Laboratorul 3

Moștenirea în limbajul C++

Ce ne propunem astăzi

Azi vom învăța să folosim una din cele mai importante posibilități pe care programarea orientată pe obiecte ni le oferă: moștenirea. Aceasta este o formă de reutilizare a codului, prin care se extind funcționalitățile unei clase. Avantajul folosirii acestui concept este că se economisește timp care este foarte binevenit în dezvoltarea software. Se preferă reutilizarea codului care a fost deja testat, deoarece în acest fel se vor reduce problemele care pot apărea după ce programul va deveni funcțional.

Pentru a folosi conceptul de moștenire într-un mod corect se va avea în vedere faptul că există trei tipuri de moșteniri: public, protected şi private. În tabelul 1 sunt sintetizate drepturile de acces a membrilor unei clase derivate în funcţie de drepturile de accesare a membrilor clasei de bază şi valoarea lui ***modificator acces***.

Atributul din clasa de baza	Modificator de acces	Accesul mostenit de clasa derivata	Accesul din exterior
private	private	inaccesibil	inaccesibil
protected	private	private	inaccesibil
public	private	private	inaccesibil
private	public	inaccesibil	inaccesibil
protected	public	protected	inaccesibil
public	public	public	accesibil

Tabel 1. Drepturi de acces

Moștenirea multiplă înseamnă posibilitatea de a defini o clasă derivată simultan din mai multe clase (numite clase de baza).

Din punct de vedere sintactic moștenirea multiplă se prezintă astfel:

```
class baza {};
class baza1 : public baza {};
class baza2 : public baza {};
class derivat : public baza1, public baza2 {};
```

Constructorii claselor derivate

O instanţiere a unei clase derivate conţine toţi membrii clasei de bază şi toţi aceştia trebuie iniţializaţi. Constructorul clasei de bază este apelat de constructorul clasei derivate. Compilatorul va executa prima oară constructorul clasei de bază.

Sfaturi utile

O clasă derivată:

- *nu poate accesa membrii private ai clasei sale de bază, dacă s-ar permite acest lucru atunci s-ar încălca conceptul încapsulării.*
- *este capabil să acceseze membrii public și protected ai clasei de bază.*
- *poate accesa membrii private ai clasei sale de bază doar indirect, dacă aceasta a implementat funcţii de acces public sau protected pentru ei.*

*Funcţiile membre ale clasei derivate **nu** au acces la membrii privaţi ai clasei de bază.*

Conversii de tip

Limbajul **C++** permite conversia implicită a unei instanţieri a clasei derivate într-o instanţiere a clasei de bază. De exemplu:

Muncitor mn;
Vânzator vanz("Popescu Ion");
mn = vanz; // conversie derivat => bază

De asemenea, se poate converti un pointer la un obiect din clasa derivată într-un pointer la un obiect din clasa de bază.

Conversia **derivat*** => **baza*** se poate face:

- **implicit** - dacă pointer-ul *derivat* moşteneşte pointer-ul *bază* prin specificatorul *public*;
- **explicit**: dacă pointer-ul *derivat* moşteneşte pointer-ul *bază* prin specificatorul *private;*

Conversia **baza*** -> **derivat*** nu poate fi făcută decât **explicit**, folosind operatorul *cast*.

Conversia inversă trebuie făcută explicit:

Muncitor *munc = &munc_1;
Vanzator *vanz;
vanz = (Vanzator *)munc;

Această conversie nu este recomandată, deoarece nu se poate şti cu certitudine către ce tip de obiect pointează pointer-ul la clasa de bază.

Partea practică. Mod de lucru

Iată pașii care trebuie urmați în cadrul acestui laborator, pentru dezvoltarea cu succes a programelor care folosesc moștenirea:

1. În exemplul următor este prezentată o schiță a unei moșteniri, unde clasa Derivata va specializa clasa de bază.

```
class Baza
{
public:
	void afisare()
	{
		cout << "Suntem in clasa de baza" << endl;
	}
};
class Derivata :public Baza
{
public:
```

```
	void afisare()
	{
		cout << "Suntem in clasa derivata" << endl;
	}
};
void main()
{
	Derivata obiect;
	obiect.afisare(); //se va utiliza functia din clasa Derivata
	obiect.Baza::afisare(); //se va utiliza functia din clasa Baza
}
```

2. ***După exemplul de mai sus să se realizeze o clasă de bază denumirtă Mamifer și două clase derivate denumite: Cangur și Urs. Programul va trebui să permită adăugarea unor înregistrări și afișarea acestora.***

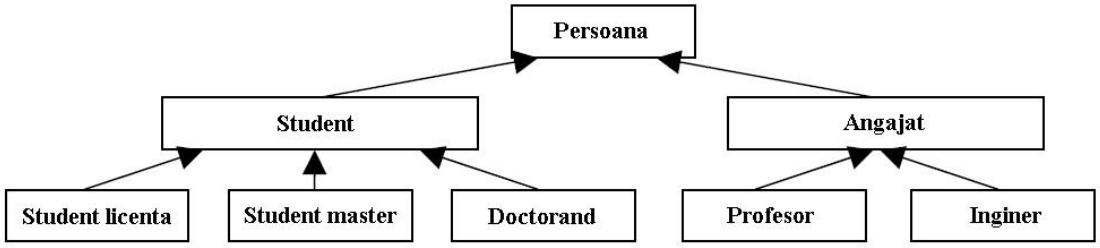
3. Vom studia exemplele următoare și vom rezolva cerințele propuse:

- În Anexa 3 exemplul 1 *avem o clasă de bază Carte și o clasă FisaBiblioteca care este drivată din clasa de bază.* După parcurgerea acestui exemplu vom ști să realizăm un program simplu folosind moștenirea claselor.
- În Anexa 3 exemplu 2 *avem clasa Forma care este clasa de bază şi clasa de Dreptunghi care este clasa derivată (moşteneşte clasa Forma).* În clasa de bază se setează valorile pentru înălţime şi pentru lăţime. Apoi în clasa derivată se calculează aria. În funcția *main* avem obiectul *Rect* de tipul *Dreptunghi*. Prin intermediul acestui obiect putem accesa atât funcţiile din clasa *Forma (setLatime* şi *setInaltimea)* cât şi funcţiile din clasa *Dreptunghi.* Funcţiile din clasa *Forma* pot fi accesate deorece clasa *Dreptunghi* moşteneşte clasa *Forma*. După parcurgerea acestui exemplu vom ști șă realizăm un program care apelează metode prin intermediul clasei derivate din clasa de bază.
- În Anexa 3 exemplul 3, în clasa Dreptunghi, constructorul clasei apelează constructorul clasei Forma și transmitere ca parametrii valorile w, l. În clasa Paralelipiped, constructorul clasei apelează constructorul clasei Dreptunghi și transmitere ca parametrii valorile w, l. De asemenea se extinde constructorul clasei Dreptunghi prin adăugarea liniei inaltimea=h;. În acest exemplu variabilele *înălțime, lungime și lățime* sunt *inițializate cu ajutorul constructorului.*
- În Anexa 3 exemplu 4 *este arătat modul în care o clasa poate moşteni mai multe clase.* În cazul nostru clasa *Paralelipiped* moștenește clasele *Forma* și *Atreiadimensiune*. De asemenea prin intermediul constructorilor se setează valorile necesare (înălţime, lungime şi lăţime).

  Constructorul clasei *Paralelipiped* apelează:

  - constructorul clasei *Forma* și transmitere ca parametrii valorile w, l (lungime şi lăţime)
  - constructorul clasei *3D* și transmitere ca parametrii valorile k (înălţime)
- În Anexa 3 exemplu 5 *este prezentat conceptul de moştenire multiplă în versiuni mai vechi de Visual Studio.NET și versiunea 2015*.
- În Anexa 3 exemplu 6 *este prezentat o clasă de bază Persoana și o clasă derivată Angajat*. Programul conține o matrice de obiecte asupra căreia se pot face diferite operații de adăugare, afișare, căutare și ștergere.

4. ***După exemplul din anexă să se realizeze o aplicație care să conțină următoarea structură de clase:***



Folosiți variabile corespunzătoare pentru fiecare tip de clasă. Realizați adăugarea, afișarea, căutarea și ștergerea informațiilor.

5. ***Extindeți aplicația de la exemplul 1 din Anexa 3 astfel încât să puteți adăuga cărți, să împrumutați, să restituiți, să ștergeți cărți, să căutați cărți.***

Cu ce ne-am ales?

Prin programele prezentate ca exemple și problemele propuse de la partea practică a laboratorului am reușit să ne familiarizăm cu conceptul de moștenire, care este una din cele mai importante mecanisme din programarea orientată pe obiecte.

Orice program orientat pe obiecte folosește moștenirea. De fapt, de abia acum începem cu adevărat să scriem programe orientate pe obiecte !

Anexa 3

Exemplu 1

```
#include<iostream>
#include<conio.h>
using namespace std;
// clasa de bază
class Carte
{
	char titlu[64];
public:
	// constructorul clasei de bază
	Carte(char *titlu)
	{
		strcpy(Carte::titlu, titlu);
	}
	void afis_carte()
	{
		cout << "titlul cartii este " << titlu << endl;
	}
protected:
	float cost;
	void afis_cost()
	{
		cout << "cartea costa " << cost << endl;
	}
};
// clasa derivata
class FisaBiblioteca :public Carte
{
	char autor[64], editura[64];
public:
	// constructorul clasei derivate
	FisaBiblioteca(char *titlu, char *autor, char *editura) :Carte(titlu)
	{
		// initializarea variabilelor
		strcpy(FisaBiblioteca::autor, autor);
		strcpy(FisaBiblioteca::editura, editura);
		cost = 49.98;
	}
	void afis_biblio()
	{
		afis_carte();
		afis_cost();
		cout << "autor: " << autor << " editura " << editura << endl;
	}
};
void main()
{
	// crearea unui obiect
	FisaBiblioteca fisa("Programare Orientata pe Obiecte", "Vasile Stoicu-Tivadar", "Politehnica");
	// afișarea obiectului
	fisa.afis_biblio();
	getch();
}
```

Exemplu 2

```
#include <iostream>
using namespace std;
// clasa de baza
class Forma
{
public:
	// funcție setare latime
	void setLatime(int w)
	{
		latime = w;
	}
	// functie setare inaltime
	void setInaltime(int h)
	{
		inaltime = h;
	}
protected:
	int latime;
	int inaltime;
};
// clasa derivata
class Dreptunghi : public Forma
{
public:
	// funcție de calculare a ariei
	int getAria()
	{
		return (latime * inaltime);
	}
};
int main(void)
{
	// crearea unui obiect
	Dreptunghi Rect;
	Rect.setLatime(5);
	Rect.setInaltime(7);
	// Afisarea ariei obiectului.
	cout << "Aria totala: " << Rect.getAria() << endl;
	return 0;
}
```

Exemplu 3

```
#include <iostream>
using namespace std;
// clasa de baza
class Forma
{
protected:
	int latime;
	int lungime;
public:
	// constructorul clasei de bază
	Forma(int w, int l) {
		latime = w;
		lungime = l;
	}
```

```
};
// clasa derivata
class Dreptunghi : public Forma
{
public:
	// constructorul clasei derivate
	Dreptunghi(int w, int l) :Forma(w, l) {};
	// funcția de claculare a ariei
	int getAria()
	{
		return (lungime*latime);
	}
};
// clasa derivata
class Paralelipiped : public Dreptunghi
{
	int inaltimea;
public:
public:
	// constructorul clasei derivate
	Paralelipiped(int w, int l, int h) :Dreptunghi(w, l) {
		inaltimea = h;
	};
	// funcția de calculare a volumului
	int getVolum()
	{
		return (lungime*latime*inaltimea);
	}
};
int main(void)
{
	// crearea unui obiect
	Dreptunghi Rect(5, 7);
	// Afisarea ariei obiectului.
	cout << "Aria totala dreptunghi: " << Rect.getAria() << endl;
	Paralelipiped par(2, 3, 4);
	cout << "Aria totala dreptunghi: " << par.getAria() << endl;
	cout << "Volumul totala paralelipiped: " << par.getVolum() << endl;
	return 0;
}
```

Exemplu 4

```
#include <iostream>
using namespace std;
// clasa de baza
class Forma
{
protected:
	int latime;
	int lungime;
public:
	// constructorul clasei de baza
	Forma(int w, int l) {
		latime = w;
		lungime = l;
	}
};
// clasa derivata
class Dreptunghi : public Forma
```

```
{
public:
	// constructorul clasei derivate
	Dreptunghi(int w, int l) :Forma(w, l) {};
	// funcția pentru calcularea ariei
	int getAria()
	{
		return (lungime*latime);
	}
};
class Atreiadimensiune
{
protected:
	int inaltimea;
public:
	Atreiadimensiune(int h) { inaltimea = h; };
};
// clasa derivata
// mosternire multipla
class Paralelipiped : public Dreptunghi, public Atreiadimensiune
{
public:
	Paralelipiped(int w, int l, int k) : Dreptunghi(w, l), Atreiadimensiune(k) {};
	int getVolum()
	{
		return (lungime*latime*inaltimea);
	}
};
int main(void)
{
	Dreptunghi Rect(8, 7);
	// Afisarea ariei obiectului.
	cout << "Aria totala dreptunghi: " << Rect.getAria() << endl;
	Paralelipiped par(9, 8, 4);
	cout << "Aria totala dreptunghi: " << par.getAria() << endl;
	cout << "Volumul totala paralelipiped: " << par.getVolum() << endl;
	return 0;
}
```

Exemplu 5

```
#include <iostream>
#include<conio.h>
using namespace std;
class Carte
{
	char titlu[64];
	float const cost2 = 100;

protected:
	void afis_cost2()
	{
		cout << "costul2 " << cost2 << endl;
	}
public:
	// cconstructor
	Carte(char *titlu)
	{
		strcpy(Carte::titlu, titlu);
		cout << "constructorul clasei de baza1" << endl;
```

```
	}
	void afis_carte()
	{

		cout << "titlul cartii este " << titlu << endl;
	}
	// destructor
	~Carte()
	{
		cout << "destructorul clasei de baza1" << endl;
	}
protected:
	float cost;
	// funcția pentru afișarea costului
	void afis_cost()
	{
		cout << "cartea costa " << cost << endl;
	}
};
class Pagini
{
protected:
	int linii;
public:
	// constructor
	Pagini(int linii)
	{
		Pagini::linii = linii;
		cout << "constructor clasa de baza2" << endl;
	}
	// functia pentru afisare
	void afis_pagini()
	{
		cout << "cartea are nr de pagini " << linii << endl;
	}
	// destructor
	~Pagini()
	{
		cout << "destructorul clasei de baza2" << endl;
	}
};
// moștenire multipla
class FisaBiblioteca :Carte, Pagini
{
	char autor[64], editura[64];
public:
	// constructor
	FisaBiblioteca(char *titlu, char *autor, char *editura) :Carte(titlu), Pagini(50)
	{
		strcpy(FisaBiblioteca::autor, autor);
		strcpy(FisaBiblioteca::editura, editura);
		cost = 100;
		cout << "Constructorul clasei derivate" << endl;
	}
	// functia de afisare
	void afis_biblio()
	{
		afis_carte();
		afis_cost();
		afis_cost2();
```

```
		cout << "autor: " << autor << " editura " << editura << endl;
		afis_pagini();
	}
	// destructor
	~FisaBiblioteca()
	{
		cout << "Destructorul clasei derivate" << endl;
	}
};
void main()
{
	// crearea unui obiect de tip FisaBiblioteca
	FisaBiblioteca fisa("Programare Orientata pe Obiecte", "Vasile STOICU-TIVADAR",
"Politehnica");
	// apelarea functiei de afișare
	fisa.afis_biblio();
	getch();
}
```

Pentru versiuni de Visual Studio inferioare 2015, la declararea constantelor de la începutul codului sursă se va folosi următoarea variantă:

```
....
class Carte
{
	char titlu[64];
	float const cost2;
protected:
	void afis_cost2()
	{
		cout << "costul2 " << cost2 << endl;
	}
public:
	Carte(char *titlu):cost2(100)
	{
		strcpy(Carte::titlu, titlu);
		cout << "constructorul clasei de baza1" << endl;
	}
	void afis_carte()
	{
		cout << "titlul cartii este " << titlu << endl;
	}
	~Carte()
	{
		cout << "destructorul clasei de baza1" << endl;
	}
protected:
	float cost;
	void afis_cost()
	{
		cout << "cartea costa " << cost << endl;
	}
};
```

Exemplu 6

```
#include<iostream>
#include<string>

using namespace std;

class Persoana {
	string nume;
	string prenume;
	unsigned int varsta;
public:
	//constructor vid folosit pentru declararea vectorului
	Persoana() {}
	//constructor pentru initializarea variabilelor
	Persoana(string nume, string prenume, unsigned int varsta)
	{
		Persoana::nume = nume;
		Persoana::prenume = prenume;
		Persoana::varsta = varsta;
	}
	//functie de afisare
	void afisare()
	{
		cout << "-----------------------------\n";
		cout << "Numele: " << Persoana::nume << "\n";
		cout << "Prenume: " << Persoana::prenume << "\n";
		cout << "Varsta: " << Persoana::varsta << "\n";
	}
	//returnare nume pentru functia de cautare
	string returnareNume()
	{
		return Persoana::nume;
	}
	//returnare prenume pentru functia de stergere
	string returnarePrenume()
	{
		return Persoana::prenume;
	}
};
class Angajat : public Persoana {
	unsigned int salar;
	string departament;
public:
	//constructor vid folosit pentru declararea vectorului
	Angajat() {}
	//constructor pentru initializarea variabilelor
	Angajat(string nume, string prenume, int varsta, string departament,
unsigned int salar) : Persoana(nume, prenume, varsta)
	{
		Angajat::departament = departament;
		Angajat::salar = salar;
	}
	//functia de afisare
	void afisare()
	{
		Persoana::afisare();
		cout << "Departament: " << Angajat::departament << "\n";
		cout << "Salar: " << Angajat::salar << "\n";
		cout << endl;
	}
};
```

```
int main()
{
	string n, p, d, pren;
	string num;
	int v, s, ok = 0, ok2 = 0;
	Angajat angajati[20];
	unsigned int nr;
	int opt;
	do {
		//meniu
		cout << "1. Introducere angajati" << endl;
		cout << "2. Afisare angajati" << endl;
		cout << "3. Cautare angajati" << endl;
		cout << "4. Stergere angajat" << endl;
		cout << "0. Iesire." << endl;
		cout << " Dati optiunea: ";
		cin >> opt;
		switch (opt)
		{
		case 1:
			//citire angajati
			cout << "Dati numarul de angajati: ";
			cin >> nr;
			for (int i = 0; i < nr; i++)
			{
				cin.get();
				cout << "Dati numele: ";
				getline(cin, n);
				cout << "Dati prenume: ";
				getline(cin,p);
				cout << "Dati varsta:";
				cin >> v;
				cin.get();
				cout << "Dati departamentul: ";
				getline(cin, d);
				cout << "Dati salarul:";
				cin >> s;
				//apelare constructor pentru fiecare angajat si
introducerea in vector a angajatilor
				angajati[i] = Angajat(n, p, v, d, s);
			}
			break;
		case 2:
			//afisare angajati
			for (int i = 0; i < nr; i++) {
				angajati[i].afisare();
			}
			break;
		case 3:
			//cautare angajat dupa nume
			fflush(stdin);
			cin.get();
			cout << "Dati numele cautat: ";
			getline(cin,num);
			for (int i = 0; i < nr; i++)
				if (num.compare(angajati[i].returnareNume()) == 0)
				{
					cout << "Am gasit angajatul!\n";
					ok = 1;
				}
			if (ok == 0)
				cout << "Angajatul nu s-a gasit!\n";
```

```
                        break;
                case 4:
                        //stergere angajat dupa prenume
                        cin.get();
                        cout << "Dati prenumele care vreti sa-l stergeti: ";
                        getline(cin, pren);
                        for (int i = 0; i < nr; i++)

                                if (pren.compare(angajati[i].returnarePrenume()) ==
0)
                                {
                                        for (int j = i; j < nr; j++)
                                                angajati[j] = angajati[j + 1];
                                        nr--;
                                        ok2 = 1;
                                }
                        if (ok2 == 0)
                                cout << "Prenumele nu exista, nu s-a sters
nimic!\n";
                        break;
                case 0:
                        break;

                }
        } while (opt != 0);
        return 0;
}
```