ÖRÖKLŐDÉS

Készítette: Vastag Atila
2017

Az Objektum Orientált Programozás egyik leghasznosabb része az örököltetés.

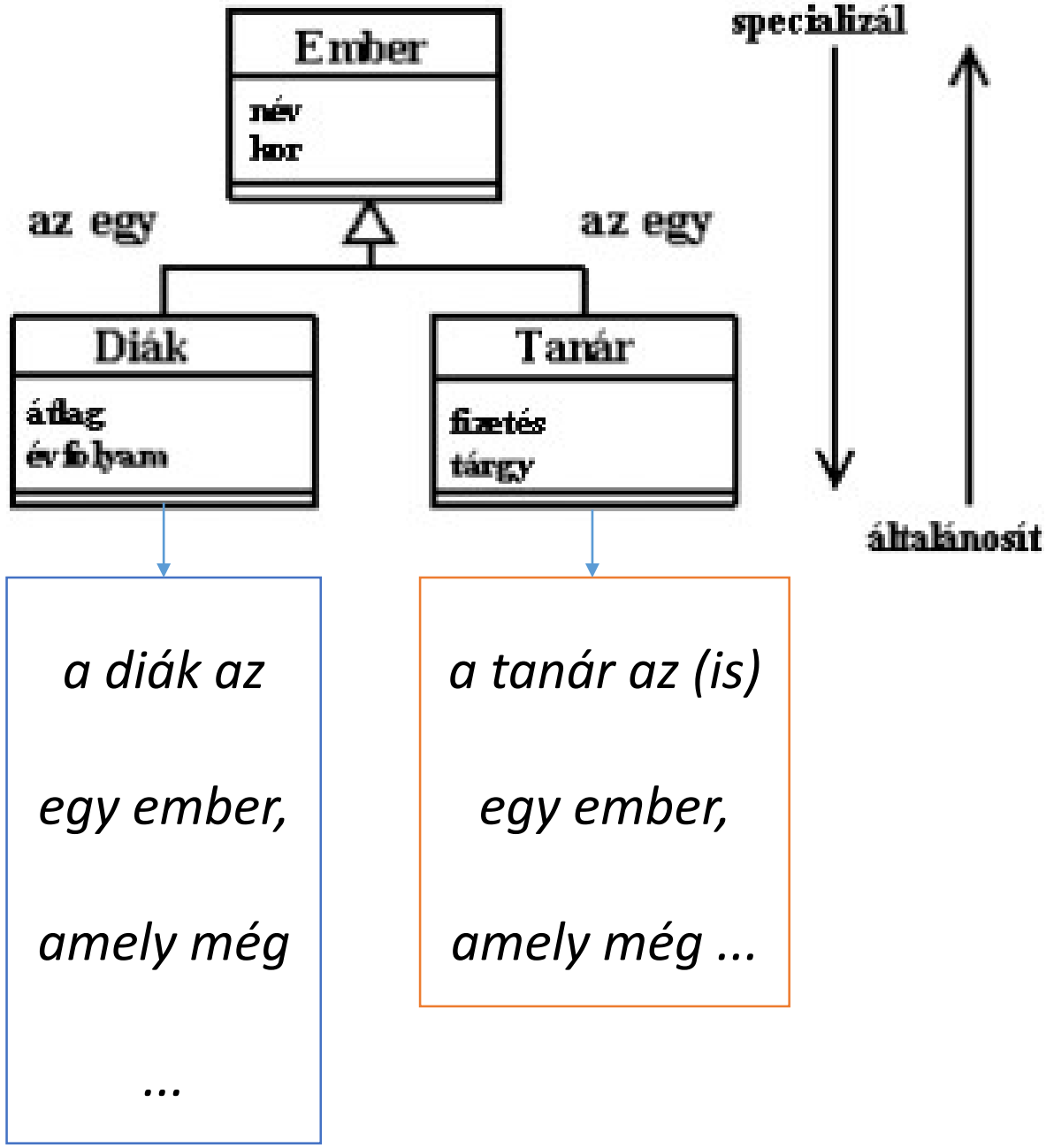
Sokszor találkozhatunk olyan, valós problémával, hogy egy objektumot néhány tulajdonság teljesen jól leír, de egy bonyolultabb változatához szükség volna még pár jellemzőre. Ezt az öröklődés nélkül úgy tenné az ember, hogy létrehozza az egyszerűbb osztályt, majd átmásolja a kódot az összetett osztályba és mellé ír még pár sort.

Ez a megoldás már az OOP nyelvekben nem elfogadható, sőt komoly tudásbeli hiányokat feltételez.

Ember **Diák** **Tanár**

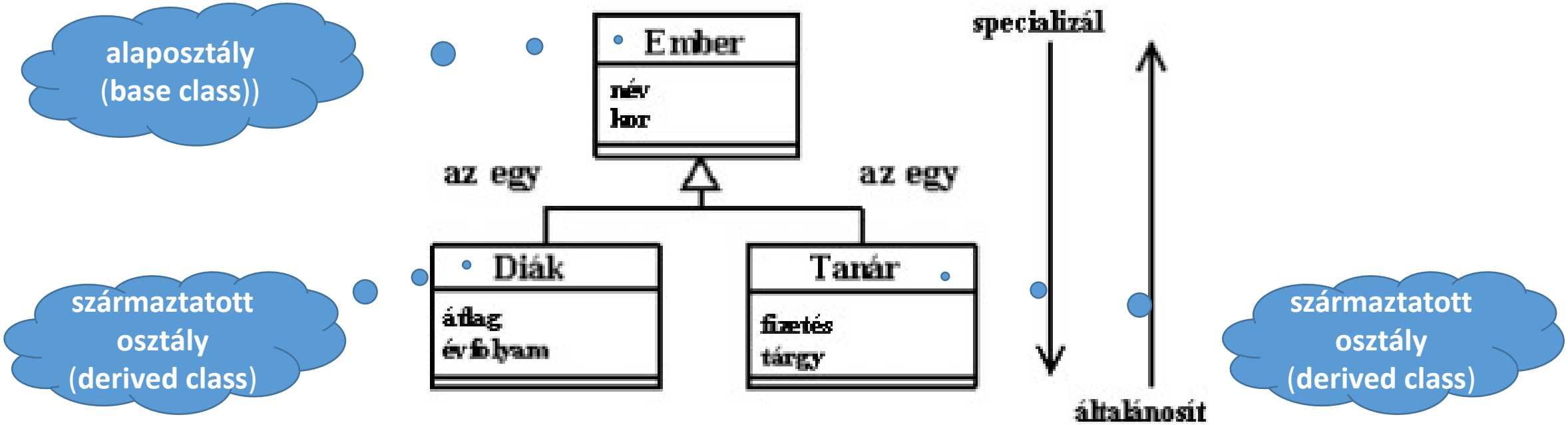
mi a közös bennük
(ha van közös – márpedig van)

Egy oktatási csoportban diákok és tanárok vannak. Közös tulajdonságuk, hogy mindnyájan emberek, azaz a diák és a tanár az ember speciális esetei, vagy fordítva az ember, legalábbis ebben a feladatban, a diák és tanár közös tulajdonságait kiemelő általánosító típus.



Az új igény, hogy **Diákot** és **Tanárt** is kezeljen a rendszerünk, új tulajdonságokkal, de úgy, hogy a korábbiak, amik az **Emnerre** jellemzőek, megmaradjanak. Így célunk, hogy a **Diáknak** legyen egy *átlag* és *évfolyam* tulajdonsága, a **Tanárn** esetében pedig az érdekel minket, hogy mekkora a *fizetése* és milyen *tárgyat* (próbal)tanítani.

Ahhoz, hogy ezt OOP-hez híven le tudjuk modellezni az ábra segítséget jelent.



Ha ezekkel az osztályokkal programot kívánunk készíteni, arra alapvetően két eltérő lehetőségünk van.

Az első lehetőség:
- 3 darab független osztályt hozunk létre, ahol az egyik az általános ember fogalomnak, a másik a tanárnak, míg a harmadik a diáknak felel meg. Sajnos ekkor az emberhez tartozó felelősségek, pontosabban a programozás szintjén a tagfüggvények, háromszor szerepelnek a programunkban.

A másik lehetőség:
- a közös rész kiemelése, melyet az **öröklődéssel** (**inheritance**) történő definíció tesz lehetővé. Ennek lépései:
• Ember definíciója. Ez az ún. **alaposztály** (**base class**). Ezt fogja örökölni a másik kettő.
• A diákot úgy definiáljuk, hogy megmondjuk, hogy az egy ember és csak az ezen felül lévő új dolgokat specifikáljuk külön: Diák = Ember + valami (adatok, műveletek)
• Hasonlóképpen járunk el a tanár megadásánál is. Miután tisztázzuk, hogy annak is az Ember az alapja, csak az tanár specialitásaival kell foglalkoznunk: Tanár = Ember + más valami

```
public class Ember
{
    public string Nev { get; set; }

    public int Kor { get; set; }
}
```

```
public class Diak : Ember
{
    public double Atlag { get; set; }

    public int Evfolyam { get; set; }
}
```

```
public class Tanar: Ember
{
    public double Fizetes { get; set; }

    public string Targy { get; set; }
}
```

C#-ban az öröklődést két osztály között a : segítségével jelöljük,

gyerek: szülő **analógia szerint.**

Öröklés szabályai:

- C#-ban egy osztály csak egyetlenegy osztálytól örökölhet (de örökölhet több *interface*-t, későbbi tananyag)

- Az öröklődés lehet több szintű is.

```
public class A
{
  // az osztaly
}
```

```
public class B : A
{
  // az osztaly
}
```

```
public class C : B
{
  // az osztaly
}
```

Ilyenkor azt mondjuk, hogy az `B` osztály rendelkezik minden tulajdonságával az `A` osztálynak, viszont a `C` osztály már rendelkezik az `A` `és` `B` osztály tulajdonságaival is, hisz a `B` osztály rendelkezik az `A` osztály tulajdonságaival is.

- A konstruktorok nem öröklődnek

– Van lehetőség meghívni az ősosztály konstruktorát a „`base`” kulcsszó segítségével (több ős konstruktor esetén a paraméterlista alapján dönt)

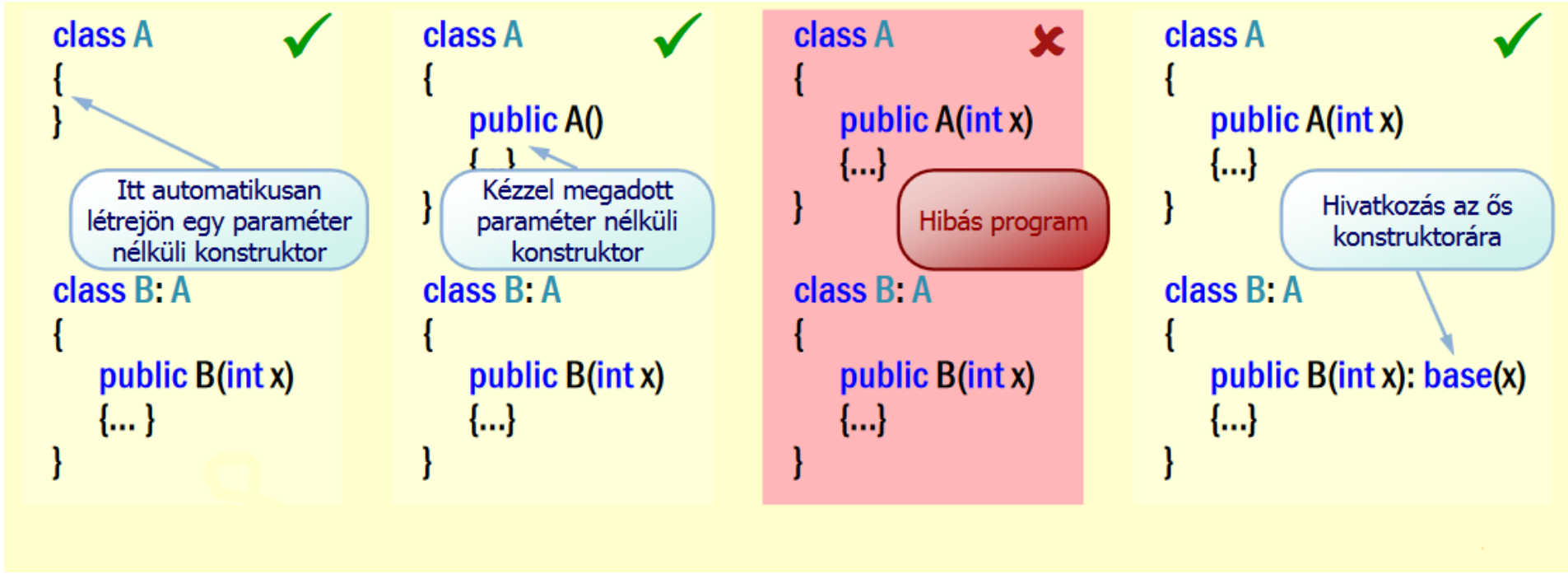
- A konstruktorok nem öröklődnek

– Van lehetőség meghívni az ősosztály konstruktorát a „base” kulcsszó segítségével (több ős konstruktor esetén a paraméterlista alapján dönt)

- Kötelező konstruktorhívás

– A leszármazottban kötelező meghívni az ős valamelyik konstruktorát

– Amennyiben nincs ilyen hívás, akkor az ős paraméter nélküli konstruktora automatikusan meghívódik (ha az ősnek nincs paraméter nélküli konstruktora, a fordító hibát jelez)



Az öröklődéssel történő megoldásnak számos előnye van:

- Hasonlóság kiaknázása miatt a végleges programunk egyszerűbb lehet. A felesleges redundanciák kiküszöbölése csökkentheti a programozási hibák számát. A fogalmi modell pontosabb visszatükrözése a programkódban világosabb programstruktúrát eredményezhet.

- Ha a későbbiekben kiderül, hogy a programunk egyes részein az osztályhoz tartozó objektumok működésén változtatni kell (például olyan tanárok is megjelennek, akik több tárgyat oktatnak), akkor a meglévő osztályokból származtathatunk új, módosított osztályokat. A származtatás átmenti az idáig elvégzett munkát anélkül, hogy egy osztály, vagy a program egyéb részeinek módosítása miatt a változtatások újabb hibákat ültetnének be programba.

Az öröklődéssel történő megoldásnak számos előnye van:

- Lényegében az előző biztonságos programmódosítás "ipari" változata az osztálykönyvtárak felhasználása. A tapasztalat azt mutatja, hogy egy könyvtári elem felhasználásának gyakori gátja az, hogy mindig "csak egy kicsivel" másként működő dologra van szükség mint ami rendelkezésre áll. A függvényekből álló hagyományos könyvtárak esetében ekkor meg is áll a tudomány. Az öröklődésnek köszönhetően az osztálykönyvtárak osztályainak a viselkedése rugalmasan szabályozható, így az osztálykönyvtárak a függvénykönyvtárakhoz képest sokkal sikeresebben alkalmazhatók. Ezen és a megelőző pontot összefoglalva kijelenthetjük, hogy az öröklődésnek, az analízis modelljének a pontos leképzésén túl egy fontos felhasználási területe a programelemek **újrafelhasználhatóságának** (**software reuse**) támogatása, ami az objektumorientált programozásnak egyik elismert előnye.