

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

Aplicatie bancara

Documentație

Ciubotaru Andrei-Mihai

Grupa: 30227

Cuprins

1. Obiectivul temei
2. Analiza problemei
3. Proiectare
 - 3.1 Diagrama de clase
4. Implementare
 - 4.1 Clase si pachete
 - 4.2 GUI
5. Testare
6. Rezultate
7. Concluzii
8. Bibliografie

1.Obiectivul temei

Obiectivul acestei teme este sa realizam o aplicatie care simuleaza o banca.

Aplicatia va permite adaugarea, editarea si stergerea clientilor sau conturilor si depunerea sau retragerea de numerar.

Aceasta va dispune de o interfata „User Friendly”, care poate fi utilizata de oricine.

2.Analiza problemei

In cadrul bancii o persoana poate avea unul sau mai multe conturi. Conturile pot fi de doua tipuri: SavingAccount si SpendingAccount. Diferenta intre cele doua tipuri este ca la SavingAccount se salveaza si dobanda, se poate sefectua o singura depunere si retragere, in timp ce la SpendingAccount se pot efectua oricate depuneri si retrageri in limita fondului disponibil.

Pentru a salva datele despre persoane si conturi am folosit HashMap. Astfel persoana este cheia, iar valoarea este reprezentata de o lista de conturi. Pentru hashCode am folosit id-iul persoanei.

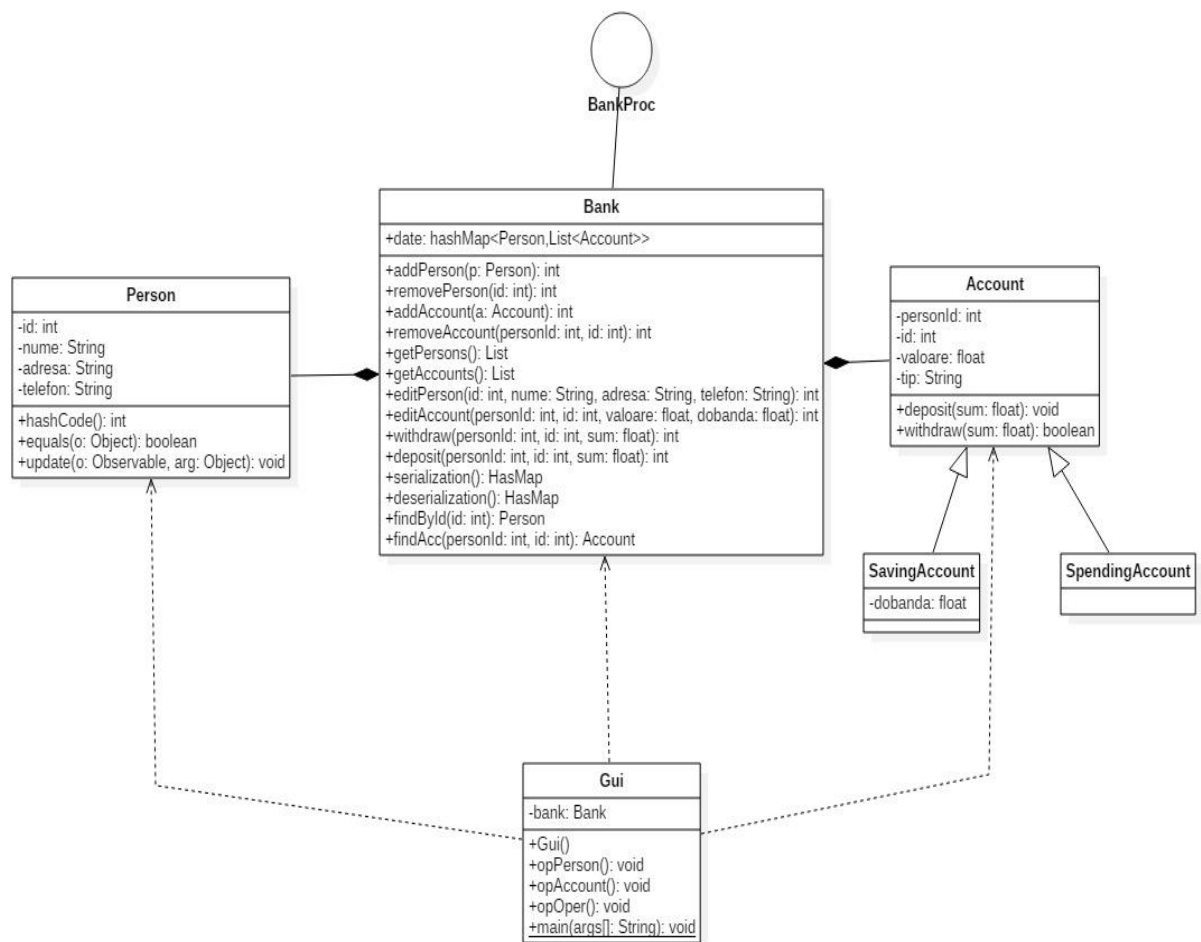
Pentru stocarea datelor am folosit fisiere seriale astfel atunci cand se porneste aplicatia datele vor fi deserializate dintr-un fisier, iar la sfarsit dupa ce se vor efectua operatii asupra datelor vor fi salvate in acelasi fisier.

3.Proiectare

Pentru a proiecta aplicatia am ales o arhitectura layered si am utilizat doua design pattern-uri: Design by contract: am declarat pre si post conditii in interfata BankProc si cu ajutorul assert-urilor le-am testat in clasa Bank, si Design Pattern Observer pentru a notifica o persoana atunci cand se petrece o modificare asupra unui cont de exemplu cand se depun sau se retrag bani.

Diagrama UML presupune modelarea unui sistem prin lucrurile care sunt importante pentru acesta. Aceste lucruri sunt modelate folosind clase.

3.1. Diagrama de clase



4.Implementare

4.1.Clase si pachete

Clasa Person are ca si variabile instantia id, nume, adresa si telefon. Clasa implementeaza interfata Serializable pentru a putea realiza salvarea datelor in fisier atunci cand se inchide aplicatia si interfata Observer pentru a realiza o notificare atunci cand se modifica un cont. Am rescris metodele hashCode si equals pentru a se realiza o inserare corecta in HashMap. La fel am facut si pentru metoda update.

```
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Person person = (Person) o;

    return id == person.id;
}

@Override
public int hashCode() {
    return id;
}

public String toString() {
    return "Persoana " + " " + id + " " + nume + " " + adresa + " " + telefon;
}

@Override
public void update(Observable o, Object arg) {
    System.out.println("Persoana " + id + " a efectuat " + arg);
}
```

In aceasta clasa se mai regasesc si settere si gettere.

Am declarat o clasa Account care are ca variabile instantia personId si id (id-ul contului) pentru a putea stii carei persoane ii apartine un anumit cont. In variabila tip este salvat tipul contului, iar in variabila valoare suma care se afla in cont.

Clasele SpendingAccount si SavingAccount mostenesc clasa Account. In plus clasa SavingAccount are variabila dobanda. La randul ei clasa Account extinde clasa Observable si foloseste metodele setChanged() si notifyObservers(„mesaj”) si implementeaza interfata Serializable pentru a se putea efectua salvarea datelor in fisier. In aceasta clasa am implementat metodele deposit si whitdraw care permit depunerea sau retragerea de numerar din cont.

```
public void deposit(float sum) {
    valoare+=sum;
    setChanged();
    notifyObservers("depunere: " + sum);
}
```

```

}

public boolean withdraw(float sum) {
    if(valoare-sum<0)
        return false;
    valoare-=sum;
    setChanged();
    notifyObservers("retragere: " + sum);
    return true;
}

```

In interfata BankProc se afla metodele ce vor fi implemetate de clasa Bank:adaugare, stergere, editare si afisarea pentru perosane si conturi si metodele de withdraw si deposit.

```

public interface BankProc {
    /**
     * @pre p!=null
     */
    int addPerson(Person p);

    int removePerson(int id);

    /**
     * @pre a!=null
     */
    int addAccount(Account a);

    int removeAccount(int personId, int id);

    List getPersons();

    List getAccounts();

    int editPerson(int id, String nume, String adresa, String telefon);

    /**
     * @pre valoare>0 && dobanda>0
     */
    int editAccount(int personId, int id, float valoare, float dobanda);

    /**
     * @pre sum>0
     */
    int deposit(int personId, int id, float sum);

    /**
     * @pre sum>0
     */
    int withddraw(int personId, int id, float sum);
}

```

In clasa Bank sunt implemetate metodele din interfata. Pentru serializare si desrializae am folosit doua metode.

```

public void serialization() {
    try {
        FileOutputStream file = new FileOutputStream("file.ser");
        ObjectOutputStream obj = new ObjectOutputStream(file);
        obj.writeObject(date);
        obj.close();
    }
}

```

```

        file.close();
        System.out.printf("Serialized HashMap data is saved in file.ser");
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public HashMap deserialization() {
    HashMap date = null;
    try {
        FileInputStream file = new FileInputStream("file.ser");
        ObjectInputStream obj = new ObjectInputStream(file);
        date = (HashMap)obj.readObject();
        obj.close();
        file.close();
        System.out.println("Deserialized HashMap data from file.ser");
    } catch (FileNotFoundException e) {
        return null;
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    return date;
}

```

Datele sunt salvate intr-o structura de tipul HashMap (private HashMap<Person,List<Account>> date;). Metoda de serializare scrie datele intrun fisier de tipul .ser, iar metoda de deserializare returneaza o structura de tipul HashMap. Rezultatul este folosit in constructorul clasei pentru a putea initializa datele.

```

public Bank() {
    date=(HashMap<Person,List<Account>>)deserialization();
    if(date==null)
        date=new HashMap<>();
    for (Map.Entry<Person, List<Account>> entry : date.entrySet()) {
        for (Account a : entry.getValue())
            a.addObserver(entry.getKey());
    }
}

```

Dupa ce initializez datele adaug observator pentru fiecare cont.

```

public Person findById(int id) {
    for(Map.Entry<Person,List<Account>> entry : date.entrySet()) {
        if(entry.getKey().getId()==id)
            return entry.getKey();
    }
    return null;
}

public Account findAcc(int personId, int id) {
    for(Map.Entry<Person,List<Account>> entry : date.entrySet()) {
        if(entry.getKey().getId()==personId)
            for(Account a : entry.getValue())
                if(a.getId()==id)
                    return a;
    }
}

```



```

    return null;
}

```

Metoda `findById` cauta un anumit client dupa id si returneaza un obiect de tipul `Person` sau `null` daca acel id nu este stocat, iar metoda `findAcc` cauta un cont.

```

public int addPerson(Person p) {
    assert p!=null;
    List<Account> accounts = new ArrayList<>();
    Person aux = findById(p.getId());
    if(aux!=null)
        return 2;//persoana exista
    date.put(p,accounts);
    return 1;
}

@Override
public int removePerson(int id) {
    Person delP = new Person();
    int err=0;
    for(Map.Entry<Person,List<Account>> entry : date.entrySet()) {
        if(entry.getKey().getId()==id) {
            delP = entry.getKey();
            err = 1;
        }
    }
    if(err==0)
        return 2;//persoana nu exista
    date.remove(delP);
    return 1;//succes
}

```

Metodele `add/removePerson` returneaza 1 in caz de succes si 2 in caz contrar (persoana exista deja in baza de date in cazul adaugarii sau nu exista in cazul stingerii)

```

@Override
public int addAccount(Account a) {
    assert a!=null;
    if(findById(a.getPersonId())==null)
        return 2; //persoana nu exista
    if(findAcc(a.getPersonId(),a.getId())!=null)
        return 3; //contul exista
    for(Map.Entry<Person,List<Account>> entry : date.entrySet()) {
        if(entry.getKey().getId()==a.getPersonId()) {
            entry.getValue().add(a);
            a.addObserver(entry.getKey());
        }
    }
    return 1;
}

@Override
public int removeAccount(int personId, int id) {
    List<Account> accounts = new ArrayList<>();
    Account delA = new Account();
    Person p = new Person();
    int err=0;
    for(Map.Entry<Person,List<Account>> entry : date.entrySet()) {
        if(entry.getKey().getId()==personId) {
            accounts.addAll(entry.getValue());
            p=entry.getKey();
            err = 1;
        }
    }
}

```

```

    if(err==0)
        return 2;//persoana nu exista
    for(Account a : accounts)
        if(a.getId()==id) {
            delA=a;
            err=2;
        }

    if(err==1)
        return 3;//contul nu exista
    accounts.remove(delA);
    date.put(p,accounts);
    return 1;//succes
}

```

Metodele add/removeAccount returneaza 1 in caz de succes sau diferite coduri de eroare.(persoana existata/ nu exista sau contul exista/ nu exista)

```

@Override
public List getPersons() {
    List<Person> persons = new ArrayList<>();
    for(Map.Entry<Person,List<Account>> entry : date.entrySet()) {
        persons.add(entry.getKey());
    }
    return persons;
}

@Override
public List getAccounts() {
    List<Account> accounts = new ArrayList<>();
    for(Map.Entry<Person,List<Account>> entry : date.entrySet()) {
        accounts.addAll(entry.getValue());
    }
    return accounts;
}

```

Metodele getPersons si getAccounts returneaza o lista de obiecte care va fi transmisa spre afisare in interfata grafica.

```

@Override
public int editPerson(int id, String nume, String adresa, String telefon) {
    int err=0;
    for(Map.Entry<Person,List<Account>> entry : date.entrySet()) {
        if(entry.getKey().getId()==id) {
            if(!nume.equals(""))
                entry.getKey().setNume(nume);
            if(!adresa.equals(""))
                entry.getKey().setAdresa(adresa);
            if(!telefon.equals(""))
                entry.getKey().setTelefon(telefon);
            err = 1;
        }
    }
    if(err==0)
        return 2;//clientul nu exista
    return 1;//succes
}

```

Pentru editare transmit noile valori ale variabilelor si cu ajutorul metodelor setter actualizez valoarea.

```

for(Account a : accounts) {
    if (a.getId() == id) {
        if (a instanceof SavingAccount) {
            if(a.getValoare()==0)
                return 4;//cont gol
            else
                System.out.println("Persoana " + a.getPersonId() + " a efectuat
retragere " +
                                a.getValoare()*((SavingAccount) a).getDobanda());
                a.setValoare(0);
        }
        else {
            if (!a.withdraw(sum))
                return 5;//valoare prea mare
        }
        err = 2;
    }
}

```

In metoda withdraw verific ce tip de cont e si apelez metoda de retragere din clasa Account. In cazul in care contul este de tip SavingAccount verific daca nu e gol si pentru ca se poate face o singura retragere setez valoarea la 0 si afisez un mesaj cu suma retrasa. O calculez cu ajutorul dobanzii(presupun ca retragerea se face dupa un an). Returnez coduri de eroare in cazuri nefavorabile (cont gol, suma prea mare..).

```

for(Account a : accounts) {
    if (a.getId() == id) {
        if (a instanceof SavingAccount)
            if(a.getValoare()>0)
                return 4;//o singura depunere
        a.deposit(sum);
        err = 2;
    }
}

```

La depunere in cazul unui de tipul SavingAccount daca se incearca mai multe depuneri se returneaza un cod de eroare.

Aplicatia se porneste din clasa Gui.

4.2.GUI

In clasa Gui se realizeaza interfata.

In constructor este creat frame-ul principal cu 3 butoane, iar la apasarea unui buton se apeleaza metoda corespunzatoare.

Metodele opPerson, opAccount, opOper realizeaza alte ferestre cu elemente specifice fiecarei operatii.

Serializarea se face latunci cand se inchide aplicatia, la apasarea butonului „X”.

```
frame.addWindowListener(new WindowAdapter() {  
    @Override  
    public void windowClosing(WindowEvent e) {  
        super.windowClosing(e);  
        bank.serialization();  
        e.getWindow().dispose();  
    }  
});
```

```
public void opPerson() {  
    final JFrame frame = new JFrame("Person");  
    frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);  
    frame.setSize(800, 400);  
    JPanel panel1 = new JPanel();  
    JLabel lid = new JLabel("id");  
    final JTextField id = new JTextField("", 3);  
    JLabel lnume = new JLabel("nume");  
    final JTextField nume = new JTextField("", 12);  
    JLabel ladresa = new JLabel("adresa");  
    final JTextField adresa = new JTextField("", 12);  
    JLabel ltelefon = new JLabel("telefon");  
    final JTextField telefon = new JTextField("", 8);  
    panel1.add(lid);  
    panel1.add(id);  
    panel1.add(lnume);  
    panel1.add(nume);  
    panel1.add(ladresa);  
    panel1.add(adresa);  
    panel1.add(ltelefon);  
    panel1.add(telefon);  
    panel1.setLayout(new FlowLayout());  
  
    JPanel panel2 = new JPanel();  
    JButton add = new JButton("add");  
    JButton edit = new JButton("edit");  
    JButton delete = new JButton("delete");  
    JButton view = new JButton("view");  
    panel2.add(add);  
    panel2.add(edit);  
    panel2.add(delete);  
    panel2.add(view);  
    panel2.setLayout(new FlowLayout());  
    final JPanel panel = new JPanel();  
  
    panel.add(panel1);  
    panel.add(panel2);  
    panel.setLayout(new BorderLayout(panel, BorderLayout.Y_AXIS));  
  
    final JTable table = new JTable();
```

```

JScrollPane tpanel = new JScrollPane(table);
panel.add(tpanel);

frame.setContentPane(panel);
frame.setVisible(true);

add.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String numeAd, adresaAd, telefonAd;
        int idAd;
        if (id.getText().equals("") || nume.getText().equals("") ||
adresa.getText().equals("") || telefon.getText().equals(""))
            JOptionPane.showMessageDialog(panel, "Date invalide");
        else {
            idAd = Integer.parseInt(id.getText());
            numeAd = nume.getText();
            adresaAd = adresa.getText();
            telefonAd = telefon.getText();
            Person p = new Person(idAd, numeAd, adresaAd, telefonAd);
            int ok = bank.addPerson(p);
            if(ok==2)
                JOptionPane.showMessageDialog(panel, "Id existent");
            else if(ok==1)
                JOptionPane.showMessageDialog(panel, "Succes");
        }
    }
});

view.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        Object column[]={"Id","Nume","Adresa","Telefon"};
        Object rows[][] = new Object[50][50];
        List<Person> list = bank.getPersons();
        int i=0;
        for(Person p : list) {
            rows[i][0] = p.getId();
            rows[i][1] = p.getNume();
            rows[i][2] = p.getAdresa();
            rows[i][3] = p.getTelefon();
            i++;
        }

        DefaultTableModel model = new DefaultTableModel(rows,column);
        table.setEnabled(false);
        table.setModel(model);

        table.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                int row = table.rowAtPoint(e.getPoint());
                int col = table.columnAtPoint(e.getPoint());
                if(col==0) {
                    try {
                        int id = (Integer) table.getValueAt(row, col);
                        JOptionPane.showMessageDialog(panel,
bank.findById(id).toString());
                    } catch (NullPointerException a) {
                        JOptionPane.showMessageDialog(panel, "empty");
                    }
                }
            }
        });
    }
});

delete.addActionListener(new ActionListener() {

```

```

@Override
public void actionPerformed(ActionEvent e) {
    int delId;
    if(id.getText().equals(""))
        JOptionPane.showMessageDialog(panel,"Id invalid");
    else {
        delId = Integer.parseInt(id.getText());
        int ok;
        ok = bank.removePerson(delId);
        if(ok==1)
            JOptionPane.showMessageDialog(panel,"Succes");
        else if(ok==2)
            JOptionPane.showMessageDialog(panel,"Perosana nu exista");
    }
}

});

edit.addActionListener(new ActionListener() {
@Override
public void actionPerformed(ActionEvent e) {
    String numeUp, adresaUp, telefonUp;
    int idUp;
    if (id.getText().equals(""))
        JOptionPane.showMessageDialog(panel, "Date invalide");
    else {
        idUp = Integer.parseInt(id.getText());
        numeUp = nume.getText();
        adresaUp = adresa.getText();
        telefonUp = telefon.getText();
        int ok = bank.editPerson(idUp,numeUp,adresaUp,telefonUp);
        if(ok==1)
            JOptionPane.showMessageDialog(panel,"Succes");
        else if(ok==2)
            JOptionPane.showMessageDialog(panel,"Persoana nu exista");
    }
}

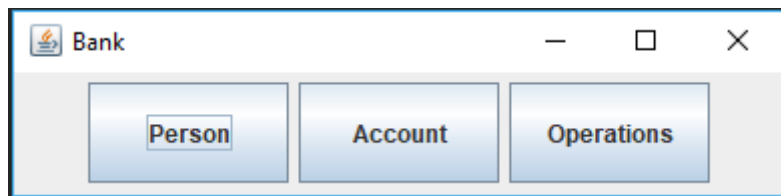
});
}

```

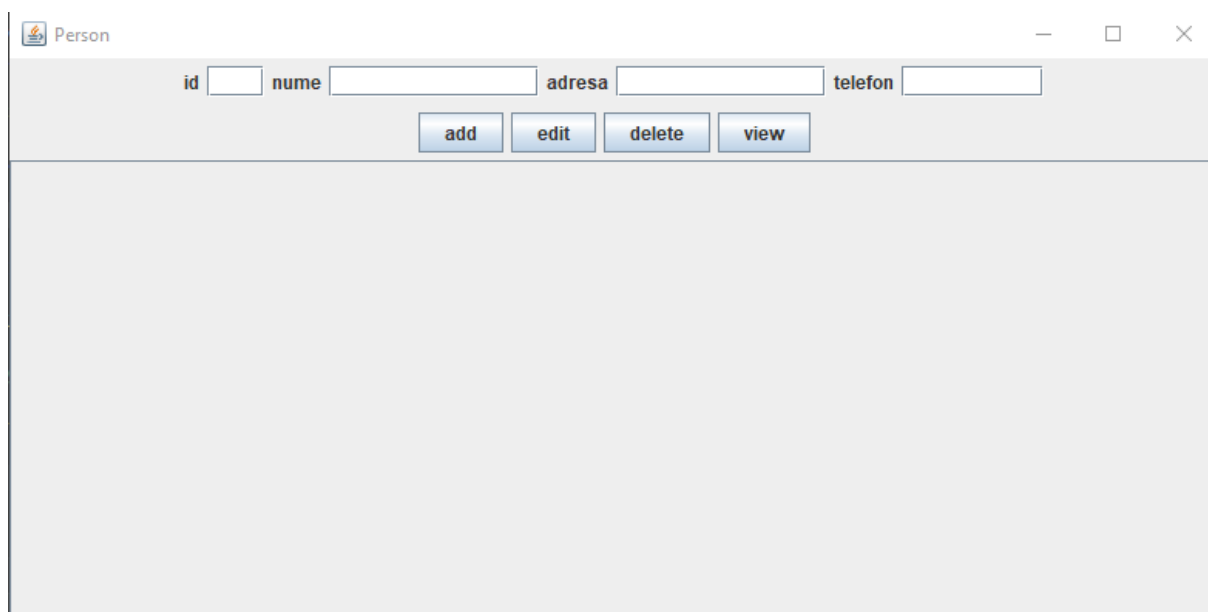
In fiecare fereastră se adauga elementele grafice (label-uri, textfield-uri, butoane) corespunzatoare operatiei. Afisarea perosanelor sau a conruilor din baza de date se face intr-un tabel. Pe tabel am implementat un listener astfel incat atunci cand se apasa pe id-iul unei pesoane sau al unui cont se vor afisa informatiile despre aceea intrare intr-o fereastră noua.


5.Testare

Din panoul principal se alege operatia dorita.



La apasarea butonului Person sau Account va apareea o noua fereastră cu operatiile ce se pot efectua asupra lor.



 Account

—

□

×

person id

id

SavingAccount

▼

valoare

dobanda

add

edit

delete

view

La apasarea butonului „view” se va afisa un tabel cu toti clientii.

[illegible]

Pentru adaugare se introduc date in toate campurile si se apasa butonul „add”, pentru editare se introduce id-ul clientului si valori in campurile care se doaresc a fi modificate, iar pentru stergere se introduce doar id-ul.

La apasarea butonului Operations se va deschide o fereastră noua in care se pot efectua operatiile de retragere si depunere.



Pentru a efectua o retragere sau o depunere se completeaza id persoana si id cont si suma. Daca persoana/contul nu exista sau suma e invalida vor fi afisate mesaje de eorare.

5.1 JUnit

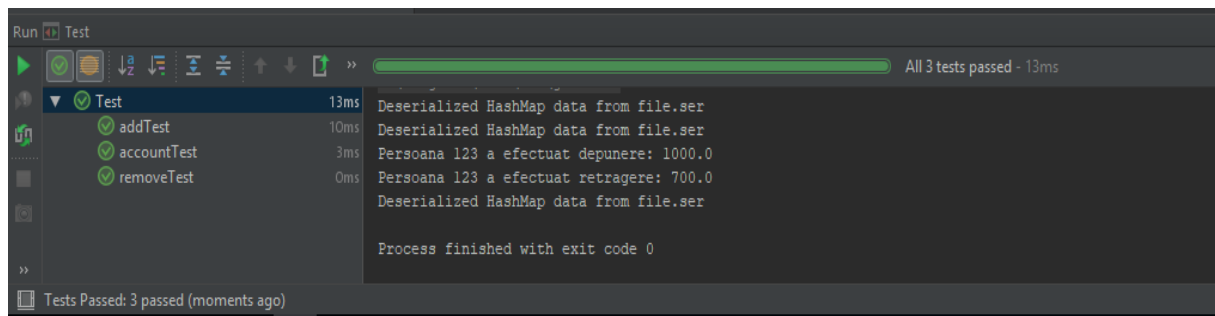
```
public class Test {

    Bank bank = new Bank();

    @org.junit.Test
    public void addTest() {
        Person p = new Person(123, "ionica", "ro", "0439593000");
        assertEquals(1, bank.addPerson(p));
        assertEquals(2, bank.addPerson(p));
    }

    @org.junit.Test
    public void removeTest() {
        Person p = new Person(123, "ionica", "ro", "0439593000");
        bank.addPerson(p);
        assertEquals(2, bank.removePerson(1223)); //id invalid
        assertEquals(1, bank.removePerson(123));
    }

    @org.junit.Test
    public void accountTest() {
        Person p = new Person(123, "ionica", "ro", "0439593000");
        Account a1 = new SavingAccount(1, 123, "SavingAccount", 0, 1.6f);
        Account a2 = new SpendingAccount(3, 123, "SpendingAccount", 0);
        Account a3 = new SpendingAccount(11, 777, "SpendingAccount", 0);
        bank.addPerson(p);
        assertEquals(1, bank.addAccount(a1));
        assertEquals(1, bank.addAccount(a2));
        assertEquals(3, bank.addAccount(a2)); //contul exista
        assertEquals(2, bank.addAccount(a3)); //persoana nu exista
        assertEquals(5, bank.withddraw(123, 3, 700)); //fonduri insuficiente
        assertEquals(1, bank.deposit(123, 3, 1000));
        assertEquals(1, bank.withddraw(123, 3, 700));
        assertEquals(300, bank.findAcc(123, 3).getValoare(), 0);
    }
}
```



6.Rezultate

Aplicatia gestioneaza o baza de date a unei banci si se pot efectua diferite operatii asupra clientilor si conturilor: adaugare, editare, stergere si depunerea sau retragerea de numerar.

7.Concluzii

In aceasta tema am invatat sa lucrez cu structura de date HashMap si fiserele seriale. Am implementat Design by Contract si Design Pattern Observer.

8.Bibliografie

www.stackoverflow.com