



Programare orientată pe obiecte

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

Computer Science



Proiectarea orientată pe obiecte

1. Descoperim clasele
2. Determinăm responsabilitățile fiecărei clase
3. Descriem relațiile dintre clase

OF CLUJ-NAPOCA

Computer Science



Unified Modeling Language (UML)

- UML este notația internațională standard pentru analiza și proiectarea orientată pe obiecte
- UML 2.0 definește treisprezece tipuri de diagrame, împărțite în două categorii:
 - **Diagrame de structură:** șase tipuri de diagrame reprezintă structura statică a aplicației:
 - *Diagrama de clase, diagrama de obiecte*, diagrama de componente, diagrama de structură compozită, diagrama de pachete și diagrama de desfășurare sistematică
 - **Diagrame de comportament:** șapte diagrame ce reprezintă tipuri generale de comportament:
 - Diagrama de activități, diagrama de interacțiune, diagrama cazurilor de utilizare (use case), diagrama de secvențe, diagrama de stare, diagrama de comunicare, diagrama de timp



Descoperirea claselor

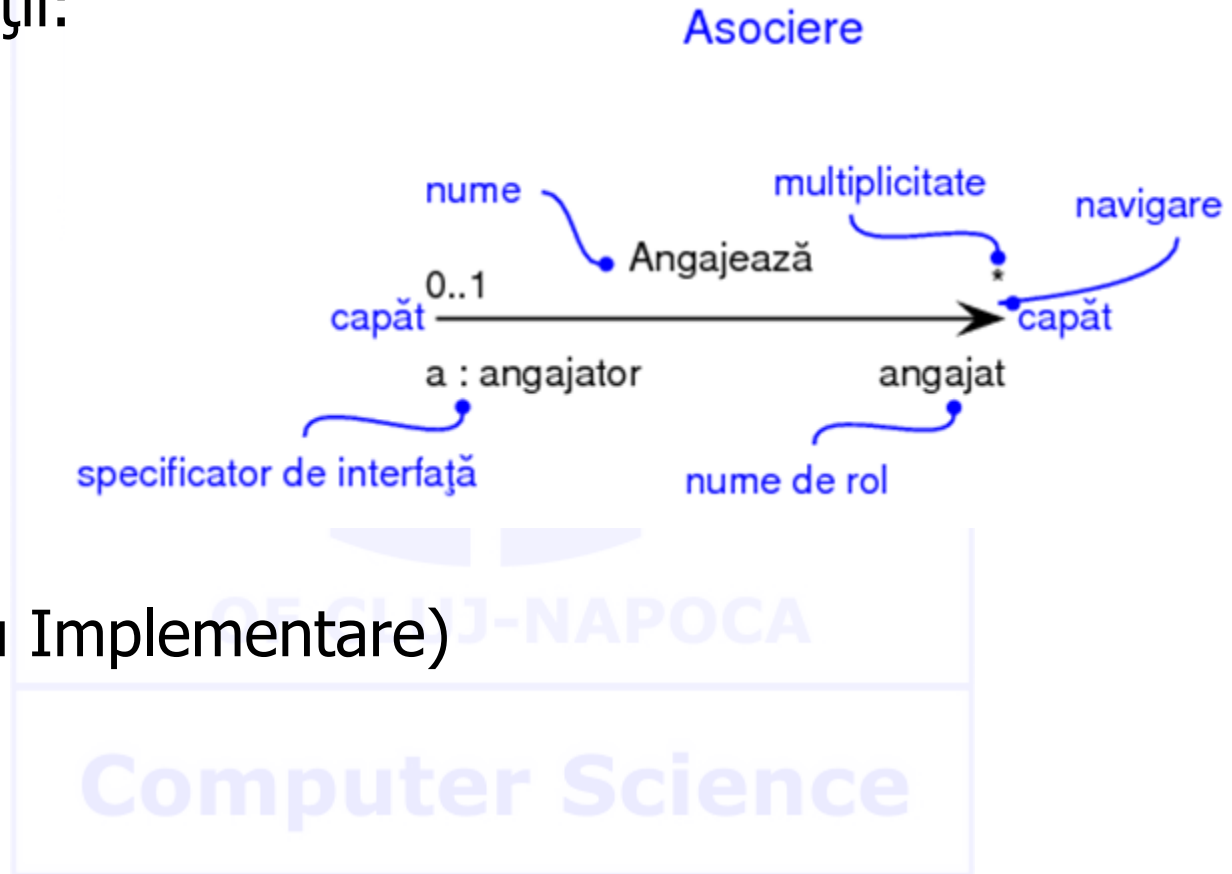
- O clasă reprezintă un concept util
 - Entități concrete: conturi bancare, elipse, produse, ...
 - Concepte abstracte: fluxuri (streams), ferestre grafice, ...
- Găsim *clasele* căutând *substantive* în descrierea sarcinii
- Definim comportamentul fiecărei clase
- Găsim *metodele* căutând *verbe* în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



Relații între entitățile reprezentate

Tipuri de relații:

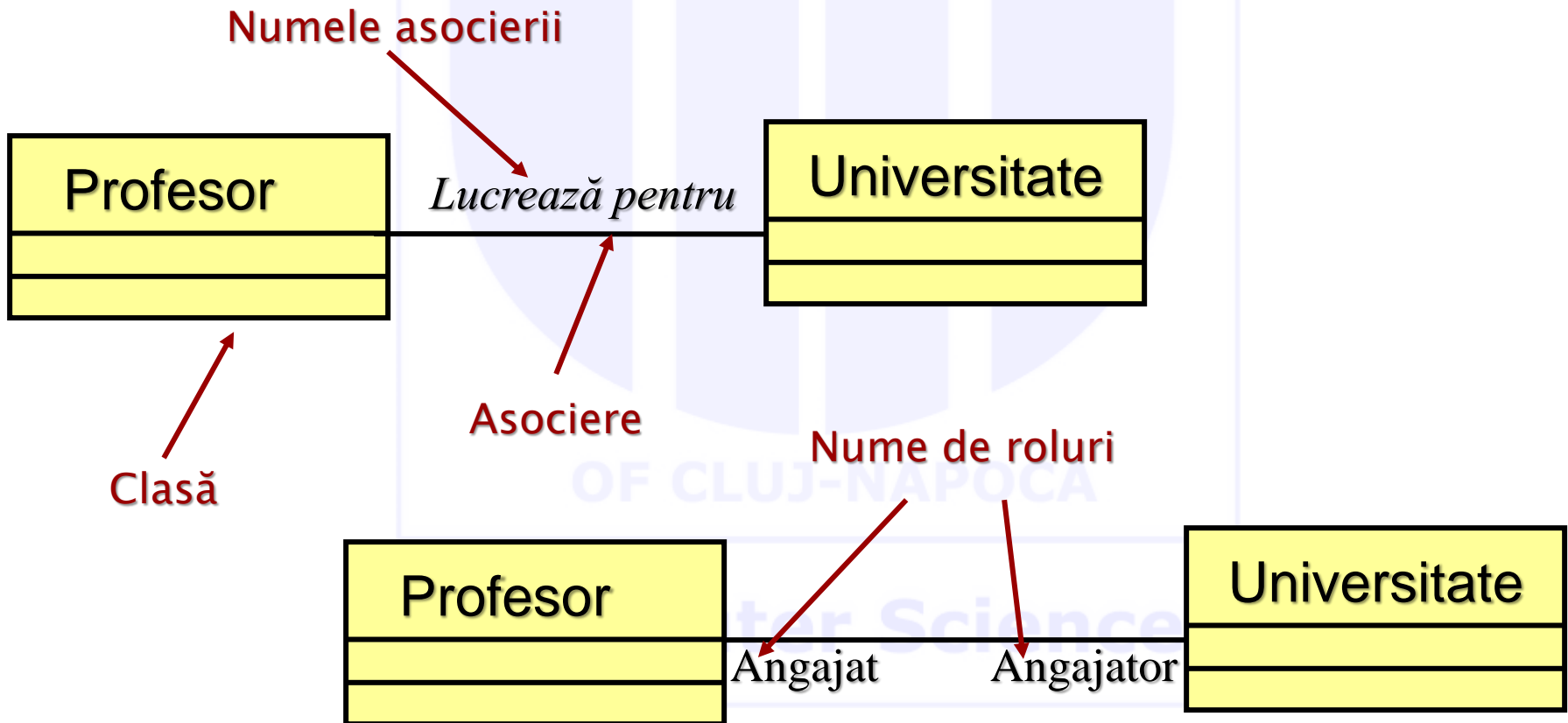
- Asocierie
 - Agregare
 - Compoziție
- Dependentă
- Generalizare
- Realizare (sau Implementare)





Relații: Asociere

- Modelează o conexiune semantică între clase





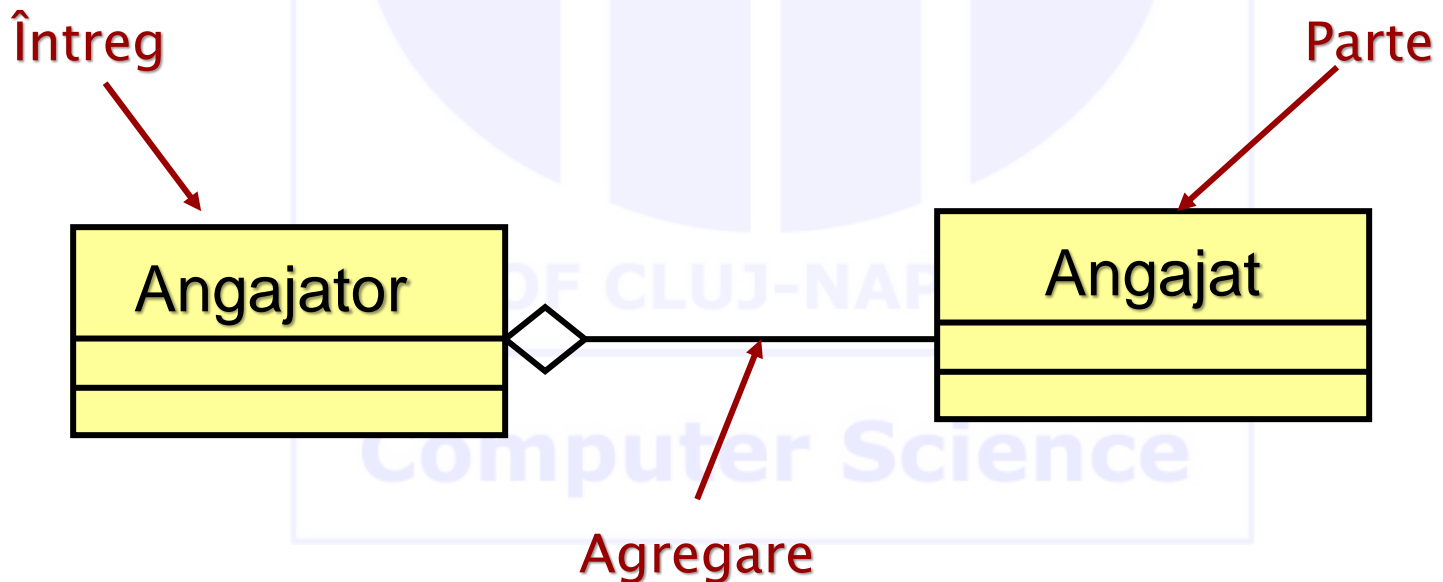
Folosirea asocierilor

- Trei scopuri generale. Pentru a reprezenta:
 - O situație în care un obiect de o clasă *folosește* serviciile unui alt obiect, sau ele își folosesc reciproc serviciile – adică un obiect îi trimite mesaje celuilalt sau își trimit mesaje între ele
 - În primul caz, navigabilitatea poate fi unidirecțională; în cel de al doilea, ea trebuie să fie bidirecțională
 - *Agregarea sau compoziția* – unde *obiecte de o clasă sunt întregi compuși din obiecte de cealaltă clasă ca părți*
 - În acest caz, o relație de tip “folosește” este implicit prezentă – întregul folosește părțile pentru a-și îndeplini funcția, iar părțile pot și ele avea nevoie să folosească întregul
 - O situație în care obiectele sunt *înrudite*, chiar dacă nu schimbă mesaje
 - Aceasta se întâmplă de obicei când cel puțin unul dintre obiecte este folosit în esență la stocare de informație



Relații: Agregare

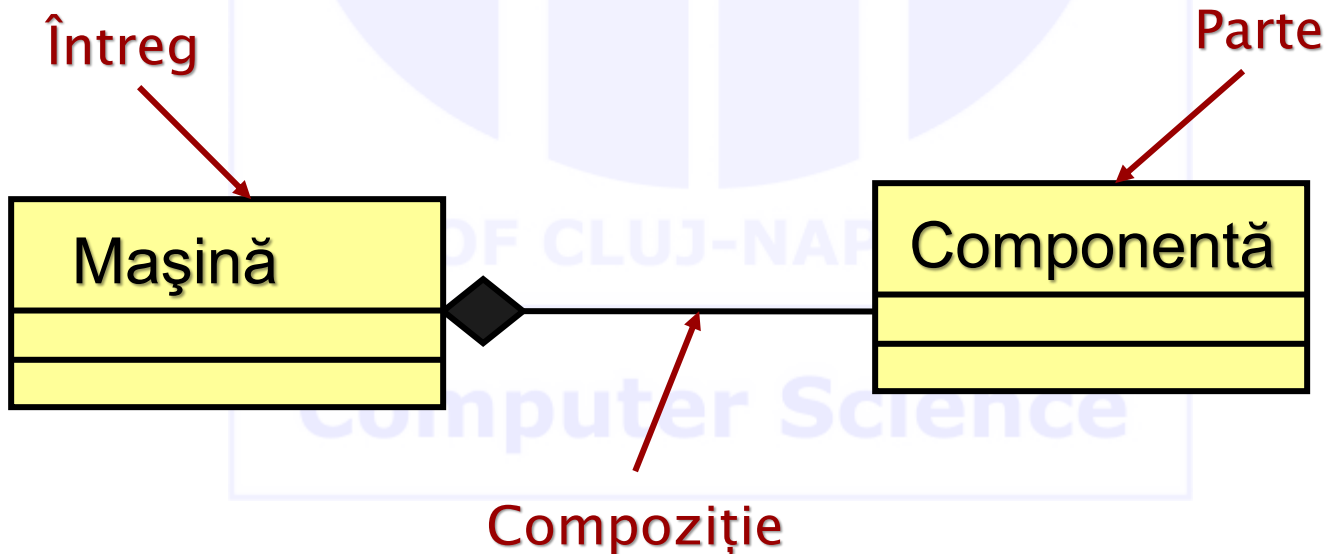
- Relatie de tip: "are o/un"
- O formă specială de asociere care modelează relația parte-întreg între un agregat (întregul) și părțile sale





Relații: Compoziție

- Relatie de tip: "este parte a"
- O formă de agregare cu posesiune *puternică* și durate de viață care coincid
 - Părțile nu pot supraviețui fără existența întregului/agregatului





Asociere: Multiplicitate și navigare

- Multiplicitatea definește câte obiecte participă într-o relație
 - Numărul de instanțe ale unei clase în raport cu o instanță a celeilalte clase
 - Specificat pentru fiecare capăt al asocierii
- Asocierile sunt implicit bidirecționale, dar adesea este de dorit să se restrângă navigarea la o singură direcție
 - Dacă navigarea este restricționată, se adaugă o săgeată pentru a indica direcția de navigare

OF CLUJ-NAPOCA
Computer Science



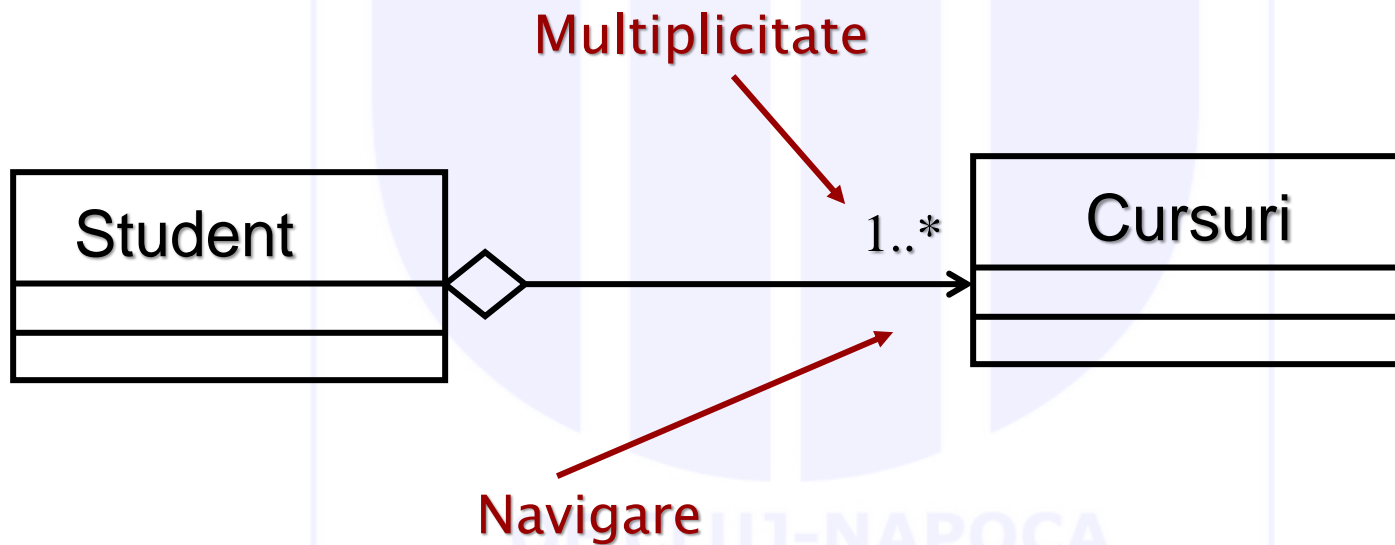
Asociere: multiplicitate

- 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

Computer Science



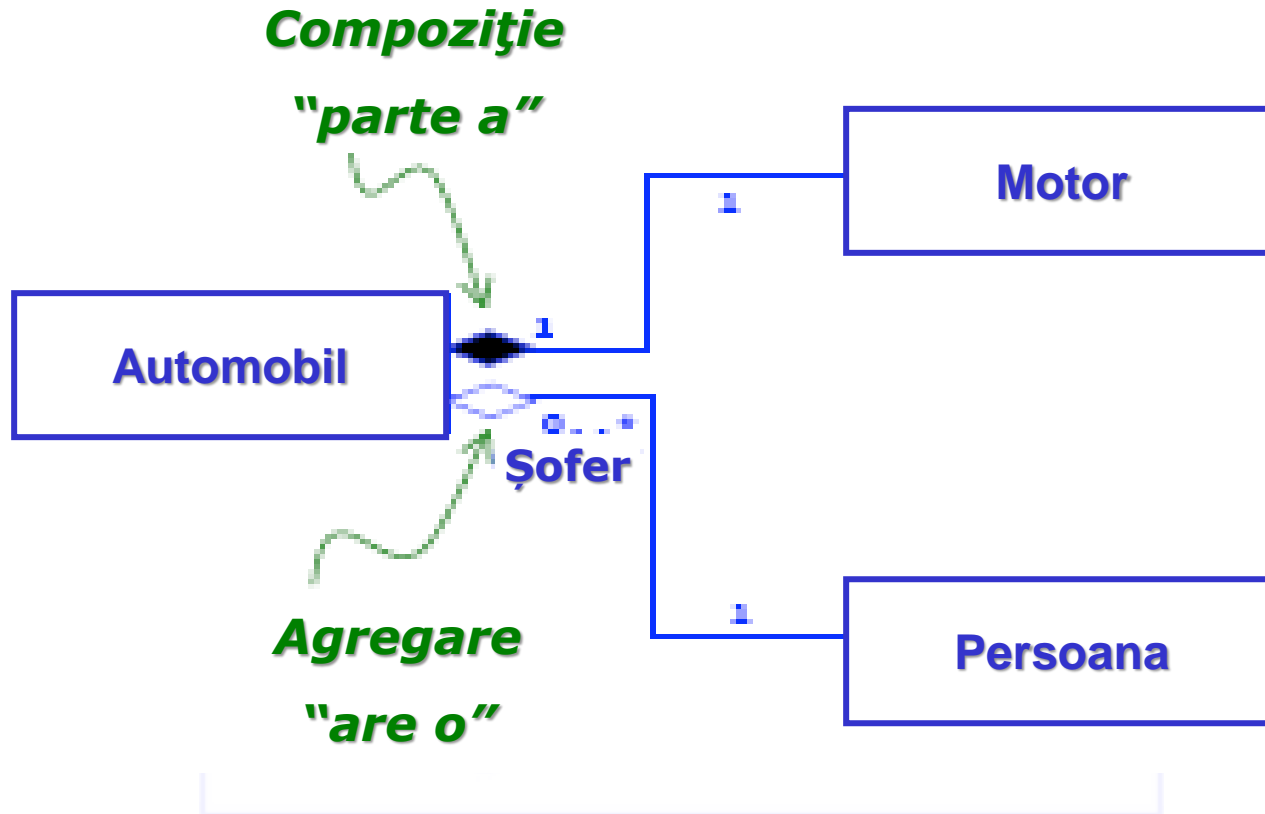
Exemplu: multiplicitate și navigare



Computer Science



Exemple de asocieri





Observații

- **Teste pentru relații parte-întreg adevărate**
 - **Tranzitivitate:** dacă "A este parte a lui B" și "B este parte a lui C" atunci "A este parte a lui C"
 - Unghia este parte a degetului, degetul este parte a mâinii; atunci unghia este parte a mâinii
 - **O problemă a unei părți este o problemă a întregului**
 - O rană la unghie este o rană a mâinii
 - Pozițiile sunt parte a sistemului electric al automobilului. Un defect al pozițiilor este un defect al automobilului
- **Este-parte-a *e diferit* de**
 - **Este-Conținut-În:** cămași, pantaloni,... --- dulap (observați că testul de defectare nu ține aici: pantalonii defecti nu înseamnă că dulapul e defect)
 - **Este-Legat-De:** dulap... --- persoană (care îl posedă)
 - **Este-Ramură-A:** artera iliacă,... --- aorta
 - **Se-Afla-În:** casă... --- stradă



Când să folosim agregarea

- **Ca regulă generală, se poate marca o asociere ca agregare, dacă sunt adevărate următoarele:**
 - Se poate spune că
 - Părțile 'sunt parte' a agregatului
sau
 - Agregatul 'este compus din' părți
 - Când ceva deține sau controlează agregatul, atunci acel ceva deține sau controlează părțile



Asociere, agregare și compoziție

Asociere

Obiectele știu unul despre altul astfel încât pot lucra împreună

Agregare

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

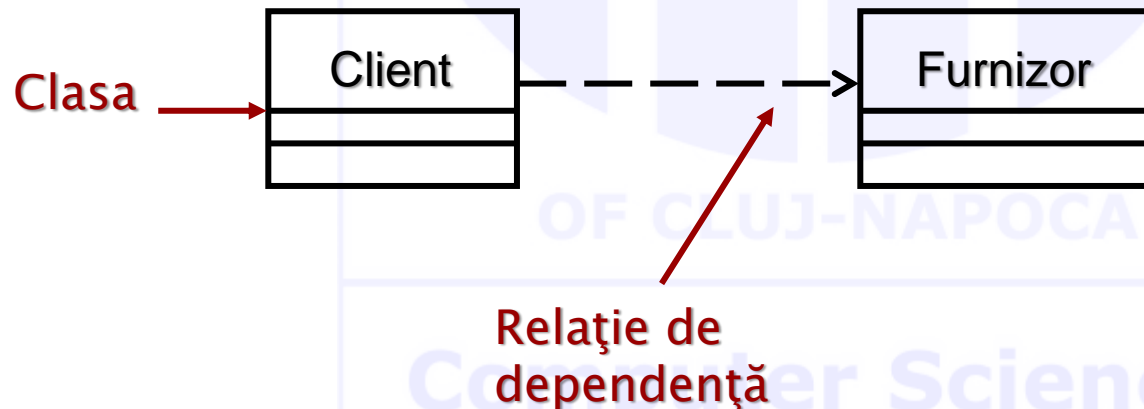
Compoziție

Fiecare parte poate fi membru al unui singur obiect agregat



Relații: dependență

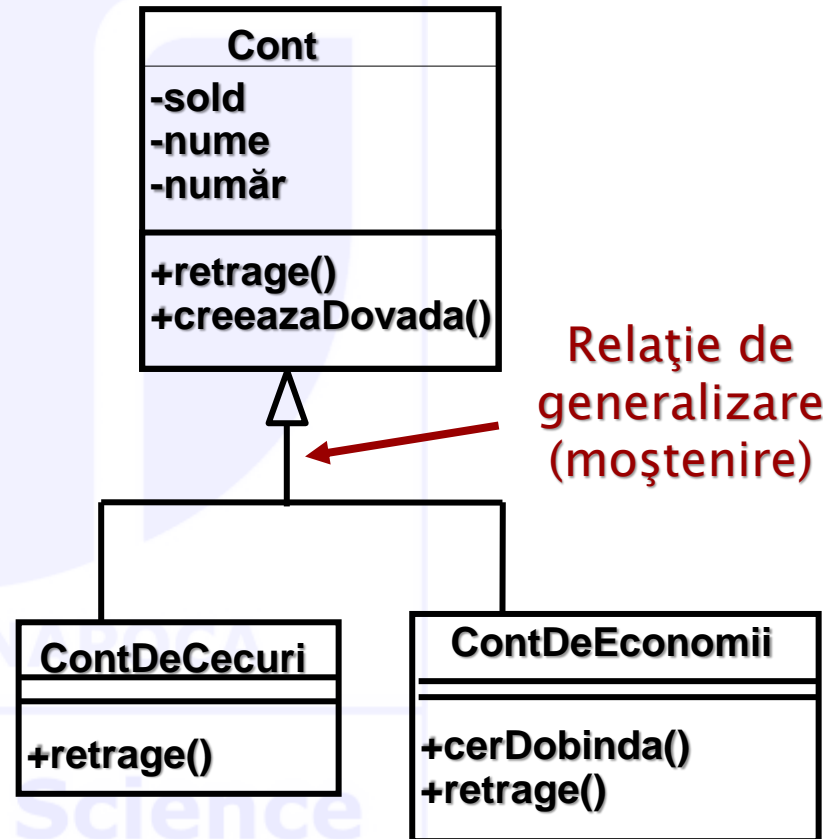
- Relație de tip "*folosește*"
- O relație între două elemente ale modelului în care o schimbare în unul dintre elemente **poate** cauza o schimbare în celălalt





Relații: generalizare (sau moștenire)

- Relație de tipul *este-o/un*
- Relație între o clasă mai generală (superclasă) și una mai specializată (subclasă)
- Exemple:
 - Orice cont de economii este un cont bancar
 - Orice cerc este o elipsă (cu lățime și înălțime egale)





Contra-exemplu de moștenire

- Uneori se abuzează de această relație
 - Ar trebui să fie clasa **Anvelopa** o subclasă a lui **Cerc**?
 - Relația *are-un* (**agregare**) ar fi mai potrivită aici
- Obiectele unei clase conțin referințe la obiectele altei clase
- Folosesc variabile instanță
 - O anvelopă are un cerc pe post de contur:

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

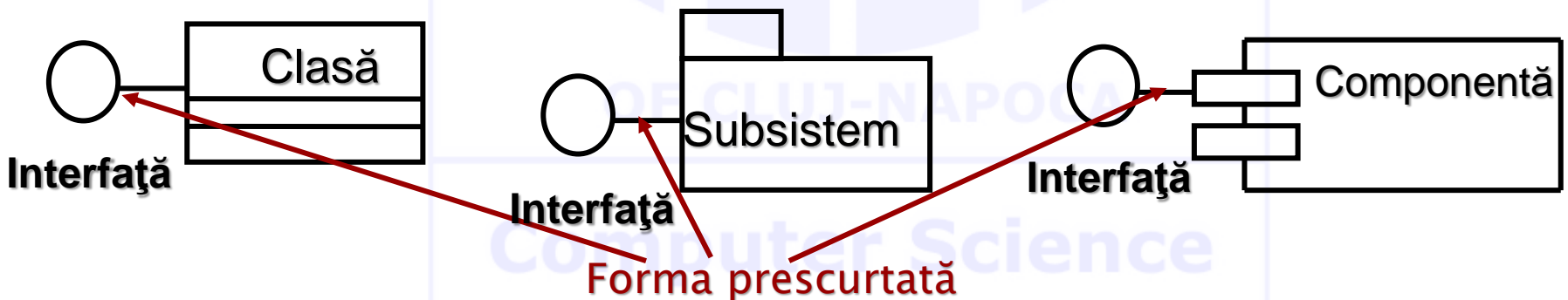
- Fiecare automobil are mai multe anvelope

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



Relații: realizare/ implementare

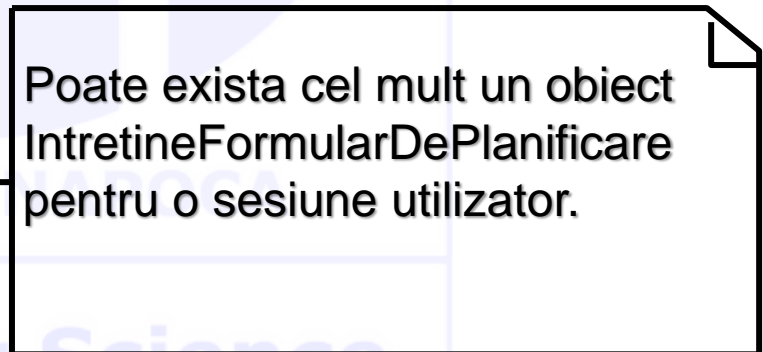
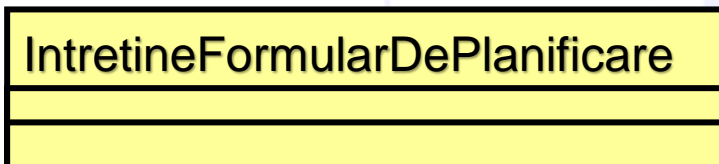
- O interfață servește pe post de contract pe care celălalt este de acord să-l îndeplinească
- Regăsit între:
 - Interfețele și clasele care le realizează (implementează)





Note (adnotări)

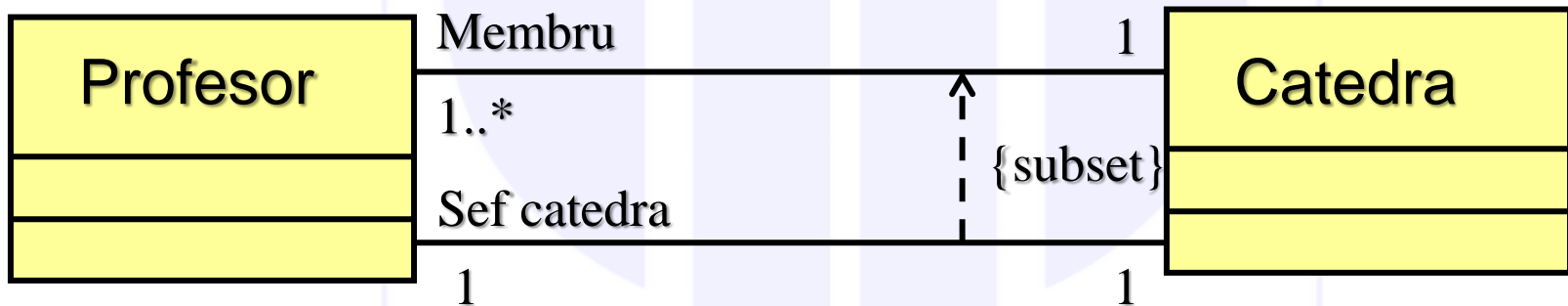
- Se poate adăuga o notă (adnotare) la orice element UML
- Notele se pot adăuga pentru a furniza informații suplimentare în diagramă
- Este un dreptunghi cu colțul dreapta sus îndoit
- Nota poate fi ancorată la un element cu o linie întreruptă





Constrângeri

- Suportă adăugarea de reguli noi sau modificarea regulilor existente



- Această notație este folosită pentru a surprinde două relații între obiecte de tip Profesor și 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



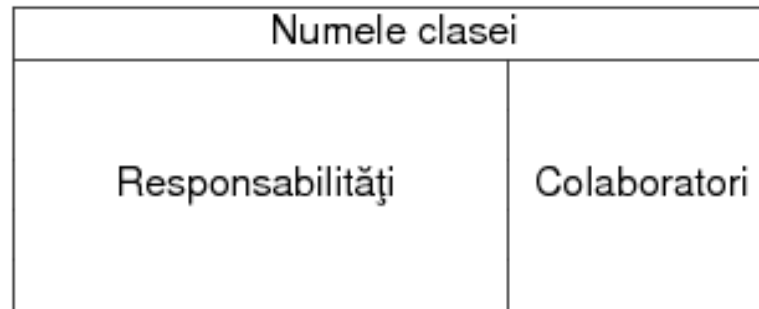
Cartela CRC (*Class-Responsibility-Collaboration*)

- Cartela CRC: descrie o clasă, responsabilitățile și colaboratorii săi
- Se folosește 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 (*Class-Responsibility-Collaboration*)

- Cartela CRC







- Un exemplu de cartelă CRC vs. diagramă de clase

Class CardReader	
Responsibilities	Collaborators
Tell ATM when card is inserted	ATM
Read information from card	Card
Eject card	
Retain card	

CardReader
- atm: ATM
+ <u>CardReader(atm: ATM)</u>
+ readCard(): Card
+ ejectCard()
+ retainCard()



Simboluri UML pentru relații

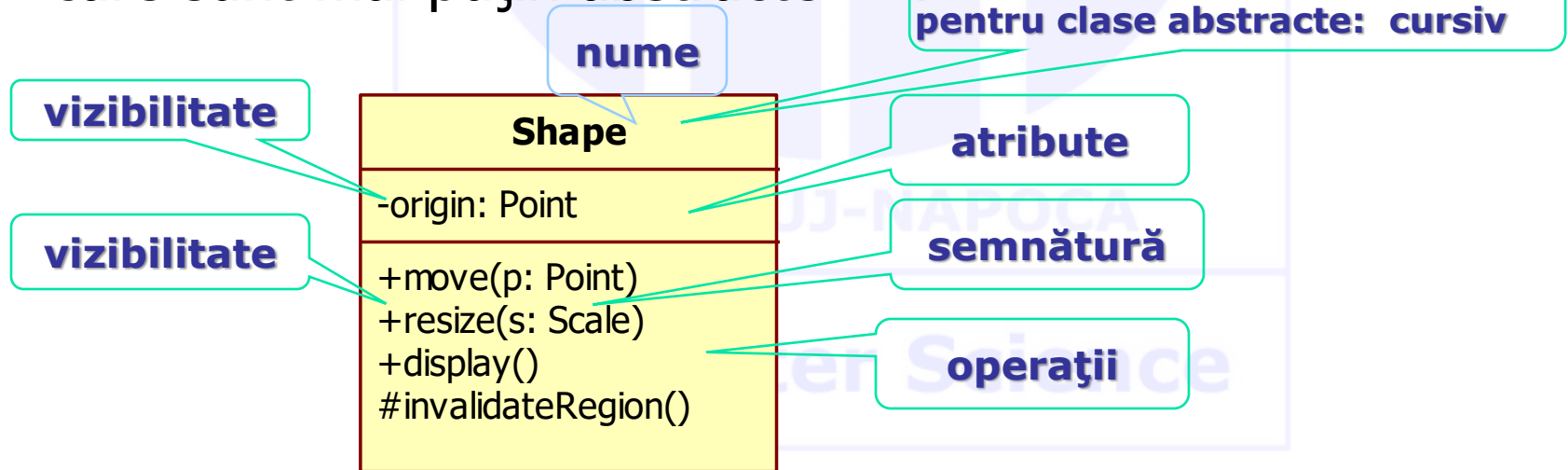
Relație	Simbol	Stil de linie	Forma vârfului
Moștenire		Solid	Triunghi
Implementare de interfață		Întrerupt	Triunghi
Agregare		Solid	Romb
Dependență		Întrerupt	Deschisă

Computer Science



Diagramă de clase

- Reprezintă un set de clase, interfețe, colaborări și alte relații
- Reflectă *proiectul static* al unui sistem
- Poate genera confuzii dacă este folosit pentru a explica dinamica sistemului; folosiți în loc diagramele de obiecte care sunt mai puțin abstracte





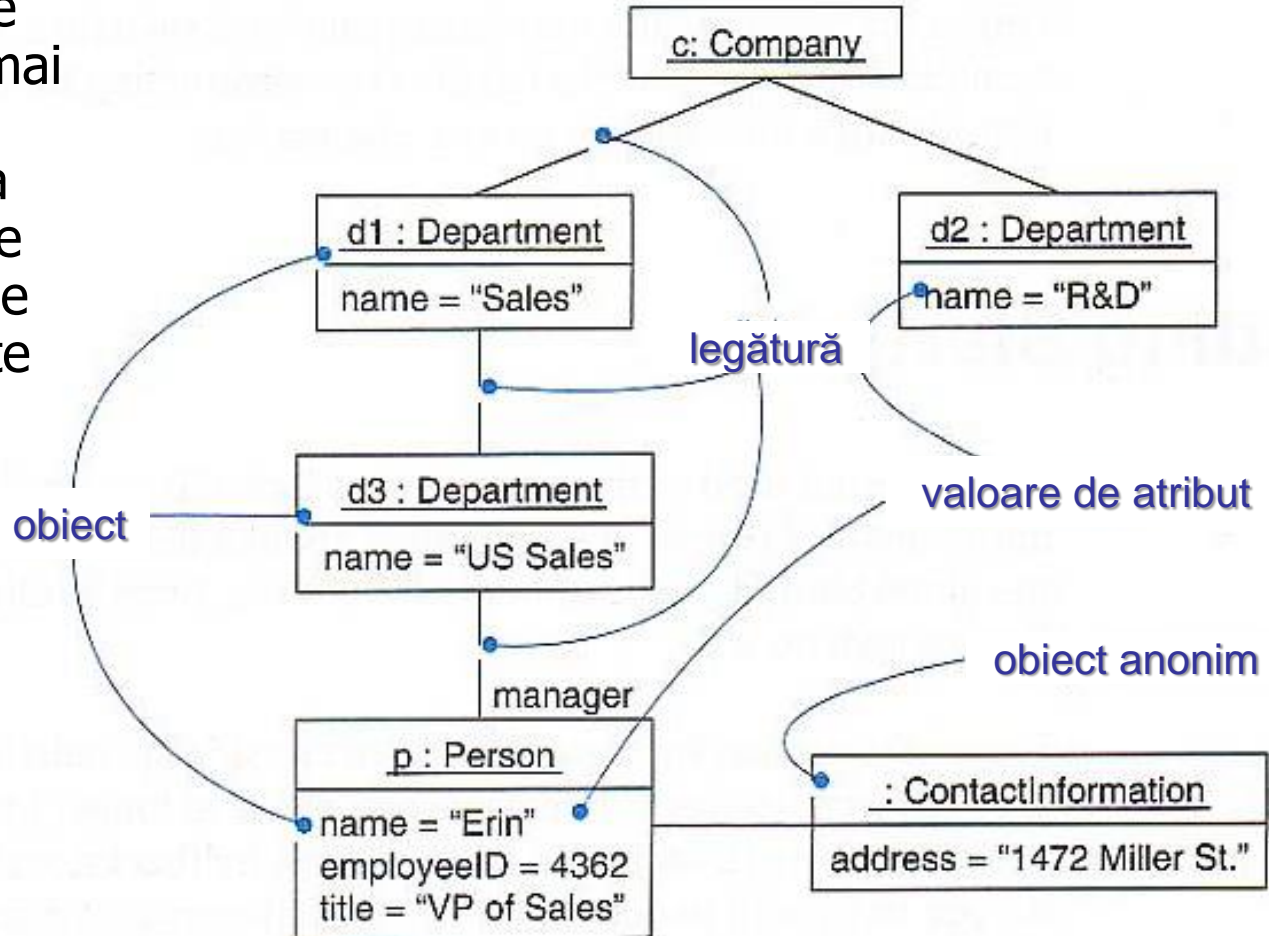
Diagrame de obiecte

- Reprezintă un set de obiecte (instanțe de clase) și relațiile dintre acestea
- Este o *vedere dinamică* a sistemului la un moment dat
- Reprezintă cazuri reale sau cazuri prototip
- Foarte utile înainte de dezvoltarea diagramelor de clase



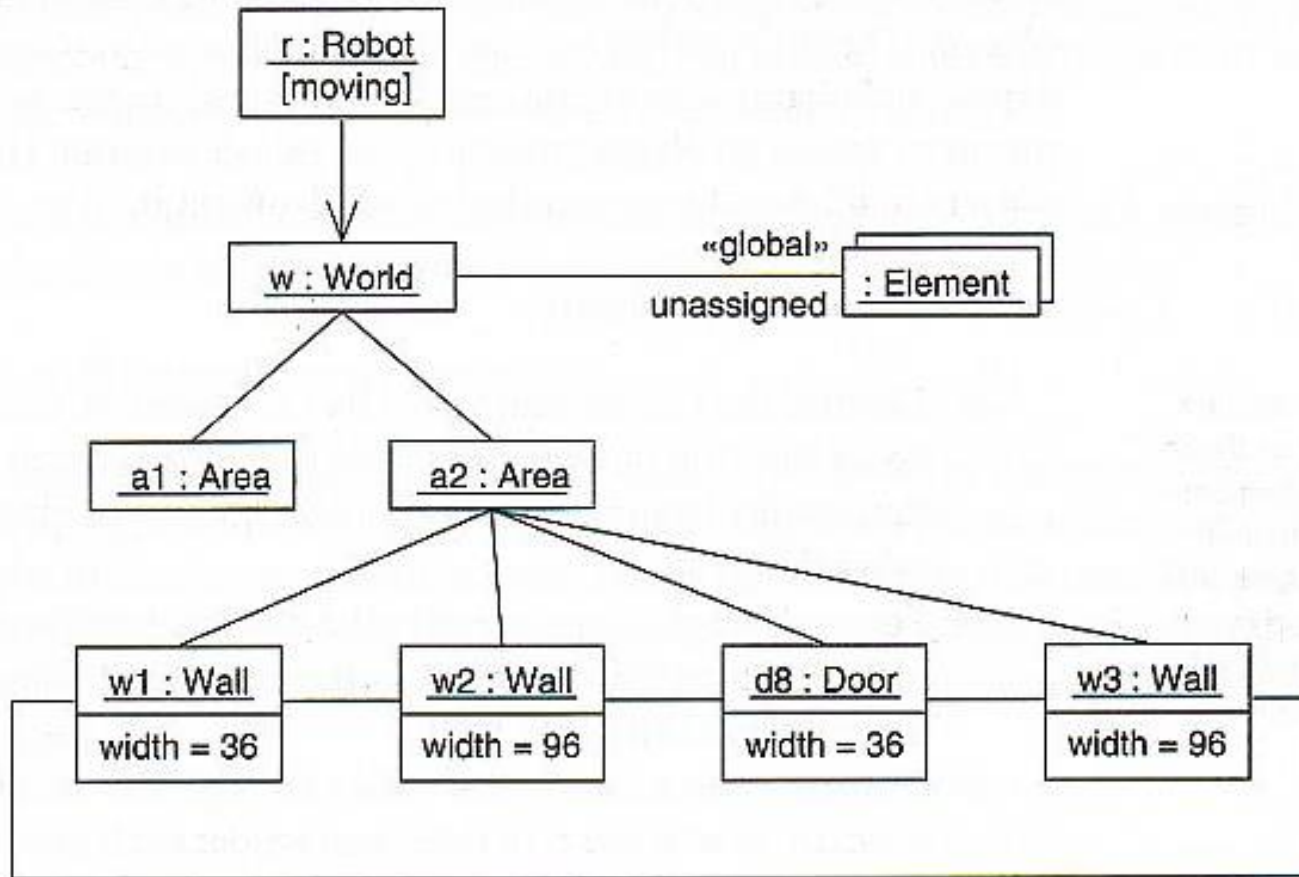
Diagramă de obiecte: exemplu

- Diagrama de obiecte de mai jos este o instanțiere a diagramei de clase (clasele fiind înlocuite de obiecte concrete)





Diagramă de obiecte: un alt exemplu





Procesul de dezvoltare a unei aplicații OO în cinci pași

- Colectăm cerințele
- Folosim cartele CRC pentru a determina clasele, responsabilitățile și colaboratorii
- Folosim diagrame UML pentru a înregistra relațiile dintre clase
- Folosim **javadoc** pentru a documenta comportamentul metodelor
- Implementăm programul



Reguli pentru determinarea claselor

- Între 3 și 5 responsabilități pe clasă
- Nu există clase singuratice
- Feriți-vă de multe clase mici
- Feriți-vă de puține clase mari
- Feriți-vă de "functoizi" – un functoid este de fapt o funcție procedurală normală deghizată în clasă
- Feriți-vă de clase omnipotente
- Evitați arborii de moștenire adânci

Computer Science



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 studentesc	3.99	2	7.98
Suma de plătit:			46.91



Exemplu: factură simplificată

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

OF CLUJ-NAPOCA

Computer Science



Determinarea claselor

- Țineți minte următoarele puncte:
 - Clasele reprezintă mulțimi de obiecte cu același comportament
 - Entitățile cu apariții multiple în descrierea problemei sunt candidați buni pentru obiecte
 - Aflați ce au în comun
 - Proiectați clasele pentru a surprinde ce este comun
 - Reprezentați unele entități 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 – cerințe

- Sarcina: tipărirea unei facturi
- Factura: descrie prețurile pentru un set de produse în anumite cantități
- Omitem lucrurile mai complicate – aici
 - Date, taxe și codurile de factură și de client
- Tipărim factura cu
 - Adresa clientului, toate rândurile, suma de plătit
- Rândul conține
 - Descriere, preț unitar, cantitatea comandată, prețul total
- Pentru simplitate nu creăm interfața cu utilizatorul
- Programul de test: adaugă rânduri în factură și apoi o tipărește



Tipărirea unei facturi – cartele CRC

- Descoperim clasele
- Substantivele identifică clasele posibile

Factura
Adresa
Rînd
Prodot
Descriere
Preț
Cantitate
Total
Suma de plătit



Tipărirea unei facturi – cartele CRC

- Analizăm clasele

```
Factura  
Adresa  
Rînd // Înregistrează produsul și 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  
Rînd  
Produs
```



Motive pentru respingerea unei clase candidate

Semnal	Motiv
<i>Clasă cu nume verb (infinitiv sau imperativ)</i>	Poate fi o subrutină, nu o clasă
<i>Clasă cu o singura metodă</i>	Poate fi o subrutină, nu o clasă
<i>Clasă descrisă că "efectuează" ceva</i>	Poate să nu fie o abstracțiune propriu-zisă
<i>Clasă fără metode</i>	Poate fi o informație opacă
<i>Clasă cu zero sau foarte puține atribute (dar care moștenește de la părinți)</i>	Poate fi un caz de exagerare în crearea de clase noi într-o taxonomie
<i>Clasă care acoperă câteva abstracțiuni</i>	Ar trebui divizată în mai multe clase, câte una pentru fiecare abstracțiune



Cartelele CRC pentru tipărirea unei facturi

- Atât **Factura** cât și **Adresa** trebuie să se poată autoformata – **responsabilități**:
 - **Factura** *formatează factura* și
 - **Adresa** *formatează adresa*
- Adăugăm **colaboratori** pe cartela facturii: **Adresa** și **Rînd**
- Pentru cartela **Produx** – **responsabilități**: *obține descrierea, obține prețul unitar*
- Pentru cartela **Rînd** – **responsabilități**: *formatează articolul, obține prețul total*



Cartelele CRC pentru tipărirea unei facturi

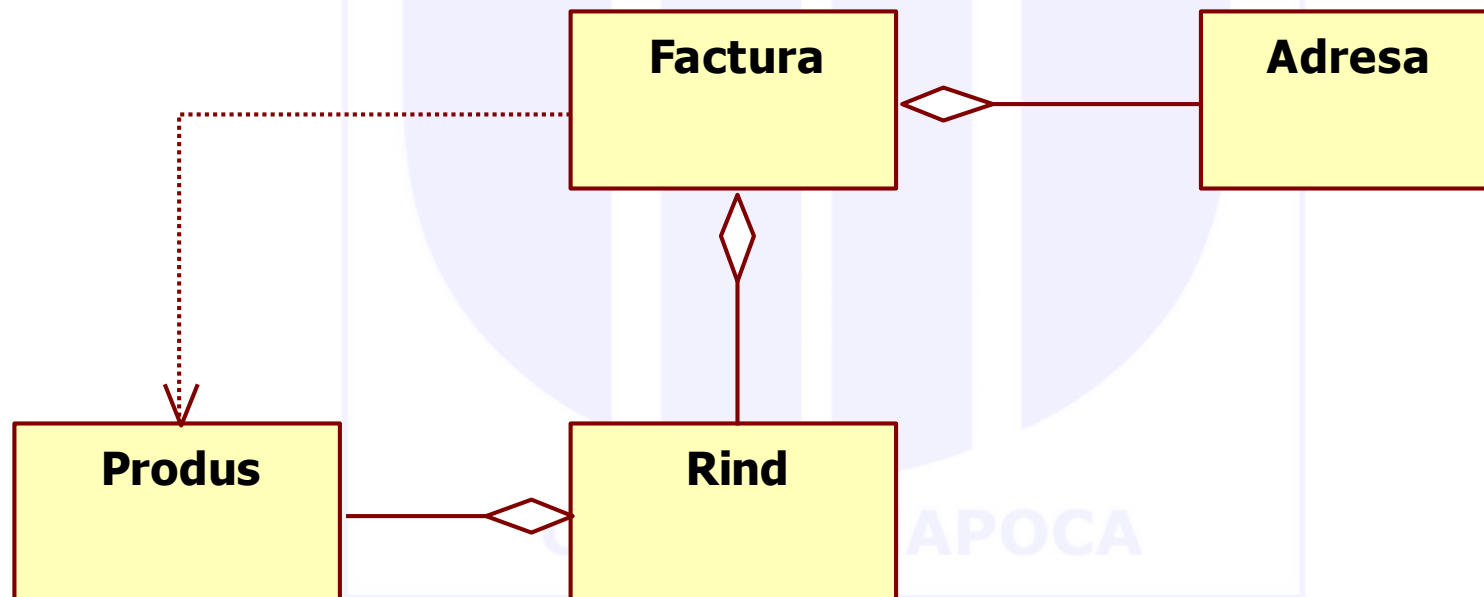
- **Factura** trebuie populată cu produse și cantități

Factura	
<i>formatează factura</i> <i>adaugă un produs și o cantitate</i>	Adresa Rind Produs

Computer Science



Tipărirea unei facturi – diagrama UML





Instrumente pentru realizarea diagramelor UML

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



Alte exemple de dezvoltare de aplicații OO

- 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/>