Programare orientată pe obiecte

#1 JAVA
Noțiuni introductive despre POO

https://www.runceanu.ro/adrian/activitate-didactica/

Obiectivele disciplinei

Obiectivele generale ale disciplinei:

1. Cunoaşterea noţiunilor privind algoritmii şi proprietăţile lor
2. Reprezentarea algoritmilor prin scheme logice, pseudocod, programe Java
3. Utilizarea platformei integrate de dezvoltare ECLIPSE

Obiectivele specifice:

1. Pentru curs:
   - Cunoaşterea limbajului Java
   - Elaborarea de programe în Java
   - Analiza şi 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: Conf. univ. dr. Adrian Runceanu
- ✓**2 ore laborator** – aplicaţii practice: Conf. univ. 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.

Resurse

1. Suport curs - varianta electronică disponibilă pe: https://www.runceanu.ro/adrian/

Notă: Actualizarea site-ului se face saptamanal.

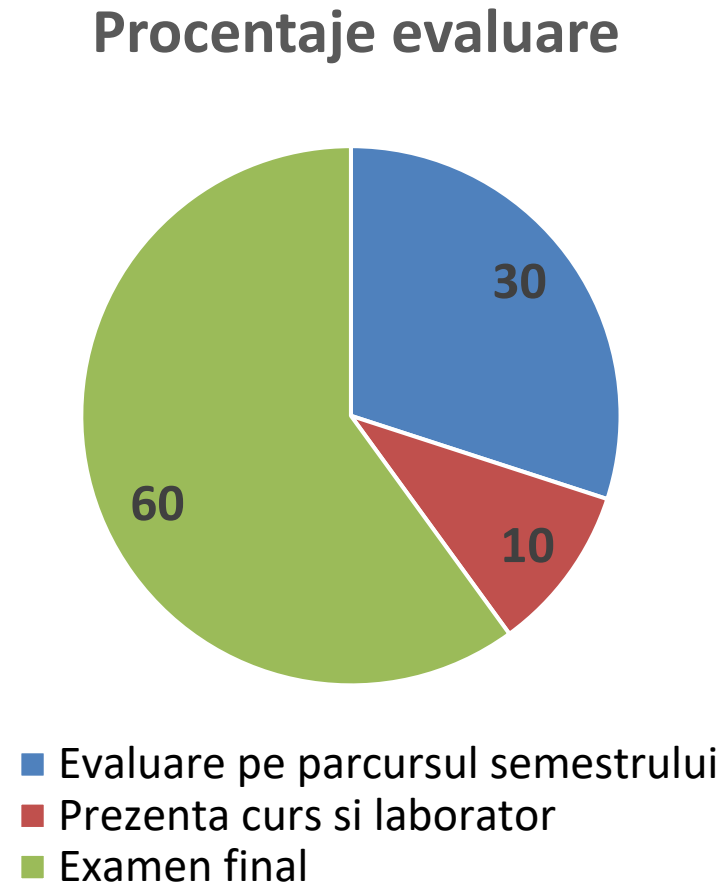
2. curs pe Teams (FI-AIA-2-Programare orientata pe obiecte-2025-2026)

3. laborator pe Teams (FI-AIA-2-Programare orientata pe obiecte-2025-2026)

Câteva precizări

Forme de examinare:

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



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 şi clase

1.2. Principii POO

1.1. Obiecte și clase

Obiecte. Clasificare, identitate, stare, comportament.

- În lumea în care trăim suntem obişnuiţi să numim obiecte acele *entităţi 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 entităţi fizice de genul *formulelor matematice*.
- Tot prin extensie, *obiectele pot aparţine unei lumi virtuale*, de genul unor evenimente ce pot avea loc, cum ar fi *apăsarea unei taste*, *producerea unei explozii intr-un joc* sau *consultarea unui cont*.

1.1. Obiecte și clase

Definiţie

Obiectul este conceptul de bază în programarea orientată obiect (OOP - **O**bject **O**riented **P**rogramming) care asociază datele împreună cu operaţiile necesare prelucrării.

1.1. Obiecte și clase

Definiţie

Datele sunt informaţii de structură descrise de o mulţime de atribute ale obiectului, iar **operaţiile** acţionează asupra atributelor obiectului şi eventual, asupra altor obiecte.

- Modelul orientat obiect se bazează pe obiect.

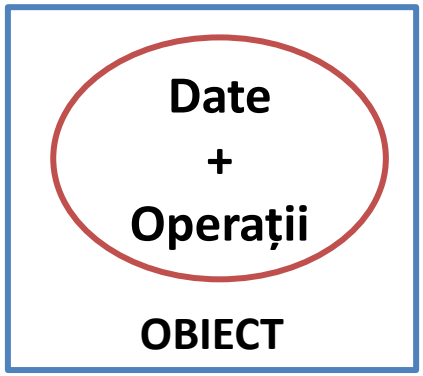


Figura 1. Modelul orientat obiect

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 conţinute în obiect şi la valorile asociate acestora*.

Acestea formează atributele care descriu proprietăţile unui obiect.

- De exemplu, atributele unui obiect „carte” pot fi *titlul, autorul, editura, număr pagini, anul apariţiei, ISBN şi preţ* atunci **starea unei cărţi** ar putea fi următoarea:

titlu = Poezii	editura = Polirom	ISBN = 973–567–545–1
autor = Mihai Eminescu	număr pagini = 125	preţ = 25
	anul apariţiei = 2007	

1.1. Obiecte și clase

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

Operaţiile se deduc din acţiunile obiectului pe care trebuie să le îndeplinească.

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

obţine_titlu, obţine_autor, modifică_preţ, calcul_TVA, afişează_informaţii

1.1. Obiecte și clase

3. Identitate este un identificator - OID (Object IDentifier) care caracterizează unic obiectul, permiţând să se construiască referiri spre obiect şi să se facă distincţia tuturor obiectelor într-o manieră non-ambiguă şi independentă de starea lor.

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

1.1. Obiecte și clase

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

1.1. Obiecte și clase

Definiţie

Clasa este conceptul de bază în POO ce reuneşte *o colecţie de obiecte care partajează aceeaşi listă de atribute informaţionale şi comportamentale*.

- O clasă va cuprinde definiţiile datelor şi operaţiilor ce caracterizează obiectele de o anumită categorie.

De exemplu:

clasa cărţi	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 şi operaţii comune sau nu.

- De exemplu, între clasa *studenți* şi clasa *profesori* pot exista una sau mai multe date identice, cum ar fi: *nume*, *vârsta*, *facultatea*, etc.
- Între clasa *studenți* şi 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 operaţiile se mai numesc metode sau funcţii-membru.
- Atributele şi metodele formează membrii unei clase.
- Fiecare clasă va avea identitate sau nume.

Nume
Atribute
Metode

Figura 2. Reprezentare grafică a conceptului de clasă

1.1. Obiecte și clase

De exemplu, putem avea clasa cărţi:

nume	cărţi
atribute	titlu, autor, editura, an_apariţie, ISBN, preţ
metode	obţine_titlu, obţine_autor, modifică_preţ, afişează_informaţii

1.1. Obiecte și clase

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

1.1. Obiecte și clase

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

atribute	titlu	Poezii	Geniu Pustiu	Marile speranţe
	autor	Mihai Eminescu	Mihai Eminescu	Charles Dickens
	editura	Polirom	Polirom	Univers
	an apariţie	2007	2005	2003
	ISBN	973–567–545–1	973–565–545–2	971–267–441 –1
	preţ	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 viaţa concretă într-un limbaj de programare.
- Ea *permite definirea atât a datelor relativ la o entitate cât şi a acţiunilor asociate prin metode.*

1. Noțiuni introductive despre programarea orientată pe obiecte

1.1. Obiecte şi clase

1.2. Principii POO

1.2. Principii POO

1. Abstractizare
2. Încapsulare
3. Moştenire. Derivare
4. Polimorfism

1.2. Principii POO

Definiţie

1. Abstractizarea reprezintă procesul prin care se izolează şi se reţin numai o parte dintre aspectele unei probleme, considerate esenţiale – funcţie de scopul urmărit – celelalte fiind ignorate.

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

1.2. Principii POO

Definiţie

2. Încapsulare. Construirea şi operarea cu obiecte.

- ***Încapsularea*** este principiul care se bazează pe combinarea datelor cu operaţiile asupra lor dintr-un obiect şi proprietatea obiectelor de ascundere de date şi operaţiile proprii faţă de alte obiecte.
- Într-un obiect, o parte din operaţii şi/sau date pot fi particulare acelui obiect şi inaccesibile pentru orice din afara sa.
- În acest fel, un obiect dispune de un nivel semnificativ de protecţie care împiedică modificarea accidentală sau utilizarea incorectă a părţilor proprii obiectului.

1.2. Principii POO

- *Accesul la atributele unui obiect* se face doar prin apelarea metodelor sale prin interfeţe sau prin mesaje, orice alt obiect neştiind nici măcar de existenţa atributelor obiectului în cauză.

- *Metodele formează interfaţa clasei*, iar mesajele reprezintă cereri adresate obiectului pentru a returna o valoare sau pentru a-şi schimba starea.

1.2. Principii POO

- Din perspectiva încapsulării, o **clasă**, indiferent de limbajul în care va fi implementată, trebuie să aibă obligatoriu 2 secţiuni:

1. *privată*
2. şi *publică*

CLASA	
Atribute şi metode private	} secţiune privată - realizează implementarea
Metode publice	} secţiune publică - formează interfaţa

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	} secţiune privată
metode : push, pop, empty, full	} secţiune publică

1.2. Principii POO

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

ÎNCAPSULAREA = COMBINARE + ASCUNDERE

COMBINAREA - AVANTAJE

- defineşte clar ce structuri de date sunt manevrate şi care sunt operaţiile legale asupra lor
- adaugă modularitate programului
- scade riscul ca datele să fie alterate de programele "slabe"
- facilitează ascunderea informaţiei

ASCUNDEREA – AVANTAJE

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

ÎNCAPSULARE (COMBINARE + ASCUNDERE) – AVANTAJE

- combinare + ascunderea informaţiei = protejarea datelor
- previne apariţia erorilor prin limitarea accesului la date
- asigură portabilitatea programelor
- facilitează utilizarea excepţiilor
- interfaţa unei clase = operaţiile cu care o clasă poate manevra datele

1.2. Principii POO

Definiţie

3. Moştenirea este principiul prin care putem reutiliza şi extinde clasele existente.

- Acest mecanism *permite unei noi clase să beneficieze de atributele şi metodele definite într-o clasă deja existentă prin declararea că moştenim acea clasă*.
- Apoi, subclasa respectivă poate să definească propriile atribute şi 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 moştenită se mai numeşte ***clasa de bază*** *(părinte, superclasa sau strămoş)*.

2. Clasa care realizează extinderea se numeşte ***clasă derivată*** *(copil, subclasă sau descendent)*.

1.2. Principii POO

Considerând o clasă oarecare A şi o clasă B care moşteneşte clasa A atunci putem avea o reprezentare grafică:

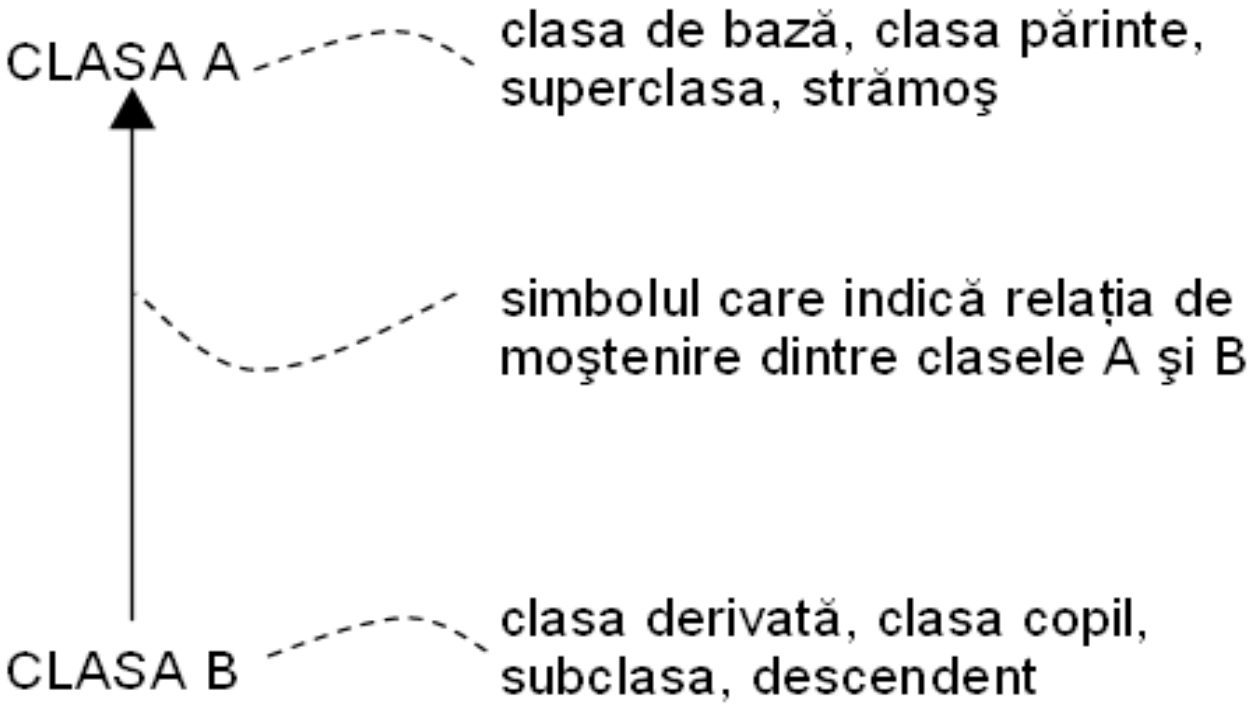


Figura 4. Reprezentarea grafică a relaţiei de moştenire care operează între clase.

1.2. Principii POO

- Dintr-o clasă de bază pot fi derivate mai multe clase şi 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 aplicaţii orientate obiect, reprezentând în fapt faza de proiectare a respectivului sistem.*
- Mecanismul moştenirii permite crearea unei ierarhii de clase şi trecerea de la clasele generale la cele particulare.
- Atunci când un obiect dintr-o clasă are proprietăţi comune (date prin atribute şi /sau metode) cu o altă clasă, nu mai trebuie creat de la zero ci putem deriva respectivul obiect.

1.2. Principii POO

- *Relaţia de moştenire dintre clase este o reflectare a ierarhizării existente între entităţile modelate de clasele respective.*
- Astfel aproape toate soluţiile orientate pe obiect au o clasă rădăcină (care, în anumite situaţii poate fi o clasă abstractă, adică o clasă care nu poate avea instanţe directe, dar pot fi asociate cu instanţe ale descendenţilor), dacă soluţia se reduce la o singură ierarhie de clase.
- În ierarhie putem întâlni şi clase intermediare, precum şi 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 şi patrulater şi atunci clasa poligon trebuie să cuprindă proprietăţile comune ale figurilor triunghi şi patrulater.
- Apoi din clasa patrulater derivăm clasa paralelogram şi clasa trapez, iar din clasa paralelogram derivăm clasa romb şi clasa dreptunghi.

1.2. Principii POO

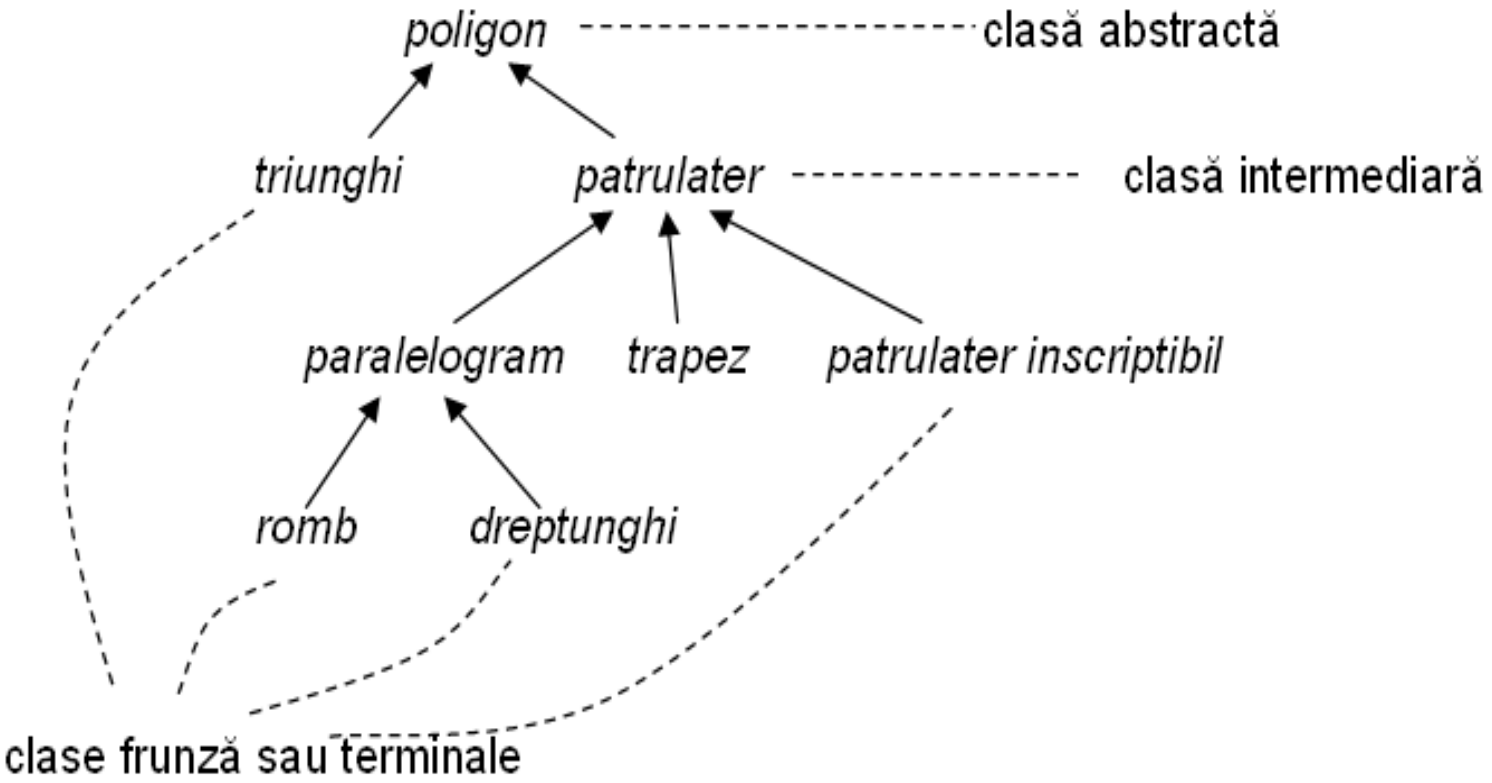


Figura 5. Exemplu de proiectare ierarhie de clase

1.2. Principii POO

Clasificări ale moștenirii:

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

De exemplu:

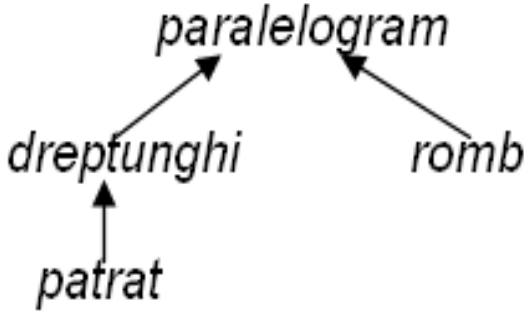


Figura 6. Exemplu de proiectare clase cu moştenire simplă

- Clasele *dreptunghi* şi *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. Moştenire multiplă este proprietatea unei clase de a deriva din mai multe clase de bază, rezultând o structură de reţea.

De exemplu:

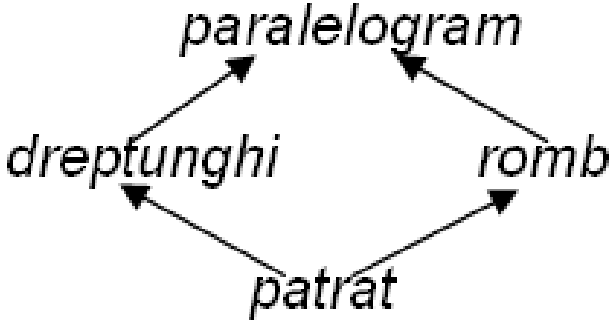


Figura 7. Exemplu de proiectare clase cu moştenire multiplă

- 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* şi este o clasa derivată din clasa *romb.*

1.2. Principii POO

- Clasa derivată *moşteneşte toţi membrii clasei de bază*, dar nu va putea avea acces niciodată la membrii declaraţi privaţi în clasa de bază.
- Însă acele proprietăţi comune date de atribute şi metode pot fi ajustate după necesităţi.
- *În declaraţia clasei derivate nu mai apar informaţiile care sunt moştenite, 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 funcţionalitate.

1.2. Principii POO

Definiţie

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

- Prin operaţie înţelegem orice metodă sau operator.
- *Polimorfismul reprezintă abilitatea unor obiecte similare de a răspunde la acelaşi mesaj în moduri diferite.*
- De exemplu, să considerăm operaţia 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 şiruri de caractere.

1.2. Principii POO

- *Într-o ierarhie de clase obţinută prin moştenire, o metodă poate avea implementări diferite la nivele diferite în acea ierarhie.*
- Deci polimorfismul dă astfel posibilitatea pentru obiectele diferitor clase, legate prin moştenire, să reacţioneze în mod diferit la apelul uneia şi aceleaşi metode.
- De exemplu, admiţând că, în clasa poligon am declarat o metodă calcul_arie atunci aceasta se poate aplica şi 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 (over-loading) – care apare la definirea de metode cu acelaşi nume în cadrul unei clase, dar cu signatură diferită (mod de apelare – parametri).

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

- Exemplu_1: supradefinirea operatorilor aritmetici (+, -, *, etc.) pentru diferite entităţi
- Exemplu_2: în cadrul clasei dreptunghi existenţa a 2 metode denumite mutare: pentru o poziţie nouă a punctului din colţul 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 clasei are acelaşi nume şi signatură cu o metodă din clasa de bază.

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

Întrebări?