Programare orientată obiect

Cursul 6

Sumar

- Supraîncărcarea operatorilor (2)
  - Alți operatori:
    - conversie (cast),
    - funcție **()**
    - indexare (**[]**)
    - virgulă (**,**)
    - gestiunea memoriei (new, delete)
- Conversii între obiecte
- Moștenire/derivare (1)
  - Elemente de bază

Alți operatori: conversie (cast)

- Prototip:
  - **operator tip();**
- Utilizat pentru conversii între obiecte și alte tipuri de date
- Tipul returnat = tipul conversiei
  - Nu are tip explicit

Alți operatori: conversie (cast)

```
class Complex
{
public:
        operator float();
};
//…
Complex::operator float()
{
        return re;
}
```

```
//…
Complex c;
float f;
f = c;
```

Conversii (obiecte)

- **Obiect** -> alt tip de dată (fundamendal, definit de utilizator)
  - Operatorul de cast supraîncărcat
- Alt tip de dată (fundamendal, definit de utilizator) -> **obiect**
  - Constructorul cu un parametru

Alți operatori: funcție ()

- Prototip:
  - **tip operator()(arg);**
- Funcție membră
- Poate avea oricîți parametri
- Posibilitatea utilizării obiectelor ca funcții
- Clase specializate: *functor*
- Permit utilizarea membrilor clasei (stări)

Alți operatori: funcție ()

```
class MaiMare
{
        int limita;
  public:
        MaiMare(int limita) : limita (lim) {}
        bool operator()(int numar) { return numar > limita; }
        void stabilesteLimita(int lim) {limita = lim; }
};

MaiMare esteMaiMare(100);
if (esteMaiMare(60)) { … }
```

Alți operatori: indexare ([])

- Prototip:
  - **Tip& operator[](tip);**
  - **const Tip& operator[](tip) const;**
- Funcție membră
- Accesul la membrii unei colecții etc.
  - Citire/scriere

Alți operatori: indexare ([]) (1)

```
class Vector
{
        int *data;
        int n;
public:
        int & operator[](int);
//..
};
```

```
int & Vector::operator[](int i)
{
        if ((data != NULL) &&
                (i >= 0 ) && (i < n))
                return data[i];
        else
                throw IOOB;
}
```

Alți operatori: indexare ([]) (2)

```
Vector v;
//
try
{
        cout<<"Elementul "<<i<<" este"<<v[i]<<endl;
        //…
}
catch(TipExc ex)
{
        if (ex == IOOB)
                cout<<"Exceptie! Indexul este in afara limitelor!"<<endl;
}
```

Alți operatori: virgulă (,)

- Prototip:
  - **tip operator,(arg);**
- Comportamentul implicit:
  - returnează ultimul parametru
- Posibilitate de utilizare personalizată:

```
Vector v;
v  += 10, 2, 3, 4, 7;
```

Alți operatori: new

- Prototip:
  - **void *operator new (size_t);**
  - **void *operator new [](size_t);**
- Funcție statică (implicit) sau independentă (globală)

Alți operatori: delete

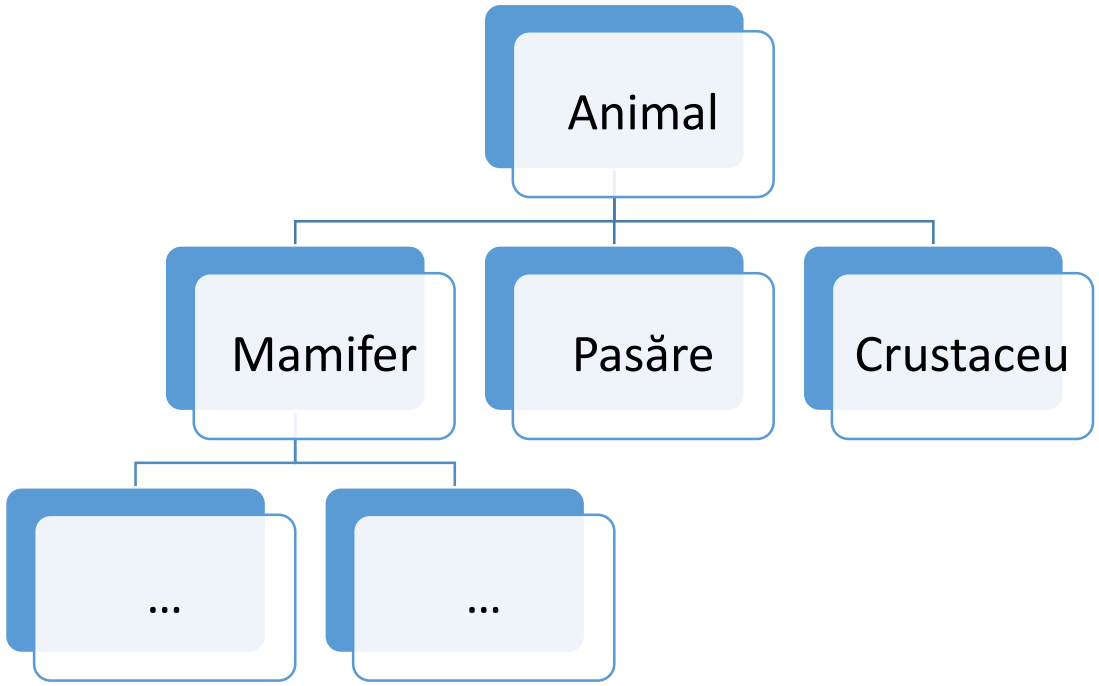
- Prototip:
  - **void operator delete (void *);**
  - **void operator delete [](void *);**
- Funcție statică (implicit) sau independentă (globală)

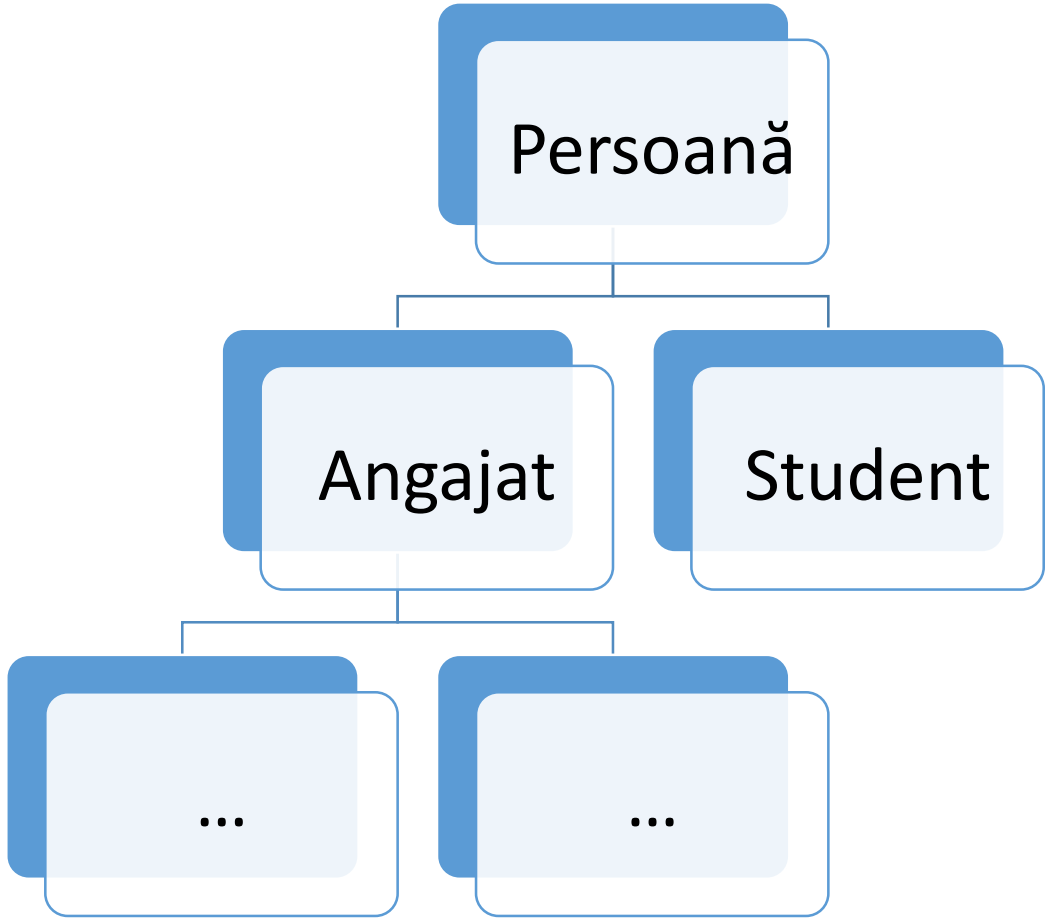
Operatorul de selecție (->)

- Prototip:
  - **Tip * operator->();**
- Implementata prin funcție membră
- Returnează adresa unui obiect de tipul clasei în care este implementat
- Utilizare
  - Accesul la membrii unei colecții (iterator)

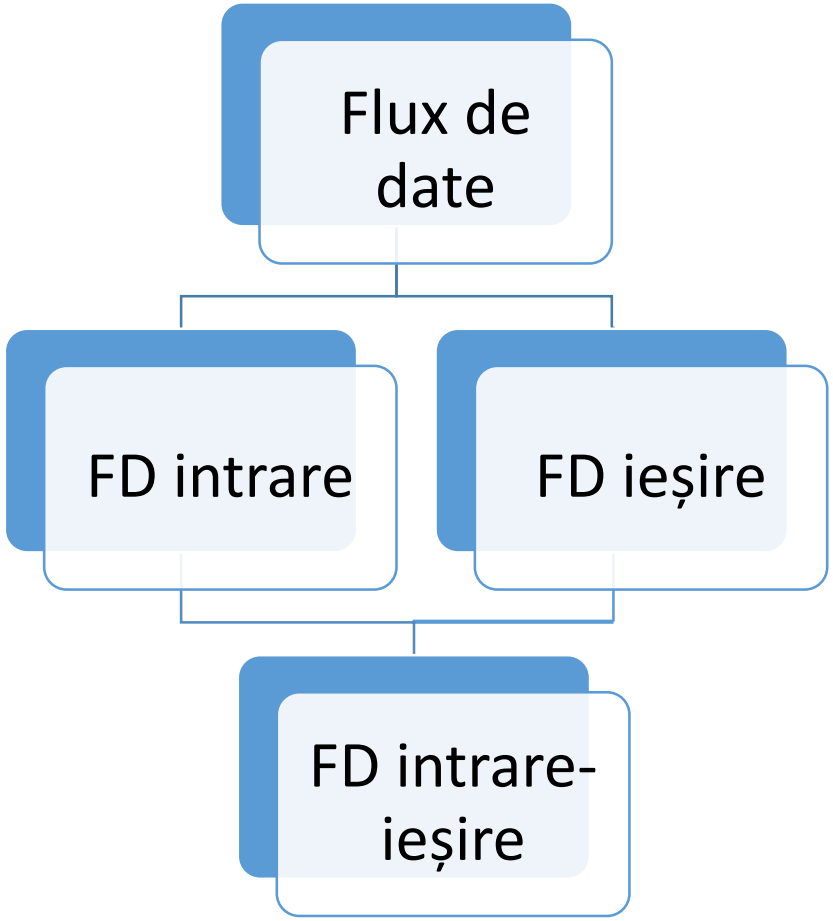
Moștenire/derivare

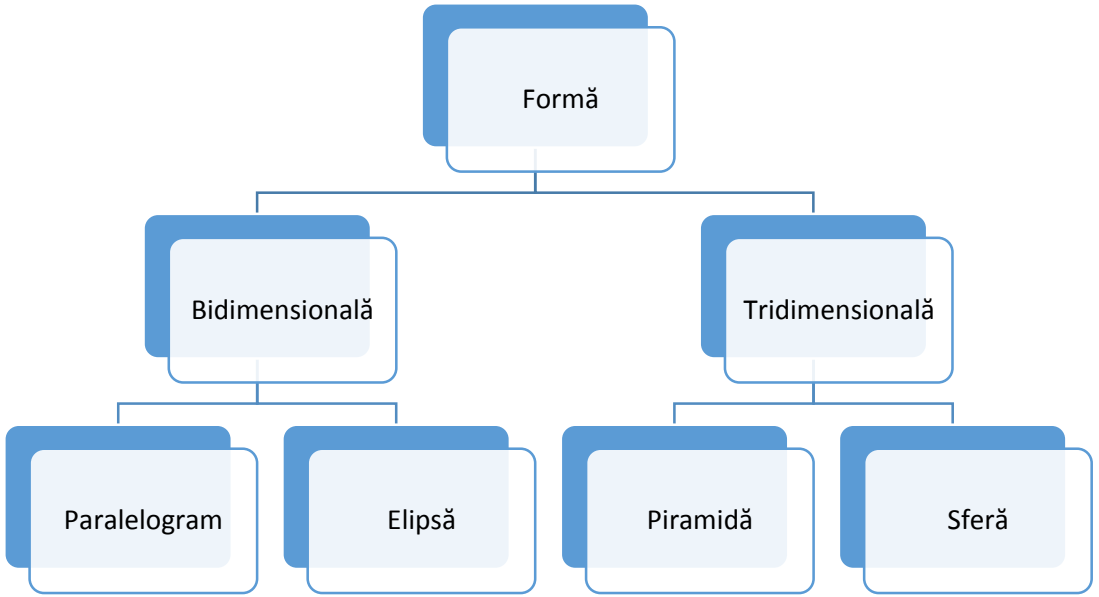
Moștenire/derivare



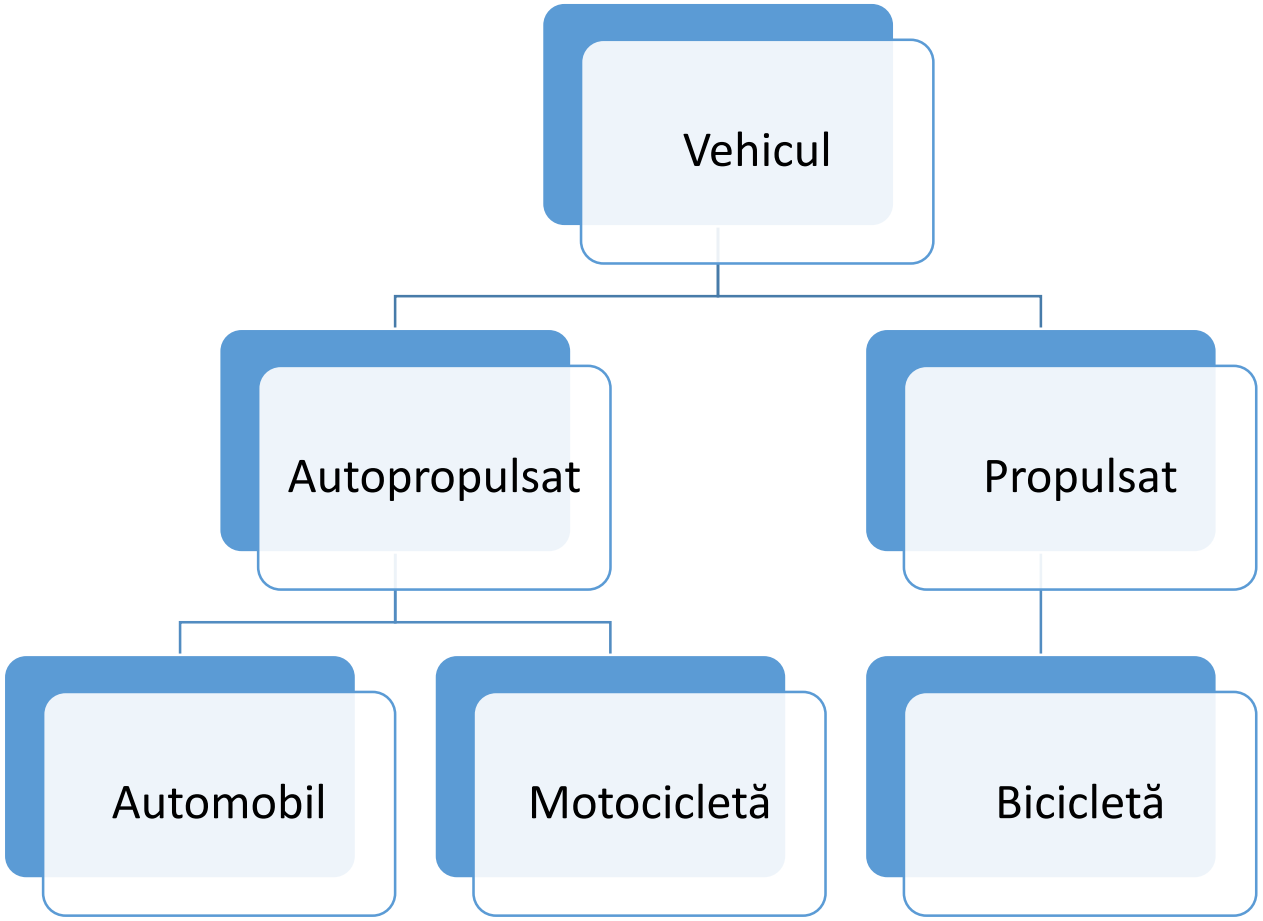


Moștenire/derivare





Moștenire/derivare

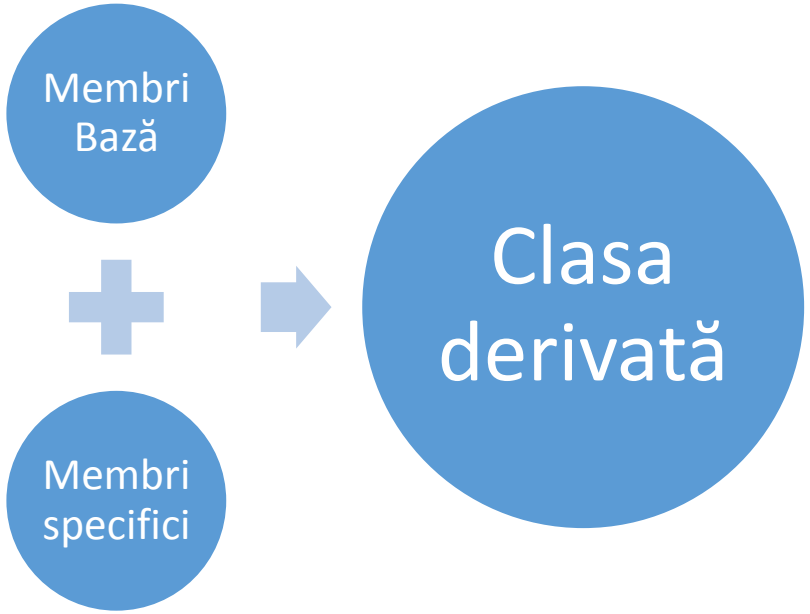


Moștenire/derivare

- O clasă (subclasă, derivată, copil) *moștenește* **toate atributele și metodele** unei alte clase (superclasă, de bază, părinte) sau mai multor clase
  - O clasă este *derivată* din altă clasă/alte clase
- Modalitate de reutilizarea a codului
- Relație ierarhică între clase
- Extinderea claselor -> specializare

Moștenire/derivare

- Abstractizare pentru
  - Partajarea similarităților între clase
  - Păstrarea diferențelor dintre clase



Relații între clase

Moștenire/derivare

ESTE-UN/O (IS-A)

- Automobilul *este un* autovehicul
- Lista criptată *este o* listă
- Contul curent *este un* cont bancar

Compunere

ARE-UN/O (HAS-A)

- Automobilul *are un* motor
- Documentul *are* paragrafe