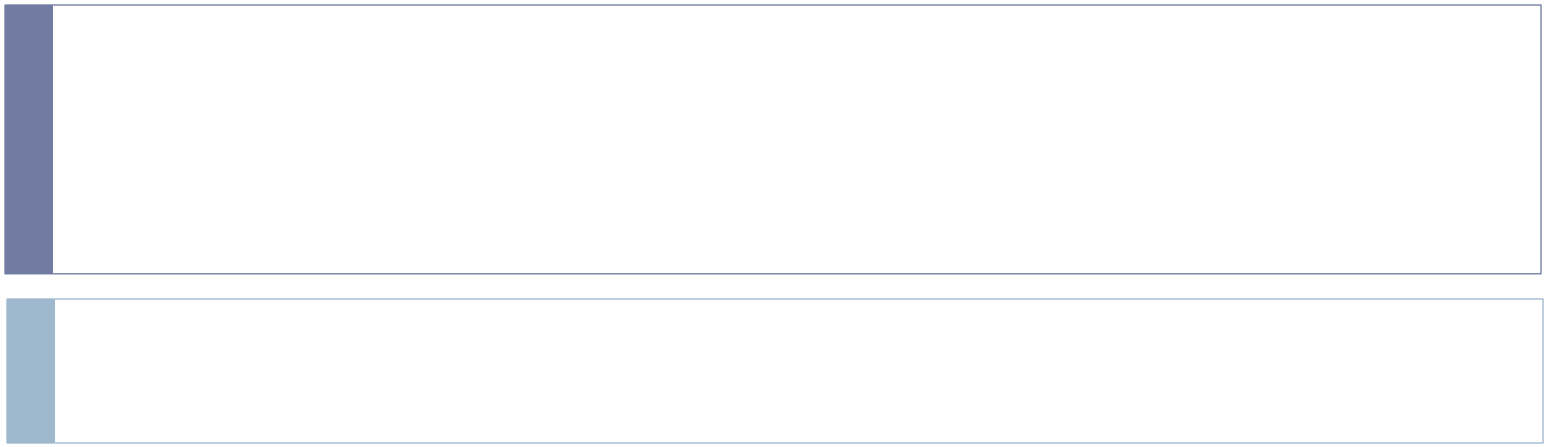
PROGRAMARE ORIENTATĂ PE OBIECTE

Curs 10

UML



Software academic şi industrial

- Exemplu: O problemă de 10000 LOC
  - Student: 2 luni (5000 LOC / lună)
  - Firmă: 1000 LOC/lună, 10 luni-om
  - Firma embedded systems: 100 LOC/lună
- Software academic
  - Versiune demo funcţională
  - Hobby: nu necesită documentaţie şi interfaţă complexă cu utilizatorul, defectele sunt corectate când apar
- Software industrial
  - Plătit de client
  - Influenţează mediul de afaceri
  - Necesită o abordare inginerească
- Toată lumea consideră importantă disciplina de lucru, dar nuexistă un acord general asupra modalităţilor de aplicare

Ingineria programării

- Ingineria programării reprezintă aplicarea unei abordări sistematice, disciplinate şi cuantificabile pentru dezvoltarea, operarea şi întreţinerea produselor software

  - *(Glosarul terminologiei ingineriei programării, IEEE, Institute of Electrical and Electronics Engineers, 1990)*

Software industrial

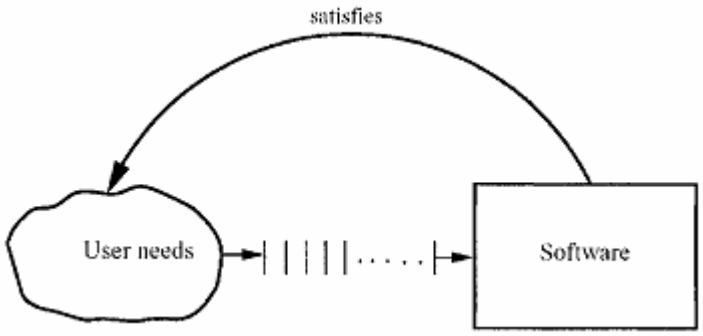
- Este construit pentru a rezolva unele probleme din organizația clientului
  - Funcţionarea incorectă poate provoca pierderi financiare şi chiar pierderea de vieţi omeneşti
- Trebuie să aibă calitate foarte bună
  - Testare riguroasă înainte de livrare (30%-50% din efortul total)
  - Dezvoltarea este împărţită pe faze pentru a corecta defectele din vreme (necesită documentaţie)
- Are cerinţe de recuperare a datelor, toleranţă la defecte, portabilitate
  - Acestea conduc la creșterea dimensiunilor

Regula lui Brooks

- Software-ul industrial în comparație cu software-ul academic
  - Productivitate: 1 / 5
  - Dimensiune dublă
- Software-ul industrial necesită de 10 ori mai mult efort decât software-ul academic

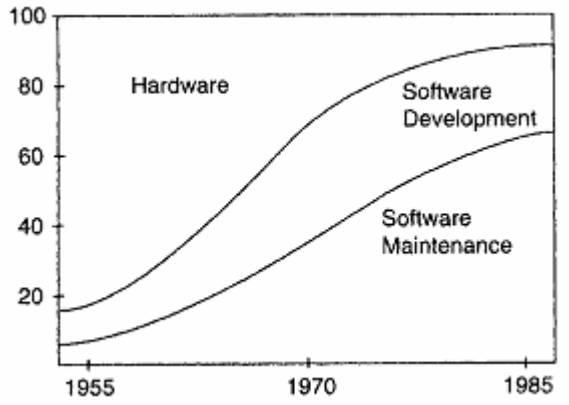
Scopul ingineriei programării

- Utilizarea unor metodologii pentru dezvoltarea de software
  - Rezultate repetabile
  - Apropiere de ştiinţă
  - Îndepărtarea de metodele ad-hoc cu rezultate imprevizibile
- Scopul dezvoltării de software este satisfacerea nevoilor clienţilor sau utilizatorilor



Costul produselor software

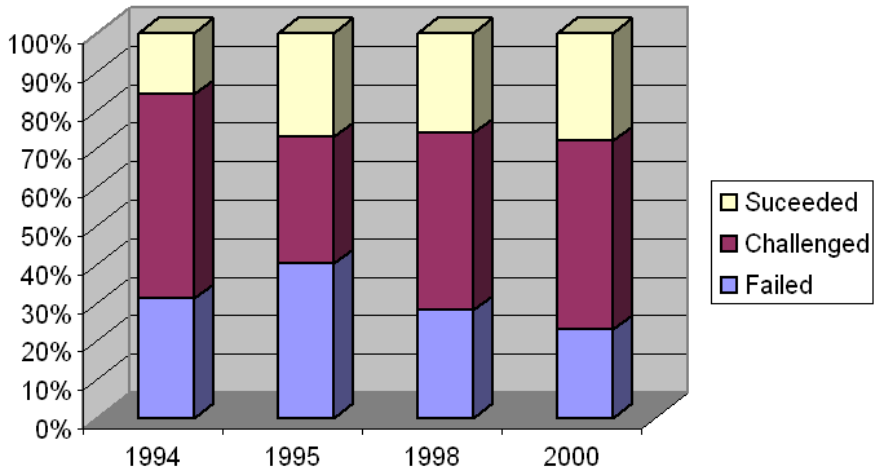
- Costul software-ului depinde în primul rând de efortul oamenilor
- Productivitatea este frecvent măsurată în linii de cod (LOC) / lună-om
- Productivitatea medie pentru o aplicaţie nouă este de 300-1000 LOC / lună-om
  - 8000 $ / lună - 8-25 $ / LOC
  - Un program mediu de 50.000 LOC poate costa aproximativ 1.000.000 $



Întârzieri şi instabilitate

- Din 600 de firme, 35% aveau proiecte informatice scăpate de sub control din punct de vedere al bugetului şi timpului de execuţie
- Raportul Standish Group privind finalizarea proiectelor IT în SUA

	Failed	Challenged	Succeeded
1994	31%	53%	16%
1995	40%	33%	27%
1998	28%	46%	26%
2000	23%	49%	28%



Lipsa de incredere

- Software care:
  - Nu face ce trebuie
  - Face ce nu trebuie
- În sisteme complexe (incluzând componente electrice, mecanice, hidraulice), de cele mai multe ori software-ul este problema cea mai mare
- Defectele software-ului nu se datorează uzurii,ci erorilor de proiectare şi implementare

Defecte „celebre”

- 28 iulie 1962 – sonda spațială Mariner I
- 1982 – conducta sovietică de gaz trans-siberiană
- 1983 – sistemul sovietic de avertizare nucleară
- 1985-1987 – acceleratorul medical Therac-25
- 1988-1996 – generatorul de numere aleatorii al protocolului Kerberos

Reprogramarea

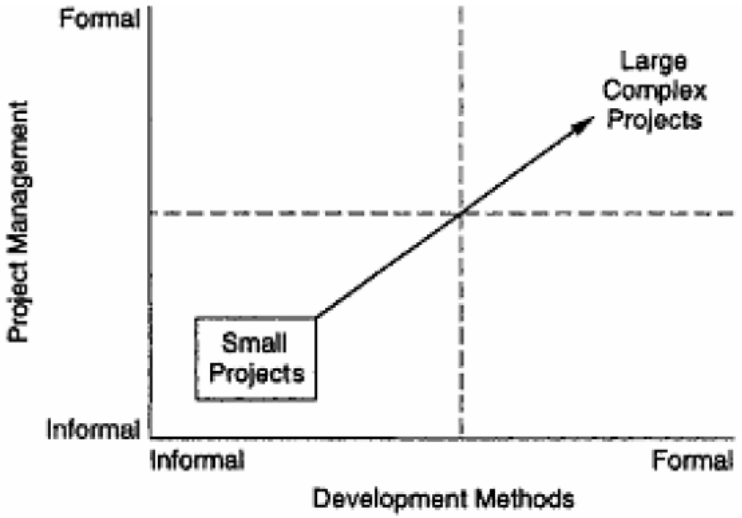
- Cerințele nu sunt specificate complet
  - Schimbarea lor conduce la refacerea tuturor fazelor ulterioare
- Pentru proiecte cu durată lungă, cerinţele clientului se modifică
- Reprogramarea consumă 30%-40% din efortul total de dezvoltare

Intretinerea

- Întreţinere corectivă
  - Eliminarea erorilor
- Întreţinere adaptivă
  - Includerea de funcţionalităţi suplimentare
  - Legea evoluţiei software-ului: software-ul schimbă mediul, apoi trebuie să se adapteze la noul mediu
- Întreţinerea costă de obicei mai mult decât dezvoltarea unei aplicaţii
  - Presupune înțelegerea codului, modificarea, testarea de regresiune
- În timpul dezvoltării, întreţinerea este deseori neglijată
- Raport de cost 60:40 – 80:20

Scala proiectelor

- Proiectele complexe necesită metode diferitede dezvoltare faţă de proiectele de mici dimensiuni
  - Presupun formalizarea procedurilor inginerești și a managementului de proiect
- Proiecte:
  - Mici: < 10 KLOC
  - Medii: 10-100 KLOC
  - Mari: 100-1000 KLOC
  - Foarte mari: peste 1 milion LOC



Scala proiectelor

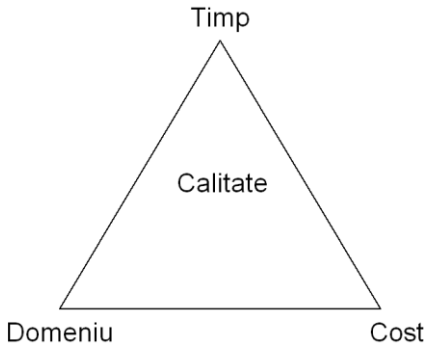
- Previziuni:
- 1946: Goldstein, von Neumann – max. 1000 instrucțiuni
- 1981: Bill Gates – max. 640 KB RAM

- Realitatea:
  - 1966: sistemul de operare IBMOS 360: 5000 de ani-om
  - 1977: naveta spaţială NASA:cc.40 milioane LOC
  - 1983: System V versiunea 4.0Unix: 3,7 milioane LOC
  - 1992: sistemul de rezervare abiletelor KLM: 2 milioane LOCîn limbaj de asamblare

Size (KLOC)	Software	Languages
980	gcc	ansic, cpp, yacc
320	perl	perl, ansic, sh
305	teTeX	ansic, perl
200	openssl	ansic, cpp, perl
200	Python	python, ansic
100	apache	ansic, sh
90	cvs	ansic, sh
65	sendmail	ansic
60	xfig	ansic
45	gnuplot	ansic, lisp
38	openssh	ansic
30,000	Red Hat Linux	ansic, cpp
40,000	Windows XP	ansic, cpp

Triunghiul managementului de proiect

- Ingineria programării este condusă de 3 factori majori: costul, timpul şi domeniul de aplicare (întinderea, anvergura)



- **Costul este o măsură a resurselor utilizate pentru sistem**
  - În cazul software-ului este dominat de costul de personal
  - Costul poate fi estimat ca efort (luni-om) * cost mediu (lunar)
  - Include costul suplimentar pentru hardware și instrumentele de dezvoltare
- **Timpul**
  - Mediul de afaceri dorește reducerea timpului de livrare
  - Creşterea productivităţii (KLOC / lună-om) determină scăderea timpului şi a costului
- **Domeniul (engl. "scope") este dat de specificații**

Calitatea

- **Calitatea presupune, conform standardului internaţional al calităţii produselor software:**
  - **Funcționalitate (functionality)**
    - Asigurarea funcțiilor care satisfac nevoile exprimate explicit sau implicite
    - Include *securitatea: persoanele neautorizate să nu aibă acces iar celor autorizate să nu le fie refuzat accesul*
  - **Încredere (reliability)**
    - Menţinerea unui nivel specificat de performanţă
  - **Utilizabilitate (usability)**
    - Capacitatea de a fi înţeles, învăţat şi utilizat
  - **Eficienţă (efficiency)**
    - Asigurarea unor performanțe adecvate relativ la volumul de resurse utilizate
  - **Mentenabilitate (maintainability)**
    - Capacitatea de a fi modificat pentru corecţii, îmbunătăţiri sau adaptări
  - **Portabilitate (portability)**
    - Capacitatea de a fi adaptat pentru medii diferite exclusiv pe baza mijloacelor existente înprodus

Calitatea

- Importanţa fiecărei dimensiuni depinde de natura proiectului
  - Sistem critic: încredere
  - Joc: utilizabilitate
- Înainte de dezvoltare, trebuie specificat obiectivul principal de calitate
- Încrederea este considerată în general cea mai importantă
  - Se măsoară în defecte / KLOC
  - Bunele practici curente: mai puțin de 1 defect / KLOC
  - Definirea unui defect depinde de proiect sau de standardele organizației dezvoltatoare

Consecvenţă şi repetabilitate

- Succesele trebuie să fie repetabile
  - Calitatea şi productivitatea trebuie să fie consecvente
- Acest lucru permite unei organizații:
  - Să prevadă cu acurateţe rezultatele proiectelor
  - Să îşi îmbunătăţească procesele de dezvoltare
- Se impune standardizarea unor proceduri și folosirea unor metodologii

Concluzii

- Ingineria programării este o colecţie de metode şi recomandări pentru dezvoltarea eficientă de programe de mari dimensiuni
- Software-ul nu este doar o mulțime de programe, ci include documentația și datele asociate
- Fazele fundamentale ale dezvoltării programelor sunt:analiza, proiectarea, implementarea şi testarea
- Istoria sistemelor de calcul şi a programării se întinde peaproape 70 de ani iar dezvoltarea prezentă a domeniuluieste fără precedent

Limbajul unificat de modelare UML

- 1.Modelarea
- 2.Limbajul unificat de modelare
- 3.Clasificarea diagramelor UML 2.0
- 4.Diagramele UML 2.0
- 5.Concluzii

Modelarea

- Un model este o simplificare a unui anumit sistem, care permite analizarea unora dintre proprietăţile acestuia
  - Reține caracteristicile necesare
- Folosirea de modele poate înlesni abordarea problemelor complexe, facilitând comunicarea și înțelegerea
  - Divide et impera
- Exemple:
  - Formalismul matematic
  - Reprezentările din fizică
- Orice limbaj „intern” poate fi folosit pentru modelare,însă într-un context formal este nevoie de standardizare

Limbajul unificat de modelare, UML

- Limbaj pentru specificarea, vizualizarea,construirea și documentarea elementelor sistemelor software
  - Un limbaj grafic care ne permite să reproducem „pe hârtie” ceea ce este produs în procesul dedezvoltare a unui sistem software
  - Poate fi folosit și pentru alte sisteme, cum ar fi procesele de afaceri (business processes)

Versiuni şi standardizare

- Ianuarie 1997: UML 1.0 a fost propus spre standardizare în cadrul OMG (Object Management Group)
- Noiembrie 1997: versiunea UML 1.1 a fost adoptată castandard de către OMG
- Martie 2003: a fost lansată versiunea 1.5
- Octombrie 2004: versiunea 2.0
- August 2011: versiunea 2.4.1
- UML este standardul ISO/IEC 19501:2005

UML

- Ca orice limbaj, UML are:
  - Notații (alfabetul de simboluri)
  - Sintaxă şi gramatică (reguli pentru combinarea simbolurilor)
- UML este un instrument de comunicare
- UML nu este o metodologie de dezvoltare
  - Dar este determinat de cele mai bune practici îndomeniu

Clase de diagrame

- **Diagrame de structură**
  - Prezintă elementele unei specificaţii independent de timp
  - Includ: diagramele de clase, structuri compuse, componente,desfăşurare (deployment), obiecte şi pachete
- **Diagrame de comportament**
  - Prezintă trăsăturile comportamentale ale sistemului
  - Includ: diagramele de activităţi, maşini de stare şi cazuri deutilizare, precum şi cele 4 diagrame de interacţiune
  - **Diagrame de interacțiune**
    - Scot în evidenţă interacţiunile dintre obiecte
    - Includ: diagramele de secvențe, comunicare, interacțiuni generale(interaction overview) și cronometrare (timing)

Diagrame de structura

- Ce contine sistemul:
  - Clase
  - *Structuri compuse*
  - Componente
  - Desfăşurare
  - Obiecte
  - *Pachete*

Diagrame de comportament

- Ce se intampla in sistem
  - Activităţi
  - Mașini de stare
  - Cazuri de utilizare

Diagrame de interactiune

- Fluxurile de control și date dintre componentele sistemului
  - Secvențe
  - Comunicare
  - *Interacțiuni generale*
  - *Cronometrare*

Diagrame UML 2.0

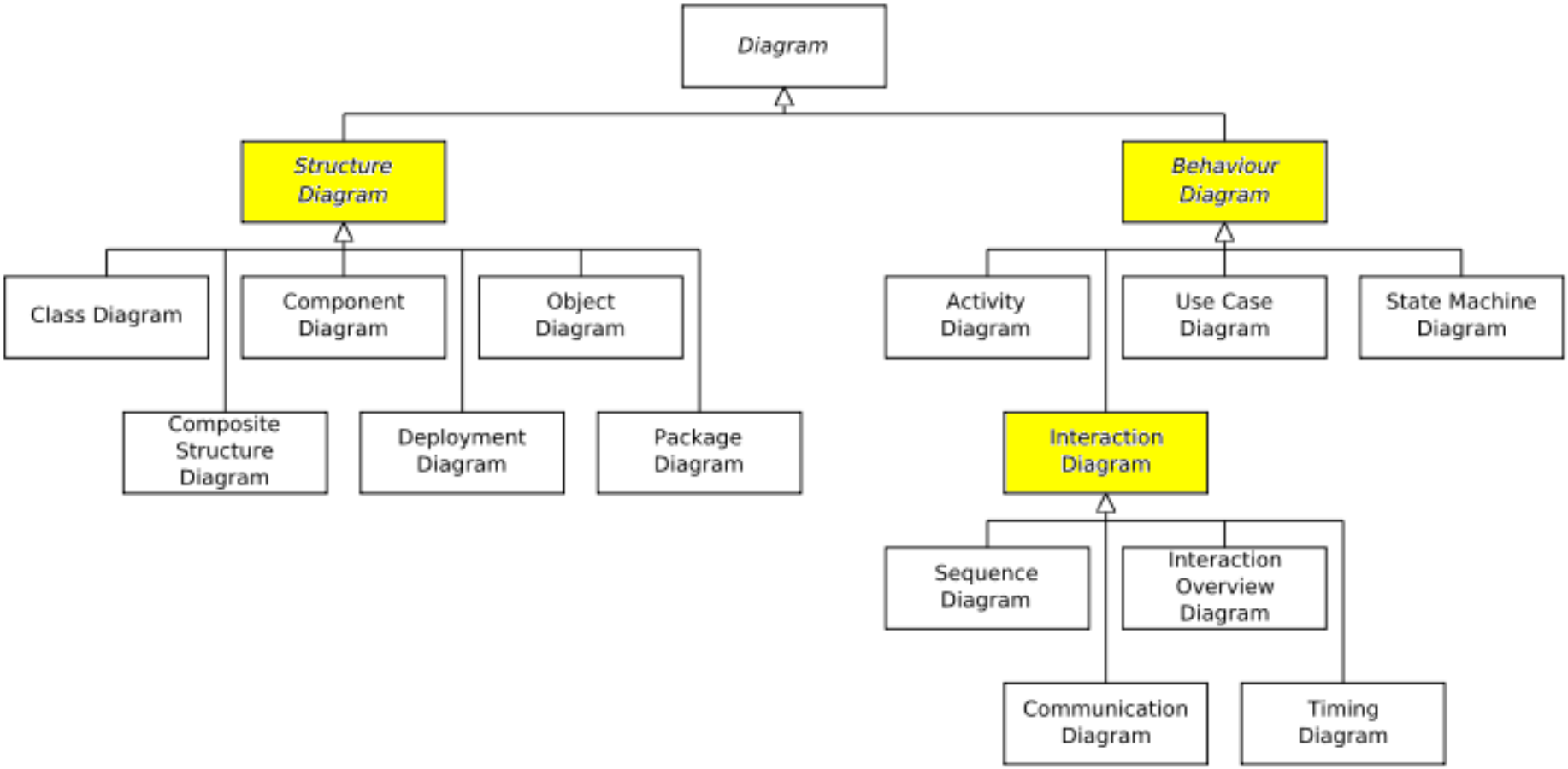
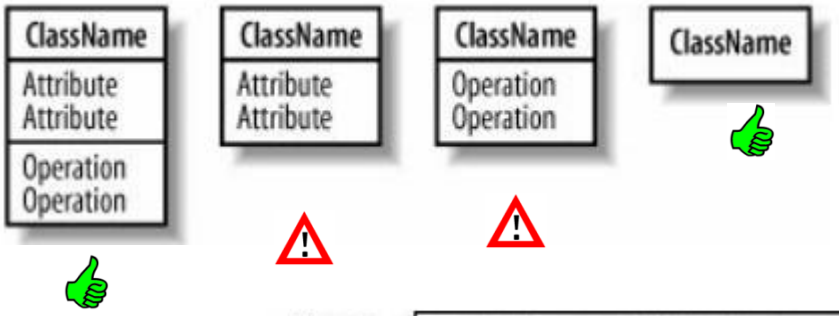
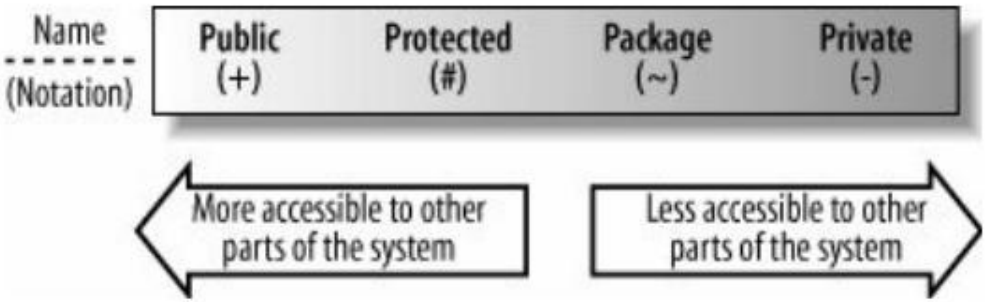


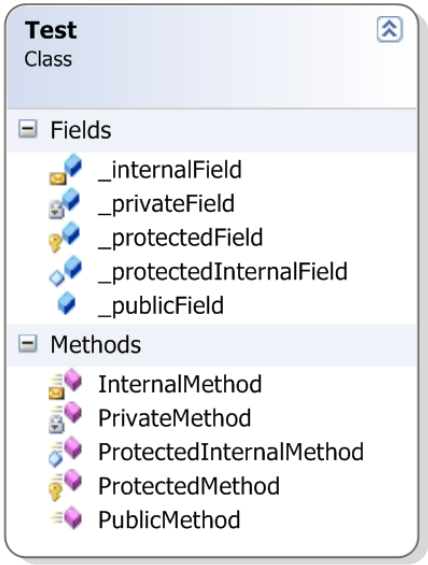
Diagrama de clase

- Clasa



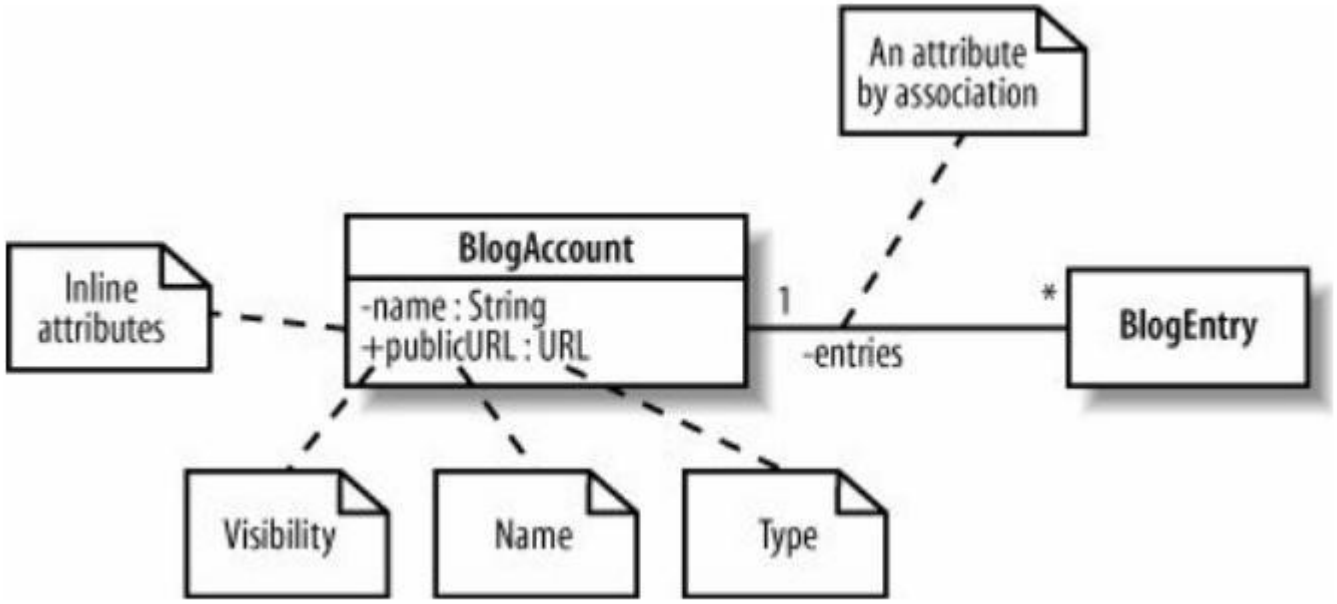
- Vizibilitatea trasaturilor
  - Public
  - Protejat
  - Pachet
  - Privat



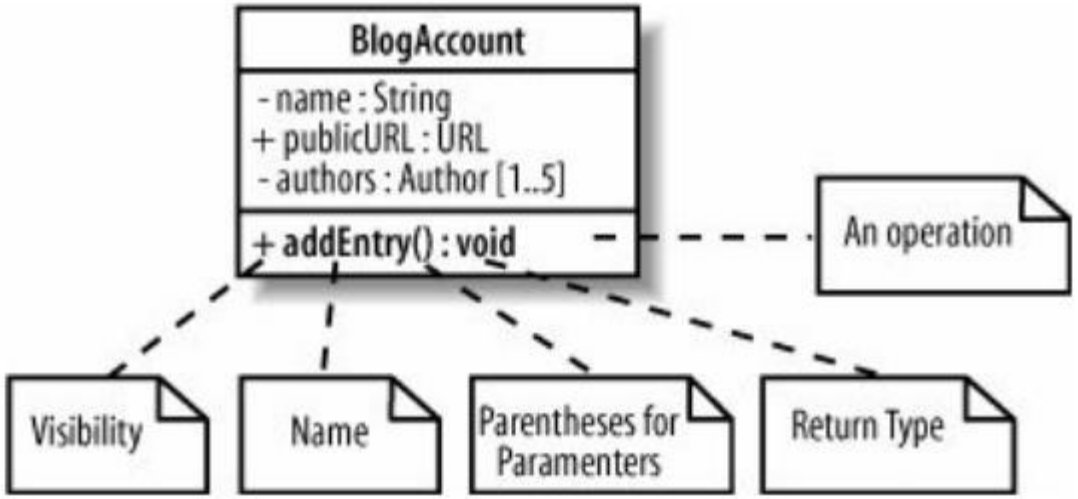


Reprezentarea atributelor

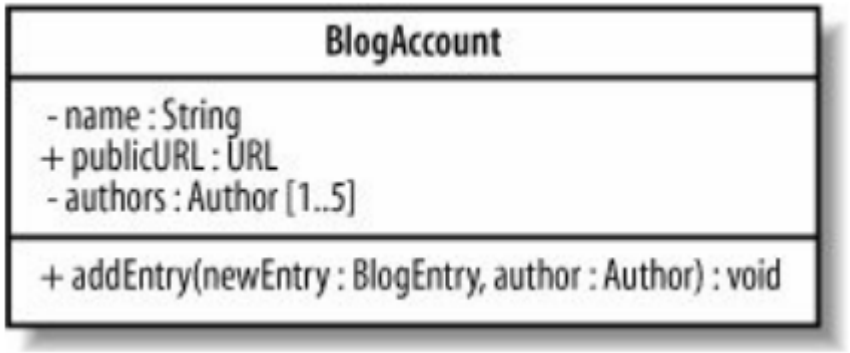
- In interorul clasei (inline)
- Prin asociere

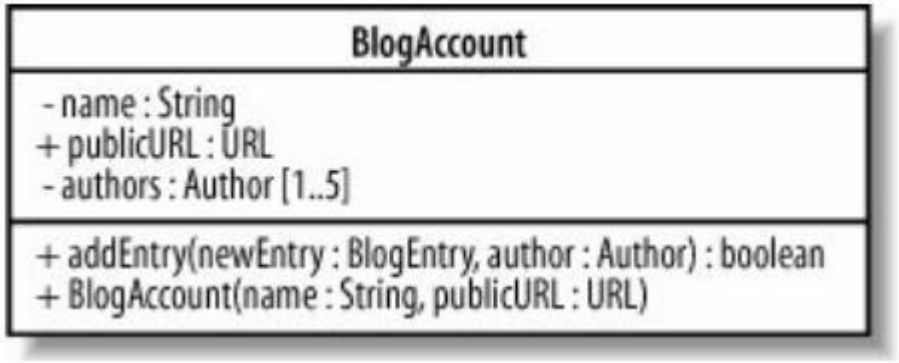


Operatii


BlogAccount
- name : String
+ publicURL : URL
- authors : Author [1..5]
+ addEntry() : void
An operation
Visibility
Name
Parentheses for Paramenters
Return Type

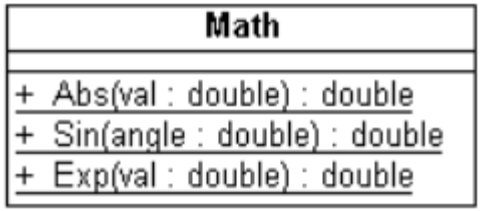
Parametri sau tipurile de return



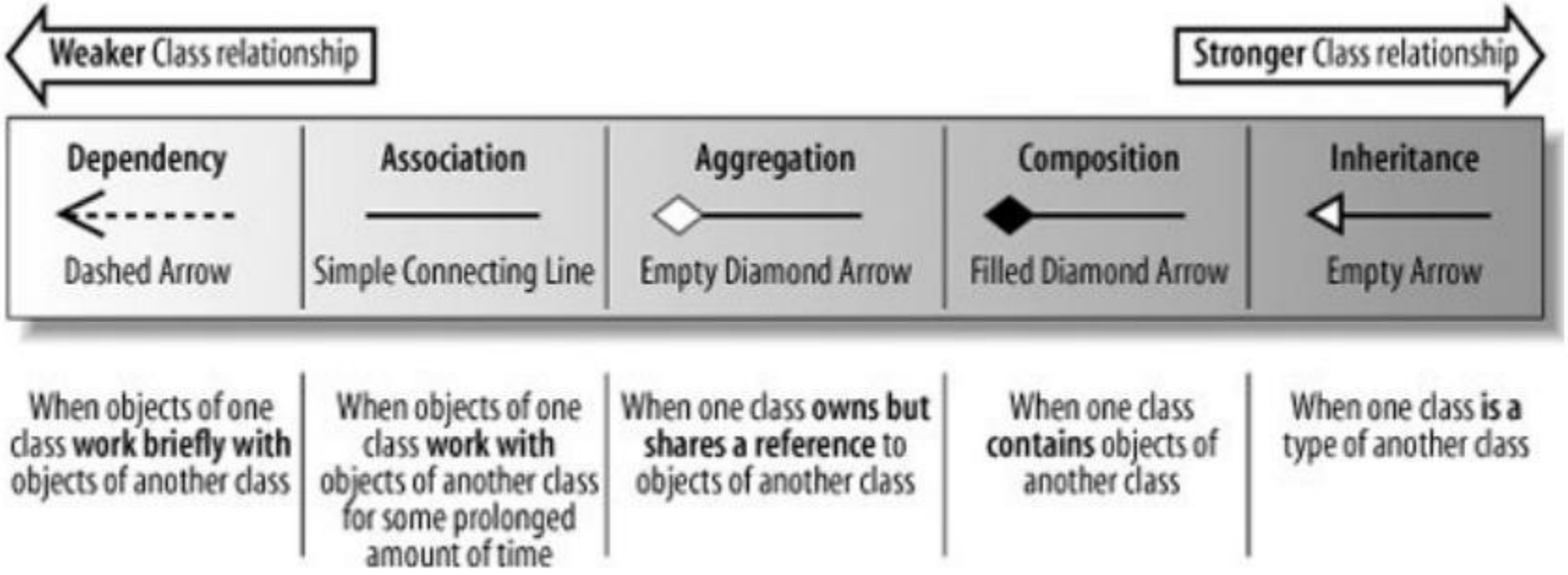


Trasaturi statice

- Trăsături (*features*) = *atribute și operații*
- Trăsăturile statice se subliniază

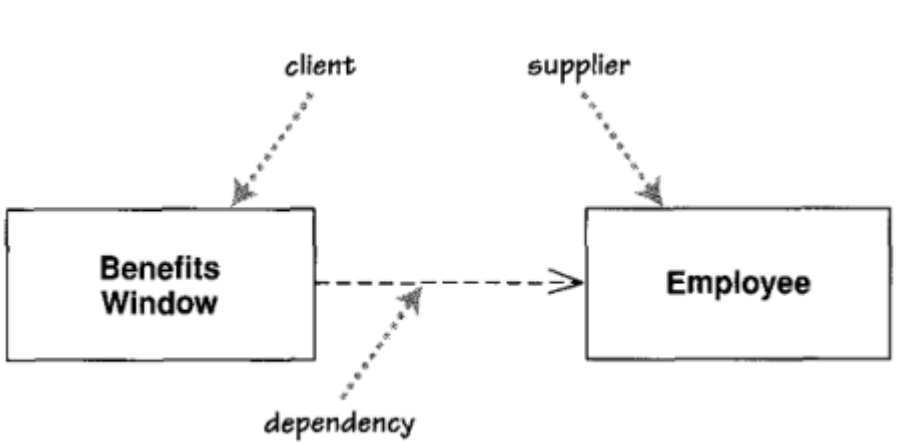


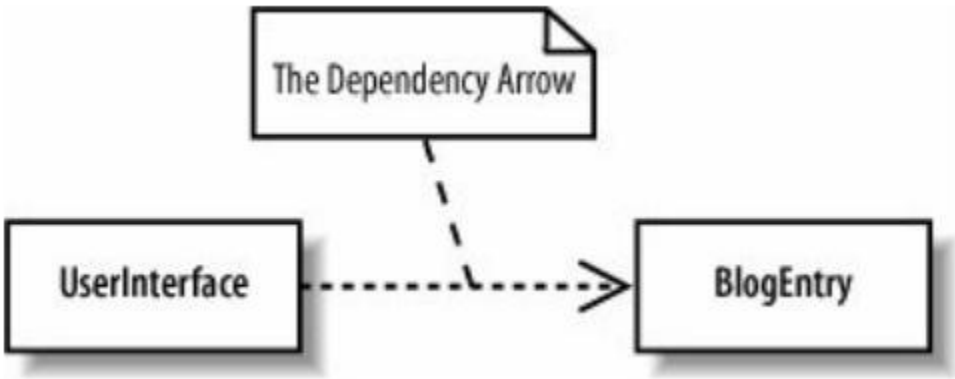
Relatii intre clase



Dependenta

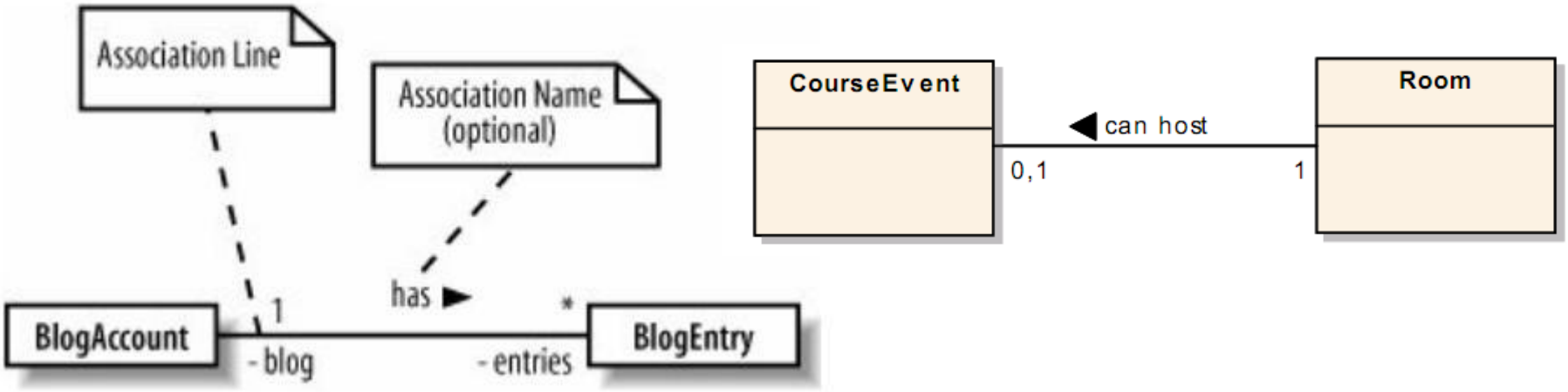
- O clasă folosește pentru scurt timp o altă clasă
  - Exemplu: trimiterea unui mesaj - metodele clasei Math
- Din punct de vedere al implementării:
  - Instanţierea unei clase într-o metodă
  - Primirea unui obiect ca parametru într-o metodă
  - Crearea și returnarea unui obiect dintr-o metodă





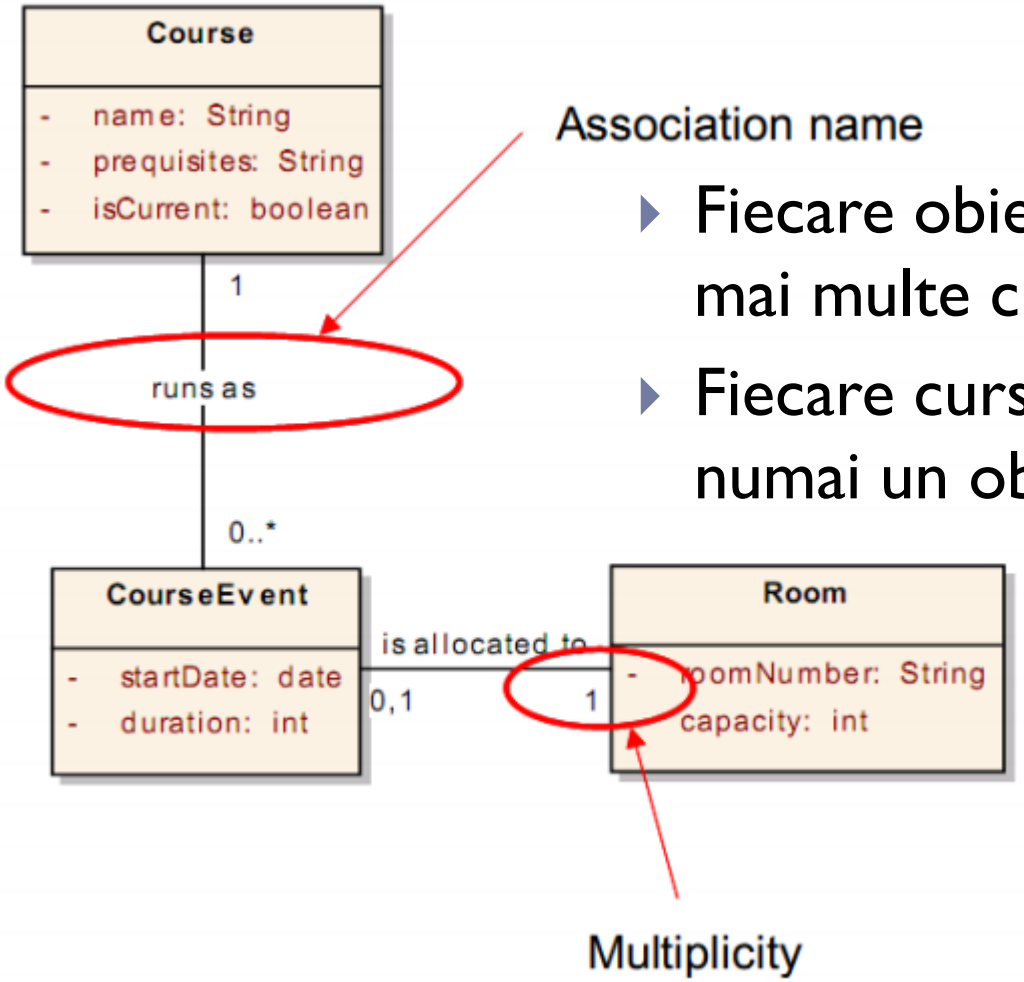
Asocierea

- O clasă foloseşte un timp îndelungat o altă clasă
- De obicei, o clasă are un câmp instanțiat din cealaltă clasă



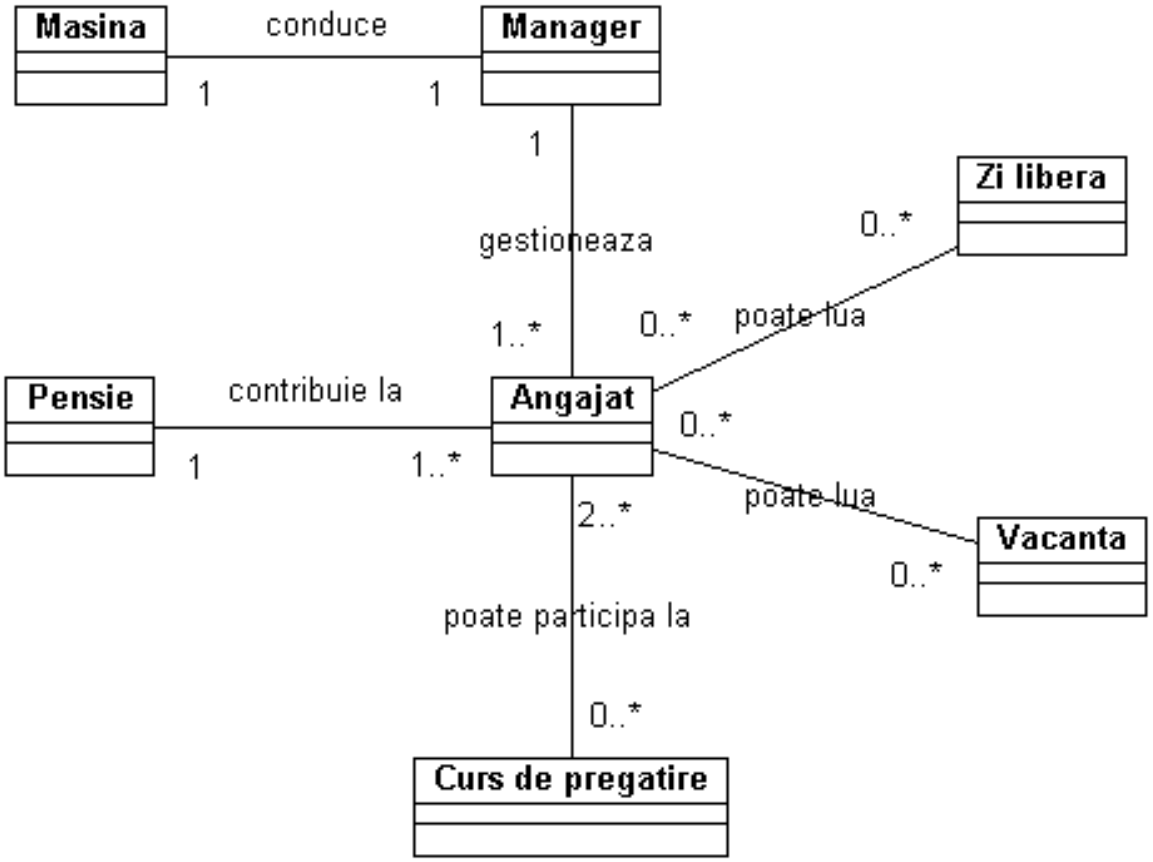
- Direcția de citire este de obicei de la stânga la dreapta și de sus în jos
- Direcția de citire se poate indica explicit
  - Săgeata care indică direcţia de citire nu trebuie pusăpe linia de asociere!

Validarea asocierilor



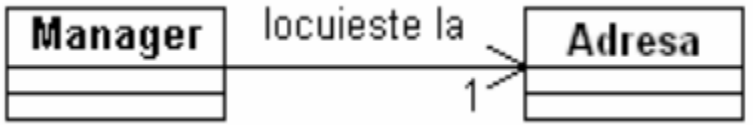
- Fiecare obiect este predat ca 0 sau mai multe cursuri.
- Fiecare curs este pentru unul și numai un obiect.

Asociere complexă

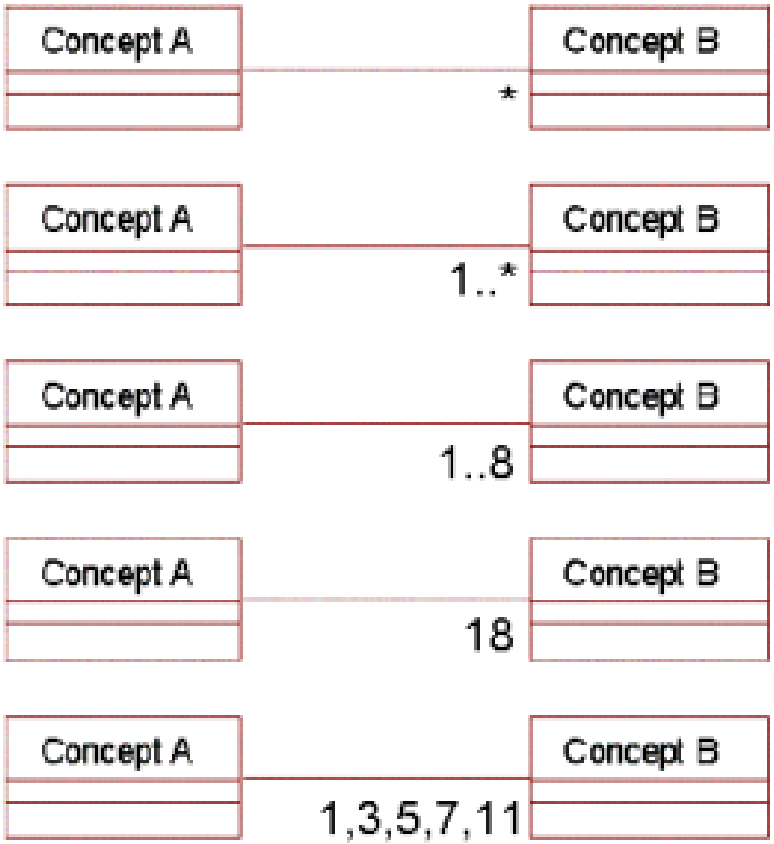


Asociere unidirectionala

- Numai o clasă „știe” de cealaltă



Multiplicitatea asocierii



Oricât de multe

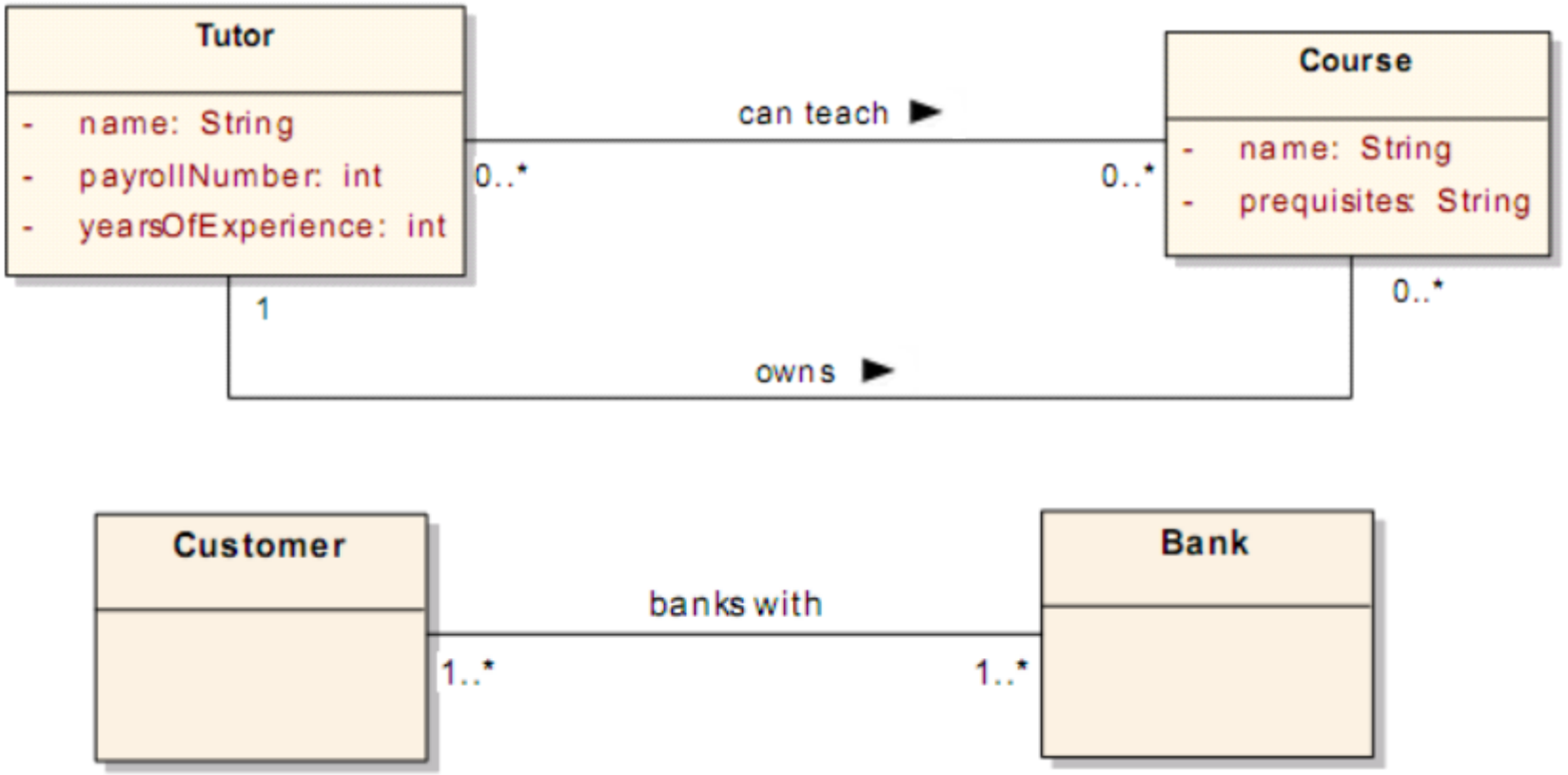
Una sau mai multe

Între 1 şi 8

Exact 18

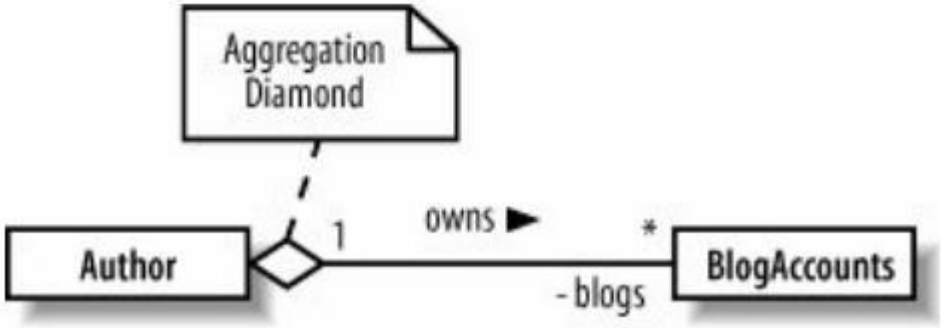
Mulțime specificată

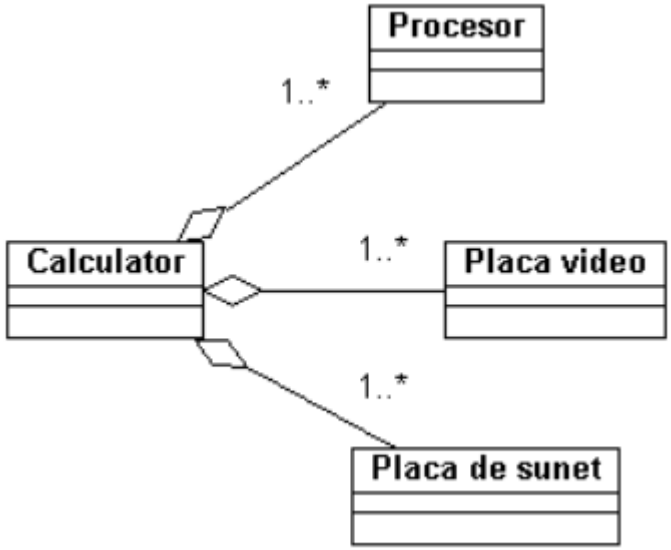
Asocieri multiple



Agregarea

- O clasă are dar *partajează obiecte dincealaltă clasă*



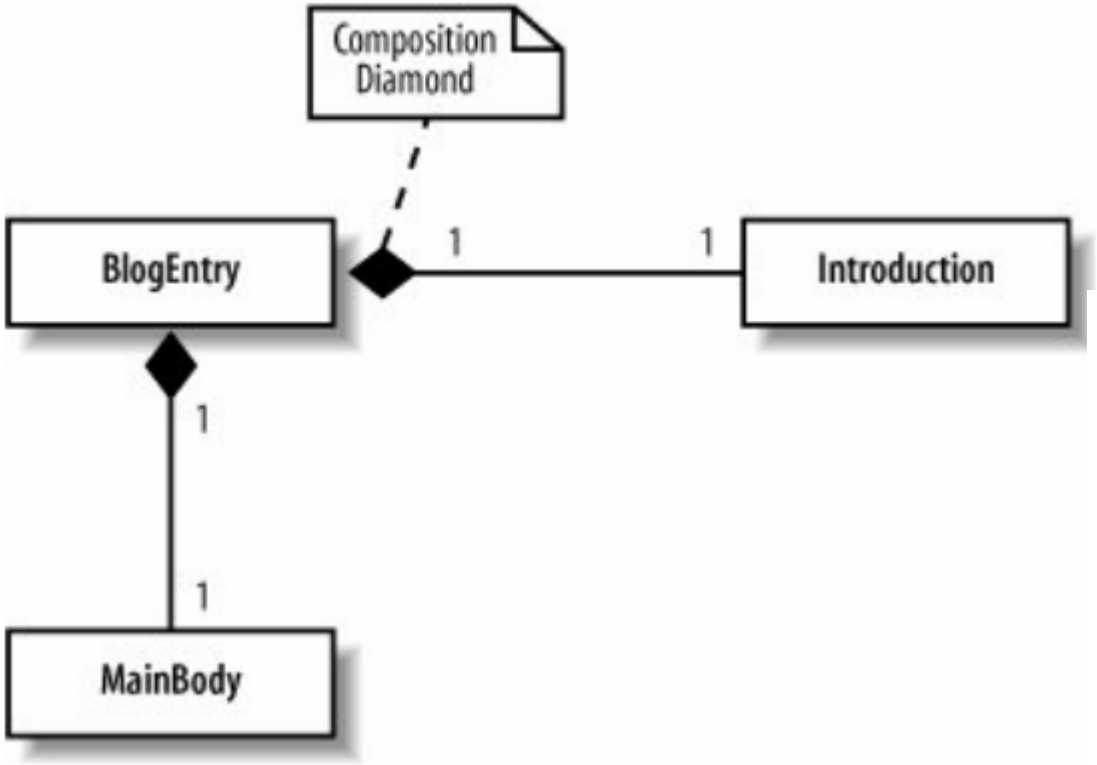


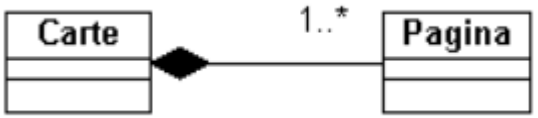
Agregarea

```
class Program
{
    static void Main(string[] args)
    {
        Procesor procesor = new Procesor();
        PlacaVideo placaVideo = new PlacaVideo();
        PlacaDeSunet placaDeSunet = new PlacaDeSunet();
        Calculator calculator1 = new Calculator(procesor, placaVideo, placaDeSunet);
        Calculator calculator2 = new Calculator(procesor, placaVideo, placaDeSunet);
    }
}

public class Calculator
{
    private Procesor _procesor;
    private PlacaVideo _placaVideo;
    private PlacaDeSunet _placaDeSunet;

    public Calculator(Procesor procesor, PlacaVideo placaVideo, PlacaDeSunet placaDeSunet)
    {
        _procesor = procesor;
        _placaVideo = placaVideo;
        _placaDeSunet = placaDeSunet;
    }
}
```

Compunerea

- Atributele compun clasa

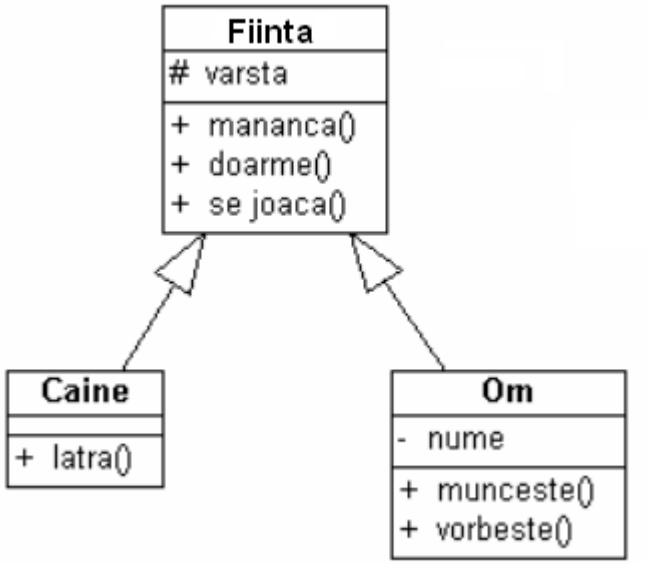


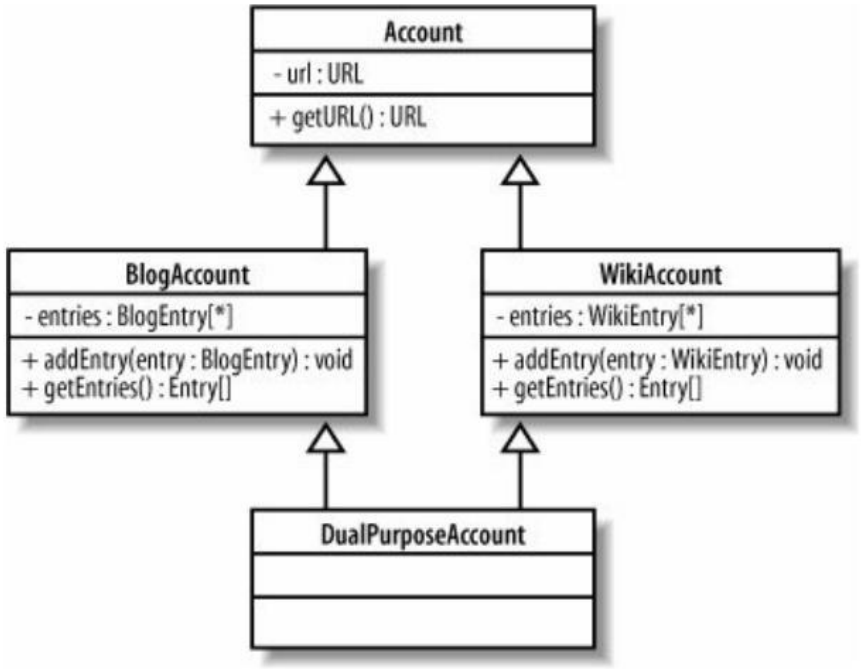



```
public class Carte
{
    private List<Pagina> _pagini;

    public Carte()
    {
        _pagini = new List<Pagina>();
    }
}

public class Pagina
{

}
```

Mostenirea

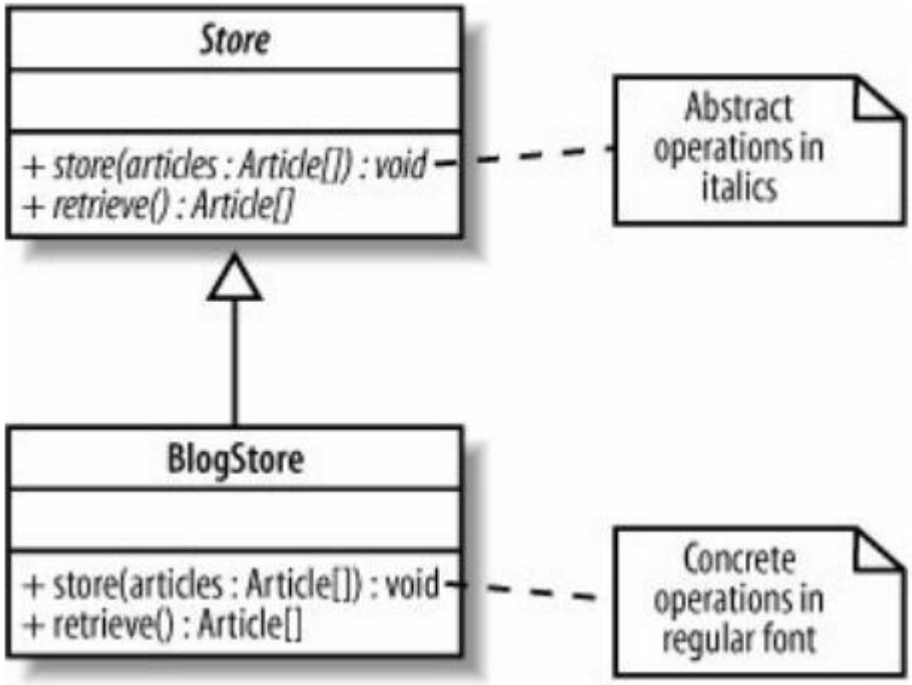
- Este o relatie de tip ESTE-UN / ESTE-O

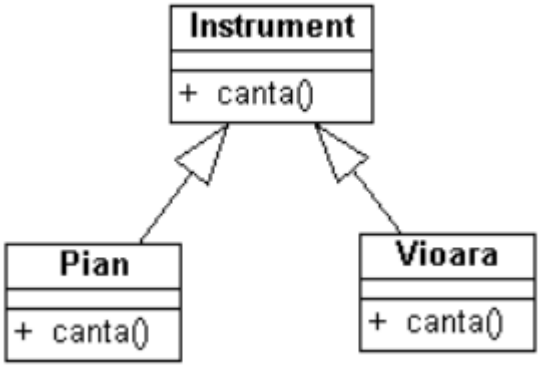


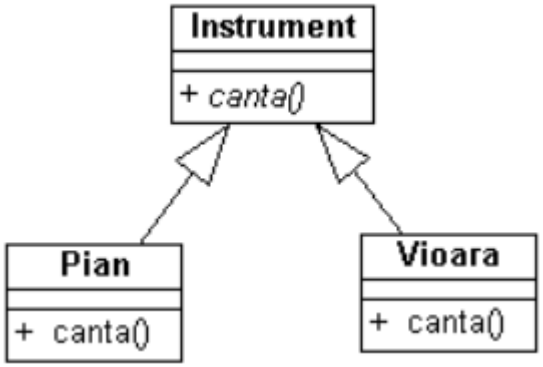


- Compunerea ar trebui preferată moştenirii
  - Moștenirea este cea mai puternică formă decuplare
  - În general, compunerea este mai ușor degestionat

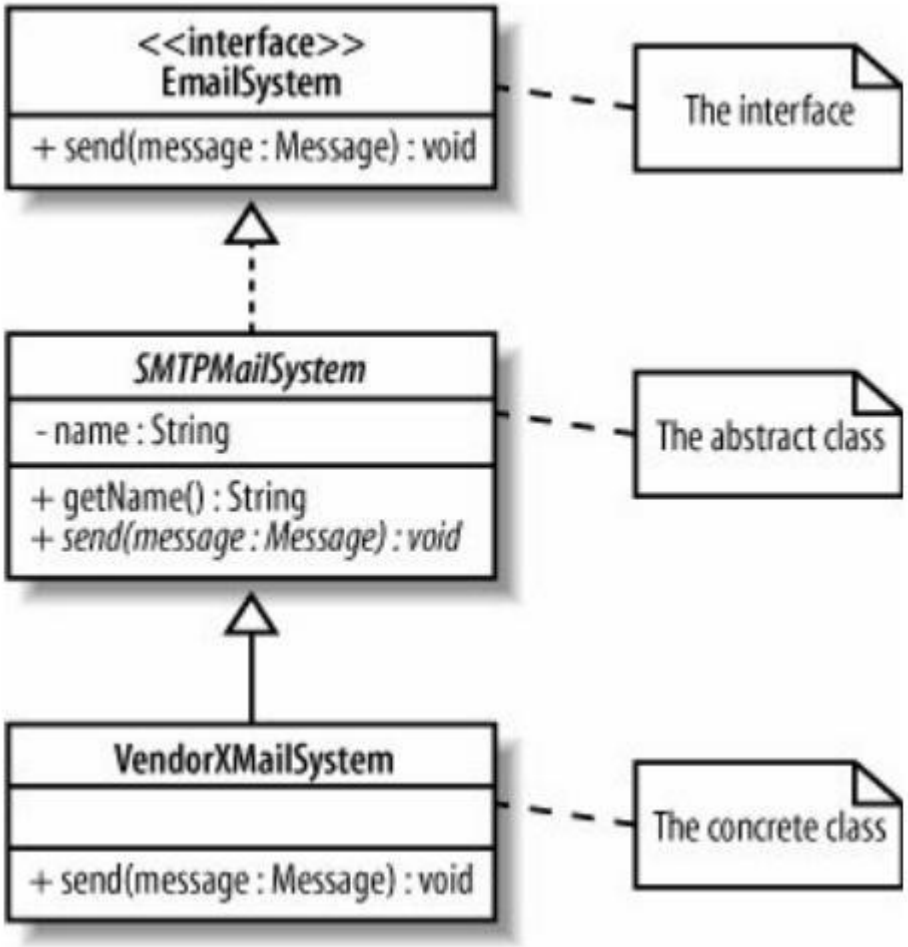
Clase şi operaţii abstracte

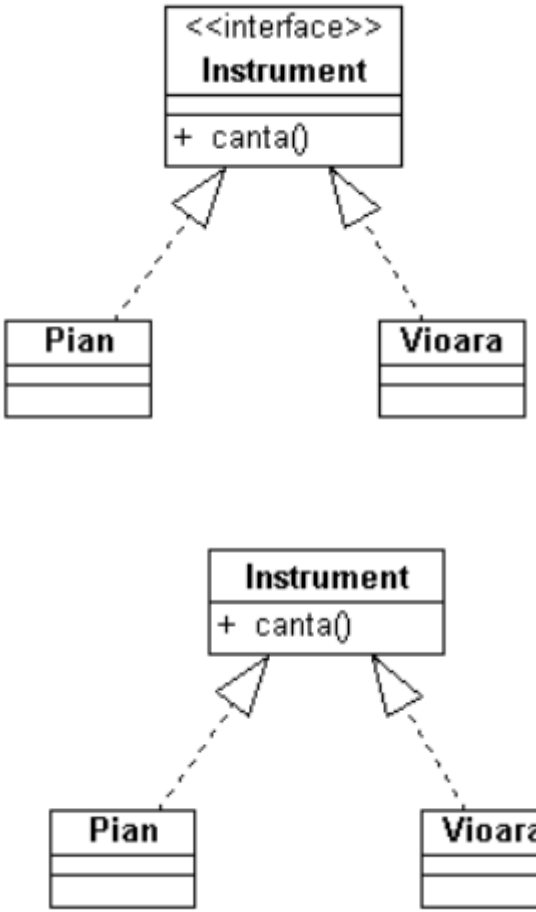






Interfete


<<interface>>
EmailSystem
+ send(message : Message) : void
The interface
SMTPMailSystem
- name : String
+ getName() : String
+ send(message : Message) : void
The abstract class
VendorXMailSystem
+ send(message : Message) : void
The concrete class


<<interface>>
Instrument
+ canta()
Pian
Vioara
Instrument
+ canta()
Pian
Vioara

Template-uri