

Programare orientată pe obiecte

#1 JAVA Noțiuni introductive despre POO

Adrian Runceanu
www.runceanu.ro/adrian

Obiectivele disciplinei

Obiectivele generale ale disciplinei:

1. Cunoașterea noțiunilor privind algoritmi și proprietățile lor
2. Reprezentarea algoritmilor prin scheme logice, pseudocod, programe Java
3. Utilizarea platformei integrate de dezvoltare ECLIPSE

Obiectivele specifice:

1. Pentru curs:

- Cunoașterea limbajului Java
- Elaborarea de programe în Java
- Analiza și proiectarea algoritmilor cu ajutorul limbajului de programare Java

2. Pentru aplicații:

- Implementarea unor algoritmi într-un limbaj de programare utilizat pe scară largă - Java

Câteva precizări

Structura cursului

- ✓ **3 ore curs** – titular curs: Sef lucr. dr. Adrian Runceanu
- ✓ **2 ore laborator** – titular aplicații practice: Sef lucr. dr. Adrian Runceanu

Câteva precizări

Bibliografia necesară cursului:

1. Tudor Sorin, Vlad Hutanu - **Bazele programarii in Java**, Editura L&S Info-Mat, Bucuresti, 2005.
2. Doina Logofatu – **Algoritmi fundamentali in Java. Aplicatii** – Editura Polirom, Iasi, 2007.
3. Horia Georgescu – **Introducere in universul Java**, Editura Tehnica, Bucuresti, 2002.

Câteva precizări

1. Suport curs - varianta electronică disponibilă pe site-ul:

www.runceanu.ro/adrian

2. Îndrumar de laborator - varianta electronică disponibilă pe site pentru fiecare lucrare de laborator.

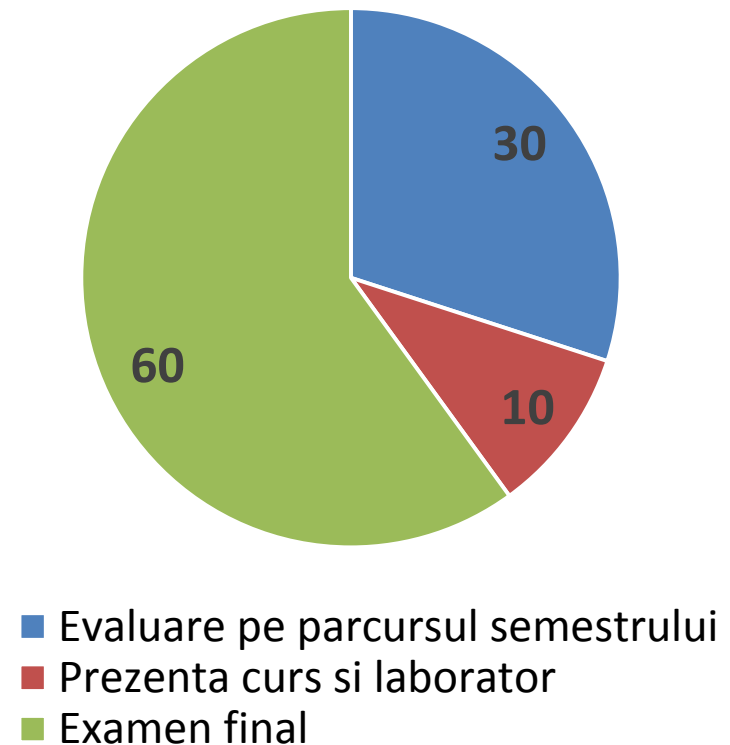
Notă: Actualizarea site-ului se face săptămânal.

Câteva precizări

Forme de examinare:

- Examen final = 60%
- Evaluare pe parcursul semestrului a activității de laborator = 30%
- Verificare finală lucrări de laborator = 10%

Procentaje evaluare



Conținutul cursului



1. Mediul de dezvoltare grafică **GREENFOOT**

2. Mediul de dezvoltare aplicații orientate-obiect **ECLIPSE**



3. Limbajul de programare **JAVA**



Curs 1

Noțiuni introductive despre programarea orientată pe obiecte



1. Noțiuni introductive despre programarea orientată pe obiecte

1.1. Obiecte și clase

1.2. Principii POO

1.1. Obiecte și clase

Obiecte. Clasificare, identitate, stare, comportament.

- În lumea în care trăim suntem obișnuiți să numim **obiecte** acele *entități care sunt caracterizate prin masă*, adică materie (exemplu: clădiri, animale, plante)
- Prin extensie, pot fi definite alte *obiecte fără masă*, care sunt mai degrabă concepte decât entități fizice de genul *formulelor matematice*.
- Tot prin extensie, *obiectele pot aparține unei lumi virtuale*, de genul unor evenimente ce pot avea loc, cum ar fi *apăsarea unei taste, producerea unei explozii într-un joc* sau *consultarea unui cont*.

1.1. Obiecte și clase

Definiție

Obiectul este conceptul de bază în programarea orientată obiect (OOP - Object Oriented Programming) care asociază datele împreună cu operațiile necesare prelucrării.

1.1. Obiecte și clase

Definiție

Datele sunt informații de structură descrise de o mulțime de attribute ale obiectului, iar **operațiile** acționează asupra atributelor obiectului și eventual, asupra altor obiecte.

➤ Modelul orientat obiect se bazează pe obiect.

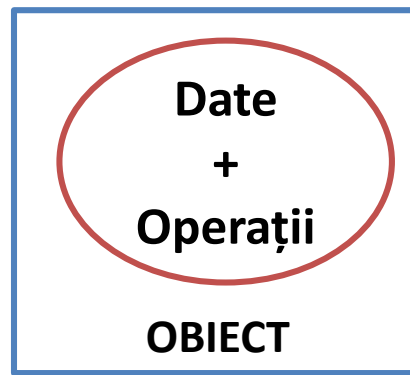


Figura 1. Modelul orientat obiect

1.1. Obiecte și clase

Un obiect este caracterizat de:

1. *Stare*

2. *Comportament*

3. *Identitate*

1.1. Obiecte și clase

1. Stare se referă la *elementele de date conținute în obiect și la valorile asociate acestora*.

Acestea formează atributele care descriu proprietățile unui obiect.

➤ De exemplu, atributele unui obiect „*carte*” pot fi *titlul, autorul, editura, număr pagini, anul apariției, ISBN și preț* atunci **starea unei cărți** ar putea fi următoarea:

titlu = Poezii

editura = Polirom

ISBN = 973–567–545–1

autor = Mihai Eminescu

număr pagini = 125

preț = 25

anul apariției = 2007

1.1. Obiecte și clase

2. Comportamentul este determinat de *operațiile pe care obiectul poate să le execute*.

Operațiile se deduc din acțiunile obiectului pe care trebuie să le îndeplinească.

- De exemplu, obiectul „*carte*” trebuie să furnizeze *acces la titlu, autor, an apariție*, etc. adică să furnizeze informații de stare a obiectului.
- În plus se poate:
 - modifica starea obiectului prin *modificare preț* sau altele,
 - obținerea unui grup de informații cum ar fi *titlu, autor și editură*,
 - se poate face un *calcul de TVA*, etc.

*obține_titlu, obține_autor, modifică_preț, calcul_TVA,
afișează_informații*

1.1. Obiecte și clase

3. *Identitate* este un identificator - **OID (Object Identifier)** care caracterizează unic obiectul, permițând să se construiască referiri spre obiect și să se facă distincția tuturor obiectelor într-o manieră non-ambiguă și independentă de starea lor.

- Astfel pot fi tratate distinct două obiecte ale căror attribute au valori identice

1.1. Obiecte și clase

De exemplu, obiectele:		
OID: carte1 titlu = Poezii autor = Mihai Eminescu editura = Polirom număr pagini = 125 anul apariției = 2007 ISBN = 973-567-545-1 preț = 25	OID: carte2 titlu = Geniu Pustiu autor = Mihai Eminescu editura = Polirom număr pagini = 140 anul apariției = 2005 ISBN = 973-565-545-2 preț = 20	OID: carte3 titlu = Marile speranțe autor = Charles Dickens editura = Univers număr pagini = 225 anul apariției = 2005 ISBN = 971-267-441 -1 preț = 35

1.1. Obiecte și clase

Definiție

Clasa este **conceptul de bază în POO** ce reunește *o colecție de obiecte care partajează aceeași listă de atribute informaționale și comportamentale*.

- O clasă va cuprinde definițiile datelor și operațiilor ce caracterizează obiectele de o anumită categorie.

De exemplu:

clasa cărți	clasa profesori	clasa materiale
clasa studenți	clasa imobile	etc.

1.1. Obiecte și clase

- Clasele de obiecte pot avea asemănări sau deosebiri datorită unor date și operații comune sau nu.
- De exemplu, între clasa *studenți* și clasa *profesori* pot exista una sau mai multe date identice, cum ar fi: *nume*, *vârsta*, *facultatea*, etc.
- Între clasa *studenți* și clasa *carti*, data(informația) *număr pagini* apare numai la clasa *carti*.

1.1. Obiecte și clase

- **Datele** definite într-o clasă se mai numesc **atribute**, iar **operațiile** se mai numesc **metode** sau **funcții-membru**.
- **Atributele** și **metodele** formează membrii unei clase.
- Fiecare clasă va avea identitate sau nume.

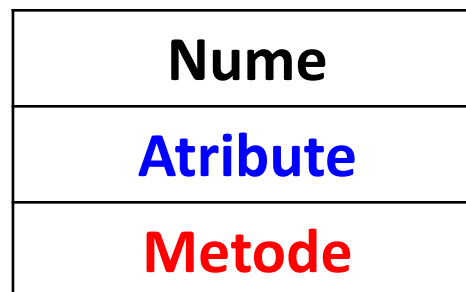


Figura 2. Reprezentare grafică a conceptului de clasă

1.1. Obiecte și clase

De exemplu, putem avea clasa cărți:

nume	cărți
atribute	titlu, autor, editura, an_apariție, ISBN, preț
metode	obține_titlu, obține_autor, modifică_preț, afișează_informații

1.1. Obiecte și clase

- Construirea obiectelor informatice pornind de la clase poartă numele de **instanțiere** sau **exemplificare**.
- *Obiectul va fi o instanță a unei clase.*
- Diferențele dintre obiectele de aceeași clasă se materializează în diferențe între valorile atributelor.
- Totodată, pentru fiecare obiect este specificat tipul clasei din care provine.
- Pentru o clasă se pot crea mai multe instanțe ale acesteia.

1.1. Obiecte și clase

- De exemplu, avem 3 instanțe ale clasei *cărți* după diferitele valori date atributelor astfel:

atribute	titlu	Poezii	Geniu Pustiu	Marile speranțe
	autor	Mihai Eminescu	Mihai Eminescu	Charles Dickens
	editura	Polirom	Polirom	Univers
	an apariție	2007	2005	2003
	ISBN	973-567-545-1	973-565-545-2	971-267-441-1
	preț	25	20	35

1.1. Obiecte și clase

- *Definirea unei clase înseamnă crearea unui nou tip de date care apoi poate fi utilizat pentru declararea obiectelor de acest tip.*
- **Clasa** este elementul de programare care ne permite transcrierea cât mai bună a unui concept din viața concretă într-un limbaj de programare.
- Ea *permite definirea atât a datelor relativ la o entitate cât și a acțiunilor asociate prin metode.*

1. Noțiuni introductive despre programarea orientată pe obiecte

1.1. Obiecte și clase

1.2. Principii POO

1.2. Principii POO

1. **Abstractizare**
2. **Încapsulare**
3. **Moștenire. Derivare**
4. **Polimorfism**

1.2. Principii POO

Definiție

1. Abstractizarea reprezintă procesul prin care se izolează și se rețin numai o parte dintre aspectele unei probleme, considerate esențiale – funcție de scopul urmărit – celelalte fiind ignorate.

- La nivelul limbajelor de programare, abstractizarea înseamnă un anumit gen de apropiere de limbajul uman și, prin aceasta, de gândirea umană.
- *Abstractizarea este procesul de simplificare a realității complexe prin modelarea de clase cât mai generale și cât mai apropiate de problema care se tratează.*

1.2. Principii POO

Definiție

2. Încapsulare. Construirea și operarea cu obiecte.

- **Încapsularea** este principiul care se bazează pe combinarea datelor cu operațiile asupra lor dintr-un obiect și proprietatea obiectelor de ascundere de date și operațiile proprii față de alte obiecte.
- Într-un obiect, o parte din operații și/sau date pot fi particulare aceluși obiect și inaccesibile pentru orice din afara sa.
- În acest fel, un obiect dispune de un nivel semnificativ de protecție care împiedică modificarea accidentală sau utilizarea incorectă a părților proprii obiectului.

1.2. Principii POO

- *Accesul la attributele unui obiect* se face doar prin apelarea metodelor sale prin interfețe sau prin mesaje, orice alt obiect neștiind nici măcar de existența atributelor obiectului în cauză.
- *Metodele formează interfața clasei*, iar mesajele reprezintă cereri adresate obiectului pentru a returna o valoare sau pentru a-și schimba starea.

1.2. Principii POO

- Din perspectiva încapsulării, o **clasă**, indiferent de limbajul în care va fi implementată, trebuie să aibă obligatoriu 2 secțiuni:
1. *privată*
 2. *și publică*

CLASA	
Atribute și metode private	} secțiune privată - realizează implementarea
Metode publice	} secțiune publică - formează interfața

Figura 3. Componentele obligatorii ale unei clase din perspectiva încapsulării

1.2. Principii POO

➤ De exemplu: clasa *stivă* (ca structură dinamică)

CLASA STIVĂ	
atribute: vârf, dimensiune maximă	} secțiune privată
metode : push, pop, empty, full	} secțiune publică

1.2. Principii POO

- Din perspectiva cuvintelor cheie, *combinare* și *ascundere*, pe care se bazează principiul încapsulării putem vedea următoarele avantaje:

ÎNCAPSULAREA = COMBINARE + ASCUNDERE

COMBINAREA - AVANTAJE

- definește clar ce structuri de date sunt manevrate și care sunt operațiile legale asupra lor
- adaugă modularitate programului
- scade riscul ca datele să fie alterate de programele "slabe"
- facilitează ascunderea informației

ASCUNDEREA – AVANTAJE

- programe mai sigure și fiabile
- eliberează clasele utilizator de grija cum sunt manevrate datele
- previne apariția erorilor produse de clasele utilizator ce manevrează datele utilizând cod "slab"

ÎNCAPSULARE (COMBINARE + ASCUNDERE) – AVANTAJE

- combinare + ascunderea informației = protejarea datelor
- previne apariția erorilor prin limitarea accesului la date
- asigură portabilitatea programelor
- facilitează utilizarea excepțiilor
- interfața unei clase = operațiile cu care o clasă poate manevra datele

1.2. Principii POO

Definiție

3. Moștenirea este principiul prin care putem reutiliza și extinde clasele existente.

- Acest mecanism *permite unei noi clase să beneficieze de attributele și metodele definite într-o clasă deja existentă prin declararea că moștenim acea clasă.*
- Apoi, subclasa respectivă poate să definească propriile attribute și metode, eventual să redefinească unele metode.

Derivarea permite definirea unei clase noi pe baza unei clase existente.

1.2. Principii POO

1. Clasa existentă, care va fi moștenită se mai numește **clasa de bază** (*părinte, superclasa sau strămoș*).
2. Clasa care realizează extinderea se numește **clasă derivată** (*copil, subclasă sau descendent*).

1.2. Principii POO

Considerând o clasă oarecare A și o clasă B care moștenește clasa A atunci putem avea o reprezentare grafică:

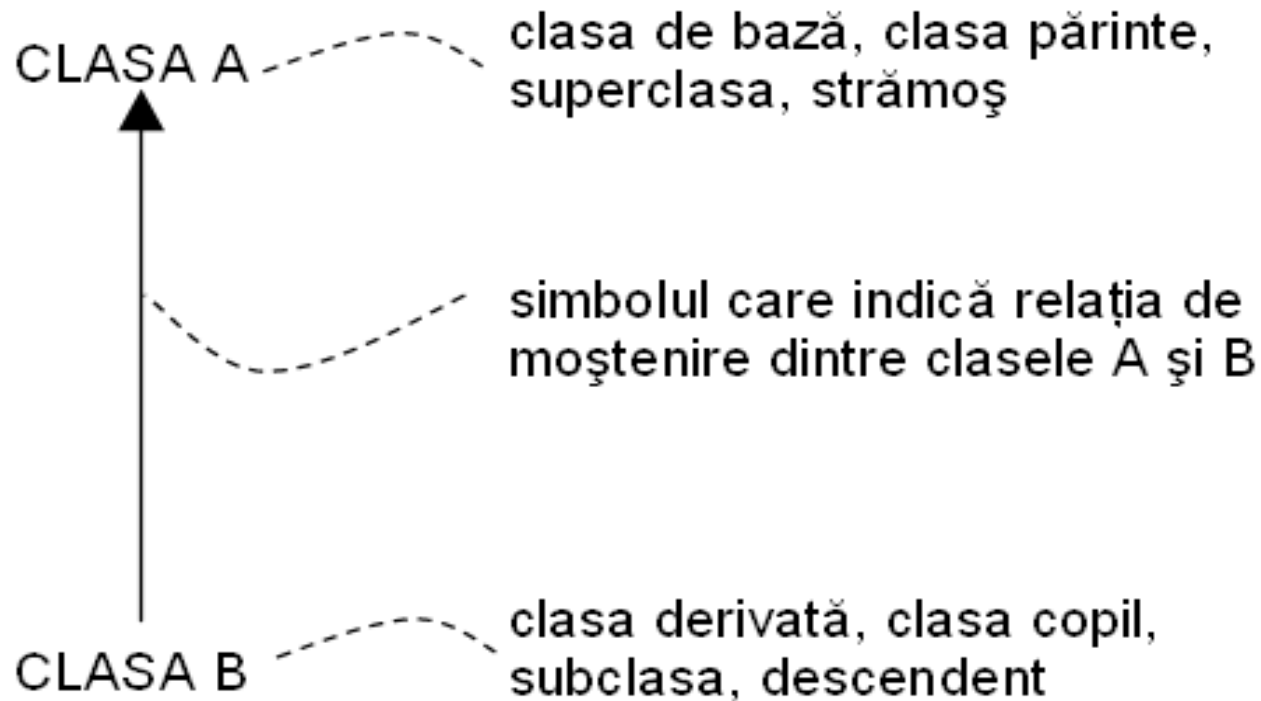


Figura 4. Reprezentarea grafică a relației de moștenire care operează între clase.

1.2. Principii POO

- Dintr-o clasă de bază pot fi derivate mai multe clase și fiecare clasă derivată poate deveni la rândul ei o clasă de bază pentru alte clase derivate.
- Se poate astfel realiza o **ierarhie de clase**, care să modeleze sisteme complexe.
- *Construirea ierarhiei de clase constituie activitatea fundamentală de realizare a unei aplicații orientate obiect, reprezentând în fapt faza de proiectare a respectivului sistem.*
- Mecanismul moștenirii permite crearea unei ierarhii de clase și trecerea de la clasele generale la cele particulare.
- Atunci când un obiect dintr-o clasă are proprietăți comune (date prin attribute și /sau metode) cu o altă clasă, nu mai trebuie creat de la zero ci putem deriva respectivul obiect.

1.2. Principii POO

- *Relația de moștenire dintre clase este o reflectare a ierarhizării existente între entitățile modelate de clasele respective.*
- Astfel aproape toate soluțiile orientate pe obiect au o clasă rădăcină (care, în anumite situații poate fi o clasă abstractă, adică o clasă care nu poate avea instanțe directe, dar pot fi asociate cu instanțe ale descendenților), dacă soluția se reduce la o singură ierarhie de clase.
- În ierarhie putem întâlni și **clase intermediare**, precum și **clase terminale** sau **frunze**.

1.2. Principii POO

- De exemplu, pornim de la o **clasă rădăcină poligon** care să modeleze corpul geometric de tip poligon.
- Din această clasă se pot **deriva clasele triunghi și patrulater** și atunci clasa poligon trebuie să cuprindă proprietățile comune ale figurilor triunghi și patrulater.
- Apoi din clasa patrulater **derivăm clasa paralelogram și clasa trapez**, iar din clasa paralelogram derivăm **clasa romb și clasa dreptunghi**.

1.2. Principii POO

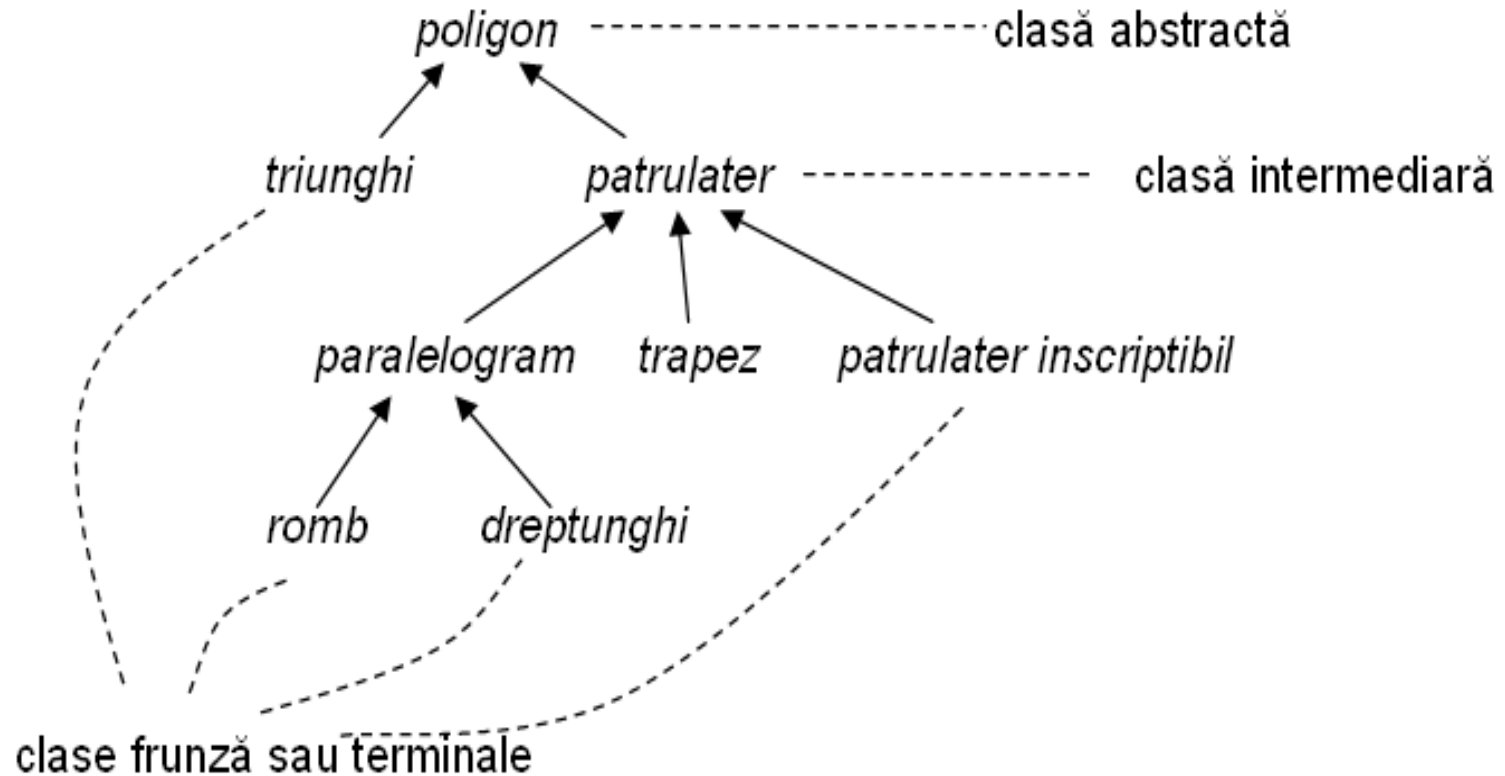


Figura 5. Exemplu de proiectare ierarhie de clase

1.2. Principii POO

Clasificări ale moștenirii:

1. Moștenire simplă este o ierarhie de clasă în care fiecare clasă derivată are o singură clasă de bază, rezultând o structură arborescentă.

De exemplu:

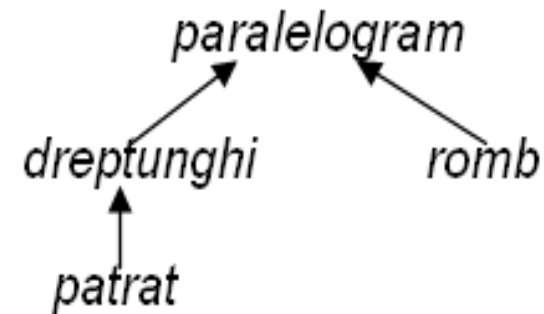


Figura 6. Exemplu de proiectare clase cu moștenire simplă

- Clasele *dreptunghi* și *romb* sunt clase derivate a clasei *paralelogram*, iar clasa *pătrat* este o clasă derivată a clasei *dreptunghi*.
- *Astfel, o ierarhie de concepte conduce la o ierarhie între clasele care implementează conceptele respective.*

1.2. Principii POO

2. Moștenire multiplă este proprietatea unei clase de a deriva din mai multe clase de bază, rezultând o structură de rețea.

De exemplu:

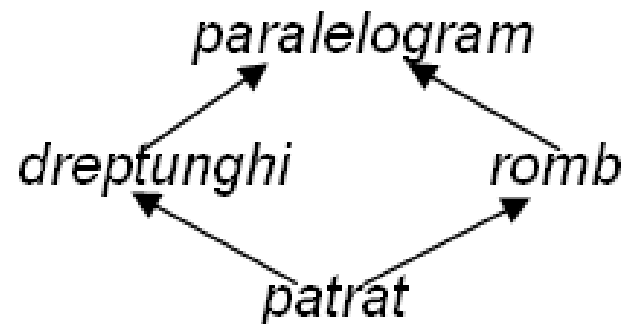


Figura 7. Exemplu de proiectare clase cu moștenire multiplă

- Pătratul se poate defini ca un romb care are un unghi drept.
- În acest caz, clasa *pătrat* devine o clasă derivată pe lângă clasa *dreptunghi* și este o clasă derivată din clasa *romb*.

1.2. Principii POO

- Clasa derivată *moștenește toți membrii clasei de bază*, dar nu va putea avea acces niciodată la membrii declarați privați în clasa de bază.
- Însă acele proprietăți comune date de attribute și metode pot fi ajustate după necesități.
- *În declarația clasei derivate nu mai apar informațiile care sunt moștenite, ele fiind automat luate în considerare de către compilator.*
- Nu mai trebuie rescrise metodele clasei de bază, ele putând fi folosite în maniera în care au fost definite.
- Mai mult, metodele din clasa de bază pot fi redefinite (polimorfism), având o cu totul altă funcționalitate.

1.2. Principii POO

Definiție

4. Polimorfism reprezintă posibilitatea de a putea aplica în moduri diferite o aceeași operație mai multor clase.

- Prin operație înțelegem orice **metodă** sau **operator**.
- *Polimorfismul reprezintă abilitatea unor obiecte similare de a răspunde la același mesaj în moduri diferite.*
- De exemplu, să considerăm operația de adunare, exprimată printr-un singur operator, semnul plus (+).
- Atunci prin expresia $x+y$ se poate indica sumarea a 2 numere întregi, a 2 numere complexe, a 2 matrici sau concatenarea a 2 șiruri de caractere.

1.2. Principii POO

- *Într-o ierarhie de clase obținută prin moștenire, o metodă poate avea implementări diferite la nivele diferite în acea ierarhie.*
- Deci polimorfismul dă astfel posibilitatea pentru obiectele diferitor clase, legate prin moștenire, să reacționeze în mod diferit la apelul uneia și aceleași metode.
- De exemplu, admițând că, în **clasa poligon** am declarat o **metodă calcul_arie** atunci aceasta se poate aplica și claselor descendente (**triunghi**, **dreptunghi**, **pătrat**), prin **diferite moduri de calcul al ariei**.

1.2. Principii POO

Polimorfismul în POO este două tipuri:

1. Polimorfism static - **supraîncărcarea metodelor (overloading)** – care apare la definirea de metode cu același nume în cadrul unei clase, dar cu semnătură diferită (mod de apelare – parametri).

Are loc în faza de compilare și se aplică atât la metodele unei clase, cât și la operatorii predefiniți ai limbajului.

- Exemplu_1: supradefinirea operatorilor aritmetici (+, -, *, etc.) pentru diferite entități
- Exemplu_2: în cadrul clasei **dreptunghi** existența a 2 metode denumite **mutare**: pentru o poziție nouă a punctului din colțul stânga sus, respectiv pentru o deplasare spre stânga cu o valoare.

1.2. Principii POO

2. Polimorfism dinamic – *prin redefinirea (suprascrierea) metodelor (over-riding)* – apare în momentul în care metoda unei clase are același nume și semnătură cu o metodă din clasa de bază.

- *Atunci spunem că metoda din clasa derivată supradefinește metoda din clasa de bază.*
- Identificarea unei metode supradefinite, în momentul execuției, se numește *legare ulterioară*, "*late binding*".
- De exemplu, *metoda calcul_arie* care se poate defini la nivel de clasă *poligon* și apoi redefini în clasele descendente conform tipului.

Întrebări?