



Programare orientată pe obiecte

1. Despre curs
2. Concepte și paradigme în POO



Obiectivele cursului

- Să vă ofere cunoștințe operaționale în dezvoltarea software folosind orientarea pe obiecte, cu accent pe conceptele, paradigmele și atitudinile necesare scrierii de cod orientat pe obiecte de calitate
- Să vă permită să vă dezvoltați îndemânarea în programare și inginerie software, să învățați să utilizați efectiv un limbaj de programare orientat pe obiecte relevant din punctul de vedere al industriei software



Despre POO

- Orele de curs: Marius Joldos
 - Marius.Joldos@cs.utcluj.ro
- Laborator: veți afla cu cine
- Web:
 - <http://users.utcluj.ro/~jim/OOPR> (public)
 - <http://t201:81/OOPR> (accesibil numai în D3)
- 14 sesiuni de curs a 2 ore
- 14 sesiuni de laborator a 2 ore
- 5 credite
- Colocviu laborator, miniproiect, examen parțial și final



Rezultatele cursului (1)

- Privitoare la cunoștințe/înțelegere
 - Fundamentele POO
 - Elementele principale în proiectarea, programarea, testarea și documentarea soluțiilor OO
 - Metode de proiectare pentru programe Java de complexitate relativ redusă
 - Elemente de bază ale UML



Rezultatele cursului (2)

- Abilități intelectuale
 - Să înțelegeți specificația unei probleme și să o rezolvați printr-un program pentru calculator în paradigma programării orientate pe obiecte.
 - Să luați o specificație parțială și să luați deciziile corespunzătoare asupra funcționalității sistemului propus.
 - Să dezvoltați o specificație în UML dintr-o descriere în limbaj natural



Rezultatele cursului (3)

- Abilități practice
 - Să folosiți eficient conceptele de programare OO în Java
 - Să dezvoltați programe de complexitate relativ redusă în Java
 - Să depanați, testați și documentați soluții folosind orientarea pe obiecte
 - Să dezvoltați applet-uri și componente GUI relativ simple



Subiecte abordate

- Concepte și paradigme în programarea orientată pe obiecte
- Abstracțiuni și tipuri de date abstracte
- Caracteristicile limbajului Java
- Tipurile de date primitive și structurile de control în Java
- Clase și obiecte
- Interfețe Java
- Excepții și tratarea lor
- Principalele API-uri și clase predefinite în Java
- Reprezentarea în UML a claselor, obiectelor și asociațiilor; diagrame de clase și obiecte
- Programarea bazată pe clase și obiecte
- Moștenirea și polimorfismul
- Programarea orientată pe obiecte
- Scurtă introducere în programarea condusă de evenimente
- Scurtă introducere în programarea applet-urilor și a componentelor GUI



Evaluare. Referințe

- **Evaluare**
 - Examen scris (E), evaluarea activității de la laborator (L), evaluarea unui mini-proiect (P).
 - Nota finală = $0.6E + 0.4(L + P)/2$
- **Referințe**
 - Tânăsă, S., Olaru, C., Andrei, Șt., *Java de la 0 la expert*, Ed. Polihrom, 2003
 - Călin Văduva, *Programarea în Java*, Ed. Albastură, 2004 (reeditare)
 - Bruce Eckel, *Thinking in Java*, 3rd edition, Prentice Hall, 2002
 - Documentația Java de la Sun Microsystems
 - Tutorialele UML introductive
 - Documentația BlueJ



Concepte și paradigme în POO

- **Temele de astăzi**
 - Paradigme de programare (programarea imperativă și programarea structurată)
 - Abstracțizarea datelor
 - Tipuri de date abstracte
 - Paradigma POO
 - Concepte POO
 - Obiecte și clase
 - Încapsulare și moștenire



Paradigme de programare

- **Paradigmă** (dict.): Un set de supoziții, concepte, valori și practici care constituie o viziune a realității pentru comunitatea care le adoptă, în special într-o disciplină intelectuală.
 - **Paradigmă de programare**
 - Un model care descrie esența și structura computației
 - Oferă (și determină) viziunea pe care o are programatorul asupra execuției programului
- Exemple:
- În POO, programatorii pot concepe programele ca fiind o colecție de obiecte care interacționează
 - În programarea funcțională un program poate fi conceput ca fiind o secvență de evaluări de funcții, fără stări



Paradigme de programare

- Programarea OO
- Abstracțizarea datelor
- Programarea structurată
- Programarea imperativă

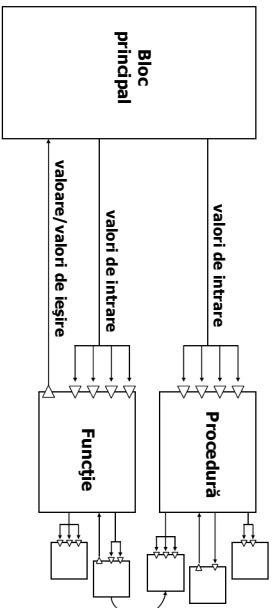


Programarea imperativă

- În modelul tradițional von Neumann, un calculator constă dintr-o unitate centrală de prelucrare și memorie și el efectuează secvențe de instrucțiuni atomice care accesează, operează asupra valorilor stocate local și de memorie adresabile individual și le modifică. Aici, o **computație este o serie de operații aritmetice și de efecte laterale, cum sunt atribuirile sau transferurile de date care modifică starea unității de stocare, intrarea sau ieșirea**
- Ne referim la acest ca **paradigmă imperativă sau procedurală**.
- Este de subliniat importanța atribuirilor și a variabililor pe post de containere pentru paradigma imperativă
- Exemple de limbaje: Fortran, Pascal, C, Ada.



Programarea structurată



oop1 - T.U. Cluj - M. Joldos

13



Beneficiile programării structurate

- Ușurează dezvoltarea software
- Evită repetarea realizării aceluiași lucru
- Munca de programare este descompusă în module independente
- Proiectare Top-down: descompunerea în subprobleme
- Facilitează întreținerea software
- Codul este mai ușor de citit
- Independența modulelor
- Favorizează reutilizarea software

oop1 - T.U. Cluj - M. Joldos

15



Abstractizarea datelor

- Abstractizarea datelor: impunerea unei separări clare între *proprietățile abstracte* ale unui *tip de dată* și *detaliile concrete* ale *implementării* lui
 - Proprietăți abstracte: acelea care sunt vizibile codului client care folosește tipul de dată – *interfața* cu tipul de dată
 - Implementarea concretă este păstrată în totalitate privată și ea se poate într-adevăr schimba, spre exemplu pentru a incorpora îmbunătățiri ale performanțelor în timp.

Computer Science

oop1 - T.U. Cluj - M. Joldos

17



Programarea structurată

- Abstractizarea operațiilor
 - Structura unui modul
 - Interfața
 - Date de intrare
 - Date de ieșire
 - Descrierea funcționalității
 - Implementarea
 - Date locale
 - Secvențe de instrucțiuni
 - Sintaxa limbajului
 - Organizarea codului în blocuri de instrucțiuni
 - Definiții de funcții și proceduri
 - Extinderea limbajului cu noi operații
 - Apeluri la proceduri și funcții noi

oop1 - T.U. Cluj - M. Joldos

14



Programarea structurată. Exemplu

```
int main()
{
    double u1, u2, m;
    u1 = 4;
    u2 = -2;
    m = sqrt (u1*u1 + u2*u2);
    printf("%d\n", m);
    return 0;
}
```

```
double module(double u1, double u2)
{
    double m;
    m = sqrt (u1*u1 + u2*u2);
    return m;
}
int main()
{
    printf("%d\n", module(4, -2));
    return 0;
}
```

oop1 - T.U. Cluj - M. Joldos

16



Tipuri abstracte de date

- Abstractizarea datelor + abstractizarea operațiilor
 - Un tip de dată abstract:
 - **Structură de date** care stochează informații pentru a reprezenta un anumit concept
 - **Funcționalitate:** set de operații care pot fi aplicate tipului de dată
 - Sintaxa limbajului
 - Modulele sunt asociate tipurilor de date
 - Sintaxa nu este neapărat nouă față de programarea modulară

Computer Science

oop1 - T.U. Cluj - M. Joldos

18



Exemplu de tip de dată abstract în C

```

struct vector {
    double x;
    double y;
}
void construct (vector *v, double v1, double v2)
{
    v->x = v1;
    v->y = v2;
}
double module(vector v)
{
    double m;
    m = sqrt (v.x*v.x + v.y*v.y);
    return m;
}

int main()
{
    vector v;
    construct(&v, 4, 2);
    printf("%d\n", module(v));
    return 0;
}

```



Extensibilitatea tipului de dată abstract

```

...
double product(vector v, vector w) {
    return v.x*w.x + v.y*w.y;
}
int main() {
    vector v;
    construct(&v, 4, 2);
    construct(&w, -1, 7);
    printf("%d\n", product(v, w));
    return 0;
}

```



Beneficiile tipurilor de date abstracte

- Conceptele din domeniu sunt reflectate în cod
- **Încapsulare:** complexitatea internă, datele și detaliile operațiilor sunt ascunse
- Utilizarea tipului de dată este independentă de implementarea sa internă
- Oferă o mai mare modularitate
- Sporește ușurința întreținerii și reutilizării codului



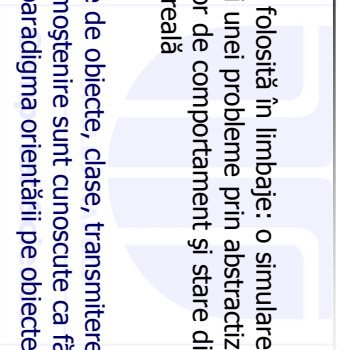
Paradigma orientată pe obiecte

- Paradigma programării structurate a avut inițial succes (1975-85)
 - Dar a început să eșueze la produse mai mari (> 50,000 LOC)
- PS avea probleme de întreținere post-livrare (astăzi această întreținere necesită, de la 70 la 80% din efortul total)
- Motivul: Metodele structurate sunt fie
 - orientate pe operații (analiza fluxului de date) fie orientate pe atribute (d.e. dezvoltarea cu metoda Jackson)...
- ...dar nu amândouă



Paradigma orientării pe obiecte

- Paradigma folosită în limbaje: o simulare a domeniului unei probleme prin abstractizarea informațiilor de comportament și stare din obiecte din lumea reală
- Conceptele de obiecte, clase, transmitere de mesaje și moștenire sunt cunoscute ca făcând parte din paradigma orientării pe obiecte.

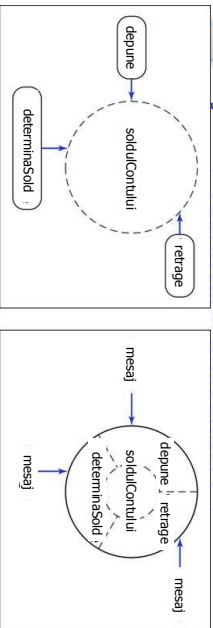


Paradigma orientării pe obiecte

- POO consideră că atât atributele cât și operațiile au importanță egală
- O viziune simplistă a unui obiect poate fi:
 - Obiect = componentă software care incorporează atât atributele cât și operațiile care se pot efectua asupra atributelor și care suportă moștenirea.
- Exemplu:
 - Cont bancar
 - Date: soldul contului
 - Acțiuni: depune, retrage, determină soldul



Comparație între paradigma structurată și cea obiectuală



- Ascunderea informației
- Proiectarea dirijată de responsabilități
- Impact asupra întreținerii și dezvoltării

oop1 - T.U. Cluj - M. Joldos

25



Ascunderea informației

- În versiunea orientată pe obiecte
 - Linia continuă din jurul lui `solidulContului` arată că în afara obiectului nu se știe cum este implementat `solidulContului`
- În versiunea clasică
 - Toate modulele au detalii privind implementarea lui `solidulContului`

oop1 - T.U. Cluj - M. Joldos

26



Punctele tari ale paradigmei OO

- Cu ascunderea informației, întreținerea post-livrare este mai sigură
 - Șansele apariției erorilor regresive sunt reduse (în software nu se repetă erori cunoscute)
- Dezvoltarea este mai ușoară
 - Obiectele au în general corespondente fizice
 - Acest lucru simplifică modelarea (un aspect cheie al paradigmei OO)

Computer Science

oop1 - T.U. Cluj - M. Joldos

27



Punctele tari ale paradigmei OO (2)

- Obiectele bine proiectate sunt unități independente
 - Tot ce se referă la obiectul real modelat este în obiect — *încapsulare*
 - Comunicarea se face prin schimb de *mesaje*
 - Această independență este augmentată prin *proiectarea dirijată de responsabilitate*

Computer Science

oop1 - T.U. Cluj - M. Joldos

28



Punctele tari ale paradigmei OO (3)

- Un produs clasic constă d.p.d.v conceptual dintr-o singură unitate (deși poate fi implementată ca un set de module)
 - Paradigma OO reduce complexitatea deoarece produsul constă, în general, din unități independente
- Paradigma OO promovează reutilizarea
 - Obiectele sunt entități independente

Computer Science

oop1 - T.U. Cluj - M. Joldos

29



Programarea orientată pe obiecte

- Oferă suport sintactic pentru tipurile de date abstracte
- Oferă facilități asociate cu ierarhiile de clase
- Schimbă punctul de vedere: programele sunt appendice ale datelor
- Introduc un concept nou: *obiect* = tip de dată abstract cu *stare* (atribute) și *comportament* (operații)

Computer Science

oop1 - T.U. Cluj - M. Joldos

30



Concepte OOP

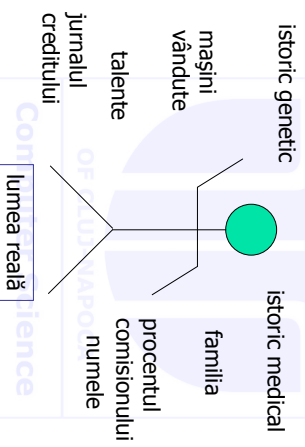
- Ne ocupăm doar de datele care prezintă interes pentru problema noastră.
- Acest proces de filtrare a detaliilor neimportante ale obiectului astfel încât să rămână doar caracteristicile importante este cunoscut sub numele de *abstractizare*.

00P1 - T.U. Cluj - M. Joldos

31



Abstractizare

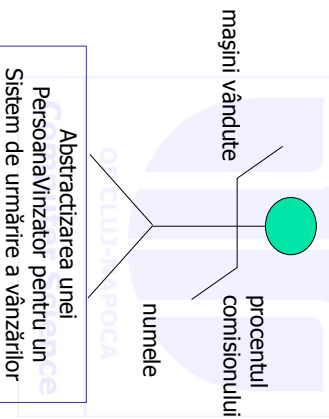


00P1 - T.U. Cluj - M. Joldos

32



Abstractizare

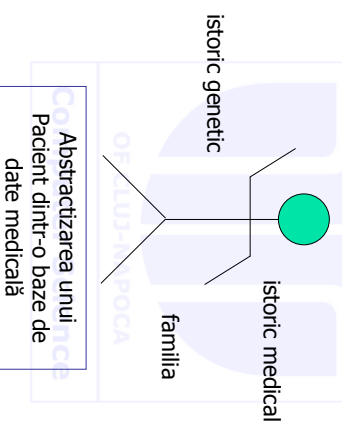


00P1 - T.U. Cluj - M. Joldos

33



Abstractizare



00P1 - T.U. Cluj - M. Joldos

34



Ce sunt obiectele software?

- Blocurile de construcție a sistemelor software
 - un program este o colecție de obiecte care interacționează
 - obiectele cooperează pentru a finaliza o sarcină
 - pentru aceasta, ele comunică trimițându-și "mesaje" unui altuia
- Obiectele modelează lucruri *tangibile*
 - persoană
 - bicicletă
 - ...

00P1 - T.U. Cluj - M. Joldos

35



Ce sunt obiectele software?

- Obiectele modelează *lucruri conceptuale*
 - întâlnire
 - dată calendaristică
- Obiectele modelează *proces*
 - aflarea drumului printr-un labirint
 - sortarea unui pachet de cărți de joc
- Obiectele au
 - capabilități: ce pot face, cum se comportă
 - proprietăți: trăsături (caracteristici) care descriu obiectele

00P1 - T.U. Cluj - M. Joldos

36



Capabilitățile obiectelor: acțiuni

- Obiectele au **capabilități** (*comportamente*) care le permit să efectueze acțiuni specifice
- obiectele sunt deștepte — ele "știu" cum să facă anumite lucruri
- un obiect face ceva doar dacă un alt obiect îi spune să-și folosească una dintre capabilități
- Capabilitățile pot fi:
 - **constructori**: stabilesc starea inițială a proprietăților obiectului
 - **comenzi**: modifică proprietățile obiectului
 - **interogări**: furnizează răspunsuri bazate pe proprietățile obiectului

00P1 - T.U. Cluj - M. Joldos

37



Capabilitățile obiectelor: acțiuni

- Exemple: *borcanele cu gem* sunt capabile să efectueze acțiuni specifice
 - **constructor**: să fie creat
 - **comenzi**: adaugă gem, golește-te
 - **interogări**: răspunde dacă este închis sau deschis capacul, dacă borcanul este plin sau gol

00P1 - T.U. Cluj - M. Joldos

38



Proprietățile obiectului: starea

- **Proprietățile** determină cum acționează un obiect
- unele proprietăți pot fi constante, iar altele variabile
- proprietățile însele sunt obiecte — pot și primi mesaje
- capacul *borcanului cu gem* și gemul în sine sunt obiecte
- Proprietățile pot fi:
 - **atribute**: lucruri care ajută la descrierea unui obiect
 - **componente**: lucruri care sunt "parte a" unui obiect
 - **asocieri**: lucruri despre care știe un obiect, dar care nu sunt parte a acelui obiect

00P1 - T.U. Cluj - M. Joldos

39



Proprietățile obiectului: starea

- **Stare**: colecție a tuturor proprietăților obiectului; se schimbă dacă o proprietate se schimbă
- unele nu se schimbă, d.e. volanul unei mașini
- altele se schimbă, d.e. culoarea mașinii
- **Exemplu**: proprietățile *borcanelor cu gem*
 - **atribute**: culoare, material, miros
 - **componente**: capac, container, etichetă
 - **asocieri**: un *borcan cu gem* poate fi asociat cu încăperea în care se află

00P1 - T.U. Cluj - M. Joldos

40



Clase și instanțe

- Concepția noastră curentă: fiecare obiect corespunde direct unui *anumit* obiect din realitate, d.e., un atom sau un automobil anume
- Dezavantaj: este mult prea nepractic să lucrăm cu obiecte în acest fel deoarece
 - ele pot fi infinit de multe
 - nu dorim să descriem fiecare individ separat, deoarece ei au multe lucruri în comun
- Clasificarea obiectelor scotește în evidență ce este comun între mulțimi de obiecte similare
 - mai întâi să descriem ce este comun
 - apoi să "ștampilăm" oricâte copii

00P1 - T.U. Cluj - M. Joldos

41



Clase ale obiectelor

- **Clasa unui obiect**
 - categoria obiectului
 - definește capabilitățile și proprietățile comune unei mulțimi de obiecte individuale
 - toate *borcanele cu gem* se pot deschide, închide și goli
 - definește un șablon pentru crearea de *instanțe de obiect*
 - unele *borcane cu gem* pot fi din plastic, pot fi colorate, de o anumită mărime etc.

00P1 - T.U. Cluj - M. Joldos

42



Clase ale obiectelor

- Clasele implementează **capabilitățile ca metode**
 - secvențe de instrucțiuni în Java
 - obiectele cooperează trimițând mesaje altor obiecte
 - fiecare mesaj "invocă o metodă"
- Clasele implementează **proprietățile ca variabile instanță**
 - locație de memorie alocată obiectului, care poate păstra o valoare care se poate schimba

OOPI - T.U. Cluj - M. Joldos

43



Instanțe de obiect

- **Instanțele obiectelor** sunt obiecte individuale
 - realizate din șablonul clasei
 - o clasă poate reprezenta un număr nedefinit de instanțe de obiect
 - realizarea unei instanțe de obiect constituie **instanțierea** obiectului respectiv
- **Prescurtare:**
 - clasă: clasa obiectului
 - instanță: instanța obiectului (a nu se confunda cu variabilele instanță)

OOPI - T.U. Cluj - M. Joldos

44



Instanțe de obiect

- Instanțe diferite ale, de e., clasei **BorcanCuGem** pot avea:
 - culoare și poziție diferită
 - diverse tipuri de gem în interior
- Astfel că, **variabilele instanță** ale lor au valori diferite
 - Notă: instanțele de obiect conțin variabile instanță — două moduri de folosire diferite, dar înrudite, a cuvântului **instanță**
- **Instanțele individuale au identități individuale**
 - aceasta permite altor obiecte să trimită mesaje unui obiect dat
 - fiecare instanță este unică, chiar dacă are aceleași capabilități
 - gândiți-vă la dăsa studenților de la acest curs

OOPI - T.U. Cluj - M. Joldos

45



Mesaje pentru comunicarea între obiecte

- Nici o instanță nu este o insulă — ea trebuie să comunice cu altele pentru a-și realiza sarcina
 - proprietățile le permit să știe despre alte obiecte
- Instanțele trimit mesaje una alteia pentru a invoca o capabilitate (adică, pentru a executa o sarcină)
 - metoda reprezentă codul care implementează mesajul
 - spunem "apelează metoda" în loc de "invocă capabilitatea" **Computer Science**

OOPI - T.U. Cluj - M. Joldos

46



Mesaje pentru comunicarea între obiecte

- Fiecare mesaj necesită:
 - **un emițător (expeditor):** obiectul care inițiază acțiunea
 - **un receptor:** instanța a cărei metodă este invocată
 - **numele mesajului:** numele metodei apelate
 - opțional **parametri:** informații suplimentare necesitate de metodă pentru a opera
 - mai multe mai târziu
- Receptorul poate (dar nu este nevoit) sa trimită un răspuns
 - vom trata **tipurile returnate** în detaliu, mai târziu

OOPI - T.U. Cluj - M. Joldos

47



Încapsulare

- Un automobil **încapsulează** multă informație
 - chiar literal, prin complexitatea construcției sale
- Dar nu este nevoie să știi cum funcționează o mașină pentru a o conduce
 - Volanul și schimbarea vitezelor constituie interfața
 - motorul, transmisia, axul cardanic, roțile, . . . , sunt implementarea (ascunsă) **Computer Science**

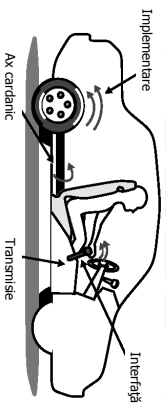
OOPI - T.U. Cluj - M. Joldos

48



Încapsulare

- Asemănător, nu e nevoie să știm cum funcționează un obiect pentru a-i trimite mesaje
- Dar, este nevoie să știm ce mesaje înțelege (adică, care îi sunt capabilitățile)
- clasa instanței determină ce mesaje îi pot fi trimise



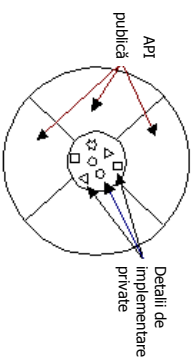
oop1 - T.U. Cluj - M. Joldos

49



Încapsularea

- Închiderea datelor într-un obiect
- Datele nu pot fi accesate direct din afară
- Oferă securitatea datelor



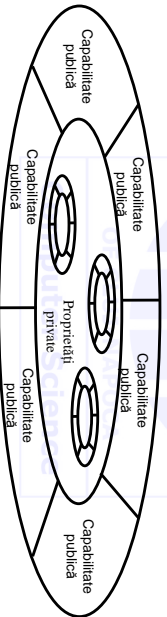
oop1 - T.U. Cluj - M. Joldos

50



Vederile unei clase

- Obiectele separă **interfața de implementare**
 - obiectul este "cutie neagră", ascunde funcționarea și părțile interne
 - interfața protejează implementarea împotriva utilizării greșite



oop1 - T.U. Cluj - M. Joldos

51



Vederile unei clase

- Interfața: vedere **publică**
 - permite instanțelor să coopereze unele cu altele fără a ști prea multe detalii
 - ca un **contract**: constă dintr-o listă de capabilități și documentație pentru cum să fie folosite
- Implementarea: vedere **privată**
 - proprietățile care ajută capabilitățile să-și îndeplinească sarcinile

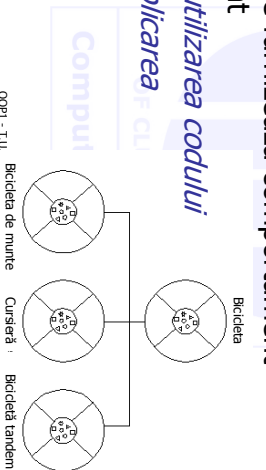
oop1 - T.U. Cluj - M. Joldos

52



Moștenirea

- O clasă (**subclasă**) poate **moșteni** atribute și metode dintr-o altă clasă (**superclasă**)
- Subclasele furnizează comportament specializat
- Oferă **reutilizarea codului**
- Evită **duplicarea** datelor



oop1 - T.U.

Bicicleta de munte

Cursară
Bicicleta tandem



Polimorfism

- Abilitatea de a lua multe forme
- **Aceeasi metodă** folosită într-o superclasă poate fi **suprascrisă** în subclase pentru a da o **funcționalitate diferită**
- De. Superclasa 'Poligon' are o metodă numită, **afisaSuprafata**
afisaSuprafata în subclasa 'Triunghi' → $a = x * y / 2$
afisaSuprafata in subclasa 'Dreptunghi' → $a = x * y$

oop1 - T.U. Cluj - M. Joldos

54



Concepte prezentate

- Paradigme în programare
 - structurată
 - orientată pe obiecte
- Abstracție
- tip de dată abstract
- Obiect
 - capabilități (comportamente):
 - acțiuni
 - constructor
 - comenzi
 - interogații
 - proprietăți: stare
 - atribute
 - componente
 - asocieri
 - instanță a unei clase
- Comunicarea între obiecte:
 - mesaje
 - Încapsulare
 - Clasă
 - șablon pentru crearea obiectelor
 - capabilitățile obiectelor:
 - implementate ca metode
 - proprietățile obiectelor: variabile
 - instanță
 - interfață – vedere publică
 - implementare – vedere privată
 - Moștenire
 - superclasă, subclasă
 - Polimorfism

OF CLUJ-NAPCA
Computer Science