Modelare cu clase

Adapted after :

Timothy Lethbridge and Robert Laganiere,
Object-Oriented Software Engineering –
Practical Software Development using UML and Java, 2005
(chapter 5)

5.1 Ce este UML?

UML (engl. *Unified Modelling Language*) este un limbaj grafic pentru modelarea sistemelor software orientate pe obiecte.

- La sfarsitul anilor ‘80 si inceputul anilor ’90 au aparut primele metodologii de dezvoltare orientata pe obiecte (OO).
  - —Proliferarea metodelor si notatiilor OO a condus la confuzie in industria software.
  - —In scurt timp s-a simtit necesitatea unei unificari si chiar a unei standardizari in acest domeniu.
- UML a fost dezvoltat incepand cu 1994, fiind in principal rezultatul eforturilor conjugate a trei persoane: Grady Booch, James Rumbaugh si Ivar Jacobson [BRJ99,JBR99,RJB99].
  - —In 1997, OMG (Object Management Group) a inceput procesul de standardizare a UML.

© Lethbridge/Laganière 2005

Diagrame UML

- Diagrame de clase
  - —Vizualizeaza clase si relatii intre clase
- Diagrame de obiecte
  - —Vizualizeaza colectii de obiecte si conexiuni intre obiecte
- Diagrame de cazuri de utilizare
  - —Vizualizeaza cazuri de utilizare, actori si relatii intre acestea
- Diagrame de interactiune (secventiere sau comunicare)
  - —Vizualizeaza comportamentul aratand modul in care interactioneaza obiectele unui sistem
- Diagrame de stare si diagrame de activitate
  - —Vizualizeaza comportamentul (intern al) unui sistem
- Diagrame de componente si diagrame de desfasurare
  - —Vizualizeaza configuratia logica si fizica a componentelor de calcul

© Lethbridge/Laganière 2005

Modelare

Un model ar trebui

- Sa utilizeze o notatie standard
- Sa fie usor de inteles de catre toti participantii la proiect
- Sa poata fi exprimat la diverse nivele de abstractizare

Modelarea faciliteaza

- Vizualizarea si documentarea sistemului
- Specificarea structurii si comportamentului
- Construirea de schite de proiectare

Limbajul UML furnizeaza o notatie standard pentru reprezentarea structurii si comportamentului sistemelor OO.

© Lethbridge/Laganière 2005

5.2 Diagrame de clase UML

Principalele simboluri utilizate in diagramele de clase:

- Clase
  - Reprezinta tipurile obiectelor
- Asocieri
  - Descriu tipul conexiunilor intre obiecte
- Atribute
  - Reprezinta date simple din clase si obiecte
- Operatii
  - Reprezinta functii efectuate de clase si obiecte
- Generalizari
  - Grupeaza clasele in ierarhii de generalizare / specializare

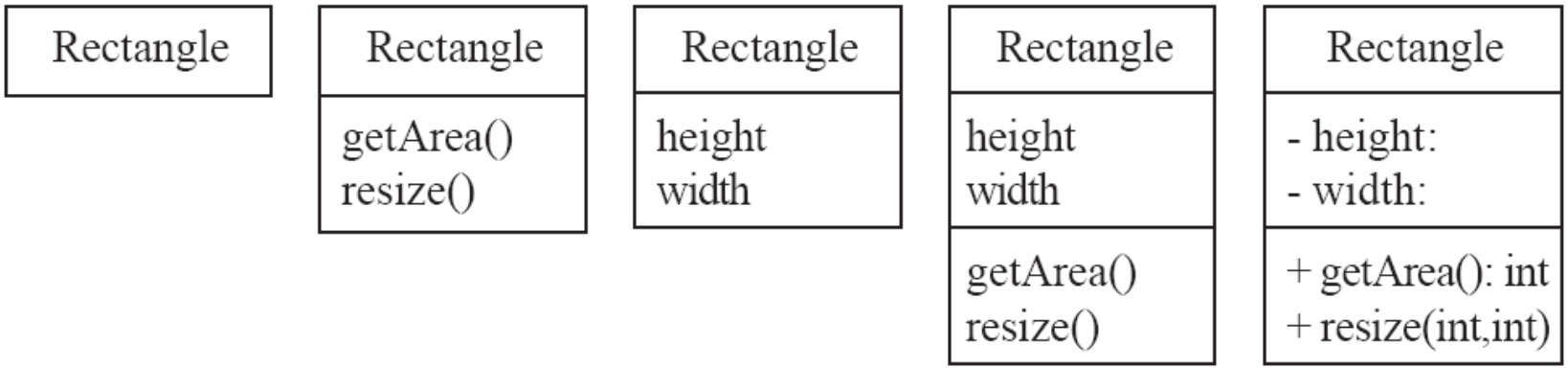
© Lethbridge/Laganière 2005

Clase

Grafic, o clasa este reprezentata ca un dreptunghi, incluzand de obicei numele clasei

- Diagrama mai poate vizualiza atributele si operatiile clasei
- Semnatura completa a unei operatii este:

  numeOperatie(numeParametru : tipParametru …): tipReturnat

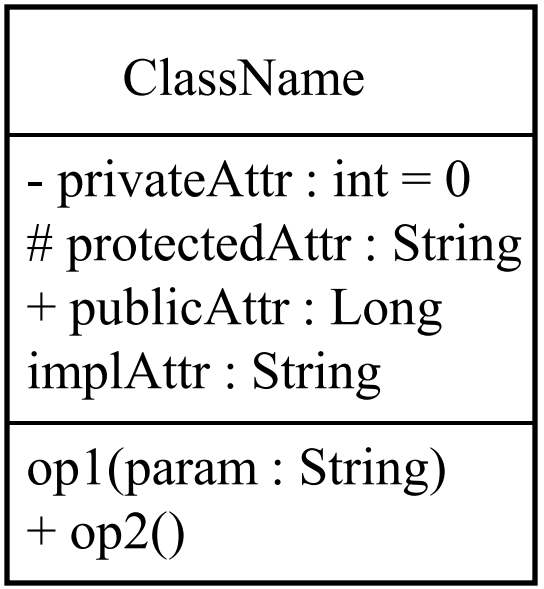


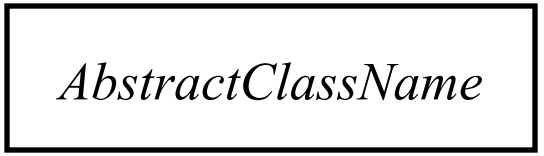
Atribute si operatii

Optiunile de vizibilitate pentru atribute si operatii sunt urmatoarele:

- private (-)
- public (+)
- protected (#)
- pachet sau implementare (nici un simbol)

- Numele claselor si operatiilor abstracte sunt redate in *italics*.



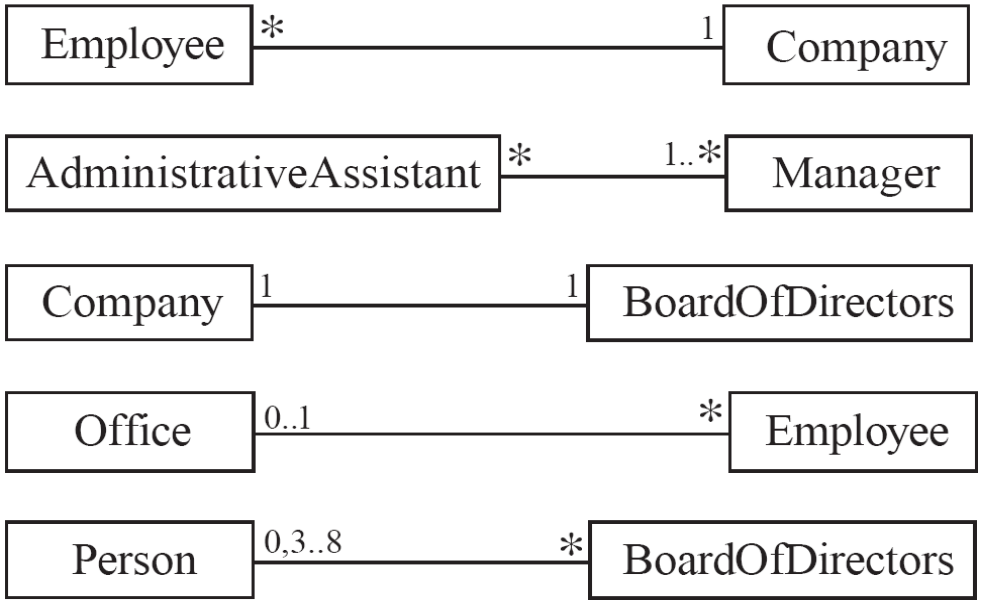


© Lethbridge/Laganière 2005

5.3 Asociere

O asociere este o relatie structurala, care specifica faptul ca obiecte de un anumit tip sunt conectate la obiecte de un alt tip [BRJ99].

- O asociere este reprezentata ca o linie, posibil orientata
  - —La fiecare capat al unei asocieri se poate specifica multiplicitatea
  - —Este posibil, desi *nu este recomadat*, ca multiplicitatea sa ramana nedefinita.



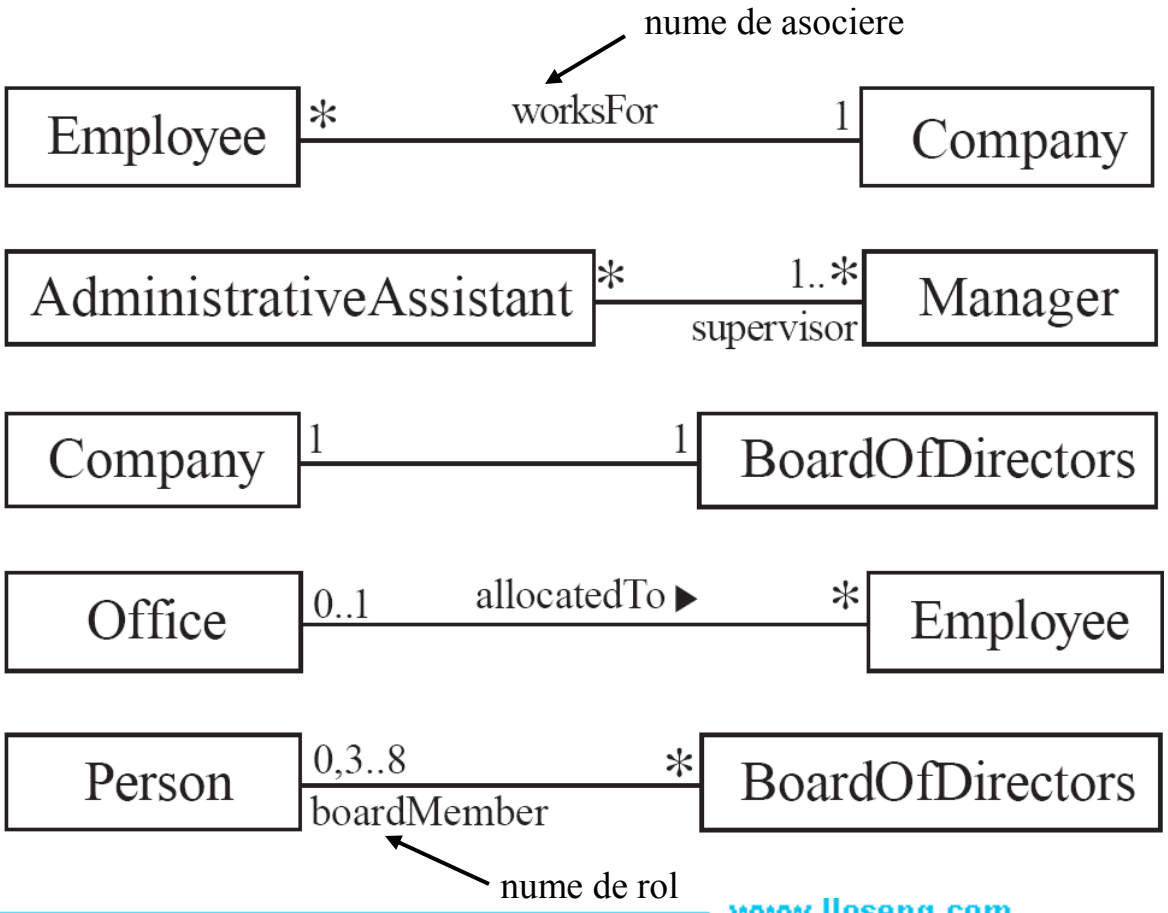
www.lloseng.com

Etichetarea asocierilor

- Fiecare asociere poate fi etichetata, pentru a face explicita natura asocierii.

Exista doua tipuri de etichete (optionale) :

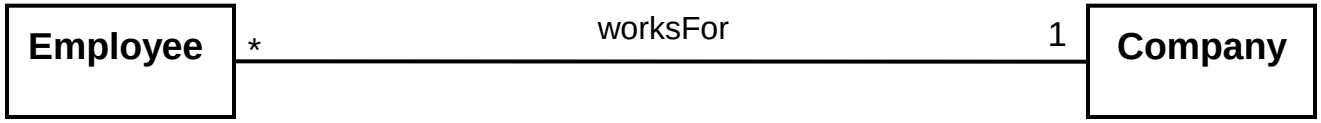
- **nume de asocieri**
  - plasate in mijlocul liniei care reprezinta asocierea
  - exprimate prin verbe (sau expresii verbale)
  - simbolul '▶' poate clarifica 'directia' unei asocieri
- **nume de roluri**
  - plasate la unul sau la ambele capete ale unei asocieri



© Lethbridge/Laganière 2005

Analiza si validarea asocierilor

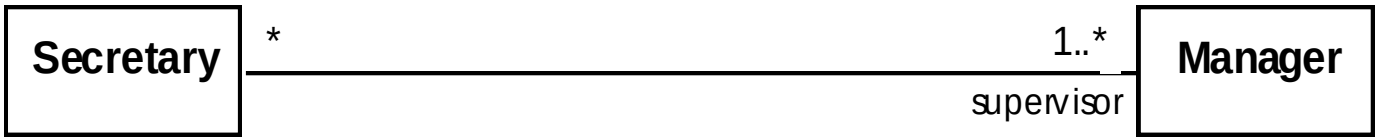
- **Relatii de tip 1-N**
  - —O companie poate avea mai multi angajati
  - —Un angajat poate lucra pentru o singura companie
    - Nn se inregistreaza date despre alte activitati ale angajatilor



© Lethbridge/Laganière 2005

Analiza si validarea asocierilor

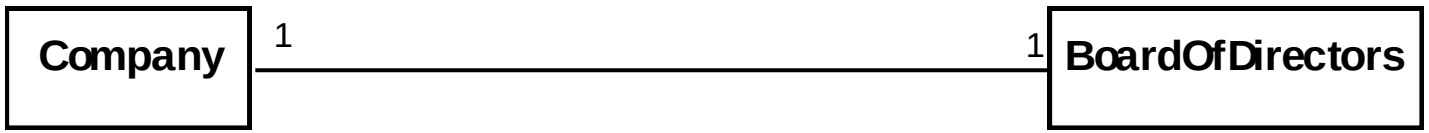
- **Relatii de tip M-N**
  - —O secretara poate lucra pentru mai multi manageri
    - Fiecare secretara are cel putin un manager
  - —Un manager poate avea zero sau mai multe secretare



© Lethbridge/Laganière 2005

Analiza si validarea asocierilor

- **Relatii de tip 1-1 (mai putin comune)**
  - —Pentru fiecare companie (Company) exista exact un consiliu de administratie (BoardOfDirectors).

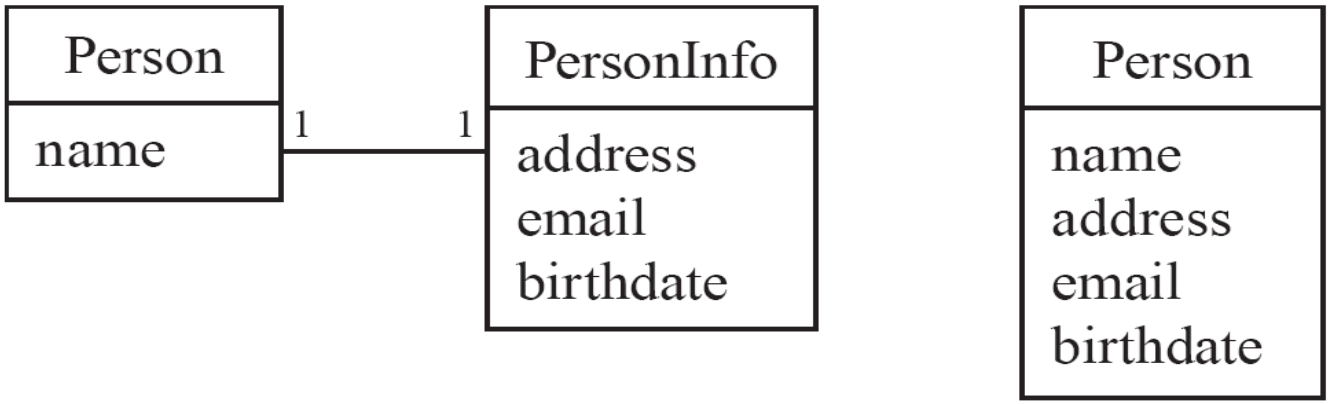


© Lethbridge/Laganière 2005

Analiza si validarea asocierilor

Evitarea asocierilor inutile de tip 1-1.

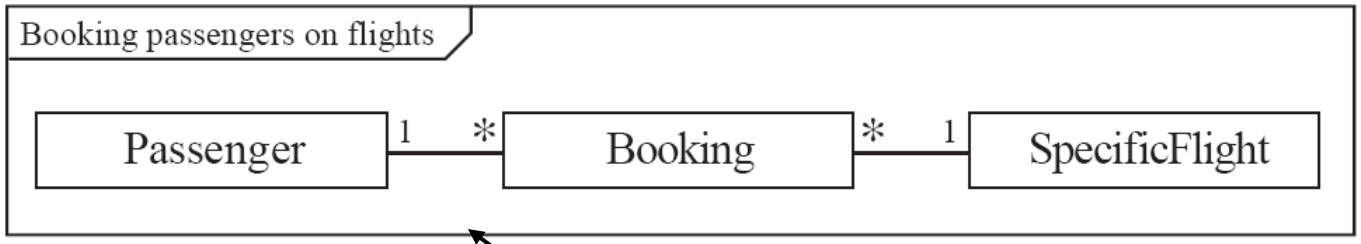
- Relatie 1-1 nepotrivita (in partea stanga)
- Solutia corecta este data in partea dreapta



© Lethbridge/Laganière 2005

Un exemplu mai complex

- O rezervare (Booking) este intotdeauna pentru exact un pasager.
- Un pasager poate avea zero sau mai multe rezervari.



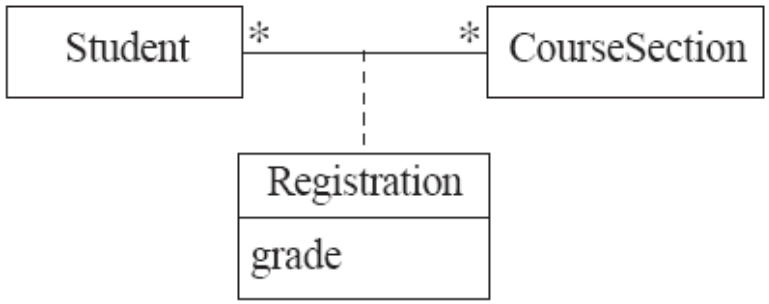
• Aceasta diagrama este inclusa intr-un **cadru** (engl. *frame*) cu **eticheta** 'Booking passengers on flights'

- Aceasta este o facilitate UML 2.0
- Orice diagrama UML 2.0 poate utiliza cadre

© Lethbridge/Laganière 2005

Clase de asociere

- Uneori, un atribut care priveste doua clase asociate nu poate fi plasat in nici una dintre clase. Solutia este sa se creeze o *clasa de asociere* (de exemplu **Registration**) care sa pastreze respectivul atribut (de exemplu **grade**).
- Cele doua diagrame date mai jos sunt semantic echivalente:



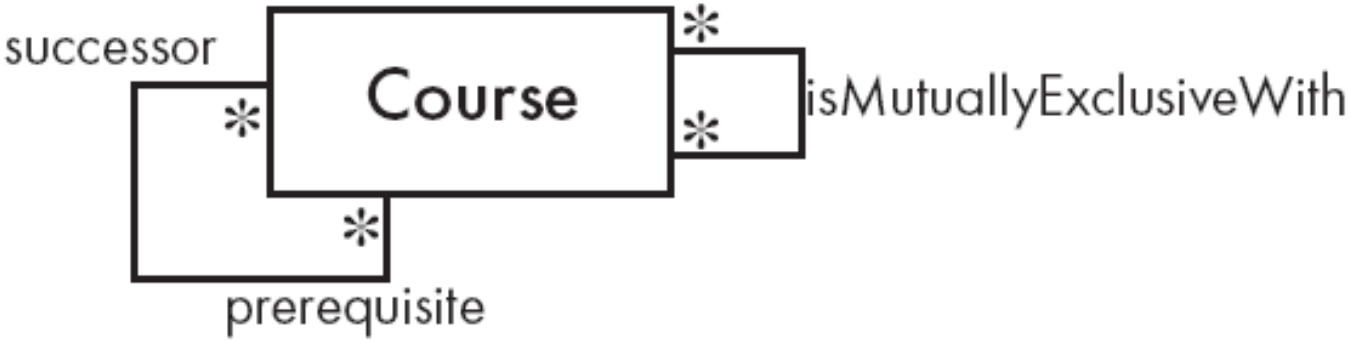
Registration este o *clasa de asociere*.

Exceptand faptul ca este atasata unei asocieri, o clasa de asociere nu difera cu nimic de orice alta clasa.

© Lethbridge/Laganière 2005

Asocieri reflexive

- O relatie de asociere poate fi reflexiva (se pot transmite mesaje intre instante ale aceleiasi clase).
  - —Un curs poate necesita ca alte cursuri sa fi fost urmate inainte ca o conditie esentiala (engl. *prerequisite*).
  - —Doua cursuri sunt mutual exclusive daca acopera (aproape) acelasi material.



© Lethbridge/Laganière 2005

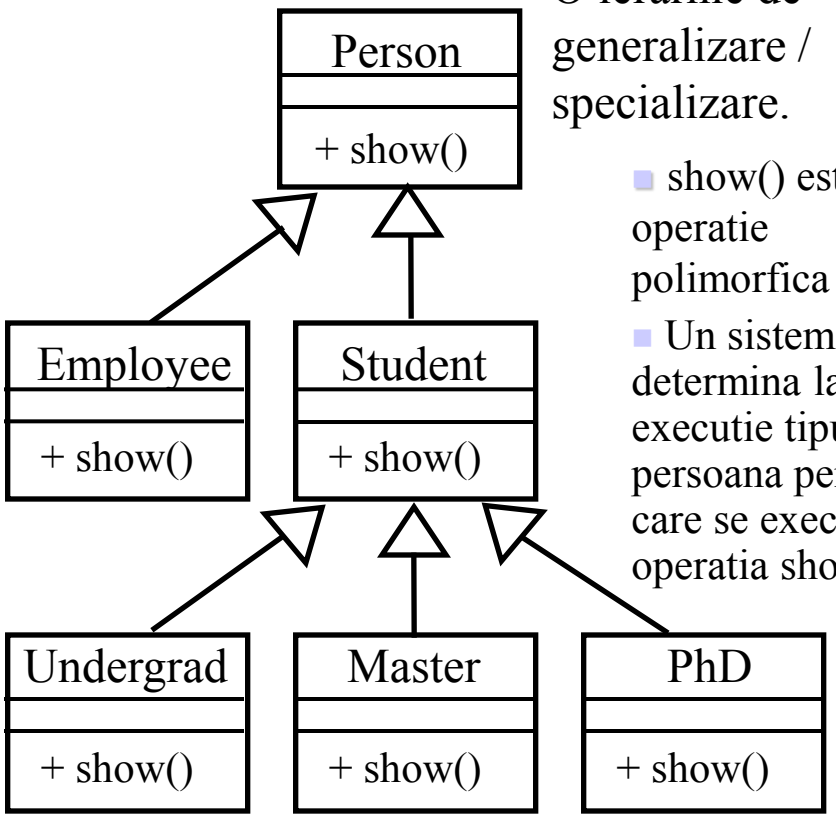
Asocieri unidirectionale

- **Asocierile sunt implicit *bidirectionale*.**
- **Este posibil sa se limiteze navigabilitatea unei asocieri prin adaugarea unei sageti la unul dintre capete; asocierea devine *unidirectionala*.**
  - De exemplu, diagrama data mai jos este proiectata pe baza urmatoarelor presupuneri:
    — utilizatorul poate asocia oricarei zile oricate note;
    — nu este necesar sa se determine ziua careia ii apartine o anumita nota.

© Lethbridge/Laganière 2005

5.4 Generalizare

- ***Generalizarea* (relatia is-a / is-a-kind-of) indica o relatie de *mostenire*.**
- Prin mostenire
  - se evita duplicarea de atribute si operatii prin crearea unei superclase separate care contine comportamentul comun.
  - se poate adauga sau modifica comportament in clasele derivate mai specializate, care "mostenesc" comportament de la clasa de baza.
- Relatia de generalizare este redata ca o linie terminata intr-un mic triunghi gol care indica spre superclasa.
- In ierarhiile de mostenire *comportamentul polimorfic* apare atunci cand o metoda dintr-o clasa derivata are aceeasi semnatura ca si o metoda din clasa parinte.



O ierarhie de generalizare / specializare.

- show() este o operatie polimorfica
- Un sistem OO determina la executie tipul de persoana pentru care se executa operatia show().

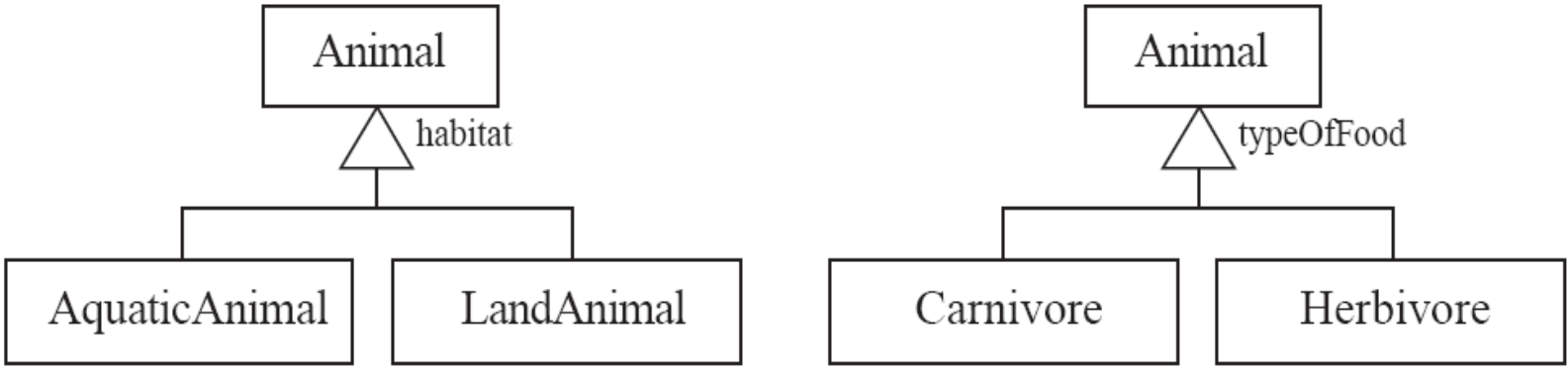
Regula **is-a**:

• Clasa A poate fi o subclasa valida a clasei B doar daca in limba engleza are sens sa se spuna 'an A is-a B' (un A este-un B).

© Lethbridge/Laganière 2005

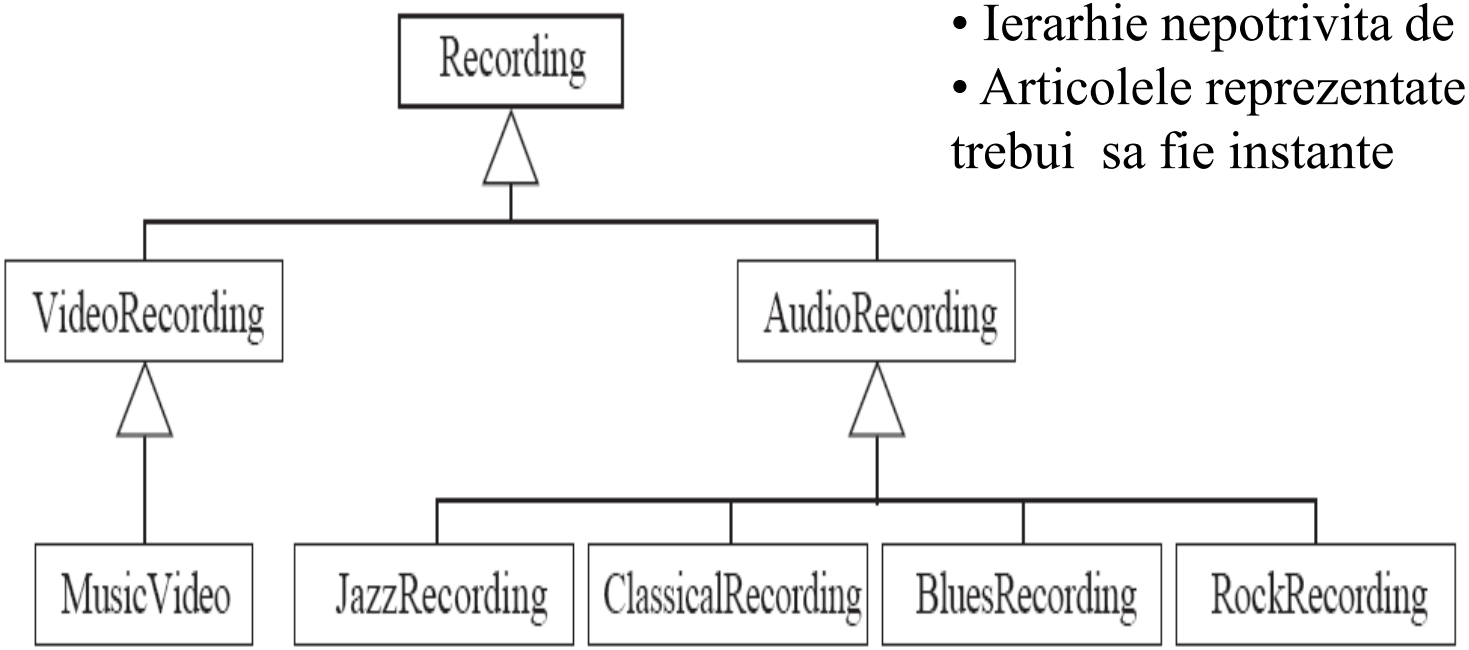
Seturi de generalizare

- Un set de generalizare este un grup etichetat de generalizari cu o superclasa comuna.
- Eticheta (numita discriminator in versiunile mai vechi ale UML) descrie criteriul utilizat la specializarea superclasei in doua sau mai multe subclase.
- In mod tipic, eticheta unui set de generalizare (de exemplu habitat, sau typeOfFood) va fi un atribut care are o valoare diferita in fiecare subclasa.



© Lethbridge/Laganière 2005

Evitarea generalizarilor inutile



- Ierarhie nepotrivita de clase
- Articolele reprezentate ar trebui sa fie instante

- Pentru justificarea existentei unei clase trebuie sa existe cel putin o operatie care se executa in mod diferit in clasa respectiva.
- Majoritatea claselor din aceasta ierarhie se comporta la fel.
  - De exemplu, articolele de tip JazzRecording, ClassicalRecording, BluesRecording si RockRecording nu difera nici prin modul in care pot fi vandute, nici prin tipul de informatie accesibil clientilor.

© Lethbridge/Laganière 2005

Evitarea generalizarilor inutile (continuare)

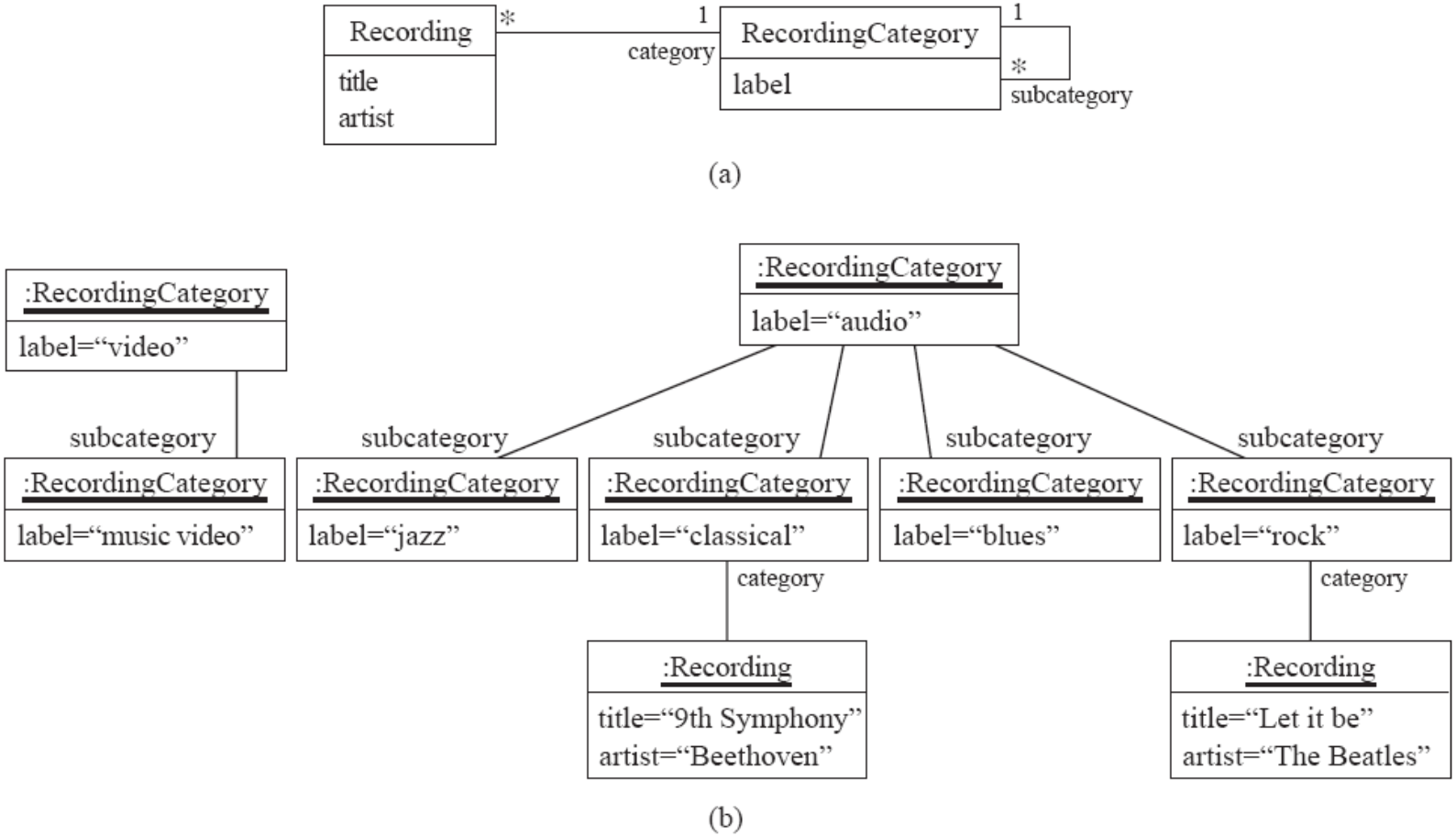


Diagrama de clasa imbunatatita (a), impreuna cu o diagrama corespunzatoare de obiecte (b)

© Lethbridge/Laganière 2005

Seturi de generalizare multiple

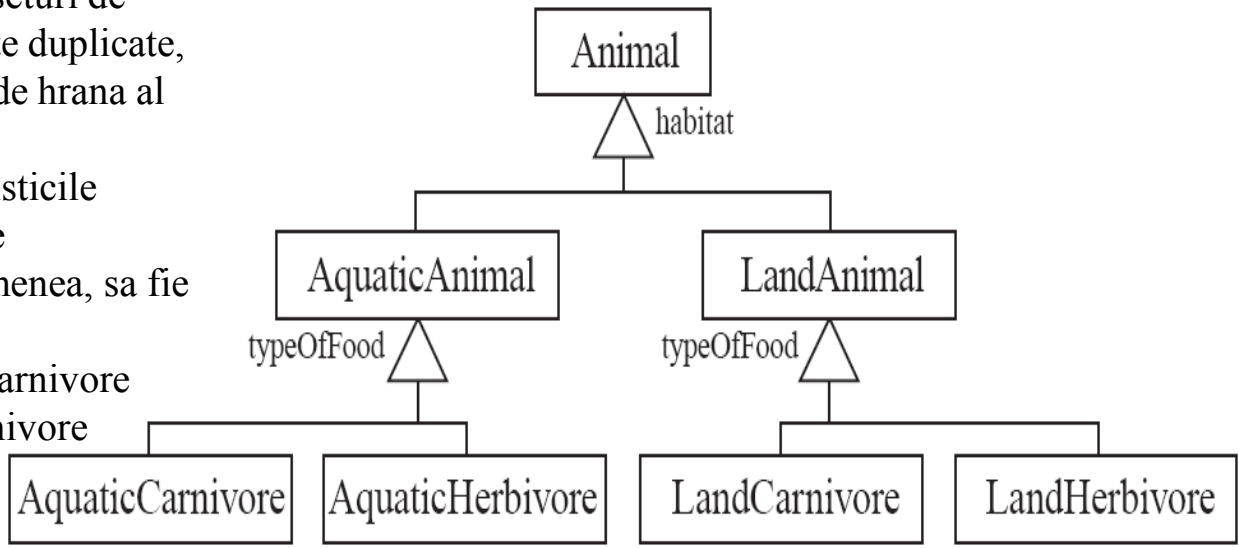
- **Problema de modelare:**
  - —Dorim sa construim o diagrama care sa permita reprezentarea tuturor combinatiilor posibile pentru habitatul si tipul de hrana al animalelor (de exemplu, dorim sa se poata reprezenta carnivore acvatice, cum sunt rechinii).

• **O solutie posibila** are la baza crearea a doua nivele de generalizare:

- •un prim nivel corespunzator habitatului,
- •un al doilea nivel cu seturi de generalizare cu etichete duplicate, corespunzator tipului de hrana al animalelor.

•**Dezavantaj:** toate caracteristicile asociate cu al doilea nivel de generalizare trebuie, de asemenea, sa fie duplicate. De exemplu:

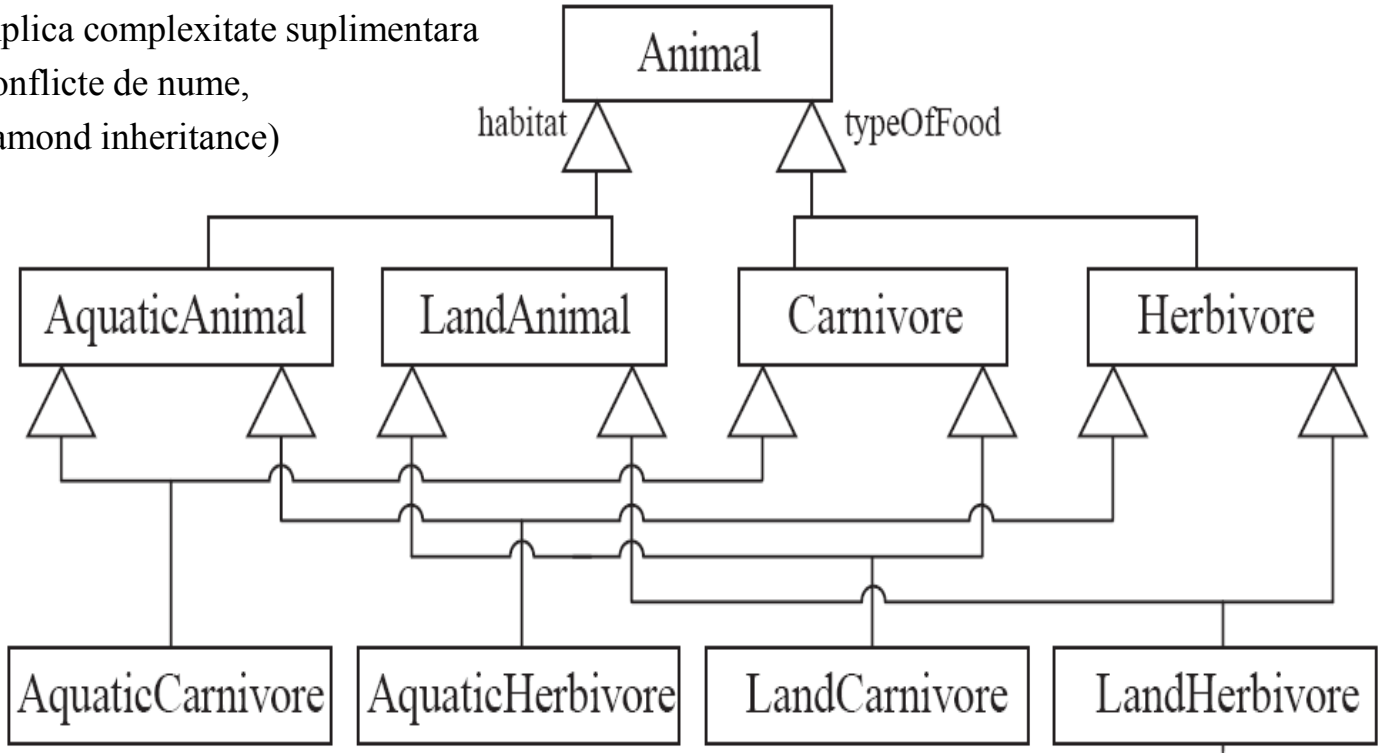
- • prada unui AquaticCarnivore
- • prada unui LandCarnivore



www.lloseng.com

Seturi de generalizare multiple

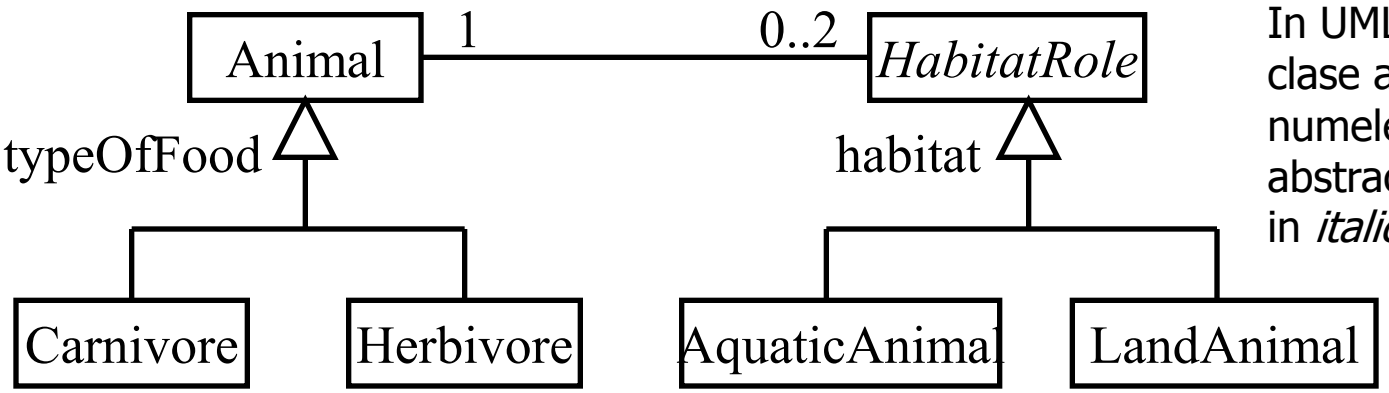
- **Solutie bazata pe mostenire multipla**
  - — Avantaj: evita duplicarea caracteristicilor (ex.: prada este asociata numai clasei Carnivore)
  - — Dezavantaje:
    - Necesita chiar mai multe clase si generalizari
    - In general, mostenirea multipla poate implica complexitate suplimentara (conflicte de nume, diamond inheritance)



www.lloseng.com

Seturi de generalizare multiple

- **O solutie superioara (bazata pe sablonul de design Player-Role)**



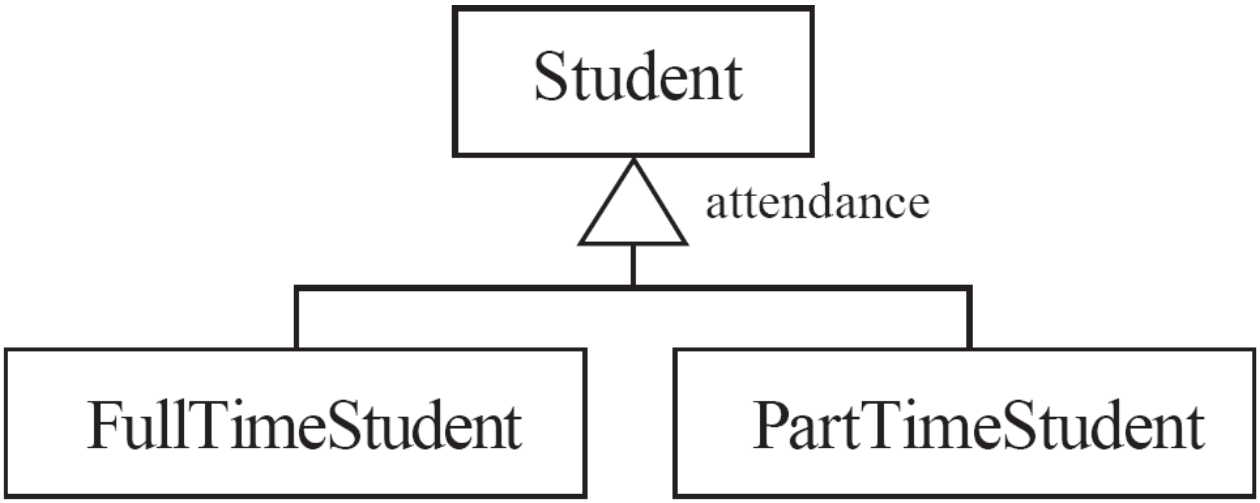
In UML numele de clase abstracte si numele de metode abstracte sunt redate in *italics*.

• Modelul sablonului de design Player-Role consta dintr-o clasa Player asociata cu o clasa abstracta Role; clasa Role este superclasa unui set de clase ce modeleaza roluri concrete posibile.
  • Un *rol* este un ansamblu particular de caracteristici asociate cu un obiect intr-un context particular.
  • Un obiect poate *juca* roluri diferite in contexte diferite.
• Desigur, in exemplul considerat este improbabil ca rolurile (AquaticAnimal, LandAnimal) sa se schimbe; in general insa, rolurile se pot modifica in functie de context.

© Lethbridge/Laganière 2005

Clasa unui obiect nu trebuie sa se modifice

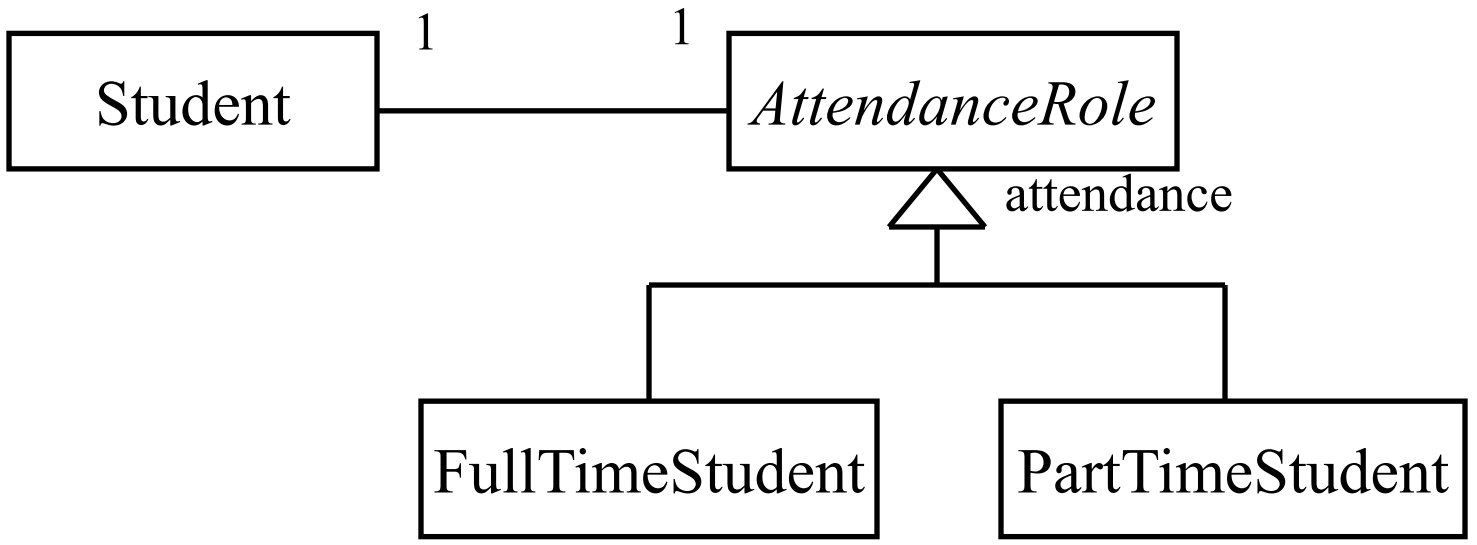
- Niciodata un obiect nu ar trebui sa isi modifice clasa!
  - In perioada studiilor, statutul unui student (in ceea ce priveste frecventa cu care audiaza cursurile) se poate modifica din FullTimeStudent in PartTimeStudent si invers.
  - Nu este de dorit ca o asemenea schimbare de statut sa fie modelata prin distrugerea unei instante FullTimeStudent (PartTimeStudent) si crearea unei instante PartTimeStudent (FullTimeStudent). **Diagrama de clase data mai jos este un model nepotrivit!**



© Lethbridge/Laganière 2005

Clasa unui obiect nu trebuie sa se modifice

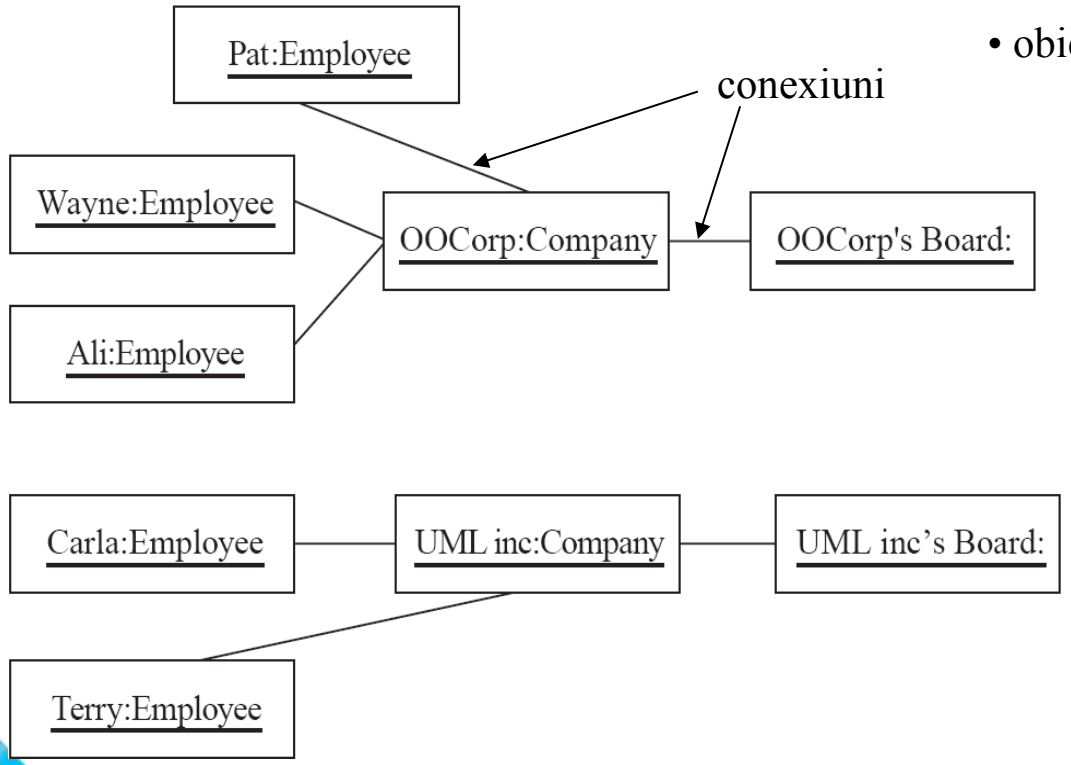
- **O solutie superioara (bazata pe sablonul Player-Role)**



5.5 Diagrame de obiecte

O **conexiune** (engl. *link*) este o instanta de asociere

- asa cum un obiect ca este instanta a unei clase



In UML un **obiect** este reprezentat sub forma unui dreptunghi (ca si o clasa, dar) cu numele obiectului subliniat. Se pot reprezenta:

- obiecte cu nume:
  - numeObiect : NumeClasa
- obiecte anonime:
  - : NumeClasa

Diagrame de obiecte pentru

- asocierea de tip 1-N intre clasele Employee si Company, si
- asocierea de tip 1-1 intre clasele Company si BoardOfDirectors.

© Lethbridge/Laganière 2005

Asocieri, generalizari si diagrame de obiecte

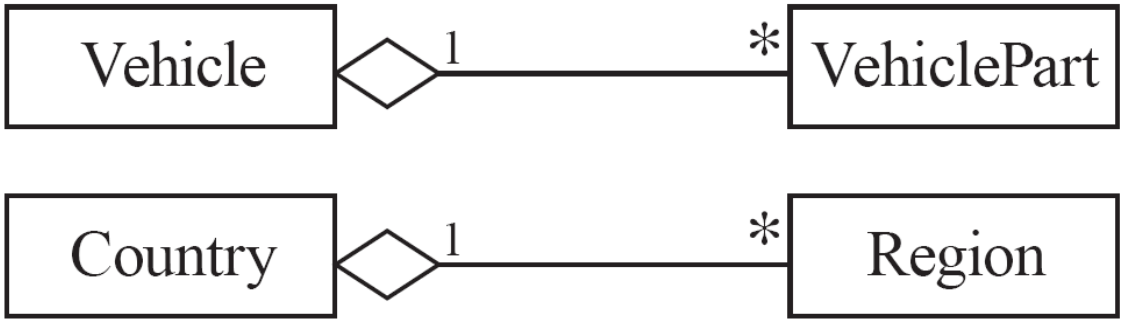
- O asociere descrie relatii care exista intre instante la executie.
  - —Intr-o diagrama de obiecte generata pe baza unei diagrame de clase, exista instante ale ambelor clase aflate intr-o relatie de asociere.

- Generalizarile descriu relatii intre clase (in diagrame de clase).
  - —Generalizarile nu apar deloc in diagramele de obiecte.
  - —O instanta a unei clase C trebuie sa fie in acelasi timp considerata a fi instanta a oricarei superclase a clasei C.

© Lethbridge/Laganière 2005

5.6 Agregare

- **Agregarea este un tip particular de asociere, reprezentand o relatie structurala intre parte si intreg (relatie ‘has-a’).**
  - —‘Intregul’ este adesea numit *ansamblu* sau *agregat*.
  - —Grafic, agregarea este reprezentata ca o linie terminata intr-un romb care indica spre ‘intreg’.



© Lethbridge/Laganière 2005

Cand se utilizeaza relatia de agregare

In general, o asociere poate fi tratata drept agregare daca:

- relatia este de tip parte-intreg, putand fi exprimata prin expresii de genul ‘is part of’ (este parte a) sau ‘is composed of’ (este compus din)
- cineva sau ceva care poseda sau controleaza ‘intregul’ poseda sau controleaza si ‘partile’.

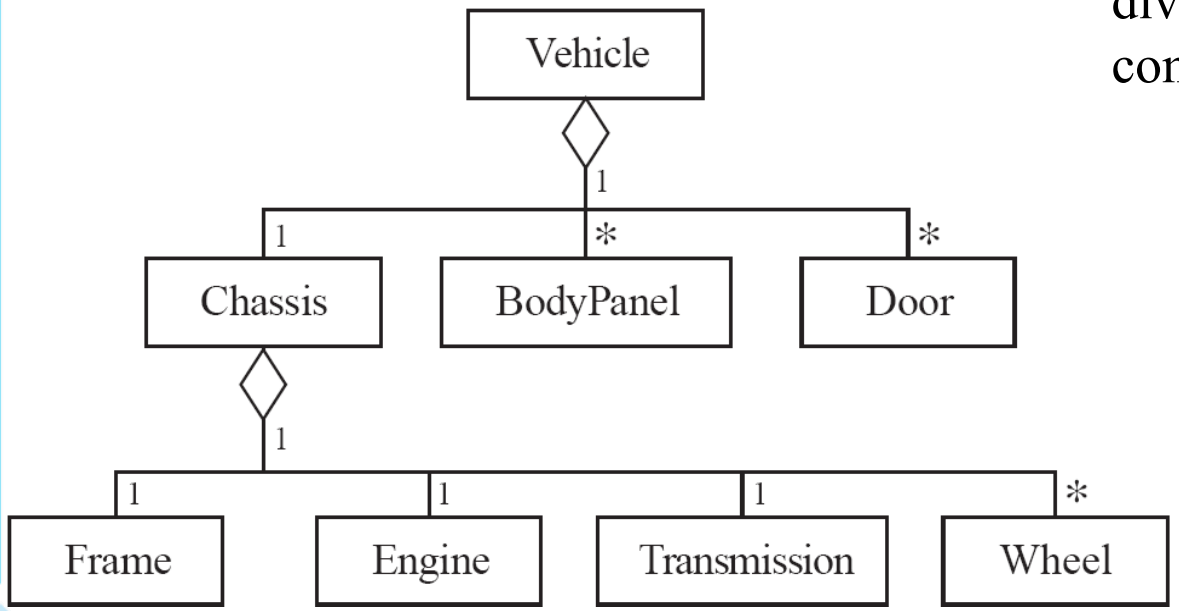
© Lethbridge/Laganière 2005

Compozitie

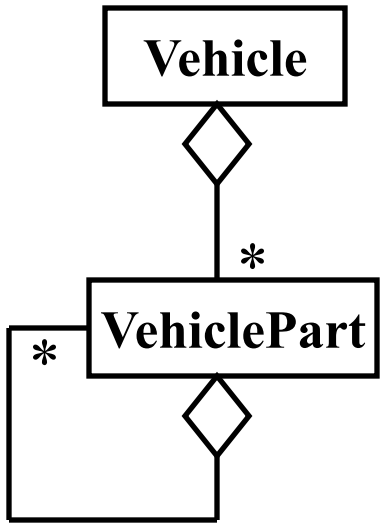
- ***Compozitia* este o forma de agregare in care**
  - —o ‘parte’ poate apartine unui singur ‘intreg’ (in agregarea simpla poate exista partajare)
  - —daca agregatul (intregul) este distrus atunci si partile se distrug.
- Grafic, compozitia este reprezentata ca si agregarea, dar rombul care indica spre ‘intreg’ este plin.

© Lethbridge/Laganière 2005

Ierarhie de agregare



Aceasta este o reprezentare mai flexibila. Poate fi adaptata pentru diverse tipuri de vehicule, care au diverse configuratii de componente (parti).



© Lethbridge/Laganière 2005

Propagare

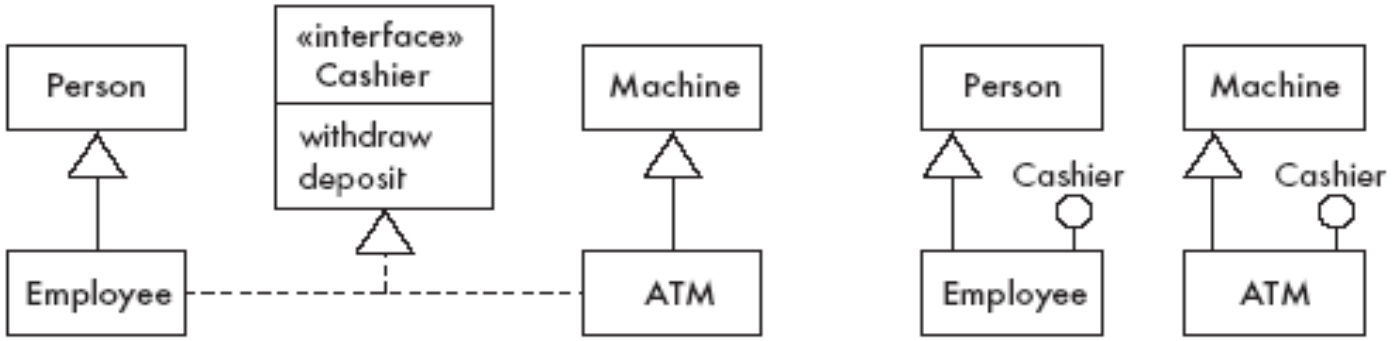
- Un mecanism prin care o operatie asupra agregatului implica operatii efectuate asupra partilor.
- Proprietatile partilor sunt adesea propagate inapoi la agregat.
  - —De exemplu, greutatea unei masini (agregatul) poate fi calculata prin insumarea greutatilor partilor componente.
- Se poate spune ca propagarea este pentru agregare ceea ce este mostenirea pentru generalizare.
  - —Principala diferenta este urmatoarea:
    - Mostenirea este un mecanism implicit
    - Propagarea trebuie sa fie programata atunci cand este necesar

© Lethbridge/Laganière 2005

Interfete

O interfata descrie o parte a comportamentului vizibil al unui set de obiecte.

- O *interfata* este similara cu o clasa, dar nu include variabile de instanta si implementari de metode.
- In UML, relatia intre o interfata si clasa (sau componenta) care o implementeaza (realizeaza) se numeste *realizare*.
  - —Grafic, relatia de realizare este reprezentata ca o linie intrerupta terminata cu un triunghi gol.
- In UML exista doua moduri de reprezentare a interfetelor, ambele ilustrate mai jos in specificarea interfetei Cashier (implementata de clasele Employee si ATM)



© Lethbridge/Laganière 2005

5.9 Construirea diagramelor de clasa

Se pot construi modele UML in diferite stadii ale procesului de dezvoltare

- **Model de explorare domeniu:**
  - Dezvoltat in faza de analiza domeniu
  - In mod normal contine unele clase, asocieri si atribute care nu fac parte din sistemul implementat (sunt necesare doar pentru intelegerea domeniului).
  - In acest stadiu inginerul software nu este constrans sa aplice tehnici de optimizare a modelelor, cum este evitarea mostenirii multiple.
- **Model domeniu sistem:**
  - Modeleaza aspecte ale domeniului care urmeaza sa fie reprezentate in sistem
  - Clasele din acest model devin module software reale iar instantele a multe dintre aceste clase sunt memorate persistent in baze de date.
  - Acest model poate include mai putin de jumatate din clasele modelului sistem (complet).
  - Ar trebui construit astfel incat sa fie cat mai independent de interafata utilizator si de clasele arhitecturale.
- **Model sistem (complet):**
  - Include modelul de domeniu sistem, impreuna cu
  - Clase utilizate in constructia interfetei utilizator, clase necesare arhitecturii sistem (clienti, servere, protocoale pentru arhitecturi Layers, etc.), precum si clase utilitare (care fac sistemul mai reutilizabil, mai usor de intretinut, etc.)

www.lloseng.com

O metodologie de analiza si proiectare

- Se identifica mai intai un set de **clase** candidate
- Se adauga **asocieri** si **atribute**
- Se identifica **generalizari**
- Se construieste lista **responsabilitatilor** pentru fiecare clasa
- Se decide asupra **operatiilor** specifice care sunt necesare in realizarea fiecarei responsabilitati
- Se **repeta** intregul proces pana cand se obtine un model satisfacator
  - —Se adauga sau se elimina clase, asocieri, atribute, generalizari, responsabilitati sau operatii
  - —Se identifica interfete, se aplica sabloane si principii de proiectare

Procesul de dezvoltare

—*nu trebuie sa fie dezorganizat,*

—*dar nici rigid.*

www.lloseng.com

Identificarea claselor

- In dezvoltarea unui model de domeniu inginerul software tinde sa *descopere* clase (pe baza descrierii cerintelor, a notelor de interviu sau pe baza rezultatelor sesiunilor de brainstorming)
  - —Uneori poate fi necesar sa se inventeze anumite elemente ale modelului de domeniu; de exemplu, poate fi necesar sa se inventeze o superclasa (al carei nume nu rezulta din analiza domeniului) pentru a se obtine o ierarhie de generalizare.
- Atunci cand se lucreaza la construirea interfetei utilizator sau la construirea arhitecturii este adesea necesar sa se *inventeze* clase
  - —Asemenea clase sunt necesare pentru rezolvarea anumitor probleme de proiectare particulare
- *Reutilizarea* trebuie sa reprezinte o preocupare continua; se pot
  - —Reutiliza cadre de aplicatie (engl. *frameworks*)
  - —Extinde sisteme existente
  - —Studia sisteme similare

© Lethbridge/Laganière 2005

O tehnica simpla pentru identificarea claselor in domeniul de aplicatie

- Se utilizeaza materiale sursa cum sunt descrierile de cerinte
- Se extrag substantivele si expresiile substantivale (engl. *noun phrases*).
- Se elimina substantivele care:
  - —Sunt redundante (doua nume pentru aceeasi clasa)
  - —Reprezinta instante
  - —Sunt vagi sau prea generale (de exemplu: ‘scopul utilizatorului’ sau ‘informatia reprezentata in aplicatie’)
  - —Nu sunt necesare in aplicatie
- Se separa clasele dintr-un model de domeniu care reprezinta *tipuri de utilizatori* sau alte categorii de *actori*
- Se face o distinctie clara intre clase si atribute de clase

© Lethbridge/Laganière 2005

O tehnica simpla pentru identificarea claselor in domeniul de aplicatie

Exemplu: Utilizand urmatoarea descriere de sistem informatic, se cere lista substantivelor si a expresiilor substantivale care ar putea reprezenta clase in modelul de domeniu sistem (vezi si descrierea in limba engleza, din carte, anexa C).

Liniile aeriene Ootumlia efectueaza curse aeriene din Java Valey, capitala Ootumlia. Sistemul informatic tine evidenta pasagerilor si a locurilor alocate acestora la diferitele curse, precum si a personalului care alcatuieste echipajul. In privinta echipajului, sistemul trebuie sa stie ce face fiecare si cine pe cine conduce sau supravegheaza. Liniile aeriene Ootumlia efectueaza zilnic curse aeriene numerotate pe baza unui program regulat. In viitor, liniile aeriene Ootumlia urmaresc sa se extinda, deci sistemul informatic trebuie sa fie flexibil; in particular, se doreste sa se adauge un serviciu de tip ‘frequent-flier plan’ (evidenta si facilitati pentru clientii care utilizeaza frecvent serviciile liniilor aeriene).

www.lloseng.com

O tehnica simpla pentru identificarea claselor in domeniul de aplicatie

Solutie:

- Substantivele care sunt puse in lista initiala de clase sunt: Cursa (sau cursa aeriana; engl. Flight), Pasager (engl. Passenger) si Angajat (engl. Employee).
- Nu se includ celelalte substantive sau expresii substantivale deoarece:
  - —‘Liniile aeriene Ootumlia’, ‘Java Valey’, ‘Ootumlia’ reprezinta instante
  - —‘Sistem informatic’ este numele sistemului insusi
  - —‘Loc’ este mai degraba un atribut
  - —‘Echipaj’ implica intregul echipaj; s-ar putea crea o clasa ‘MembruEchipaj’, dar clasa Angajat (Employee) ofera o mai buna flexibilitate.
  - —‘Program’ reprezinta informatie complexa care este descrisa de atributele si relatiile unor clase, cum este Cursa.
  - —‘Viitor’ – acest concept nu este parte a sistemului care urmeaza sa fie construit
  - —‘Frequent-flier plan’ – acest serviciu este planificat a fi dezvoltat intr-o versiune ulterioara a sistemului

© Lethbridge/Laganière 2005

Identificare asocieri si atribute

- Se porneste de la clasele care sunt considerate a fi cele mai importante (centrale) pentru aplicatie
- Pentru fiecare din aceste clase se identifica atributele si asocierile cu alte clase.
- Se continua progresiv spre clase considerate a fi mai putin importante.
- Se evita orice complexitate inutila.
  - —Un sistem este mai simplu daca manipuleaza mai putina informatie.
  - —Nu se vor adauga decat atribute sau asocieri care sunt in mod cert relevante pentru aplicatie (sunt necesare pentru implementarea unor cerinte).

© Lethbridge/Laganière 2005

Identificare asocieri

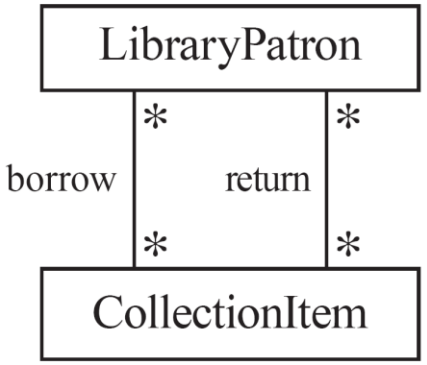
- O relatie de asociere exista daca o clasa
  - *poseda*
  - *controleaza*
  - *este connectata la*
  - *este in relatie cu*
  - *este parte a*
  - *are drept parte*
  - *este un membru al*, sau
  - *are drept membru*

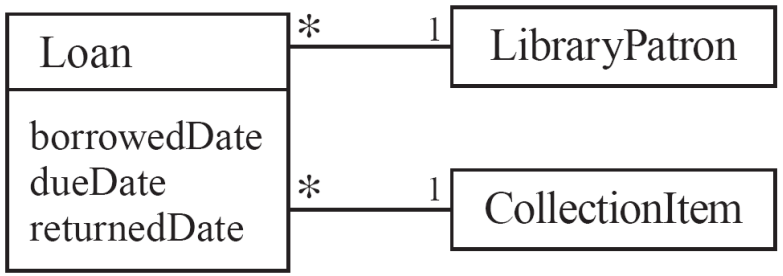
  o alta clasa din model (asemenea informatie poate fi adesea extrasa din documentele de definire sau de specificare a cerintelor)
- Se specifica multiplicitatea la ambele capete
- Se utilizeaza nume sugestive pentru asociere sau roluri

© Lethbridge/Laganière 2005

Actiuni si asocieri

- O eroare comuna este sa se reprezinte *actiuni* (sau cazuri de utilizare) ca fiind asocieri





Model eronat: contine asocieri care reprezinta, de fapt, actiuni.

Solutie superioara: operatia de imprumut (engl. *borrow*) creeaza o instanta Loan, iar operatia de returnare (engl. *return*) modifica valoarea atributului returnDate

© Lethbridge/Laganière 2005

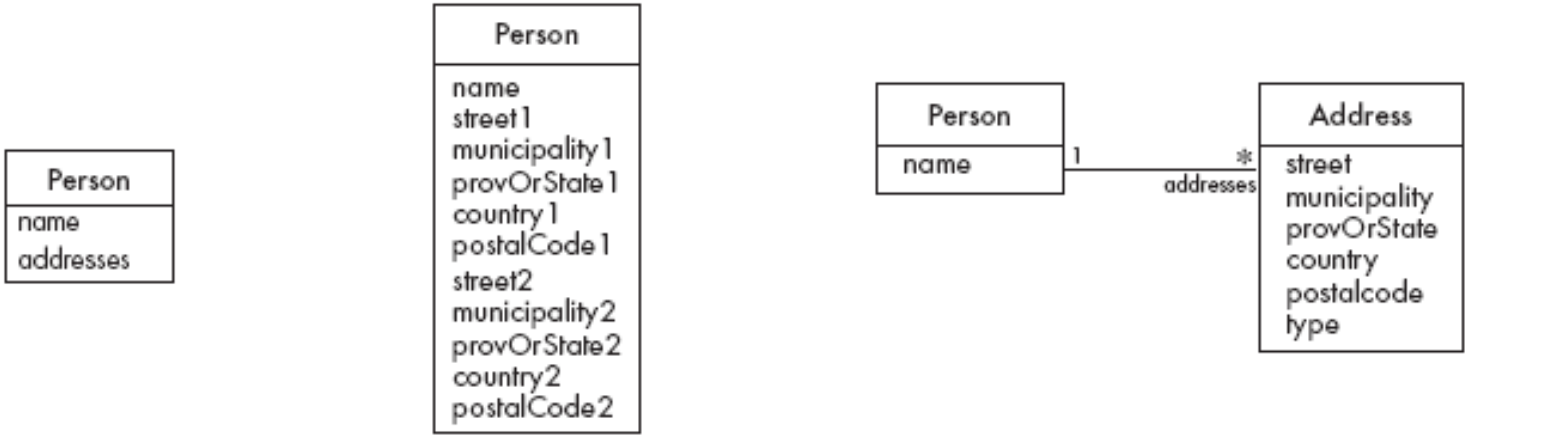
Identificare atribute

- Se identifica informatia care trebuie pastrata in fiecare clasa.
- Anumite substantive care nu au fost selectate initial spre a deveni clase in modelul de domeniu sistem pot deveni atribute.
- In general, un atribut contine o valoare simpla
    - un string
    - un numar

© Lethbridge/Laganière 2005

Identificare atribute

- Se va evita duplicarea atributelor
- Daca o parte dintre atributele unei clase formeaza un grup coerent, atunci se poate crea o clasa distincta care sa contina respectivele atribute (vezi partea dreapta a figurii de mai jos)



Bad, due to a plural attribute

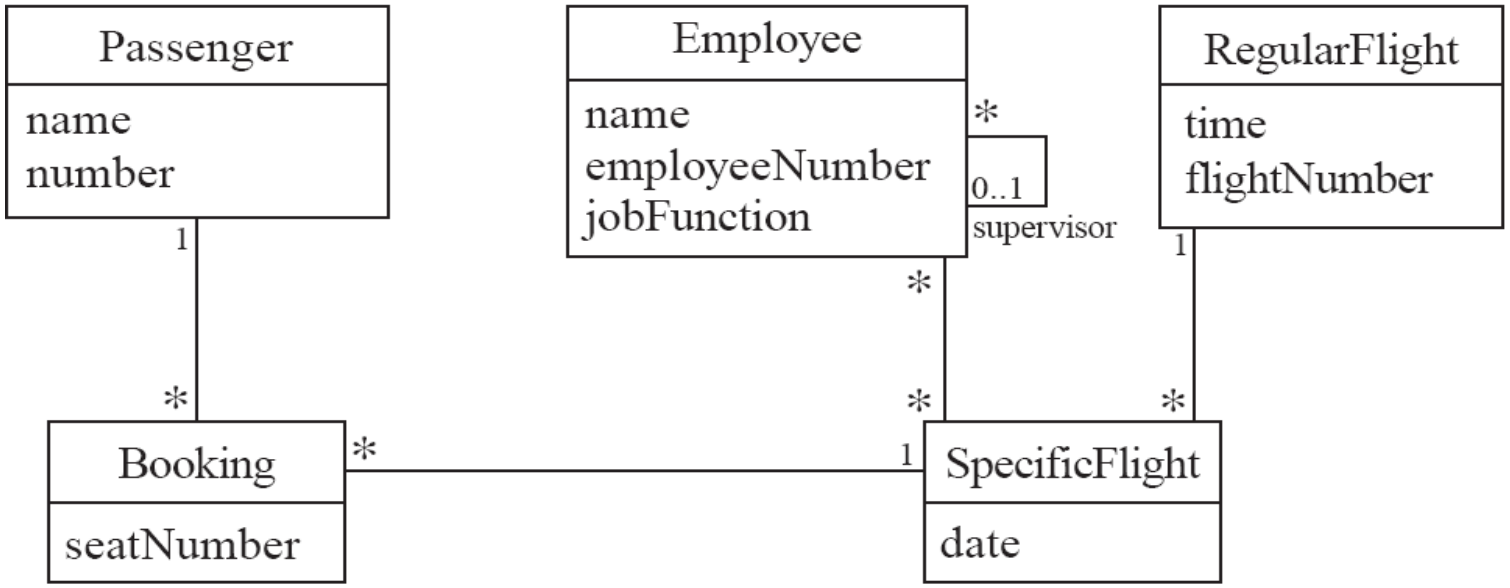
Bad, due to too many attributes, and the inability to add more addresses

Good solution. The type indicates whether it is a home address, business address etc.

© Lethbridge/Laganière 2005

Exemplu: atribute si asocieri in sistemul de rezervari pentru curse aeriene

• Clasa centrala de la care se porneste in identificarea asocierilor si atributelor este Flight (Cursa). Pentru evitarea redundantelor aceasta este impartita in: RegularFlight (cursa regulata) si SpecificFlight (cursa specifica, sau zbor).
• In continuare se studiaza modul in care se fac rezervarile de locuri pentru pasageri (se introduce clasa Booking).
• Expresia ‘cine pe cine conduce’ implica existenta unei asocieri reflexive pentru clasa Employee (se utilizeaza numele de rol ‘supervisor’).

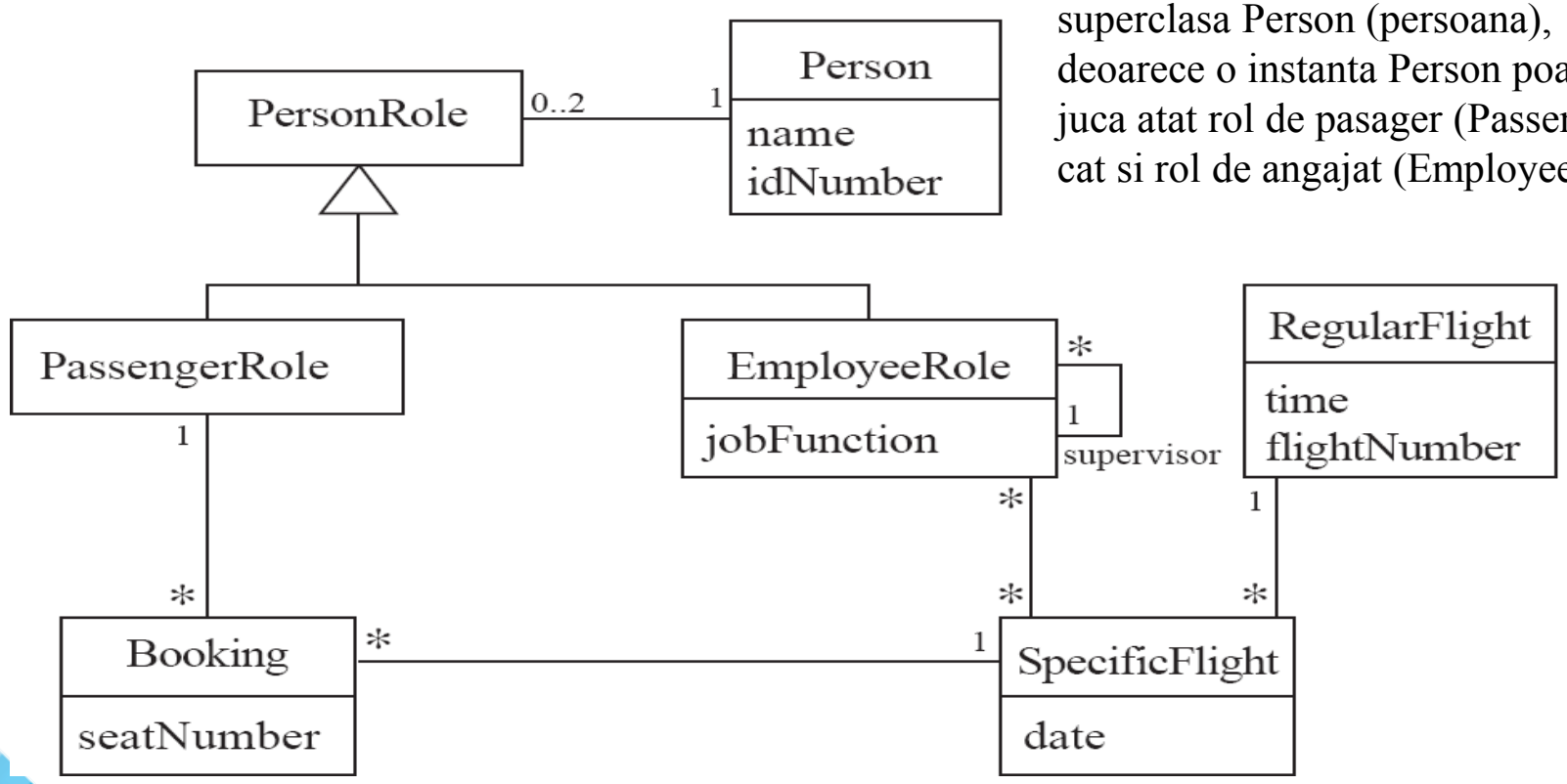


Identificare generalizari si interfete

- Exista doua moduri de a identifica generalizari:
  - —bottom-up
    - Se grupeaza impreuna doua sau mai multe clase care partajeaza comportament (atribute, asocieri sau operatii) prin crearea unei noi superclase (care reprezinta comportamentul comun)
  - —top-down
    - Se identifica mai intai clasele mai generale, care sunt apoi specializate daca este necesar
- In locul unei superclase se poate crea o *interfata* daca
  - —Clasele sunt foarte diferite, exceptand cateva operatii comune
  - —Una sau mai multe dintre clase au deja superclase
  - —Se doreste limitarea operatiilor care pot fi efectuate asupra unei variabile doar la cele specificate de interfata

© Lethbridge/Laganière 2005

Exemplu: generalizare



• Clasele Passenger si Employee partajeaza atributele pentru nume si numar de identificare.
• Nu este suficient sa se creeze o superclasa Person (persoana), deoarece o instanta Person poate juca atat rol de pasager (Passenger) cat si rol de angajat (Employee).

© Lethbridge/Laganière 2005

Alocarea de responsabilitati la clase

- Principalele responsabilitati sunt date de cerintele functionale.
- Fiecare responsabilitate trebuie sa fie atribuita unei clase (in general si alte clase vor colabora pentru realizarea respectivei sarcini de calcul).
  - —Daca o clasa are prea multe responsabilitati atunci poate fi impartita in clase mai mici
  - —Daca o clasa nu are nici o responsabilitate atunci, probabil, este *inutila*
  - —Cand o responsabilitate nu poate fi atribuita nici uneia dintre clasele existente atunci trebuie sa se creeze o *noua clasa*
- Pentru determinarea responsabilitatilor
  - —Se analizeaza cazurile de utilizare
  - —Se cauta verbe sau substantive care descriu *actiuni* in descrierile sistem

© Lethbridge/Laganière 2005

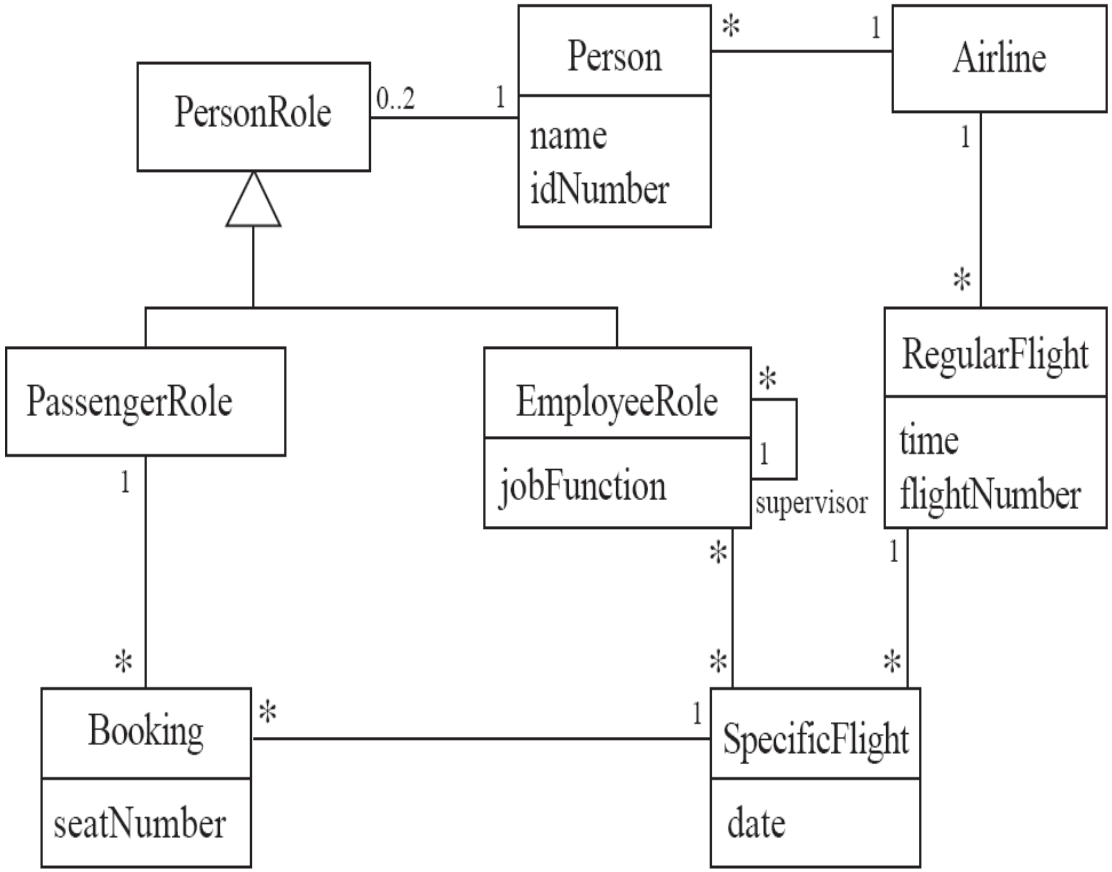
Categorii de responsabilitati

- Interogare sau modificare valori de atribute
- Creare si initializare obiecte (instante) noi
- Incarcare din baze de date sau salvare in baze de date
- Distrugere obiecte (instante)
- Adaugare sau stergere conexiuni intre obiecte
- Copiere, conversie, transformare, transmitere sau formatare date
- Calcularea unor rezultate numerice
- Navigare si cautare
- Alte sarcini specifice

© Lethbridge/Laganière 2005

Exemple de responsabilitati

- Crearea instanta RegularFlight
- Modificare atribute pentru o instanta RegularFlight
- Crearea instanta SpecificFlight
- Rezervare loc pentru un pasager



© Lethbridge/Laganière 2005

Prototipizare diagrame de clase prin analiza de tip Clasa-Responsabilitate-Colaborare (CRC)

- La identificarea unei clase se scrie numele acesteia pe un cartonas
- Pe masura ce sunt identificate, atributele si responsabilitatile se ataseaza claselor
  - —Daca responsabilitatile nu incap pe cartonas
    - Asta sugereaza ca respectiva clasa ar trebui impartita in clase mai mici.
- Cartonasele se dispun pe o tabla de scris intr-o diagrama de clase.
  - —Se deseneaza asocieri si generalizari pentru reprezentarea relatiilor dintre clase.

© Lethbridge/Laganière 2005

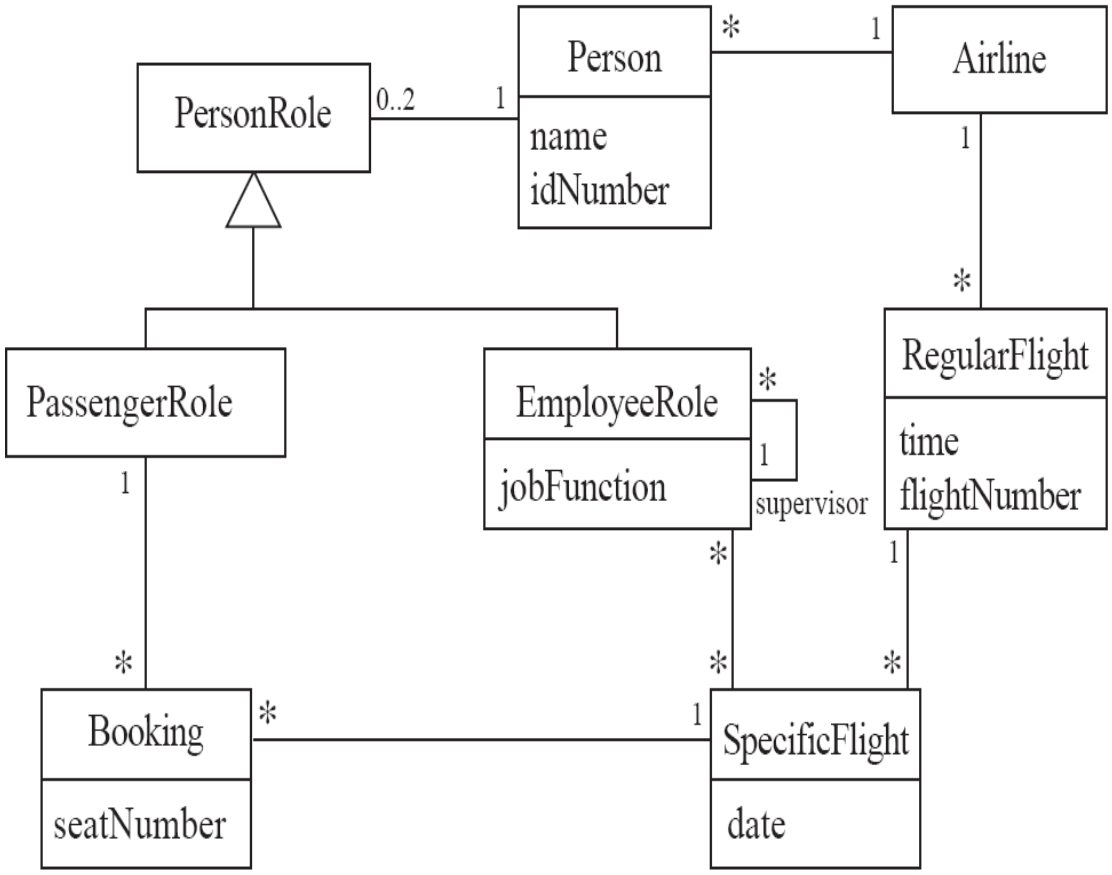
Identificare operatii

Fiecare responsabilitate este implementata prin una sau mai multe operatii

- Operatia principala care implementeaza responsabilitatea este in mod normal declarata **public**, devenind parte a interfetei intregului sistem.
- Celelalte metode care colaboreaza la implementarea unei responsabilitatii trebuie sa fie (cat mai) private.

© Lethbridge/Laganière 2005

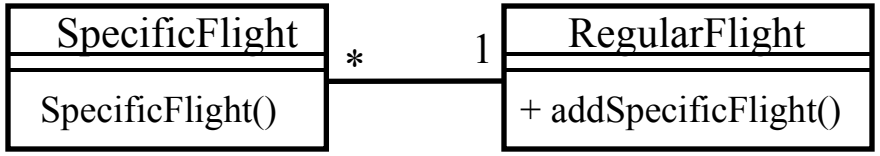
Exemple de operatii



© Lethbridge/Laganière 2005

Operatii pentru crearea unui obiect si conectarea sa la un obiect existent (conexiune bidirectionala)

Exemplu: Crearea unui obiect SpecificFlight si conectarea sa la un obiect RegularFlight. Aceasta responsabilitate poate fi realizata cu ajutorul urmatoarelor operatii (implementarea Java este data in sectiunea 5.10):

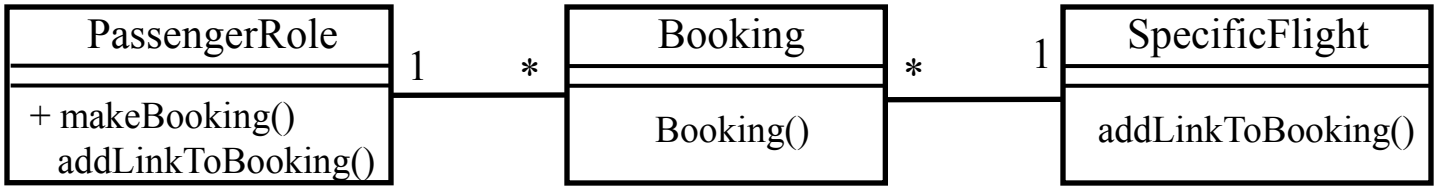


1. (public) Instanta RegularFlight
   - Apeleaza constructorul SpecificFlight; apoi, efectueaza o conexiune unidirectionala catre noua instanta SpecificFlight (de fapt, fiecare instanta RegularFlight pastreaza o lista de obiecte SpecificFlight; lista este necesara pentru implementarea relatiei de tip 1-N).
2. (non-public) Constructorul clasei SpecificFlight (pe langa alte actiuni)
   - Realizeaza o conexiune unidirectionala inapoi catre instanta RegularFlight.

© Lethbridge/Laganière 2005

Operatii pentru crearea unei instante de clasa de asociere

Exemplu: crearea unui obiect Booking, care realizeaza legatura intre un obiect SpecificFlight si un obiect PassengerRole (existente dinainte). Aceasta responsabilitate poate fi realizata in patru pasi (se doreste o asociere bidirectionala):



1. (public) Instanta PassengerRole
   - Apeleaza constructorul clasei Booking (operatia 2)
2. (non-public) Constructorul clasei Booking (pe langa alte actiuni)
   - Face o conexiune unidirectionala catre instanta PassengerRole
   - Face o conexiune unidirectionala catre instanta SpecificFlight
   - Apeleaza operatiile 3 and 4.
3. (non-public) Instanta SpecificFlight
   - Face o conexiune unidirectionala catre instanta Booking
4. (non-public) Instanta PassengerRole
   - Face o conexiune unidirectionala catre instanta Booking.

www.lloseng.com

5.10 Implementarea diagramelor de clase in Java

- Atributele sunt implementate ca variabile de instanta
- Generalizarile sunt implementate utilizand cuvantul cheie **extends**
- Realizarile de interfete sunt implementate utilizand cuvantul cheie **implements**
- Asocierile sunt implementate utilizand variabile de instanta
  - Intr-o asociere bidirectionala fiecare clasa are o variabila de instanta de tipul clasei de la celalalt capat al asocierii
  - Pentru o asociere unidirectionala in care multiplicitatea la celalalt capat este ‘unu’ sau ‘optional’
    - —Se declara o variabila de tipul clasei respective (o referinta)
  - Pentru o asociere de tip 1-N in care multiplicitatea la celalalt capat este ‘mai multi’ sau ‘mai multe’:
    - Se utilizeaza o clasa care implementeaza o colectie, cum este ArrayList sau LinkedList

© Lethbridge/Laganière 2005

Exemplu: SpecificFlight

```
class SpecificFlight
{
    private Calendar date;
    private RegularFlight regularFlight;
    ...
   // Constructor that should only be called from
   // addSpecificFlight
   SpecificFlight( Calendar aDate, RegularFlight aRegularFlight)
   {
      date = aDate;
      regularFlight = aRegularFlight;
    }
}
```


© Lethbridge/Laganière 2005

Exemplu: RegularFlight

```
class RegularFlight
{
    private List specificFlights;
    ...
    // Method that has primary responsibility
    public void addSpecificFlight(Calendar aDate)
    {
        SpecificFlight newSpecificFlight;
        newSpecificFlight = new SpecificFlight(aDate, this);
        specificFlights.add(newSpecificFlight);
    }
    ...
}
```


© Lethbridge/Laganière 2005

Referinte suplimentare

[BRJ99] G. Booch, J. Rumbaugh and I. Jacobson. *The Unified Modeling Language user guide*. Addison-Wesley, 1999.

[JBR99] I. Jacobson, G. Booch and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, 1999.

[RJB99] J. Rumbaugh, I. Jacobson and G. Booch. *The Unified Modeling Language reference manual*. Addison-Wesley, 1999.

© Lethbridge/Laganière 2005