POO – C++ - Laborator 7

Cuprins

1. Moştenirea

Moştenirea este posibilitatea de a crea clase noi, definite ca extensii ale altor clase. Clasa iniţială se numeşte ***clasă de bază***, iar clasa nouă, care extinde clasa de bază – ***clasă derivată***. Proprietatea cea mai importantă a unei clase derivate este aceea de a ***moşteni*** toţi membrii definiţi în clasa de bază. Deasemenea, clasa derivată poate să includă şi membri noi, acesta fiind unul din scopurile pentru care este creată. Moştenirea este cel de-al 2-lea principiu al programării orientate obiect, următorul după încapsulare.

De exemplu putem să declarăm clasa `Persoana`, care să reprezinte orice persoană. Din `Persoana` putem să derivăm clasa `Student`. Clasa `Persoana` va avea câmpurile private `nume` şi `prenume`, şi metodele publice `setValoriPersoana()` şi `afisare()`. Clasa `Student` va moşteni toţi membrii clasei `Persoana`, şi va avea în plus câmpul privat `grupa`, şi metodele publice `setValoriStudent()` şi o altă metodă `afisare()`, despre care vom discuta mai jos.

Ca să declarăm o clasă derivată, folosim următoarea sintaxă:

```
class nume_clasa_derivata: public nume_clasa_baza {
      /*corpul clasei derivate*/
};
```

Clasa de bază trebuie să fie declarată mai sus, folosind sintaxa obişnuită.

Prezentam un exemplu complet de moştenire:

```
      //clase derivate
#pragma warning(disable : 4996)
#include<iostream>
#include<conio.h>
#include<string.h>
using namespace std;

class Persoana {
protected:
      char nume[20], prenume[20];
public:
      void setValoriPersoana(char nume[], char
prenume[]);
      void afisare();
};

class Student: public Persoana {
private:
      int grupa;
public:
      void setValoriStudent(char nume[], char prenume[],
            int grupa);
      void afisare();
};

void Persoana::setValoriPersoana(char nume[], char
prenume[]) {
      strcpy(this->nume, nume);
      strcpy(this->prenume, prenume);
}

void Persoana::afisare() {
      cout << nume << " " << prenume << endl;
}

void Student::setValoriStudent(char nume[], char
prenume[], int grupa) {
      setValoriPersoana(nume, prenume);
      this ->grupa = grupa;
}

void Student::afisare() {
      cout << nume << " " << prenume << ", grupa: " <<
grupa << endl;
}

int main() {
      Persoana persoana;
      Student student;
      persoana.setValoriPersoana("Vasile", "Dumitrescu");
      student.setValoriStudent("Ion", "Ciubotaru", 1104);
      persoana.afisare();
      student.afisare();
      _getch();
      return 0;
}
```

Ieşire:

```
Vasile Dumitrescu
Ion Ciubotaru, grupa: 1104
```

Acest exemplu conţine mai multe aspecte noi.

1. Modul de acces protected. Observăm că membrii `nume` şi `prenume` din clasa `Persoana` au fost definiţi cu un nou mod de acces – `protected`. Acest mod de acces este strans legat de moştenire. Membrii definiti cu mod `protected` pot fi accesaţi de oriunde din interiorul clasei în care au fost definiţi, sau din clasele derivate. Avem nevoie de un astfel de mod pentru cîmpurile din `Persoana`, pentru a le putea accesa din `Student::afisare()`.

Practic, drepturile pentru modul `protected` se situează între modurile `private` şi `public`. Mai jos prezentăm un sumar cu modurile de acces:

Mod de acces	**public**	**protected**	**private**
Membri ai aceleiaşi clase	da	da	da
Membri ai claselor derivate	da	da	nu
Non-membri	Da	nu	nu

Aici non-membri semnifică orice funcţie din afara clasei sau a claselor derivate, cum ar fi funcţia `main()`, orice funcţie globală sau metodele altei clase.

2. **Modul de moştenire.** Observăm declaraţia clasei `Student`:

```
class Student: public Persoana
```

Cuvântul cheie `public` semnifică modul de acces maxim pe care îl vor avea membrii moşteniţi din clasa de bază. Specificând modul de moştenire `public`, toţi membrii moşteniţi îşi vor păstra modul de acces iniţial.

Dacă moştenim clasa de bază în modul `private`, toţi membrii moşteniţi din `Persoana` îşi vor păstra modul de acces pentru clasa `Student`, dar vor deveni privaţi pentru restul programului. De exemplu, vom putea accesa metoda `setValoriPersoana()` din `setValoriStudent()`, şi programul de mai sus se va compila. Dar nu vom putea accesa `student.setValoriPersoana()` din funcţia `main()`. Se poate folosi şi modul de moştenire `protected`.

În aproape toate cazurile se foloseşte moştenirea publică. Celelalte moduri de moştenire nu sunt recomandate.

3. Ascunderea metodei `afisare()`. Metoda `afisare()` a fost declarată atât în clasa `Persoana`, cât şi în `Student` cu acelaşi nume şi aceeaşi listă de parametri. Este permisă o astfel de declaraţie. În acest caz, metoda afişare din clasa derivată ***ascunde*** metoda cu acelaşi prototip din clasa de bază. În apelul

```
student.afisare();
```

se va apela metoda afişare din clasa `Student`. De fapt clasa `Student` va avea 2 metode cu acelaşi nume şi aceiaşi parametri – `Persoana::afisare()` şi `Student::afisare()`. Dar a 2-a metodă o ascunde pe prima. Totuşi, metoda `Persoana::afisare()` nu este dispărută definitiv, ea poate fi accesată de exemplu de alte metode ale clasei `Persoana`.

2. Membrii moşteniţi din clasa de bază

Clasa derivată moşteneşte următorii mebri ai clasei de bază:

- Câmpurile
- Metodele
- Operatorii supraîncărcaţi, mai puţin operatorul =.

Nu sunt moşteniţi din clasa de bază:

- Constructorii
- Destructorul
- Membrii `operator=()`
- Prietenii

3. Crearea şi ştergerea obiectelor instanţiate dintr-o clasă derivată

Deşi constructorul şi destructorul clasei de bază nu sunt moşteniţi, aceştia au un rol şi pentru clasa derivată.

Atunci când o nouă instanţă a clasei derivate este creată, compilatorul apelează mai întâi constructorul clasei de bază fără parametri, şi pe urmă constructorul clasei derivate.

Atunci când instanţa unei clase derivate este ştearsă, compilatorul va apela mai întâi destructorul clasei derivate, şi mai apoi destructorul clasei de bază.

Alternativ, putem să specificăm ce constructor al clasei de bază trebuie apelat pentru fiecare constructor al clasei derivate. Folosind următoarea sintaxă pentru a declara constructorul:

```
nume_clasa_derivata(parametri constructor)
    : nume_clasa_de_baza (parametri constructor clasa de
baza) {...}
```

De exemplu:

```
// constructori si clase derivate
#include<iostream>
using namespace std;
#include<conio.h>

class A {
public:
      A() {
            cout << "A: fara parametri\n";
      }
      A(int a) {
            cout << "A: parametru int\n";
      }
};

class B : public A {
public:
```

```
	B (int a) {
		cout << "B: parametru int\n\n";
	}
};

class C : public A {
public:
	C(int a) : A(a){
		cout << "C: parametru int\n\n";
	}
};

int main () {
	B b(0);
	C c(0);
	_getch();
	return 0;
}
```

Ieşire:

```
A: fara parametri
B: parametru int

A: parametru int
C: parametru int
```

Remarcăm că atunci când se crează un obiect de tip `B`, se apelează constructorul fără parametri al clasei `A`. Iar atunci când se instanţiază `C`, se apelează constructorul cu parametru al lui `A`. Diferenţa se datorează declaraţiei constructorilor claselor `B` şi `C`:

```
B(int a) /*nu s-a specificat constructorul bazei*/
	/* se apeleaza constructorul implicit*/
C(int a) : A(a) /* apeleaza constructorul bazei
specificat*/
```

În exemplele de până acum clasa derivată moştenea o singură clasă de bază, ceea ce se numeşte **moştenire simplă**. În C++ există posibilitatea ca o clasă să moştenească mai multe clase de bază. Această proprietate se numeşte **moştenire multiplă**.

4. Ierarhii de clase

Dacă considerăm o mulţime de obiecte diferite dar înrudite, şi dacă din ele grupăm toate obiectele care au numai proprietăţile comune pentru toată mulţimea, acestea vor fi instanţieri ale unei clase pe care am numit-o clasă de bază. Dacă grupăm în continuare obiecte care au proprietăţile clasei de bază la care se adaugă proprietăţi proprii grupului, obţinem una sau mai multe clase derivate. În felul acesta se ajunge la noţiunea de ierarhie a claselor.

Clasa de bază a ierarhiei are întotdeauna nivelul 0. Clasele derivate din ea au nivel 1. Clasele derivate din clasele de nivel 1 au nivel 2, etc. Clasa de nivel 1 din care se derivează o clasă de nivel 2 este clasă de bază pentru aceasta. Deci noţiunea de clasă de bază – clasă derivată are caracter recursiv.

Exemplu:

```
class X {
	//...
};
class Y :public X{
	//...
};
class Z : public Y{
	//...
};
```

În exemplul prezentat avem o ierarhie de 3 clase.

5. Proiectul Persoana

Mai jos este dat codul sursă a unui program ce conţine 2 clase – `Persoana` şi `Data`. Exerciţiile din acest laborator vor presupune completarea acestui program cu facilităţi noi. Pentru început, să creăm un proiect pe baza acestor fişiere:

Data.h

```
#ifndef _Data_h_
#define _Data_h_

class Data {

private:
	int an,luna,zi;

public:
	Data(){}
	Data(int an, int luna, int zi);
	int getAn();
	int getLuna();
	int getZi();

	/*returneaza 1 daca this > data2, 0 daca this <=
data2*/
	int maiMare(Data data2);
};

#endif
```

Data.cpp

```
#include<iostream>
#include"Data.h"

Data::Data(int an, int luna, int zi){
	this->an = an;
	this->luna = luna;
	this->zi = zi;
}

int Data::getAn() {
	return an;
}

int Data::getLuna() {
	return luna;
}

int Data::getZi() {
	return zi;
}


/*returneaza 1 daca data1 > (este mai recenta decat)
data2, 0 in caz contrar*/
int Data::maiMare(Data data2) {
	if (an > data2.an) {
		return 1;
	} else if (an < data2.an) {
		return 0;
	} else {
		if (luna > data2.luna) {
			return 1;
		} else if (luna < data2.luna) {
			return 0;
		} else {
			if (zi > data2.zi) {
				return 1;
			} else if (zi < data2.zi) {
				return 0;
			} else {
				return 0;
			}
		}
	}
}
```

Persoana.h

```
#include"Data.h"

#ifndef _Persoana_h_
#define _Persoana_h_
#pragma warning(disable : 4996)

class Persoana {
private:
	char *nume, *prenume;
	Data *dataNastere;

protected:
	void afisarePersoana();

public:
	Persoana(char *nume, char *prenume, Data
*dataNastere);
	~Persoana();
	char *getNume();
	char *getPrenume();
	Data *getDataNastere();
	void afisare();
};

#endif
```

Persoana.cpp

```
#include<iostream>
#include<string.h>
#include "Data.h"
#include"Persoana.h"
using namespace std;

Persoana::Persoana(char *nume, char *prenume, Data
*dataNastere) {
	this->nume = new char[strlen(nume) + 1];
	this->prenume = new char[strlen(prenume) + 1];
	strcpy(this->nume, nume);
	strcpy(this->prenume, prenume);
	this->dataNastere =
			new Data(*dataNastere);
}

Persoana::~Persoana() {
	delete[] nume;
	delete[] prenume;
	delete dataNastere;
}

char *Persoana::getNume(){
	return nume;
}

char *Persoana::getPrenume() {
```

```
	return prenume;
}

Data *Persoana::getDataNastere() {
	return dataNastere;
}

void Persoana::afisarePersoana() {
	cout << nume << " " << prenume
	    << ", data nastere: "
	      << dataNastere->getAn() << "."
	      << dataNastere->getLuna() << "."
	      << dataNastere->getZi();
}

void Persoana::afisare() {
	afisarePersoana();
	cout << endl;
}
```

DemoMain.cpp

```
#include"Data.h"
#include"Persoana.h"

int main() {

	Data data(2000, 3, 20);
	Persoana radu("Radu", "Stefan", &data);

	radu.afisare();
	return 0;
}
```

6. Exerciţii

1. Creaţi o clasă `Student`, derivată din `Persoana`. Clasa va avea următorii membri noi:
    - cîmpul privat grupa de tip întreg. Grupa va lua valori între `1001` şi `1999`
    - `afisare()` – va afişa studentul la consolă.

Scrieţi o funcţie main care să creeze un vector de 5-10 studenţi, din 3 grupe diferite, şi să-i afişeze sortaţi după grupe.

2. Scrieţi o clasă `Angajat`, derivată din `Persoana`. Clasa angajat va avea următorii membri:
    - câmpul privat `dataAngajare` de tip `Data`
    - câmpul privat `salariu` de tip `int`– salariu lunar.
    - Constructor care să iniţializeze toate câmpurile pe baza parametrilor
    - Destructor
    - Metode `getDataAngajare()`, `getSalariu()`, care să returneze valoarea câmpurilor.
    - `int getVarstaAnagajare()` – va returna vârsta la care s-a făcut angajarea, în ani împliniţi.

- `void afisarePerioadaMuncita(Data &dataCurenta)` – va afişa periada muncită în firmă, până la data curentă. Şirul va avea formatul "ani.luni.zile". Exemplu: "2.9.20". Vom considera pentru simplitate că toate lunile au câte 30 de zile.
- `afisare()` – va afişa informaţiile despre angajat la consolă. Doar informaţiile stocate în câmpurile clasei, nu şi cele calculate. Este indicat să se folosească metoda `afisarePersoana()`.

Scrieţi o funcţie `main` care să creeze o listă de 3-4 angajaţi, şi să afişeze vârsta angajării, perioada muncită şi detaliile despre angajat, pentru toţi angajaţii, folosind metodele definite.