Lab 8 – P2

Cuprins

1. Moștenirea

Moștenirea este principul de bază al programării orientate obiect ce permite crearea unor clase noi, definite ca extensii ale altor clase. Moștenirea este cel de-al 3-lea principiu al programării orientate obiect, după abstractizare și încapsulare. Este un principiu extrem de folosit în programare pentru că permite crearea unor ierarhii de clase.

Pentru a putea implementa conceptul de moștenire, este nevoie de minim două clase:

- **Clasa de bază**
- **Clasa derivată** – clasa ce extinde clasa de bază

Moștenirea înseamnă că se permite clasei derivate să moștenească toți membrii clasei de bază. Din punct de vedere tehnic, moștenirea înseamnă că în clasa derivată se pot accesa membrii clasei de bază, în funcție de drepturile de acces ale acestora.

Un obiect de tipul clasei derivate va avea deci ca membri atât membrii proprii, cât și membrii moșteniți de la clasa de bază.

Pentru declararea unei clase derivate, se folosește următoarea sintaxă:

```
class nume_clasa_derivata: public nume_clasa_baza
{
    ...
};
```

Bineînțeles, clasa de bază trebuie să fie declarată anterior declarării clasei derivate. Dacă se declară fiecare clasă în fișierul header propriu, atunci fișierul header în care a fost declarată clasa de bază se va include în fișierul header al clasei derivate.

Ca exemplu, declarăm clasa `Persoana`, care să conțină ca membri, informațiile reprezentative unei persoane (pentru simplitate am ales doar `m_sNume` și `m_sAdresa`). Din clasa `Persoana` derivăm clasa `Student`. Clasa `Persoana` va avea, pe lângă variabilele membre `m_sNume` și `m_sAdresa`, metodele publice `schimbareAdresa()` și `afisareProfil()`.

Clasa `Student` va avea variabilele membre private `m_iAnStudiu` și `m_iNotaP2` precum și metodele publice `inscriereAnStudiu()` și `afisareProfil()`. Faptul că avem clasa Student care este derivată din clasa Persoana înseamnă că un obiect de tip Student va avea în componența sa variabilele și funcțiile membre proprii, dar și variabilele și funcțiile membre din clasa de bază, clasa Persoana. Cu alte cuvinte, un obiect Student va avea ca membri `m_iAnStudiu`, `m_iNotaP2` dar și `m_sNume` și `m_sAdresa`. Ca funcții, un obiect Student va avea acces la funcțiile proprii clasei Student, inscriereAnStudiu() și afisareProfil() dar și la funcțiile din clasa Persoana. Un caz special îl reprezintă funcția afișareProfil care are același prototip și în clasa Student dar și în clasa Persoană. Acest caz special va fi tratat ulterior.

Codul C++ aferent celor două clase descrise mai sus este:

Persoana.h

```
class Persoana
{
protected:
	std::string m_sNume;
	std::string m_sAdresa;
public:
	Persoana(std::string nume, std::string adresa);
	void schimbareAdresa(std::string adresaNoua);
	void afisareProfil();
};
```

Student.h

```
class Student: public Persoana
{
	int m_iAnStudiu;
	int m_iNotaP2;
public:
	Student(std::string nume, std::string adresa, int anStudiu, int notaP2);
	void inscriereAnStudiu(int anStudiuNou);
	void afisareProfil();
};
```

Persoana.cpp

```
Persoana::Persoana(std::string nume, std::string adresa)
{
	m_sNume = nume;
	m_sAdresa = adresa;
}
void Persoana::afisareProfil()
{
	std::cout << "Nume: " << m_sNume << ", Adresa: " << m_sAdresa << std::endl;
}
```

Student.cpp

```
Student::Student(std::string nume, std::string adresa, int anStudiu, int notaP2) :
		Persoana(nume, adresa), m_iAnStudiu(anStudiu), m_iNotaP2(notaP2) {
}

void Student::afisareProfil()
{
	//Persoana::afisareProfil();
	std::cout << "Nume: " << m_sNume  <<  ", Adresa: " << m_sAdresa <<
std::endl;
	std::cout << "An studiu: " << m_iAnStudiu << ", Nota P2: " << m_iNotaP2 <<
std::endl;
}
```

Test.cpp

```
int main()
{

	Persoana p1("Ion", "str. Libertatii");
	Persoana p2("Andrei", "str. Calea Victoriei");
	Student s1("Mihai", "str. Cuza Voda", 2, 5);
	Student s2("Petru", "str. Stefan cel mare", 2, 3);

	p1.afisareProfil();
	p2.afisareProfil();
	s1.afisareProfil();
	s2.afisareProfil();
	return 0;
}
```

1.1. Modul de acces protected

Observăm că membrii nume și prenume din clasa Persoana au fost definiți cu un nou mod de acces – protected. Acest mod de acces este strâns legat de moștenire.

Membrii definiți cu mod protected pot fi accesați de oriunde din interiorul clasei în care au fost definiți, sau din clasele derivate. Avem nevoie de un astfel de mod pentru câmpurile din Persoana, pentru a le putea accesa din `Student::afisareProfil()`.

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

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

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

1.2. Modul de moștenire

Observăm declarația clasei Student: `class Student: public Persoana`

Cuvântul cheie public semnifică modul de acces maxim pe care îl vor avea membrii moșteniți din clasa de bază. Specificând modul de moștenire public, toți membrii moșteniți își vor păstra modul de acces inițial.

Dacă moștenim clasa de bază în modul private, toți membrii moșteniți din Persoana își vor păstra modul de acces pentru clasa Student, dar vor deveni privați pentru restul programului. De exemplu, vom putea accesa metoda `schimbareAdresa()` din metodele clasei Student, și programul de mai sus se va compila. Dar nu vom putea accesa `student.schimbareAdresa()` din funcția `main()`. Se poate folosi și modul de moștenire protected.

În aproape toate cazurile se folosește moștenirea publică. Celelalte moduri de moștenire nu sunt recomandate.

1.3. Apelarea constructorului din clasa de bază

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) {...}
```

Dacă nu este specificat ce constructor al clasei de bază trebuie apelat, atunci, implicit, este apelat constructorul fără argumente.

În exemplul de mai sus, constructorul cu patru argumente din clasa Student apelează constructorul cu două argumente din clasa de bază Persoana.

```
Student::Student(string nume, string adresa, int anStudiu, int notaP2) :
      Persoana(nume, adresa), m_iAnStudiu(anStudiu), m_iNotaP2(notaP2) { }
```

1.4. Ascunderea metodei afisareProfil()

Metoda `afisareProfil()` a fost declarată atât în clasa Persoana, cât și în Student cu același nume și aceeași listă de parametri. Este permisă o astfel de declarație. În acest caz, metoda afișare din clasa derivată ascunde metoda cu același prototip din clasa de bază. În apelul

```
s1.afisareProfil();
```

se va apela metoda `afisareProfil` din clasa Student. De fapt clasa Student va avea 2 metode cu același nume și aceiași parametri: `Persoana::afisareProfil()` și `Student::afisareProfil()`. Dar a 2-a metodă o ascunde pe prima. Totuși, metoda `Persoana::afisareProfil()` nu este dispărută definitiv, ea poate fi accesată de exemplu de alte metode ale clasei Persoana.

Apelează metoda `afisareProfil` din clasa Persoana în cadrul clasei Student se poate face cu ajutorul operatorului de rezoluție astfel: `Persoana::afisareProfil();`

2. Membrii moșteniți din clasa de bază

Clasa derivată moștenește următorii membri ai clasei de bază:

- Câmpurile
- Metodele
- Operatorii supraîncărcați, mai puțin operatorul =.

Nu sunt moșteniți din clasa de bază:

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

3. Exerciții

1. Creați un proiect și reproduceți aplicația oferită ca exemplu în laborator. Urmăriți cu Debug-ul variabilele de tip Student și observați cum influențează moștenirea modul în care arată un obiect Student (în fereastra de watch).
2. Extindeți aplicația creată și implementați următoarele:

- Definiți metoda `schimbareAdresa()` din clasa Persoana.
- Definiți metoda `inscriereAnStudiu()` din clasa Student.
- Declarați și definiți în clasa Student o metodă cu numele `schimbaNota` care permită schimbarea notei studentului.
- Creați o clasă Profesor care să extindă clasa Persoana și care să aibă în plus proprietățile `m_sGradDidactic` de tipul string și un vector de obiecte de tipul Student cu numele `m_studenti` (se va utiliza clasa vector din std – vezi secțiunea Referințe + curs).
- Declarați și definiți în clasa Profesor:
  - o metodă cu numele `acordaNota` care să primească două argumente: poziția studentului în vectorul `m_studenti` și nota pe care o va primi studentul respectiv.
  - o metodă cu numele `afiseazaStudenti()` care va folosi un iterator pentru a apela metoda `afisareProfil` pentru fiecare student din vectorul `m_studenti`.
  - o metodă care va sorta studenții descrescător după notă. (se va folosi funcția sort din std).
- Testați în `main` toate metodele implementate.

4. Referințe

- http://www.cplusplus.com/reference/string/
- http://www.cplusplus.com/reference/vector/vector/
- http://www.cplusplus.com/reference/vector/vector/begin/
- http://www.cplusplus.com/reference/iterator/
- http://www.cplusplus.com/reference/algorithm/sort/