**Facultatea de Automatica și Calculatoare**

**Calculatoare și Tehnologia Informației**



**Tehnici de Programare**

**Documentație**

**Tema 4:**

**Design by Contract**

**Programming Techniques**

**Profesor îndrumator: Realizat de:**

**Cristina Pop Câmpean Casiana Ștefana**

**Grupa: 30228**

**Cuprins**

**1. Obiectivul temei**

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

**3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase,**

**interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor)**

**4. Implementare**

**5. Testare**

**6. Rezultate**

**7. Concluzii**

**8. Bibliografie**

**Cuprins**

**1. Obiectivul temei**

Obiectivul acestui proiect este implementarea tehnicilor de Design by Contract. Pe baza unei diagrame de clase trebuie sa realizam mai multe cerinte:

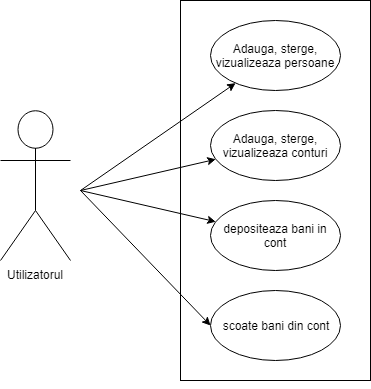
1. Sa definim interfata BankProc care cuprinde mai multe metode (add/ remove clients etc.) si sa ii specificam pre si post conditiile pentru aceste metode.
2. Sa implementam clasele Person, Account, SavingAccount, SpendingAccount.
3. Sa impementam un Observer Design Pattern care va avea rolul de a notifica titularul contului in legatura cu orice modificare a unui cont.
4. Sa implementam in clasa Bank o colectie predefinita care utilizeaza hastable. Cheile din hashtable vor fi generate pe baza titularului contului. O persoana poate fi tittul pentru mai multe conturi. Pentru afisarea datelor se va folosi un JTable. Implementam clasa folosind Design by Contract (folosim pre si post conditii, assert, invariants) . Declaram metoda de tip „well formed”.

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

Pentru a realiza proiectul, a fost necesar sa analizam cu atentie cerintele temei si baremul acesteia. Din acestea reiese ca va trebui sa creem minim 5 clase si o interfata, mai exact clasele Bank, Person, Account, SavingAccount, SpendingAccount si interfata BankProc. Pentru BankProc va trebui sa specificam pre si post conditiile pentru fiecare metoda, astfel a fost necesara studierea tehnicilor de Design by Contract, aplicate totodata si in clasa Bank, unde a trebuit sa folosim assert si sa implementam metoda WellFormed. Pe langa acestea, pentru a implementa proiectul trebuie respectata digrama de clase a acestuia, asadar intre clasele Person si Account se va realiza o relatie de agregare, intre clasele SavingAccount, SpendingAccount si Account se va realiza o relatie de mostenire, iar clasa Bank va implementa interfata BankProc.

Totodata din baremul temei reiese faptul ca va trebui impementata interfata care va realiza operatiile de adaugare, stergere si vizualizare a persoanelor si conturilor bancare si se vor putea realiza operatiile de deposit si withdrawal. De asemenea cele doua tipuri de conturi se ecomonii si de cheltuiei se vor diferentia prin faptul ca cel de economii va putea realiza doar o singura operatie de depositare si scoatere a banilor si va avea si o dobanda, pe cand cel de cheluiala poate realiza oricate operatii de depositate si scoate a banilor si nu va tine cont de o dobanda.

O alta parte a cerintei o constituiea afisarea datelor referitoare la persoane si account. Aceasta se va realiza in interfata prntr-un JTable, iar pentru acesta se va implementa un click listener. Toate aceste date vor fi serializate. Nu in ultimul rand pentru a indeplini o alta cerinta se va implementa conceptul de Observer Design Pattern care va avea rolul de a transmite un mesaj titularului contului de fiecare data cand un cont de-al sau este modificat.



Utilizatorul poate realiza mai multe operatii, una dintre ele fiind:

Use case: Scoate bani din contul de cheltuieli

Actorul: utilizatorul aplicatiei

Asumptii: date introduse corect

Scenariu succes:

1. Utilizatorul apasa pe butonul Deposit / Withdrawal

2. Utilizatorul introduce corect numele titularului de cont, id-ul contului si suma pe care vrea sa o scoata

3. Utilizatorul apasa pe butonul Withdrawal

4. Pe ecran se va afisa daca a reusit sa scoata banii si i se va afisa ce holder si din ce cont a scos banii, iar daca suma de bani de scos este prea mare se va afisa un mesaj de fonduri insufecinte. Totodata, in consolsa se va notificata titularul prin faptul ca s-a realizat scoaterea banilor din contul sau.

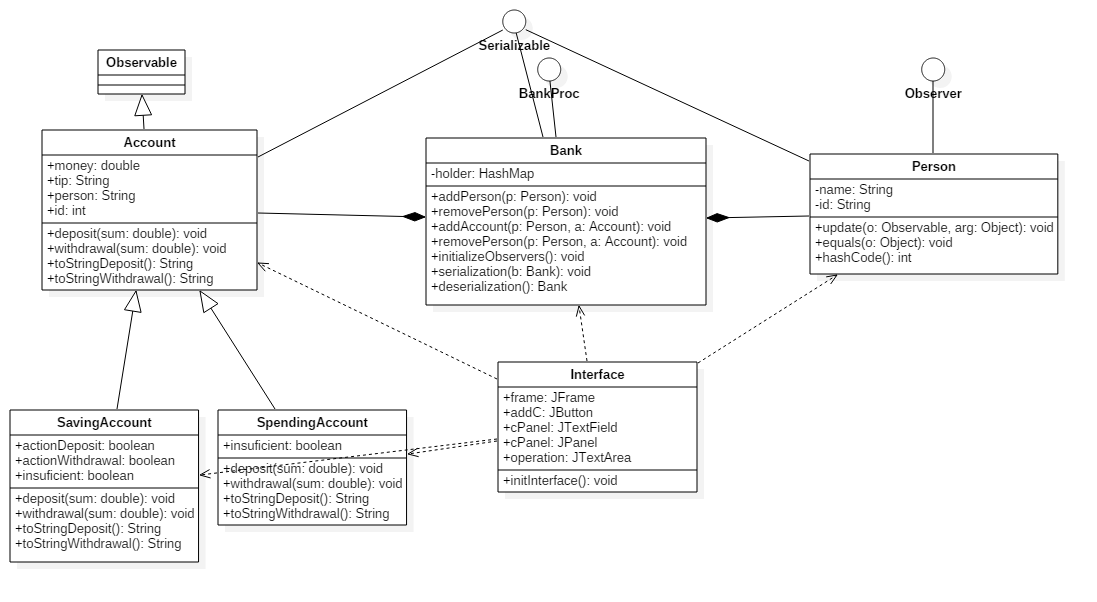
**3. Proiectare (decizii de proiectare, diagrame UML, structuri de date, proiectare clase,**

**interfete, relatii, packages, algoritmi, interfata utilizator, modul de tratare a erorilor)**

**Decizii de proiectare**

Pentru a implementa proiectul, am ales sa urmez cu atentie diagrama de clase data si de asemenea sa urmez cerintele temei si baremul. Pentru aceasta am implementat clasele Bank, Account, SavingAccount, SpendingAccount, Interface, interfata BankProc si Junit-ul Test.

**Diagrame UML**



**Structuri de date**

La acest proiect am folosit structuri de date de tip HashMap si ArrayList. HashMap-ul este folosit in clasa Bank ca si atribut care retine ca si chei obiecte de tip Person si ca si valori ArrayList-uri de tip Account. Fiecare persoana poate avea mai multe conturi, persoana fiind cheie pentru lista de conturi. Pentru folosirea acestei structuri, am suprascris metodele HashCode si equals in clasa Person.

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

**int** result = 1;

resul t= 31 \* result + ((cnp == **null**) ? 0 : cnp.hashCode());

**return** result;

}

@Override

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

**if** ( obj == **null** ) **return** **false**;

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

**return** **false**;

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

**return** **true**;

**return** **this**.getId() == (( Person ) obj ). getId();

}

**Interfete**

Interfete utilizate: Serializable, BankProc, si Observer.

Interfata serializabile este utilizata pentru a realiza serializarea datelor. Interfata BankProc este utilizata de clasa Bank si este implemetaata in aceasta conceptul principal al temei, acela de Design by Contract, iar interfata Observer ajuta la realizarea altei cerinte a temei, aceeai de Observer Design Pattern.

**Interfata utilizator**

Interfata utilizator este alcatuita din mai multe paneluri care corespund cerintelor proiectului. Cand pornim interfata utilizator pe ecran va aparea meniul principal si si in functie de ce buton vom apasa se va deschide alt panel. Meniul contine 3 butoane: Clients Operation, Account Operations si Add / Withdrawal. Buton de Client Operations dupa ce va fi apasat va deschide un alt panel care se ocupa de adaugarea, stergerea, editarea si vizualizarea tutoror clientilor din banca. Astfel in panel sunt 5 butoane, 2 text field-uri si un JTable unde se vor afisa toti clientii. Apasand butonul de Account Operations vom putea realiza de asemenea inserare, stergere, editate si vizualizarea tutore conturilor bancare. Ultimul buton daca va fii apasat va aparea un alt panel care se ocupa de operatiile de deposit si withdrawal. Tot ce trebuie facut este sa introducem datele despre contul bancar, titulatul contuli si suma pe care vrem sa o scoate. In functie de de cont si de suma din acesta in JTextArea aflat pe ecran ni se va afisa operatia facuta, si rezultatul acesteia si suma curenta. in fiecare panel exista un buton numit Done care fiind apasat va reveni la panel-ul pentru meniu pentru a putea face alta operatie.

**4. Implementare**

Proiectul contine 6 clase, o interfata si un Junit.

Clasele: Bank, Person, Account, SavingAccount, SpendingAccount, Interface.

Intefata: BankProc.

Junit: Test.

In primul rand, am implementat **interfata BankProc**. Aceasta contine metodele addPerson, removePerson, addAccount si removeAccount, acestea fiind flosite pentru a adauga sau a sterge o persoana sau un cont in clasa Bank. Pentru a indeplini cerinta de a folosi Design by Contract, am introdus in interfata pre si post conditii. Preconditiile sunt expresii booleene care trebuie indeplinite inainte de a fi apelata metoda, iar post conditiile sunt expresii booleene care trebuie indeplinite dupa ce s-a apelat metoda.

Pentru metoda addPerson am pus ca si preconditii ca Person p != null, iar ca si postconditii ca si HashMap-ul holder sa contina cheia p holder. containsKey(p); si marimea hashMapului sa fie egala cu marimea HashMap-ului anterrior + 1 holder. size() == holder.size()@pre + 1. Pentru metoda removePerson am pus ca si preconditii ca Person p != null, si marimea HashMap-ului sa fie mai mare decat 0 holder. size() > 0, iar ca si postconditii ca marimea HashMap-ului holder sau scada cu 1 holder.size() == holder.size() @pre – 1. Pentru metoda addAccount am pus ca si preconditii ca argumentele de tip Person si Account sa nu fie nule p!= null && a != null, iar ca si postconditii am pus ca si ArrayList aflat in holder la cheia p sa contina contul tocmai adaugat holder.get (p).contains (a) si ca marimea ArrayList-ului aflat in HashMap-ul holder la cheia p sa se mareasca cu 1 holder. get(p). size() == holder. get(p). size()@pre + 1. Pentru metoda removeAccount am pus ca si preconditii ca Person si Account sa nu fie nule p!= null && a!= null, si marimea ArrayList-ului din HashMap sa fie mai mare decat 0 holder. get(p). size() > 0, iar ca si postconditii ca marimea ArrayList-ului din HashMap aflat la cheia p sa scada cu 1 holder. get(p). size() == holder. get(p). size()@pre – 1.

**Clasa Bank** contine obiecul holder de tip HashMap<Person,ArrayList<Account>> care are si cheie un obiect de tip persoana si ca si valori o lista de obiecte de tip Account. Pentru implementare HashMap-ului a fost necesar sa suprascriem metodele HashCode() si equals(), impementate in clasa Person. Aceasta clasa implementeaza doua interfete, una dintre acestea fiind Serializable deoarece a trebuit ca si date folosite din HashMap sa fie serializate, sa fie salvate intr-un fisier. Pentru a realiza aceasta serialziare am implementat doua metode de scriere si citire din acel fisier, mai exact serialization si deserialization. Aceste metode folosesc obiecte de tip FileOutputStream si FileInputStream si utilizeaza metodele oout.writeObject(b) si in.readObject(). In aceste fisiere am salvat obiectul de tip Bank in care voi avea toate datele necesare aflate in HashMap-ul holder.

In clasa Bank am impementat interfata BankProc, si in fecare dintre metodele impementate am realizat si assertions. Pentru metoda addPerson am utilizat la inceputul metodei **assert** p != **null**;, si am initializat variabila pentru marime **int** sizePre = holder. size(); iar la final am utilizat **assert** holder. containsKey(p);, am initializat variabila pentru marime **int** sizePost=holder. size(); si am folosit assert si pentru verificarea acesteia **assert** sizePost == sizePre+1; Aceasta metoda adauga in hashMap noua cheie p de tip Person prin metoda put holder.put (p, **new** ArrayList<>());.La fel si metodatele addAccount si removeAccount funtioneaza in aceeasi idee.

**Clasa Person** contine atributele necesare pentru a defini un client al bancii, mai exact name si id. Aceasta clasa implementeaza interfetele Serializable si Observer, prima clasa fiind utilizata pentru a putea face serializarea datelor necesare in fisierul folosit pentru proiect, iar cea din urma clasa este necesara pentru a putea implementa conceptul de Observer Design Pattern. Pentru acest concept este suprascrisa metoda update a acestei interfete care primeste va printa in consola argumentul dat de metoda notiflyAllObservers. Deoarece in clasa Bank avem HashMap care are ca si chei un obiect de tip Person, am suprascris metodele HashCode() si equals.

**Clasa Account** contine este o clasa abstracta fiind mostenita de clasele SavingAccount si SpendingAccount, clasa continand toate datele necesare pentru a putea defini un cont bancar, mai exact id, money, tip, si person, reprezentand id-ul contului bancar, numele titularului, suma aflata in cont si tipul contului( de economii sau de cheltuieli). Aceasta clasa are ca si metode abstracte deposit, withdraw, toStringDeposit si toStringWithdraw care sunt impelementate in clasele copii. De asemenea, aceasta clasa extinde clasa Observable care are ca rol implementarea conceptului de Observer Design Pattern. Deoarece in cerinta ni se cere ca titularul unui cont sa fie notificat in legatura cu orice modificare asupra unui cont de-al sau, in fiecare metoda unde s-ar putea moficia ceva, folosim meodele setChanged() si notifyObservers, ultima primind ca si argument un String care va fii afisat pe ecran de fiecare data cand ceva se schimba.

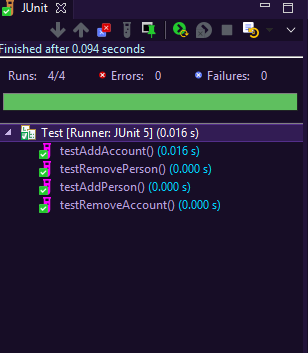
Clasele care mostenesc Account, mai exact **SavingAccount** si **SpendingAccount** sunt aproape asemanatoare. Conform cerintei, deosebirea dintre cele 2 este faptul ca cel de economii va putea realiza doar o sigura operatie de scoatere sau de depositare a banilor si va exista o dobanda pentru acesta, pe cand cel de cheltuieli poate sa realizeze oricate operatii de scoatere si de depositare a banilor, iar pentru acesta nu exista nicio dobanda.

**5. Testare**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Date intrare | Rezultat asteptat | Rezultat obținut | Pass/Fail |
| AddPerson | Bank b=new Bank();  Person p=new Person(„15432”,”ccc”);  SavingAccount a1=new SavingAccount (321,450,”ccc”); | 1 | 1 | Pass |
| RemovePerson | 0 | 0 | Pass |
| AddAccount | 1 | 1 | Pass |
| RemoveAccount | 0 | 0 | Pass |

Testarea se realizeaza in clasa Test unde testam cele patru metode implemetate din interfata BankProc: addPerson, removePerson, addAccount, si removeAccount.

**6. Rezultate**



Toate rezultatele testelorau iesit corect, rezultand faptul ca operatiile testate sunt bine implementate.

**7. Concluzii**

In concluzie, in urma acestui proiect am reusit sa invat multe concepte noi despre tehinicile de Design by Contract, HashMap, serializare, Observer Design Pattern si despre click Listener pentru JTable.

**8. Bibliografie**

1. <https://beginnersbook.com/2013/12/hashmap-in-java-with-example/>
2. <http://www.tutorialspoint.com/java/java_serialization.htm>
3. <https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>
4. <https://docs.oracle.com/javase/7/docs/api/java/util/Observable.html>
5. <https://stackoverflow.com/questions/11415160/how-to-enable-the-java-keyword-assert-in-eclipse-program-wise>
6. <https://javarevisited.blogspot.ro/2012/01/what-is-assertion-in-java-java.html>
7. <https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html>
8. <https://stackoverflow.com/questions/7350893/click-event-on-jtable-java?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa>
9. <https://allaboutbasic.com/2010/12/29/jtable-cell-click-event-click-on-the-jtable-cell-and-show-the-value-of-that-cell/>