Curs 09

SUPRADEFINIREA

Domenii de nume, derivare multiplă, funcții virtuale

Domenii de nume (namespaces)

- Utilizate pentru a organiza codul sursă
- Permit gruparea de clase, obiecte, funcții sau variabile globale într-o singură entitate
- Evită coliziunile de nume (și astfel erorile de ambiguitate)
- Organizarea în namespaces se poate face în funcție de diferite criterii:
  - funcționalitate comună
  - modul/bibliotecă specific(ă)
  - tip aplicație
  - ș.a.

Exemplu: Domenii de nume

```
namespace CursOOP
{
     int numar_cursuri = 14;
     class TelefonMobilPliabil
     {
          //...
     };
     //...
};
```

```
namespace DispozitiveInteligente
{
     void read();
     class DevicePliabil
     {
          //...
     };
     //...
};
```

Exemplu: Domenii de nume

```
int main()
{
     cout << CursOOP::numar_cursuri;
     CursOOP::TelefonMobilPliabil t1;
     DispozitiveInteligente::DevicePliabil d;
     DispozitiveInteligente::read();
};
```

Exemplu: Domenii de nume

```
using namespace CursOOP;
using namespace DispozitiveInteligente;

int main()
{
	cout << numar_cursuri;
	TelefonMobil t1;
	DevicePliabil d;
	//atentie la coliziunile de nume
	read();
	//daca mai exista o alta functie read() in scop se va utiliza
	DispozitiveInteligente::read();
};
```

Derivarea multiplă

- În C++ este permisă derivarea din mai multe clase simultan
- Utilizarea acesteia modelează faptul că tipul derivat este, în același timp, și oricare dintre tipurile de bază
- Se pot deriva oricât de multe clase, iar tipul de derivare se decide pentru fiecare clasă ce este derivată în parte
- Ordinea de apel a constructorilor/destructorilor în acest caz este dată de ordinea derivării (nu de cea definită explicit în cazul unor constructori)
- Upcasting-ul se poate face către oricare dintre clasele de bază

Exemplu: Derivarea multiplă

```
class TelefonMobilPliabil : public TelefonMobil, protected DevicePliabil
{
    //...
};
```

Derivarea din clase ce conțin membri cu același nume

- Una dintre problemele ce poate apărea în cazul derivării multiple este aceea a claselor de bază ce conțin un membru cu aceeași denumire
- În acest caz, este obligatorie utilizarea rezoluției de clasă pentru a specifica în mod explicit la care dintre membrii moșteniți se face referire
- Definirea unui nou membru cu același nume în clasa derivată, va duce la ascunderea („hiding") membrilor cu nume comun moșteniți

Exemplu: Membri cu același nume

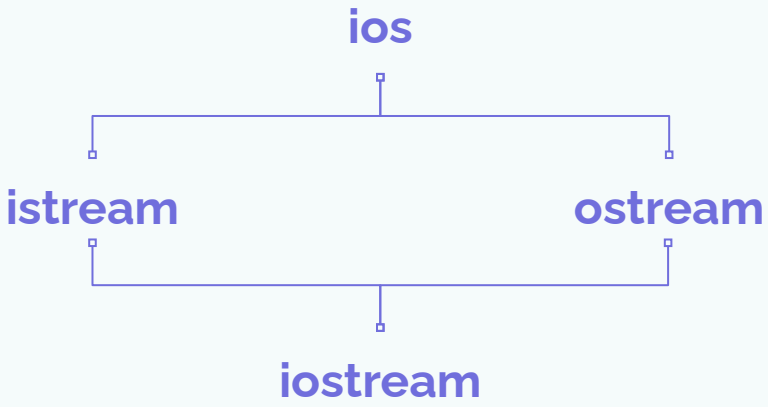
```
class TelefonMobil
{
public:
	string nume;
	//...
};
```

```
class DevicePliabil
{
public:
	string nume;
	//...
};
```

```
class TelefonMobilPliabil : public TelefonMobil, public DevicePliabil
{
	//...
	void metoda()
	{
		//nume = "ceva";
		TelefonMobil::nume = "ceva"; DevicePliabil::nume = "ceva";
	}
};
```

Problema moștenirii în romb (The Deadly Diamond)

- Din cauza moștenirii multiple, poate apărea o situație bizară în C++ atunci când o clasă se derivează multiplu din cel puțin două clase cu o bază comună



Problema moștenirii în romb (The Deadly Diamond)

- În aceste cazuri membrii clasei de bază sunt moșteniți de mai multe ori în clasa finală (de fiecare dată pe filiera unei clase de pe nivelul intermediar)
- Astfel se multiplică spațiul de memorie ocupat de aceștia în clasa finală din ierarhia de derivări
- În plus, upcasting-ul nu este permis direct către prima clasă din ierarhie, putându-se face doar etapizat
- Dacă acest lucru nu deranjează, lucrurile pot rămâne așa

Exemplu: The Deadly Diamond

```
class Device
{
public:
        string producator;
};

class TelefonMobil : public Device
{
};

class DevicePliabil : public Device
{
};

class TelefonMobilPliabil : public TelefonMobil, public DevicePliabil
{
        //...
};
```

Exemplu: The Deadly Diamond

```
int main()
{
	TelefonMobilPliabil tel;
	tel.TelefonMobil::producator = "Samsung";
	//va afisa string vid
	cout << tel.DevicePliabil::producator << endl;
	//va afisa Samsung
	cout << tel.Device::producator << endl;
	tel.Device::producator = "Apple";
	//va afisa string vid
	cout << tel.DevicePliabil::producator << endl;
	//va afisa Apple
	cout << tel.TelefonMobil::producator << endl;
}
```

Exemplu: The Deadly Diamond

```
int main()
{
      Device d;
      TelefonMobil tm;
      DevicePliabil dp;
      TelefonMobilPliabil tmp;
      //eroare de compilare
      //d = tmp;
      d = tm = tmp;
      //sau
      d = dp = tmp;
}
```

Problema moștenirii în romb (The Deadly Diamond)

- Recomandarea generală este ca această situație să fie evitată
- Atunci când acest lucru nu este posibil, poate fi rezolvată prin derivarea virtuală a claselor de pe nivelul intermediar (a nu se confunda conceptul cu cel de funcție virtuală, despre care vom discuta puțin mai târziu)
- Dacă se utilizează derivarea virtuală, atunci membrii comuni sunt moșteniți o singură dată, iar upcasting-ul direct către clasa părinte este permis

Exemplu: The Deadly Diamond

```
class Device
{
public:
        string producator;
};

class TelefonMobil : virtual public Device
{
};

class DevicePliabil : virtual public Device
{
};

class TelefonMobilPliabil : public TelefonMobil, public DevicePliabil
{
        //...
};
```

Exemplu: The Deadly Diamond

```
int main()
{
	TelefonMobilPliabil tel;
	tel.TelefonMobil::producator = "Samsung";
	//va afisa Samsung
	cout << tel.DevicePliabil::producator << endl;
	//va afisa Samsung
	cout << tel.Device::producator << endl;
	tel.Device::producator = "Apple";
	//va afisa Apple
	cout << tel.DevicePliabil::producator << endl;
	//va afisa Apple
	cout << tel.TelefonMobil::producator << endl;
}
```

Exemplu: The Deadly Diamond

```
int main()
{
	Device d;
	TelefonMobil tm;
	DevicePliabil dp;
	TelefonMobilPliabil tmp;
	//upcastingul se poate face direct
	d = tmp;
}
```

Supradefinirea

- Cea de-a doua formă de polimorfism
- Considerată polimorfism pur sau puternic, deoarece funcțiile au exact aceeași signatură, iar alegerea funcției potrivite (aferente apelului) se face la momentul execuției („late binding”)
- Poate avea loc doar în contextul unei ierarhii de derivări, și doar dacă funcțiile/metodele sunt marcate ca virtuale
- Dacă funcția nu este virtuală, are loc doar ascunderea metodei din clasa de bază
- Se manifestă doar pe adrese, în funcție de conținutul de la acea zonă de memorie

Exemplu: Funcție virtuală

```
class TelefonMobil
{
public:
	virtual int durataDeIncarcare()
	{
		return 2;
	}
};

class TelefonMobilPliabil : public TelefonMobil
{
	//caracterul virtual al functiei se mosteneste
	int durataDeIncarcare()
	{
		return 3;
	}
};
```

Exemplu: Funcție virtuală

```
int main()
{
	TelefonMobil tm;
	TelefonMobilPliabil tmp;

	//va afisa 2
	cout << tm.durataDeIncarcare() << endl;

	//va afisa 3
	cout << tmp.durataDeIncarcare() << endl;

	tm = tmp;
	//va afisa tot 2
	cout << tm.durataDeIncarcare() << endl;
}
```

Exemplu: Funcție virtuală

```
int main()
{
	//supradefinirea are loc doar pe pointeri sau referinte
	TelefonMobil tm;
	TelefonMobilPliabil tmp;

	TelefonMobil& rtm = tmp;
	//va afisa 3
	cout << rtm.durataDeIncarcare() << endl;

	TelefonMobil* ptm = &tmp;
	//va afisa 3
	cout << ptm->durataDeIncarcare() << endl;
}
```

Supradefinire vs Supraîncărcare

- Are loc la execuție
- Funcțiile au aceeași signatură
- Poate avea loc doar în contextul unei derivări
- Funcția din clasa de bază trebuie marcată ca virtuală
- Este necesar ca funcția să fie apelată pe un pointer/referință ca supradefinirea să aibă loc

- Are loc la compilare
- Funcțiile nu au aceeași signatură (diferă prin numărul sau tipul parametrilor)
- Poate avea loc doar în același context (aceeași clasă, ierarhie de clase, zona globală, etc.)
- Funcțiile nu sunt marcate în mod special
- Apelul se face atât pe obiecte, cât și pe pointeri

Exemplu: Clientul care nu știe ce vrea

- Avem un client ce dorește o aplicație pentru a gestiona pontajul angajaților
- Spune că vrea să folosească aplicația doar în România, unde săptămâna de lucru este de 5 zile, de luni până vineri
- De asemenea spune că pontajul va fi introdus doar de la tastatură

Exemplu: varianta programatorului fără experiență

```
class Pontaj
{
private:
        int pontaj[31];

public:
        void introducerePontaj()
        {
                for (int i = 0; i < 22; i++)
                {
                        cout << "Pontaj pentru ziua " << (i + 1) << ": ";
                        cin >> pontaj[i];
                }
        }
```

Exemplu: varianta programatorului fără experiență (cont.)

```
        void afisarePontaj()
        {
                for (int i = 0; i < 22; i++)
                {
                        switch (i % 5)
                        {
                                case 0:
                                {
                                        cout << "Lu: "; break;
                                }
                                case 1:
                                {
                                        cout << "Ma: "; break;
                                }
                                case 2:
                                {
                                        cout << "Mi: "; break;
                                }
                                case 3:
                                {
                                        cout << "Jo: "; break;
                                }
                                case 4:
                                {
                                        cout << "Vi: "; break;
                                }
                        }
                        cout << pontaj[i] << endl;
                }
        }
};
```

Exemplu: varianta programatorului fără experiență (cont.)

```
int main()
{
     Pontaj pontaj;
     pontaj.introducerePontaj();
     pontaj.afisarePontaj();
}
```

Exemplu: Clientul care nu știe ce vrea

- Clientul, așa cum ne așteptam, se răzgândește
- Ar vrea să vândă aplicația și în țări ce nu încep săptămâna de lucru luni
- În plus, ar vrea să importe pontajul din fișiere CSV, în loc să le citească de la tastatură

Exemplu: Varianta programatorului cu experiență

- Declară funcțiile de la început virtuale
- Apelează funcțiile pe un pointer ce, momentan, reține adresa unui obiect de același tip
- Atunci când clientul vrea o nouă funcționalitate, derivează o clasă din cea existentă și supradefinește funcțiile
- Tot ce mai are de modificat e obiectul a cărui adresă este salvată de pointer, care acum este de tipul unei clase derivate
- În cazul în care clientul dorește o nouă opțiune sau să revină la cea veche, tot ce are de schimbat este adresa pe care o salvează pointerul

Exemplu: varianta programatorului cu experiență (cont.)

```
int main()
{
     Pontaj* pontaj = new Pontaj();
     pontaj->introducerePontaj();
     pontaj->afisarePontaj();
     delete pontaj;
}
```

Exemplul 2

Cum pot salva un vector de diferite tipuri de animale, astfel încât, atunci când îl parcurg, fiecare să scoată sunetul specific?