Reutilizarea codului. Moştenirea Claselor

Mihai Gabroveanu

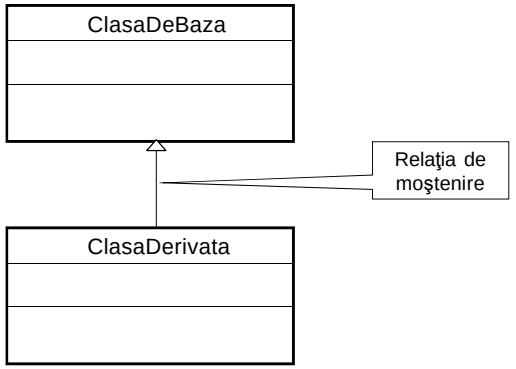
Ce este **moştenirea**?

- Mecanismul prin care o clasă preia structura (datele membre) şi comportamentul (metodele) unei alte clase la care adaugă elemente specifice.
- Stabileste o relatie de tipul "*este un/este o*" (**is-a**)
- **Clasă de baza** = clasa *de la care se preia* structura şi comportamentul
- **Clasa derivată** = clasa *care preia* structura şi comportamentul

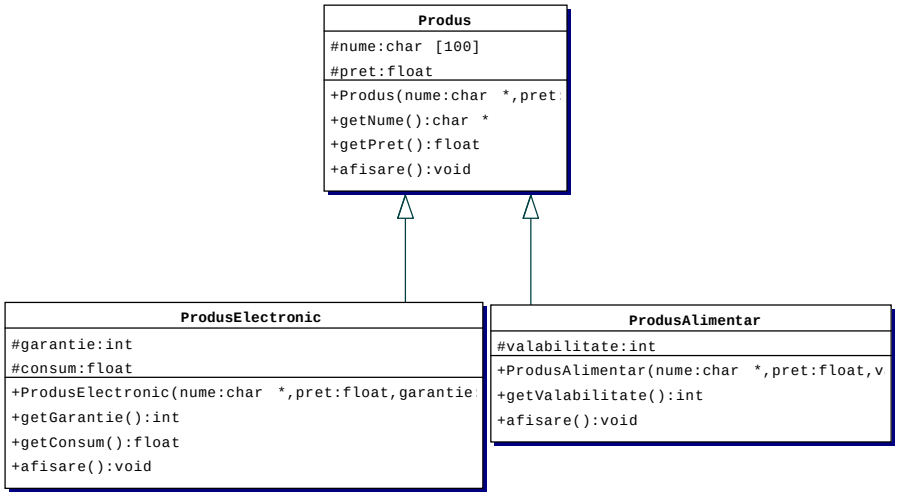
Avantaje

- posibilitatea reutilizării codului
- obţinerea extensiei unei clase fară a fi necesară o recompilare a clasei iniţiale;
- utilizarea **polimorfismului** în timpul executiei programului prin folosirea funcţiilor **virtuale**

Reprezentare UML



Exemplu



Ce se moşteneşte?

- În principiu, fiecare dată sau funcţie membru a clasei de bază se moşteneşte în clasa derivată:
  - diferă doar protecţia acestora
- Există şi excepţii:
  - Constructorii şi destructorii
  - Supraîncărcarea operatorului =

Aceştia nu se moştenesc, fiind metode specifice clasei

Sintaxa definirii unei clase derivate

- Sintaxa definirii unei clase derivate este următoarea:

```
class IdClasaDerivata: modif_acces1 IdClasaB1, ..., modif_accesN IdClasaBn{
    //date si metode specifice clasei derivate
};
```

unde:

- IdClasaDerivata este numele clasei derivate
- IdClasaB1, ..., dClasaBn sunt clasele de bază de la care se moştenesc datele şi metodele
- modif_acces1, ..., modif_accesN sunt modificatori de acces: *public, protected, private*

Accesul asupra membrilor moşteniţi

Protectia în clasa de bază	Modificator de acces utilizat în lista claselor de bază	Drept de acces în clasa derivată
public private protected	public public public	public inaccesibil protected
public private protected	private private private	private inaccesibil private

Exemplu

```
class Produs{
  protected:
     char nume[100];
     float pret;
  public:
     Produs(char *nume, float pret);
     char* getNume();
     float getPret();
     void afisare();
};
```

Clasa derivată

Clasa de bază

```
class ProdusAlimentar: public Produs{
     protected:
     int valabilitate;
     public:
     ProdusAlimentar(char *nume,
float pret, int valabilitate);
     int getValabilitate();
     void afisare();
};
```

modificator acces

Utilizarea datelor şi metodelor moştenite

```
void Produs::afisare(){
   cout<<"Nume:"<<nume<<endl;
   cout<<"Pret:"<<pret<<endl;
}

void ProdusAlimentar::afisare(){
  Produs::afisare();
cout<<"Valabilitate:"<<valabilitate;
cout<<endl;
}
```

```
void main(){
   ProdusAlimentar pa("Iaurt",15,30);
   pa.afisare();
   pa.Produs::afisare() ;
}
```

Output

```
Nume:Iaurt
Pret:15
Valabilitate:30

Nume:Iaurt
Pret:15
```


Exemplu

Clasa derivata

Clasa de baza

```
class ProdusElectronic: private Produs{
     protected:
     int garantie;
     float consum;
     public:
     ProdusElectronic(char *nume, float
pret, int garantie, float consum);
     int getGarantie();
     float getConsum();
     void afisare();
};
```

modificator acces

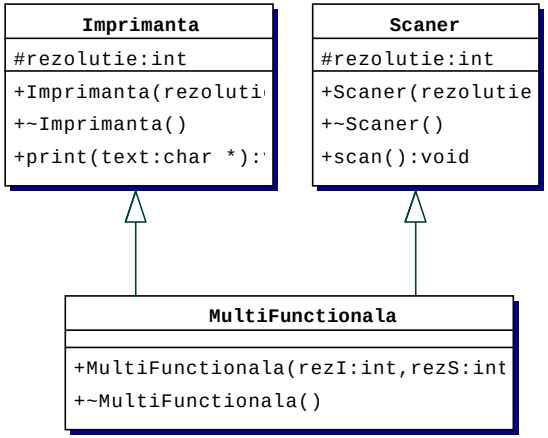
```
int main(){
  ProdusAlimentar pa("Iaurt",15,30);
  pa.afisare();
  cout<<"Pretul p.a"<<pa.getPret();
  ProdusElectronic pe("LCD",1000,24,12);
  pe.afisare();
  cout<<"Pretul p. e." <<pe.getPret();

}
```

inaccesibila

Tipuri de Moştenire

- **Moştenire simplă** – clasa derivată preia caracteristicile şi metodele unei singure clase de bază
- **Moştenire multiplă** - clasa derivată preia caracteristicile şi metodele de la mai multe clase de bază

Moştenire Multiplă. Exemplu



Constructori în procesul de moştenire

- Dacă o clasă D este derivată din clase (B1, ..., Bn) atunci constructorul clasei D va avea suficienţi parametri pentru a iniţializa datele membru ale claselor B1,..., Bn.
- La crearea unui obiect al clasei D se vor apela mai întâi constructorii claselor B1, ... ,Bn, în ordinea specificată în lista de moştenire, pentru a se iniţializa datele membre ale claselor de bază şi apoi se vor executa intrucţiunile constructorului clasei D.

Apelul constructorilor claselor de bază

- **Explicit –** Lista de initializare

```
class D: public B1, ..., public Bn{
    D(lista de parametri);
};

D::D(lista de parametri):B1(sub_lista_param1), . . ., Bn(sub_lista_param1){
  instructiuni
}
```

Apelul explicit

- **Implicit** – dacă constructorul clasei derivate nu apelează explicit constructorul uneia din clasele de bază atunci compilatorul va apela automat constructorul implicit al acelei clase de bază

Destructori în procesul de moştenire

- La distrugerea unui obiect al clasei derivate compilatorul va executa mai întâi destructorul clasei derivate şi apoi destructori claselor de bază în ordinea inversă apelului constructorilor

Exemplu

```
class Imprimanta{
    protected:
      int rezolutie;
    public:
      Imprimanta(int rezolutie=600){
        this->rezolutie = rezolutie;
        cout<<"Apel Constr.
Imprimanta\n";
      }
      ~Imprimanta(){
        cout<<"Apel Destr.
Imprimanta\n";
      }
      void print(char *text){
          cout<<"Print "<<text<<"\n";
      }
};
```

```
class Scaner{
  protected:
     int rezolutie;
  public:
     Scaner(int rezolutie=1200){
        this->rezolutie = rezolutie;
        cout<<"Apel Constr. Scaner\n";
     }
     ~Scaner(){
        cout<<"Apel Destr. Scaner\n";
     }
     void scan(char *numeFisier){
          cout<<"Scanez..."<<endl;
          cout<<"Salvez ";
          cout<<numeFisier<<endl;
     }
};
```


Exemplu

```
class MultiFunctionala: public Imprimanta, public
Scaner{
  public:
    MultiFunctionala(int rezI, int rezS):
Imprimanta(rezI), Scaner(rezS){
      cout<<"Apel Constr. MultiFunctionala\n";
    }
    ~MultiFunctionala(){
      cout<<"Apel Destr. MultiFunctionala\n";
    }
};
void main(){
    MultiFunctionala m(300,600);
    m.print("hello");
    m.scan("test.pdf");
}
```

```
OUTPUT

Apel Constr. Imprimanta
Apel Constr. Scaner
Apel Constr. MultiFunctionala
Print hello
Scanez...
Salvez test.pdf
Apel Destr. MultiFunctionala
Apel Destr. Scaner
Apel Destr. Imprimanta
```

Se afişează la distrugerea obiectului m

Constructorul de copiere în procesul de moştenire

- Dacă **clasa derivată nu are** definit un **constructor de copiere** atunci compilatorul generează unul automat care va realiza copierea datelor membre moştenite apelând la constructorii de copiere ai claselor de bază (definiţi de utilizator sau generaţi de compilator), iar datele specifice clasei derivate se vor copia bit cu bit.
- Dacă **clasa derivată are definit un constructor de copiere** atunci acesta se va realiza copierea tuturor datelor membre (moştenite sau proprii) - apelând eventual la constructorii claselor de bază

Supraîncărcarea operatorului de atribuire

- Dacă clasa derivată **nu are** supraîncărcat operatorul de atribuire atunci compilatorul va realiza copierea datelor membre moştenite apelând la supraîncărcările operatorului de atribuire definte în clasele de bază (dacă există), iar datele membre specifice vor fii copiate bit cu bit
- Dacă clasa derivată **are supraîncărcat** operatorul de atribuire atunci acesta va realiza copierea tuturor datelor membre.

Exemplu

```
class Persoana{
    protected:
        char *nume;
        int varsta;
    public:
        /** constructor cu parametri*/
        Persoana(char *nume, int varsta=0);
        /** Constructor de copiere*/
        Persoana(const Persoana &p);
        ~Persoana();
        void afisare();
        Persoana& operator = (Persoana &p);
};
Persoana::Persoana(char *nume, int varsta){
    this -> nume = new char[strlen(nume)+1];
    strcpy(this -> nume, nume);
    this -> varsta = varsta;
    cout<<"Apel constr. Persoana\n";
}
Persoana::Persoana(const Persoana &p){
    nume = new char[strlen(p.nume)+1];
    strcpy(nume, p.nume);
    varsta = p.varsta;
    cout<<"Apel Constr. de copiere Persoana\n";
}
```

```
Persoana::~Persoana(){
    delete []nume;
    cout<<"Apel destructor Persoana\n";
}


void Persoana::afisare(){
    cout<<"Nume:" << nume <<endl;
    cout<<"Varsta:" << varsta << endl;
}

Persoana& Persoana::operator = (Persoana &p){
    cout<<"Apel atribuire Persoana\n";
    if (this != &p){
       delete []nume;
       nume = new char[strlen(p.nume)+1];
       strcpy(nume, p.nume);
       varsta = p.varsta;
    }
    return *this;
}
```


Exemplu (cont.)

```
class Student: public Persoana {
  protected:
    char *facultate;
  public:
    Student(char *nume, int varsta, char *
facultate);
    Student(const Student &s);
    ~Student();
    void afisare();
    Student& operator = (Student &p);
};
Student::Student(char *nume, int varsta, char *
facultate):Persoana(nume, varsta){
    this -> facultate = new
char[strlen(facultate)+1];
    strcpy(this -> facultate, facultate);
    cout<<"Apel constr. Student\n";
}
Student::Student(const Student &s):
                  Persoana(s.nume, s.varsta){
    facultate = new char[strlen(s.facultate)+1];
    strcpy(facultate, s.facultate);
    cout<<"Apel Constr. de copiere Student\n";
}
```

```
Student::~Student(){
    delete []facultate;
    cout<<"Apel destructor Student\n";
}
void Student::afisare(){
    Persoana::afisare();
    cout<<"Fac:"<<facultate<<endl;
}

Student& Student::operator = (Student &s){
    cout<<"Apel atribuire Student\n";
    if (this != &s){
       Persoana::operator=(s);
       delete []facultate;
       facultate = new
char[strlen(s.facultate)+1];
       strcpy(facultate, s.facultate);
    }
    return *this;
}
```

Apelul metodei moştenite din clasa Persoana

Exemplu (cont.)

```
int main(){
   Student s1("Mihai",19, "Informatica");
   s1.afisare();
   Student s2=s1;
   s2.afisare();
   Student s3("Maria",19, "Informatica");
   s1 = s3;
   s1.afisare();
   cout<<"Sfarsit program"<<endl;
}
```

Apel supraîncărcării operatorului de atribuire

```
Apel constr. Persoana
Apel constr. Student
Nume:Mihai
Varsta:19
Fac:Informatica
Apel constr. Persoana
Apel Constr. de copiere Student
Nume:Mihai
Varsta:19
Fac:Informatica
Apel constr. Persoana
Apel constr. Student
Apel atribuire Student
Apel atribuire Persoana
Nume:Maria
Varsta:19
Fac:Informatica
Sfarsit program
Apel destructor Student
Apel destructor Persoana
Apel destructor Student
Apel destructor Persoana
Apel destructor Student
Apel destructor Persoana
```


Temă

- Implementaţi clasa Angajat ce derivă din clasa Persoana.
- Implementaţi o ierarhie de clase ce reprezinta diferite figuri geometrice din plan.
- Implementati ierarhia de clase Telefon, TelefonFix, TelefonMobil.