METODE AVANSATE DE PROGRAMARE

Laborator 6

1. Doriți să dezvoltați un sistem bancar online simplu care să permită utilizatorilor să efectueze operațiuni bancare de bază, precum gestionarea conturilor și efectuarea de tranzacții..

   Cerințe Funcționale:

   1. Autentificare Utilizator:
      - Utilizatorii trebuie să poată să se autentifice în sistem utilizând un nume de utilizator și o parolă.
   2. Gestionare Conturi:
      - Utilizatorii trebuie să poată să adauge și să șteargă conturi bancare asociate contului lor.
      - Pentru fiecare cont, sistemul trebuie să afișeze soldul curent și să permită utilizatorilor să efectueze operațiuni precum depozitare și retragere de numerar.
   3. Efectuarea Tranzacțiilor:
      - Utilizatorii trebuie să poată să efectueze transferuri de bani între conturile lor sau către alte conturi din sistem.
      - Sistemul trebuie să valideze tranzacțiile și să actualizeze soldurile conturilor implicate.
   4. Cerințe Tehnice:
      - Implementarea sistemului trebuie o arhitectură cu clase, subclase și interfețe.
      - Codul trebuie să fie bine structurat, modular și să respecte principiile de programare orientată pe obiecte.
      - Entități:
        - Interfața BankAccount: API pentru operații bancare, precum depozit, withdraw, getBalance
        - Interfața Transaction: API pentru tranzatii bancare
        - Clasă TransferTransaction care implementează interfața Transaction

- Clase care implementează interfața BankAccount pentru un cont de economii, respectiv pentru un cont curent
- Clasa User: modelare pentru utilizatorul sitemului bancar

2. Sa se scrie un program Java care modeleaza activitatea unui ghiseu bancar. Sistemul este format din urmatoarele entitati:

✓ **ContBancar** cu urmatoarele atribute:
- numarCont(String)
- suma(float)

✓ **Client** cu urmatoarele atribute:
- nume(String)
- adresa(String)
- conturi

Conturile bancare pot fi de mai multe feluri: in LEI și în EURO. Conturile în EURO au o dobanda fixa 3% pe an, daca suma din cont este mai mare decat 500 EURO sau 0 in caz contrar, astfel acest tip de cont trebuie sa ofere serviciul **public float getDobanda()**. Pot exista transferuri intre conturile în LEI si numai intre ele, mai concret un cont de acest tip trebuie sa ofere serviciul **public void transfer(ContBancar contDestinatie, float suma).**

Toate conturile implementeaza o interfata **SumaTotala** care are o metoda public float **getSumaTotala()**. Pentru conturile în lei suma totala este chiar suma existenta în cont iar pentru conturile în EURO este suma*4.92.

✓ **Banca** cu urmatoarele atribute:
- clienti(tablou de elemente de tip Client)
- codBanca(String)

Conturile, pe langa implementarea interfetei **SumaTotala**, vor avea metode pentru setarea respectiv citirea atributelor ca unica modalitate de modificare (din exterior) a continutului unui obiect de acest tip precum si metodele public float **getDobanda()**, **void transfer(ContBancar contDestinatie, float suma)** dar numai acolo unde este cazul.

Clasa Client va contine un set de metode pentru setarea respectiv citirea atributelor ca unica modalitate de modificare (din exterior) a continutului unui obiect Client, un constructor prin intermediul caruia se vor putea inițializa numele, adresa clientului

precum si conturile detinute de acesta; clasa trebuie sa ofere și o metoda pentru afisare. Clasa Banca va implementa metode pentru efectuarea urmatoarelor operatii, în contextul ın care nu pot exista mai multi clienti cu acelasi nume.

• adaugarea unui client nou **public void add(Client c)**

• afisarea informatiilor despre un client al carui nume se transmite ca parametru **public void afisareClient(String nume)** în urmatoarea forma:

– nume adresa

– pentru fiecare cont detinut, se va afisa doar suma totala pe o linie separata