



UNIVERSITATEA BABEȘ-BOLYAI  
Facultatea de Matematică și Informatică



# Programare orientată obiect

---

Curs 08

Laura Dioșan

# POO

---

## □ Clase

- Clase abstracte
- Polimorfism
- Legare timpurie și târzie
- Mecanismul virtual
- Metode pure
- UML
- Avantajele polimorfismului

# Polimorfism

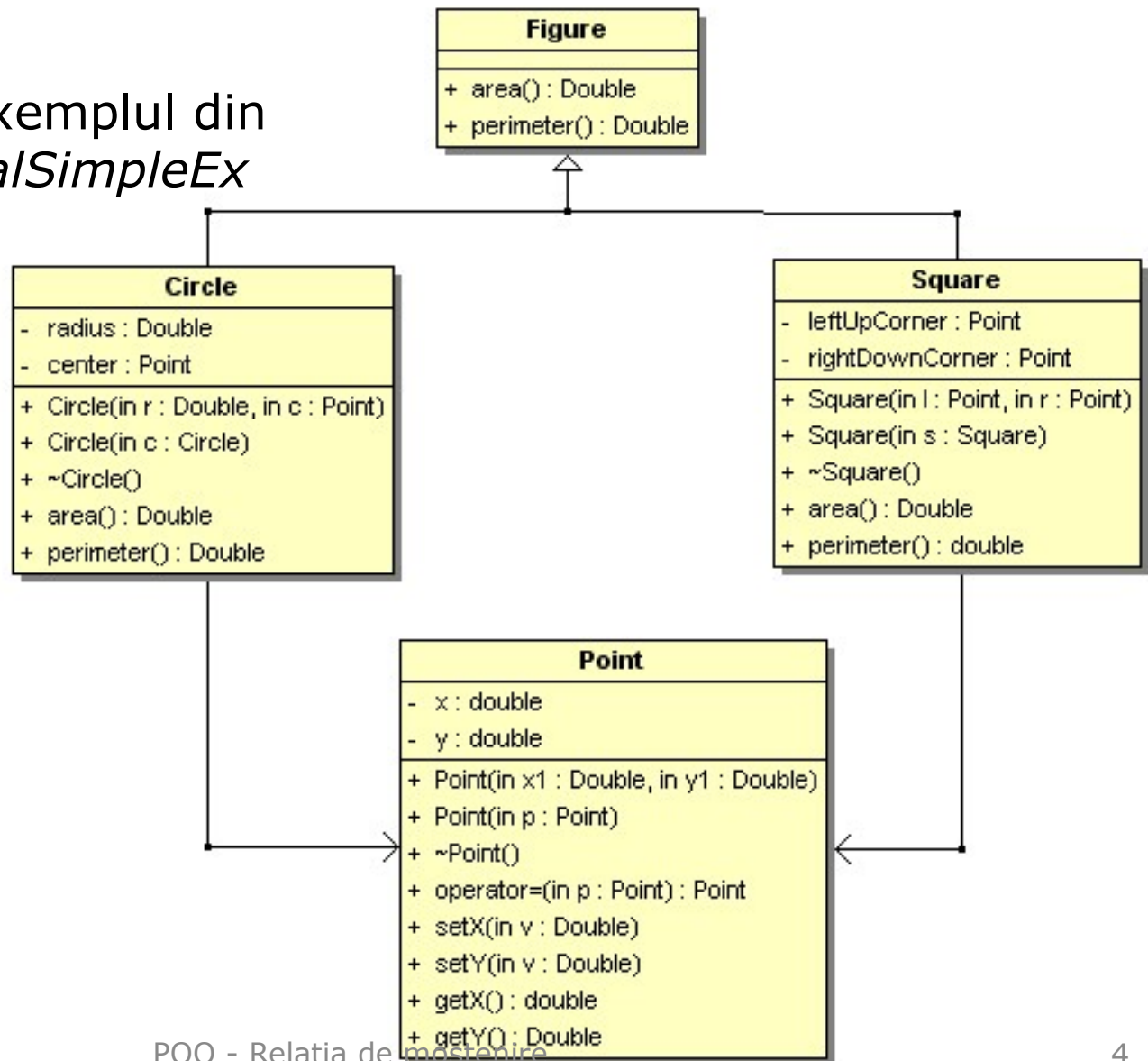
---

- Proprietatea unor entități de:
  - A se comporta diferit în funcție de starea lor
  - A reacționa diferit la același mesaj
  
- Proprietate a unui limbaj OO de a permite manipularea unor obiecte diferite prin intermediul unei interfețe comune
  - Există o relație de derivare între interfață și celelalte clase

# Exemplu

□ A se consulta exemplul din directorul *VirtualSimpleEx*

- Figure.h
- Point.h
- Circle.h
- Square.h



# Explicații

---

- Metodele unei clase
  - Clasice – adresa metodei:
    - este fixată la compilare/link-editare
    - nu poate fi modificată în timpul rulării
  - Polimorfice – adresa metodei:
    - poate fi modificată în timpul rulării
  
- Legare (binding)
  - Conectarea unui apel de funcție cu corpul funcției
  
- Legare timpurie (legare statică) - *early binding*
  - când legarea se realizează înainte de rularea programului (de către compilator și link-editor)
  - mecanism implicit
  
- Legare târzie (legare dinamică sau la rulare) - *late binding*
  - când legarea are loc la rulare, bazându-se pe tipul obiectelor
  - cauzată de folosirea funcțiilor virtuale

# Funcții virtuale

---

- ❑ Legarea târzie apare doar:
  - prin folosirea metodelor virtuale în clase de bază
  - când o adresă a clasei de bază (unde se află acele funcții virtuale) este folosită

- ❑ Definire

```
virtual type fc_name(fpl);
```

- ❑ Cuvântul rezervat *virtual* se folosește:
  - doar pentru declararea funcției (nu și pentru definirea ei)
  - doar în clasa de bază (deoarece acea funcție virtuală va fi virtuală în toate clasele derivate)
  - doar funcțiile membre pot fi virtuale
  - funcțiile globale prietene nu pot fi virtuale
- ❑ O metodă virtuală din clasa de bază poate fi:
  - Moștenită în clasa(ele) derivată(e)
  - Redefinită în clasa(ele) derivată(e) → suprascriere (*overriding*)
    - ❑ redefinirea unei funcții virtuale într-o clasă derivată

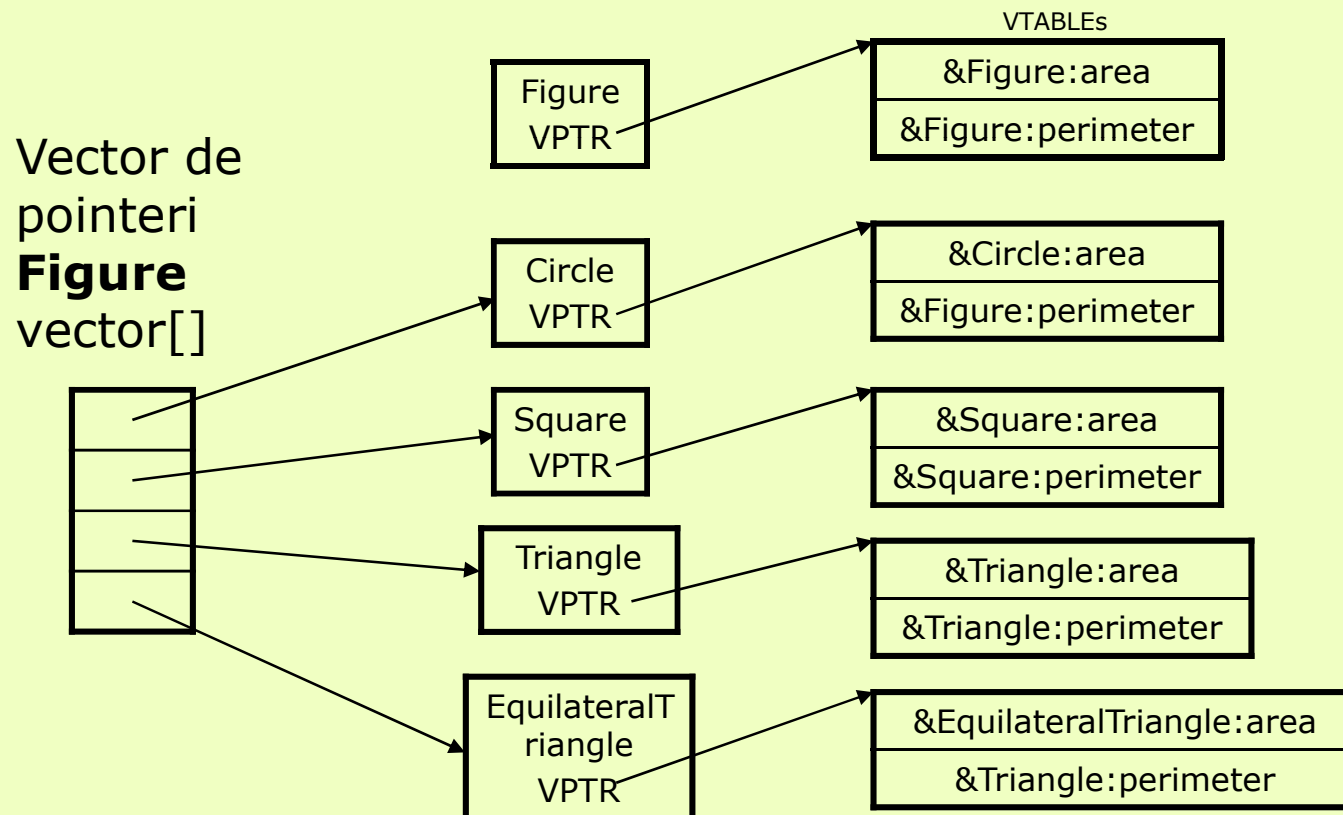
# Mecanismul legării întârziate

---

- necesită
  - Funcții virtuale
  - Referințe sau pointeri la clasa de bază
  
- Compilatorul:
  - Crează o singură tabelă (numită VTABLE) pentru fiecare clasă care conține funcții virtuale
  - Plasează adresele funcțiilor virtuale ale clasei respective în VTABLE
  - Plasează (în mod secret) un pointer în fiecare clasă cu funcții virtuale, numit *vpointer* (abreviat VPTR), care referă VTABLE-ul acelui obiect
  
- Când o funcție virtuală este apelată printr-un pointer la clasa de bază (**apel polimorfic**), compilatorul:
  - inserează cod pentru plasarea VPTR și
  - se uită la adresa funcției în VTABLE, apelând astfel funcția corectă – are loc legarea întârziată

# Mecanismul legării întârziate

- A se consulta exemplul din directorul *VirtualSimpleEx*
  - Figure.h, Point.h
  - Circle.h, Square.h, Triangle.h, EquilateralTriangle.h





# De ce legare întârziată?

---

- Funcțiile virtuale – o opțiune
  - Care nu este chiar eficientă
    - 2 apeluri sofisticate (în locul unui simplu apel la o adresă absolută)
  - C++ provine din moștenirea C-ului, unde eficiența este critică
  - Cuvântul rezervat **virtual** este folosit pentru eficiența prezentării (adaptării)

# Funcții virtuale – mai multe detalii

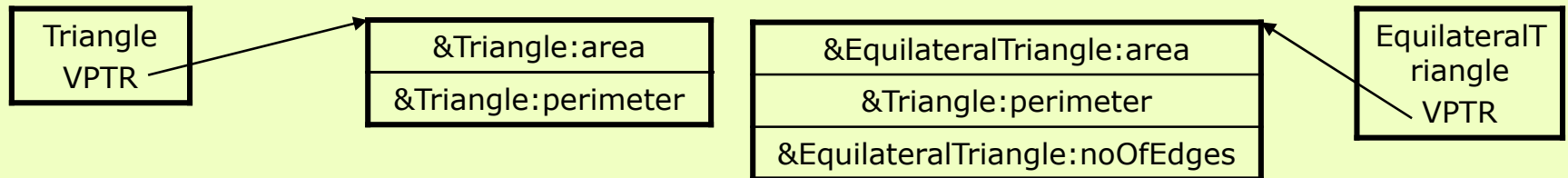
---

- Nu pot să fie virtuali
  - constructorii
    - VPTR este instanțiat în cadrul constructorului
  - metodele statice
  - funcțiile globale prietene
  
- Pot să fie virtuale
  - metodele ordinare
  - metodele prietene
  - destructorii
    - uneori aceștia TREBUIE să fie virtuali

# Funcții virtuale – mai multe detalii

---

- A se consulta exemplul din directorul *VirtualConstrDestructor*
- Moștenirea și VTABLE
  - Funcții virtuale noi în clasele derivate
  - Downcast
  - A se consulta exemplul din directorul *VirtualSimpleEx*
    - *Figure.h, Triangle.h, EquilateralTraingle*
    - *Test.cpp* and *addNewVirtualFunctionInDerivedClass()*



# Metode nule

---

- Metode cu corpul vid
  - Declarate in clasa de **bază**
  - Definite în clasa de **bază**
- Metode folosite pentru declararea unui comportament general (abstract) la nivelul clasei de bază

- Definire

```
virtual type fc_name(fpl) {  
};
```

# Metode pure

---

- Metode fără corp
  - Declarate în clasa de **bază**
  - Definite în clasa(ele) **derivată(e)**
- Metode folosite pentru declararea unui comportament general (abstract) la nivelul clasei de bază

- Definire

```
virtual type fc_name(fpl) = 0;
```

- Compilatorul rezervează un *slot* pentru funcția pură în VTABLE, dar nu pune (încă) o adresă concretă în acel slot
  - Chiar dacă doar o singură metodă a unei clase este virtuală pură, VTABLE este incompletă
- Nu pot fi apelate
  - Scopul lor este de a anunța doar semnatura metodei

# Metode pure - exemplu

---

```
class Figure{
public:
    virtual ~Figure(){}
    virtual float area() = 0;
    virtual float perimeter() = 0;
};

class Circle : public Figure{
private:
    float radius;
    Point center;
public:
    Circle(float r, Point &c):radius(r), center(c){
    Circle(const Circle &c){
        radius = c.radius;
        center = c.center;
    }
    ~Circle(){}
    float area(){
        cout << "[Circle] : area " << endl;
        return PI * radius * radius;
    }
    float perimeter(){
        cout << "[Circle] : perimeter " << endl;
        return 2 * PI * radius;
    }
};

void printArea(Figure* f){
    cout << f->area() << endl;
}

void printPerimeter(Figure* f){
    cout << f->perimeter() << endl;
}

void virtualSimpleDerived(){
    Figure* figPointer;
    figPointer = new Circle(5, Point(2,3));
    printArea(figPointer);
    printPerimeter(figPointer);
}

int main(){
    virtualSimpleDerived();
    return 0;
}
```

# Clase abstracte

---

- ❑ O clasă cu cel puțin o metodă pură virtuală
- ❑ Folosite pentru a declara un comportament comun mai multor clase derivate (din clasa abstractă)
- ❑ O clasă abstractă poate să conțină și:
  - Date comune (tuturor claselor derivate)
  - Metode comune (tuturor claselor derivate)

# Clase abstracte

---

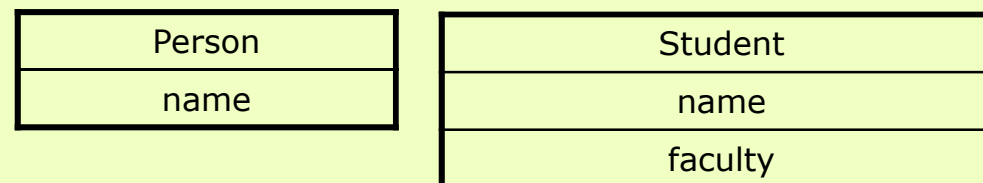
- O clasă abstractă nu poate fi instanțiată
  - Amintiți-vă:
    - Compilatorul rezervă un *slot* pentru funcția pură în VTABLE, dar nu pune (încă) o adresă concretă în acel slot
    - Chiar dacă doar o singură metodă a unei clase este virtuală pură, VTABLE este incompletă
  - Întotdeauna trebuie folosit un pointer sau o referință la clasa abstractă
  
- Funcțiile virtuale pure:
  - Sunt utile pentru că:
    - Ele explicitiază abstractul unei clase
    - Indică utilizatorului și compilatorului ceea ce se dorește a fi făcut
  - Previn transmiterea prin valoare a claselor abstracte
    - O modalitate de a evita felierea obiectelor (*object slicing*)



# Object slicing

---

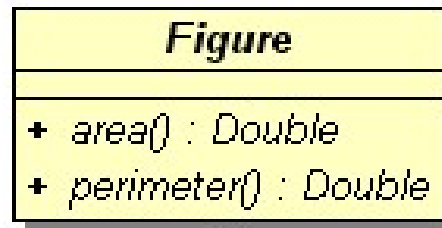
- ❑ A se consulta directorul *ObjectSlicing*
- ❑ Pentru că **print( )** acceptă un obiect **Person** (în loc de un pointer sau o referință), orice apel al funcției **print( )** va determina punerea pe stivă a unui obiect de dimensiunea **Person** (care va fi curățat la sfârșitul funcției)
- ❑ Dacă un obiect al unei clase moștenite din clasa **Person** (e.g. **Student**) este transmis funcției **print( )**, compilatorul îl va accepta, dar va copia pe stivă doar partea corespunzătoare tipului **Person** din acel obiect
  - Se folosește constructorul de copiere al clasei **Person**, care inițializează VPTR cu VTABLE al clasei **Person** și copiază doar partea corespunzătoare clasei **Person** din acel obiect
- ❑ Se feliază și se pierde partea necomună



# Notăție UML

---

## □ Fonturi italice



# Polimorfism

---

## □ Avantaje

- Reutilizarea codului
- Extensibilitate

## □ Funcții polimorfice

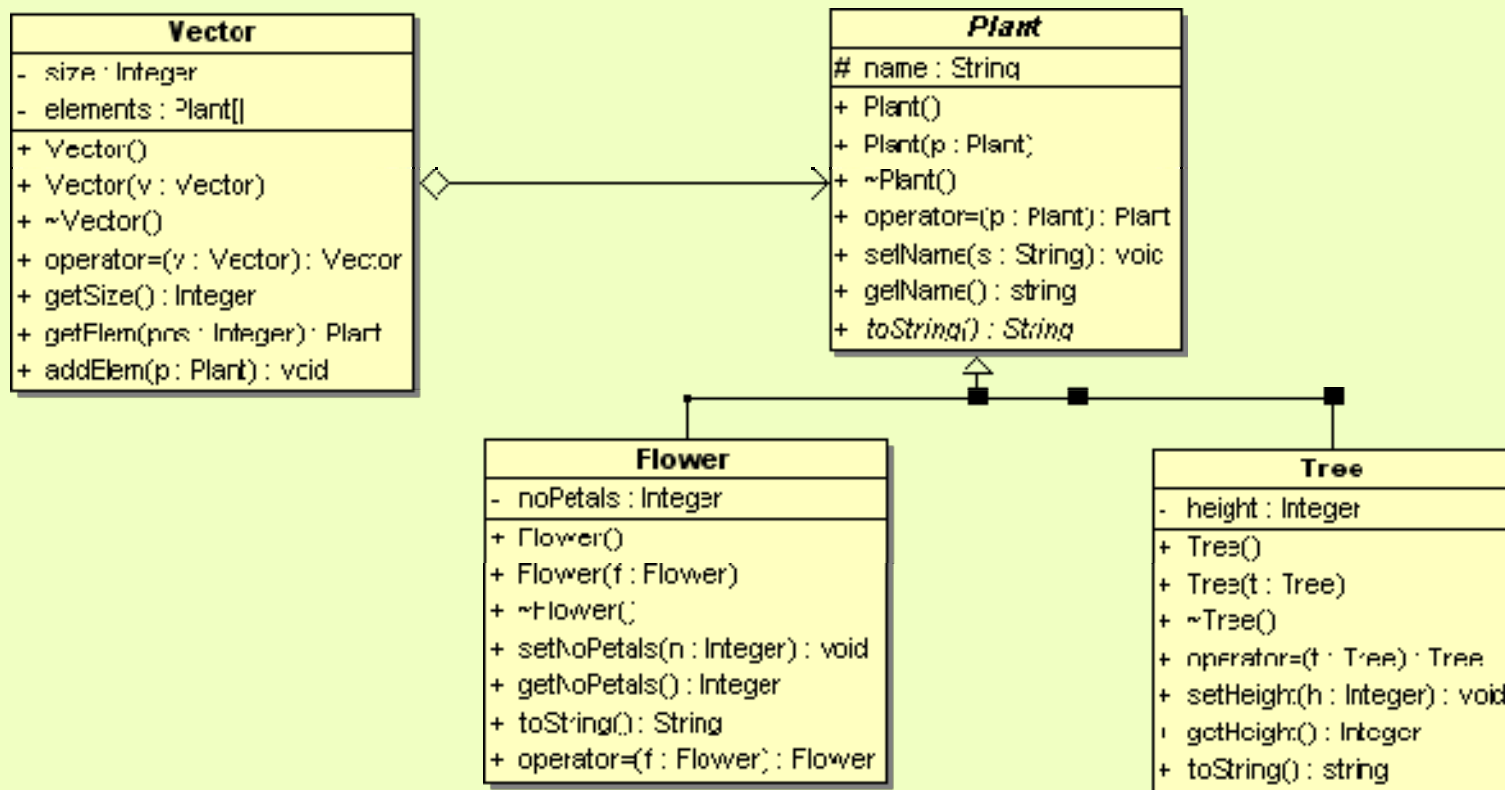
- Parametrii sunt pointeri la obiecte

## □ Structuri de date polimorfice

- void\*
- Clase abstracte

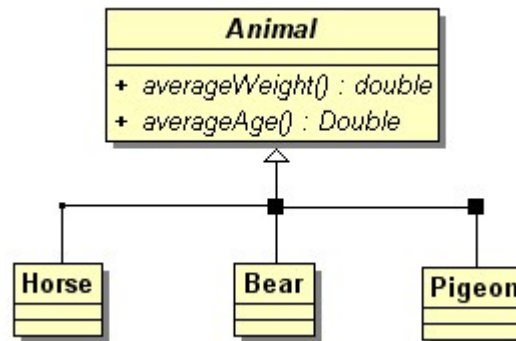
# Polimorfism

- A se consulta exemplul din directorul *inheritanceVectorPolymorphic*



# Temă

- Implementați următoarea diagramă UML:
  - Fiecare animal se caracterizează prin:
    - Greutate și greutate medie (a acelei categorii)
    - Vârstă și vârstă medie (a acelei categorii)
  - Creați mai multe animale și afișați informațiile despre ele.



# Cursul următor

---

- Clase
  - Polimorfism
  - Interfețe
  - Colecții cu elemente generice