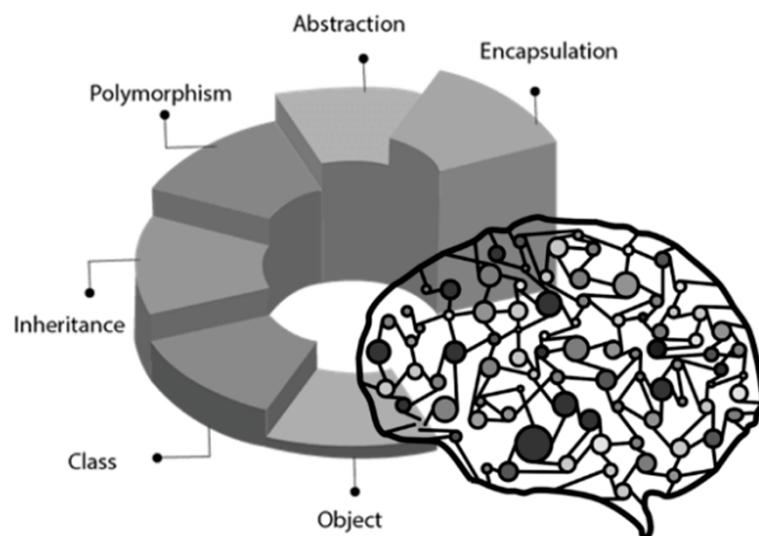


M. Caramihai, © 2022

PROGRAMAREA ORIENTATA OBIECT

CURS 2

Conceptele de baza ale POO



Introducere

Descompunerea sistemelor de mari dimensiuni:

1. algoritmic: pune in evidenta ordinea evenimentelor
2. Prin intermediul obiectelor: pune in evidenta elementele ce actioneaza / sunt supuse unor actiuni

Diferente fata de abordarea clasica:

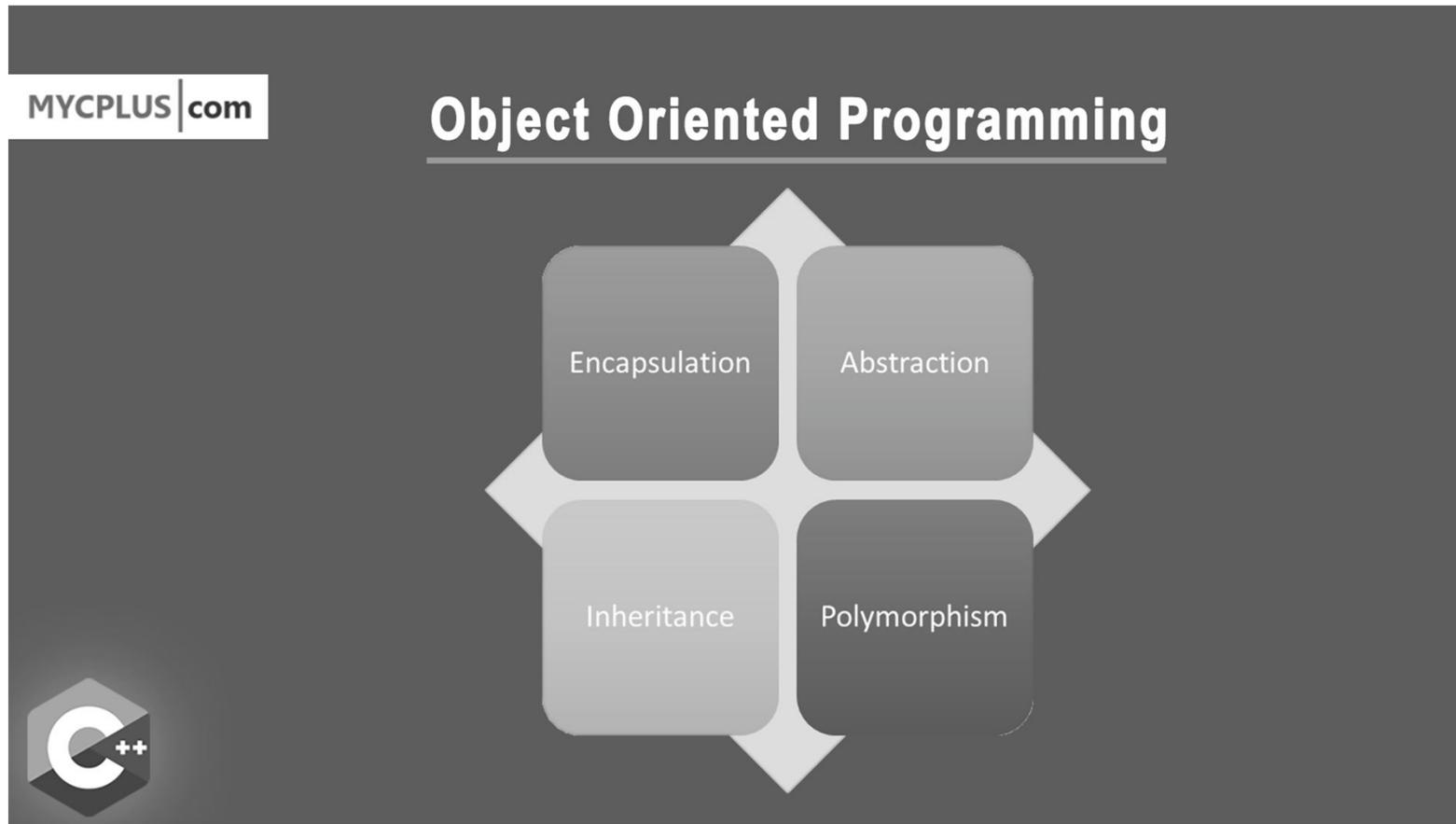
- POO se bazeaza pe obiecte si nu pe algoritmi
- Fiecare obiect este o instanta a unei clase
- Clasele pot fi legate intre ele prin relatii de mostenire

Obs: daca una din aceste caracteristici lipseste, nu este vorba de programare OO

Stiluri de programare

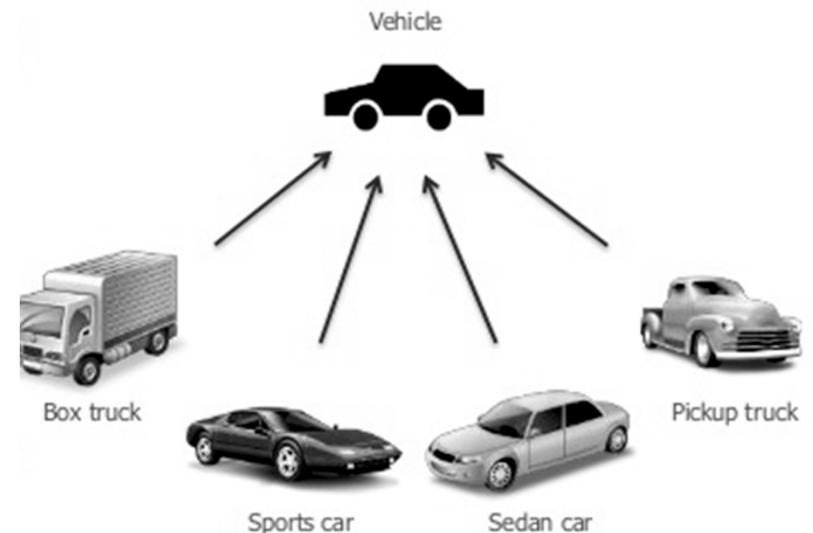
1. Orientatate catre proceduri → algoritmi
2. Orientatate catre obiecte → clase si obiecte
3. Orientatate catre logica → scop (calcul predicativ)
4. Orientatate catre reguli → reguli de tip “if – then”
5. Orientatate catre constrangeri → relatii de invarianta

Principii de baza ale abordarii obiectuale



Ce este abstractizarea (1) ?

- Exemplul **atlasului**: → scara de reprezentare a marimilor
- Definitie: procedura de mascare a detaliilor unui obiect in scopul accentuarii / punerii in evidenta a altora specte / detalii / structuri
- Nivele de abstractizare:
 1. Cel mai de sus: comunitatea obiectelor ce trebuie sa interactioneze unele cu altele in vederea indeplinirii unui scop comun



Ce este abstractizarea (2) ?

2. Valabil in anumite limbaje OO: grup de obiecte ce lucreaza impreuna si care sunt grupate in *unit-uri*, d.e. *package* (Java), *name spaces* (C++) si *units* (Delphi)
3. Nivelul de interactiune intre 2 obiecte (interactiune *client / server*, C/S). In acest caz exista 2 sub-nivele de abstractizare:
 - Abstractizare *client* → *server*
 - Abstractizare *server* → *client*

Ce este abstractizarea (3) ?

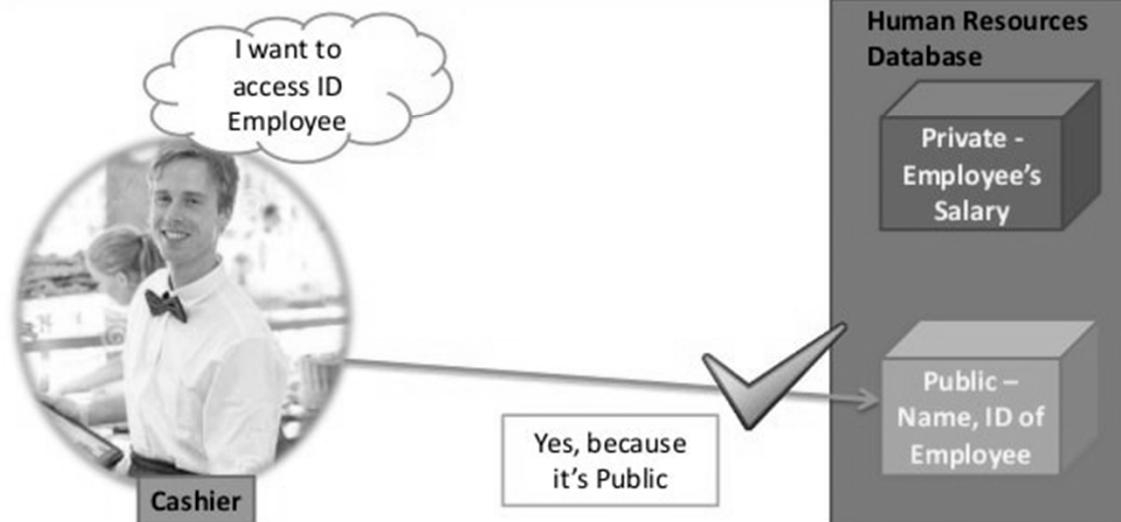
- Tipuri de abstractizari: entitate / actiune / ms virtuala / incident
- Abstractizarea este legata de notiunea de "invarianta"; ruperea unei invariante conduce la lipsa vizibilitatii unor obiecte cu altele
- Orice abstractizare are proprietati
 - Statice, de ex nume fisier
 - Dinamice, de ex valoarea unei prop statice → un nume de fisier poate fi schimbat
- **Consecinta: *abstractizarea si encapsularea sunt concepte complementare, i.e. pentru ca abstractizarea sa functioneze, trebuie sa existe encapsulare***

Ce este incapsularea ?

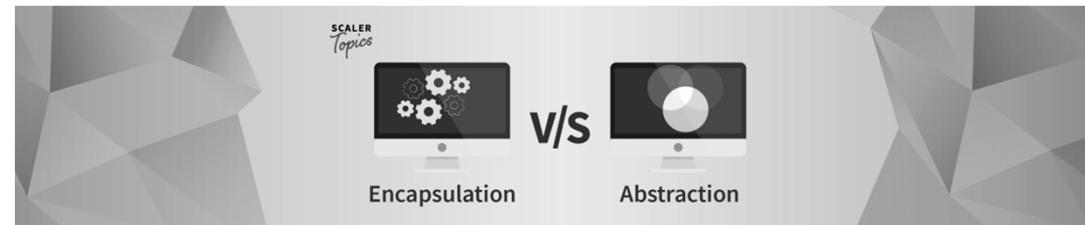
Ascunde
implementarea
fata de clienti

→ Clientii
depind de
interfata

Encapsulation



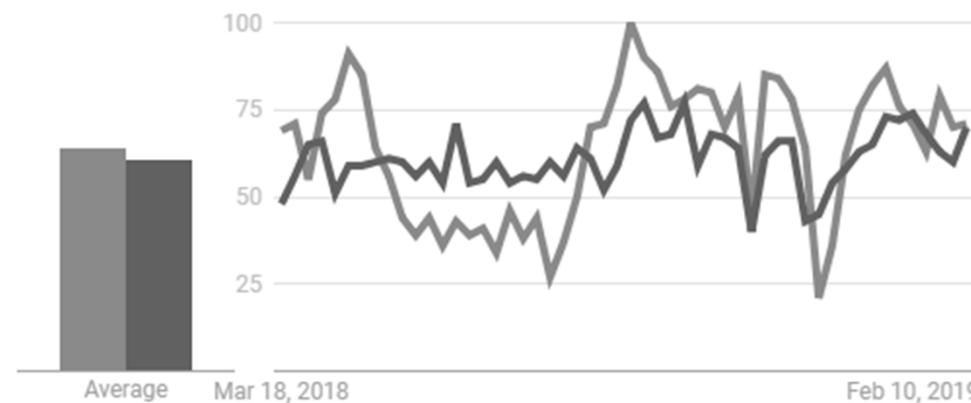
Incapsulare vs abstractizare



Interest over time

Google Trends

● abstraction ● encapsulation



Incapsulare vs abstractizare

Utilizare

- Abstractizare: rezolvă dificultatile ce apar în etapa de proiectare.
- Încapsularea: rezolvă dificultatile ce apar în etapa de implementare.

Focus

- Abstractizare: focalizarea pe ceea ce face obiectul în loc de modul în care o face.
- Încapsularea: permite ascunderea codului / datelor într'o singură unitate pentru a le securiza de lumea exterioară.

Incapsulare vs abstractizare

Implementare

Abstractizare: se utilizeaza folosind Interfața și Clasa abstractă.

Încapsulare: se utilizeaza folosind Modificatori de acces (Public, Protejat și Privat.)

Utilizare

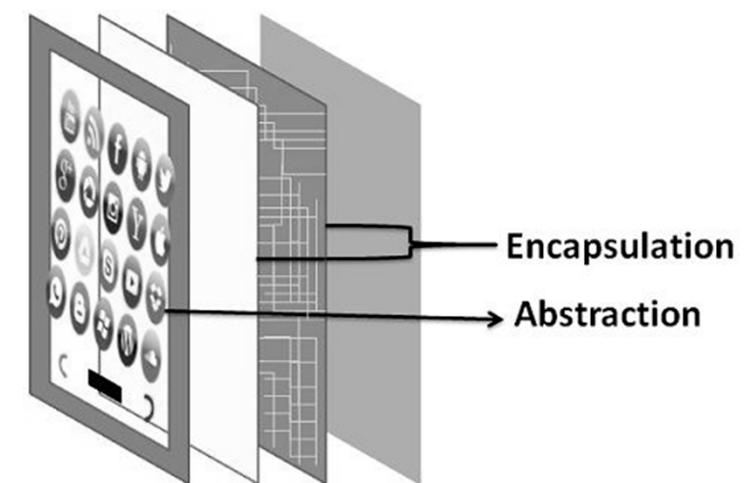
Abstractizare: ceea ce ar trebui făcut.

Încapsulare: modul în care ar trebui făcut.

Aplicabilitate

Abstractizare: În timpul proiectarii.

Încapsulare: În timpul implementarii.

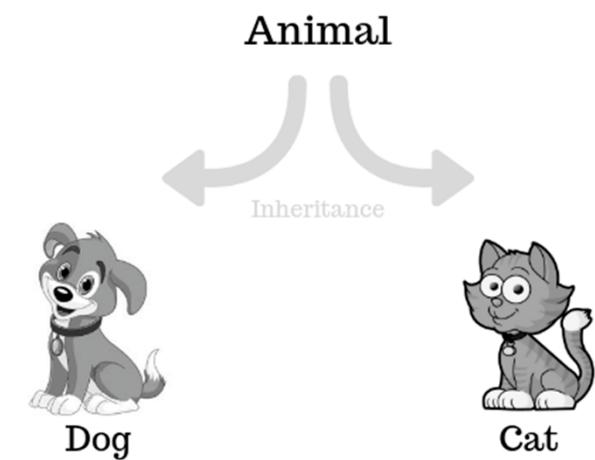
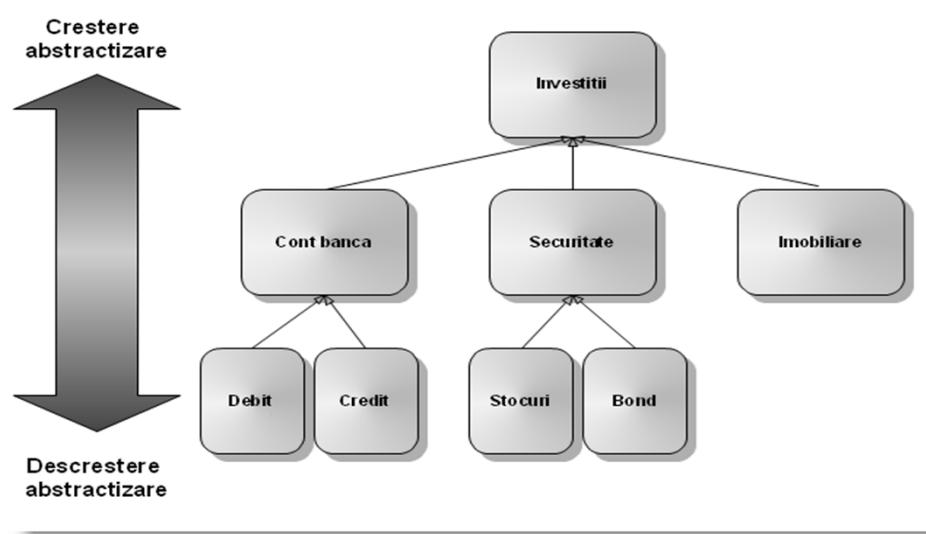


Ce este polimorfismul ?

- descrie situațiile în care “ceva” se întâmplă în mai multe forme diferite.
- În informatică, polimorfismul descrie conceptul accesării de obiecte de diferite tipuri prin aceeași interfață.



Ce este mostenirea ?



Ierarhia reprezinta o ordonare a abstractizarii.

Concepțe de bază ale POO

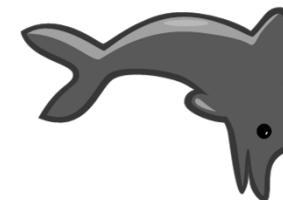
- Obiect
- Clasa
- Atribut
- Operatie
- Componenta / modularitate
- Mostenire
- Incapsularea
- Polimorfism



Obiecte (1)

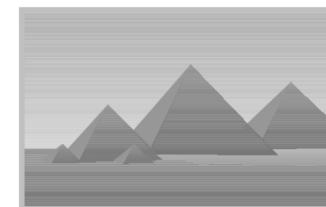
- Un obiect reprezinta o entitate (fizica, conceptuala sau soft)

1. Entitate fizica



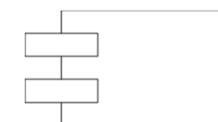
Peste

2. Entitate conceptuala



Piramide

3. Entitate software



Componenta
(subsistem)

Obiecte (2)

- Definitie formală: un obiect este un concept (real sau abstract) cu *frontiere* bine definite și posibilitate de acțiune.
- Se caracterizează prin:
 - Evoluție – influențată de istoricitatea obiectului (d.e. automatul pentru bauturi); repreză modul în care obiectele acionează / reacționează în termeni de modificare de stare și schimb de mesaje
 - Stare: orice obiect are o stare, i.e. ocupa un spațiu fizic de memorie
 - Identitate unică: “propietatea unui obiect ce îl face distinct față de un alt obiect” (Copeland)

ELEMENTS OF AN OBJECT

Attribute/ state	{ <ul style="list-style-type: none">• properties used to define characteristics
Behaviour	{ <ul style="list-style-type: none">• means the object can perform actions & can have actions performed on it.
Identity	{ <ul style="list-style-type: none">• means the object can be called & used as a single unit.

4

Starea unui obiect

- Starea unui obiect reprezinta una din conditiile posibile de existenta ale acestuia
- In mod normal, starea unui obiect se schimba in timp
- Este reprezentata de:
 - ◆ Valorile atributelor
 - + - ◆ Instantele relatiilor (*relationship instances*)

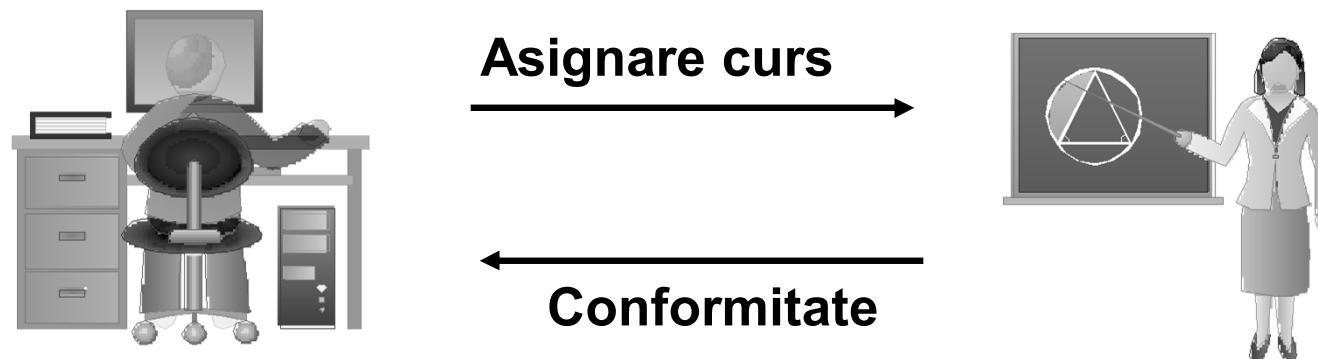


Nume:	Ion Popescu
ID:	123456789
Data angajarii:	01.01.01

Evolutia unui obiect (1)

- Evolutia determina modul in care obiectele (re)actioneaza la solicitarile altor obiecte.
- Evolutia unui obiect este reprezentata de totalitatea raspunsurilor la setul de mesaje primite (i.e. totalitatea operatiilor pe care un obiect le poate realiza)

Observatie: orice obiect are o identitate unica (chiar daca starea sa este identica cu cea a unui alt obiect)

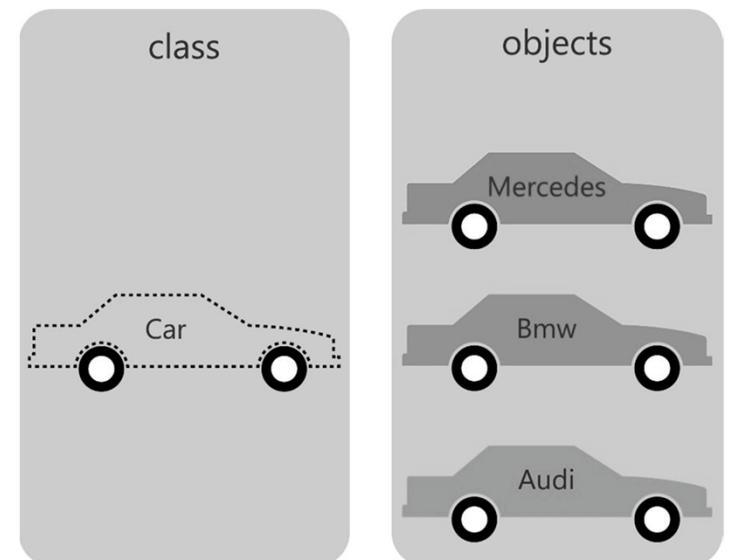


Evolutia unui obiect (2)

- Operatii
 - De modificare: altereaza starea unui obiect
 - De selectie: operatia acceseaza starea unui obiect dar NU o altereaza
 - De iteratie: operatia prin care se permite ca toate partile unui obiect sa poata fi accesate intr'o ordine predefinita
- **Constructor:** crearea unui obiect si/sau initializarea starii acestuia
- **Destructor:** elibereaza starea unui obiect si/sau il distrugе

Clase

- Ansamblul obiectelor cu aceeasi semantica, proprietati (attribute) si comportament comun
- Un obiect reprezinta *instanta* a unei clase
- Gruparea obiectelor in clase reprezinta rezultatul unei operatii de abstractizare:
 - Evidenierea caracteristicilor comune
 - Eliminarea elementelor nespecifice



Clasa – exemplu



Masina

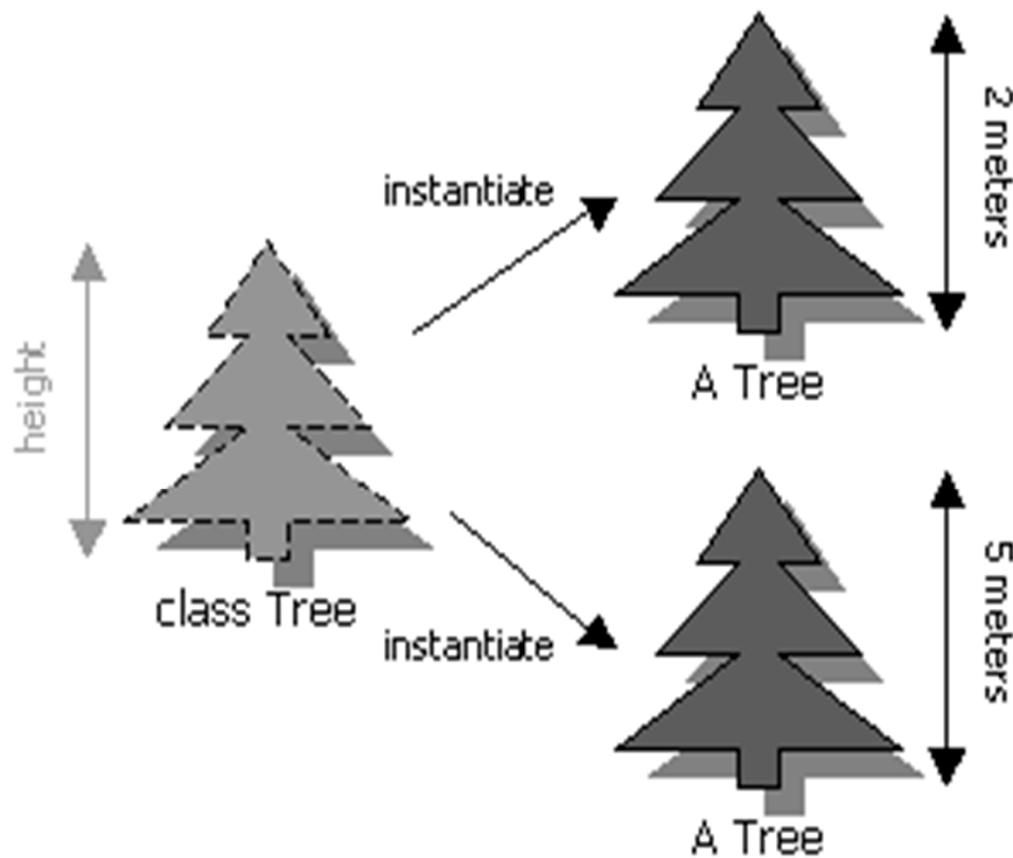
- Atribute:

- ➔ Model
- ➔ Serie_motor
- ➔ An_fabricatie

- Operatii:

- ➔ Deplasare
- ➔ Reparare
- ➔ Opritre

Clase & obiecte – exemplu



Clasa “Arbore”

Obiecte (instantieri ale clasei)

arbori, cu atribut inaltime

Fig. 1: Instantiating two Trees from the Tree class

Tipuri de clase

- Clasa abstracta.
 - O clasa *incompleta* care defineste elementele comune a mai multor clase
 - Fara instantiere.
- Clasa concreta
 - O clasa completa
 - Descrie complet un anumit concept
 - Poate avea instantieri

A B S T R A C T C L A S S V E R S U S C O N C R E T E C L A S S

ABSTRACT CLASS

A class declared with an abstract keyword which is a collection of abstract and non-abstract methods

Programmer cannot create objects using an abstract class

An abstract class can have unimplemented methods

CONCRETE CLASS

A class that allows to create an instance or an object using the new keyword

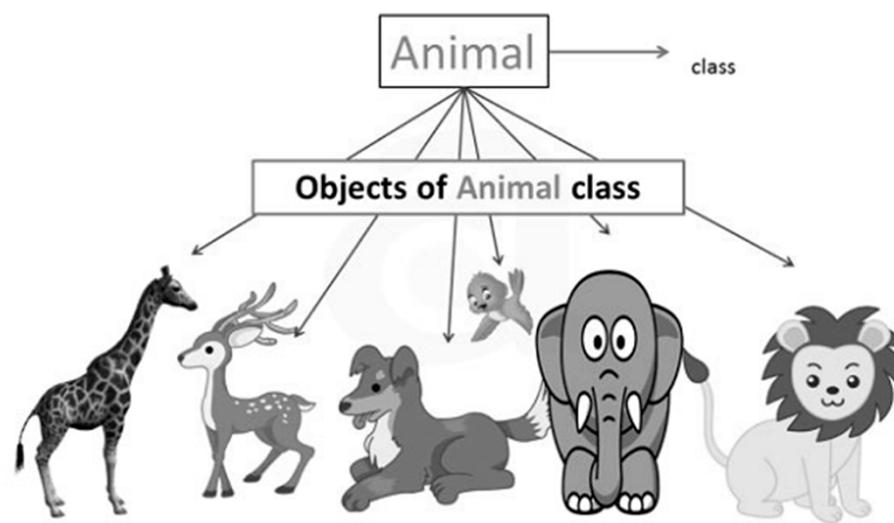
Programmer can create objects using a concrete class

All methods in a concrete class are implemented

Visit www.PEDIAA.com

Relatii intre obiecte si clase

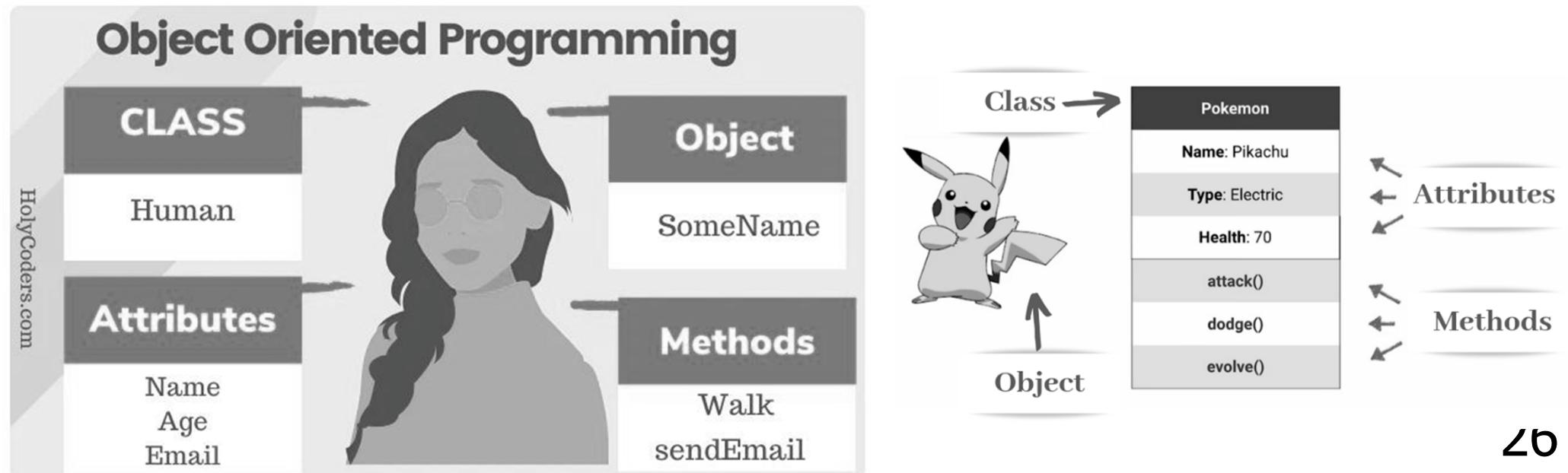
- O clasa reprezinta o definitie abstracta a unui obiect
 - Defineste structura si evolutia pentru fiecare obiect ce apartine clasei respective
 - Poate fi un *template* pentru crearea de obiecte
- Obiectele pot fi grupate in clase



Atribute & operatii

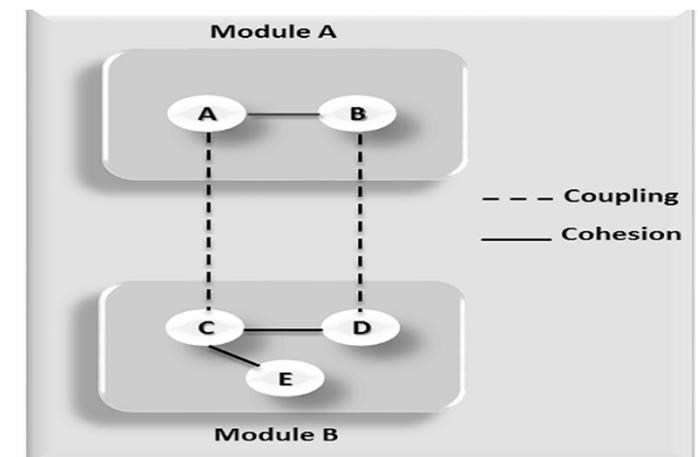
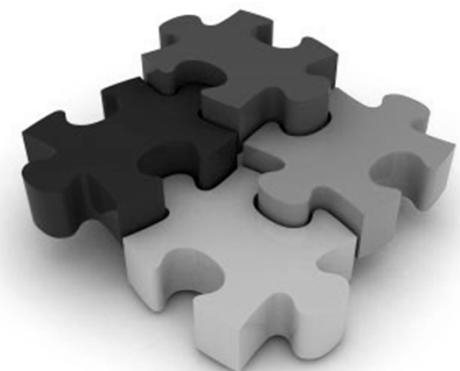
Atributele reprezintă diferitele caracteristici ale clasei. Ele pot fi, de asemenea, denumite caracteristici ale clasei și pot diferenția o clasă de alte clase.

Operatiile (metodele) sunt responsabile cu definirea / modificarea comportamentului unei clase și al obiectelor aferente acesteia.



Modularitate

- Modularitatea reprezinta procesul de descompunere a unei probleme (program) într'un set de module, astfel încât să reducă complexitatea generală a problemei.
- Modularitatea (Booch): „Modularitatea reprezinta proprietatea unui sistem care a fost descompus într-un set de module coeziive și slab cuplate.”
- Modularitatea este intrinsec legată de încapsulare. Modularitatea poate fi vizualizată ca o modalitate de a mapa abstractiile încapsulate în module reale, fizice, cu o coeziune ridicată în cadrul modulelor, iar interacțiunea sau cuplarea lor între module trebuie să fie scăzută.

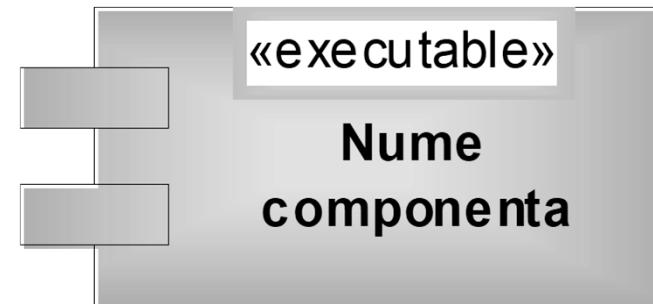
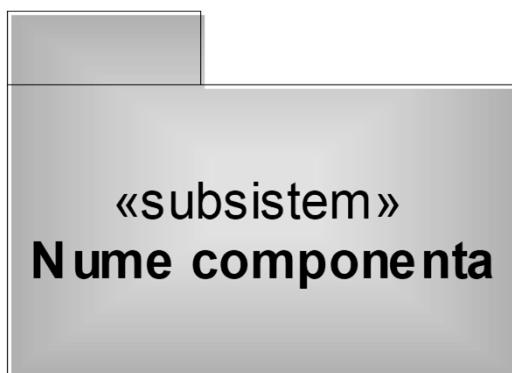


Caracteristici modularitate

- Exista programe (d.e. Smalltalk) ce nu contin module (i.e. clasa reprez. singura unitate fizica de descompunere)
 - Scopul modularizarii: reducerea costurilor prin accesarea independenta a fiecarui modul
- In general clasa + obiect se regasesc in module

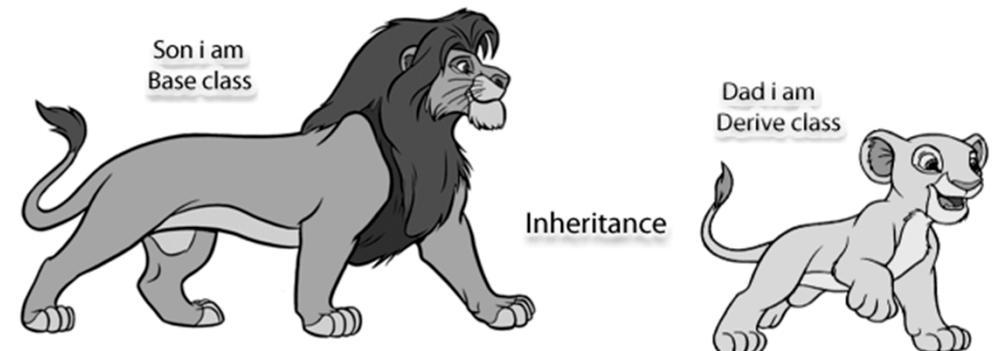
Componenta

- Partea unui sistem ne-triviala si inlocuibila ce indeplineste o functie bine precizata in contextul unei arhitecturi bine definite.
- O componenta poate fi:
 - ➔ Cod sursa
 - ➔ O structura executabila

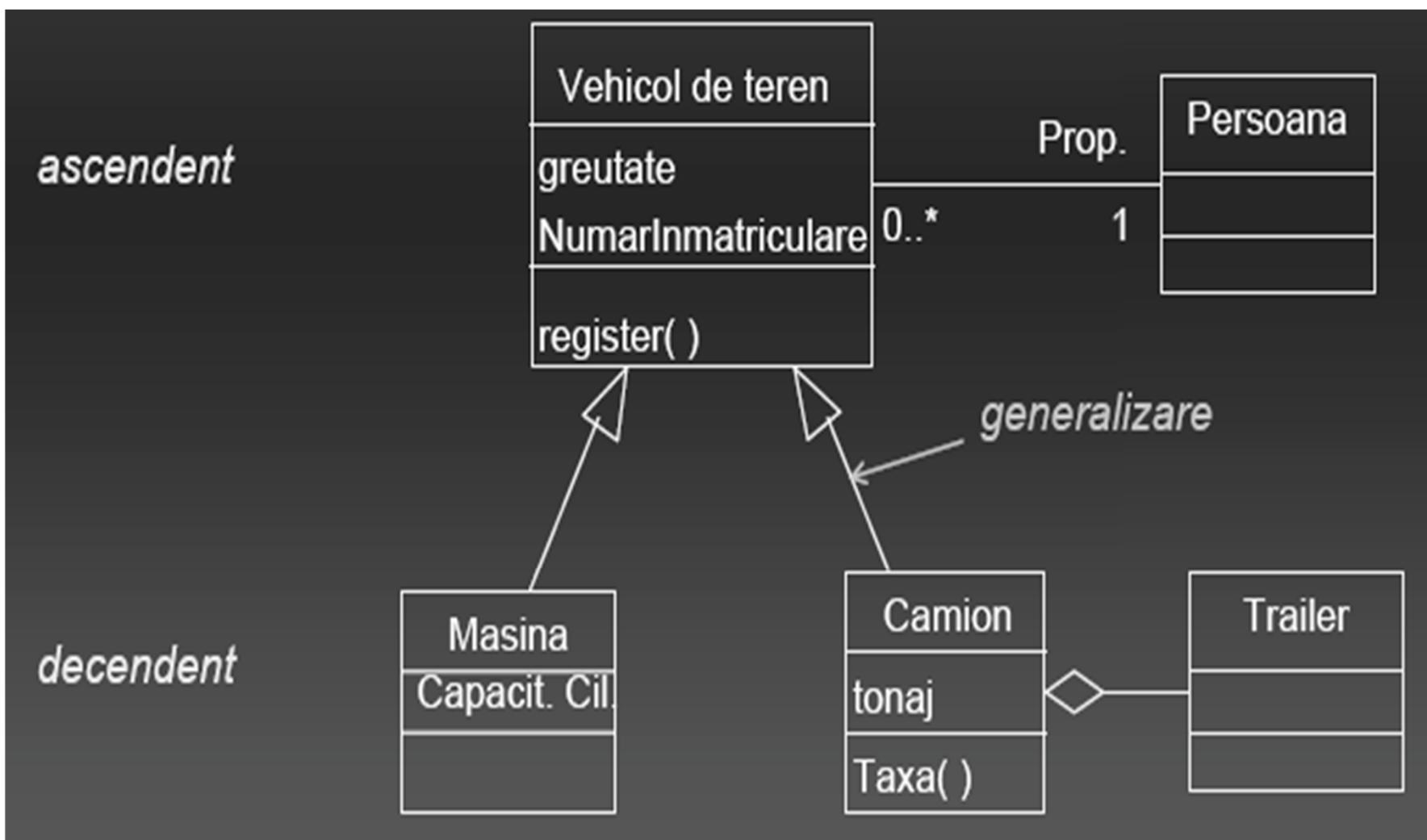


Mostenire

- Proprietatea de partajare a atributelor si operatiilor unei / unor clase de catre alta / alte clase in cadrul relatiilor ierarhice, diferențele ramanand neschimbate
- Prin mostenire se pot exprima relatii de:
 - Generalizare
 - Specializare
 - Clasificare
 - Aproximatie
 - Evolutie
- Tipuri de mosteniri:
 - Simple
 - Multiple

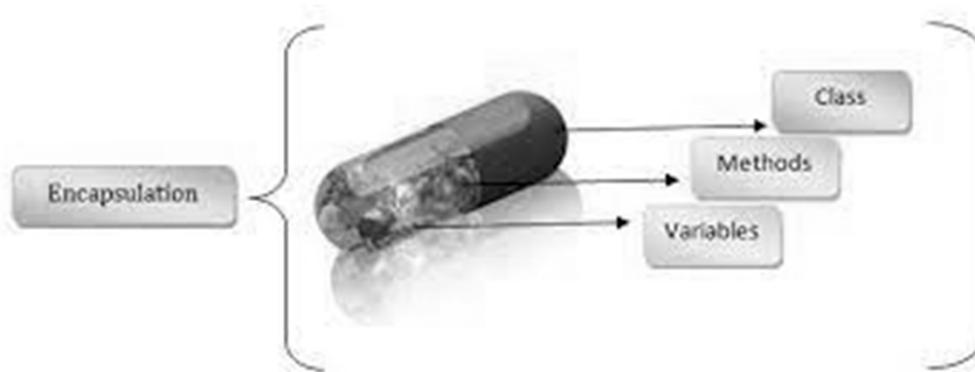


Generalizare



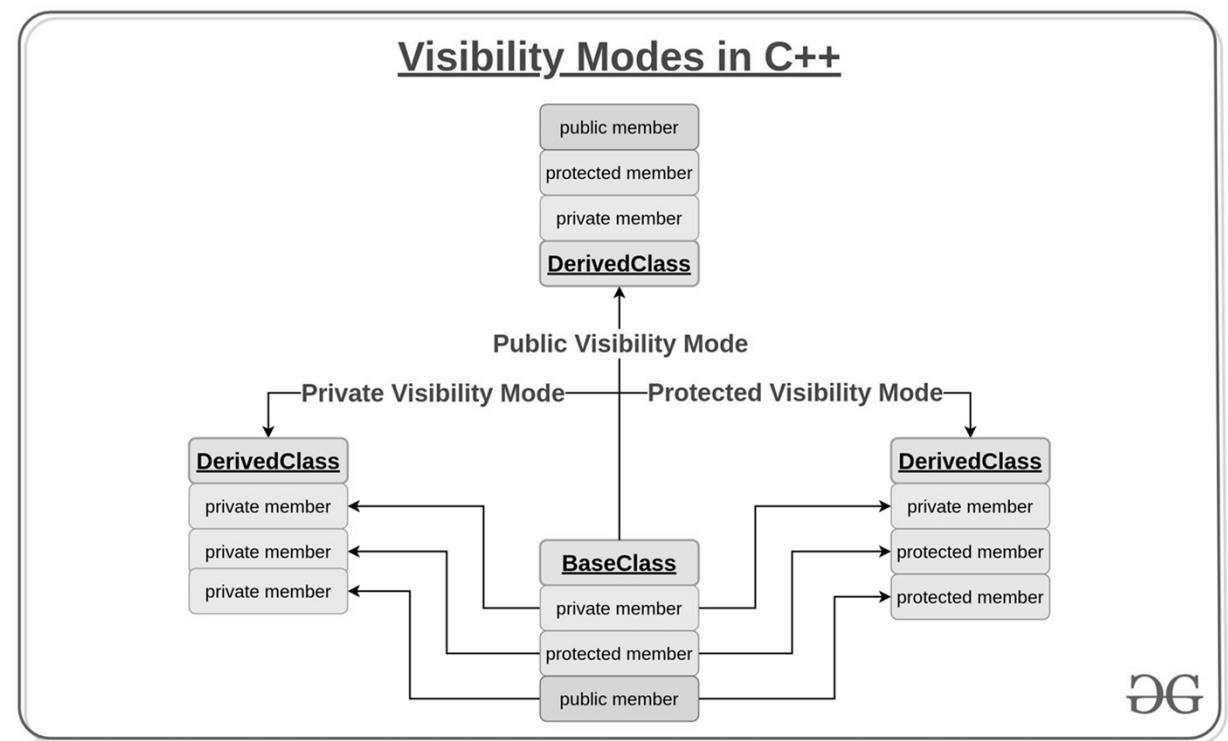
Incapsularea

- Gruparea detaliilor de implementare într'o zonă ascunsă, inaccesibilă din exterior
- Consecință: reducerea gradului de interdependență dintre elemente



Incapsularea

- Vizibilitatea claselor este structurata pe trei nivale:
 - Public → toti!
 - Privat → proprie unei clase
 - Protejat → vizibilitate la nivelul subclasei

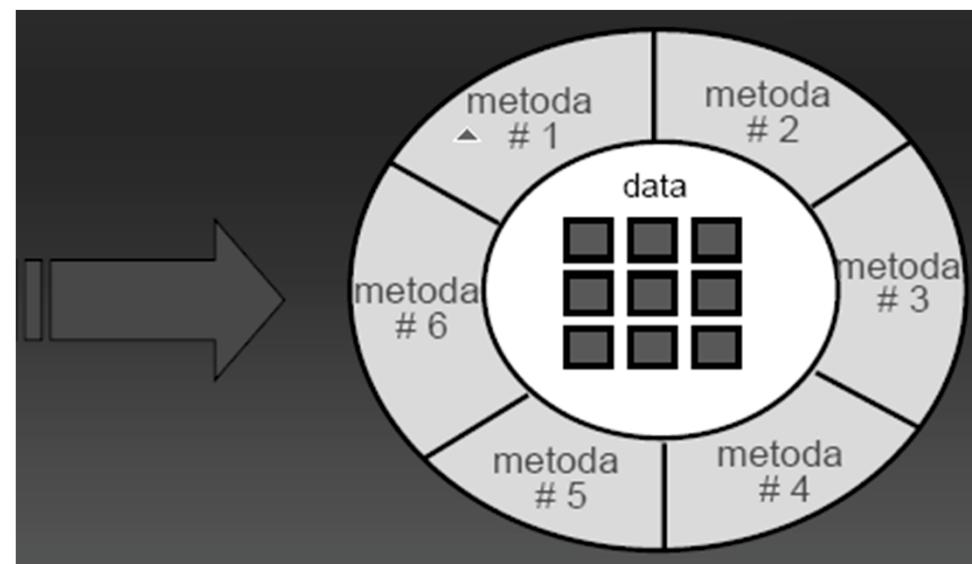


Principiile lui David Parnas

1. Dezvoltarea de componente software trebuie sa furnizeze utilizatorilor toata informatia necesara spre o buna utilizare a componentelor, dar **NICI** o alta informatie in plus.
2. Dezvoltatorul de componente software trebuie sa primeasca toata informatia necesara in vederea finalizarii cu succes a responsabilitatilor sale referitoare la componente, dar **NICI** o alta informatie in plus.

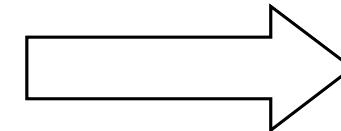
Incapsulare – consecinte

- Obiectele incapsuleaza atat datele cat si procedurile aferente manipularii datelor

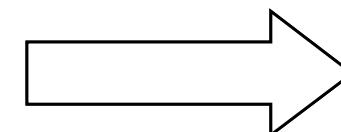


Polimorfism (1)

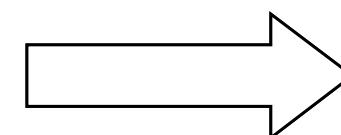
- Capacitatea unui operator de a se aplica obiectelor din clase diferite



Deplasare



Deplasare

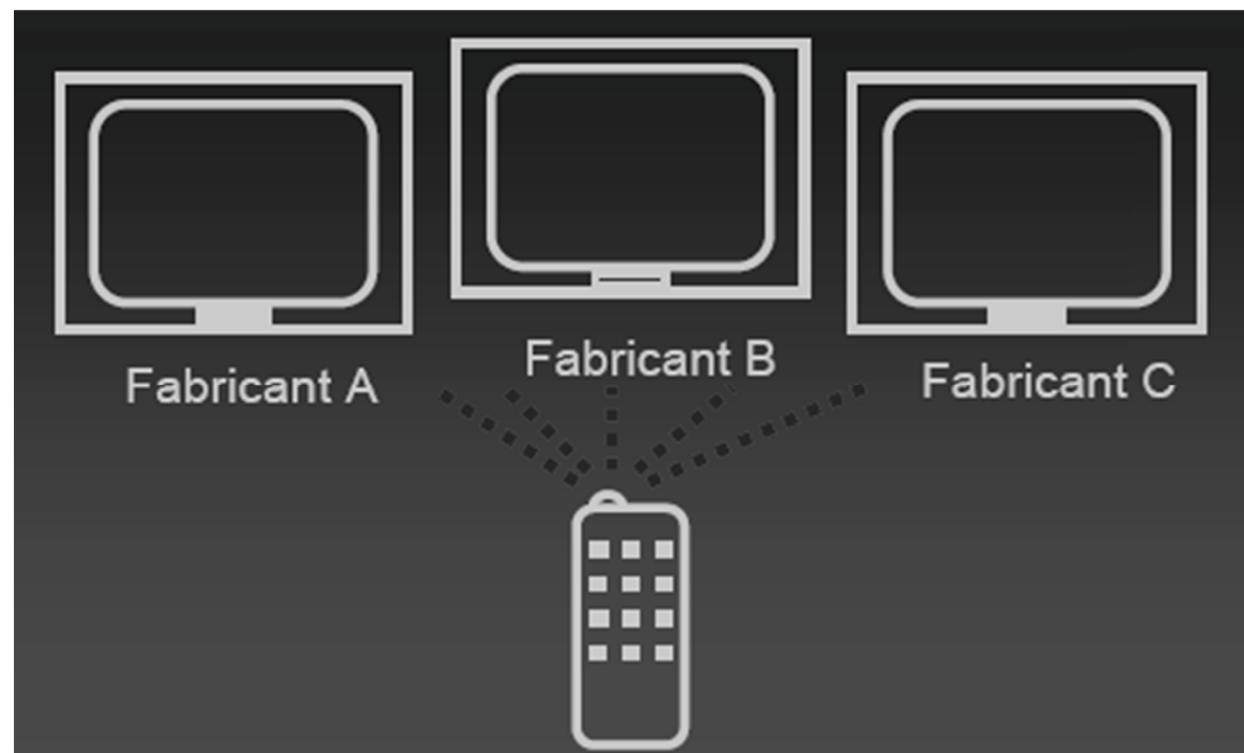


Deplasare

Polimorfism (2)

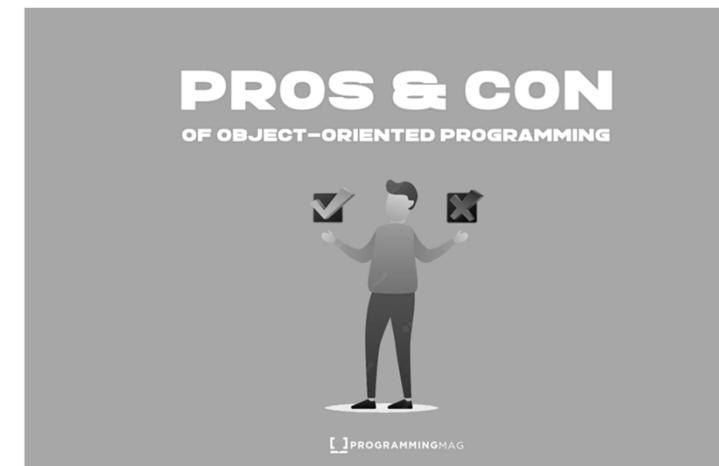
sau

- Abilitatea unui operator de a masca implementari diferite in spatele unei interfete unice



Puncte *tari* ale POO

- O singura paradigma
- Un singur limbaj utilizat de utilizatori, analisti, proiectanti, etc.
- Faciliteaza re-utilizarea codului si structurile arhitecturale
- Modeleaza mai corect lumea reala:
 - Descriere mai precisa a fluxurilor de date si a proceselor aferente
 - Descompuneri naturale ale structurilor mari
- Usor de inteles si de intretinut
- Stabilitate
- O mica schimba la nivelul cerintelor nu conduce la modificari profunde in sistemul aflat in faza de dezvoltare



Puncte *slabe* ale POO

- Lungimea programelor dezvoltate folosind limbajul OOP este mult mai mare decât abordarea procedurală. Deoarece programul devine mai mare, este nevoie de mai mult timp pentru a fi executat, ceea ce duce la o execuție mai lentă a programului.
- Nu se poate aplica OOP peste tot, deoarece nu este un limbaj universal. Se aplică numai atunci când este necesar. Nu este potrivit pentru toate tipurile de probleme.
- Procesul de gândire implicat în programarea orientată pe obiecte poate să nu fie natural pentru unii oameni.

