PROGRAMARE ORIENTATĂ OBIECT (C++) 2005-2006

Prof. univ. dr. G. GRIGORAŞ

www.infoiasi.ro/~grigoras/poo/poo.html

Exemple utilizate la curs:

http://thor.info.uaic.ro/~dlucanu/pc/pcII/html/ex_index.htm

BIBLIOGRAFIE

- **Liviu Negrescu :** *Limbajele C si C++ pentru incepatori*, vol II, III, Microinformatica, Cluj-Napoca, Ed.Libris 1997
- **H. Schildt**: C++ manual complet, Teora, 2000
- **D. Kaler, M.J. Tobler, J. Valter**: C++, Teora, 2000
- **Bjarne Stroustrup:** The C++ Programming Language, Adisson-Wesley, 3nd edition, 1997
- **Stanley B. Lippman:** *C++ Primer,* Addison Wesley, 1992
- **K.Jamsa**, Succes cu C++, Editura All 1997

Manuale electronice

http://www.infoiasi.ro/fcs/biblioteci.php
http://lib.info.uaic.ro/index.php

- **Peter Müller :** Introduction to Object-Oriented Programming Using C++
- **Bruce Eckel :** Thinking in C++, 2nd Edition
- ***** :** Online C++ tutorial

CERINȚE

- EVALUARE
  - Activitatea la laborator (AL), testele scrise (TS)
  - AL >= 6, TS >= 4
  - AL: fiecare tema de laborator va fi notata cu note de la 1 la 10
  - TS: 2 teste scrise (15 aprilie, 20 mai), fiecare test conținând 8 întrebări grilă şi 1-2 probleme.
  - formula notei finale: 40% AL + 60% TS
- Rexaminarea constă din:
  - **1 test scris** care contribuie cu 60% din nota finală
  - rezultatul obținut la **activitatea de laborator**

Ce este C++

- C++ este o extensie a lui C

  (Bjarne Stroustrup, AT&T Labs, 1984)
  - Programele C scrise bine sunt si programe C++
  - Programele care sunt si programe C si programe C++ au acelasi inteles(semantica); exceptiile sunt minore.
  - Există totuşi cod C care nu este C++:

```
void f(); void g(){f(2);}
struct S{int x, y;} f();
void g(struct S {int x, y;} y);
```

Ce este C++

- Este un C mai bun;
- Programatorii buni in C au un dezavantaj: au tendința sa scrie cod C++ in stilul C pierzând beneficiile aduse de C++
- Suportă abstractizarea datelor
- Suportă programare orientată obiect
- Suportă programare generică

Paradigme de programare

- C++ nu a fost proiectat să impuna un anume stil de programare, el suporta stilurile (paradigmele) tradiționale şi pe cele "avansate":
  - **Programare procedurală**: Decide ce proceduri sunt necesare, si alege cei mai buni algoritmi pentru ele
  - **Programare modulară**: Decide ce module sunt necesare, împarte programul asa fel incat datele sa fie "ascunse" în module
  - **Tipuri abstracte de date** (Tipuri definite de utilizator): Decide ce tipuri îți sunt de folos si construeşte câte un set complet de operații pentru acestea
  - **Programare orientată obiect** (POO): Decide ce **clase** sunt necesare, proiectează cate un set complet de operații pentru acestea si explicitează părțile comune ale lor prin relația de **moştenire** (derivare, generalizare), şi foloseşte **polimorfismul**
  - **Programare generică**: Decide algoritmii ce sunt necesari şi parametrizează-i pentru a funcționa pe o varietate de tipuri şi de structuri de date

CLASE SI OBIECTE – o primă viziune

- Un **obiect** este caracterizat de:
  - nume – un identificator
  - atribute – date de un anume tip
    - valorile atributelor la un moment dat definesc o ***stare*** a obiectului
  - metode (servicii, operații) – funcții ce accesează atributele obiectului modificând eventual starea acestuia; ele definesc ***comportarea obiectului***

CLASE SI OBIECTE – o primă viziune

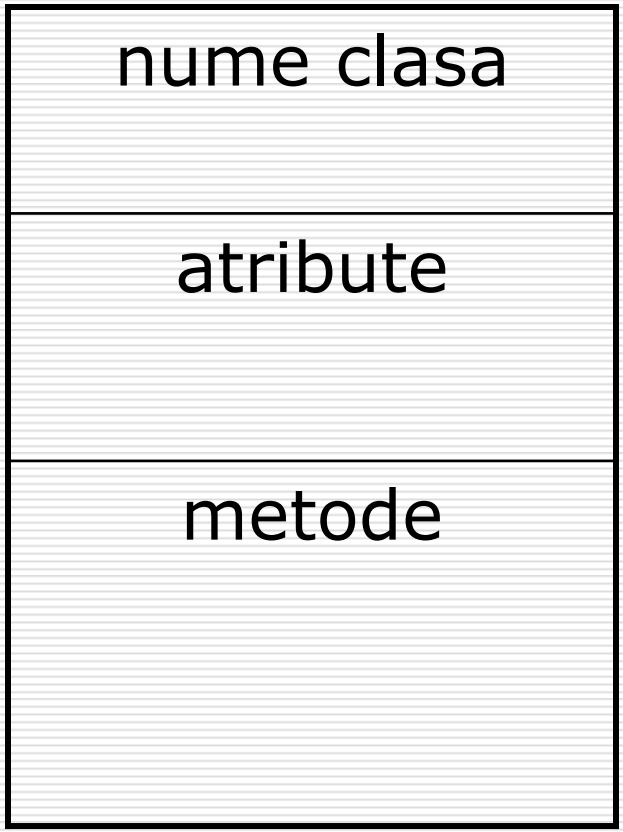
- O **clasă** descrie unul sau mai multe obiecte ce pot fi precizate printr-un set uniform de atribute(date) şi metode(funcționalitate).

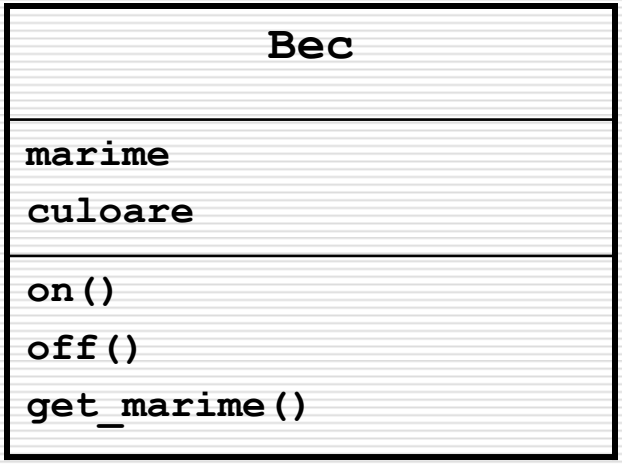
- Orice **obiect** are memoria sa proprie unde se păstrează valorile tuturor atributelor sale

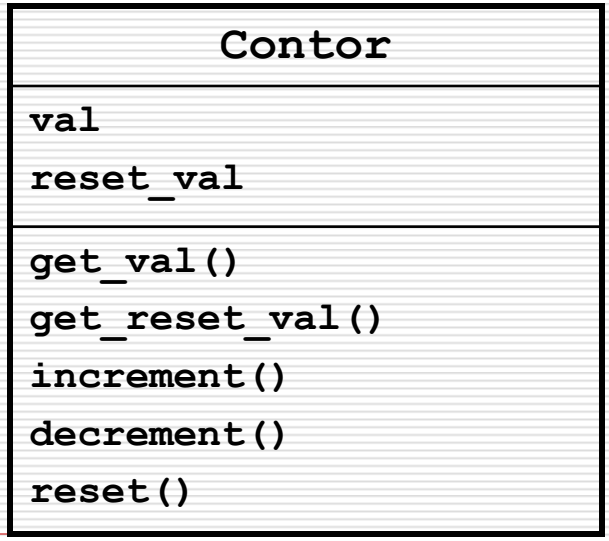
- Orice **obiect** are un tip; un obiect este o ***instanță*** a unei clase

- **Clasa** defineşte caracteristicile si comportarea obiectelor

CLASE SI OBIECTE –Reprezentare grafică







Clasa cont

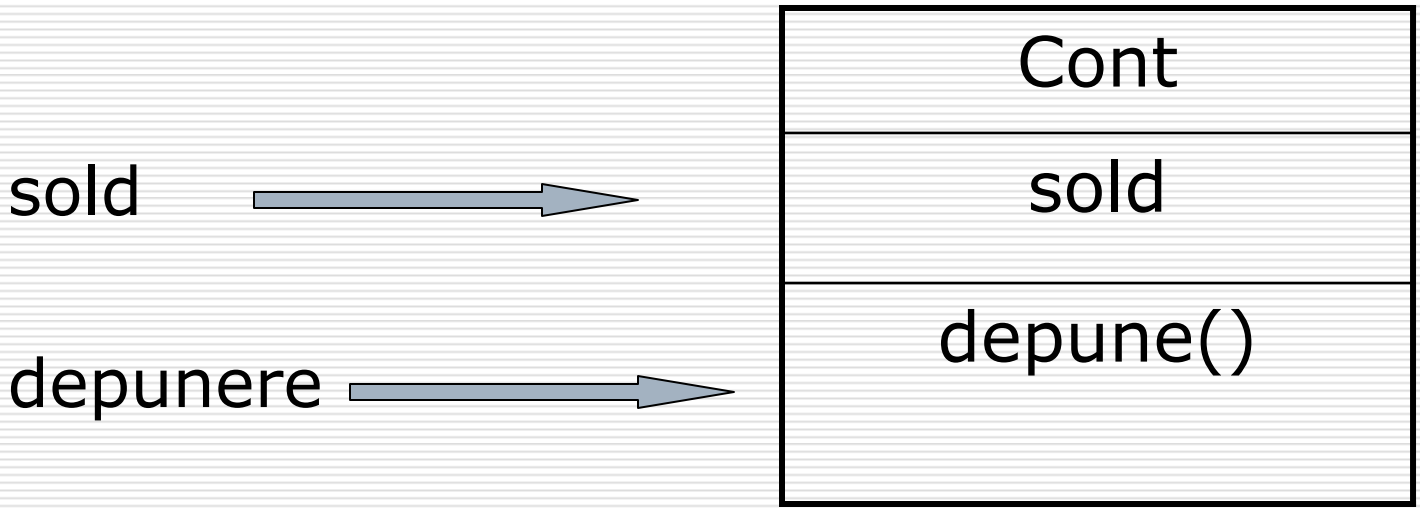
- Specificarea clasei
  - Un cont bancar:
    - are un titular, sold, o rată a dobânzii, numar de cont
    - se pot efectua operații de depunere, extragere, actualizare sold
- Extragerea atributelor:
  - titular, sold, rată a dobânzii, număr de cont
- Extragerea metodelor:
  - depunere, extragere, actualizare sold
- Completare specificație
  - actualizare sold ➔ data ultimei operații

Clase - proiectare

- Definiția unei clase presupune:
  - combinarea datelor cu operațiile ce se aplică acestora
  - ascunderea informației

Combinarea datelor cu operațiile

- datele si operațiile ce se aplică asupra lor sunt incluse în aceeaşi unitate sintactică; în acest mod datele şi procesele devin legate



ASCUNDERE şi ÎNCAPSULARE

- Ascunderea informației:
  - Modul de structurare a datelor nu este cunoscut: datele sunt declarate intr-o secțiune "privată" a clasei.
  - Accesul la date, pentru consultare, modificare, etc. se face numai prin intermediul funcțiilor clasei: acestea sunt declarate intr-o secțiune "publică" a clasei – **interfața clasei**.
  - Pot exista date "publice" precum si funcții "private": decizia este a programatorului.

- Încapsulare = Combinare + Ascundere
  - Datele si funcțiile (operațiile) care pot acționa asupra datelor sunt incluse în aceeaşi unitate sintactică
  - În C++ acest lucru se realizeaza prin tipul utilizator **`struct`** şi **`class`**

AVANTAJE

- Combinarea datelor:
  - defineşte clar ce structuri de date sunt manevrate şi care sunt operațiile legale asupra lor.
  - programul capătă modularitate.
  - scade riscul alterării datelor din exterior.
  - facilitează ascunderea informației.
- Ascunderea informației:
  - programe mai sigure şi mai fiabile
  - eliberează clasele utilizator de grija manevrării datelor
  - previne apariția erorilor

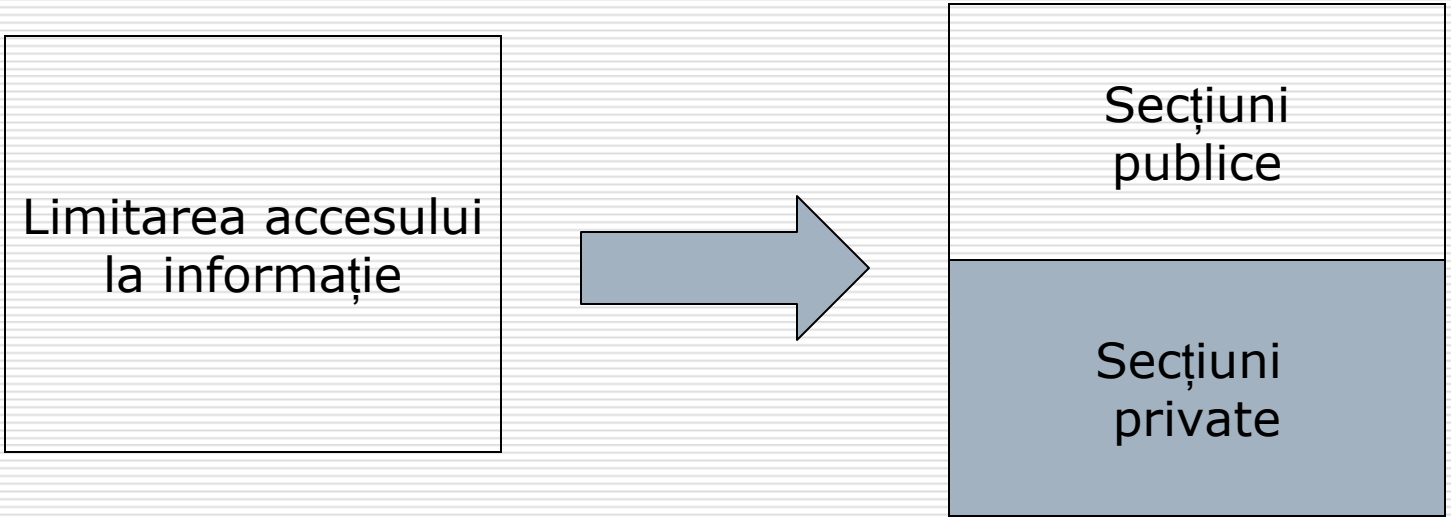
AVANTAJE

- Încapsulare:

  Combinare + Ascunderea informației =
  Protejarea datelor

  - previne apariția erorilor prin limitarea accesului la date
  - asigură portabilitatea programelor
  - facilitează utilizarea excepțiilor
  - interfața unei clase = operațiile cu care o clasă utilizator poate manevra datele

Structurarea nivelului de acces la informație



Clasa cont – Fişierul cont.h

```
class Cont
{
public:
        Cont();
        ~Cont();
        void depune(double);
        void extrage(double);
        void setDob(double);

private:
        char *titular;
        double sold;
        double rataDob;
        char *dataUltOp;
        void actualizeazaSold();
        void atribuieNrCont();
};
```

Clasa cont – Fişierul cont.cpp

```
Cont::Cont()
{
        titular = NULL;
        dataUltOp = NULL;
        sold = 0;
        rataDob = 0;
        atribuieNrCont();
}

Cont::~Cont()
{
        delete titular;
        delete dataUltOp;
}
```

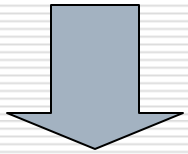
```
void Cont::depune(double suma)
{
        sold += suma;
}

void Cont::extrage(double suma)
{
        actualizeazaSold();
        if(suma > sold)
            throw "Sold insuf.";
        sold -= suma;
}
```

Clasa cont – Fişierul demo.cpp

```
void main()
{
Cont cont;
cont.depune(2000);
}
```

```
cont.sold += 2000;
```



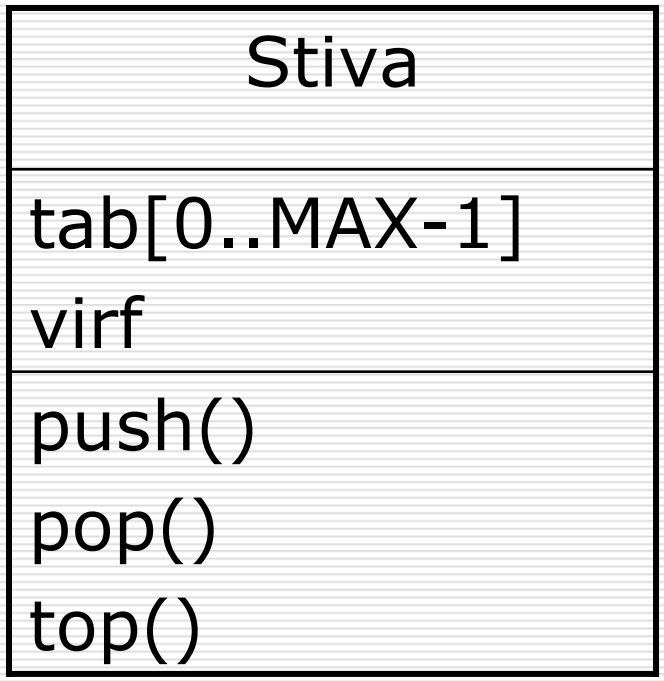
```
error C2248: 'sold' : cannot access private
   member declared in class 'Cont'
```

Clase si Obiecte - Tipuri abstracte de dată

- Clasa, Obiect – o alta viziune (definiție):
  - O **clasa** este o implementare a unui tip de dată abstract. Ea defineşte:
    - **atribute** care implementează structura de dată respectivă
    - **metode** care implementează operațiile tipului de dată abstract.
  - Un **obiect** este o instanță a unei clase. El este unic determinat de numele său şi defineşte o **stare** reprezentată de valorile atributelor sale la un moment dat

Stiva

- tipul de data abstract Stiva
  - entitati de tip dată: liste LIFO
  - operații:
    - push()
    - pop()
    - top()



Stiva

```
// Stiva.h
#define MAX_STIVA 10
class Stiva {
public:
  Stiva();
  ~Stiva(){};
  void push(char);
  void pop();
  char top();
  bool este_vida();
private:
  char tab[MAX_STIVA];
  int virf;
};
```

Stiva

```
// Stiva.cpp
Stiva::Stiva(){ virf = -1;}
void Stiva::push(char c){
  if (virf == MAX_STIVA-1)
          throw "Depasire superioara.";
  tab[++virf] = c;
}
void Stiva::pop(){
  if (virf < 0) throw "Depasire inferioara.";
  virf--;
}
char Stiva::top(){
  if (virf < 0) throw "Depasire inferioara.";
  return tab[virf];
}
```

Stiva: Stiva_demo.cpp

```
void main(void){
  Stiva S; char c='a';
  try {
        while (true){
              S.push(c++);
              cout << S.top() << ' ';
        }
  }
  catch (char *mes_err) {
    cout << '\n' << mes_err << endl;
  }
}
// a b c d e f g h i j
// Depasire superioara.
```

Utilizarea de clase

- există multe bibloteci de clase
  - STL
  - MFC
  - etc
- pentru utilizare, trebuie cunoscută doar interfața (elementele publice)
- nu interesează implementarea
- programul care utlizează clasa este independent de implementarea clasei

Utilizarea de clase

- Exemplu: string
  - my-string.h
  - STL

```
// my_string.h
class string
{
public:
   // Constructori/destructori
   string();
   string(char* new_string);
   string(const string& new_string);
   . . .
   ~string();
```

Utilizarea de clase

```
// Operatori
    friend string operator+ (const string& lhs,
                                        const  string& rhs);
    string& operator+= (const string& a_string);
    string& operator= (const string& a_string);
    friend bool operator< (const string& lhs,
                                        const string& rhs);
    friend bool operator> (const string& lhs,
                                        const string& rhs);
    friend bool operator== (const string& lhs,
                                        const string& rhs);
    friend ostream& operator<< (ostream& os,
                                const string& a_string);
private:
    char* char_arr;
    int string_length;
};
```

Utilizarea de clase

```
string s1(123); // creare sir dintr-un numar
string s2;  // creare sir vid
s2 = string("abc") + s1; // creare sir dintr-un sir C,
                         //operatorul de concatenare si
                         //operatorul de atribuire
s1 = s2 ;  // operatorul de atribuire
cout << s1 << endl << s1 + s2 << endl;
          // scrierea in fluxul standard de iesire
s1 = "armata";    // apelare constructor
                  // si apoi operator de atribuire
cout << s1 << endl;
s2 = "armata";
cout << ((s1 == s2)?1:0) << endl;
          // operatorul de testare a egalitatii
string s3(s2);    // creare prin copiere
                  //(apel constructor de copiere
```

…

Utilizarea de clase

- STL (Standard Template Library)

```
#include <string>
using namespace std;
```

Cont – versiunea 2

```
// cont.h          char* ==> string
#include <string>
using namespace std;
class Cont
{
public:
     Cont();
     Cont(string nume, double soldInit);
     ~Cont();
     void depune(double);
     void extrage(double);
     void setDob(double);
private:
     string titular;
     string nrCont;
     double sold;
     double rataDob;
     string dataUltOp;
     void actualizeazaSold();
     void atribuieNrCont();
};
```

Cont – versiunea 2

```
// cont.cpp      #include "cont.h"

Cont::Cont()
{
     dataUltOp = "";
     sold = 0;
     atribuieNrCont();
}


Cont::Cont(string nume, double soldInit) : titular(nume), sold(soldInit)
{
     // dataUltOp = azi();
}

void Cont::depune(double suma)
{
     sold += suma;
}
```

Cont – versiunea 2

```
// demo.cpp
#include "cont.h"

void main()
{
   Cont cont("Ionescu", 10000);
   cont.extrage(2000);
}
```