Programare orientată obiect

Cursul 2, 3

rev 1.3/PP/20131022

Sumar

- Obiecte și clase
- Concepte POO
- Membri: variabile de instanță și metode
  - Funcții inline
- Constructori, Destructor
- Metode de acces la date membre
- Pointerul *this*
- Membri statici
- Funcții/Clase prietene (*friend*)
- Pointeri la membri

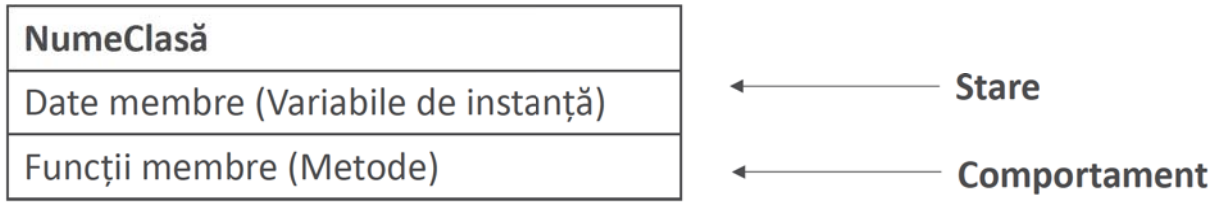
Obiecte (1)

- Modele ale entităților din lumea reală/virtuală
- Orice poate fi reprezentat ca un obiect
- Distincte unul față de altul
- Caracteristici [Booch]:
  - Stare – structura și valorile asociate
  - Comportament – mesaje care pot fi transmise
  - Identitate
- Reprezintă ceva concret

Clase

- Șablon pentru crearea de obiecte
- Tipuri *abstracte* de date
- Definite de utilizator
- Structura (date) și implementarea comportamentului (funcții)
- Clasa = Date + Operații
- Declarare, definire, utilizare

Clase


NumeClasă
Date membre (Variabile de instanță)
Funcții membre (Metode)
Stare
Comportament

Obiecte (2)

- Instanțe ale claselor
- Variabile de tipul claselor

POO

- Încapsulare
  - Ascunderea informației
- Polimorfism
- Moștenire
- Reutilizare

Încapsulare

- Gruparea datelor și a funcțiilor de prelucrare/acces
- Interfață limitată de acces la obiect
  - Controlul accesului
- Ascunderea detaliilor de implementare
- Asigurarea unei stări consistente a unui obiect

Date membre

- Variabile de instanță/Cîmpuri/Atribute
- Valorile acestora: starea unui obiect
- Recomandare: inaccesibile din afara clasei

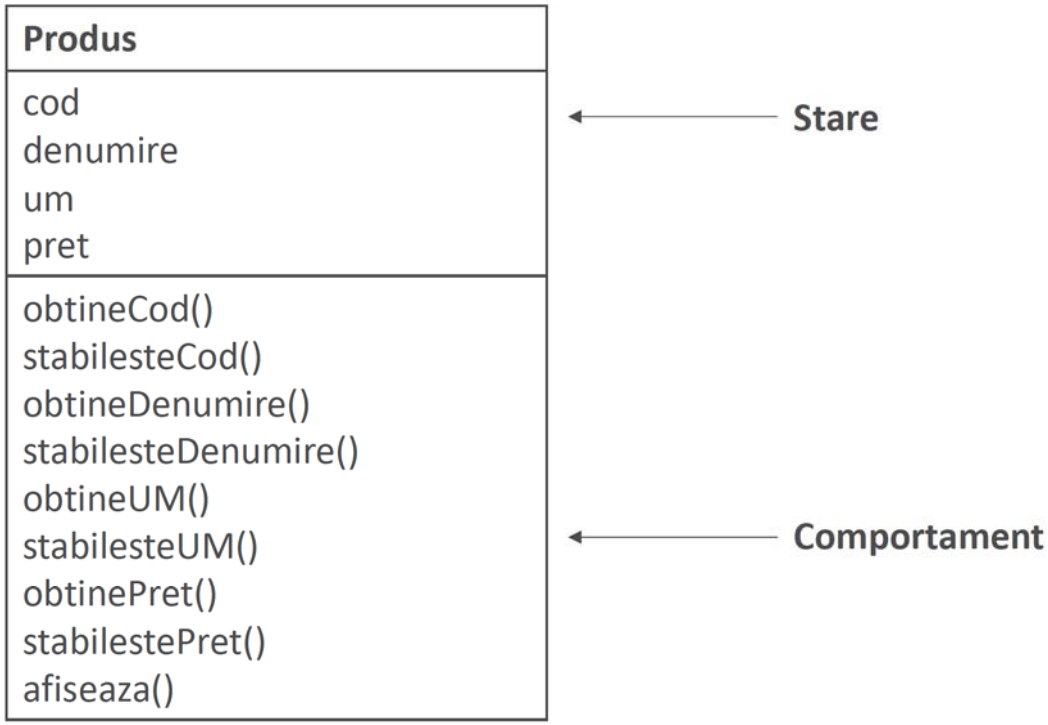
Funcții membre (Metode)

- Definesc comportamentul unui obiect
- Metodele accesibile din afara clasei: interfața unui obiect
- Apeluri de metode – transmitere de mesaje
- Metode speciale/dedicate:
  - Constructori
  - Destructor
  - De acces (get/set) – recomandat

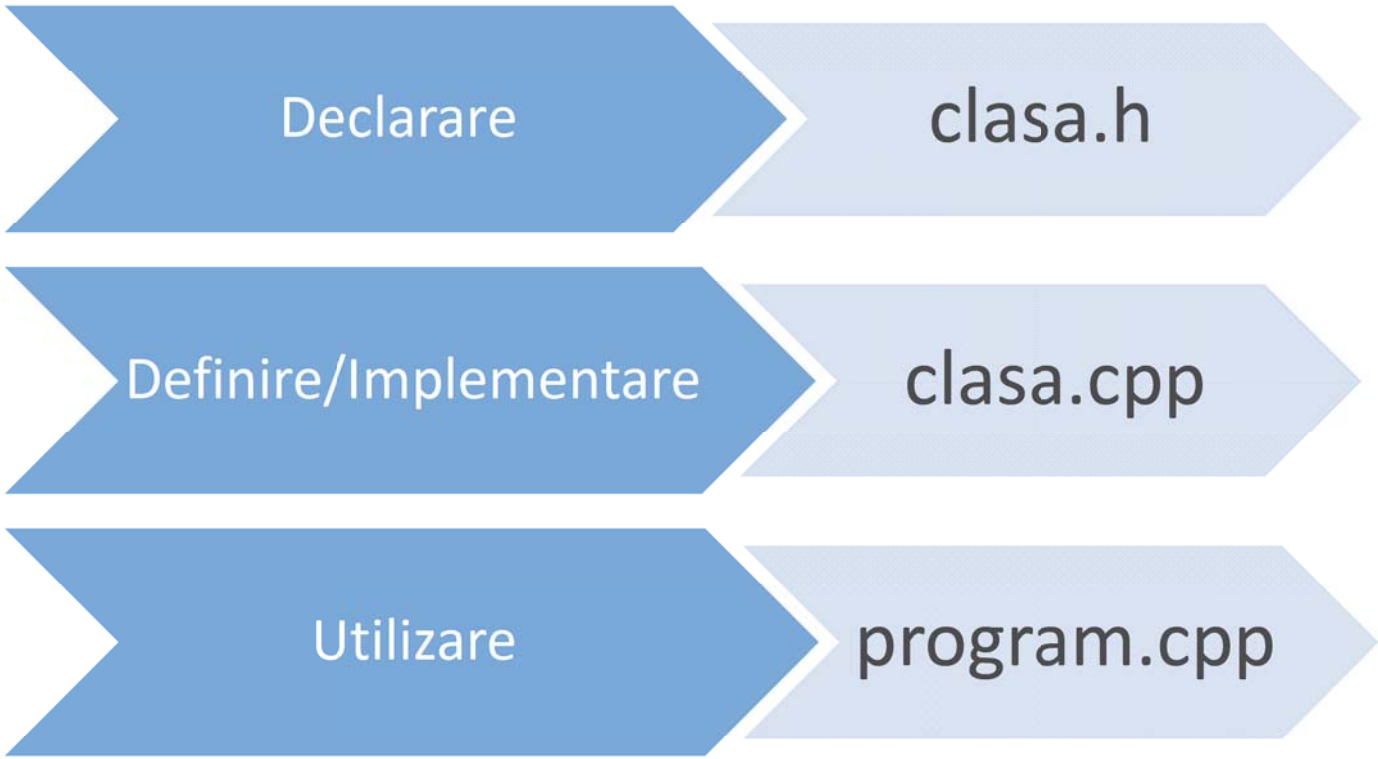
Funcții membre (Metode)

- Sînt declarate în cadrul clasei
- Uzual, se implementează în afara clasei (C++) – fișier sursă
- Pot fi implementate și în cadrul clasei (funcții *inline*) – fișier antet
  - Nerecomandat

Clase


Produs
cod
denumire
um
pret
obtineCod()
stabilesteCod()
obtineDenumire()
stabilesteDenumire()
obtineUM()
stabilesteUM()
obtinePret()
stabilestePret()
afiseaza()
Stare
Comportament

Clase în C++


Declarare
clasa.h
Definire/Implementare
clasa.cpp
Utilizare
program.cpp

Definire clase

```
class NumeClasa
{
        [modificator de acces 1]:
        //membri
        [modificator de acces 2]:
        //membri
        [modificator de acces 3]:
        //membri
        …
};
```

Modificatori de acces

- private
  - Implicit
- protected
- public

Implementare clase

```
Tip_returnat NumeClasa::metoda(param)
{
        //corp metoda
}
```

Referirea membrilor obiectelor

- **Definire obiecte**
  - Clasa ob;
  - Clasa *pOb = **new** Clasa();
- **Apel de metode**
  - ob.metoda();
  - pOb->metoda();
- **Acces la date membre**
  - ob.cimp;
  - pOb->cimp;

Exemplu

- Clasa Produs

Constructor

- Alocare spațiu obiecte
- Inițializare obiecte:
  - Variabile de instanță
  - Alocare memorie (!)
  - Deschidere fișiere (!)
- Pot exista oricîte versiuni (supraîncărcări)
- Are numele clasei , nu prezintă tip returnat și poate avea oricîți parametri:
  - **NumeClasa([parametri]) : [lista de inițializare];**

Constructor

- Constructor fără parametri (implicit)
  - Creat de către compilator dacă nu există nici un constructor
  - Apelat la definirea masivelor de obiecte
- Constructor cu un parametru
  - Utilizat la conversii implicite
- Constructor cu mai mulți parametri
- Constructor cu valori implicite
  - Atenție la ambiguități!

Constructor fără parametri

```
class Produs
{
        public:
                Produs();
};
...
Produs::Produs() { ...};
...
Produs prod;
Produs *pProd = new Produs();
```

Constructor cu un parametru

```
class Produs
{
                    float pret;
        public:
                    Produs(float pret);
                    Produs aduna(float pret);
};
…
Produs::Produs(float pretNou) : pret(pretNou) {… };
…
Produs prod(9.99f);
Produs prod1 = 10.5f;
…
prod.aduna(prod1);
```

Constructor cu mai mulți parametri

```
class Produs
{
                int cod;
                float pret;
        public:
                Produs(int, float);
};
...
Produs::Produs(int codNou, float pretNou) : cod(codNou), pret(pretNou) { };
...
Produs prod(10, 20.5f);
Produs *prod1 = new Produs(100, 0.5f);
```

Constructor cu valori implicite

```
class Produs
{
                    int cod;
                    float pret;
          public:
                    Produs(int cod, float pret =0);
};
…
Produs::Produs(int codNou, float pretNou) : cod(codNou), pret(pretNou) {…};
…
Produs prod(10, 20.5f);
Produs prod1(100);
Produs *prod3 = new Produs(100, 0.5f);
```

Destructor

- Metodă apelată la "distrugerea" unui obiect
- Metodă utilizată pentru:
  - eliberarea memoriei alocate pentru variabile de instanță
  - eliberarea resurselor (fișiere, obiecte grafice etc.)
  - închiderea conexiunilor (rețea, baze de date etc.)
- Poate exista maxim unul
- Generat implicit de către compilator
- Se apelează implicit la obiecte
- Se apelează explicit (prin *delete*) la pointeri la obiecte
- Are numele clasei precedat de simbolul **~**, nu prezintă tip returnat și nu are parametri:
  - **~NumeClasa()**;

Metode de acces

- Metode publice
- Permit accesul la variabile de instanță (R/W):
  - Obținerea valorilor (get) – *selector*
  - Modificarea valorilor (set) - *modificator*
    - Validare (Ex: vîrsta > 18, cantitate > 0 etc.)

Pointerul this

- Pointer special: adresa obiectului care apelează metoda
- Pointer **constant** transmis implicit **tuturor** metodelor nestatice
- Accesibil din orice metodă, permite referirea obiectului curent
- Exemplu:

```
void Produs::stabilestePret(float pretNou)
{
    pret = pretNou;
    //sau
    this->pret = pretNou;
}
```

Pointerul this

- În program:

  Produs prod;

  prod.afiseaza();
- Compilatorul:

  afiseaza(Produs * const this);
- Apel real:

  afiseaza(&prod);

Membri statici

- Modificatorul *static*
- Membrii statici sînt definiți la nivelul clasei
  - Nu aparțin nici unui obiect
- Utilizare
  - Gestiunea membrilor clasei
    - Număr, parcurgere, contorizare etc.
  - Modele de programare
    - Singleton

Date membre statice

- Declarare (în clasă):
  static tip dataMembraStatica;
- Definire (în afara clasei):
  tip Clasa::dataMembraStatica;
- Utilizare
  Clasa::dataMembraStatica;

Funcții statice

- Nu au nici un obiect asociat
- Nu primesc pointerul this!
- Operează pe date membre statice sau obiecte primite ca parametri
- Declarare (în clasă):

  static tip functieStatica(param);
- Definire (în afara clasei):

  tip Clasa::functieStatica(param) { }
- Utilizare

  Clasa::functieStatica(arg);

Membri constanți

- Date membre
  - Definire:
    - const tip dataConst;
  - Inițializare
    - Constructor: Clasa(param) **: dataConst(valConst)** {…}
- Metode
  - tip metoda(param) **const**;
    - Metoda nu modifică obiectul care o apelează
  - **const** tip metoda(param);
    - Metoda returnează o valoare constantă
  - **const** tip metoda(param) **const**;
    - Metoda nu modifică obiectul care o apelează și returnează o valoare constantă

Funcții/clase prietene

- Permit accesul la membrii de tip *private* și *protected*
- Relația de stabilește în clasa care acordă dreptul de acces
- Utilizare (oriunde în cadrul unei clase):
  - *friend* class ClasaPrietena;
    - *Toate* metodele din ClasaPrietena au access la membrii privați ai clasei
  - *friend* tip_ret functie(param);
    - Funcția *functie* are acces la membrii privați ai clasei

Pointeri la membri

- Accesul la membrii unei clase prin intermediul adreselor acestora
- Pointeri la variabile de instanță (date membre)
- Pointeri la metode
- Sînt inițializați cu deplasamentul unui cîmp/metodă în cadrul clasei

Pointeri la membri

- Definire:
  - Tip **Clasa::*pointerData;**
  - Tip **(Clasa::*pointerMetoda)(param);**
- Inițializare
  - **pointerData = &Clasa::dataMembra;**
  - **pointerMetoda = &Clasa::metoda;**
- Utilizare/Referire
  - Clasa obiect, *pObiect;
  - **obiect.*pointerData** *sau* **pObiect->*pointerData**
  - **(obiect.*pointerMetoda)(arg)** *sau* **(pObiect->*pointerMetoda)(arg)**

Pointeri la membri

- Definire
  - **int Produs::*pCod;**
- Inițializare
  - **pCod = &Produs::cod;**//cod este accesibil
- Utilizare
  - **Produs prod, *pProd;**
  - **prod.*pCod = 200;**
  - **pProd->.pCod = 300;**

Pointeri la membri

- Definire
  - **int (Produs::*pObtineCod)();**
- Inițializare
  - **pObtineCod = &Produs::obtineCod;**
- Utilizare
  - Produs prod, *pProd;
  - **int x = (prod.*obtineCod());**
  - **x = (pProd->*obtineCod());**