M. Caramihai, © 2022

PROGRAMAREA ORIENTATA OBIECT

CURS 4

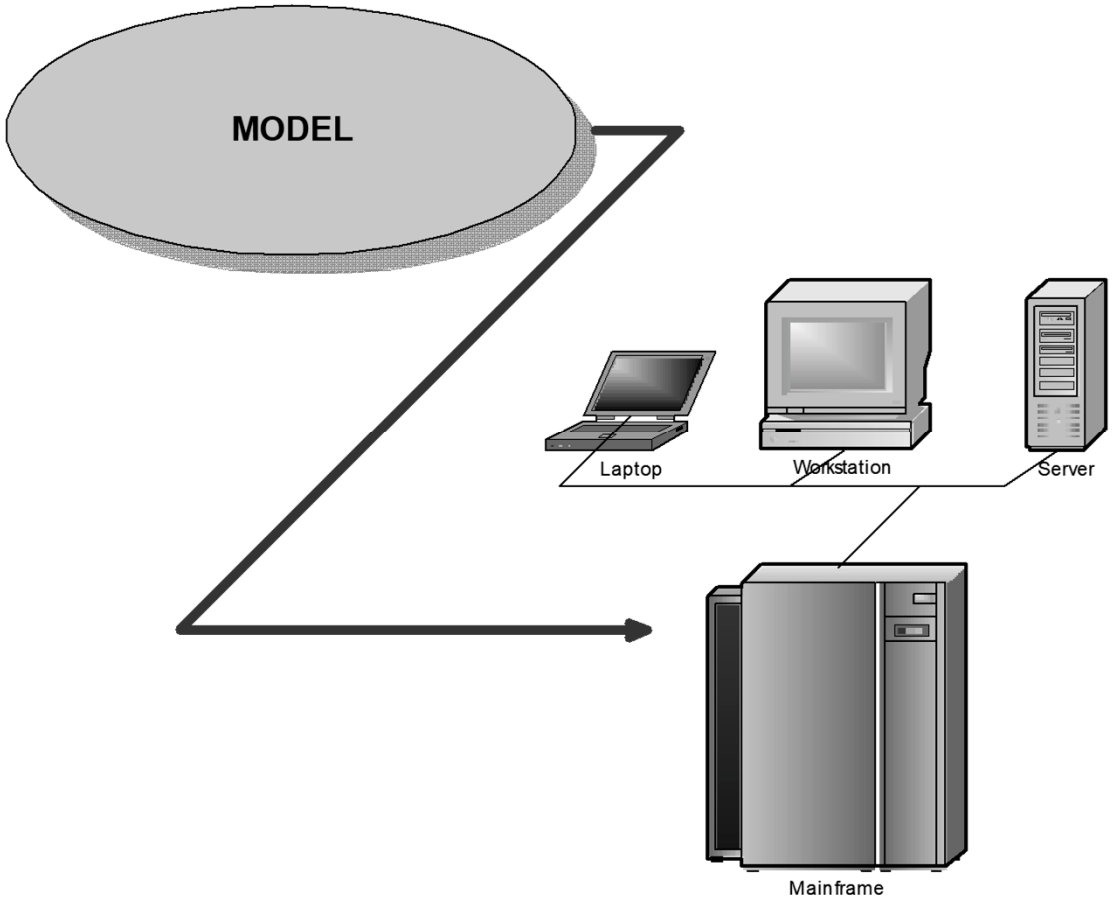
Introducere in UML (1)

Ce este modelarea ?

- Aspecte generale privind modelarea
- Trasaturile modelarii:
  - Simplificare (rezultatul abstractizarii)
  - Subordonare unui scop
  - Reprezentarea unei realitati
  - Divizare
  - Ierarhizare
  - Comunicare
- Grupuri tinta: client/utilizatori si membrii echipa proiect

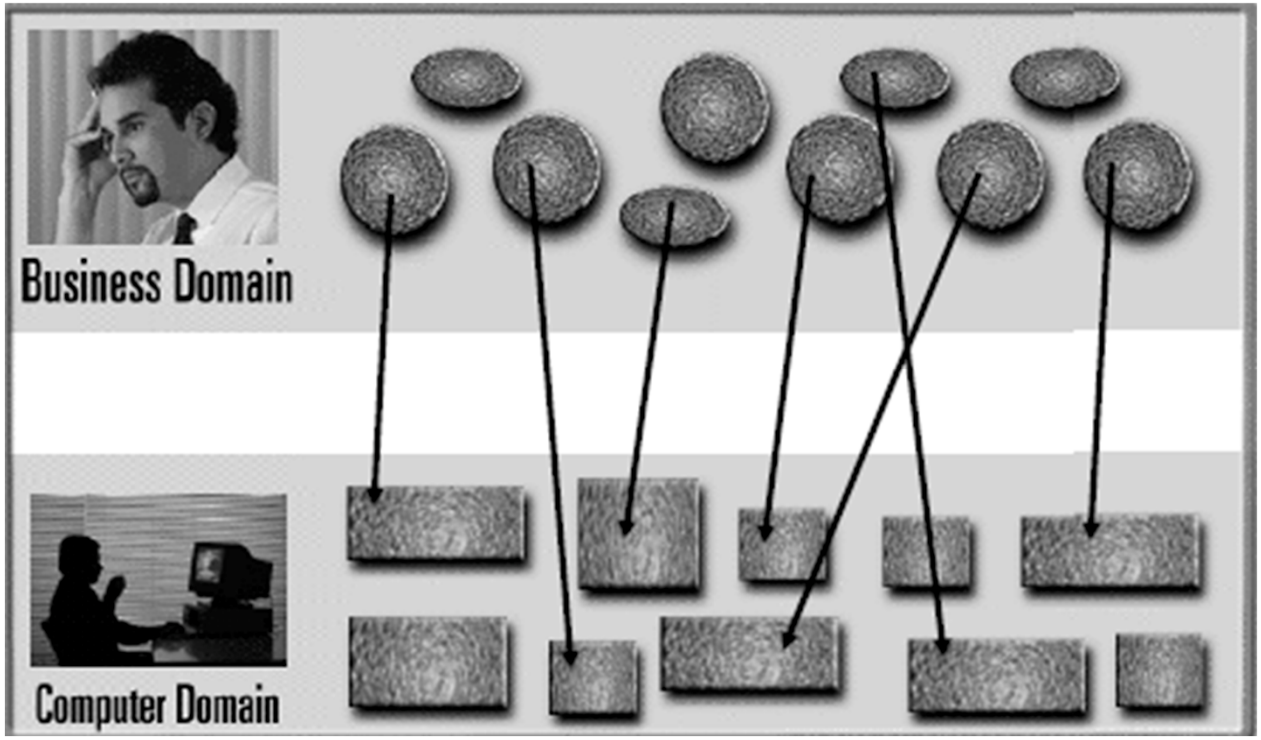
Ce este modelarea vizuala (1) ?

- Modelarea vizuala permite modelarea prin notatii grafice standard
- Principii:
  - Modul de creare tine de scopul utilizarii modelului
  - Orice model poate fi conceput la diferite niv de abstractizare
  - Orice sistem real poate fi reprezentat printr'o suita de modele



Ce este modelarea vizuala (2) ?

Prin modelarea vizuala se pot abstractiza
obiectele & logica proceselor de afaceri



Prin modelarea vizuala poata fi
analizata & proiectata o aplicatie (informatica)

Ce este UML ?

UML este prescurtarea de la **Unified Modeling Language**

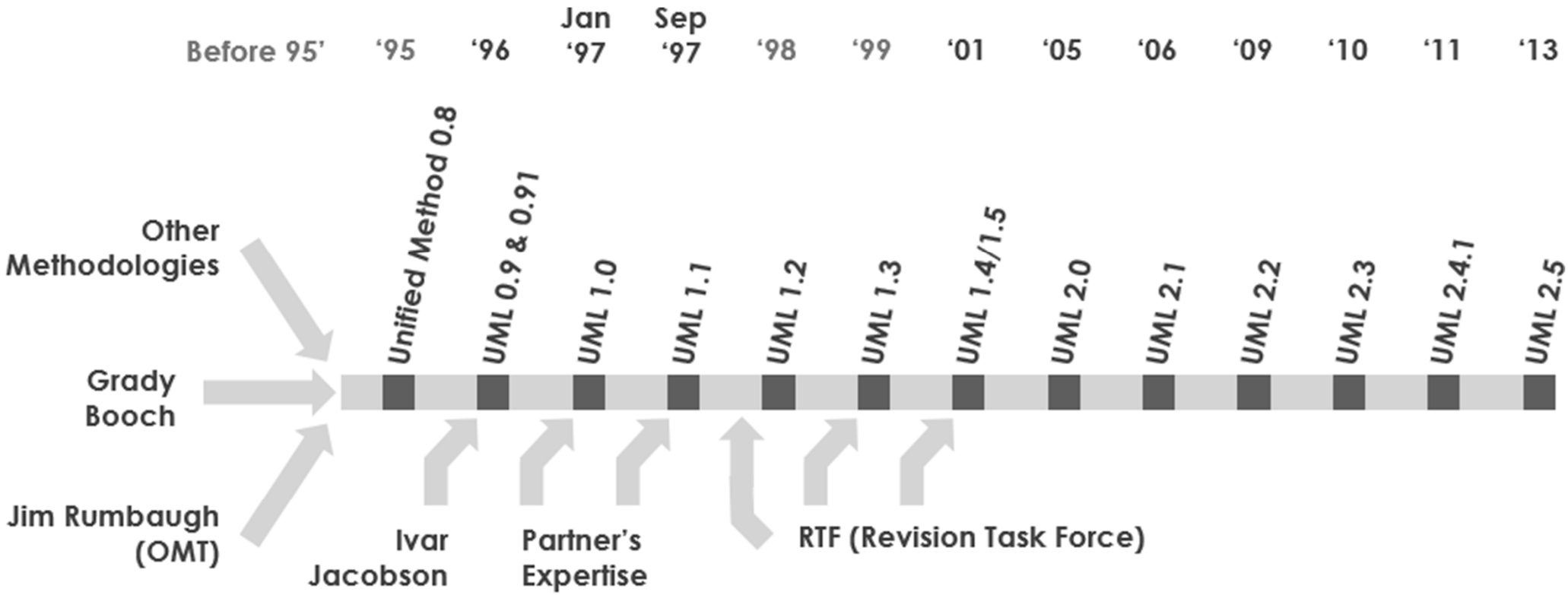
- **Definitie**: limbaj de modelare pentru specificarea, vizuualizarea, constructia si documentarea componentelor unei aplicatii informatice

- UML reprezinta o sinteza a:
  - Conceptele *Data Modeling* (*Entity Relationship Diagrams*)
  - *Business Modeling* (*work flow*)
  - Modelarea obiectelor
  - Modelarea componentelor

- UML este limbajul standard "for visualizing, specifying, constructing, and documenting the artifacts of a software intensive system" [The UML definition was led by Grady Booch, Ivar Jacobson, and Jim Rumbaugh], http://www.omg.org

- *Focus* pe elementele **conceptuale** si **fizice** ale reprezentarii unui sistem.

Istoria dezvoltarii UML



Before 95' - Fragmentation ▶ 95' - Unification ▶ 98' - Standardization ▶ 99' - Industrialization

Conceptele UML

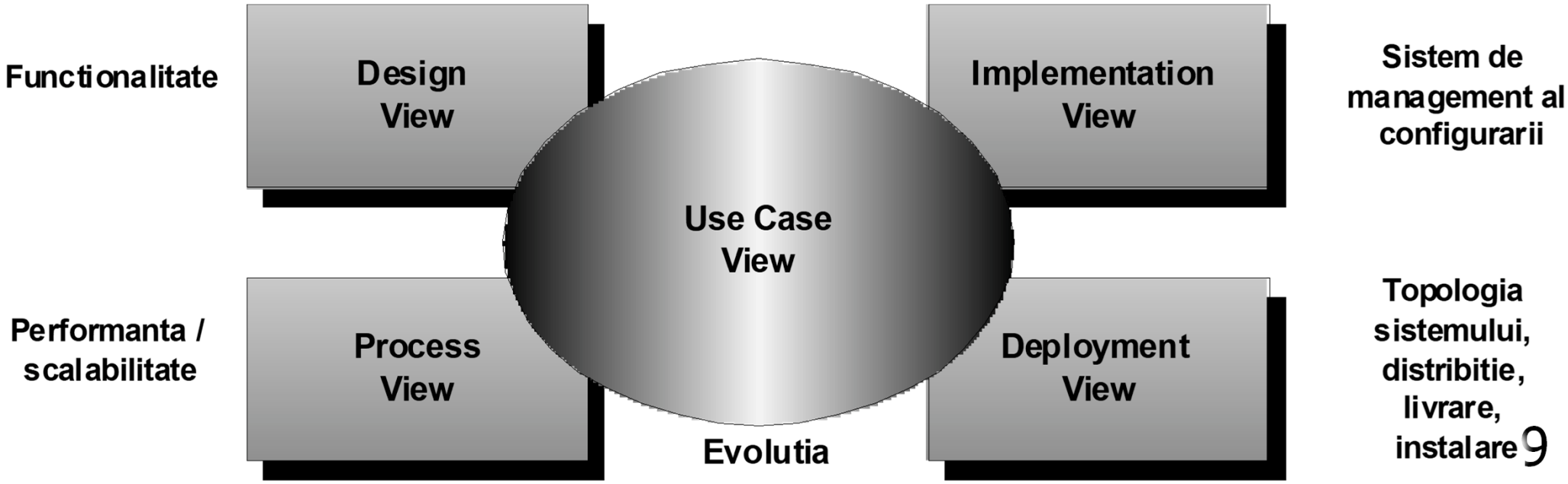
UML poate fi utilizat pentru:

- Descrierea limitelor unui sistem si a functiilor sale principale (*use cases & actori*) → comportamentul functional al sistemului
- Ilustrarea implementarii cazurilor de utilizare prin *diagramele de interactiuni*
- Reprezentarea structurii statice a unui sistem prin *diagrama de clase* (utilizarea claselor+interfetelor pt model. entitatilor statice din sistem) si diagrama de obiecte (arata instantele claselor + legaturi)
- Modelarea evolutiei obiectelor prin *diagrama de tranzitii*
- Structurarea implementarii fizice a arhitecturii sistemului cu *diagramele de componente si operationale* → arata org. elementelor din sistem
- *Diagrama de pachete:* tip special de diagrame de clase; focus pe gruparea claselor
- *Diagrama de activitati:* descr flux de evolutie ale activitatilor
- *Diagrama d secvente:* surprinde tipul si ordinea mesajelor

Arhitectura UML (1)

Arhitectura = setul de decizii privind:

i. Organizarea sistemului *software*.
ii. Selectia elementelor structurale & interfete din care un sistem este format.
iii. Evolutia & colaborarea elementelor.
iv. Structura si evolutia elementelor.
v. Arhitectura sistemului.



Arhitectura UML (2)

Use Case View

- Analiza Cazurilor de utilizare (CU) este o tehnica de "captura" a proceselor din perspectiva utilizatorului.
- Include evolutia "vazuta" de utilizatori, analisti si testatori.
- Specifica "fortele" ce influenteaza arhitectura.
- Aspectele statice sunt reprezentate in diagramele CU.
- Aspectele dinamice sunt reprezentate in diagramele de interactiuni, de activitati si de stari

Design View

- Cuprinde clasele, interfetele si colaborarile ce definesc "vocabularul" unui sistem.
- Definesc necesitatile functionale ale sistemului.
- Aspectele statice sunt reprezentate in diagrama de clase si diagrama de obiecte

Arhitectura UML (3)

Process View

- Defineste cozile si procesele concurentiale si de sincronizare.
- Reprezinta performanta si scalabilitatea.
- Aspecte statice si dinamice reprezentate la fel ca in *Design View*.

Implementation View

- Defineste componentele si fisierele utilizate in realizarea unui sistem fizic.
- Reprezinta managementul configuratiei.
- Aspectele statice sunt reprezentate in diagrama de componente.
- Aspectele dinamice sunt redate in diagramele de interactiuni, de activitati si de stari.

Deployment View

- Defineste nodurile ce formeaza topologia hardware.
- Reprezinta distributia si instalarea sistemului.
- Aspectele statice sunt reprezentate in diagrama de distributii.
- Aspectele dinamice sunt redate in diagramele de interactiuni, de activitati si de stari.

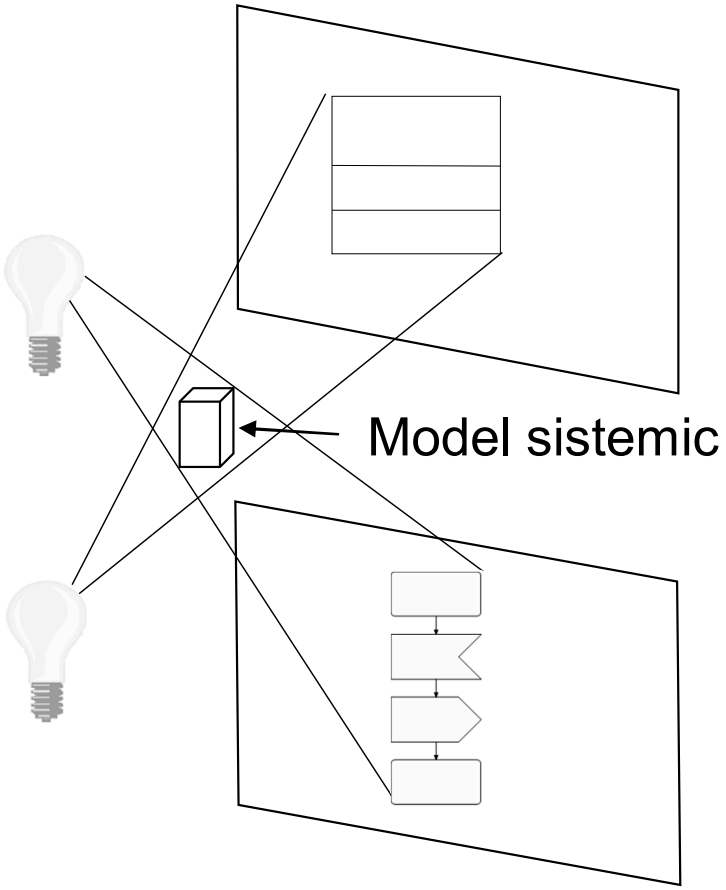
Arhitectura UML (4)

Arhitectura structurata

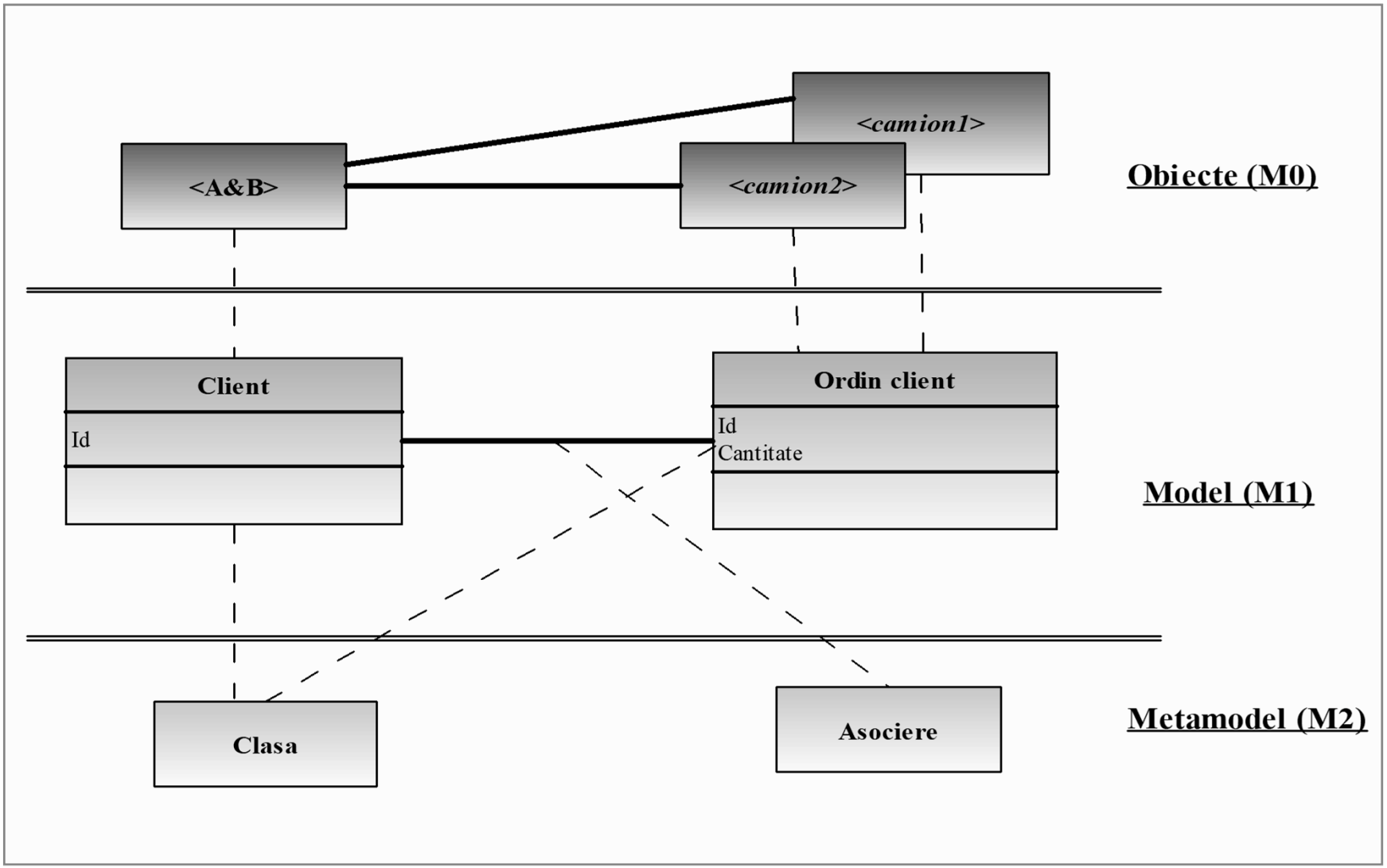
- **Meta-metamodel**: limbaj specific metamodelului
- **Metamodel**: limbaj specific modelului
- **Model**: limbaj specific domeniului
- **Obiecte**: instante ale unui model

Arhitectura UML (5)

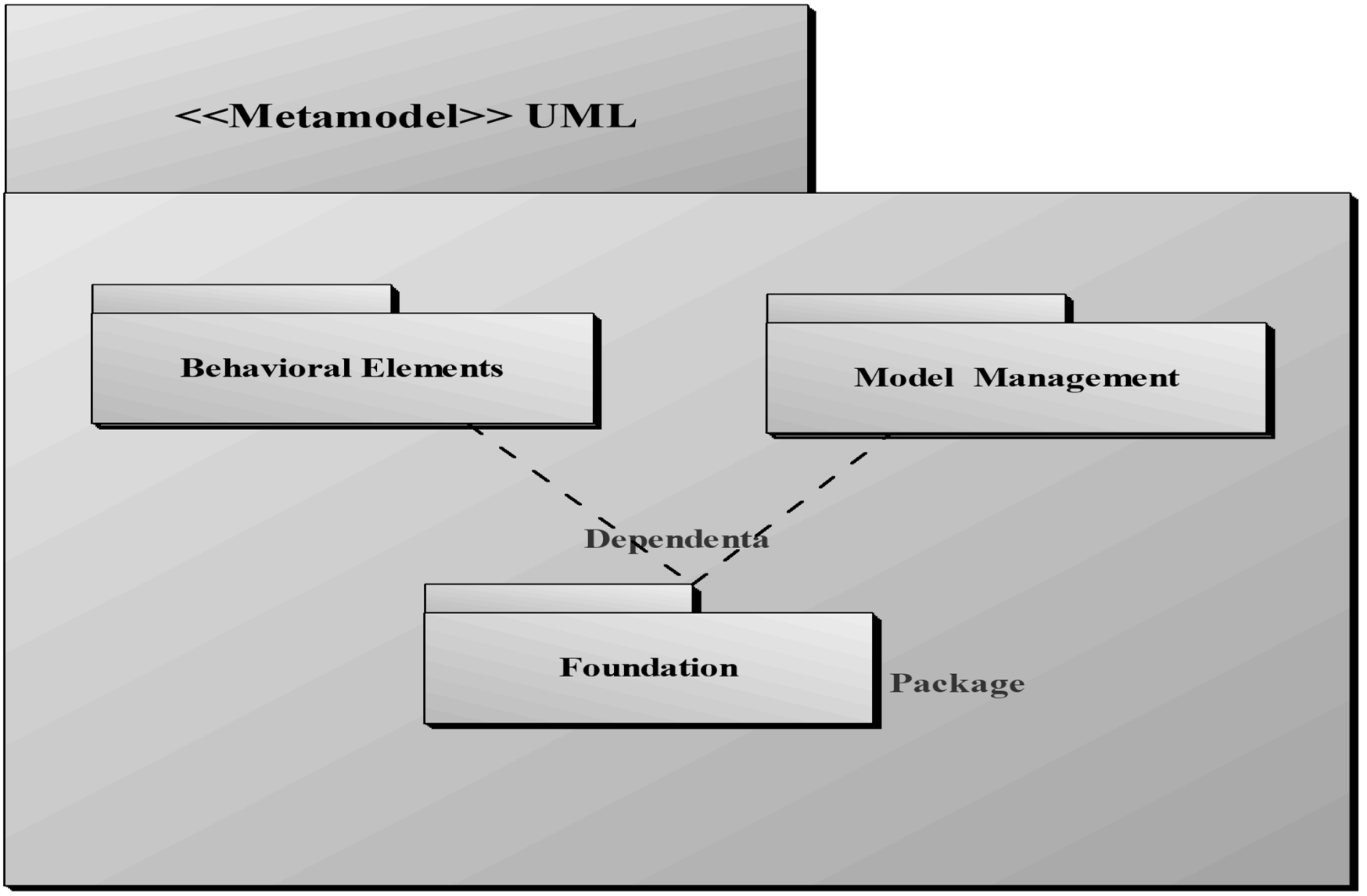
Imagine de ansamblu = suma diagramelor UML



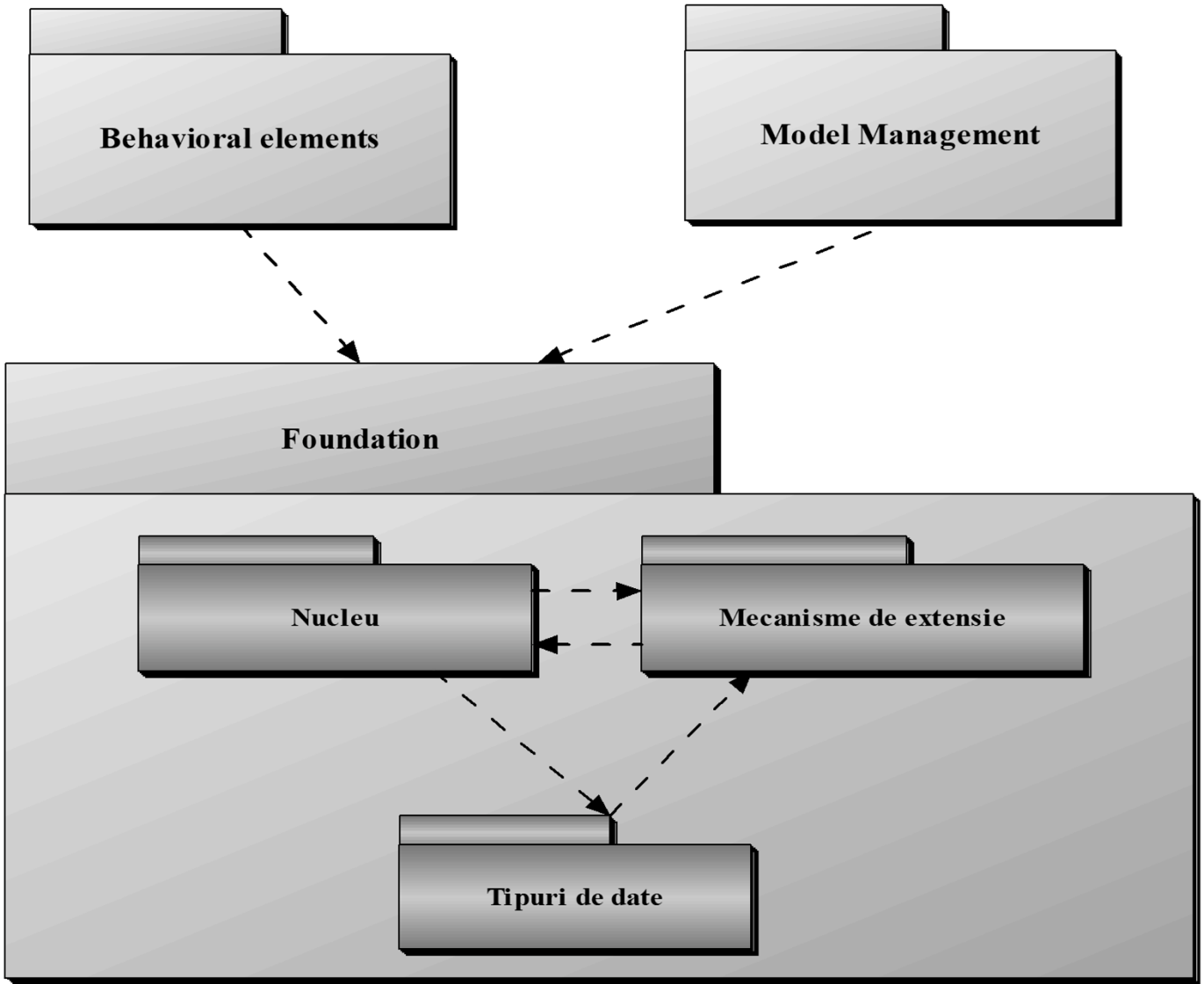
Metamodelul UML (1)



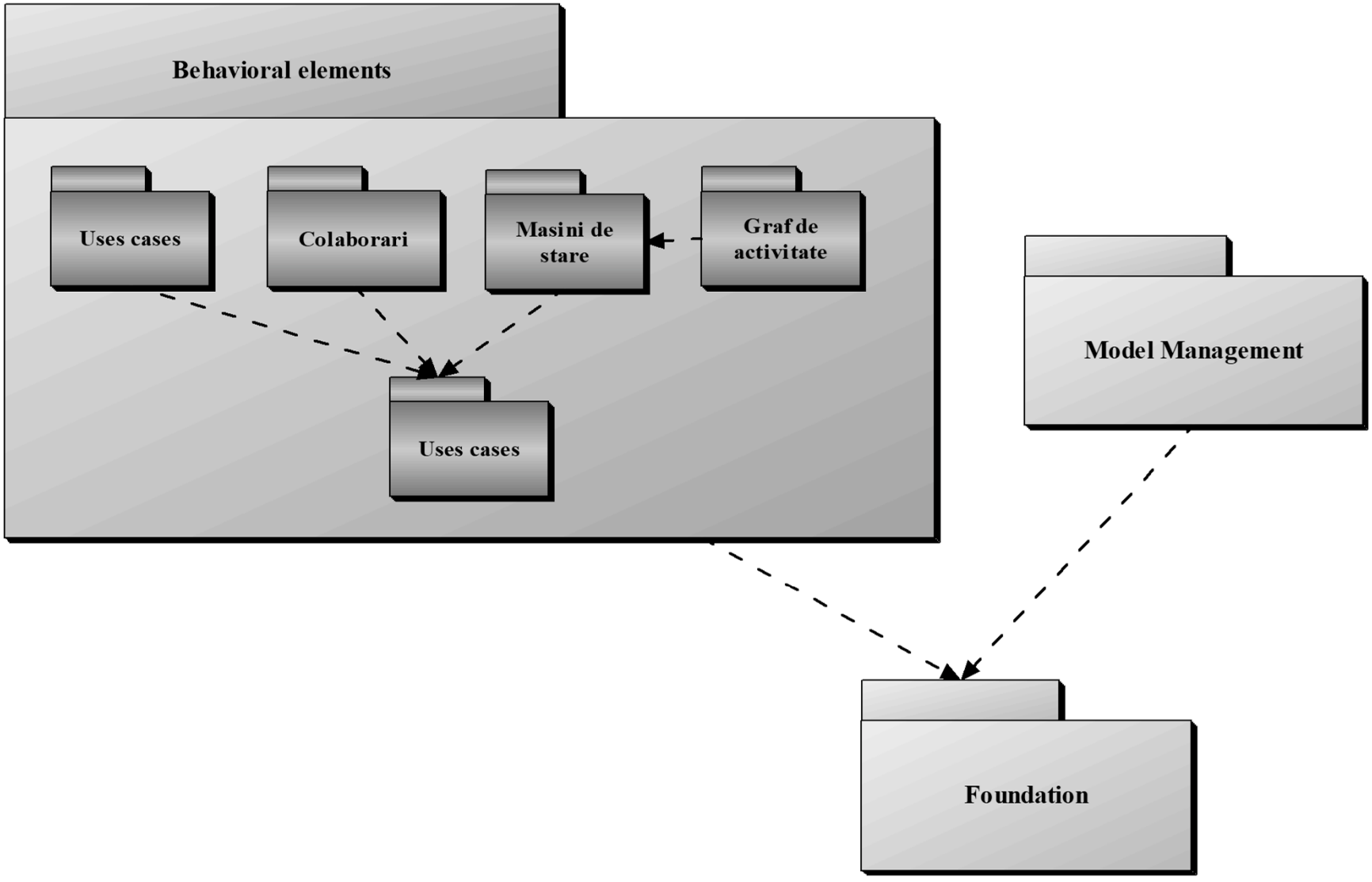
Metamodelul UML (2)



Metamodelul UML (3)



Metamodelul UML (4)



Elemente constructive in UML

Lucruri

Concept de modelare (elementul de individualitate).

Relatii

Leaga elementele individuale (I.e. conceptele)

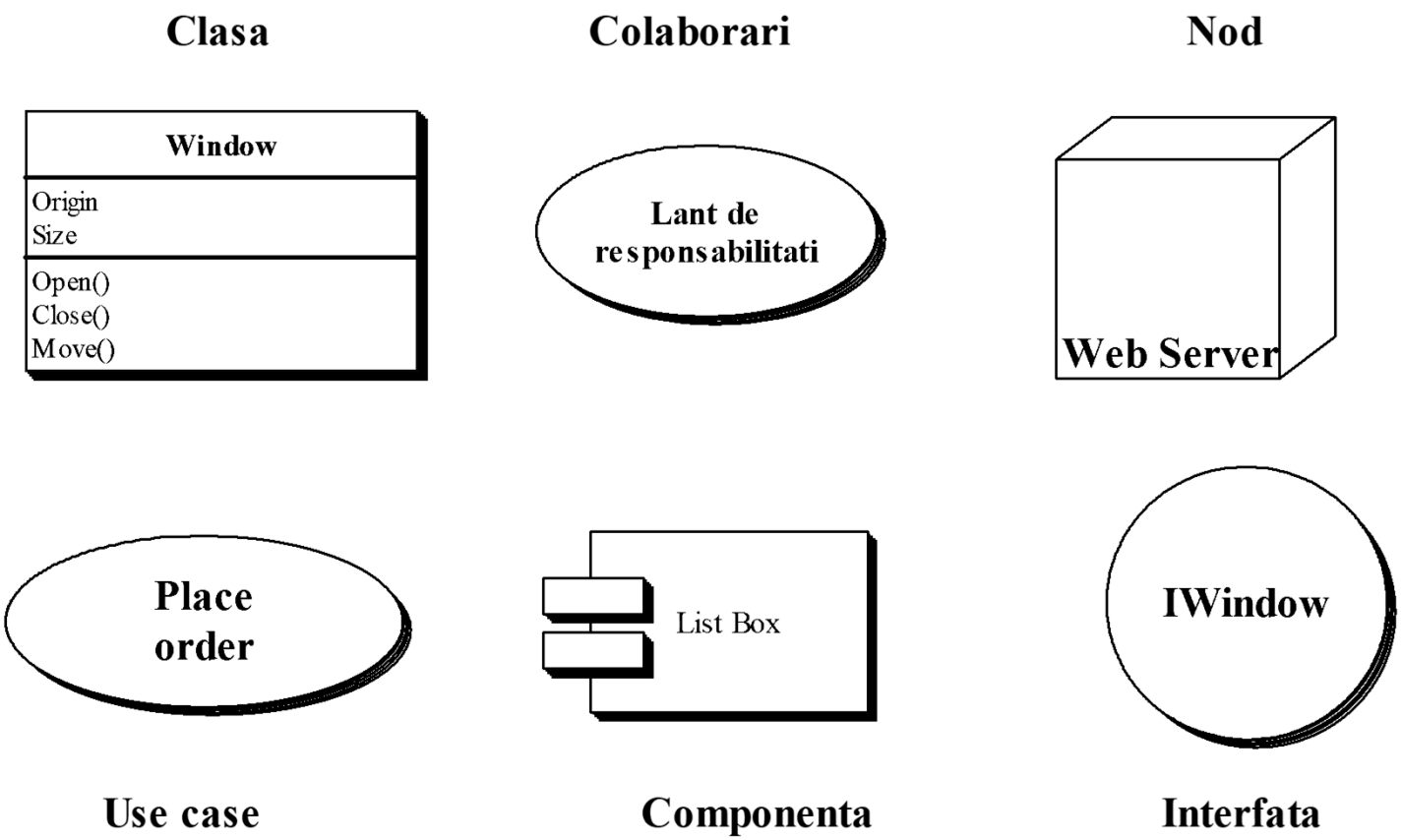
Diagrame

Grupeaza colectiile organizate de elementele individuale (lucruri / relatii)

Lucruri - componente

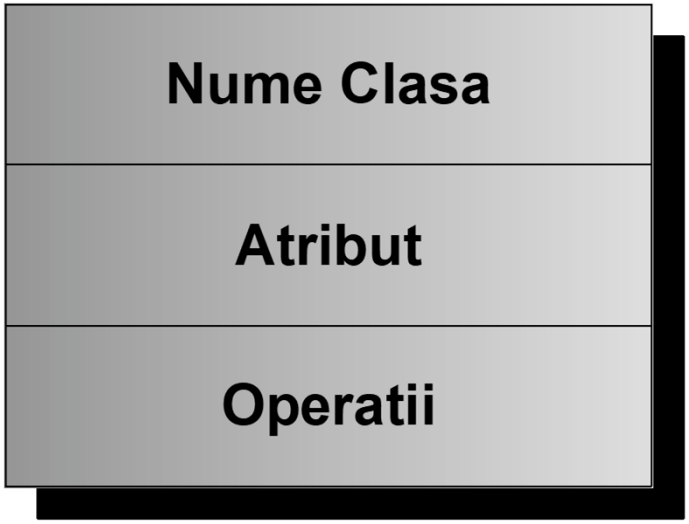
- **Structuri**: "substantivul" modelelor UML
- **Evolutii**: componenta dinamica a modelelor UML
- **Grupari**: componenta organizationala a modelelor UML
- **Adnotari**: componenta explicativa a modelelor UML

Structuri



- Elemente conceptuale sau fizice

Clase (1)

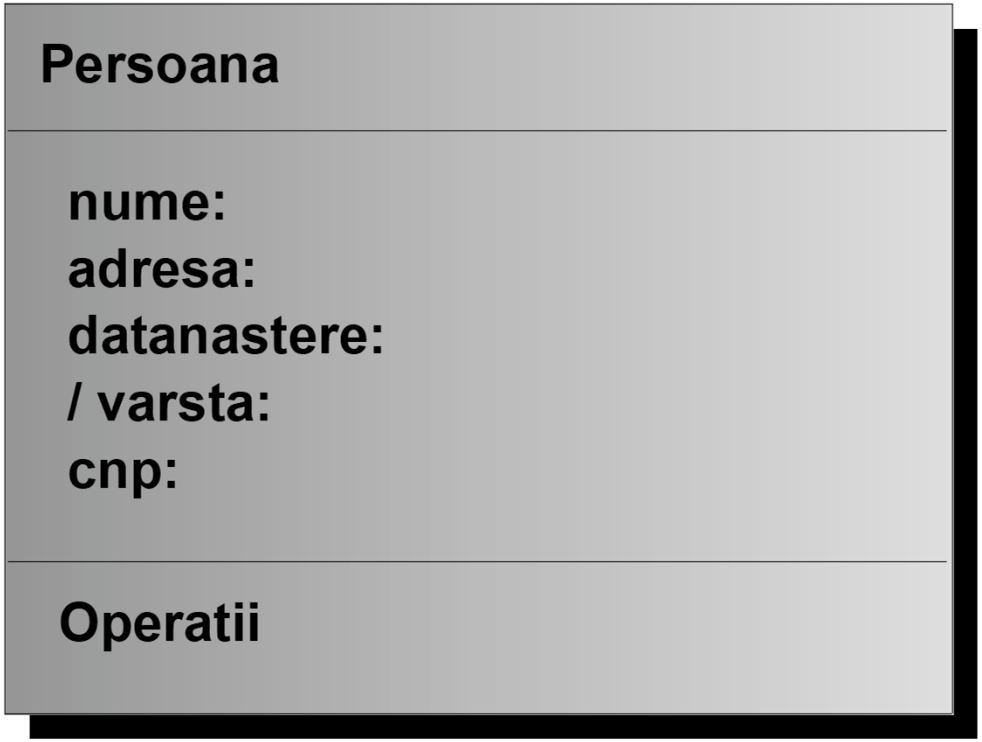


- Din punct de vedere grafic, o clasa este redata ca un dreptunghi;
- Include in mod uzual numele clasei, atribute si operatii (in compartimente separate).
- Numele clasei este obligatoriu
- Atributul este numele propietatii unei clase; forma:

attributeName: Type

Clase (2)

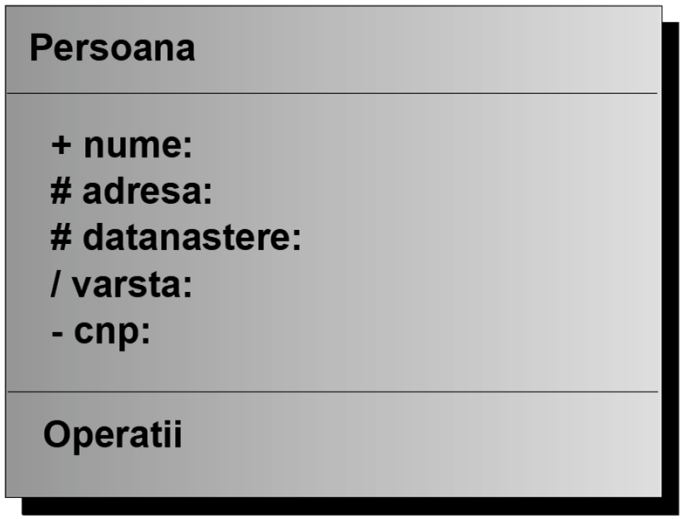
- Un atribut derivat poate fi calculat din alte atribute (el ne-existand in forma directa)
- d.e. varsta unei persoane poate fi calculata in raport de data nasterii si data curenta; reprezentare:



/ varsta: Date

Clase (3)

- Atributele pot fi:
  - + public (permit accesul la date si metode din afara clasei)
  - # protected (interzic accesul din afara clasei, dar il permit din clasele derivate)
  - - private (interzic accesul la date si metode in afara clasei)
  - / derived



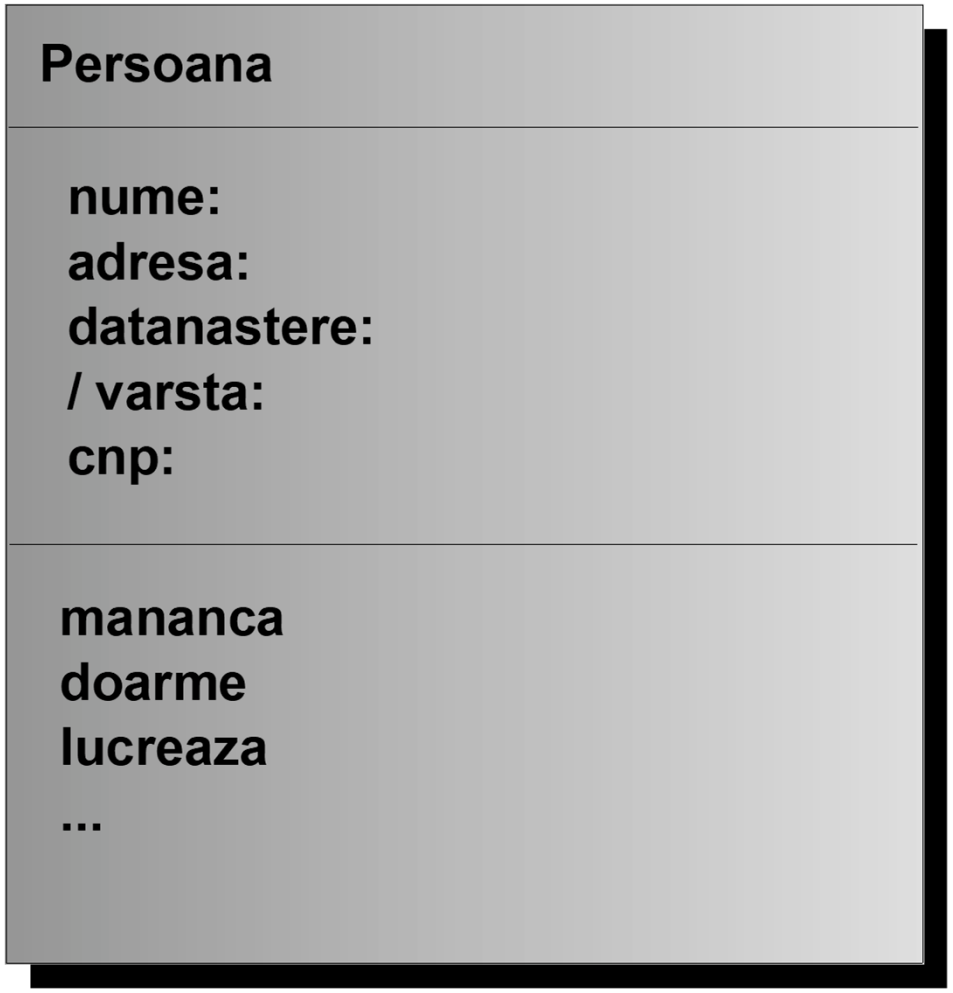
Clase (4)

Vizibilitatea membrilor unei clase si reprezentarea in UML

public	**+**	anywhere in the program and may be called by any object within the system
private	**-**	the class that defines it
protected	**#**	(a) the class that defines it or (b) a subclass of that class
package	**~**	instances of other classes within the same package

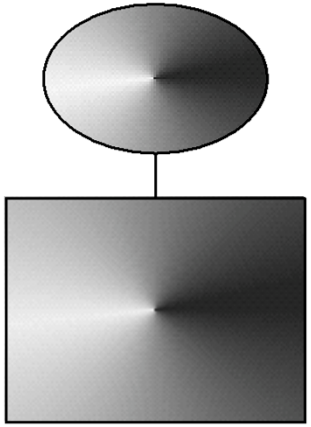
Clase (4)

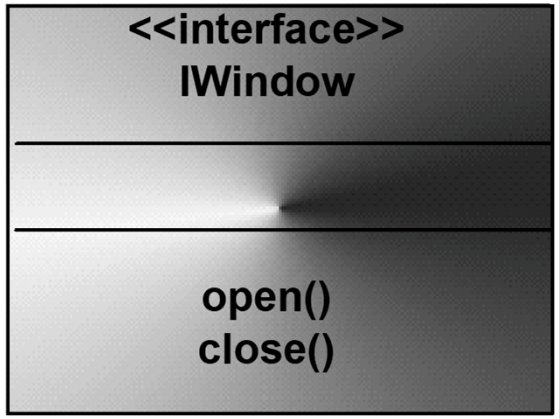
- **Operatiile**: descriu evolutia unei clase



Interfata

- Descrie un set de operatii ce specifica evolutia obiectelor fara a descrie structura lor interna.
- Se reprezinta cu stereotipul **<<interface>>** in fata numelui



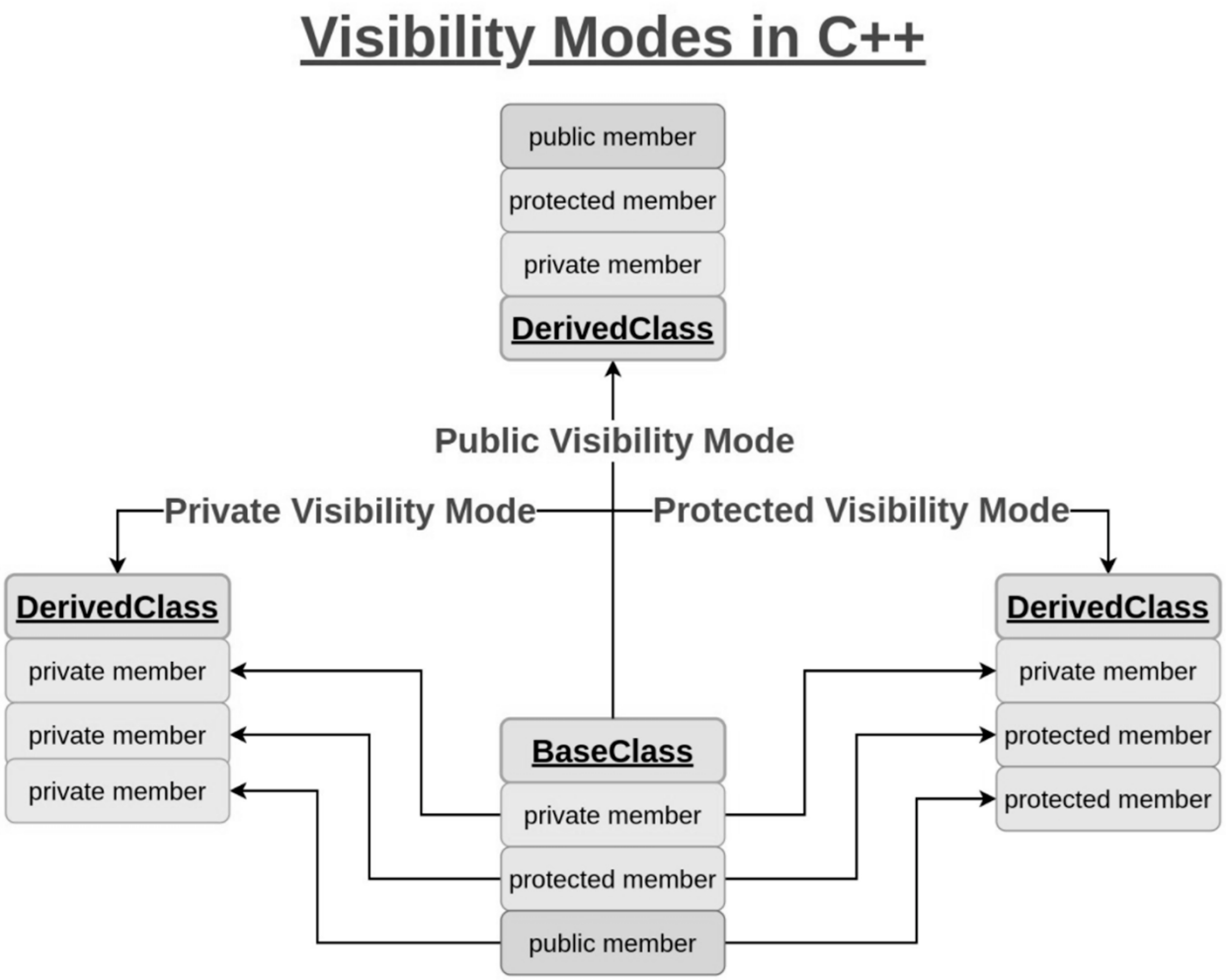


Interfata nu poate fi instantiata

Nu au atribute sau stari

Pot specifica serviciile oferite de o clasa asociata

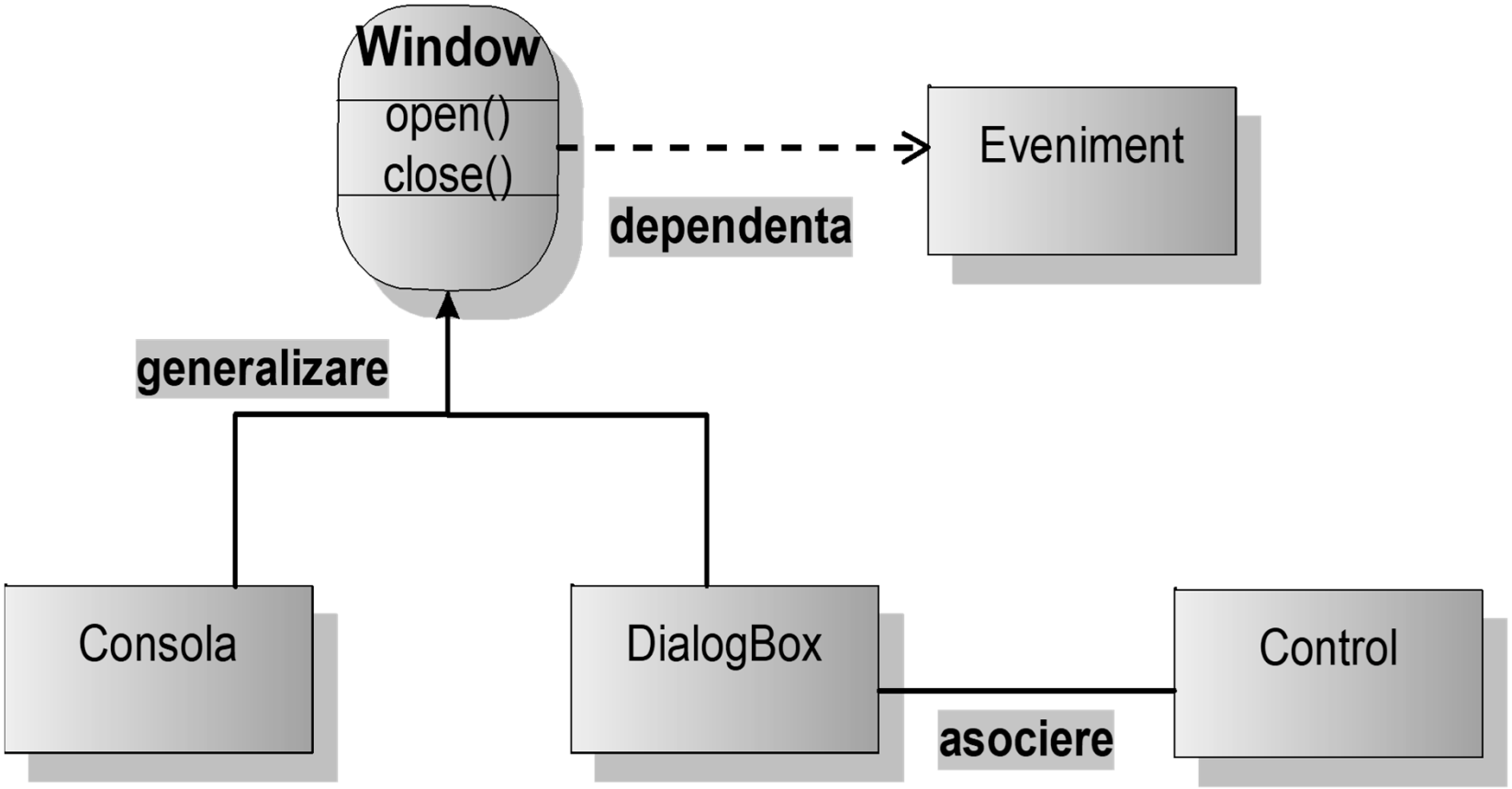
Vizibilitatea claselor in C++



Relatii (1)

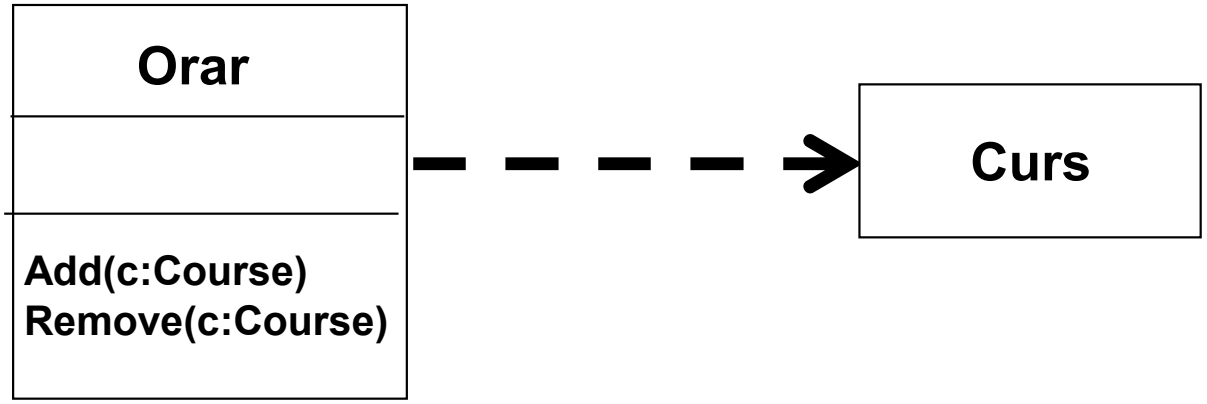
- **Relatiile** ofera o structura de comunicare intre obiecte
- Diagramele de secvente / colaborari sunt examinate pentru a determina ce legaturi trebuiesc stabilite intre obiecte pentru a permite evolutia sistemului; daca doua obiecte "vorbesc", o legatura (*link*) trebuie sa existe intre ele.
- Tipuri de relatii:
  - **Dependenta** – o clasa utilizeaza o alta clasa ("uses")
  - **Asociere** – o clasa este in relatie cu o alta clasa pe o durata mai mare de timp ("has a")
  - **Generalizare**
  - **Realizare** – unul dintre elementele relatiei garanteaza finalizarea asteptatat din partea celuilalt element

Relatii (2)


Window
open()
close()
Eveniment
dependenta
generalizare
Consola
DialogBox
Control
asociere

Dependenta (1)

- Relatia de **dependenta** indica o relatie semantica intre doua sau mai multe elemente.
- O schimbare intr'unul din obiecte (independent, sursa) conduce la modificari semantice la celalalt obiect (dependent, destinatie)



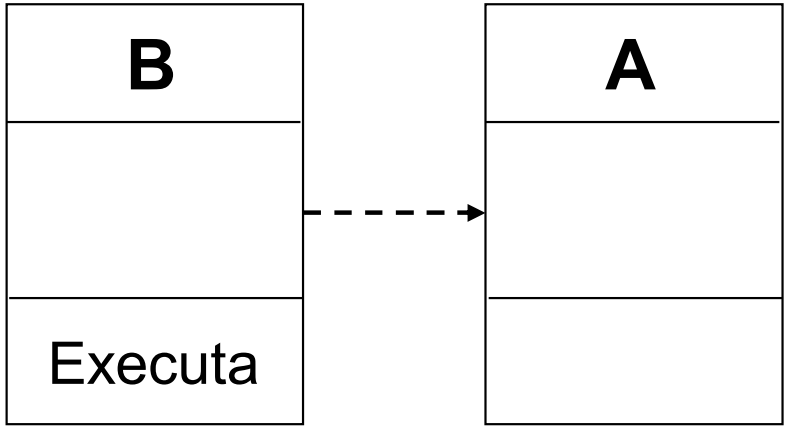
Dependenta (2)

- Caracteristici:
  - Dependența semnifica „O clasă o folosește pe cealaltă”
  - O relație de dependență indică faptul că o schimbare într-o clasă poate afecta clasa dependentă, dar nu neapărat si invers.
  - O relație de dependență este adesea folosită pentru a arăta că o metodă are obiectul unei clase ca argument.

- Forme predefinite:
  - **Refine:** “rafinarea” unui element de modelare prin intermediul altuia
  - **Trace:** acelasi concept, dar pe un nivel de abstractizare diferit
  - **Use:** relatia prin intermediul careia un element solicita prezenta unui alt element pentru buna sa desfasurare.

Dependenta – implementare in C++

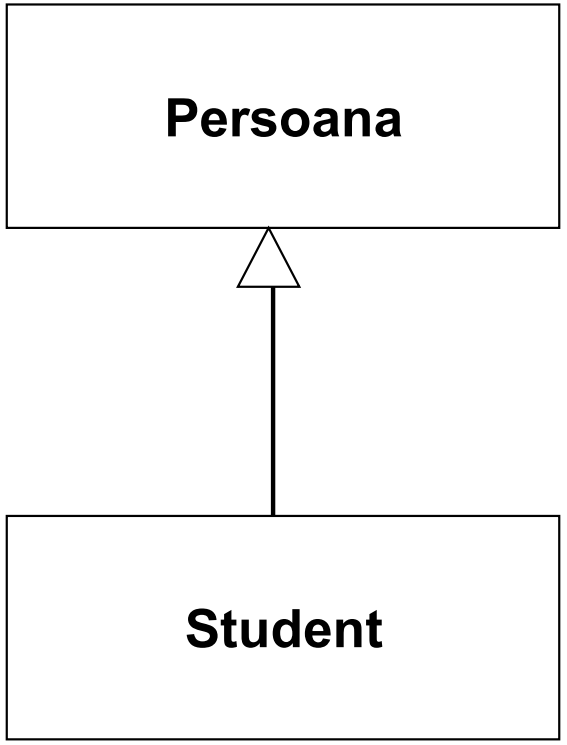
```
class A { ... };
class B
{public:void
    Executa1(void);};


void B::Executa1(void)
{A a;
a * a1 = new A;
// ...
delete a1;}
```



Generalizare (1)

- O relatie de **generalizare / specializare** leaga o subclasa de o superclasa.

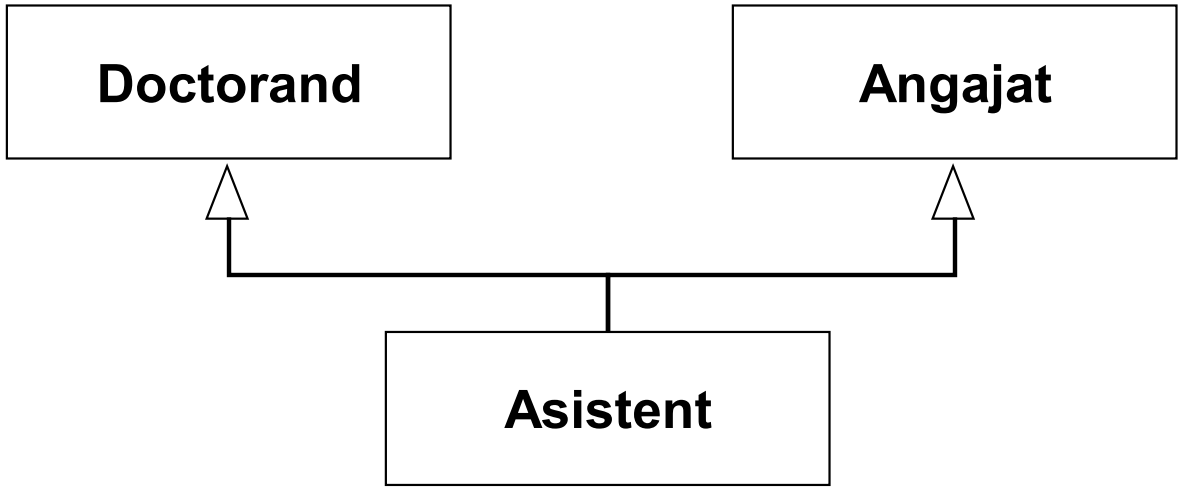
- Indica o **mostenire** a atributelor si actiunilor de la nivelul superclasei la cel al subclasei (sau o **specializare** la nivelul subclasei a elementelor generale din superclasa)



- Reprezentare 

Generalizare (2)

- UML permite ca o clasa sa mosteneasca mai multe superclase.
- **Observatie:** anumite limbaje OO (d.e. Java) nu permit mostenirea multipla



Generalizare (3)

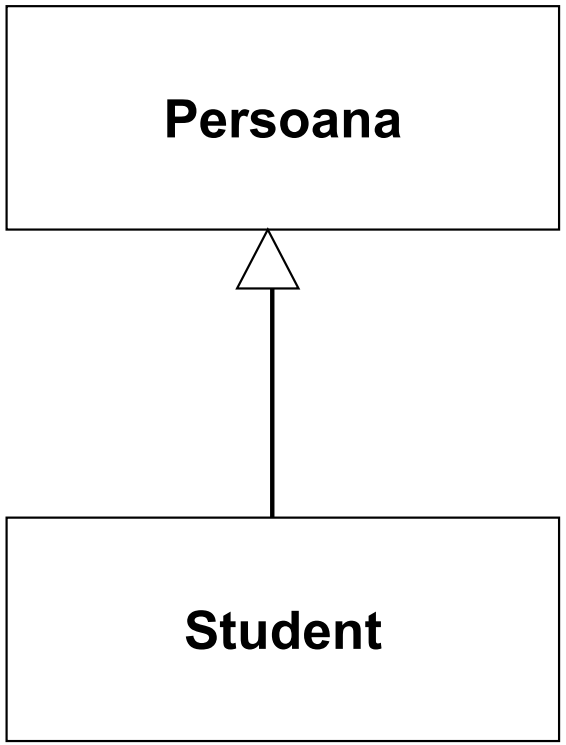
- Mostenire multipla
  - **Avantaje**:
    - Structurarea elementelor in diferite forme si cu diferite legaturi (ca in lumea reala)
    - Posibilitati multiplede utilizare a atributelor si operatiilor din clasele parinte
  - **Dezavantaje**
    - Orice modificare la nivelul superclasei duce la modificari si in subclasa
    - Cand o subclasa mosteneste aceleasi attribute / operatii de la o superclasa, trebuie selectat cu grija ce va fi utilizat

Generalizare (4)

- Restrictii:
  - Overlapping: un element apartine simultan la mai multe subclase
  - Disjoint: contrar situatiei anterioare
  - Complete: sunt specificate toate subclasele
  - Incomplete: alte subclase pot sa apara ulterior

Generalizare – implementare in C++

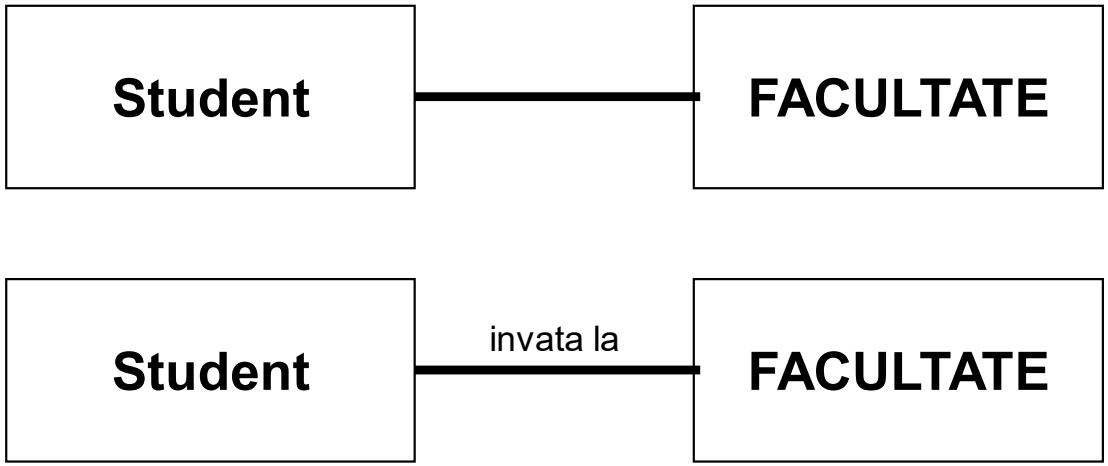
```
class Persoana{ ...
    };
class Student :
    public Persoana{
    ... };
```



Asociere

- Daca doua clase (in cadrul unui model) trebuie sa comunice una cu alta, trebuie sa existe o legatura (*link*) intre ele.
- Asocierea (delegarea) reprezinta o asemenea legatura.

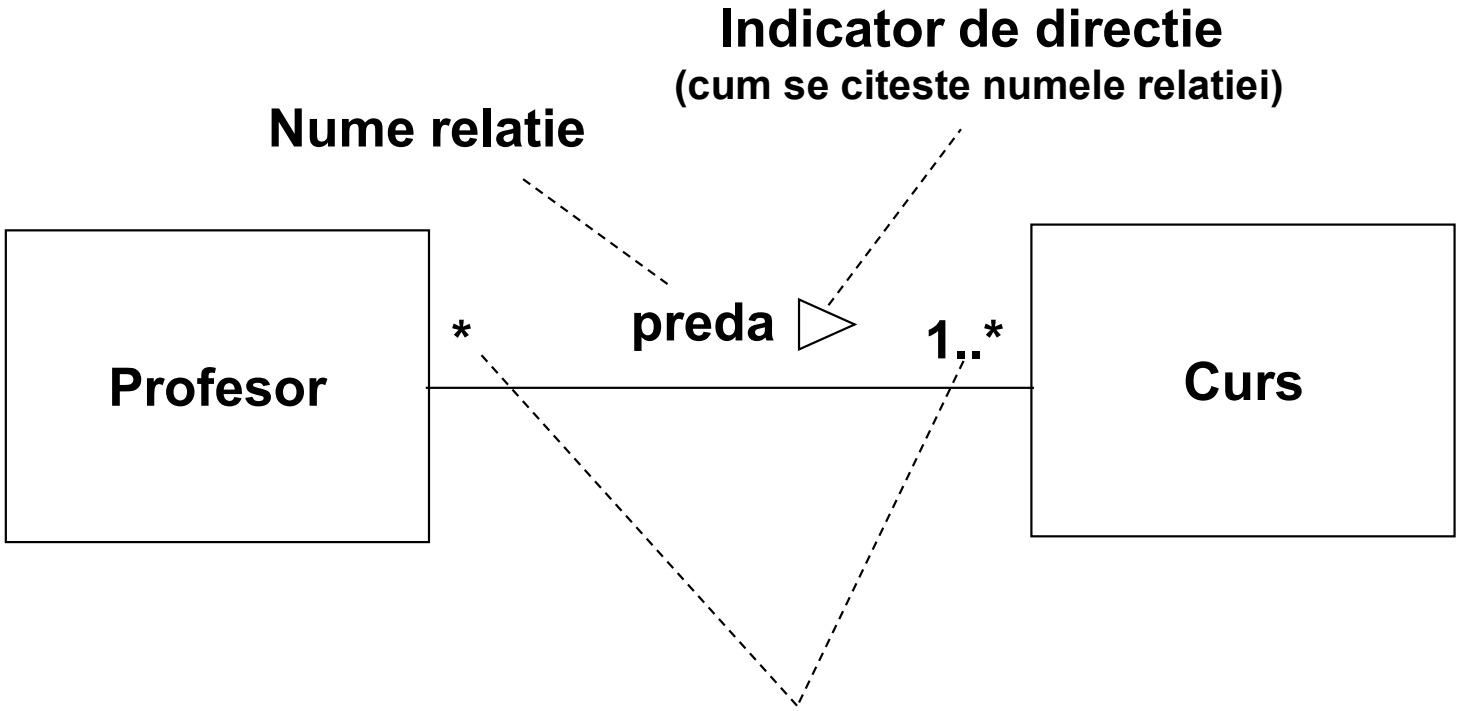
Reprezentare: —— sau →



Proprietatile asocierii

- **Nume**
  - *Numele asocierii*
- **Rol** (nu mai exista in UML2; inlocuit cu "association end name")
  - *Rolul specific al asocierii*
- **Multiplicitate**
  - *Indica numarul de obiecte ce sunt conectate*
- **Tip**
  - *Asociere, agregare, compozitie;* Caz particular: asociere reflexiva (in obj aceleiasi clase)
- **Directie**
- **Calificator**: atribut / grup de atribute a caror valoare serveste pt partajarea ansamblului de obj. participante la o asociere

Asocierea - exemplu



Multiplicitate (defineste numarul de obiecte asociate cu o instanta a asocierii

Multiplicitatea

O clasa poate fi legata de alta printr'o relatie:

- One-to-one
- One-to-many
- One-to-one or more
- One-to-zero or one
- One-to-a bounded interval (one-to-two through twenty)
- One-to-exactly n
- One-to-a set of choices (one-to-five or eight)

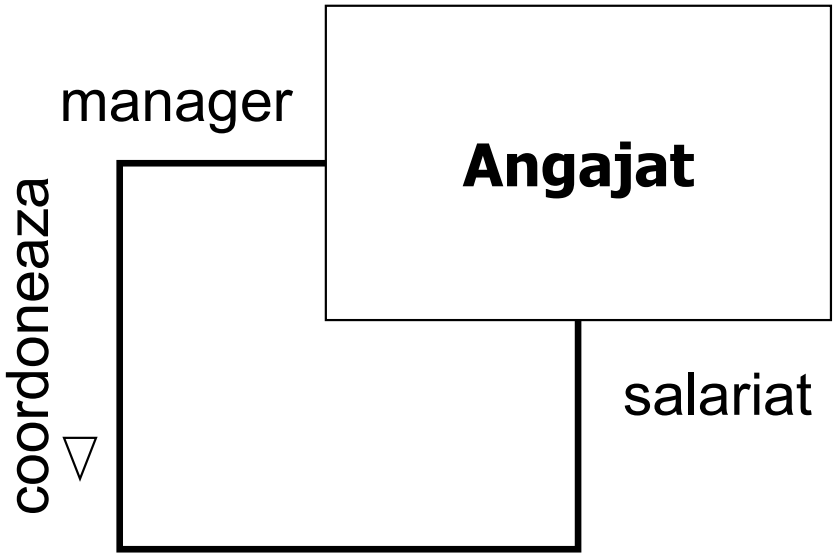
Exprimarea multiplicitatii:

- Exact unu -1
- Zero sau unu - 0..1
- Multe - 0..* sau *
- Unu sau mai multe - 1..*
- Valoare exacta - d.e. 3..4 sau 6
- Reprezentare complexa –d.e. 0..1, 3..4,6..* semnifica orice numar de obiecte, altul decat 2 sau 5

Asocierea reflexiva

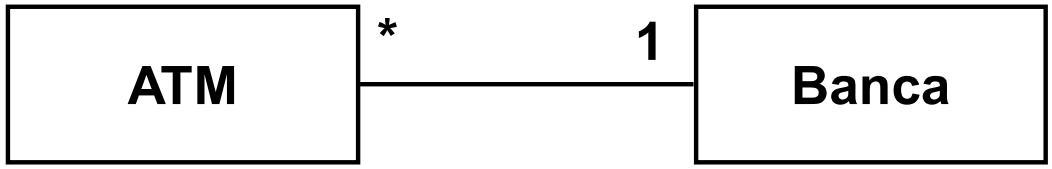
- Asocierea a 2 instante ale aceleiasi clase
- O clasa are responsabilitati multiple:

→ d.e. angajatul unei firme poate fi seful unui grup format din 10 alti angajati.

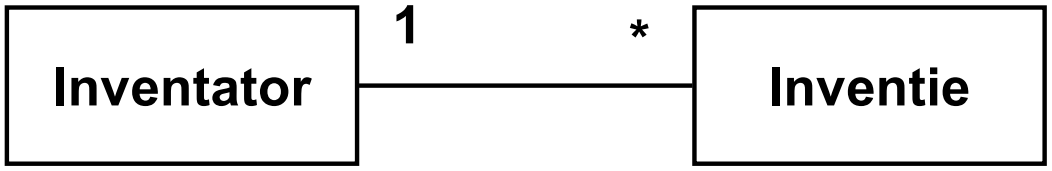


Multiplicitatea asocierilor (1)

Multi-la-unu: o banca are mai multe ATM-uri; 1 ATM este legat doar la o banca

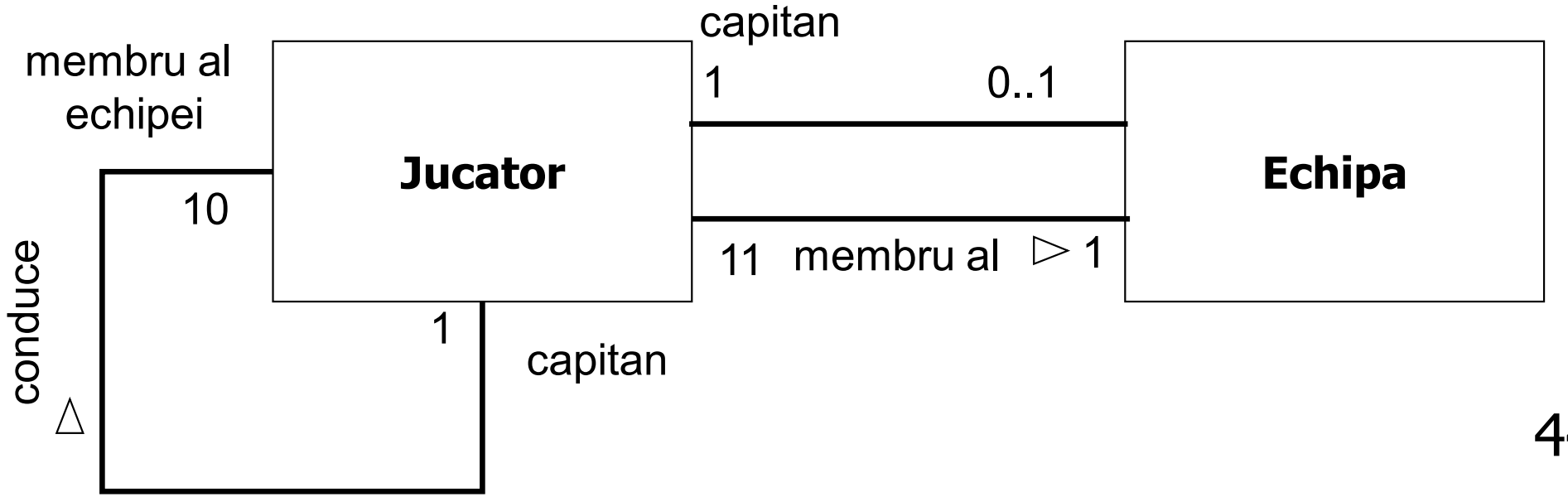


Unu-la-multi: un inventator are mai multe inventii, o inventie este a unui singur inventator

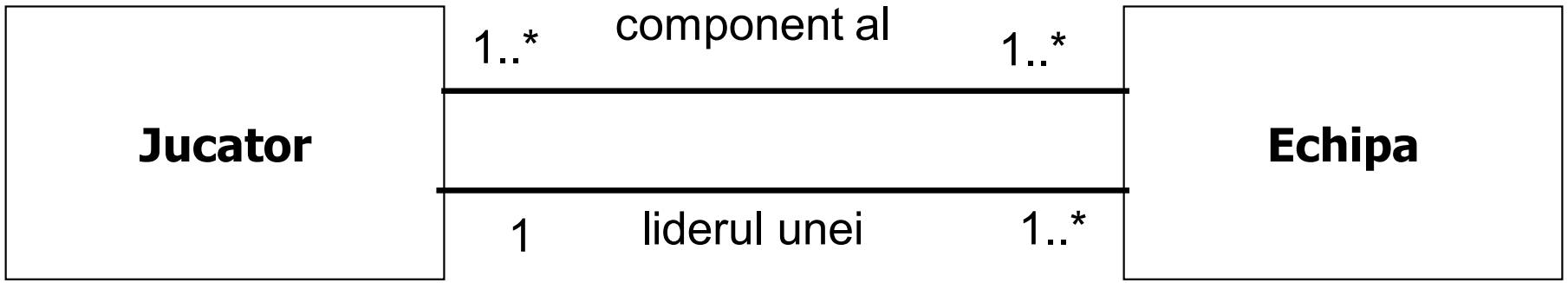


Multiplicitatea asocierilor (2)

- O echipa de fotbal are 11 jucatori. Unul dintre ei este capitanul echipei.
- Un jucator poate juca numai intr'o echipa.
- Un capitan de echipa poate conduce o singura echipa.

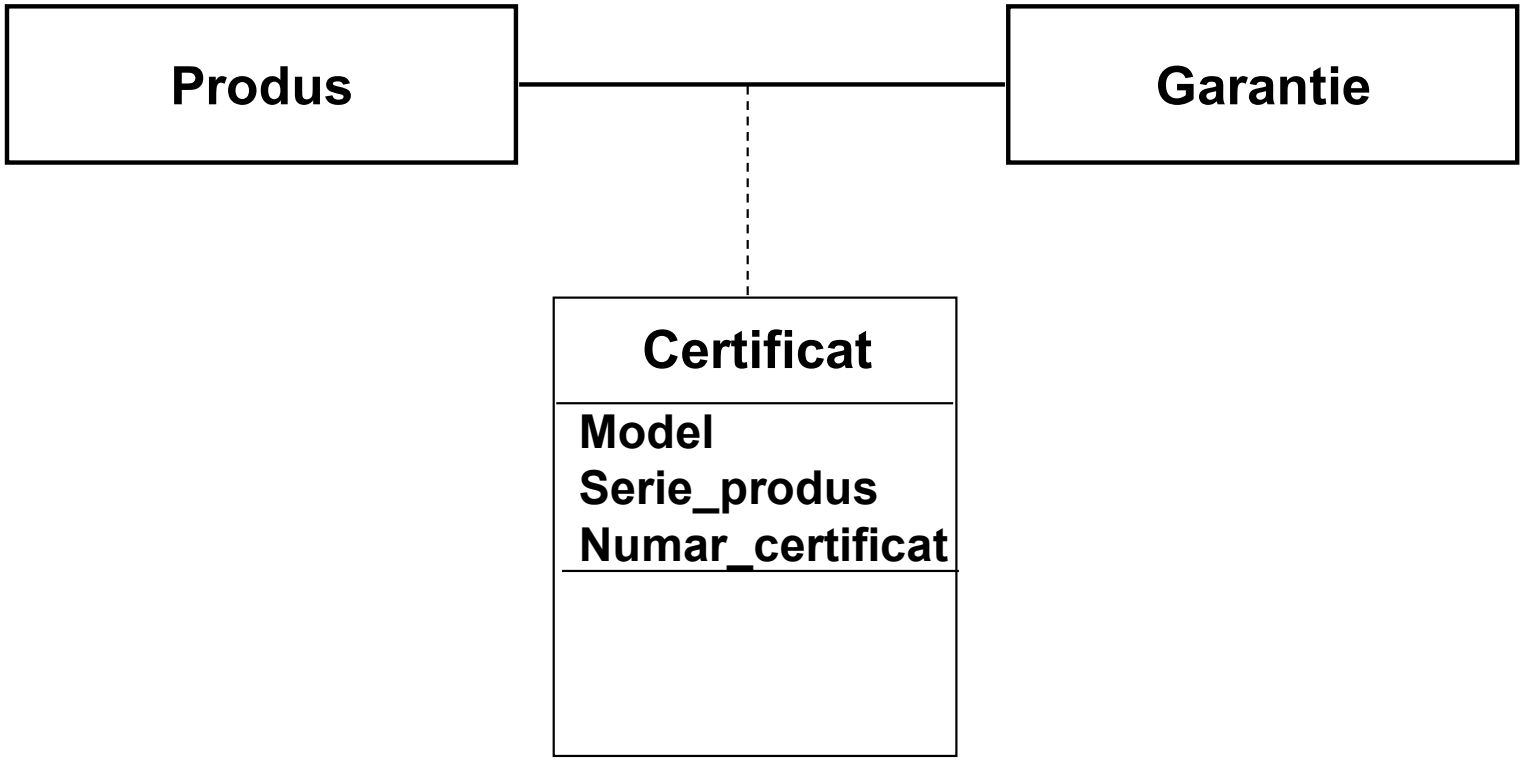


Asocieri duale



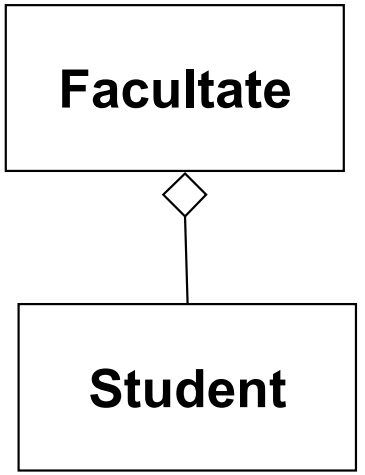
Clasa unei asocieri

- Asocierile pot fi si obiecte in sine si reprezinta instantieri ale unor clase (*link classes* | *association classes*).

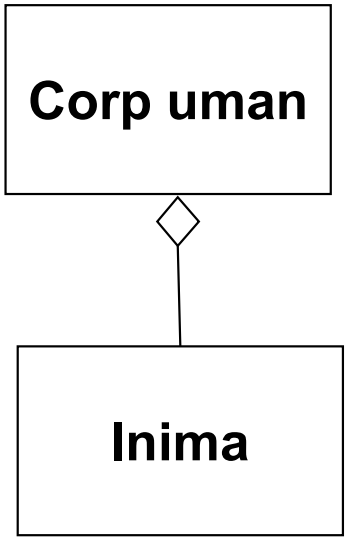


Asociere – agregare - compunere

- **Agregarea** defineste o relatie “parte–intreg”, in care “partea” poate exista in afara “intregului”.
- Este introdusa o relatie de tip: "**has a**".

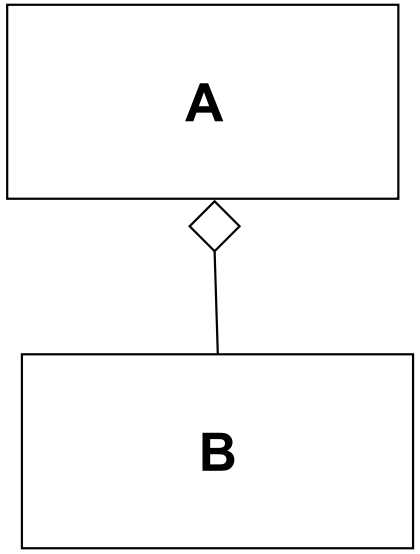


- **Compunerea** este un tip special de agregare: intregul contine mai multe parti, iar partile nu pot exista in afara intregului. Elementele componente “traiesc” si “mor” odata cu intregul



Agregare – reprezentare in C++

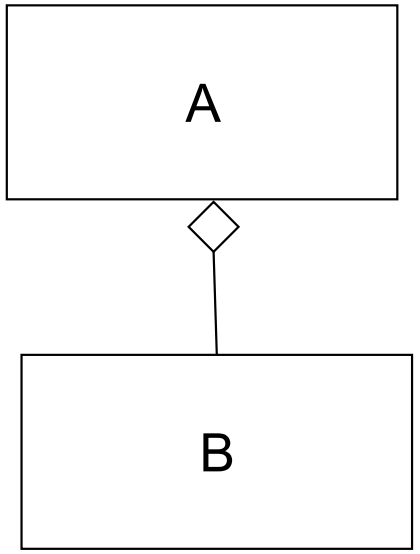
```
class B { ... };
class A
{
private:
A * a;
};
```



- Observatie: **a** este instantierea lui **A**

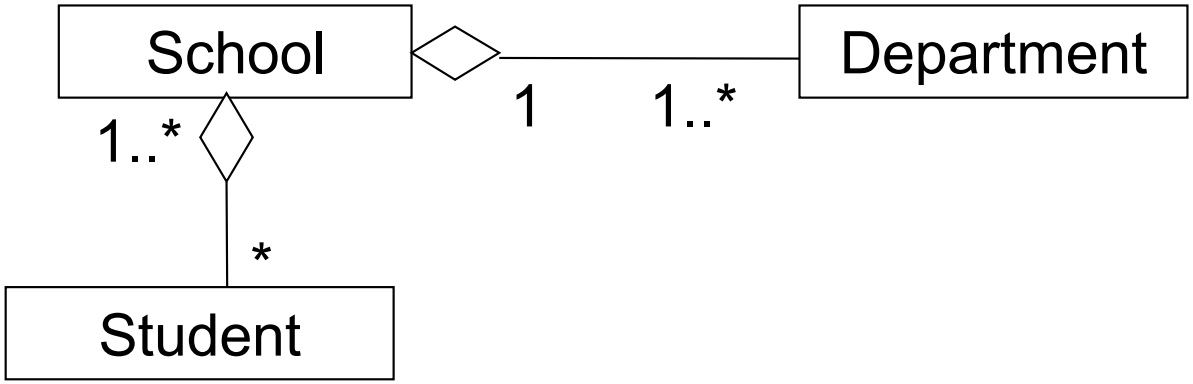
Compozitie – reprezentare in C++

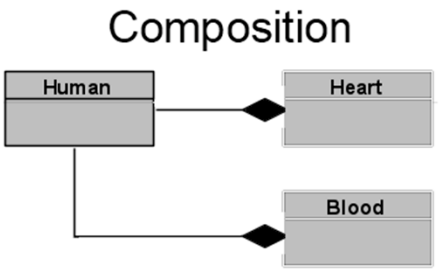
```
class B { ... };
class A
{
private:
A a;
};
```

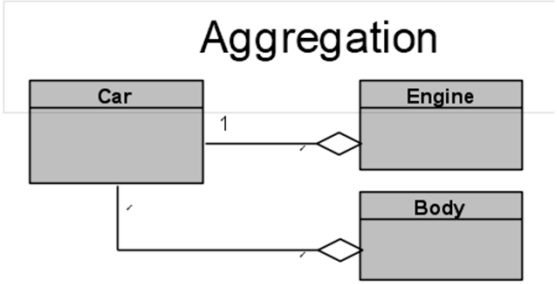


- Observatie: **a** este instantierea lui **A**

Exemple


School
Department
1
1..*
1..*
*
Student


Composition
Human
Heart
Blood

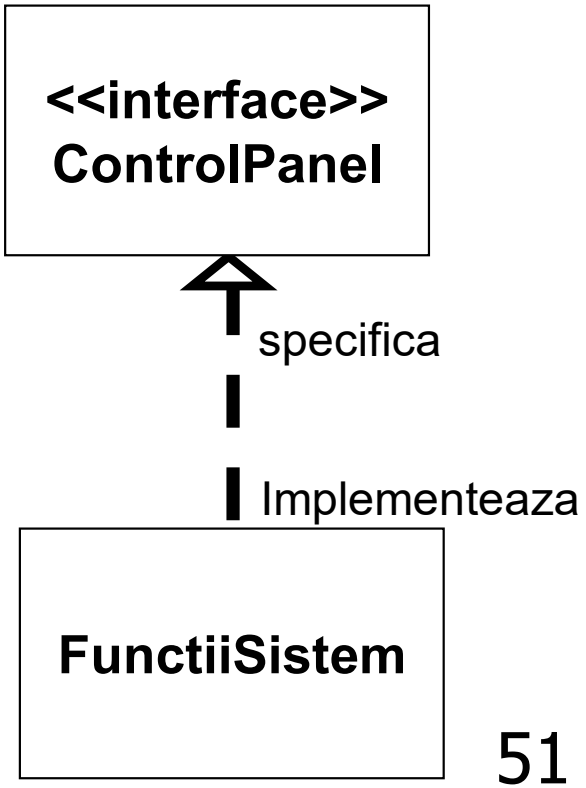

Aggregation
Car
Engine
1
Body

Realizare

- O relatie semantica intre doua elemente, in care unul dintre elemente "garanteaza" finalizarea actiunilor asteptate din partea celuilalt element.

Realizare 

- **Interfata:** specifica operatiile vizibile ale unei clase / pachet fara a define si structura interna a acestora
- **Realizarea** este relatia ce leaga o clasa de *interfata* aferenta

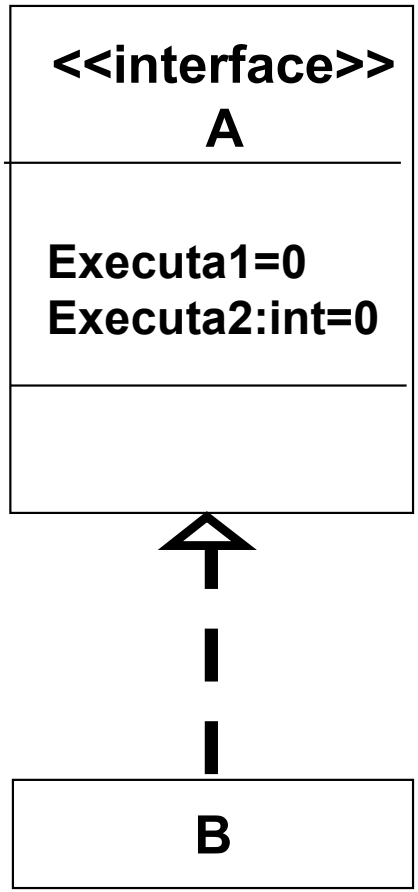


Observatie: in UML interfata este o clasa abstracta

Exemple

- Faculty & student (asociere)
- Hospital & doctor (asociere)
- Door & Car (agregare)
- Member & Organization (agregare)
- People & student (generalizare)
- Circle & point (generalizare)
- Department & Faculty (asociere)
- Employee & Faculty (generalizare)
- Item & Printer (generalizare)

Interfata – implementare in C++




```
class A
{
public:
virtual void Executa1(void) = 0;
virtual int Executa2(void) = 0;
};
class B : public A
{
public:
virtual void Executa1(void);
virtual int Executa2(void);
};
void Bar::doSomething(void) {...};
Int Bar::doSomethingOther(void)
    {...};
```

Nu este realizabila in aceasta forma.
Se poate implementa o clasa virtuala (fara atribute, fara metode private / protejate, fara implementari de metode)

Reguli generale

Modelarea relatiilor

- Se utilizeaza dependentele daca relatiile nu sunt structurate (in alt mod).
- Se utilizeaza generalizarea in cazul unei relatii "**is-a**".
- Nu se recomanda introducerea generalizarilor ciclice.
- *Balance generalizations - Not too deep, not too wide.*

Trasarea unei relatii in UML

- Se vor folosi in special liniile drepte si cele oblice.
- Se va evita intersectia liniilor.
- Vor fi reprezentate doar relatiile necesare pentru o buna intelegere a unui grup de clase / obiecte.
- Se vor evita asocierile redundante.