Laborator 7. Relatia de compozitie. Mecanismul de mostenire

1. O facultate este formată din mai multe specializări. Fiecare facultate este caracterizată prin denumirea ei, iar despre specializările fiecărei facultăţi se cunosc următoarele informaţii: denumirea specializării, dacă este acreditată sau nu, număr de studenţi şi numărul de profesori care predau la specializarea respectivă. Modelul UML al acestei probleme este prezentat în Figura 1.

După cum se observă, clasa `Specializare (sau Program de studiu)` are următoarele operaţii:

- doi constructori: unul iniţializează toate proprietăţile clasei cu datele primite ca parametri ai constructorului şi celălalt iniţializează numai numele specializării, impunând ca predefinit numărul de studenţi: 0, număr de profesori: 0 şi nu este acreditată.
- metoda-predicat `esteAcreditata()` care întoarce o valoare booleană corespunzătoare dacă specializarea este acreditată sau nu.
- metoda `calculeazaRaport()` care întoarce un număr întreg care reprezintă numărul de studenţi care-i revin unui profesor.
- rescrie metoda `toString()` a clasei `Object` pentru a furniza informaţii complete despre o anumită specializare, ca de exemplu:

```
    Denumire specializare: Informatica
    Numar studenti: 100
    Numar profesori: 25
Clasa Facultate
    Denumire facultate: Facultatea de Matematica si Informatica
    Denumire specializare: Informatica
    Numar studenti: 100
    Numar profesori: 25
```

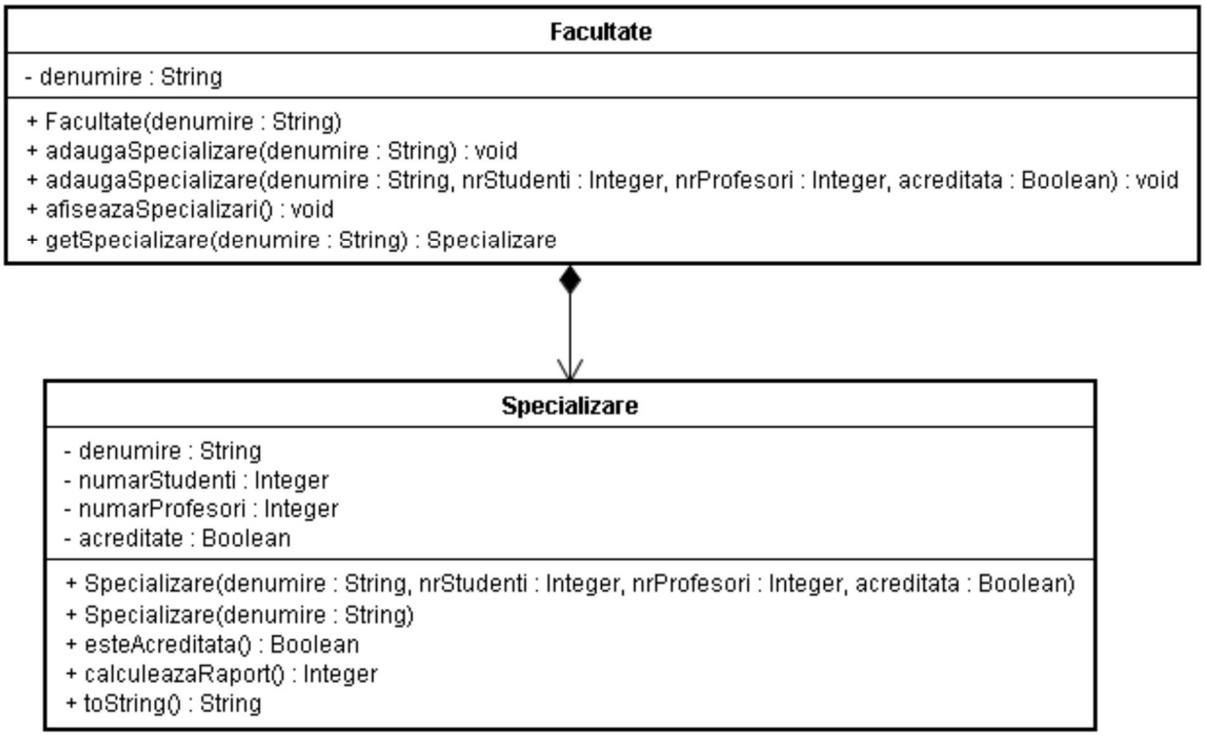

Figura 1. Relaţia de compoziţie dintre Facultate şi Specializare

Utilizăm aceste clase pentru a crea o clasă de test numită `TestFacultate` ce conţine metoda `main()` şi:

- cerem utilizatorului să introducă date de la tastatură cu ajutorul unei ferestre de tip `InputDialog` pentru crearea unui obiect a clasei `Facultate`,
- cerem utilizatorului să introducă date de la tastatură cu ajutorul unei ferestre de tip `InputDialog` pentru crearea a două obiecte ale clasei `Specializare`,

- creăm două obiecte ale clasei `Specializare`, unul cu primul constructor şi celălalt cu al doilea constructor,
- afişăm informaţiile despre cele două specializari ale facultăţii create,
- afişăm dacă a doua specializare este acreditată sau nu,
- afişăm raportul dintre studenţi şi profesori, în cazul celei de-a doua specializări.

2. O bancă oferă împrumuturi tuturor persoanelor care doresc să cumpere apartamente sau produse de uz casnic. Un împrumut este identificat printr-un număr unic, date despre debitor (nume, prenume şi adresă), şi alte date despre împrumut, cum ar fi suma împrumutată (care este egală cu costul produsului), rata şi frecvenţa de plată. Implicit, rata este egală cu 10% din costul produsului (apartament sau produs de uz casnic) şi plata se face lunar. Debitorul poate alege o altă frecvenţă de plată în funcţie de tipul împrumutului. Dacă debitorul doreşte să cumpere un apartament, acesta poate să aleagă să plătească trimestrial, dar banca adaugă la rată un comision de 3% din suma pe care trebuie s-o plătească. Dacă debitorul vrea să cumpere un produs de uz casnic, el poate alege sa plătească o dată la 6 luni, numai că, în acest caz, banca adăugă un comision de 7% la rată.

Programul ar trebui să ştie şi să afişeze suma pe care debitorul trebuie s-o plătească. La început, suma este egală cu împrumutul şi este actualizată de fiecare dată când debitorul plăteşte o rată (i.e., metoda `platesteRata()` este apelată). Modelul programului este prezentat in Figura 2.

Cerinţe:

1. Implementaţi clasele din model.
2. Pentru a testa aceste clase, scrieţi clasa `TestImprumut` ce conţine metoda `main()` şi îndeplineşte următoarele operaţii:
    a. Presupunem că au fost făcute două împrumuturi: unul pentru un apartament şi celălalt pentru un produs de uz casnic (adică, se vor crea două obiecte, câte unul din clasa `ImprumutApartament` şi `ImprumutProdusUzCasnic`). În plus, presupunem că al doilea debitor a ales să plătească o dată la 6 luni.
    b. Apelaţi metodele `calculeazaRata()` şi `platesteRata()`.
    c. Afişaţi informaţiile despre cele două împrumuturi.

3. O agenţie de asigurări oferă două tipuri de poliţe de asigurare: de viaţă şi în caz de accident. O poliţă de asigurare are un număr unic, datele persoanei asigurate (nume, prenume şi adresă) şi alte date precum suma plătită de către asigurat, frecvenţa de plată şi suma asigurată. Numărul poliţei de asigurare este generat automat de către program.

Implicit, frecvenţa de plată este lunar şi suma implicită este de 13€. Dar asiguratul poate alege când să plătească în funcţie de tipul asigurării. Dacă persoana are o poliţă de viaţă, poate alege să plătească trimestrial, dar agenţia va adăuga un comision de 2% la sumă. Dacă persoana are o poliţă în caz de accident, poate alege să plătească semestrial, dar agenţia va adăuga un comision de 5%.

Suma asigurată este egală cu suma tuturor ratelor plătite de asigurat şi este calculată de fiecare dată când persoana plăteşte o rată. Programul ştie când a plătit o persoană, mai precis când este apelată metoda `calculeazaSuma()`.

Modelul programului este prezentat în Figura 3.

Cerinţe:

a. Implementaţi clasele din model.
b. Pentru a testa aceste clase, scrieţi clasa `TestAsigurare` ce conţine metoda `main()` şi îndeplineşte următoarele operaţii:
    1. Crearea unei poliţe de asigurare de viaţă şi una de accident (adică, un obiect al clasei `AsigurareViaţă` şi celălalt din clasa `AsigurareAccident`). Presupunem că a doua persoană asigurată a ales să plătească de două ori pe an.
    2. Apelarea metodelor `calculeazaRata()` şi `calculeazaSuma()`.
    3. Afişarea informaţiilor despre cele două asigurări.

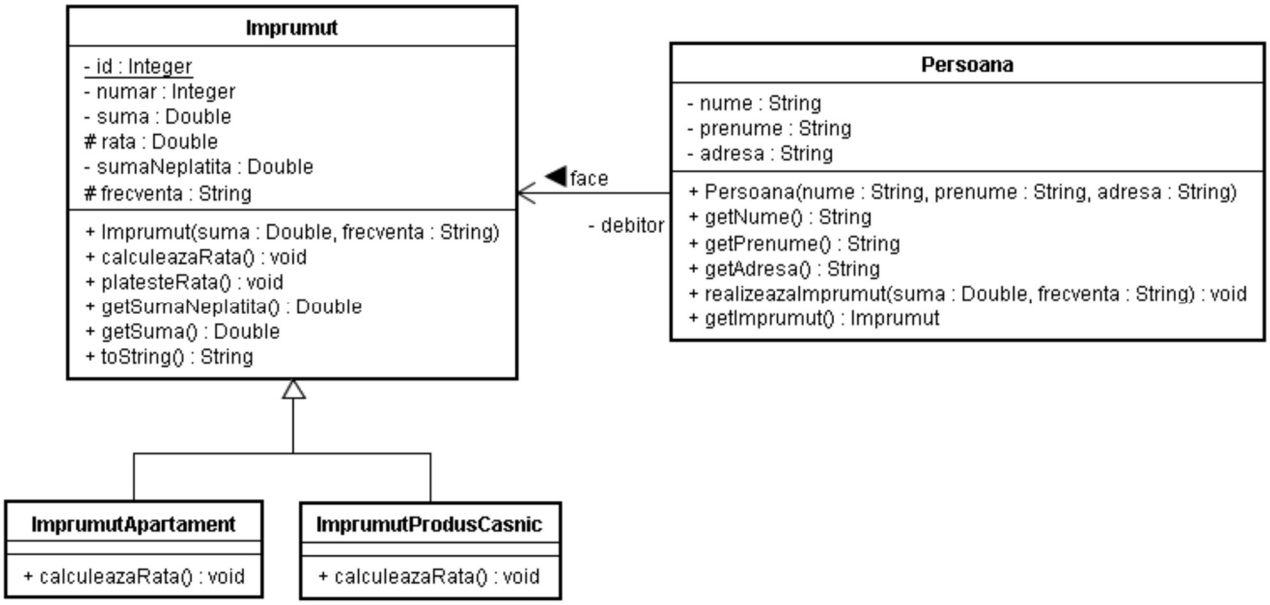

Figura 2. Diagrama UML incompleta de clase a problemei 2

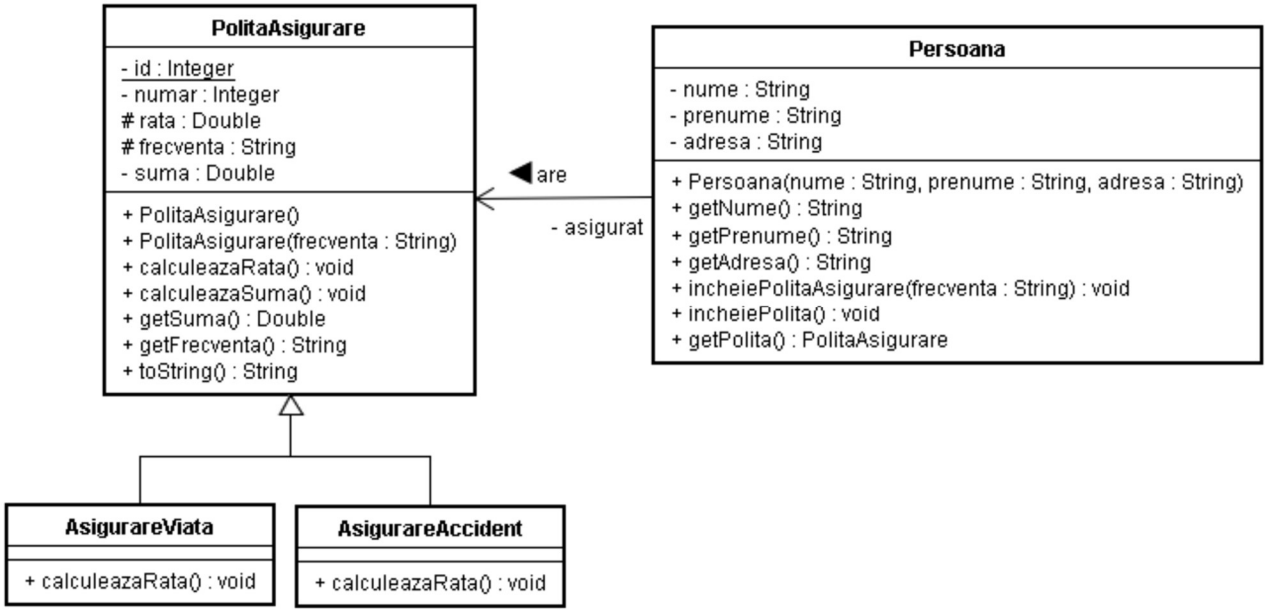

Figura 3. Diagrama UML incompleta de clase a problemei 3