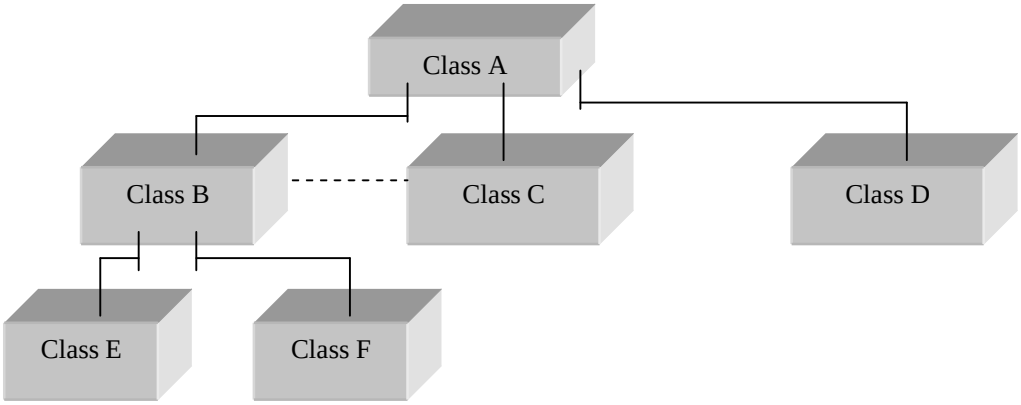
Mostenirea permite crearea ierarhiilor de clase, deci a ierarhiilor de concepte (Un concept poate fi implementat printr-o clasa. Clasele grupeaza obiecte de acelasi tip; reprezinta aceeasi idee, acelasi concept).
Aceasta proprietate se manifesta prin faptul ca din orice clasa putem deriva alte clase.

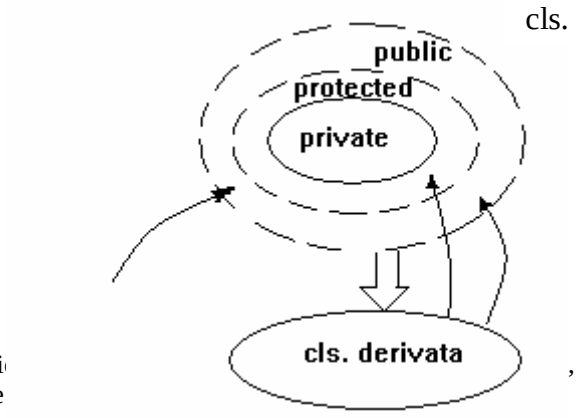


A – clasa de baza;
B, C, D – clase derivate din clasa de baza A
E, F – clase derivate din clasa de baza B
Informatia comuna apare in clasa de baza, iar informatia specifica - in clasa derivata. Clasa derivata reprezinta o specializare a clasei de baza. Orice clasa derivata mosteneste datele membru si metodele clasei de baza. Deci acestea *nu trebuie redeclarate* in clasa derivata.

Mostenirea poate fi: simpla (orice clasa are o singura superclasa) sau multipla (o clasa are mai multe superclase).

Mostenirea simpla

Declararea unei clase derivate
class <nume_clasa_derivata>: <modificator_de_acces> <nume_clasa_de_baza>
{ //corpul clasei derivate - elemente specifice clasei derivate
};



In functi , in declararea clasei derivate si in clasa derivata, lucrurile

Modificator acces in clasa de baza	**Modificator de acces (protectie) din declararea clasei derivate**	**Accesul in clasa derivata la elementul mostenit de la clasa de baza**
private	private, protected, public	inaccesibil
protected sau public	private	private
public	protected	protected
protected	protected	protected
protected	public	protected
public	public	public

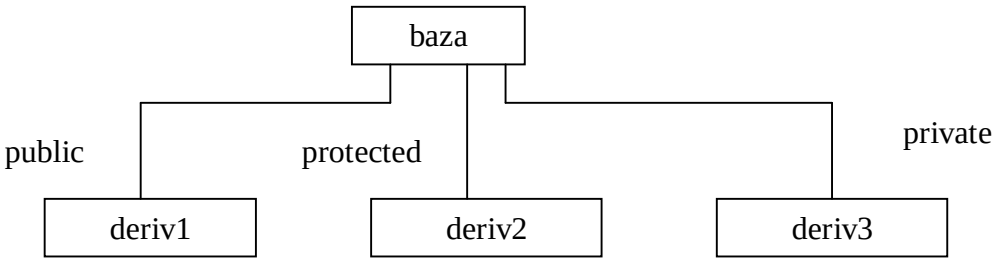
Constructorii claselor derivate

Constructorii si destructorii sunt functii membre care **nu** se mostenesc. La instantierea unui obiect din clasa derivata se apeleaza mai intai constructorul clasei de baza, apoi constructorul clasei derivate. La distrugerea obiectelor, se apeleaza intati destructorul clasei derivate, apoi destructorul clasei de baza.

Transmiterea argumentelor unei functii constructor din clasa de baza se face folosind o forma extinsa a declaratiei constructorului clasei derivate, care transmite argumentele unui sau mai multor constructori din clasa de baza.

In general, clasele utilizeaza constructori definiti de programator. In cazul in care acestia lipsesc, compilatorul genereaza automat un constructor implicit pentru clasa respectiva. Acelasi lucru se intampla si in cazul constructorilor de copiere.

La instantierea unui obiect din clasa derivata, o parte din valorile primite ca parametri folosesc la initializarea datelor membru ale claselor de baza, iar restul initializeaza datele membru specifice clasei derivate.
In exemplul urmator, este construita urmatoarea ierarhie de clase:




```
#include <iostream.h>
class baza
{
      int a;
protected:
      double w;
      void seteaza_a (int a1){a=a1;}
      void seteaza_w (int w1) {w=w1;}
public:
      int c;
      baza (int a1, double w1, int c1)
      {a=a1; w=w1; c=c1;cout<<"Constructor cls. baza\n";}
      ~baza()
      {cout<<"Destructor baza\n";}
      void arata()
      {cout<<a<<' '<<w<<' '<<c<<'\n';}
      double calcul()
      {return a+w+c;}
      friend ostream & operator<<(ostream &, const baza &);
};

class deriv1: public baza
{
      int b;
public:
deriv1 (int a1, double w1, int c1, int b1):baza(a1, w1, c1)
      {b=b1; cout<<"Constructor deriv1\n";}
      ~deriv1()
      {cout<<"Destructor deriv1\n";}
      double calcul()
      {return w+c+b;} // a nu poate fi folosit, fiind private
      // o alternativa:
      // double calcul(){return baza::calcul()+b;}
      friend ostream &operator<<(ostream &, const deriv1 &);
};

class deriv2: protected baza
{
      int b;
```

```
public:
	deriv2(int a1, double w1, int c1, int b1):baza(a1, w1, c1)
	{b=b1; cout<<"Constructor deriv2\n";}
	~deriv2()
	{cout<<"Destructor deriv2\n";}
	double calcul()
	{return w+c+b;}
	friend ostream &operator<<(ostream &, const deriv2 &);
};

class deriv3: private baza
{
	int b;
public:
	deriv3(int a1, double w1, int c1, int b1):baza(a1, w1, c1)
	{b=b1; cout<<"Constructor deriv3\n";}
	~deriv3()
	{cout<<"Destructor deriv3\n";}
	double calcul()
	{return w+c+b;}
	friend ostream &operator<<(ostream &, const deriv3 &);
};

ostream &operator<<(ostream &ies, const baza &b)
{ies<<b.a<<' '<<b.w<<' '<<b.c<<'\n'; return ies;}

ostream &operator<<(ostream &ies, const deriv1& d1)
{ies<<d1.w<<' '<<d1.c<<' '<<d1.b<<'\n'; // a private
return ies;}

ostream &operator<<(ostream &ies, const deriv2& d2)
{ies<<d2.w<<' '<<d2.c<<' '<<d2.b<<'\n'; // a private
return ies;}

ostream &operator<<(ostream &ies, const deriv3& d3)
{ies<<d3.w<<' '<<d3.c<<' '<<d3.b<<'\n'; // a private
return ies;}
void main()
{
baza x(1, 1.23, 2);                    // Constructor cls. baza
deriv1 y(2, 2.34, 3, 4);               // Constructor cls. baza
                                       // Constructor deriv1
deriv2 z(3, 3.45, 4, 5);               // Constructor cls. baza
                                       // Constructor deriv2
deriv3 v(4, 5.67, 6, 7);      // Constructor cls. baza
                                       // Constructor deriv3
cout<<"x="<<x<<endl<<"z="<<z<<endl<<"v="<<v<<endl;
        // x=1 1.23 2
        // z=3.45 4 5
        // v=5.67 6 7
cout<<"x.calcul()="<<x.calcul()<<endl;   // x.calcul()=4.23
cout<<"y.calcul()="<<y.calcul()<<endl;   // y.calcul()=9.34
cout<<"z.calcul()="<<z.calcul()<<endl;   // z.calcul()=12.45

cout<<"v.calcul()="<<v.calcul()<<endl;   // v.calcul()=18.67
cout<<"x.c="<<x.c<<endl;                       // x.c=2
cout<<"y.c="<<y.c<<endl;                       // y.c=3
/*      Destructor deriv3
        Destructor baza          ptr. v
Destructor deriv2
Destructor baza                  ptr. z
        Destructor deriv1
        Destructor baza          ptr. y
```

*Destructor baza ptr x */*
}

In clasa de baza membrul a este private, w este protected si c este public.
In clasa de baza, cat si in clasele derivate exista constructori care initializeaza datele membru.
Membrii private dintr-o clasa de baza pot fi folositi doar in cadrul acesteia (de metodele sale), nu si in clasele derivate.
Pentru clasa *deriv1*:

- Membrii privati din clasa baza sunt inaccesibili (a exista, dar este incapsulat)
- Pentru a putea fi folositi, se acceseaza metoda din clasa de baza in care apare a
- Daca in clasa derivata exista o metoda cu acelasi nume cu al unei metode din clasa de baza (redefinirea unei metode in clasa derivata), aceasta din urma se poate utiliza in clasa derivata folosind un apel de forma:
  baza::calcul( ) sau *y.baza::calcul( )*

Pentru clasa *deriv2*:

- Membrii publici din clasa de baza devin protejati in clasa deriv2
- Membrii protejati din clasa de baza devin protejati in clasa deriv2
- Daca in functia *main( )* am incerca folosirea :
  cout<<z.baza::calcul( ) , metoda calcul din z este inaccesibila, ea devenind protejata in clasa *deriv3*.

Pentru clasa *deriv3*:

- Membrii public sau protected din clasa de baza au devenit privati in clasa *deriv3*.
- Se pot folosi toti membrii clasei de baza, cu exceptia celor privati (a).
- In cazul constructorilor, se apeleaza constructorul din clasa de baza
- Un obiect y din clasa deriv2 va incorporeaza un obiect deja initializat cu ajutorul constructorului din clasa de baza

OBS:
Daca aveam: *deriv1(int a1, double b1, int c1, int b1){a=a1; b=b1; c=c1; d=d1;}*
nu era corect, deoarece clasa baza nu are constructori fara parametri, deci nu exista constructor implicit; data a este private in deriv1.
Apelarea constructorului se face apeland explicit constructorul din clasa de baza.
O clasa poate poate contine mai multe obiecte cu aceeasi structura (aceleasi date membru si metode), dar ele difera prin valorile luate.

EX:

```
#include "sir.h"
#include <conio.h>

class persoana
{ protected:
sir numele,prenumele;
char sexul;

public:
  persoana ()                          //constructor vid
  {numele="";prenumele="";sexul='m';}
  persoana(const sir&,const sir&,const char); //constructor
  persoana (const persoana&); //constr. copiere
  virtual ~persoana(); //destructor
  const sir& nume();
  const sir&prenume();
  char sex();
  virtual void afisare();
  friend  ostream & operator<<(ostream &, const persoana &);
  friend istream & operator>>(istream &,  persoana &);
};

class student:public persoana
{ protected:
  sir facultatea,specializarea;
  int anul,grupa;
```

```
 public:
  student();
  student(const sir&,const sir&,const char,const sir&,const sir&,const int,const int);
  student(const persoana&,const sir&,const sir&,const int,const int);
  student(const student&);
  virtual ~student();
  const sir& facult(){return facultatea;}
  const sir& spec(){return specializarea;}
  int an(){return anul;}
  int grup(){return grupa;}
  virtual void afisare();
  friend  ostream & operator<<(ostream &, const student &);
//  friend istream & operator>>(istream &,  student &);
};


class student_bursier:public student
{protected:
  char tipul_bursei;

 public:
  student_bursier(const student&,char);
  student_bursier(const student_bursier&);
  virtual ~student_bursier();
  char tip_bursa() {return tipul_bursei;}
  double valoare_bursa();
  virtual void afisare();
//  friend  ostream & operator<<(ostream &, const student_bursier &);
//  friend istream & operator>>(istream &,  student_bursier &);
};


persoana::persoana(const sir& nume,const sir& prenume,const char sex)
{numele=nume;prenumele=prenume;sexul=sex;
cout<<"Constr. PERSOANA\n";}


persoana::persoana(const persoana& pers)
{ numele=pers.numele;prenumele=pers.prenumele;sexul=pers.sexul;
cout<<"Constructor copiere PERSOANA\n";}

persoana::~persoana()
{cout<<"Destructor PERSOANA\n";}

const sir& persoana::nume()
        {return numele;}

const sir& persoana::prenume()
        {return prenumele;}

char persoana::sex()
        {return sexul;}

void persoana::afisare()
{cout<<"Afisare PERSOANA:\n";
cout<<numele<<"   "<<prenumele<<"   "<<sexul;}

ostream & operator<<(ostream &monitor, const persoana &p)
{monitor<<"\nNume pers:"<<p.numele<<"\nPrenume pers:"<<p.prenumele<<"\nSex
pers:"<<((p.sexul=='m')?"BARBATESC":"FEMEIESC");
return monitor;}

istream & operator>>(istream & tastat,  persoana &p)
```

```
{tastat>>p.numele>>p.prenumele>>p.sexul;
return tastat;}

// METODE CLS. STUDENT
student::student(const sir&nume,const sir&prenume,const char sex,const sir& facult,const sir& spec,const int
an,const int gr):persoana(nume,prenume,sex)
{numele=nume;prenumele=prenume;
sexul=sex;facultatea=facult; specializarea=spec; anul=an; grupa=gr; }

student::student()
{numele="";prenumele="";sexul=0;facultatea=0;specializarea="";anul=0; grupa=0;}

student::student(const persoana &pers,const sir& facult,const sir& spec,const int an,const int
gr):persoana(pers)
{ numele=pers.nume();prenumele=pers.prenume();
facultatea=facult; specializarea=spec;anul=an;grupa=gr; }

student::student(const student& stud):persoana(stud.numele,stud.prenumele,stud.sexul)
{ facultatea=stud.facultatea; specializarea=stud.specializarea; anul=stud.anul;
grupa=stud.grupa; }

student::~student()
{ cout<<"Destructor student!!\n"; }

void student::afisare()
{ cout<<numele<<"   "<<prenumele<<'\n';
cout<<"Sex:"<<sexul<<'\n';
cout<<"Facultatea: "<<facultatea<<" Specializare: "<<specializarea<<'\n';
cout<<"Anul: "<<anul<<" Grupa:"<<grupa<<'\n';
}

ostream & operator<<(ostream &monitor, const student &s)
{monitor<<"\nNume stud:"<<s.numele<<"\nPrenume stud:"<<s.prenumele<<"\nSex
stud:"<<((s.sexul=='m')?"BARBATESC":"FEMEIESC");
monitor<<"\nFacultate :"<<s.facultatea<<"\nSpecializare :"<<s.specializarea;
monitor<<"\nAnul :"<<s.anul<<"\nGrupa :"<<s.grupa;
return monitor;}

//  friend istream & operator>>(istream &,  student &);

//METODE CLS. STUDENT_BURSIER
/*  student_bursier(student&,char);
  student_bursier(const student_bursier&);*/

student_bursier::student_bursier(const student &stud,char tip_bursa):student(stud)
{tipul_bursei=tip_bursa;}

student_bursier::student_bursier(const student_bursier
&stud):student(stud.numele,stud.prenumele,stud.sexul,stud.facultatea,stud.specializarea,stud.anul,stud.grupa)
{tipul_bursei=stud.tipul_bursei;}

double student_bursier::valoare_bursa()
{ double val;
  switch (tipul_bursei)
  { case 'A':val=85000;
              break;
    case 'B':val=70000;
              break;
  }
 return val;
}

student_bursier::~student_bursier()
```

```
{cout<<"Desctructor cls. 3\n";}

void student_bursier::afisare()
{ student::afisare();
  cout<<"Tip bursa: "<<tipul_bursei<<" Valoare: "<<valoare_bursa()<<'\n';}

void main()
{clrscr();
persoana x("POP","ION",'m');
x.afisare();cout<<'\n'; getch();
persoana x1(x); cout<<x1<<'\n'; getch();
cout<<"Inf despre persoana:\n";
persoana x2; cin>>x2;
cout<"Inf introduse:\n"; cout<<x2; getch();
x1.afisare(); cout<<'\n';
student s(x, "N.I.E.", "EA", 1, 2311);
s.afisare();cout<<'\n'; getch();
student s1(s); cout<<s1<<'\n'; getch(); }
```