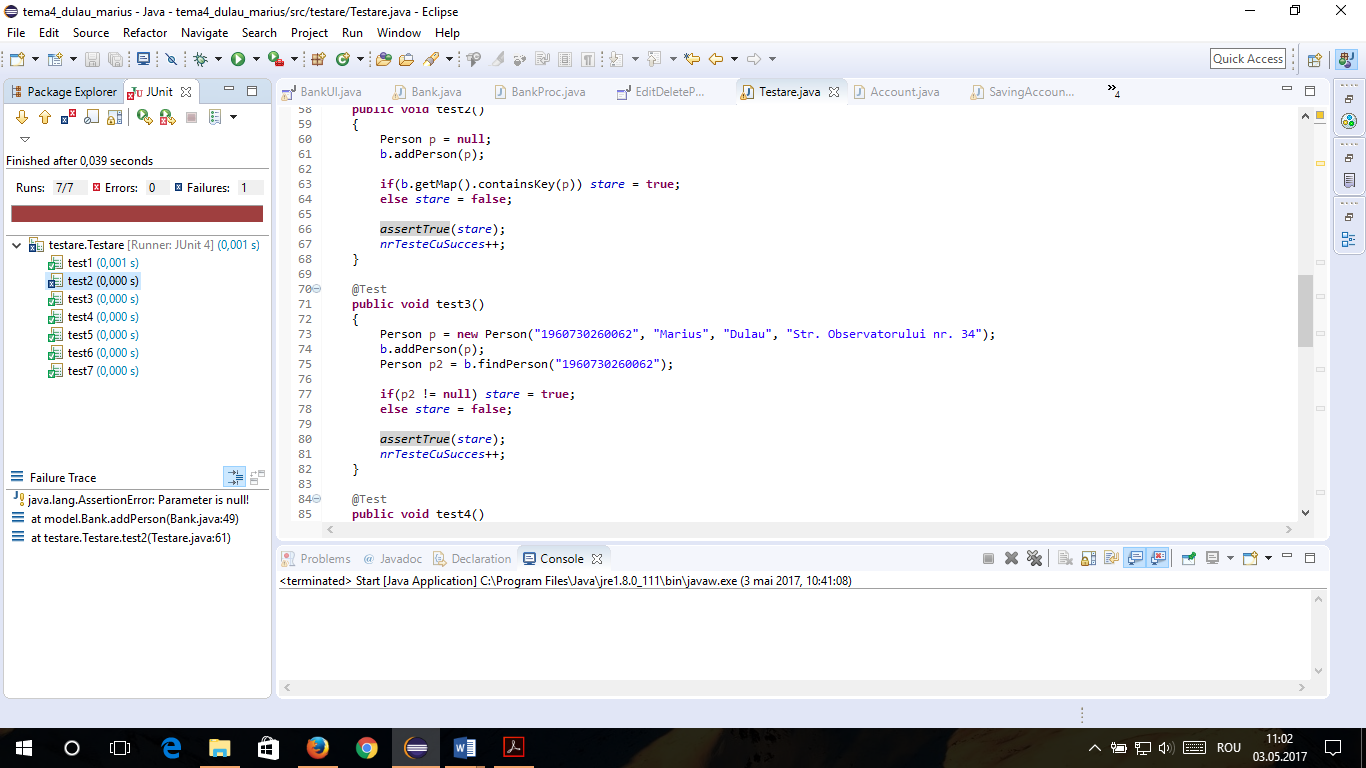


Fig. 8. Rezutatul testării cu JUnit

Testul 2 a eșuat pentru că s-a testat inserarea unui obiect null, iar clasa Bank în metoda de inserare are o linie de assert ce verifică acest lucru.

# Concluzii și dezvoltări ulterioare

În urma dezvoltării acestui proiect am învățat să utilizez serializarea, precum pre și post condițiile. În cazul serializării a fost necesar ca obiectul Map din clasa Bank să fie scris într-un fișier. În urma scrierii, atunci când aplicația pornește se citește fișierul și se preiau valorile ce se aflau în colecție înainte de închiderea anterioară a aplicației. De asemenea, am folosit pattern-ul Observer în proiect.

Printre dezvoltările ulterioare se numără:

* Accesul la aplicație să fie pe bază de cont
* Implementarea de Internet Banking
* Unele clase care implemetează ActionListener să fie realizate folosind reflection
* Trimiterea unui mail cu soldul clientului după fiecare tranzacție
* Migrarea aplicației din spațiul desktop în spațiul web

# Bibliografie

<https://www.mkyong.com/java/how-to-write-an-object-to-file-in-java/>

<http://stackoverflow.com/questions/2257309/how-to-use-hashmap-with-jtable>

<http://stackoverflow.com/questions/7778958/how-can-i-change-jtables-header-background-color>

<http://stackoverflow.com/questions/5876278/using-openfileoutput-in-a-class-not-an-activity>

<http://stackoverflow.com/questions/4956844/hashmap-with-multiple-values-under-the-same-key>

<http://stackoverflow.com/questions/35993201/finding-number-of-values-in-a-hashmap>

<http://stackoverflow.com/questions/29775469/add-value-to-a-set-inside-a-map>

<http://stackoverflow.com/questions/14852719/double-click-listener-on-jtable-in-java>

<http://stackoverflow.com/questions/3867195/how-to-retrieve-header-values-from-jtable>

<https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>