

Inginerie Software 2024

Laboratorul 6

Modelare structuri. Diagrama De Clase.*

Mădălina Erașcu, Alexandru Munteanu, Cristian Pal, Ionica Puiu

March 2024

1 INTRODUCERE

Diagrama de clase se află în centrul procesului de modelare orientată obiect. Ea este diagrama principală pentru capturarea tuturor regulilor ce guvernează definiția și utilizarea obiectelor.

Ca depozitar al tuturor regulilor este și sursa principală pentru inginerie directă (forward engineering – transformarea modelului în cod) ca și ținta ingineriei inverse (reverse engineering – transformarea codului într-un model).

Atributele descriu **cunoștințele** unei clase și **operăriile** descriu **comportamentul** acesteia.

Elemente adiționale ca stereotipurile, valorile etichetate și constrângerile descriu modul în care pot fi adaptate clasele pentru a facilita dezvoltarea într-un domeniu particular.

Diagrama de clase este probabil cea mai utilizată diagramă UML. Diagrama de clase este diagrama principală pentru *generarea codului*.

Figura 1 următoare arată că deși fiecare din celealte diagrame ajută modelatorului să descopere informații valoroase despre un subiect, *toate* aceste descoperiri trebuie să fie reflectate în diagrama de clase.

Notăriile din diagrama de clase oferă un mijloc simplu și puternic de a specifica tot ce este necesar pentru generarea codului. Cu rigoare și cu suportul software corespunzător, diagrama de clase poate deveni în totalitate executabilă. Aceasta înseamnă că modelul poate fi testat prin executarea de scenarii și poate genera aplicația într-o varietate de contexte de implementare.

*Bazat pe resursele de laborator ale Conf. Dr. Cristina Mândruță

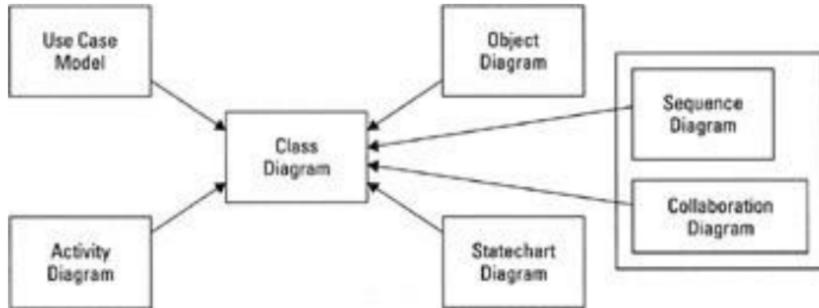


Figure 1: Diagrama de clase din perspectiva modelatorului

2 ELEMENTELE DE MODELARE

CLASA – este o definiție pentru o resursă; descrie un tip de obiect; descrie caracteristicile unui set de entități și modul de folosire a acestora. O definiție de clasă descrie mai multe obiecte de același tip. Un obiect este o instanță a unei clase.

RELATII – Scopul unei relații este de a stabili motivul pentru care două clase de obiecte trebuie să se cunoască și regulile care guvernează relația. În UML există trei tipuri fundamentale de relații: asociere, generalizare și dependență.

2.1 ELEMENTE PENTRU MODELARE CLASE

Nume: definește identitatea clasei. Trebuie să fie unic în cadrul unui pachet; când se face referire la clase cu același nume aflate în pachete diferite se va folosi numele calificat format din nume-pachet::nume-clasă.

Atribut: definește un tip de informație ce poate fi deținut și gestionat de tipul de obiect definit de clasa ce are atributul.

Operație (și metodă): definește cum poate fi invocat un comportament pe un obiect. Operația este definiția iar metoda este implementarea operației. Astfel e posibilă definirea de metode diferite, în clase diferite, pentru aceeași operație.

Vizibilitatea: definește nivelul de acces permis la un element al modelului (clasă, atribut sau operație). Valorile vizibilității sunt: private, public, protected și package.

Multiplicitate: definește numărul de valori ce pot fi asignate unui element al modelului (attribute sau parametru). Acest număr se poate exprima ca un domeniu, o enumerare, un număr fix sau o limită nedefinită (*).

Ordonare și unicitate: desemnează ordonarea și unicitatea valorilor dintr-un set de valori specificate la multiplicitate.

2.2 TIPURI DE RELAȚII ȘI ELEMENTE DE MODELARE

2.2.1 Asociere și link:

Asociere = relație semantică între două elemente ale modelului (clase). Include regulile pentru stabilirea și menținerea integrității relației atât la crearea cât și la utilizarea ei în aplicație.

Un link este cea mai simplă formă de relație care sprinjină **comunicarea între obiecte**. Două obiecte sunt conștiente unul de altul deoarece fiecare deține o referință la celălalt.

Un link este o *instanță a unei asocieri*. O **asociere** definește un **tip de link** ce poate exista între **tipuri de obiecte**.

- **Multiplicitate:** Fiecare capăt al unei asocieri trebuie să definească numărul de obiecte ce pot participa în asociere cu un obiect de la celalalt capăt.
- **Constrângere:** O constrângere este o regulă ce trebuie respectată pentru ca asocierea să fie validă. O constrângere este plasată la un capăt al unei asocierii pentru a dicta condițiile ce trebuie satisfăcute înainte ca obiecte (de tipul definit de clasa de la acel capăt al asocierii) să poată participa în relație. Constrângerile sunt închise între accolade ().
- **Nume rol:** Un nume de rol explică modul în care un obiect participă la un link. Numele rolului este plasat la capătul asocierii de lângă clasa (tipul de obiect) care joacă rolul.
- **Navigabilitate:** Fiecare capăt al unei asocieri este desemnat fie ca nedefinit (nicio notație), fie navigabil (o săgeată la capătul asocierii), fie nenavigabil (un 'X' la capătul asocierii).

Tipuri speciale de asociere

- **Asocierea reflexivă:** O asociere reflexivă definește un tip de relație ce poate exista între obiecte de același tip (instantație din aceeași clasă).
- **Agregarea** este un tip de asociere în care o clasă definește obiectele ce participă ca părți componente într-un ansamblu sau configurație, iar cealaltă clasă reprezintă întregul ansamblu (agregatul).
- **Compoziția** este un tip de agregare în care un obiect membru poate fi parte componentă în cel mult un obiect aggregat (și nu poate exista independent de acesta).

2.2.2 Generalizarea

Oferă un mijloc de organizare a similarităților și diferențelor dintr-un set de obiecte care au același scop. Generalizarea este o relație între clase în care o clasă, numită superclasă, conține trăsături partajate de toate obiectele din acea clasă, iar o altă clasă, numită subclasă, conține doar trăsături care diferențiază obiectele din această clasă de obiectele definite de superclasă.

- **Specializarea** descrie procesul de identificare a trăsăturilor ce diferențiază obiectele dintr-o superclasă. ”O specializare” se referă la o anumită subclăsă.
- **Moștenirea** descrie faptul că o subclasă are acces la trăsăturile superclasei atunci când este utilizată pentru a instanția obiecte.

2.2.3 Dependențe (rafinare, realizare, etc.)

O dependență reprezintă o abstractizare de nivel înalt a relației dintre entități ale modelului (clase sau pachete). O dependență definește doar existența unui fel de relație fără a specifica tipul acesteia. Natura generală a dependenței poate fi descrisă utilizând un stereotip (ex. <<include>>, <<extend>>, <<realization>>, <<refinement>>).

Nume clasă - definește identitatea și scopul. (who am I)
Atributele - definesc ceea ce cunoaște clasa. (what I know)
Relațiile - definesc pe cine cunoaște clasa. (who I know)
Operațiile - definesc capabilitățile clasei. (what I do)

Figure 2: Elemente de modelare clase

3 NOTAȚII

3.1 Clase

3.1.1 Notație generală

Nume Atribute Operații Definit de utilizator ...	Trei compartimente predefinite. Nume – obligatoriu Atribute – optional Operații – optional Suplimentar, UML susține compartimente definite de utilizator a.i. modelatorul poate adăuga informații referitoare la proiect sau orice consideră că este necesar.
--	---

Figure 3: Notație generală UML diagrama de clase

3.1.2 Stereotip - utilizat pentru definirea suplimentară a unei clase.

UML modelează un stereotip prin punerea sa între paranteze unghiulare duble (ex. <<entity>>, <<control>>).

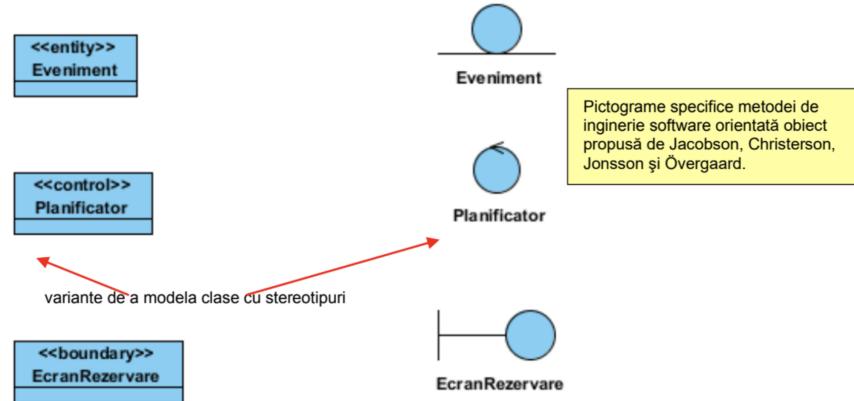


Figure 4: UML Modelare stereotip (entity,control)

3.1.3 Vizibilitate

Tip	Domeniu	Notăție
private	În clasă	-
package	În pachet	~
public	În sistem	+
protected	Într-un arbore de moștenire	#

Table 1: Tipuri de membri și notății in UML

3.1.4 Multiplicitate

Exprimă	Notăție
Domeniu de valori	$[V_{min}..V_{max}]$
Valoare specifică	$[V]$
Domeniu nelimitat superior	$[V_{min}..*]$
Set de valori discrete	$[v_1, v_2, \dots v_n]$

Table 2: Expresie multiplicitate

- Specifică numărul de valori ce pot fi asociate cu un element al modelului.
- Se modelează sub formă de expresie.

Poate fi adnotată cu constângerile de ordonare ordered și unicitate unique.

3.1.5 Compartimentul atributelor

- Compartiment de tip listă
- Poate conține doar atribute și este plasat întotdeauna sub compartimentul cu numele clasei.
- Conține toate informațiile pe care un obiect le are în proprietate.

Un obiect poate avea trei tipuri de informații:

- Cunoștințe despre el însuși, adică structura proprie și starea curentă, numită starea obiectului.
- Relațiile imediate.
- Informații statice.

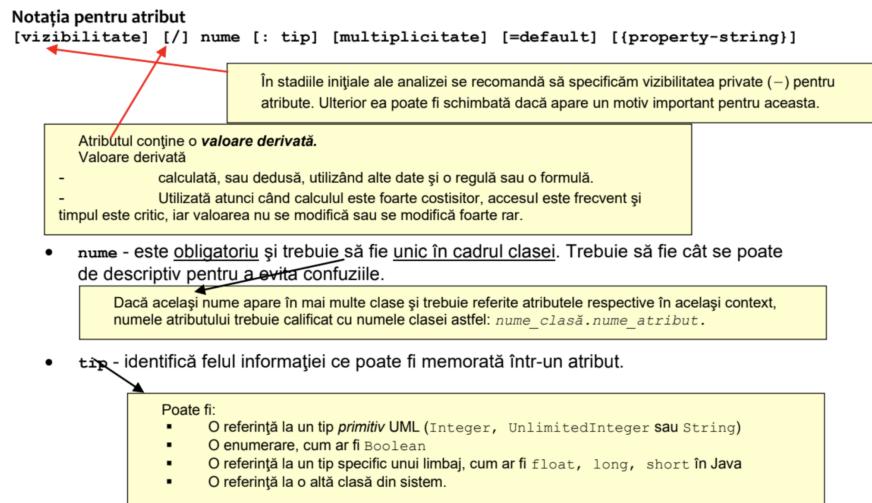


Figure 5: Notații pentru atribut

Multiplicitate – specifică numărul de valori ce pot fi asociate cu un atribut.

- **default** – valori implicate ce servesc două scopuri importante:
 - Protejarea integrității sistemului față de lipsa sau invaliditatea unor valori.

- Simplificarea semnificativă a utilizării.

Valorile implicate (default) se aplică atributului la crearea obiectului.

- **property string** – loc pentru orice informație despre atribut care nu poate fi preluată în câmpurile predefinite.

În mod obișnuit proprietățile se folosesc pentru păstrarea constrângerilor care garantează integritatea valorii atributului pe durata de viață a obiectului.

Aceste reguli se pot exprima sub formă de constrângerile asupra valorii atributului. Constrângerile pot fi implementate/impuse în orice metodă care încearcă să modifice valoarea tributului.

3.1.6 Compartimentul Operațiilor

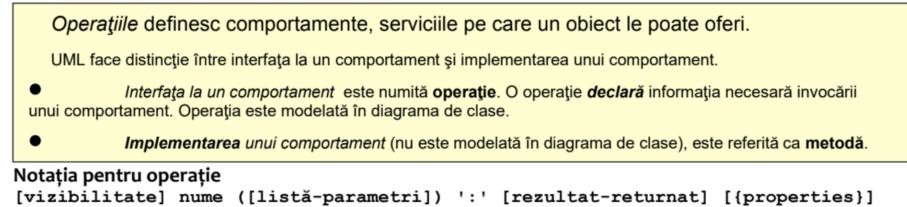


Figure 6: Notații pentru operație

Suplimentar, UML oferă suport pentru modelarea excepțiilor, pre-condițiilor și post-condițiilor.

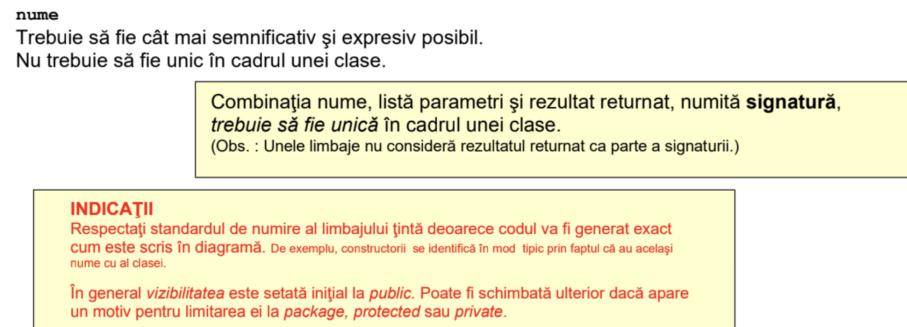


Figure 7: Denumirea unei operații

- **listă-parametri** – listă ordonată de attribute care definesc împreună intrările unei operații. Lista de parametri este optională, adică o operație nu trebuie să aibă în mod obligatoriu parametri.

Format: perechi **nume:tip** separate cu virgulă.

- **rezultat-returnat** este ieșirea unei operații; nu are nume ci numai tipul datelor returnate.

3.1.7 Compartimente definite de utilizator

Compartiment listă cu un nume, proprietăți, stereotipuri, etc. Loc în care se poate pune orice informație ce nu se potrivește în unul din compartimentele predefinite.

3.1.8 Caracteristici avansate ale claselor

- O **clasă template** (clasa parametrizată) permite crearea unei varietăți de clase utilizând parametri pentru a descrie fiecare tip specific de clasă.

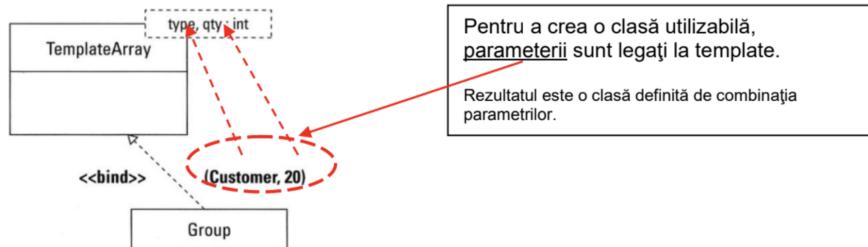


Figure 8: Clasa template

- O interfață permite definirea unui mod comun de a interacționa cu diferite clase. Modul de a invoca un comportament se numește operație sau sig-

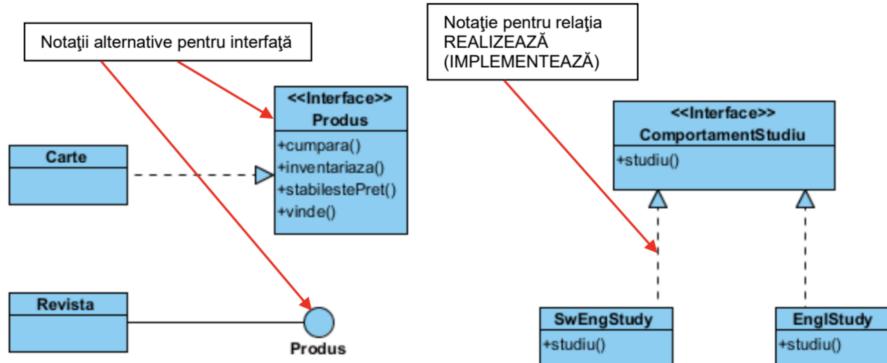


Figure 9: Notații pentru interfețe

natura operației. Signatura unei operații include nume, parametri și rezultat returnat.

Combinația unică a acestor elemente definește o interfață.

Implementarea unei interfețe⁹ este oferită de clasa care realizează sau implementează interfața.

- O clasă abstractă este o clasă ce nu poate fi instanțiată. O clasă abstractă se modelează prin scrierea italică a numelui clasei, prin specificarea proprietății *{abstract}* în compartimentul cu numele clasei sau prin adăugarea proprietății *{abstract}* la operația care nu are metodă.

3.2 Relații

O aplicație software necesită un set de resurse (obiecte).

- Pentru a le utiliza, fiecare resursă trebuie descrisă utilizând o definiție de clasă.

Clasele descriu tipurile de resurse, scopurile lor și caracteristicile pe care le oferă, inclusiv atribută și operații.

- Pentru a coordona interacțiunea acestor resurse, trebuie explicitat modul în care ele pot comunica. Pentru a comunica ele trebuie să fie conștiente una de alta.

3.2.1 Asociere

Modelul unei asocieri conține:

- clasele participante.
- o conexiune
- 2 capete ale asocierii - *Cel mai cunoscut tip de asociere este asocierea binară. Se pot modela însă și asocieri n-are.*
- reguli care guvernează asocierea:
 - nume (semantică)
 - rolurile claselor participante
 - multiplicitate
 - (alte) constrângeri

Numele

- Exprimă intenția asocierii.
- Uzual – un verb sau o construcție verbală.
- Nu generează cod

Numele unei asocieri devine și mai important dacă două clase au mai multe motive de colaborare decât între ele există mai multe relații de asociere.

Exemplu 10: O sală de spectacol poate sponsoriza un spectacol în timp ce altele găzduiesc spectacolul. Altădată o sală de spectacol poate fi atât sponsorul căt și gazda unui eveniment. Figura următoare utilizează două asociații pentru a reprezenta faptul că regulile de găzduire a unui eveniment diferă de regulile de sponsorizare a evenimentului. Cele două asocieri trebuie definite și gestionate separat.



Fiecare asociere reprezintă un set separat de reguli.

Figure 10: Un exemplu de asociere

Scopul numelui unei asocieri este de a defini clar și concis scopul relației dintre cele două tipuri de obiecte. Acest scop va ghida definirea comunicării dintre obiecte și definirea rolului pe care fiecare obiect îl joacă în cadrul acestei comunicări.

Rolul

- Explică modul în care obiectele participă în relație
- Poate genera cod

Multiplicitatea reprezintă numărul valid de obiecte ce pot intra în relație cu un obiect asociat.

Multiplicitatea asignată unui capăt al unei asocieri definește numărul de obiecte ce pot fi asociate cu un singur obiect de la celalalt capăt al asocierii. Cuvântul cheie ordered se utilizează pentru a specifica faptul că obiectele din grup trebuie aranjate în secvență.

Constrângere – definește o restricție ce trebuie respectată de către un element de modelare pentru a se asigura integritatea acestuia pe toată durata de viață a sistemului.

Format – un text (într-un limbaj specific) ce poate fi atașat la aproape oricare element din model. Limbajul de exprimare a constrângerii poate fi OCL (Object Constraint Language), un limbaj de programare sau chiar un limbaj natural (ca Engleză sau Română).

Constrângerile documentează cerințele de **IMPLEMENTARE** pentru capătul asocierii la care sunt plasate. Ele sunt implementate în metode care crează și modifică acele referințe la obiecte care reprezintă relația.

Navigabilitatea descrie necesitatea ca un obiect să acceseze un alt obiect prin "navigare de-a lungul unui link." Obiectul de la celălalt capăt poate accesa obiectul de la capătul navigabil. Navigabilitatea se modeleză cu o săgeată plasată la capătul navigabil. Într-o asociere fără săgeți ambele capete sunt navigabile.

3.2.2 Asocieri speciale

- **Asociere reflexivă**

O asociere reflexivă modeleză relația dintre obiecte din aceeași clasă.



Figure 11: Asociere ad reflexiva

- **Agregare**

Definește un ansamblu sau o configurație de elemente ce formează o unitate complexă (agregat). Un obiect agregat reprezintă interfața la ansamblu și își asumă responsabilitatea pentru coordonarea comportamentului agregării.

De exemplu, putem modela sub formă de agregare o echipă pentru dezvoltarea unui proiect. Clasele reprezintă persoanele care sunt "conectate" prin acorduri și alocări sarcini.

Echipa poate avea un punct de control, reprezentat de exemplu prin conducătorul de proiect, care coduce activitatea echipei. Dacă echipa este repartizată la o nouă locație sau dacă i se alocă un proiect nou, fiecare membru al echipei participă la relocare sau la activitățile noului proiect. Ei funcționează împreună ca un tot unitar condus de conducătorul de proiect.

Agregatul reprezintă întreaga configurație și oferă un unic punct de control al elementelor componente.

Ansamblul pare să funcționeze ca un singur obiect – primește instrucțiuni ce afectează întreaga colecție de obiecte și dictează modul în care membrii colecției vor răspunde.

Interfața obiectului agregat este invocată din exterior iar restul este gestionat în cadrul ansamblului.

Exemplu:



Interpretarea modelului:

- Un agent poate fi parte a uneia sau mai multor agenții, dar poate fi și independent (relație de agregare).
- Agenția este compusă din cel puțin un agent dar numărul lor poate fi oricără de mare (multiplicitate).
- Un agent este considerat un angajat al agenției (nume rol).
- Fiecare agent trebuie să aibă un contract valid pentru a fi angajat al agenției (constângere).

Figure 12: Exemplu Aggregare

- **Compoziție**

Compoziția (aggregare tare) = agregare în care durata de viață a obiectului membru depinde de durata de viață a agregatului.

Agregatul controlează comportamentul membrului inclusiv crearea și distrugerea acestuia (obiectul membru nu poate exista în afara agregatului).

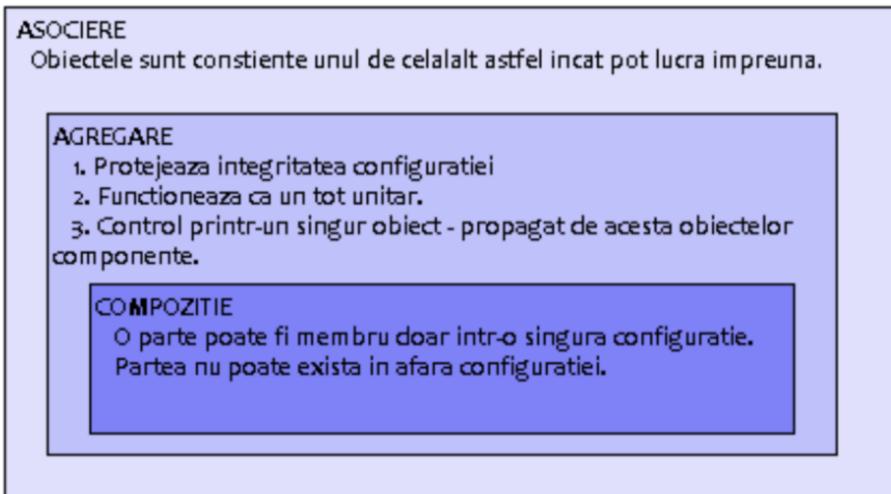


Figure 13: Diagrama Gradată de Cuplare

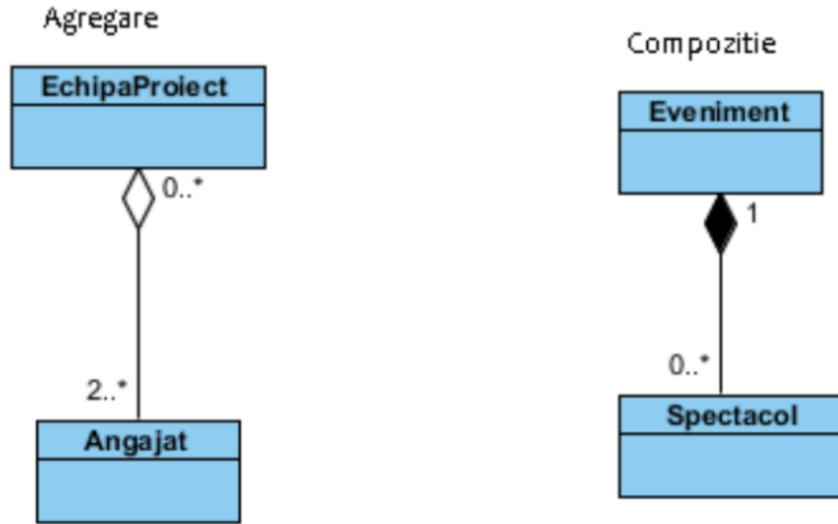


Figure 14: Exemplu de Agregare și Compoziție

3.2.3 Generalizare

Relație în care obiectele unei **clase specialize** (subclasă) sunt substituite pentru obiecte al unei **clase mai generale** (superclasă).

Relația de generalizare între clase implică proprietatea de moștenire. Moștenirea este mecanismul prin care elemente specifice încorporează structura și comportamentul unor elemente mai generale.

Generalizarea pune în relație clase care fiecare conține un subset de caracteristici necesare definirii unui tip de obiect. Prin instanțierea tuturor seturilor de caracteristici din fiecare clasă de pe o cale verticală a arborelui de generalizare rezultă un obiect.

Exemplu 15: pentru a crea obiectul Drama, trebuie combinate clasele Drama, PiesaTeatru și Eveniment pentru a obține toate caracteristicile — atributele, operațiile și asociările — care definesc obiectul Drama. Din setul combinat de caracteristici se va crea un obiect de tip Drama.

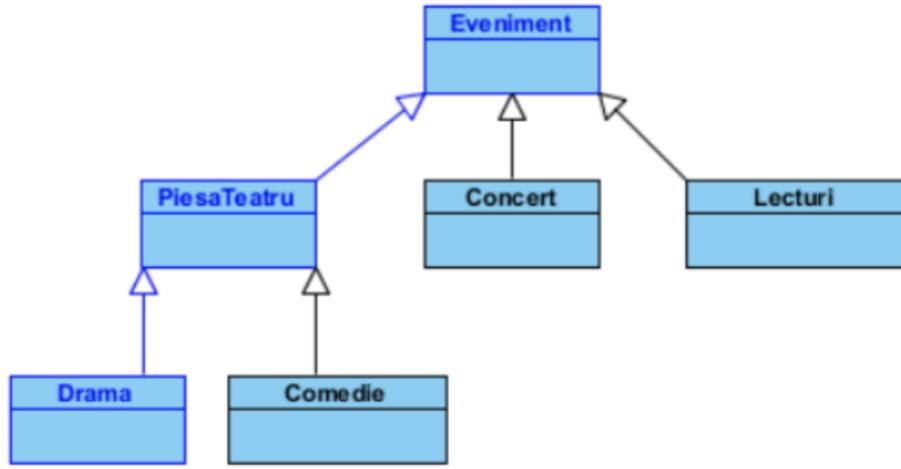


Figure 15: Exemplu de Generalizare

3.2.4 Dependență

Dependență reprezintă o relație de tip client-furnizor în care o schimbare la furnizor induce o schimbare la client.

Dependență poate, de asemenea, reprezenta o relație de precedență.

Notăția pentru dependență

O dependență se modelează utilizând o săgeată punctată între elementele modelului (unul sau mai mulți clienți și unul sau mai mulți furnizori). UML predefineste patru tipuri generale de dependență: abstractizare, legare, permisiune și utilizare.

3.3 Feluri speciale de clase

Clasă utilitară (utility)

- Nu are instanțe.
- Reprezintă o colecție de atribute și operații statice (la nivelul clasei).

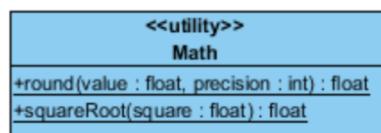


Figure 16: Exemplu de clasă utilitară

Enumerare

- Tip de dată definit de utilizator, care definește un set de valori constante.
- Valorile definite într-o clasă de enumerare sunt folosite de obicei pentru validări.

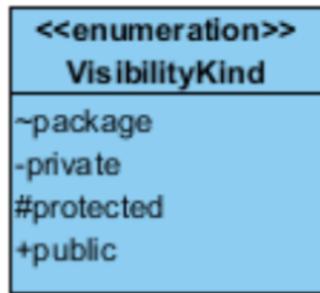


Figure 17: Exemplu de Enumerare

Exemplul din Figura 17 arată enumerarea `VisibilityKind` utilizată pentru definirea vizibilității de către specificația UML. Ea definește setul de valori valide pentru setarea domeniului de acces la atribut, la operații și la alte elemente ale modelului. Fiecare intrare este un literal, un sir static de caractere text (ex. "public") care reprezintă o opțiune validă pentru o anumită valoare.

4 CORESPONDENȚA UML - JAVA

În continuare sunt ilustrate principalele construcții UML din diagrama de clase și corespondența lor în cod Java.

Codul Java prezentat alături de clasă nu este codul final, ci acel extras care implementează construcția UML ilustrată.

Clasă

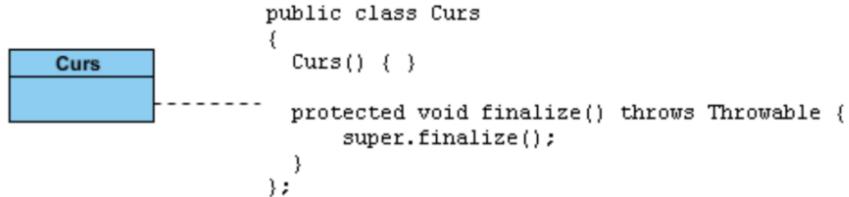


Figure 18: Corespondență clasă JAVA

Atribute și operații la nivel de clasă

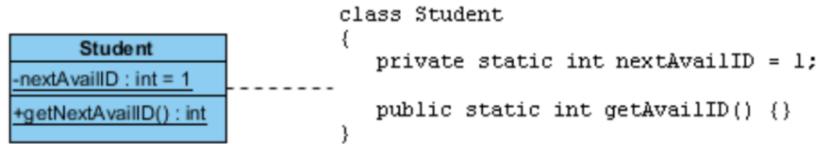


Figure 19: Corespondență atribute și operații JAVA

Multiplicitate atribute

Când un atribut conține mai multe valori el se implementează cu un tip de array sau cu o referință la o clasă de tip container.

Vizibilitate pentru atribute și operații

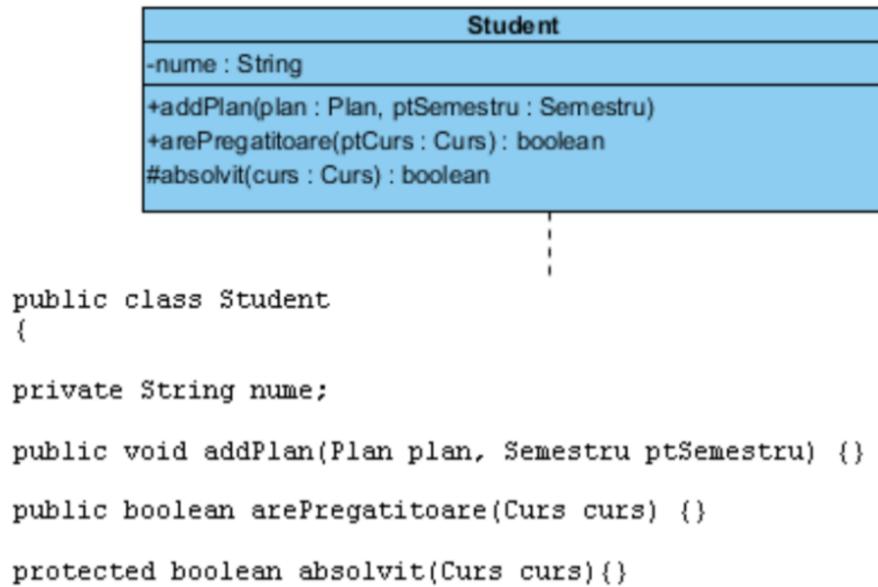


Figure 20: Corespondență pentru vizibilitate

Asociere

Pentru ca un obiect să lucreze, el trebuie în mod tipic să interacționeze cu alte obiecte.

Pentru aceasta, o clasă trebuie să definească setul valid de relații și regulile pentru stabilirea și întreținerea acestor relații.

Similar, clasele trebuie să definească mijloace prin care fiecare obiect să contacteze alte obiecte astfel încât să poată colabora. Clasa definește un atribut care poate conține o referință la alt obiect. Referințele la tipuri de obiecte diferite vor fi păstrate în attribute diferite.

O referință poate fi la o colecție de obiecte, toate pentru același scop. În acest caz, un singur atribut păstrează referințe multiple. În UML acest lucru se modeliază cu multiplicitatea. Într-un limbaj de programare se folosește un array sau un tip de clasă container (ex. În Java un subtip de Collection cum ar fi Set).

ATENȚIE ! Pentru fiecare relație de asociere de pe diagrama de clase, programatorul (sau facilitatea generator din instrumentului de modelare) trebuie să genereze un atribut de tip referință sau colecție de referințe.

Asociere bi-directională

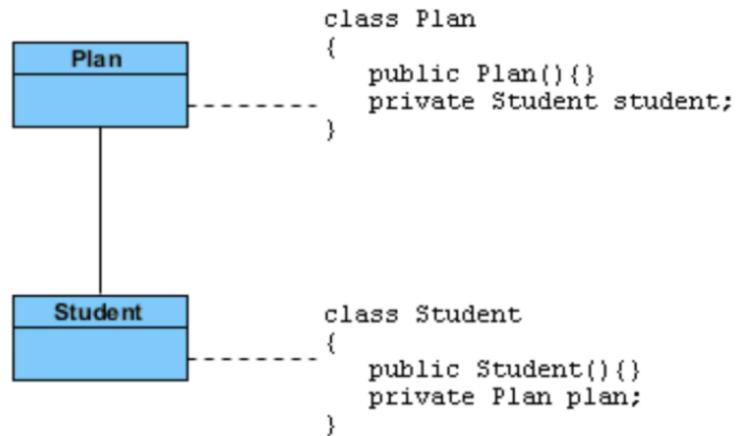


Figure 21: Corespondență pentru asocierea bi-directională

Asociere uni-directională

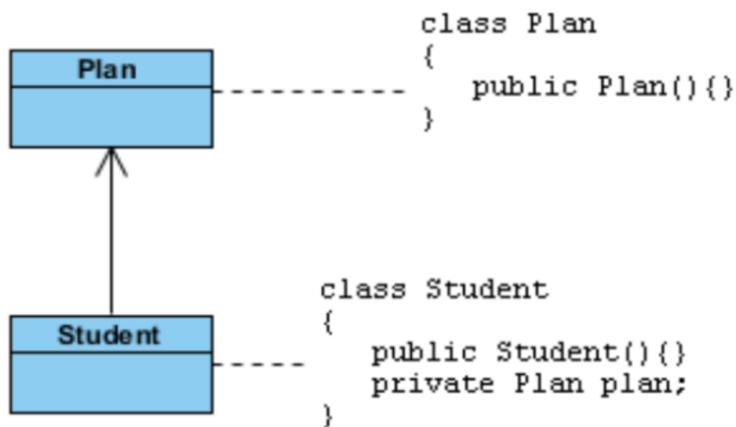


Figure 22: Corespondență pentru asocierea uni-directională

Implicit navigabilitatea este definită pentru ambele capete. Dacă navigabilitatea la unul din capete nu este necesară, se pot evita scriere de cod și operații de întreținere suplimentare prin specificarea asocierii uni-directionale.

Roluri într-o asociere Așa cum am văzut, fiecare obiect trebuie să păstreze o referință la obiectul sau obiectele asociate. Referința este ținută într-un atribut al obiectului. Pentru o asociere există un atribut care ține referința.

În codul generat, atributul va primi numele rolului obiectului referit.

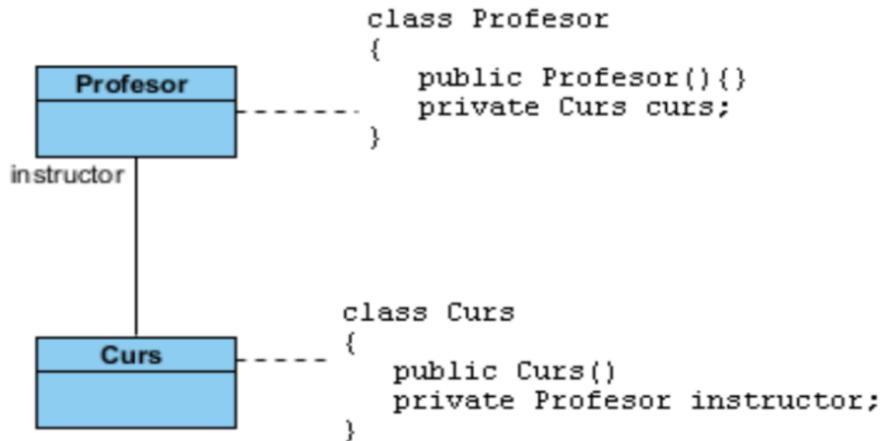


Figure 23: Corespondență pentru rol

Multiplicitate la asociere

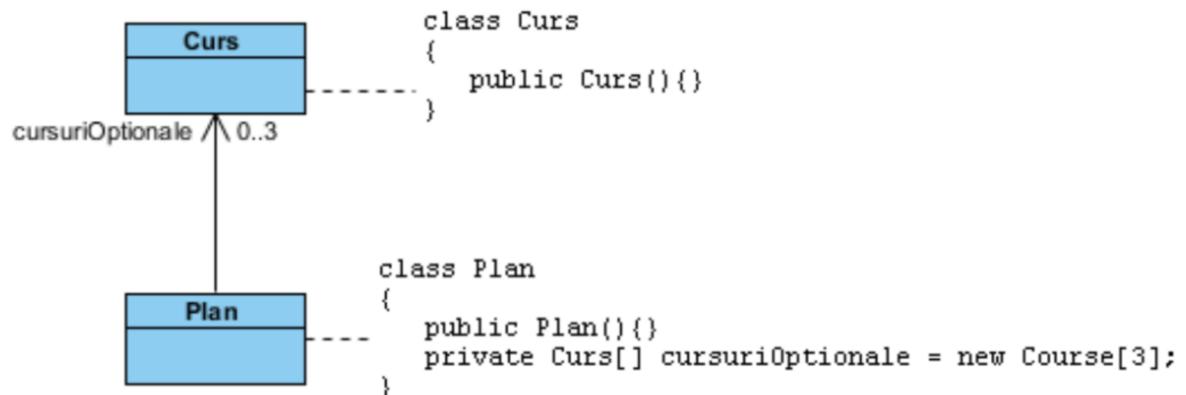


Figure 24: Corespondență pentru multiplicitate

Asociere reflexivă

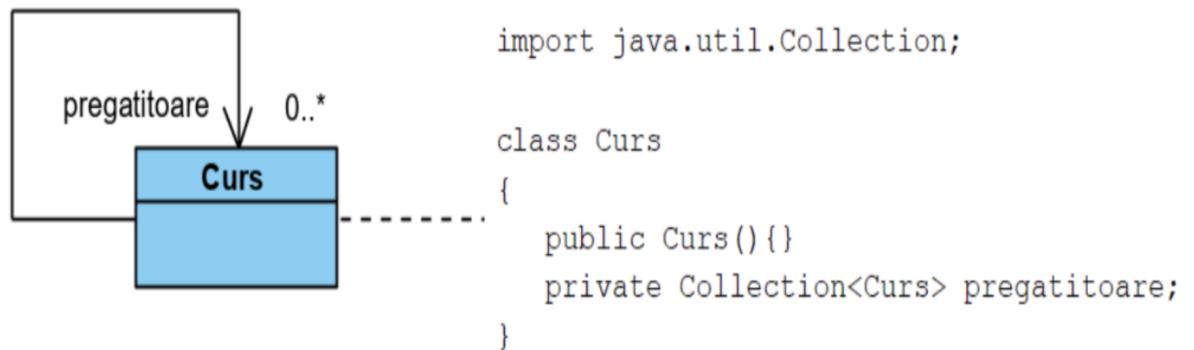


Figure 25: Corespondență pentru asocierea reflexivă

Agregare

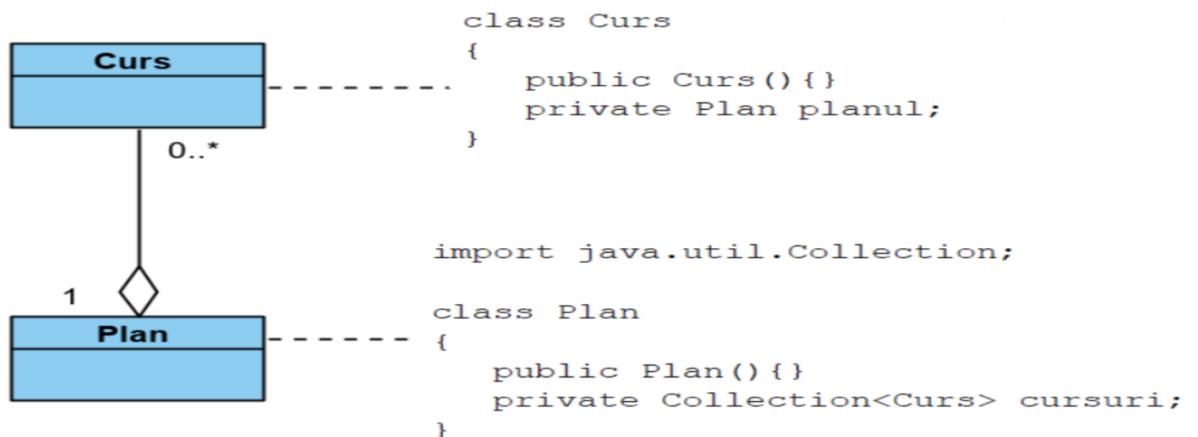


Figure 26: Corespondență pentru Aggregare

Compoziție

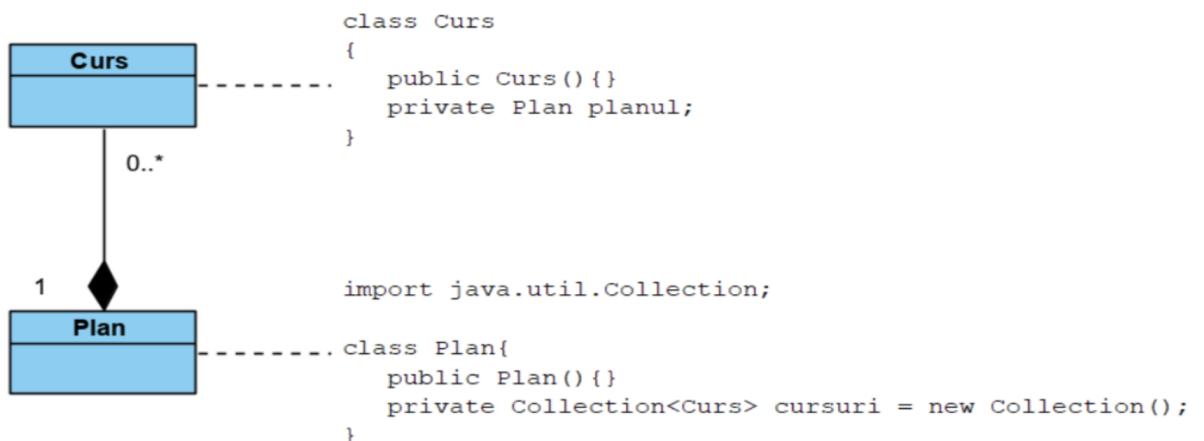


Figure 27: Corespondență pentru Compoziție

Interfață și relația de realizare

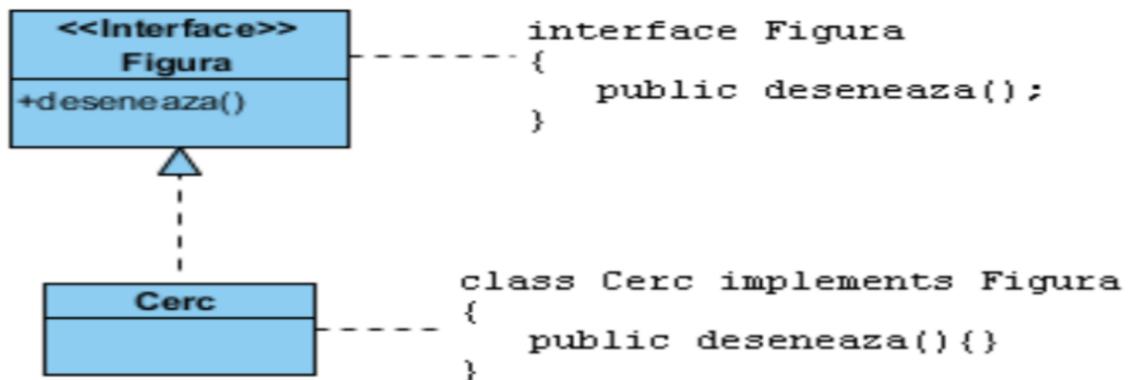


Figure 28: Corespondență pentru realizare

Generalizare

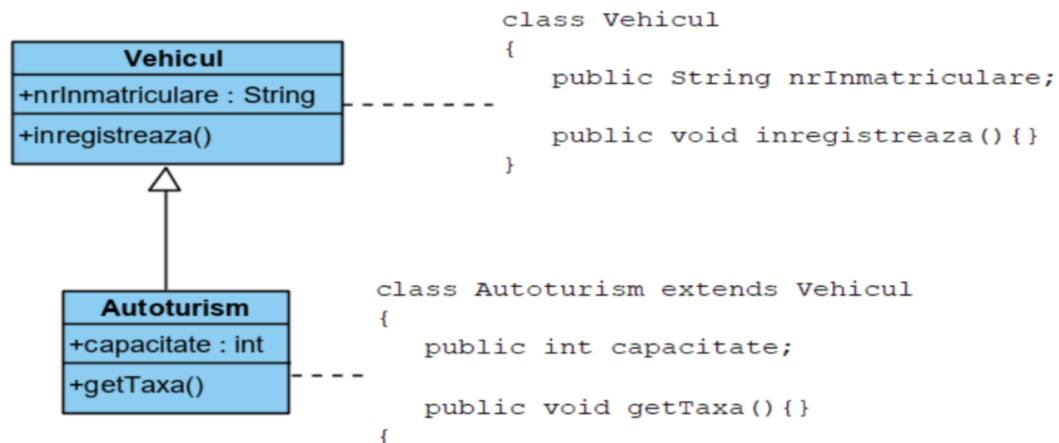


Figure 29: Corespondență pentru generalizare

Clasă abstractă

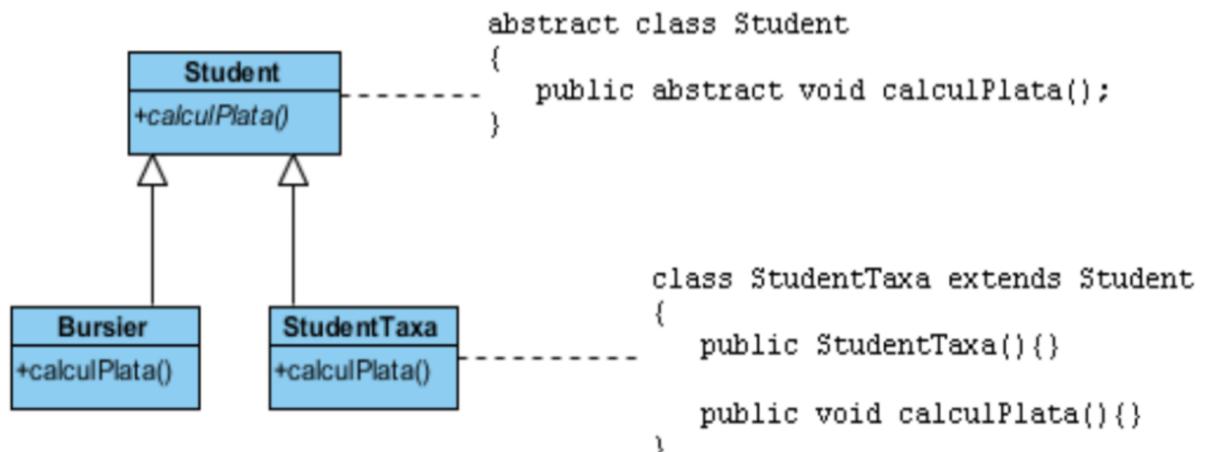


Figure 30: Corespondență pentru clasă abstractă

Dependență

Implementarea în cod pentru relația de dependență are mai multe variante.

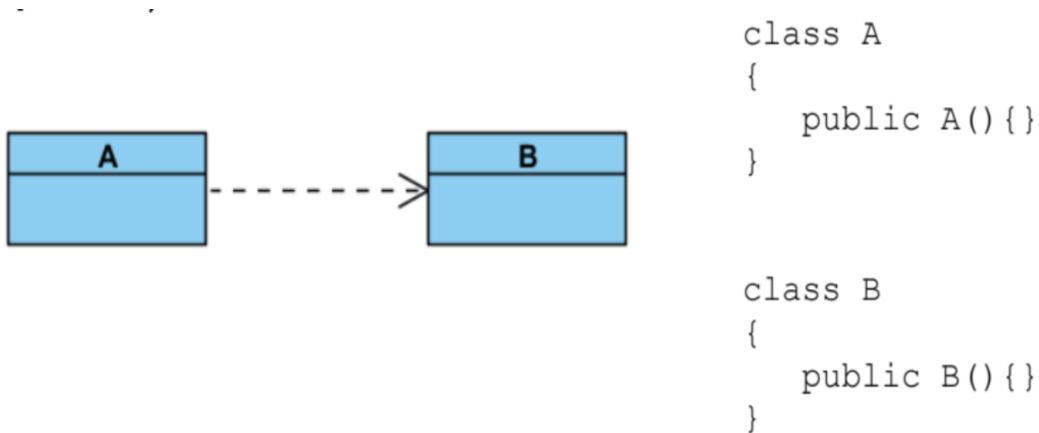


Figure 31: Corespondență pentru dependență

Această relație presupune fie transferul unui obiect ca argument al unei operații (a) sau crearea unui obiect de un anumit tip în cadrul unei operații (b).

În acest exemplu, pentru clasa A putem avea una din următoarele variante:

- public oOperatie(B b) {...}
- public oOperatie() {B myB = new B(); ... }

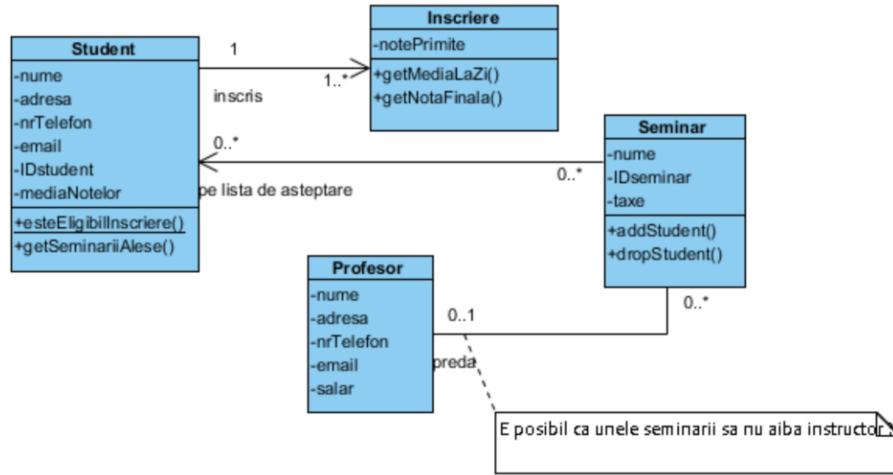
În cazul relației de dependență nu putem deduce doar din reprezentarea de mai sus care variantă va fi utilizată.

Varianta (a) poate fi dedusă doar dacă în clasa A este reprezentată, cu tot cu parametri, operația oOperatie.

5 ACTIVITATE INDIVIDUALĂ

Studiați <https://www.visual-paradigm.com/features/uml-and-sysml-modeling/> secținea Class diagram.

Exercițiul 1: Reprezentați utilizând VP for UML și explicați toate elementele sintactice din următoarea diagrame.



Exercițiul 2: Rezolvați testele din pachetul aflat în fișierul IS_clase.pdf. Pentru fiecare exercițiu indicați varianta(e) corectă(e) și **explicați de ce fiecare din celelalte variante este incorrectă**.

Exercițiul 3:

(a) Desenați o diagrame de clase ce reprezintă o carte definită astfel: “O carte este compusă dintr-un număr de părți, care la rândul lor sunt compuse dintr-un număr de capituloare. Capitolele sunt compuse din secțiuni.” Concentrați-vă numai pe clase și relații.

(b) Adăugați multiplicitate.

(c) Extindeți diagramea de clase pentru a include următoarele attribute:

- cartea are un editor, o dată de publicare și un ISBN
- fiecare parte are un titlu și un număr
- fiecare capitol are un titlu, un număr și un rezumat
- fiecare secțiune are un titlu și un număr

(d) Observați că toate clasele Parte, Capitol și Secțiune includ un atribut titlu și un atribut număr. Aplicând procedeul de abstractizare extrageți aceste attribute comune într-o nouă clasă. Folosiți apoi relația de generalizare pentru a lega această clasă (superclasă) de clasele pe care le generalizează (subclase).

(e) Scrieți codul Java pentru clasele diagramei: attributele (care vor avea vizibilitate private) și metodele de acces (de tip get() și set()) la aceste attribute.

Definiți operațiile și scrieți metodele pentru ca unei cărți să i se poată adăuga părți, unei părți să i se poată adăuga capitole și unui capitol să i se poată adăuga secțiuni.

Exercițiu 4:

(a) Desenați o diagramă de clase în care să reprezentați relația dintre părinți și copii. Țineți cont de faptul că o persoană poate avea atât părinți cât și copii. Indicați rolurile și multiplicitatele pentru relațiile de asociere.

(b) Scrieți codul Java ce va conține atributele clasei și metodele de acces la acestea. Scrieți o clasă main care va instanția mai multe persoane, va stabili relații de rudenie între acestea și va afișa părinții și copiii uneia din ele.

Indicație: Construieți o clasă Persoana și adăugați-i două relații de asociere reflexivă (sau o asociere reflexivă bidirectională) pentru a modela relațiile persoanei cu părinții și cu copiii săi.

Exercițiu 5:

O companie e compusă din departamente. Departamentele sunt localizate în unul sau mai multe birouri. Unul dintre birouri este are rol de sediu central. Fiecare departament are un manager care este recrutat dintre angajații.

(a) Modelați sistemul ce reprezintă compania cu o diagramă de clase ce constă din toate clasele din sistem, atributele lor, relațiile dintre clase, specificarea multiplicităților și alte elemente de model pe care le considerați potrivite.

(b) Scrieți codul Java ce va conține atributele claselor și operațiile de acces la acestea. Scrieți o clasă main care va instanția componentele întreprinderii, va stabili relațiile dintre acestea și va afișa structura întreprinderii.

Exercițiu 6:

O rețea locală (LAN) bazată pe token-ring este o rețea formată din noduri. În cadrul rețelei sunt transmise pachete. Fiecare nod are un nume unic în cadrul rețelei și o referință la nodul următor.

Există diferite tipuri de noduri: stațiile de lucru sunt noduri de origine ale mesajelor; serverele și imprimantele sunt noduri ce pot recepta mesaje. Pachetele conțin ca informații originea, destinația și conținutul mesajului, și sunt transmise în rețea. O rețea LAN este o configurație circulară de noduri.

(a) Desenați o diagramă de clase ce constă din toate clasele sistemului, atributele și relațiile lor, specificarea multiplicităților și alte elemente de modelare pe care le considerați potrivite.

(b) Adaptați diagrama de clase de la punctul (a) pentru modela o rețea LAN în care se găsesc două tipuri de imprimante: imprimantele ASCII pot imprima doar documente ASCII. Imprimantele PostScript pot imprima atât documente ASCII cât și documente PostScript. Documentele sunt trimise în rețea sub formă de conținut de pachete.

(c) Scrieți codul Java ce va conține atributele claselor și operațiile de acces la acestea.