
# Programare orientată pe obiecte

1. Dezvoltarea aplicațiilor OO
2. Diagrame UML de clase și obiecte


# Proiectarea orientată pe obiecte

1. Descoperim clasele
2. Determinăm responsabilităţile fiecărei clase
3. Descriem relaţiile dintre clase


# Unified Modeling Language (UML)

- UML este notaţia internaţională standard pentru analiza şi proiectarea orientată pe obiecte.
- UML 2.0 defineşte treisprezece tipuri de diagrame, împărţite în două categorii:
  - **Diagrame de structură**: şase tipuri de diagrame reprezintă structura statică a aplicaţiei:
    - *diagrama de clase, diagrame de obiecte*, diagrama de componente, diagrama de structură compozită, diagrama de pachete şi diagrama de desfăşurare sistematică.
  - **Diagrame de comportament**: trei diagrame ce reprezintă tipuri generale de comportament:
    - Diagrama de activităţi, diagrama de interacţiune, diagrama cazurilor de utilizare (use case), diagrama de secvenţe, diagrama de stare, diagrama de comunicare, diagrama de timp.

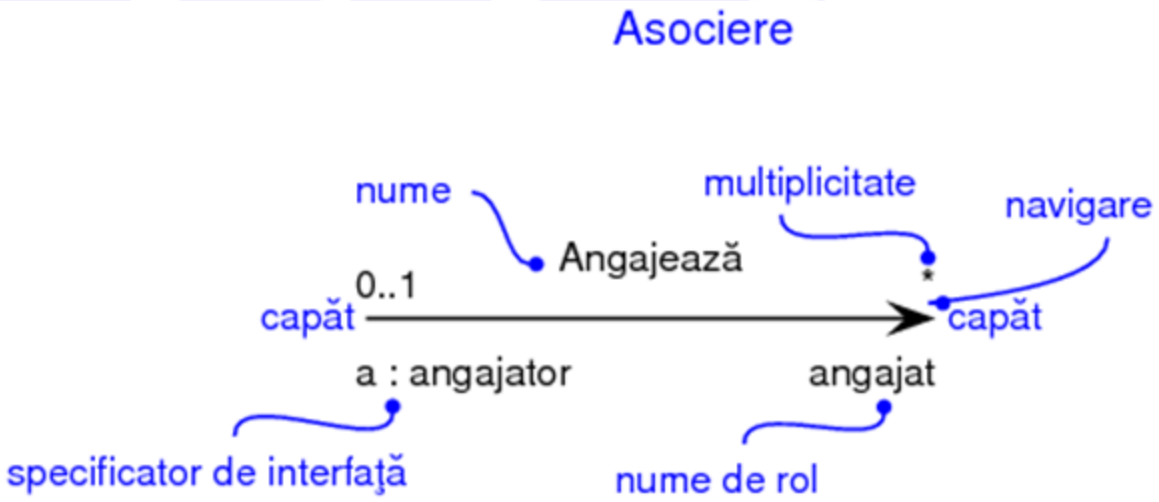

# Descoperirea claselor

- O clasă reprezintă un concept util
  - Entităţi concrete: conturi bancare, elipse, produse
  - Concepte abstracte: fluxuri (streams) şi ferestre
- Găsim *clasele* căutând *substantive* în descrierea sarcinii
- Definim comportamentul fiecărei clase
- Găsim *metodele* căutând *verbe* în descrierea sarcinii

# Relaţii între entităţile reprezentate

Tipuri de relaţii:

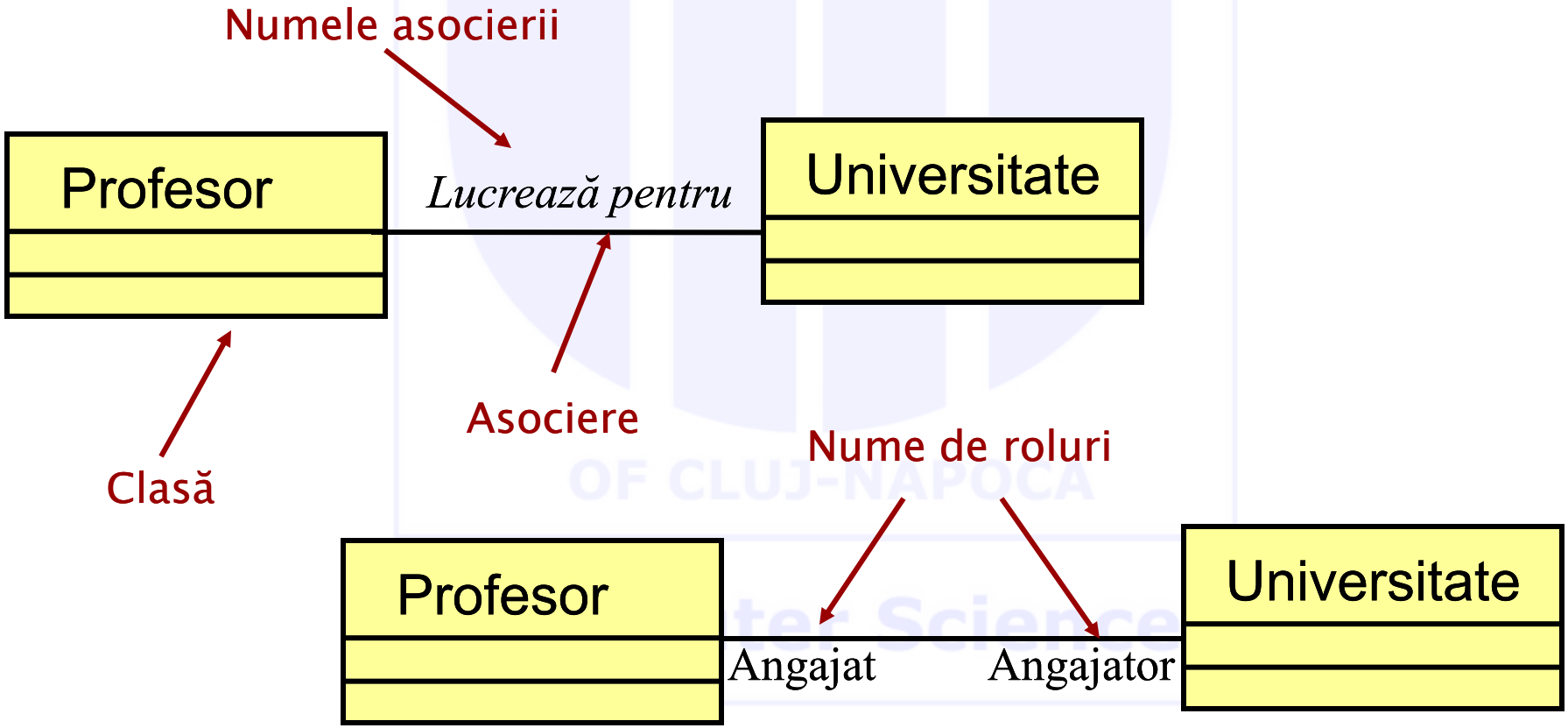
- Asociere
  - Agregare
  - Compoziţie




- Dependenţă
- Generalizare
- Realizare

# Relaţii: Asociere

- Modelează o conexiune semantică între clase

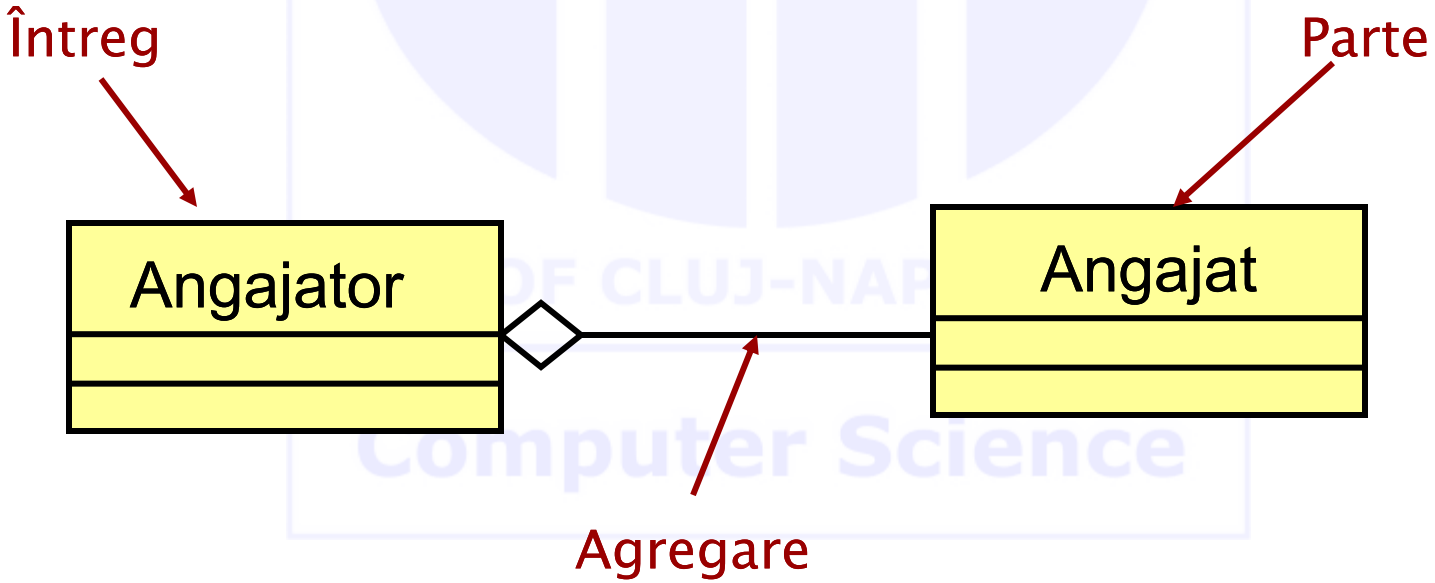



# Folosirea asocierilor

- Trei scopuri generale. Pentru a reprezenta:
  - O situaţie în care un obiect de o clasă *foloseşte* serviciile unui alt obiect, sau ele îşi folosesc reciproc serviciile – adică un obiect îi trimite mesaje celuilalt sau îşi trimit mesaje între ele.. (În primul caz, navigabilitatea poate fi unidirecţională; în cel de al doilea, ea trebuie să fie bidirecţională.)
  - *Agregarea* sau *compoziţia* – unde obiecte de o clasă sunt întregi compuşi din obiecte de cealaltă clasă ca părţi. În acest caz, o relaţie de tip foloseşte este implicit prezentă – întregul foloseşte părţile pentru a-şi îndeplini funcţia, iar părţile pot si ele avea nevoie să folosească întregul.
  - O situaţie în care obiectele sunt *înrudite*, chiar dacă nu schimbă mesaje. Aceasta se întâmplă de obicei când cel puţin unul dintre obiecte este folosit în esenţă la stocare de informaţie.

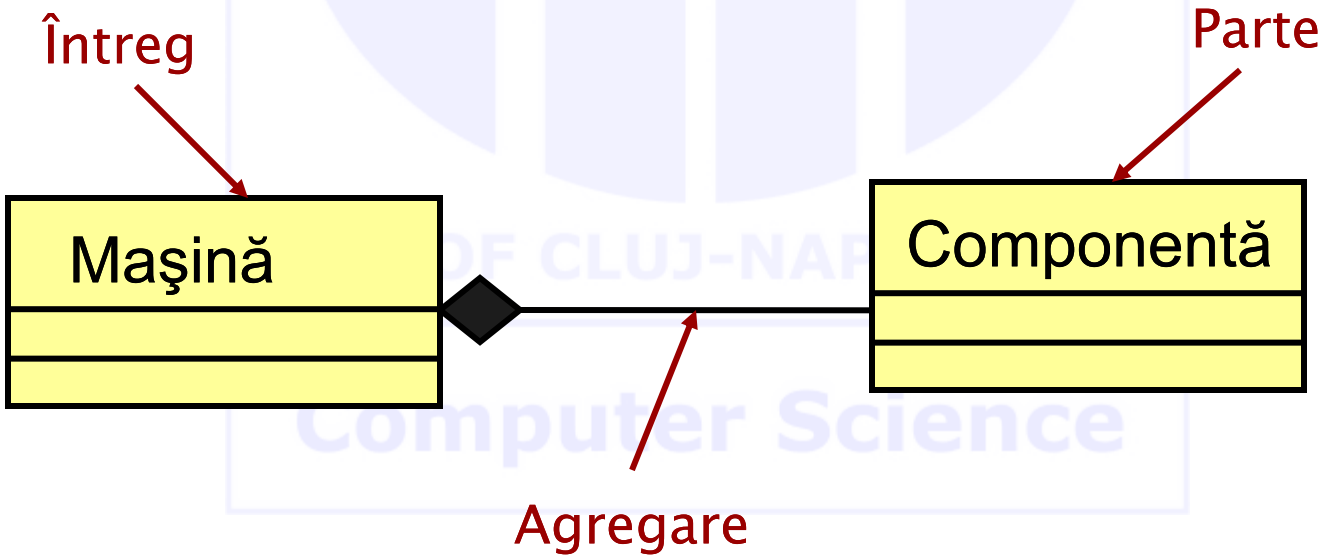
# Relaţii: Agregare

- O formă specială de asociere care modelează relaţia parte–întreg între un agregat (întregul) şi părţile sale



# Relaţii: compunere

- O formă de agregare cu posesiune *puternică* şi durate de viaţă care coincid
  - Părţile nu pot supravieţui întregului/agregatului



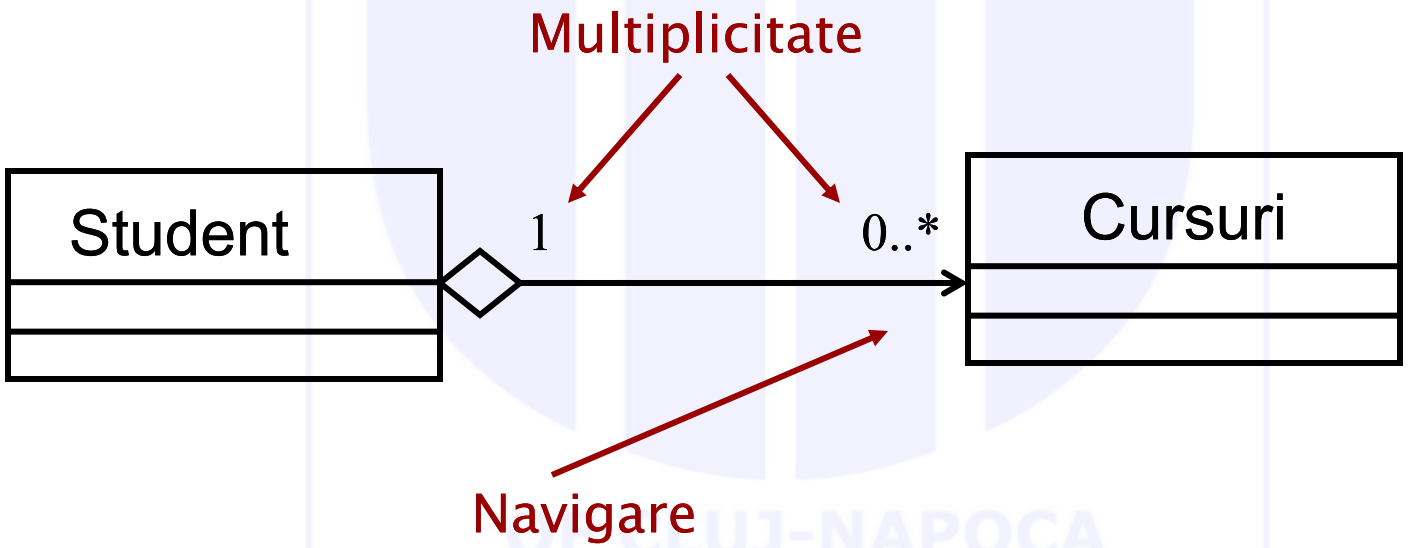

# Asociere: Multiplicitate şi navigare

- Multiplicitatea defineşte câte obiecte participă într-o relaţie
  - Numărul de instanţe ale unei clase în raport cu una instanţă a celeilalte clase
  - Specificat pentru fiecare capăt al asocierii
- Asocierile şi agregările sunt implicit bidirecţionale, dar adesea este de dorit să se restrângă navigarea la o singură direcţie
  - Dacă navigarea este restricţionată, se adaugă o săgeată pentru a indica direcţia de navigare

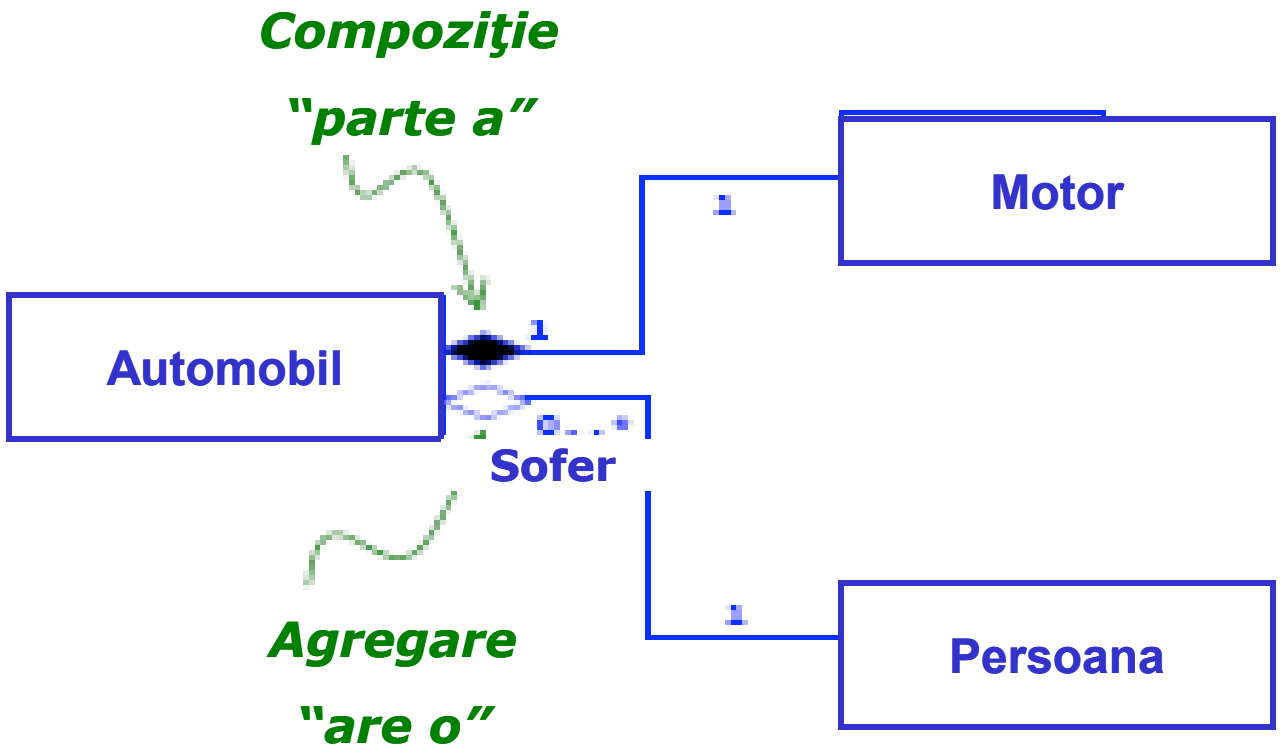

# Asociere: multiplicitate

- Nespecificată
- Exact una — 1
- Zero sau mai multe (multe, nelimitat) — 0..* / *
- Una sau mai multe — 1..*
- Zero sau una — 0..1
- Gama specificată — 2..4
- Game multiple, disjuncte — 2, 4..6

# Exemplu: multiplicitate şi navigare



# Exemple de asocieri




# Observaţii

- **Teste pentru relaţii parte-întreg adevărate**
  - **tranzitivitate: dacă "A este parte a lui B" & "B este parte a lui C" atunci "A este parte a lui C"**
    - "Unghia este parte a degetului, degetul este parte a mâinii; mâna este parte a extremităţii superioare a corpului"
  - **O problemă a unei părţi este o problemă a întregului**
    - O rană la unghie este o rană a corpului
  - "Poziţiile sunt parte a sistemului electric al automobilului. Un defect al poziţiilor este un defect al automobilului"
- **Este-parte-a *e diferit* de**
  - **esteConţinutÎn** : cămăşi, pantaloni,... --- valiza [observaţi ca testul de defectare nu ţine aici: pantalonii stricaţi nu înseamnă valiză stricată]
  - **esteLegatDE** : valiză --- persoană (care o duce)
  - **esteRamurăA** : artera iliacă,... --- aorta
  - **seAflaÎn** : casă... --- stradă


# Când să folosim agregarea

- **Ca regulă generală, se poate marca o asociere ca agregare, dacă sunt adevărate următoarele:**
  - Se poate spune că
    - Părţile 'sunt parte' a agregatului
    - sau agregatul 'este compus din' părţi
  - Când ceva deţine sau controlează agregatul, atunci acel ceva deţine sau controlează părţile


# Agregare şi compoziţie

## Asociere

**Obiectele ştiu unul despre altul astfel încât pot lucra împreună**

### Agregare

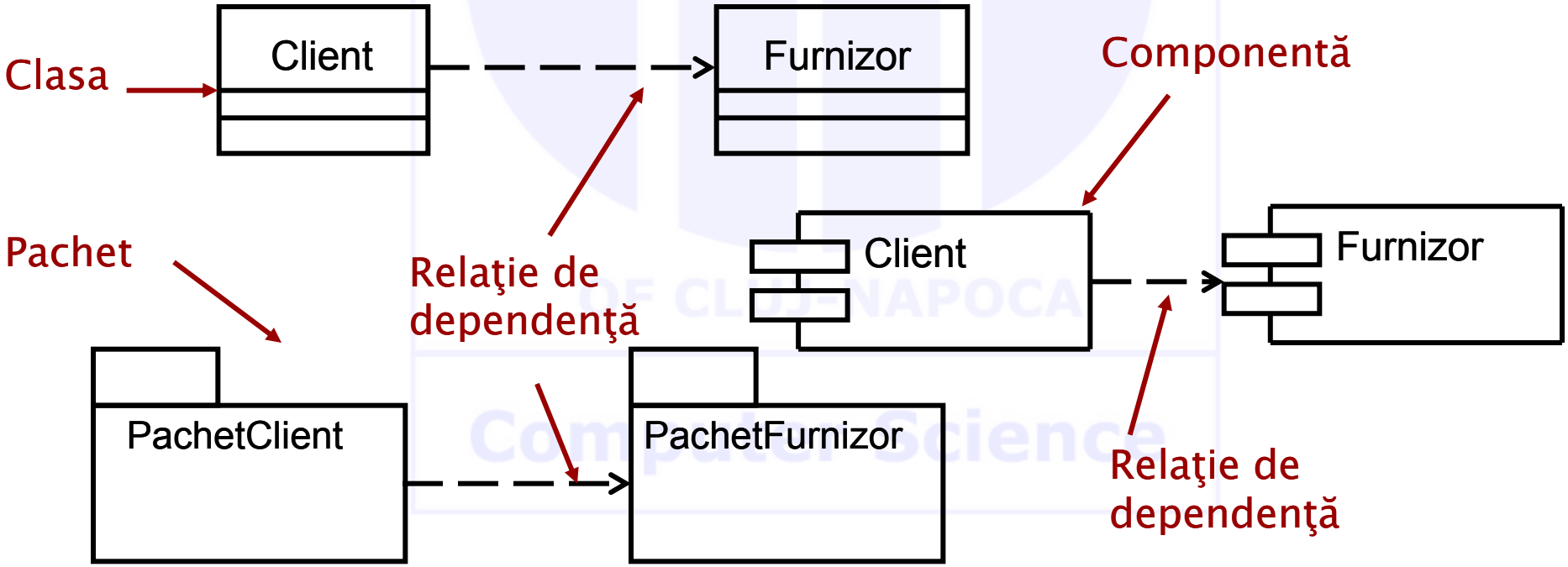
- **Protejează integritatea configuraţiei**
- **Funcţionează ca un singur tot**
- **Controlul se face printr-un obiect– propagarea este în jos**

#### Compoziţie

**Fiecare parte poate fi membru al unui singur obiect agregat**


# Relaţii: dependenţă

- O relaţie între două elemente ale modelului în care o schimbare în unul dintre elemente **poate** cauza o schimbare în celălalt
- Relaţie nestructurală, de tip "foloseşte"

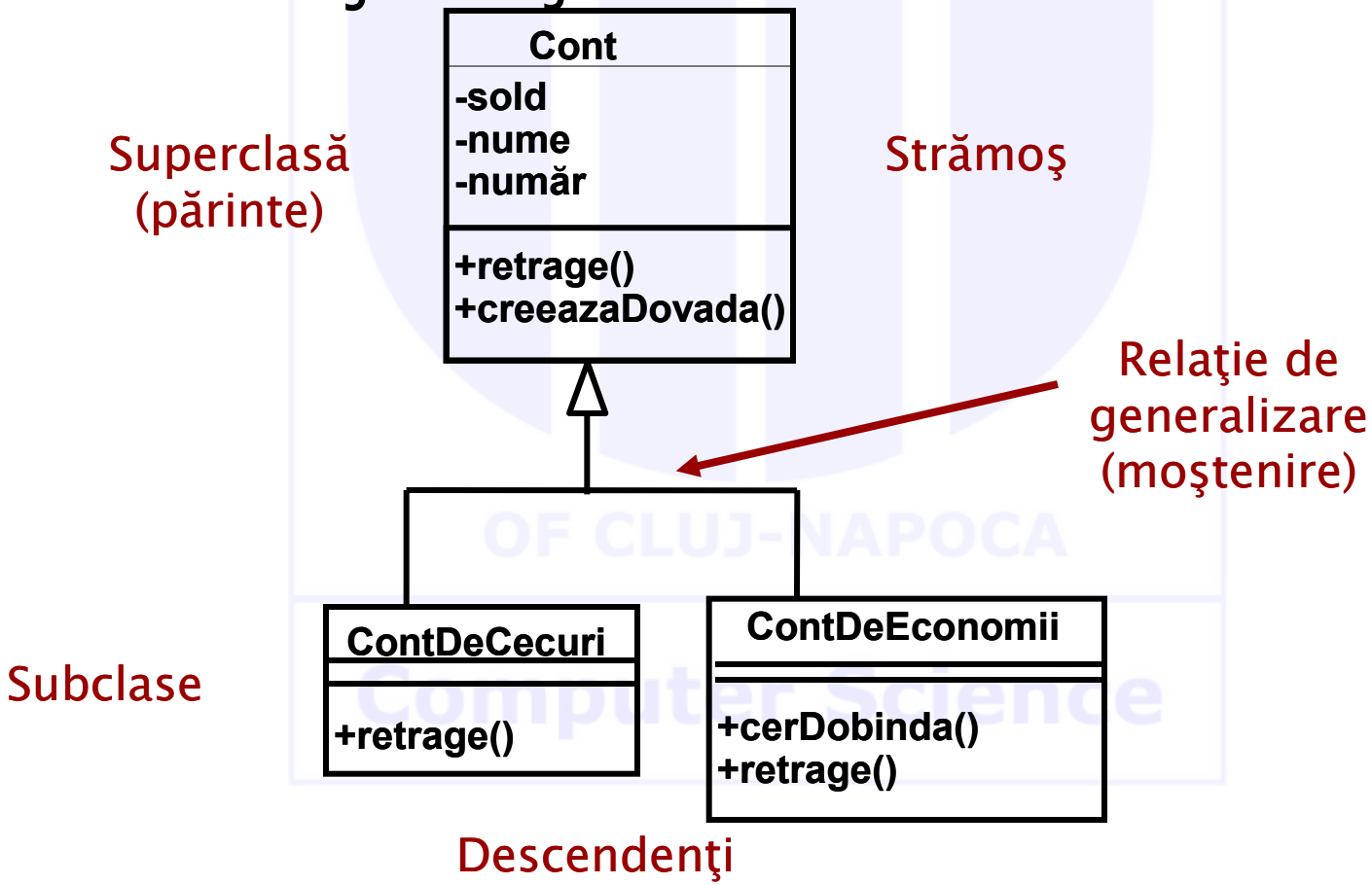



# Relaţii: generalizare

- O relaţie între clase în care o clasă partajează structura şi/sau comportamentul uneia sau mai multor clase
- Defineşte o ierarhie de abstracţiuni în care o subclasă moşteneşte de la una sau mai multe superclase
  - Moştenire simplă/de la o singură clasă
  - Moştenire multiplă
- Generalizarea este o relaţie de tipul "este-un-fel-de"

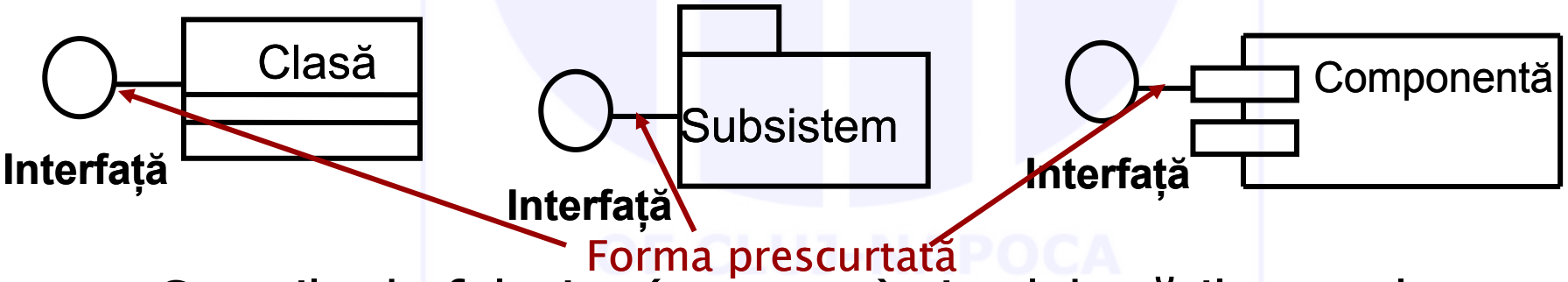
# Exemplu: Moştenirea simplă

- O clasă moşteneşte de la alta

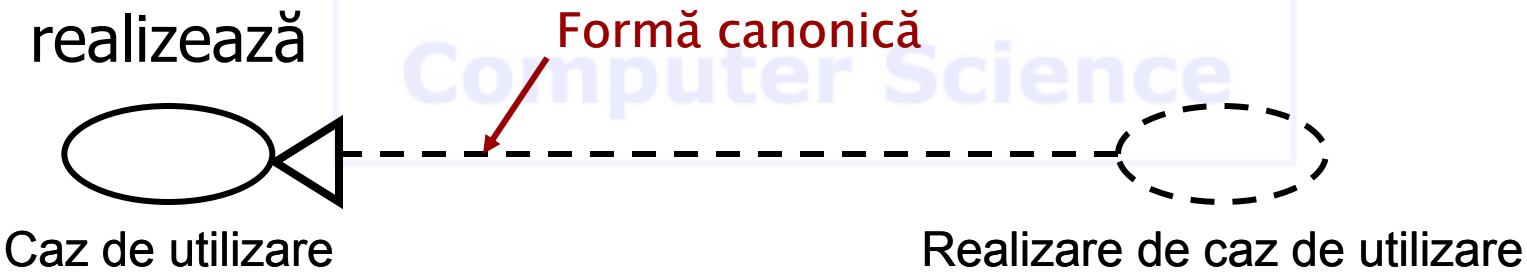


# Relaţii: realizare

- Un clasificator serveşte pe post de contract pe care celălalt este de acord să-l îndeplinească
- Regăsit între:
  - Interfeţele şi clasificatorii care le realizează

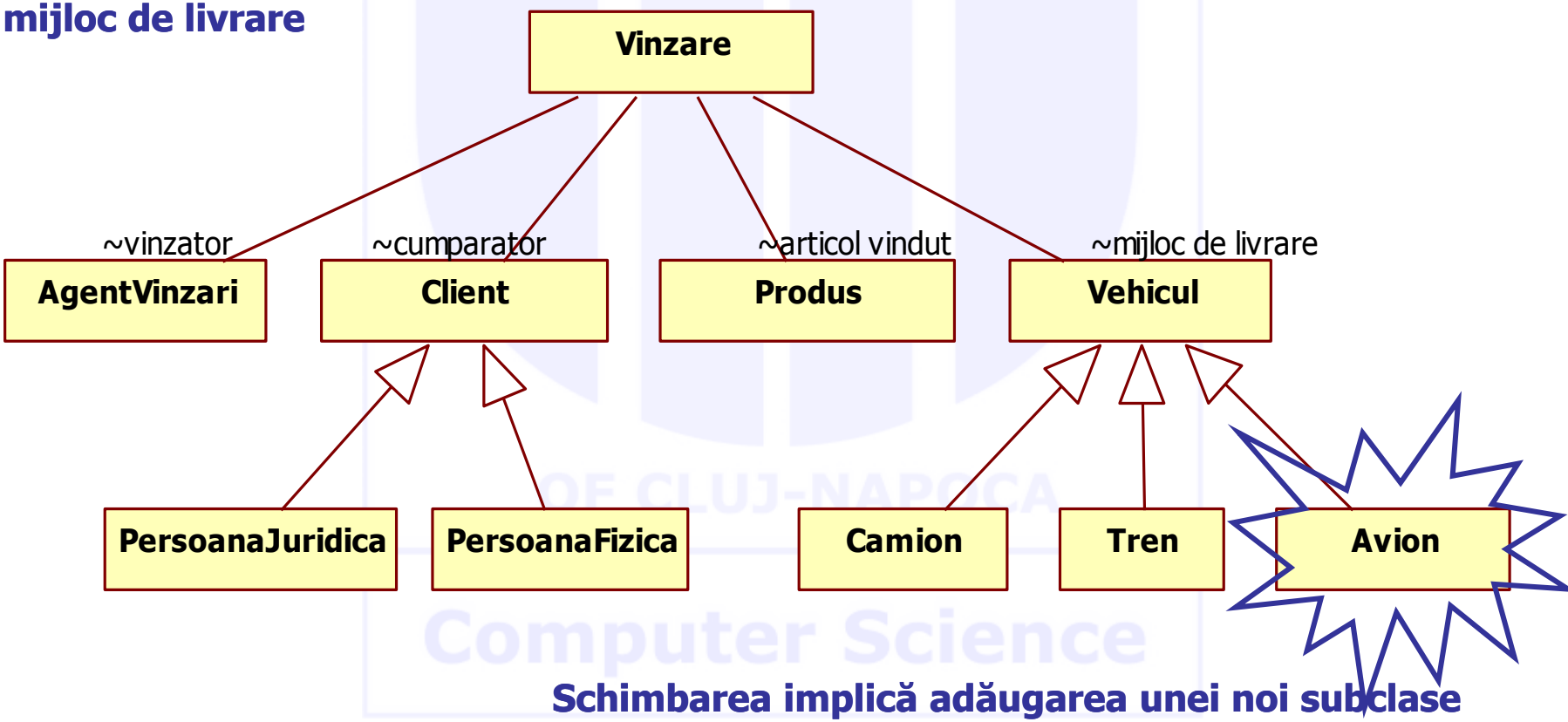



  - Cazurile de folosire (use case) şi colaborările care le realizează



# Efectul schimbării cerinţelor

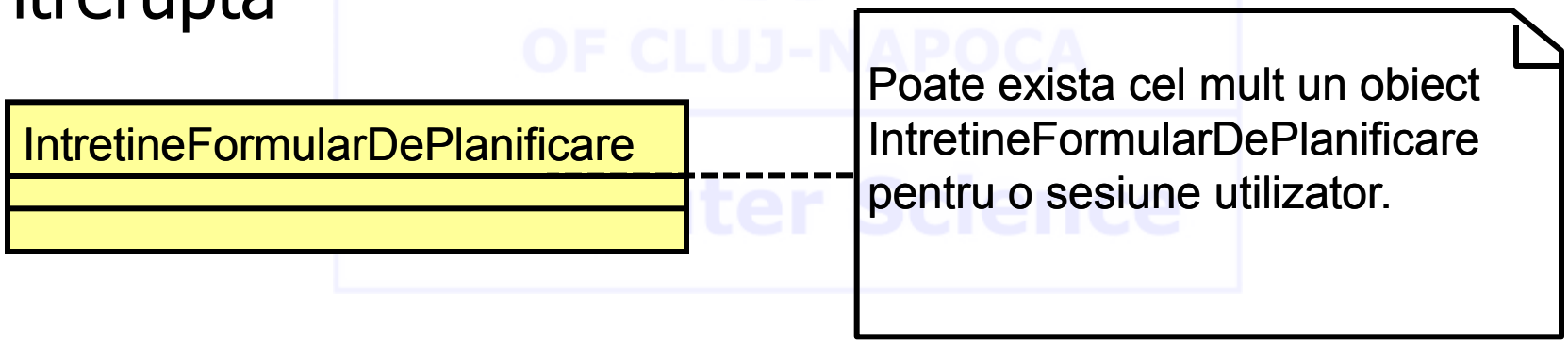
**Presupunând că e nevoie de un nou mijloc de livrare**




**Schimbarea implică adăugarea unei noi subclase**

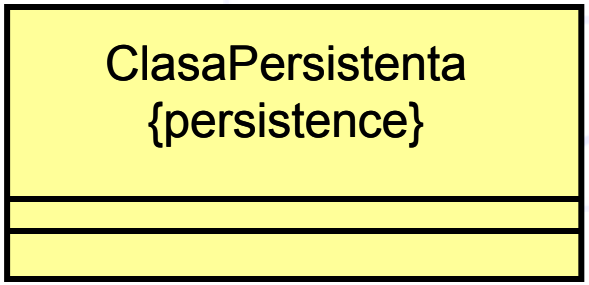
# Note (adnotări)

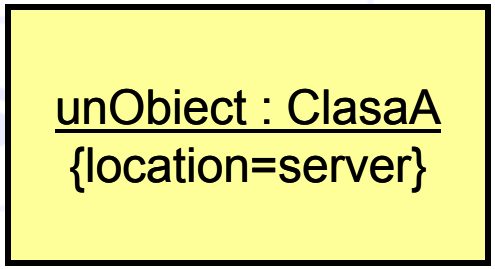
- Se poate adăuga o notă (adnotare) la orice element UML
- Notele se pot adăuga pentru a furniza informaţii suplimentare în diagramă
- Este un dreptunghi cu "ureche de câine"
- Nota poate fi ancorată la un element cu o linie întreruptă




# Valori etichetate

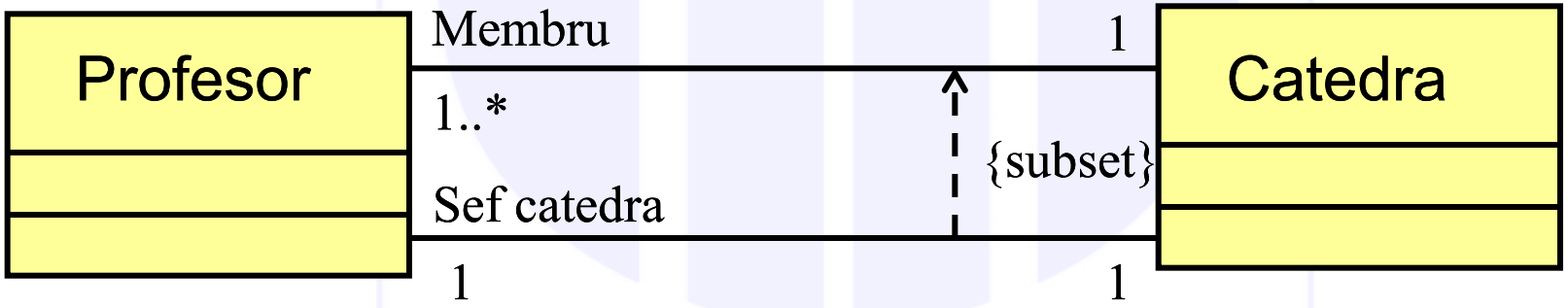
- Sunt extensii ale proprietăţilor sau ale anumitor atribute ale unui element UML
- Unele proprietăţi sunt definite de UML
  - Persistenţa
  - Localizarea (d.e., client, server)
- Proprietăţile pot fi create de modeluri UML în orice scop



# Constrângeri

- Suportă adăugarea de reguli noi sau modificarea regulilor existente




  - Această notație este folosită pentru a surprinde două relații între obiecte de tip Profesor şi obiecte de tip Catedra, unde una dintre relații este un subset al celeilalte.
  - Arată cum se pot croi diagramele UML pentru a modela corect relații  exacte


# Cartela CRC

- Cartela CRC: descrie o clasă, responsabilităţile şi colaboratorii săi
- Se foloseşte o cartelă pentru fiecare clasă
- Alegem clasa care trebuie să fie răspunzătoare de fiecare metodă (verb)
- Scriem responsabilitatea pe cartela clasei
- Indicăm ce alte clase sunt necesare pentru a îndeplini responsabilitatea (colaboratorii)

# Cartela CRC. Relaţii între clase

- Cartela CRC

| Numele clasei | |
|---|---|
| Responsabilităţi | Colaboratori |

- Relaţii între clase
  - Moştenire
  - Agregare
  - Dependenţă


# Moştenire

- Relaţie de tipul *este-o/un*
- Relaţie între o clasa mai generală (superclasă) şi una mai specializată (subclasă)
- Exemple:
  - Orice cont de economii este un cont bancar
  - Orice cerc este o elipsă (cu lăţime şi înălţime egale)
- Uneori se abuzează de această relaţie
  - Ar trebui să fie clasa **`Anvelopa`** o subclasă a lui **`Cerc`**?
  - Relaţia *are-o* ar fi mai potrivită aici


# Agregare

- Relaţie de tipul *are-o/un*
- Obiectele unei clase conţin referinţe la obiectele altei clase
- Folosesc variabile instanţă
  - O anvelopă are un cerc pe post de contur:

```
class Anvelopa {
    . . .
    private String catalogare;
    private Cerc contur;
}
```

- Fiecare automobil are o anvelopă (de fapt are patru)

```
class Automobil extends Vehicul{
    . . .
    private Anvelopa[] anvelope;
}
```


# Dependenţă

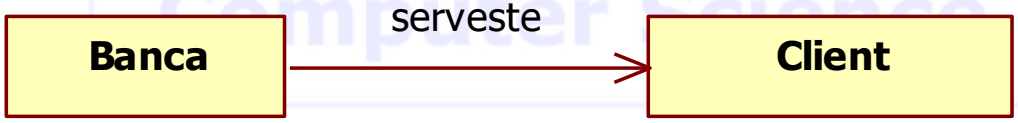
- Relaţie de tip *foloseşte*
- Exemplu: multe dintre aplicaţiile noastre depind de clasa Scanner pentru a citi intrarea
- Agregarea este o formă mai puternică de dependenţă
- Folosim agregarea pentru a ţine minte un alt obiect între apelurile metodelor


# Simboluri UML pentru relaţii

| Relaţie | Simbol | Stil de linie | Forma vârfului |
|---|---|---|---|
| Moştenire |  | Solid | Triunghi |
| Implementare de interfaţă |  | Întrerupt | Triunghi |
| Agregare |  | Solid | Romb |
| Dependenţă |  | Întrerupt | Deschisă |

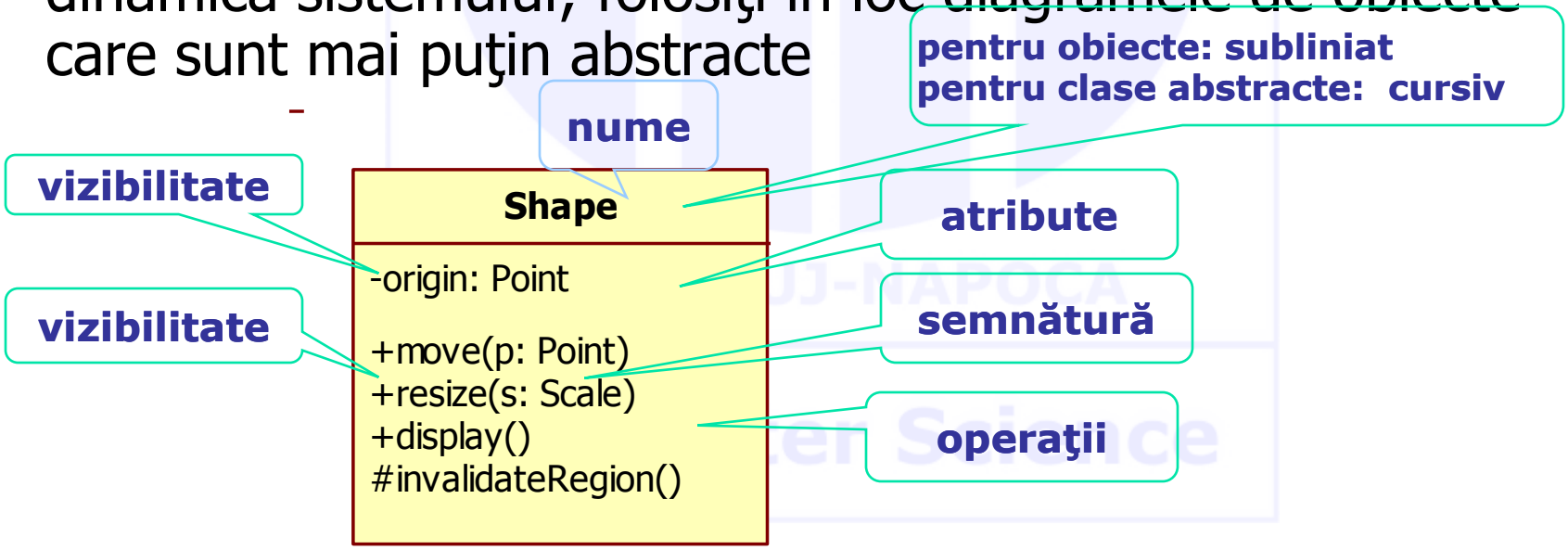
# Agregare şi asociere

- Asocierea: o formă mai generală de relaţie între clase
- Se foloseşte de *timpuriu* în faza de proiectare
- O clasă este asociată cu o alta dacă putem naviga de la obiectele unei clase la obiectele celeilalte clase
- Fiind dat un obiect a **`Banca`**, putem naviga la obiecte **`Client`**



# Diagramă de clase

- Reprezintă un set de clase, interfeţe, colaborări şi alte relaţii
- Reflectă *proiectul static* al unui sistem
- Poate genera confuzii dacă este folosit pentru a explica dinamica sistemului; folosiţi în loc diagramele de obiecte care sunt mai puţin abstracte

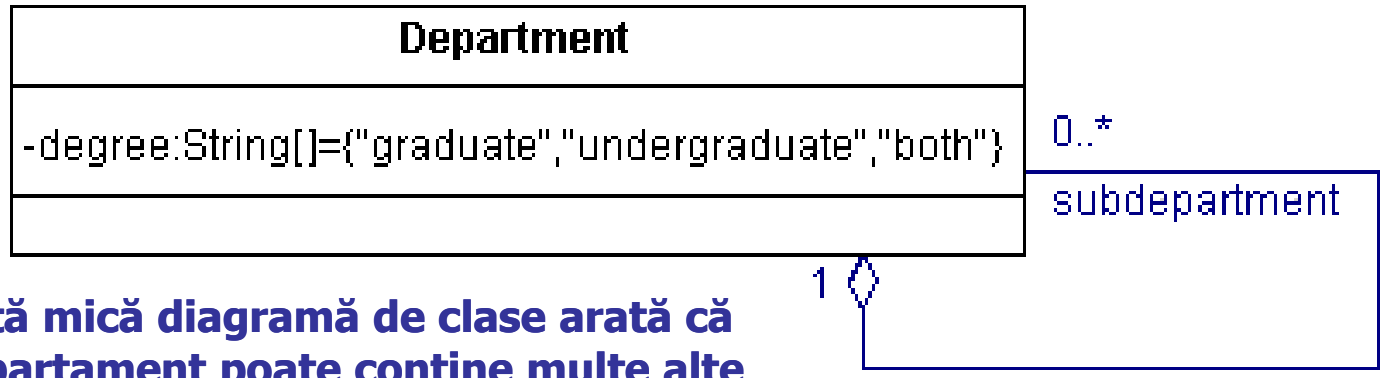



# Diagrame de obiecte

- Reprezintă un set de obiecte (instanţe de clase) şi relaţiile dintre acestea.
- Un instantaneu static al unei vederi dinamice a sistemului.
- Reprezintă cazuri reale sau cazuri prototip.
- Foarte utile înainte de dezvoltarea diagramelor de clase.
- Merită salvate ca elaborări ale diagramelor de clase.

# Diagrame de obiecte

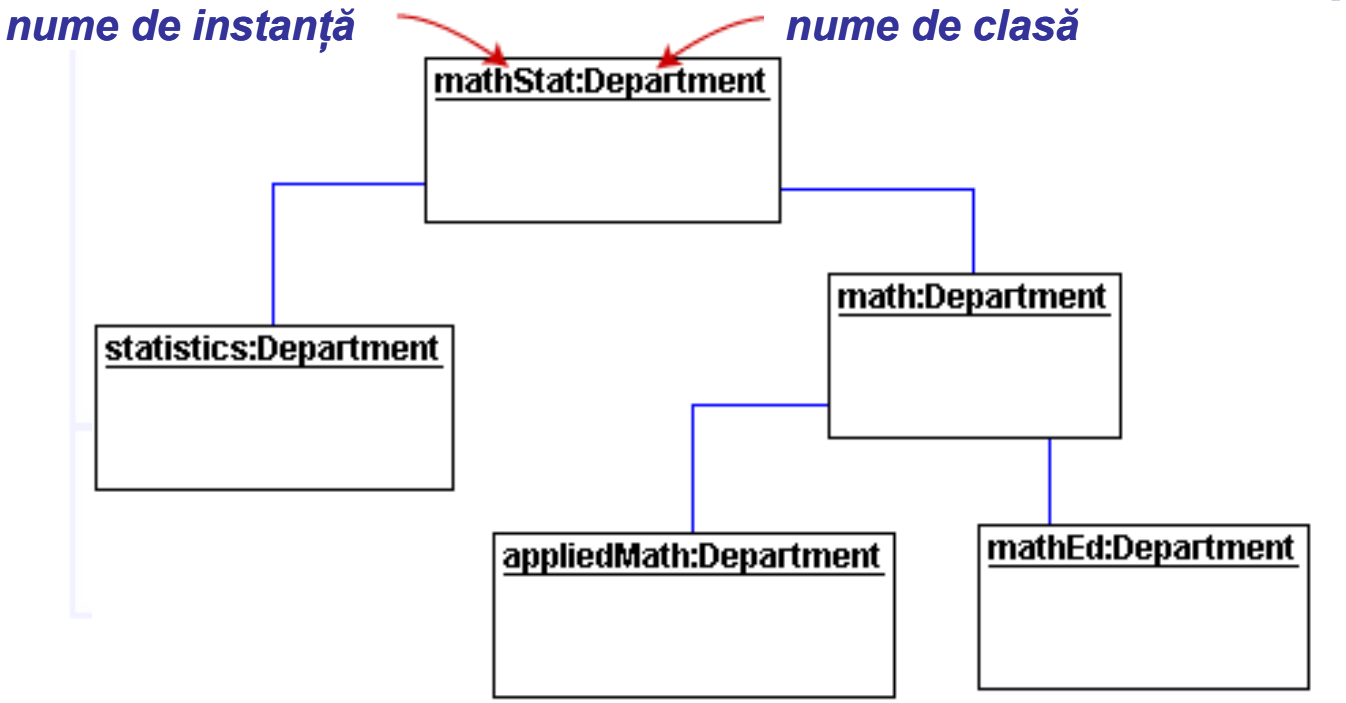
- Utile în explicarea de părţi mici cu relaţii complicate, mai ales în cazul *relaţiilor recursive*
- Fiecare dreptunghi din diagrama de obiecte corespunde unei singure instanţe.
- Numele instanţelor (numele obiectelor) sunt subliniate în diagramele UML.



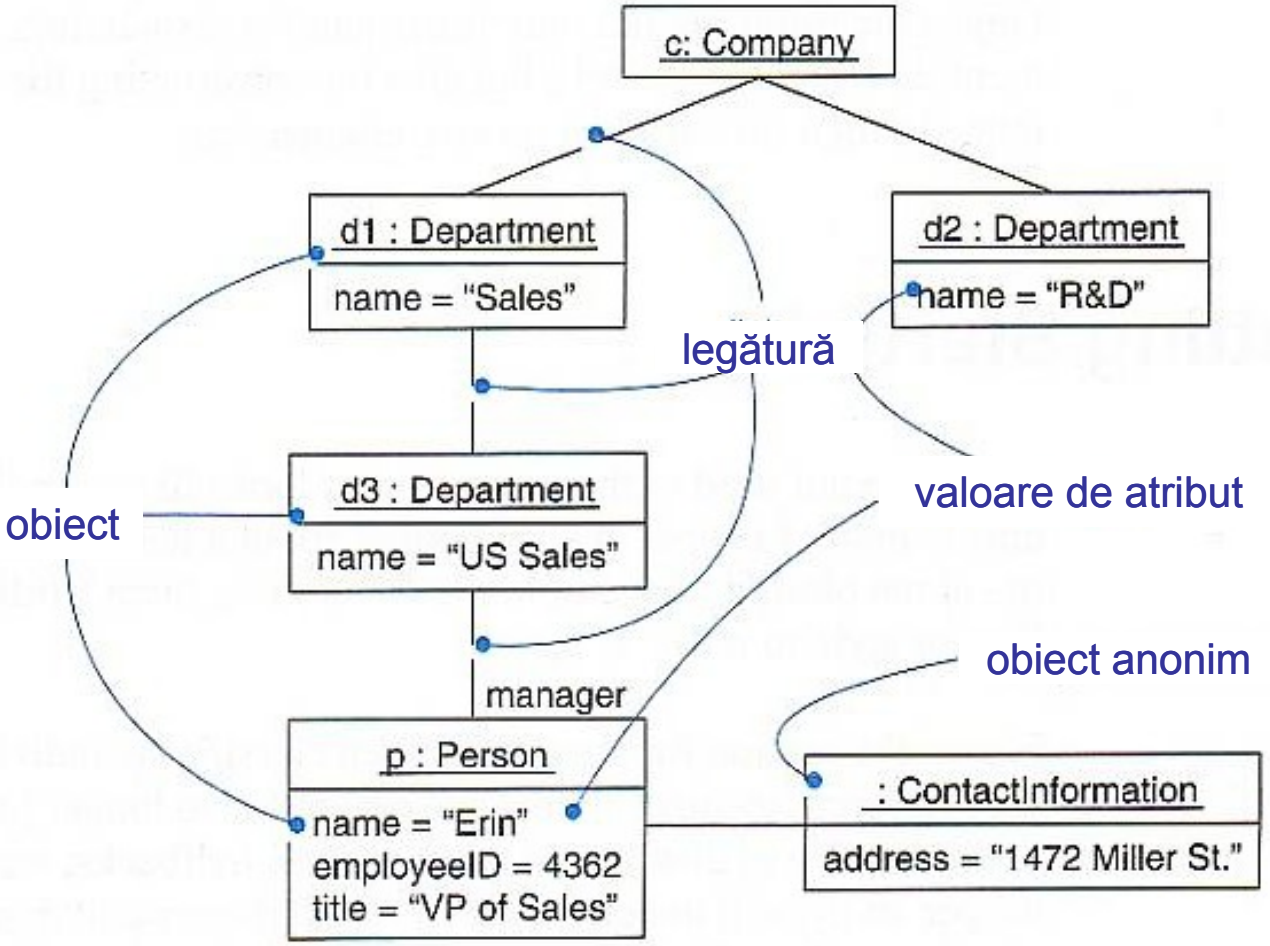

**Această mică diagramă de clase arată că un departament poate conţine multe alte subdepartamente.**

# Diagrame de obiecte

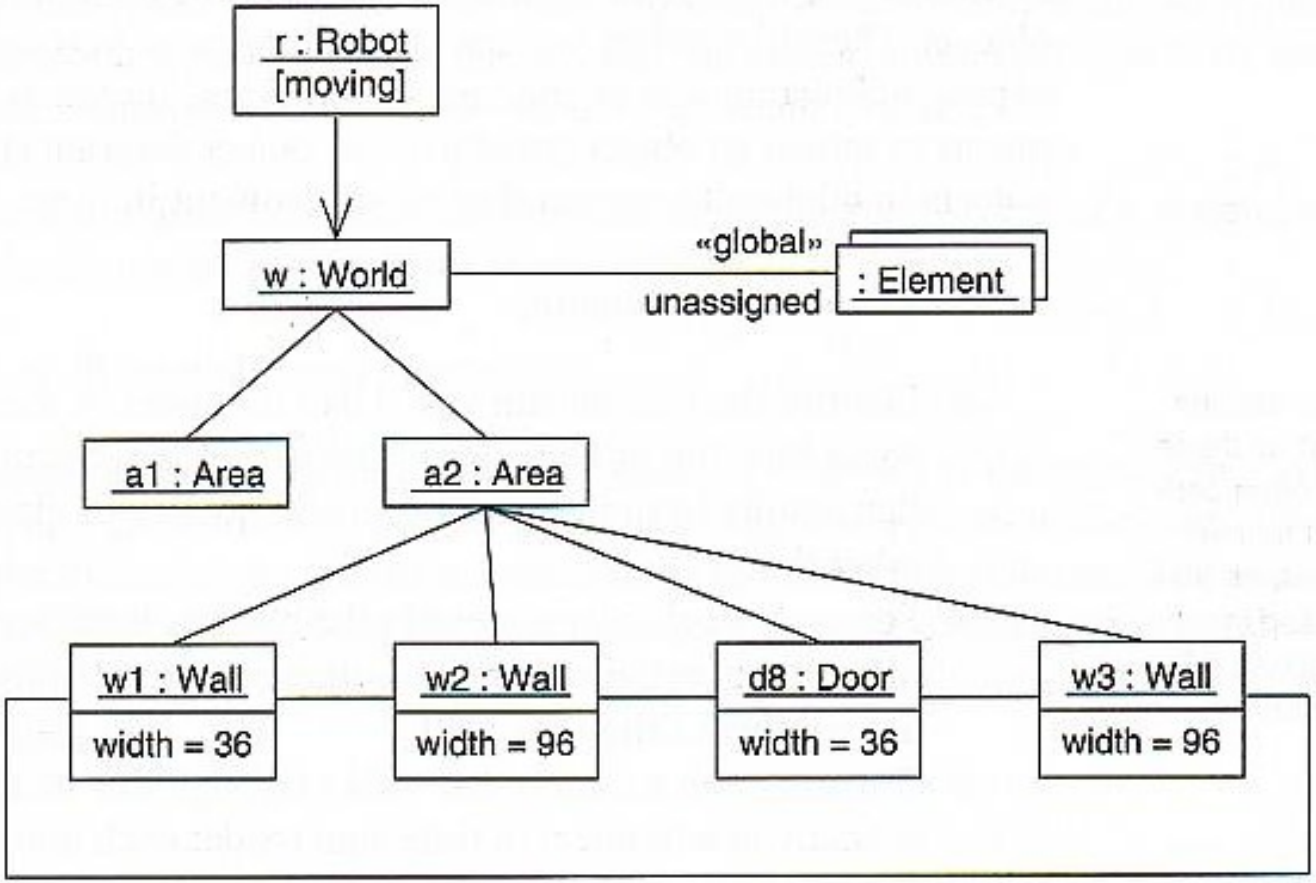
- Diagrama de obiecte de mai jos este o instanţiere a diagramei de clase, clasele fiind înlocuite de exemple concrete
- Numele de clase sau instanţe pot fi omise dacă semnificaţia diagramei rămâne clară.



# Alt exemplu de diagramă de obiecte



# Alt exemplu de diagramă de obiecte






# Procesul de dezvoltare în cinci paşi


# Procesul de dezvoltare în cinci paşi

- Colectăm cerinţele
- Folosim cartele CRC pentru a determina clasele, responsabilităţile şi colaboratorii
- Folosim diagrame UML pentru a înregistra relaţiile dintre clase
- Folosim **`javadoc`** pentru a documenta comportamentul metodelor
- Implementăm programul


# Reguli pentru determinarea claselor

- Între 3 şi 5 responsabilităţi pe clasă
- Nu există clase singuratice
- Feriţi-vă de multe clase mici
- Feriţi-vă de puţine clase mari
- Feriţi-vă de "functoizi" – un functoid este de fapt o funcţie procedurală normală deghizată în clasă.
- Feriţi-vă de clase omnipotente
  - Căutati clase care au "system" sau "controller" în nume!
- Evitaţi arborii de moştenire adânci


# Exemplu: factură simplificată

FACTURA

Maria s.r.l.
str. Mare nr. 1
Un oraş, 554400

| Descriere | Preţ unitar | Cantitate | Total |
|---|---|---|---|
| Spray XXL | 8.99 | 3 | 26.97 |
| Şerveţele Super | 2.99 | 4 | 11.96 |
| Caiet studenţesc | 3.99 | 2 | 7.98 |

Suma de plătit: 46.91


# Exemplu: factură simplificată

- Clase care vin în minte: **`Factura`**, **`Rând`**, şi **`Client`**
- Este o idee bună să păstrăm o listă de clase candidate
- Folosim brainstorming-ul, pur şi simplu punem toate ideile de clasă în listă
- Le putem tăia pe cel inutile ulterior


# Determinarea claselor

- Ţineţi minte următoarele puncte:
  - Clasele reprezintă mulţimi de obiecte cu acelaşi comportament
    - Entităţile cu apariţii multiple în descrierea problemei sunt candidaţi buni pentru obiecte
    - Aflaţi ce au în comun
    - Proiectaţi clasele pentru a surprinde ce este comun
  - Reprezentaţi unele entităţi ca obiecte, iar altele ca tipuri primitive
    - Ar trebui să facem Adresa o clasă sau să folosim un String?
  - Nu toate clasele pot fi descoperite în faza de analiză
  - Unele clase pot exista deja


# Tipărirea unei facturi – cerinţe

- Sarcina: tipărirea unei facturi
- Factura: descrie preţurile pentru un set de produse în anumite cantităţi
- Omitem lucrurile mai complicate – aici
  - Date, taxe şi codurile de factură şi de client
- Tipărim factura cu
  - Adresa clientului, toate rândurile, suma de plătit
- Rândul conţine
  - Descriere, preţ unitar, cantitatea comandată, preţul total
- Pentru simplitate nu creăm interfaţa cu utilizatorul
- Programul de test: adaugă rânduri în factură şi apoi o tipăreşte


# Tipărirea unei facturi – cartele CRC

- Descoperim clasele
- Substantivele identifică clasele posibile

**Factura**
**Adresa**
**Rând**
**Produs**
**Descriere**
**Preţ**
**Cantitate**
**Total**
**Suma de plătit**


# Tipărirea unei facturi – cartele CRC

- Analizăm clasele

```
Factura
Adresa
Rând              // Înregistrează produsul şi cantitatea
Produs
Descriere         // Câmp al clasei produs
Pret              // Câmp al clasei produs
Cantitate         // Nu este un atribut al unui Produs
Total             // Calculat, nu se memorează
Suma de plătit    // Calculată, nu se memorează
```

- Clasele după un proces de eliminare

**Factura**
**Adresa**
**Rând**
**Produs**


# Motive pentru rejectarea unei clase candidate

| **Semnal** | **Motiv** |
|---|---|
| ***Clasa cu nume verb (infinitiv sau imperativ)*** | Poate fi o subrutina, nu o clasa |
| ***Clasa cu o singura metodă*** | Poate fi o subrutina, nu o clasa |
| ***Clasă descrisa ca "efectuează" ceva*** | Poate să nu fie o abstracţiune propriu-zisă |
| ***Clasă fără metode*** | Poate fi o informaţie opacă, nu un TDA, sau poate fi un TDA la care s-au omis rutinele |
| ***Clasă cu zero sau foarte puţine atribute (dar care moşteneşte de la părinţi)*** | Poate fi un caz de "taxomanie" |
| ***Clasă care acoperă câteva abstracţiuni*** | Ar trebui divizată în mai multe clase, câte una pentru fiecare abstracţiune |


# Exemple foarte bune

- Carte de adrese
  - http://www.math-cs.gordon.edu/courses/cs211/AddressBookExample/
- Automat bancar (Automatic Teller Machine)
  - http://www.math-cs.gordon.edu/courses/cs211/ATMExample/


# Cartelele CRC pentru tipărirea unei facturi

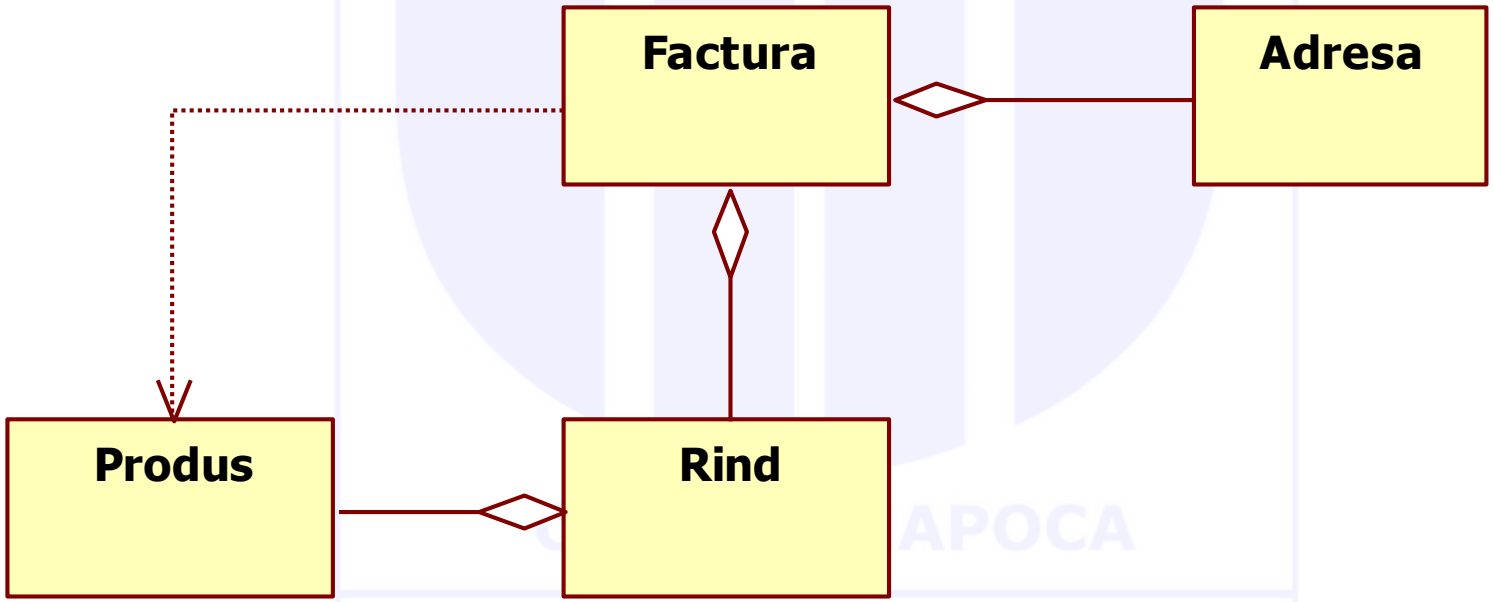
- Atât **`Factura`** cât şi **`Adresa`** trebuie să se poată autoformata – responsabilităţi:
  - **`Factura`** *formatează factura* şi
  - **`Adresa`** *formatează adresa*
- Adăugăm colaboratori pe cartela facturii: **`Adresa`** şi **`Rand`**
- Pentru cartela **`Produs`** – responsabilităţi: *obţine descrierea, obţine preţul unitar*
- Pentru cartela CRC **`Rand`** – responsabilităţi: *formatează articolul, obţine preţul total*


# Cartelele CRC pentru tipărirea unei facturi

- **`Factura`** trebuie populată cu produse şi cantităţi:

| Factura | |
|---|---|
| *formatează factura*<br>*adaugă un produs şi o cantitate* | `Adresa`<br>`Rind`<br>`Produs` |

# Tipărirea unei facturi – diagrama UML




# Instrumente pentru realizarea diagramelor UML

- ArgoUML:
  - http://argouml.tigris.org
  - rulează pe orice platformă Java
- StarUML
  - http://sourceforge.net/projects/whitestaruml/
  - proiect "open source"
- Poseidon for UML Community Edition
  - http://gentleware.com/downloadcenter.0.html
- IBM Rational Modeler:
  - http://www-03.ibm.com/software/products/en/ratimode
- Direct online:
  - https://www.gliffy.com
  - https://www.draw.io/


# Tipuri de specificaţii

- Diagrame de clase
- Diagrame de obiecte
- Diagrame de activităţi (diagrame de curgere a controlului)
- Aserţiuni (precondiţii, postcondiţii, invarianţi)
  - Altele
- Reţineţi că primele trei sunt specificaţii incomplete


# Specificarea claselor

- O specificaţie software indică sarcina (sau un aspect al sarcinii) care se presupune a fi efectuată la execuţia software respectiv
- O specificaţie de clasă defineşte semantica (comportamentul) unei clase prin:
  - invariant de clasă care descrie ceea ce este întotdeauna adevărat pentru obiectele clasei.
  - specificaţii pentru fiecare dintre metodele clasei.
- Fiecare *specificaţie de metodă* constă din
  - o precondiţie (opţional),
  - o clauză modifică (opţional), şi
  - o postcondiţie.


# Specificarea metodelor

- O *precondiţie* declară condiţiile care sunt necesare pentru ca metoda să se execute în mod corespunzător
- O clauză *modifică* reprezintă o listă de obiecte care ar putea fi modificate prin execuţia metodei.
- O *postcondiţie* declară ce este adevărat atunci când metoda şi-a finalizat execuţia


# Reading

- Eckel: capitolul 10
- Barnes: capitolele 5, secțiunile 10.6, 10.7
- Deitel: secțiunea 10.7, capitolele 12, 13