

Programare orientată pe obiecte

Curs 1



Curs

Ș.l.dr.ing. Mihaela CRIȘAN-VIDA



mihaela.vida@upt.ro

Sala B019

Laborator

Drd.ing. Adina NIȚULESCU

Ș.l.dr.ing. Norbert Gal-Nădășan

Sala B019

Curs

Luni 14 – 16 A117

Marti 12 – 14 ASPC

Laboratoarele: OBLIGATORII!

Laboratoare

Marti 14- 16 sala B019

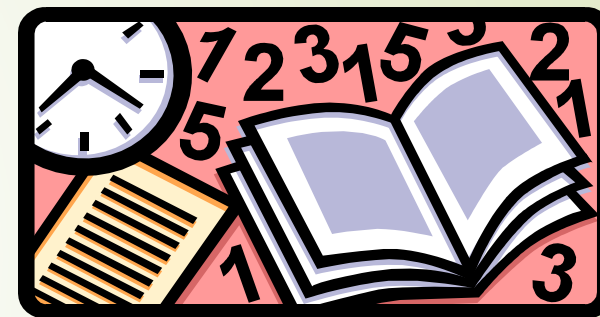
Miercuri 10 – 12 sala B019

12 – 14 sala B019

14 – 16 sala B019

Vineri 8 – 10 sala B019

10 – 12 sala B019

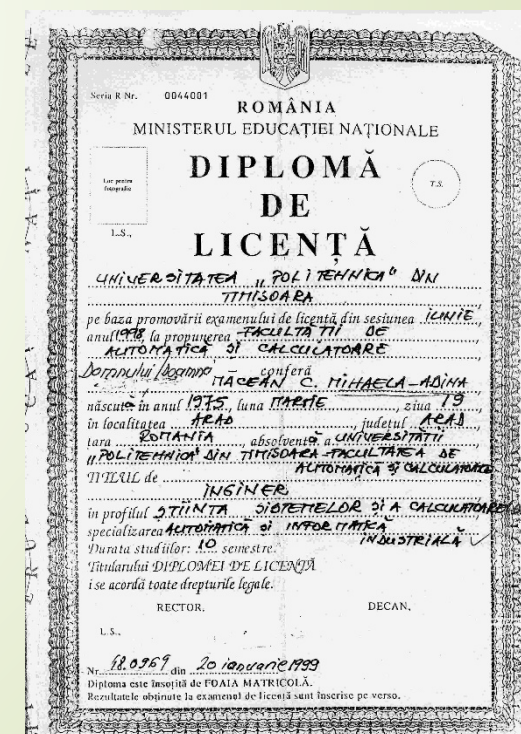


Direcții de dezvoltare (Didactice)

- Visual Studio.NET
- Baze de date ORACLE (lucru cu Oracle Developer)
- Aplicații mobile -> dinamică mare
- Programarea sistemelor încorporate (embedded)
(ex. la Continental, Hella)
- UML
- ECLIPSE (IBM) – JAVA
- programare INTERNET (aplicații WEB, SOA, cloud, JAVASCRIPT/HTML5 etc.)

Locuri de muncă (majoritatea foarte bine plătite)

**NOKIA, CONTINENTAL, IBM, HELLA, OCE, SAGUARO,
ENEL, Eta2u, Visma, Berg Computers, Lasting etc**



Obiective ale cursului

- ❑ Să furnizeze cunoștințe și deprinderi practice pentru Programarea Orientată pe Obiecte (cu accent pe C++) pentru a putea ulterior aborda la nivel profesional programarea folosind medii profesionale (ex. Visual C++ etc.)
- ❑ Să furnizeze cunoștințe elementare despre Proiectarea Orientată pe Obiecte, inclusiv elemente de UML și sabloane de proiectare



Condiții de evaluare

- **Activitatea pe parcurs** (teste, interes, calitatea activității)
– pentru **fiecare** laborator
- **Examen** -> **Teste grilă** cu întrebări (“teorie” – dar cu accente practice – vor fi fragmente de cod) și de completat cod sursă.



Structura cursului. Bibliografie

Cursul se referă la standardul

- Aspecte generale C++ - trecere în revistă
- Aspecte specifice C++ - clase și obiecte, matrice, pointeri, referințe, supraîncărcarea, moștenirea, polimorfismul, funcțiile virtuale, operații de intrare/ieșire, tratarea excepțiilor
- Studii de caz
- Proiectarea Orientată pe Obiecte; Tipare, colecții, UML, șabloane / patterns (fișier pdf)



Curs 1 - Programare orientată pe obiecte



Capitolul 1. Paradigma programării orientate pe obiecte

Drumul spre tehnologiile OO

proiectarea structurată - abordare top-down, structuri de control fundamentale (blocul de instrucțiuni, decizia, bucla (PASCAL) - accent pe analiza funcțională

versus *proiectarea orientată pe obiecte*

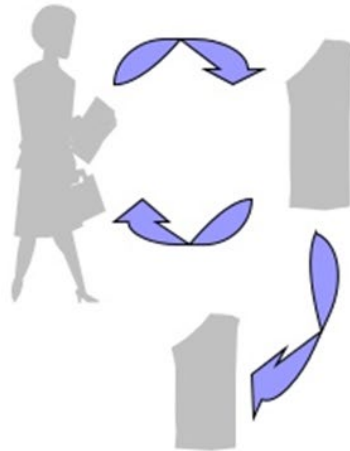
DE CE apar probleme cu proiectarea structurată ? :

- crearea unui program nou ia **mult timp** și deci este costisitoare; prin schimbări minore în cerințele utilizatorului, fragmente mari de program trebuie rescrise
- din pricina timpului îndelungat pentru dezvoltare, un program nou apare pe piață poate **prea târziu**, după ce au apărut programe mai performante
- programele noi conțin **erori** (bugs) care se manifestă în funcționare
- s-ar putea ca programul nou să nu corespundă cerințelor utilizatorului întrucât acesta și proiectantul au avut puncte de vedere diferite și **nu au înțeles** același lucru prin specificațiile stabilite de comun acord

Capitolul 1. Paradigma programării orientate pe obiecte

Procedural vs. Object-Oriented

■ Procedural



Withdraw, deposit, transfer

■ Object Oriented

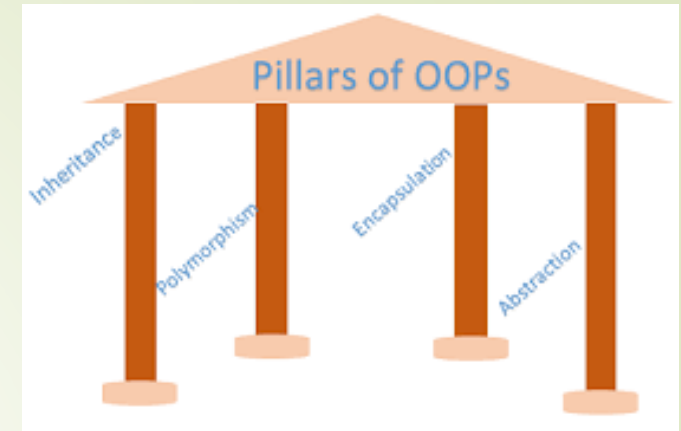


Customer, money, account

Tendința actuală

- împingere a efortului de proiectare super-calificată cât mai departe de utilizatori (în medie 80% dintr-o aplicație (chiar peste 90% câteodată) constă din elemente de interfață ale căror specificații sunt identice cu cele ale altor aplicații)
- a apărut metodologia orientată spre obiecte (***Object-Oriented* - OO**) :
 - “inversează” metodologiile funcționale, fragile mai ales la reutilizare (la mici schimbări ale cerințelor trebuie restructurat masiv întreg sistemul).
 - se focalizează pe obiectele din domeniul aplicației, adică entitățile independente care modelează realitatea; la aceste obiecte se stabilesc ulterior proceduri de acces.
 - termenul “orientat spre obiecte” înseamnă organizarea programului ca o colecție de obiecte, fiecare înglobând structuri de date și având asociat un anumit comportament.

Abordarea Orientată pe Obiecte (AOO)



A. obiect - entitate care încorporează atât structuri de date (atribute) cât și comportament (operații)

Exemple:

Caracteristici

identitate : obiectul are o identitate discretă, care se distinge dintre alte entități.

*căciula studentului
o fereastră deschisă pe un calculator
un triunghi desenat pe hârtie*

clasificare: obiectele cu aceleași atribute și operații se grupează în clase. Fiecare obiect poate fi considerat ca o instanță a unei clase.

*căciulă
fereastră
triunghi*

polimorfism: aceeași operație (cu același nume) poate să aibă comportament diferit în clase diferite

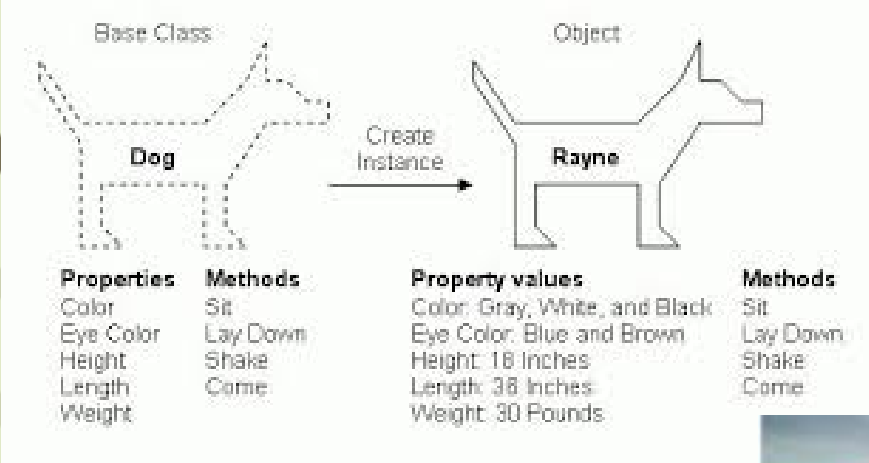
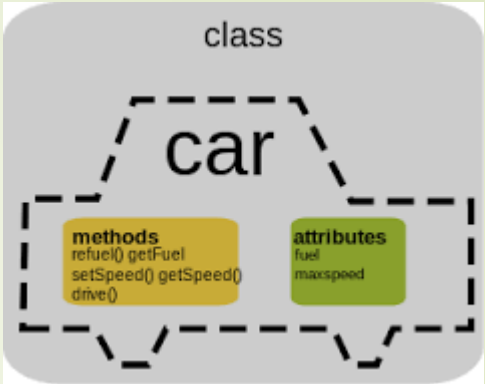
*a muta: o căciulă, o fereastră, un
triunghi*




Implementarea concretă a unei operații într-o anumită clasă se numește metodă (method)

moștenire: atributele și operațiile se transmit de-a lungul claselor

triunghi isoscel, triunghi echilateral

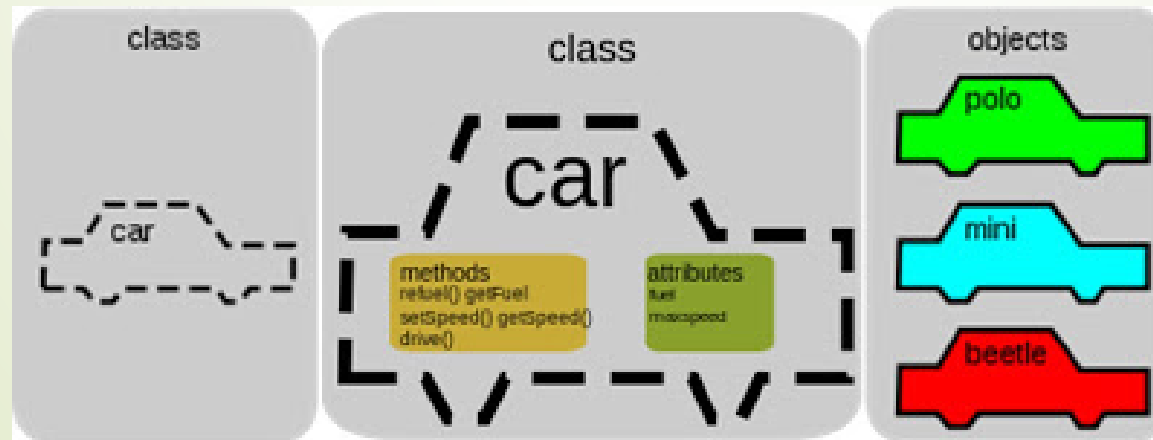
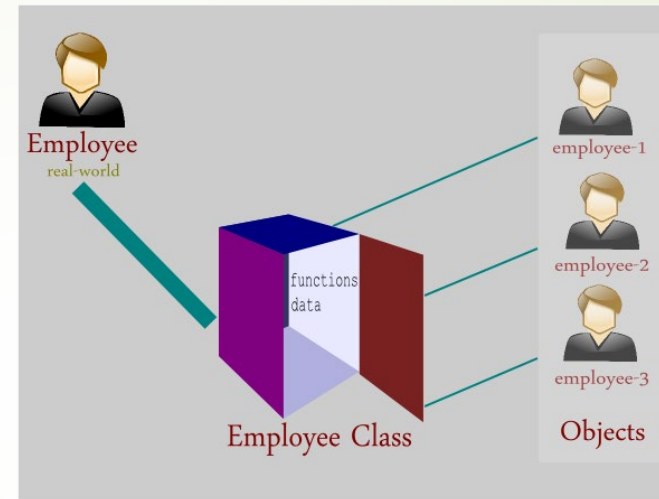
Obiect



		
Car	Auto-Rickshaw	Bike
Properties <ul style="list-style-type: none">• Color : Orange• Wheels : 4• Doors : 2	Properties <ul style="list-style-type: none">• Color : Green/Yellow• Wheels : 3• Doors : 0	Properties <ul style="list-style-type: none">• Color : Blue• Wheels : 2• Doors : 0
Methods <ul style="list-style-type: none">• Steer• Accelerate• Brake	Methods <ul style="list-style-type: none">• Steer• Accelerate• Brake	Methods <ul style="list-style-type: none">• Steer• Accelerate• Brake

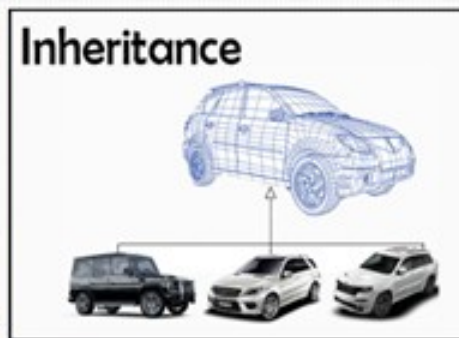


Identitate, clasificare

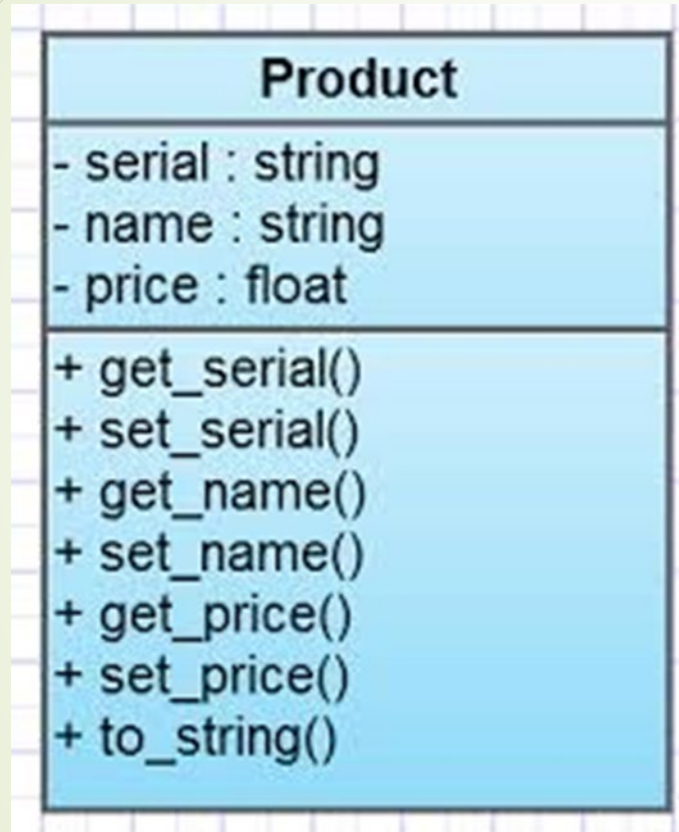


Basic OOP Concepts

Concepts >



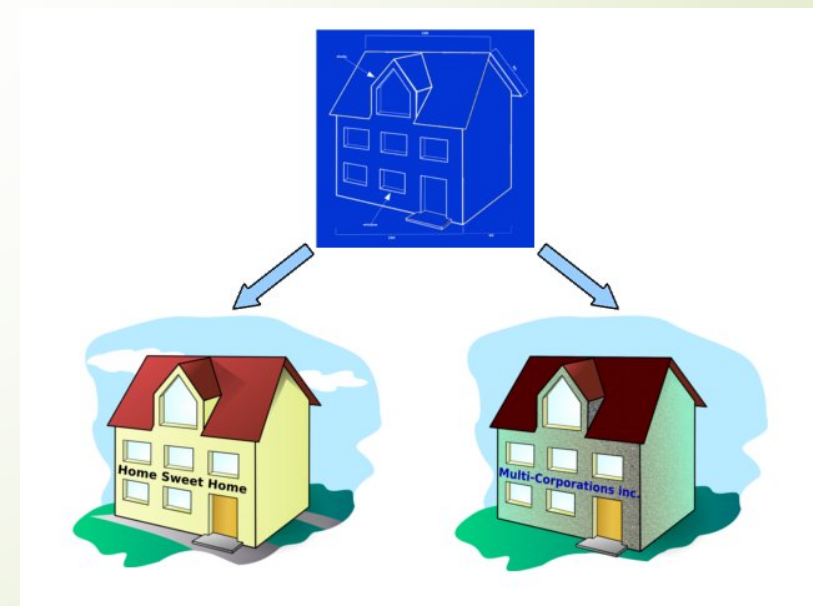
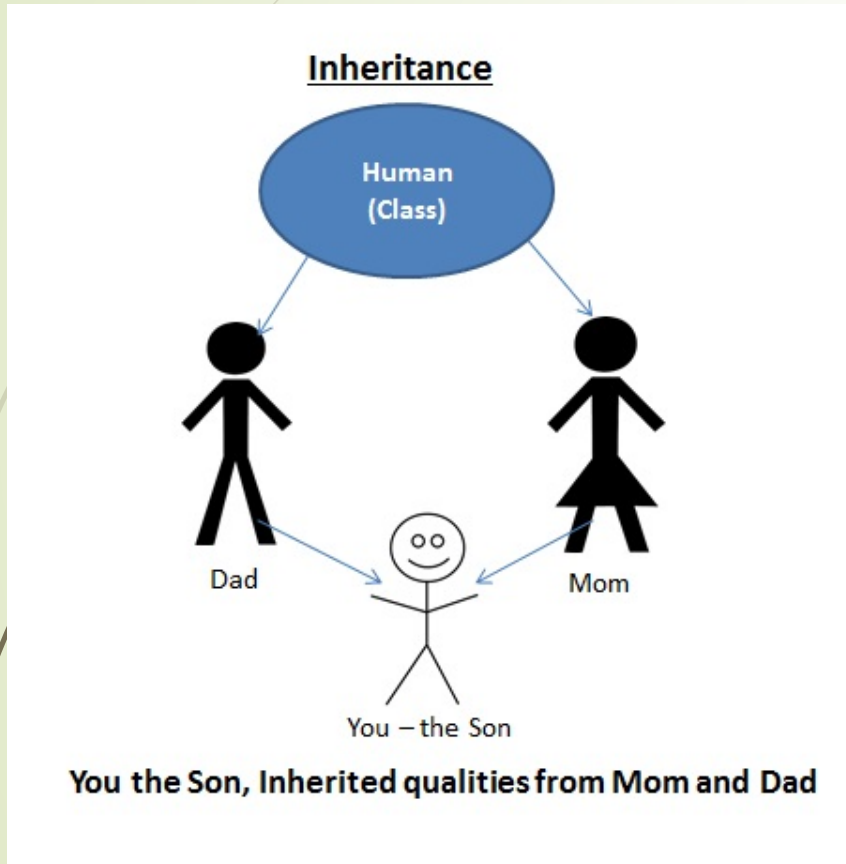
Clasa



Polimorfism



Moștenire



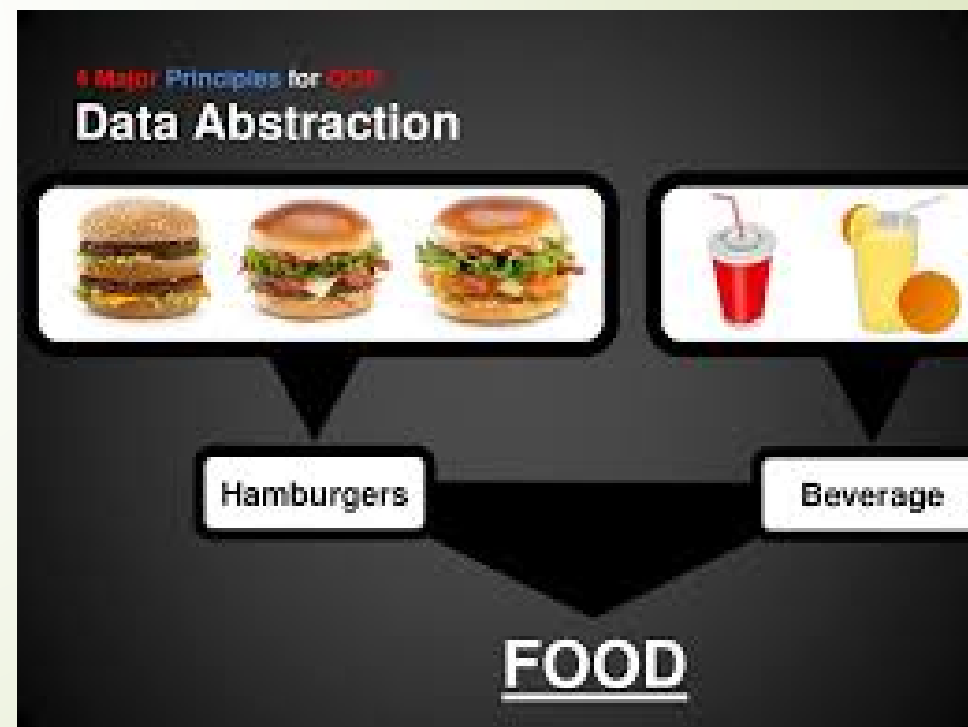
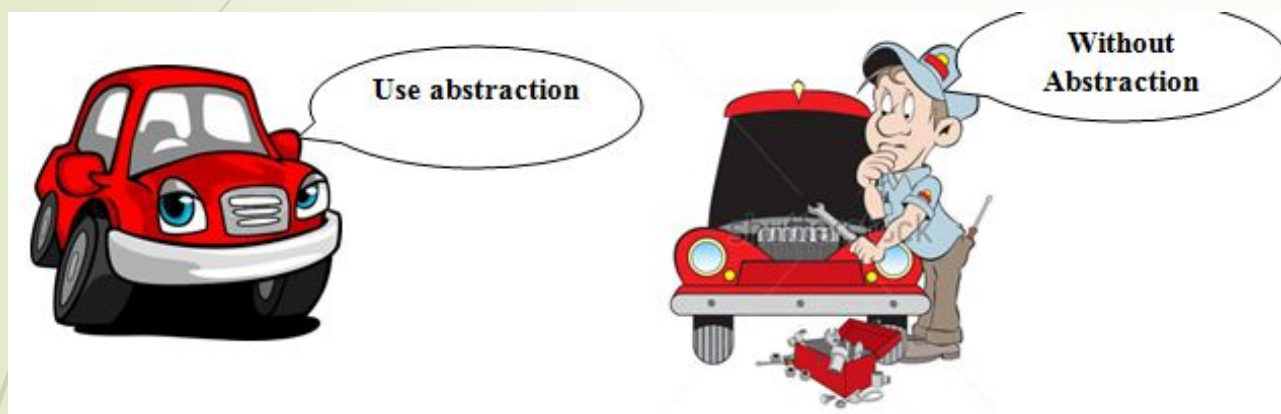
Concepte de bază

- **abstractizarea**: focalizarea pe aspectele esențiale ale unei entități. Accentul, pentru un obiect, se pune pe ce este acesta și pe ce trebuie să facă, înainte de a stabili concret detaliile de implementare. De aceea, etapa esențială în crearea unei aplicații orientate spre obiecte este **analiza** și nu implementarea, care trebuie să devină aproape mecanică.
- **încapsularea (ascunderea) informației**: separarea aspectelor externe ale unui obiect, care sunt accesibile altor obiecte, de aspectele de implementare internă, care sunt ascunse celorlalte obiecte. Utilizatorul poate accesa doar anumite atribute și operații ale obiectului (**publice**), alte operații îi rămân inaccesibile (**private**).

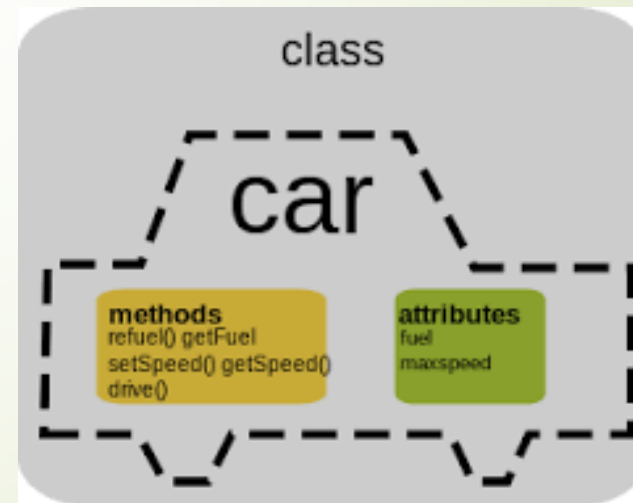
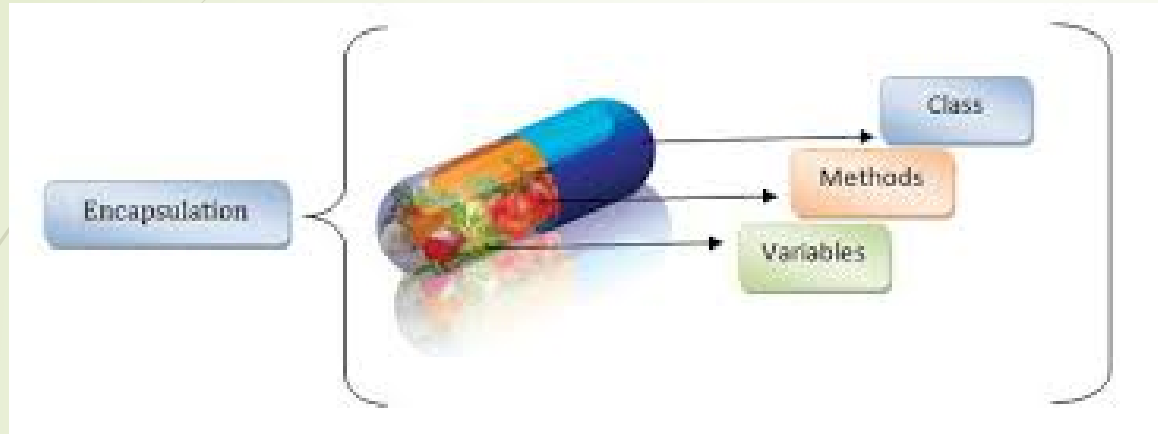
Concepte de bază

- **“împachetarea” datelor și a comportamentului în același obiect**: un obiect conține atât structuri de date cât și operații și este utilă întrucât vom ști întotdeauna cărei clase aparține o metodă, chiar dacă există mai multe operații cu aceeași denumire, în clase diferite
- **partajarea (sharing)**: transmiterea acelorași structuri de date și respectiv operații, de-a lungul unor clase dintr-o ierarhie de clase, și aceasta are un efect benefic asupra economisirii de spațiu
- **accentul pe structura de obiect, nu pe cea de procedură**: în tehnicile orientate spre obiecte, accentul cade pe înțelegerea sistemului din punctul de vedere al descompunerii acestuia în entități (obiecte) și al stabilirii relațiilor dintre acestea, deci pe ce este un obiect

Abstractizarea






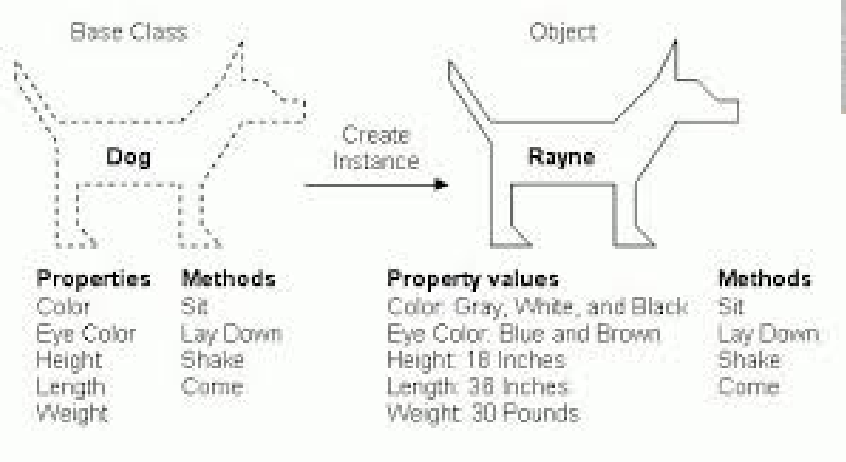
Încapsularea



Curs 1 - Programare orientată pe obiecte



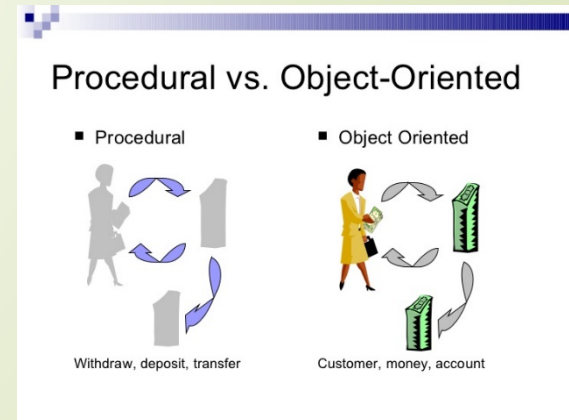
		
Car	Auto-Rickshaw	Bike
Properties <ul style="list-style-type: none">Color : OrangeWheels : 4Doors : 2	Properties <ul style="list-style-type: none">Color : Green/YellowWheels : 3Doors : 0	Properties <ul style="list-style-type: none">Color : BlueWheels : 2Doors : 0
Methods <ul style="list-style-type: none">SteerAccelerateBrake	Methods <ul style="list-style-type: none">SteerAccelerateBrake	Methods <ul style="list-style-type: none">SteerAccelerateBrake



Metoda OO versus metode clasice

- dimensiuni tradiționale (comportamentale) ale programelor: corectitudine, tratarea erorilor și excepțiilor, eficiența, portabilitatea și independența de periferice.
- **dimensiuni non-comportamentale:**
 - **mentenabilitatea:** proprietatea unui program de a fi ușor de schimbat pentru a fi eliminate defecte)
 - **extensibilitatea:** proprietatea ca un program să poată fi modificat pentru a trata noi clase de intrări).

Paradigma OO este potrivită pentru a ajuta la îmbunătățirea celor două calități non-comportamentale



De ce C++?

Foloase dacă știi C++ :

- poți învăța ușor Visual C++, dar și alte medii similare;
Visual C++ este foarte folosit
- cea mai bună compatibilitate cu C
- poți învăța ușor C#, JAVA;
- înțelegi bine conceptele legate de programarea orientată pe obiecte și deci inclusiv programarea în oricare limbaj al platformei .NET
- găsești deci mai ușor de lucru, în locuri bine plătite



Capitolul 2. Limbajul C++. Generalități

➔ **ISTORIC:**

- creat de Bjarne Stroustrup în **1980** în laboratoarele Bell din Murray Hill, New Jersey (**C** cu clase)
- **1983** - numele stabilit la **C++**

Year	ISO/IEC Standard	Informal name
1998	14882:1998	C++98
2003	14882:2003	C++03
2011	14882:2011	C++11
2014	14882:2014	C++14
2017	14882:2017	C++17
2020	14882:2020	C++20
TBA	14882:2024	C++23
TBA		C++26

Caracteristicile POO permit programelor să fie structurate pentru a fi clare, extensibile și ușor de întreținut, fără pierderea eficienței

<https://en.cppreference.com/w/cpp/language/history>

