Programare Orientată Obiect

Laborator 7

Despre ce vorbim azi

- Derivarea (moștenirea)

Moștenire vs includere

- Includerea se folosește pentru a exprima o relație de tip *has a* și se folosește între clase diferite (*Vehicul* și *Roata* spre exemplu)
- Moștenirea se folosește pentru a exprima relații de tipul *is a* și se folosește pe clase de același fel (similare parțial)
- Când conceptul din lumea reală este componentă a altui concept avem de-a face cu includere (compunere)
- Când conceptul din lumea reală este o specializare a altui concept avem de-a face cu derivare (moștenire)

Avantaje

- Putem crea noi clase care preiau caracteristicile unei clase deja definite
- Scriem mai puțin cod prin reutilizarea părții comune celor două clase implicate în proces (cea din care se face derivarea și cea care moștenește caracteristici)
- Ne putem alege modul în care facem derivarea: **public**, **protected**, **private**

Derivarea publică

- Membrii publici ai clasei de bază rămân publici și în clasa derivată
- Membrii protected ai clasei de bază (asupra căreia clasa derivată are acces) devin protected în clasa derivată
- Membrii private ai clasei de bază sunt inaccesibili în clasa derivată (asta nu înseamnă că nu sunt moșteniți, ci doar că nu pot fi accesați)

Derivarea private

- Membrii publici devin private în clasa derivată
- Membrii protected devin private în clasa derivată
- Membrii private sunt și de această dată inaccesibili
- Observăm că derivarea private blochează accesul la toți membrii în cazul unei noi derivări
- Dacă nu se specifică explicit, este cea implicită

Derivarea protected

- Membrii publici devin protected prin moștenire
- Membrii protected rămân protected
- Membrii private sunt inaccesibili
- Derivarea protected păstrează accesul asupra zonei publice doar pentru clase derivate

Publicizarea

- Așa cum am văzut, prin derivare private membrii publici devin privați
- Putem alege ca o parte din aceștia să îi menținem publici printr-un procedeu numit *publicizare*
- Exemplu:

```
class A
{
        int x;
        public:
                int y, z;
};
```

```
class B : A
{
        public:
                A::y;
};
```

Precizări

- Clasa derivată are constructori proprii (expliciți sau impliciți) ce apelează la constructorii clasei de bază (implicit sau explicit)
- Se vor folosi constructorii clasei de bază pentru a inițializa partea de obiect moștenită și cei proprii pentru membrii noi introduși
- Așadar ordinea de apel pentru constructori este: întâi cel din bază, apoi cel derivat
- În cazul destructorilor ordinea de apel este exact invers: întâi cel din clasa derivată, apoi cel din bază

Exemplu

- Clasa *Vehicul* și clasa *Mașină*

Conversia derivat - bază

- Conversia derivat - bază este făcută implicit (orice mașină este un vehicul, nu și invers)
- De aceea putem folosi atribuiri de genul:

```
Vehicul v;
Masina m;
v = m;
```

- Conversia bază – derivat nu este făcută implicit datorită sensului unic de moștenire și poate fi implementată prin supraîncărcarea operatorului = doar în cazurile în care moștenirea este publică

Alte precizări

- Putem redefini funcții din bază în clasa derivat
- Apelul va fi făcut în funcție de tipul obiectului
- De asemenea putem supradefini operatori în clasele derivate (în caz contrar ei se moștenesc integral, cu excepția operator=)
- Funcțiile friend din bază rămân friend și în derivată
- Clasele friend nu se moștenesc

Funcții care nu se moștenesc integral

- Constructorii
- Destructorii
- operator=
- Operatorul cast