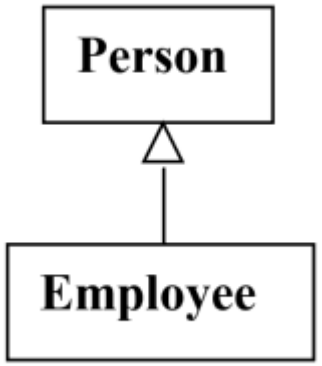
A Programozás Alapjai 2
Objektumorientált szoftverfejlesztés

Dr. Forstner Bertalan
forstner.bertalan@aut.bme.hu

BME AUT
Department of Automation and Applied Informatics

Öröklés

- Az OOP nyelvek harmadik fontos eleme
- Kétféle megközelítés:



  - **Specializáció**: ha az Alkalmazott egyfajta személy, akkor „örökölnie” kell a Személy tulajdonságait.
  - **Általánosítás**: ha az alkalmazott egyfajta személy, akkor az Alkalmazott bárhol használható, ahol a Személy.
    - A nyíl az ábrán a behelyettesíthetőséget jelenti.

Szintaxis és szemantika

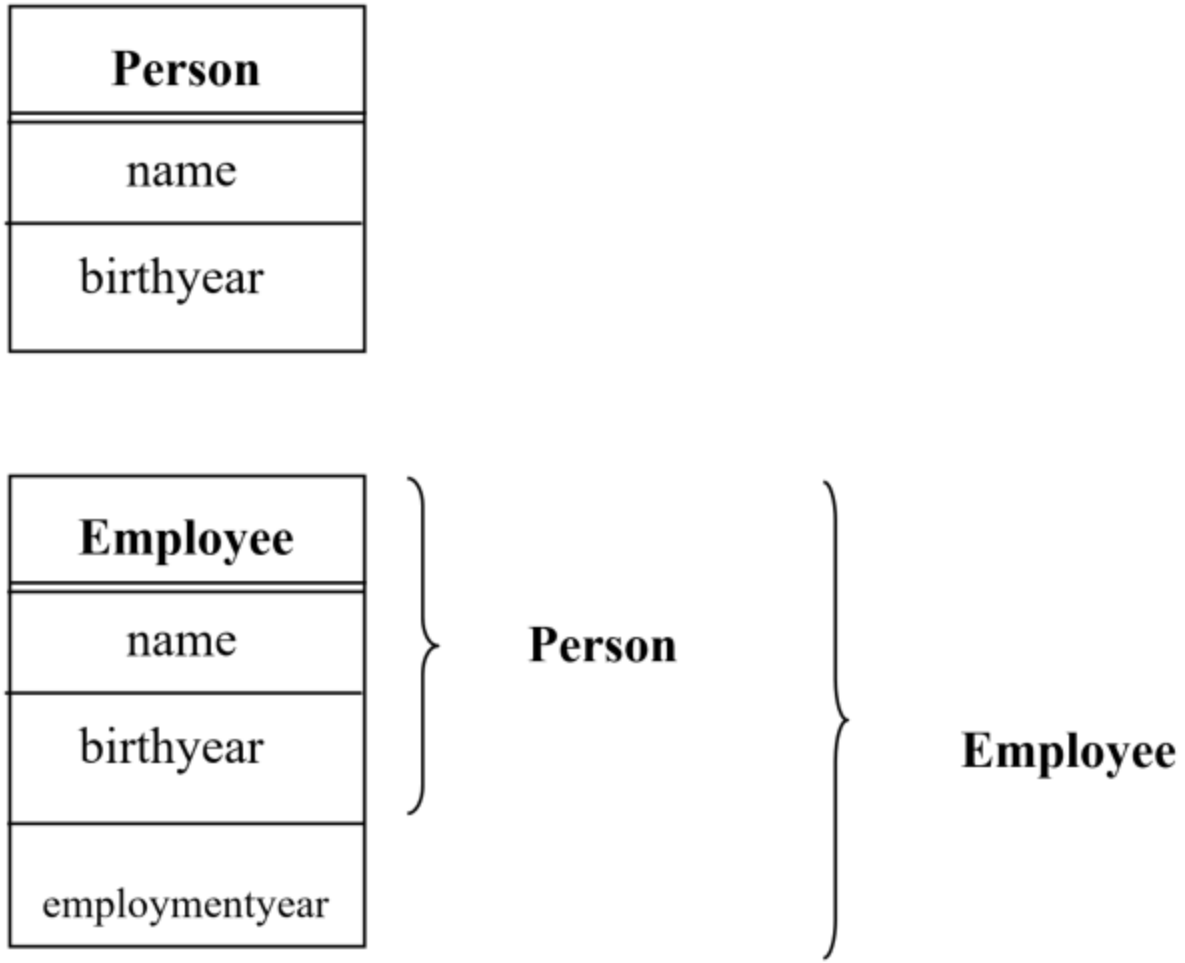
- Példa

- *ősosztály, alaposztály, ancestor class, ebből örököl:*
- *leszármazott osztály, derived class*
- *öröklés, származtatás, inheritance*

Tulajdonságok

- A leszármazott örökli a szülő **összes attribútumát** (tulajdonságait) és újakat adhat hozzá.
  - **Elvenni nem lehet**: felborulna az a fontos tulajdonság, hogy a dolgozó valójában egy személy.
- A leszármazott örökli a szülő **összes műveletét** (viselkedését)
  - újakat adhat hozzá, valamint a meglevőket felüldefiniálhatja (más implementációt adhat hozzá).
  - *A fejléc ugyanaz, csak a törzse más!*

Memória rajz



Behelyettesíthetőség

- Függvény felüldefiniálása:
  - Egy ősben meglévő függvénynek új implementációt adunk
  - Melyik hívódik meg, ha a leszármazotton hívjuk?
    - Mindig a felüldefiniált
    - (vagy a legkésőbb felüldefiniált az öröklési láncban)
  - Ha nem definiáltuk felül, megörökli az eredeti implementációt
  - Az ősosztálybeli hívható a scope operátorral a leszármazottból:
    Os::fuggvenynev()

Behelyettesíthetőség

- Egy dolgozóra tekinthetünk úgy, mint egy személyre, ha nem vesszük figyelembe a csak a leszármazottra érvényes tulajdonságokat (ez teljesen logikus).
- Például a Print kiírja a személy adatait. (De többet is szeretnénk: employmentYear)
- Példa
  - Print felüldefiniálása hogyan?

Előnyök

- kevesebbet kell írni
- normalizált megoldás, csak egy helyen kell módosítani a közös részeket
- ha felveszünk egy új dolgozót (pl. beszerző), akkor nem kell a meglevő osztályokat módosítani.
- az automatikus konverzió miatt: egységes kezelése különböző objektumoknak (majd később…)

Összefoglalva

- A leszármazott örökli a szülő:
  - attribútumait, tulajdonságait
  - interfészét: ugyanazokat az üzeneteket elfogadja, ugyanazok a műveletek végrehajthatók rajta.
    - Ezek a publikus műveletek.
    - Ebből adódik a behelyettesíthetőség: egy dolgozó egy személy is, így is tekinthetünk rá, és **minden**, ami fennáll a személyre, fennáll a dolgozóra is
- Implementációját
  - kivéve a private tagváltozók és metódusok.
  - Az öröklés emiatt a kód-újrafelhasználás eszköze.

Láthatóság

- Ami **public**: nem védett, mindenki eléri
- Ami **private**: csak az adott osztály metódusaiból érhető el
  - A többi osztály és globálisak nem érik el.
- Ami **protected**: az adott osztályban és a leszármazottban elérhető.
  - Ő is védett.

Az öröklött tagok láthatósága

- Tudjuk: az ősosztályban levő tagokat (attr. és metódus) a leszármazottak öröklik.
  - Mintha a leszármazottban is definiálnánk implicit módon.
- Mi lesz ezek láthatósága a leszármazottban?

Az öröklött tagok láthatósága

- Tudjuk: az ősosztályban levő tagokat (attr. és metódus) a leszármazottak öröklik.
  - Mintha a leszármazottban is definiálnánk implicit módon.
- Mi lesz ezek láthatósága a leszármazottban?
- Háromféleképpen lehet leszármaztatni. Szintaktika:
  - class B: _____ A {…};
    - Mi kerül a vonalra?

Az öröklött tagok láthatósága

	Öröklés típusa		
Ősosztályban lévő láthatóság	**public**	**protected**	**private**
public	public	protected	private
protected	protected	protected	private
private	nem látható	nem látható	nem látható

Az öröklött tagok láthatósága

	Öröklés típusa		
Ősosztályban lévő láthatóság	**public**	**protected**	**private**
public	public	protected	private
protected	protected	protected	private
private	nem látható	nem látható	nem látható

- Ha nincs kiírva, akkor private az öröklés.
- Példa

```
class A {
  private: int a;
  protected: int b;
  public: int c;
   void f1() { a = 1; b = 1; c = 1; }
};

class B : public A {
  void f2() {
    //a=1  //Nem latszodik
    b = 2;
    c = 3;
  }
};
```

```
void f3() {
        A o1;
        //o1.a = 1; Nem latszodik
        //o1.b = 1; Nem latszodik, protected
        o1.c = 1; //OK

        B o2;
        //o2.a = 1; Nem latszodik
        //o2.b = 1; Nem latszodik,
        o2.c = 1; //OK ha publikus az orokles
}
```

A konstruktorok helyzete

- A konstruktor nem öröklődik, meg kell írni.
  - Mi legyen a leszármazott konstruktorában? Ld. a Person és Employee esetében előkerült dilemmát.
- Ehhez meg kell értenünk, hogyan konstruálódik egy öröklött objektum, mik a hívási sorrendek.
- Nézzünk egy példát.

```
class Seged {
   double segedAdat;
  public:
   Seged() { segedAdat = 1.0; }
};

class Tartalmazott {
   int tartalmazottAdat;
  public:
   Tartalmazott() { tartalmazottAdat = 0; }
   Tartalmazott(int param) { tartalmazottAdat = param; }
};
```

```
class Szulo {
   Seged seged;
   int szuloAdat;
 public:
   Szulo() { szuloAdat = 0; }
   Szulo(int param) { szuloAdat = param; }
};

class Leszarmazott : public Szulo {
   Tartalmazott adat;
  public:
   Leszarmazott(int szuloAdatParam, int tartalmazottAdatParam)
   {???  Mi keruljon ide?}
};
```

A sorrend egy Leszarmazott létrehozásánál:

1. Területfoglalás az ősosztály, *Szulo* részeinek. Ha *Szulo* tartalmaz objektumokat, azok inicializálása, konstruktorainak hívása (*Seged* konstruktora)
2. *Szulo* konstruktorának hívása
3. Területfoglalás *Leszarmazott* részeinek. Ha *Leszarmazott* tartalmaz objektumokat, azok inicializálása, konstruktorainak hívása (*Tartalmazott* konstruktora)
4. *Leszarmazott* konstruktorának hívása

Problémák:

- Ha nem jelölünk ki Szülő konstruktort a leszármazott konstruktorában, akkor a folyamat során az ősosztály default konstruktora hívódik meg
  - Hogyan hívhatunk egy leszármazottban az ősnek egy nem default konstruktorát?
- Példa

Példa rossz ötletekre

```
class Leszarmazott : public Szulo {
   Tartalmazott adat;
 public:
   Leszarmazott(int szuloAdatParam, int tartalmazottAdatParam) {
      //szuloAdat = szuloAdatParam; Nem jo, mert privat
      //Butasag, mar elkeszult default konstruktorral:
      adat = Tartalmazott(tartalmazottAdatParam);
      //Ososztaly konstruktorahoz otlet:
      //Butasag: keszitunk egy Szulo objektumot es kidobjuk.
      Szulo(szuloAdatParam);
   }
};
```

Megoldás: inicializációs lista

```
class Leszarmazott : public Szulo {
   Tartalmazott adat;
 public:
   Leszarmazott(int szuloAdatParam, int tartalmazottAdatParam)
       : Szulo(szuloAdatParam), adat(tartalmazottAdatParam)
   {}
};
```

Szabályok az inic. listához

- Csak a közvetlen szülő konstruktorát lehet hívni
- Ha a szülőnek nincs def. konstruktora, akkor muszáj itt meghívni egy megfelelő konstruktort
- Az inic. listában a sorrend tetszőleges
  - A tagváltozók az osztálydeklarációban való felsorolásuknak a sorrendjében jönnek létre

Vissza az Employee konstruktorához

```
Employee(string pname, int pBirthYear, int pEmploymentYear)
      : Person(pname, pBirthYear), employmentYear(pEmploymentYear)
 {}
```

To be continued…