

Moștenire (Inheritance) în C++

Anul Universitar 2023 – 2024, Semestrul II

Programare Orientată pe Obiecte

LABORATOR 5, GRUPA 133

Cuprins

I.	Introducere în Moștenire (Inheritance) în C++	2
1.	Ce este Moștenirea?	2
2.	Clasa de Bază și Clasa Derivată.....	2
II.	Importanța moștenirii în OOP și în producție	2
III.	Proprietățile (atributele și metodele) <i>public</i>, <i>protected</i>, <i>private</i> la moștenire	3
IV.	Specificatori de acces pentru moștenire: <i>public</i>, <i>protected</i> sau <i>private</i>	3
V.	Ce se moștenește din clasa de bază si ce nu se moșteneste	4
VI.	Ordinea apelării constructorilor și a destructorilor în cadrul moștenirii.....	5
1.	Ordinea apelării constructorilor.....	5
2.	Ordinea apelării destructorilor.....	6
VII.	De reținut din laboratorul curent	7
VIII.	Exercițiu exemplu moștenire – clasa <i>Vehicul</i> și clase derivate (<i>Masina</i>, <i>Autobuz</i>, etc).....	7

Autor: Wagner Ștefan Daniel

I. Introducere în Moștenire (Inheritance) în C++

Observație: pentru a evita redundanța, pe parcursul laboratorului vom numi atributele + metodele unei clase proprietăți membre (pe scurt: proprietăți) prin abuz de notație. A nu se confunda cu proprietățile din C#.

1. Ce este Moștenirea?

Moștenirea este unul dintre principiile fundamentale ale programării orientate pe obiecte (OOP), care permite unei clase să preia ("inherit") proprietăți (atribute și metode) de la altă clasă.

2. Clasa de Bază și Clasa Derivată

- **Clasa de Bază (Superclass/Parent Class/Base Class)**

Este clasa ale cărei proprietăți sunt moștenite de alte clase. Clasa de bază definește atributele și metodele comune care vor fi partajate sau extinse de clasele derivate.

- **Clasa Derivată (Subclass/Child Class/Derived Class)**

Este clasa care moștenește sau derivă dintr-o clasă de bază. Clasa derivată poate suprascrie metodele moștenite de la clasa de bază pentru a oferi o implementare specifică și poate introduce și proprietăți noi.

II. Importanța moștenirii în OOP și în producție

Prin utilizarea moștenirii, putem crea noi clase care sunt construite pe baza claselor existente, extinzându-le funcționalitatea și reutilizând codul existent.

Acest mecanism care sprijină realizarea de ierarhii de clase și promovează reutilizarea și extensibilitatea codului este esențial în proiectele mari din producție/open source. Evităm "copy/paste", și modificăm într-un singur loc când apare o schimbare.

De asemenea, indiferent câte clase noi moștenesc din clasa de bază, toate vor avea acces la proprietățile comune expuse. Astfel obținem posibil chiar și mii de linii de cod reutilizate de mai multe clase.

III. Proprietățile (atributele și metodele) *public*, *protected*, *private* la moștenire

Atenție: deși am vorbit despre proprietăți (atribute + metode) în întregul capitol, nu vrem să declarăm niciodată atributele *public* (nici la acest curs, nici în general), după cum v-ați obișnuit deja.

- Proprietățile declarate *public* sunt accesibile pentru clasele derivate, însă, în funcție de tipul moștenirii (*public*, *private* sau *protected*), se schimbă specificatorii de acces în clasele derivate.
- Proprietățile declarate *protected* în clasa de bază sunt accesibile din clasa în care sunt definite și din clasele derivate din aceasta. Acest nivel de acces este intermediar între *private* și *public*. Prin utilizarea proprietăților *protected*, putem asigura că datele sunt protejate de accesul direct din exteriorul ierarhiei de clase, la fel precum variabilele *private*, păstrând totodată flexibilitatea de a fi utilizate în clasele derivate, ca și cum ar fi *public*.
- Proprietățile *private* sunt accesibile doar în clasa în care sunt definite, nefiind accesibile în clasele derivate, indiferent de tipul de nivel de acces specificat la moștenire.

IV. Specificatori de acces pentru moștenire: *public*, *protected* sau *private*

Observație importantă: indiferent de specificatorul de acces folosit la moștenire, proprietățile *private* nu sunt accesibile/moștenite în clasa/clasele derivate.

- Moștenirea *public* – toate proprietățile își păstrează specificatorii de acces.
- Moștenirea *protected* – proprietățile *public* și *protected* ale clasei de bază devin proprietăți *protected* în clasa sau clasele derivate.

Drept rezultat, proprietățile din clasa de bază vor fi accesibile în clasa derivată și în orice clase care derivă direct din bază.

Însă, cel mai important, nu vor fi accesibile din exteriorul ierarhiei de clase, similar proprietăților *private*, astfel păstrăm **încapsularea datelor și pentru moștenire**.

- Moștenirea *private* – proprietățile *public* și *protected* ale clasei de bază devin proprietăți *private* în clasa sau clasele derivate.

Drept rezultat, proprietățile din clasa de bază vor fi accesibile în clasa derivată și în orice clase care derivă direct din bază.

Similar moștenirii *protected*, proprietățile moștenite nu vor fi accesibile din exteriorul ierarhiei de clase.

Diferit față de moștenirea *protected*, dacă avem *class B : private class A* și *class C : public class B*, atunci *class C* nu va mai avea acces direct la proprietățile *public* și *protected* din *class A*, deoarece ele au devenit *private* în urma moștenirii inițiale, și sunt inaccesibile în urma celei de a doua moșteniri.

Tot ce este descris în cuvinte mai sus în cadrul capitolului curent, vedeți ilustrat în tabelul de mai jos (sursă imagine originală [aici](#)):

Base class member access specifier	Type of Inheritance		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

V. Ce se moștenește din clasa de bază și ce nu se moștenește

Atunci când o clasă derivată moștenește de la o clasă de bază, ea preia majoritatea proprietăților clasei de bază, dar există excepții și reguli specifice care determină ce anume este moștenit și ce nu.

1. Se moștenește:

- Proprietățile **public** și **protected**
- Constructorii, destructorii, și supraîncărcările operatorilor: Deși aceștia nu sunt "moșteniți" în sensul tradițional, clasele derivate pot apela constructorii și destructorul clasei de bază. Supraîncărcările operatorilor trebuie redefinite dacă comportamentul dorit diferă de cel al clasei de bază.

2. Nu se moștenește:

- Constructorul de copiere, operatorul de atribuire (**operator=**), și constructorul de mutare (vom vedea în Laboratorul 7 rolul și implementarea lui):

Aceste funcții speciale sunt generate automat de compilator pentru fiecare clasă și nu sunt moștenite direct. Dacă este necesar, trebuie redefinite explicit în clasa derivată pentru a gestiona corect resursele moștenite.

- Proprietățile **private**: Atributele și metodele marcate ca **private** în clasa de bază nu sunt accesibile direct din clasa derivată.

VI. Ordinea apelării constructorilor și a destructorilor în cadrul moștenirii

Ordinea în care constructorii și destructorii sunt apelați în ierarhia de moștenire este esențială pentru înțelegerea și utilizarea corectă a OOP în C++, dar și alte limbaje OOP fără **garbage collector**.

1. Ordinea apelării constructorilor

Atunci când se creează un obiect al unei clase derivate, constructorul clasei de bază este apelat primul, urmat de constructorii tuturor claselor intermediare în ierarhia de moștenire, în ordinea de la bază către derivat. Acest lucru asigură că toate proprietățile moștenite sunt inițializate corespunzător înainte de inițializarea proprietăților specifice ale clasei derivate.

Exemplu: ex1_constructori.cpp, inclus în îndrumar aici dar și pe GitHub în laboratorul curent:

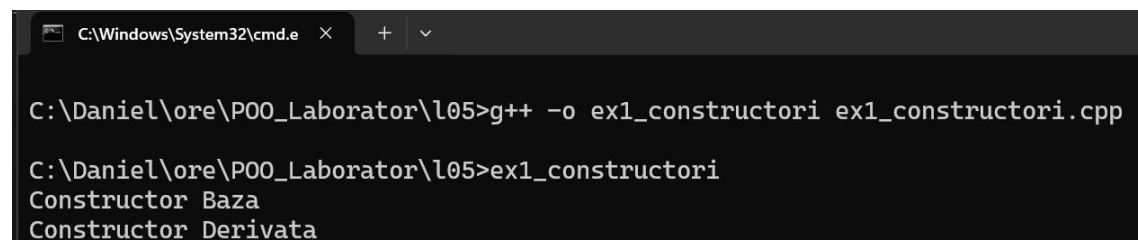
```
#include <iostream>
class Baza {
public:
    Baza() { std::cout << "Constructor Baza\n"; }
};

class Derivata : public Baza {
public:
    Derivata() { std::cout << "Constructor Derivata\n"; }
};

// La crearea unui obiect Derivata, iesirea va fi:
// Constructor Baza
// Constructor Derivata

int main()
{
    Derivata derivata;
    return 0;
}
```

Output-ul programului ex1_constructori.cpp:



```
C:\Windows\System32\cmd.e  ×  +  ▾

C:\Daniel\ore\P00_Laborator\l05>g++ -o ex1_constructori ex1_constructori.cpp

C:\Daniel\ore\P00_Laborator\l05>ex1_constructori
Constructor Baza
Constructor Derivata
```

2. Ordinea apelării destructorilor

Destructorii sunt apelați în ordinea inversă față de constructori, începând cu clasa derivată și continuând cu clasele de bază, în ordine inversă moștenirii.

Acest lucru asigură că resursele alocate de clasa derivată sunt eliberate înainte de eliberarea resurselor clasei de bază, prevenind astfel "scurgerile de memorie" (memory leaks) și alte probleme legate de gestionarea resurselor.

Exemplu: ex2_destructori.cpp, inclus în indrumar aici dar și pe GitHub în laboratorul curent:

```
#include <iostream>

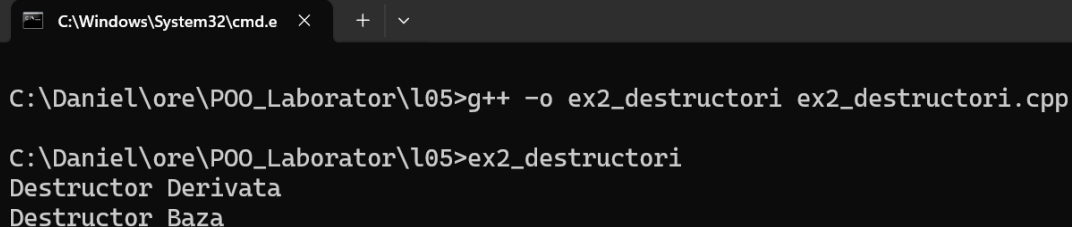
class Baza {
public:
    // avem constructorul implicit
    ~Baza() { std::cout << "Destructor Baza\n"; }
};

class Derivata : public Baza {
public:
    // constructor implicit idem Baza
    ~Derivata() { std::cout << "Destructor Derivata\n"; }
};

// La crearea unui obiect Derivata, output-ul va fi:
// Destructor Derivata
// Destructor Baza

int main()
{
    Derivata derivata;
    // destructor apelat automat cand variabila locala iese din scop
    return 0;
}
```

Output-ul programului ex2_destructori.cpp:



```
C:\Windows\System32\cmd.e  X  +  v

C:\Daniel\ore\P00_Laborator\l05>g++ -o ex2_destructori ex2_destructori.cpp

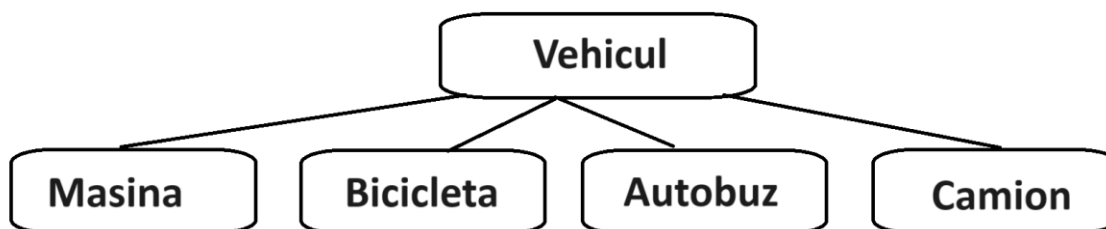
C:\Daniel\ore\P00_Laborator\l05>ex2_destructori
Destructor Derivata
Destructor Baza
```

VII. De reținut din laboratorul curent

1. Noțiunea de clasă de bază / clasă derivată
2. Importanța accesului protected pentru attribute la moștenire.
3. Polimorfism: Prin moștenire, putem folosi polimorfismul pentru a ține obiecte (sau colecții STL de obiecte) din clase derivate ca și cum ar fi obiecte din clasa de bază, permițând flexibilitate în apelurile de funcții și gestionarea/modificarea datelor.
4. Capitolele III - VI

VIII. Exercițiu exemplu moștenire – clasa *Vehicul* și clase derivate (*Masina*, *Autobuz*, etc)

Diagramă clase:



În cadrul timpului rămas din laboratorul curent, dorim să facem un program care ne reține diferite tipuri de vehicule.

Toate vehiculele au anumite attribute comune, cum ar fi numele mărcii și viteza maximă, posibil și metode comune pentru clase mai complexe.

Aceste attribute comune pot fi definite într-o clasă de bază, *Vehicul*, din care pot fi derivate alte clase specifice, cum ar fi *Masina*, *Bicicleta*, *Autobuz*, *Camion*, etc.

Deoarece am definit variabilele membre comune în clasa *Vehicul*, și eventual metode membre comune, ne putem folosi prin intermediul moștenirii pentru clasele derivate exemplificate mai sus de proprietățile membre comune.