Opjektum Orientált Programozás

ÖRÖKLŐDÉS

Készítette: Vastag Atila

2020

Az Objektum Orientált Programozás egyik leghasznosabb része az örököltetés.

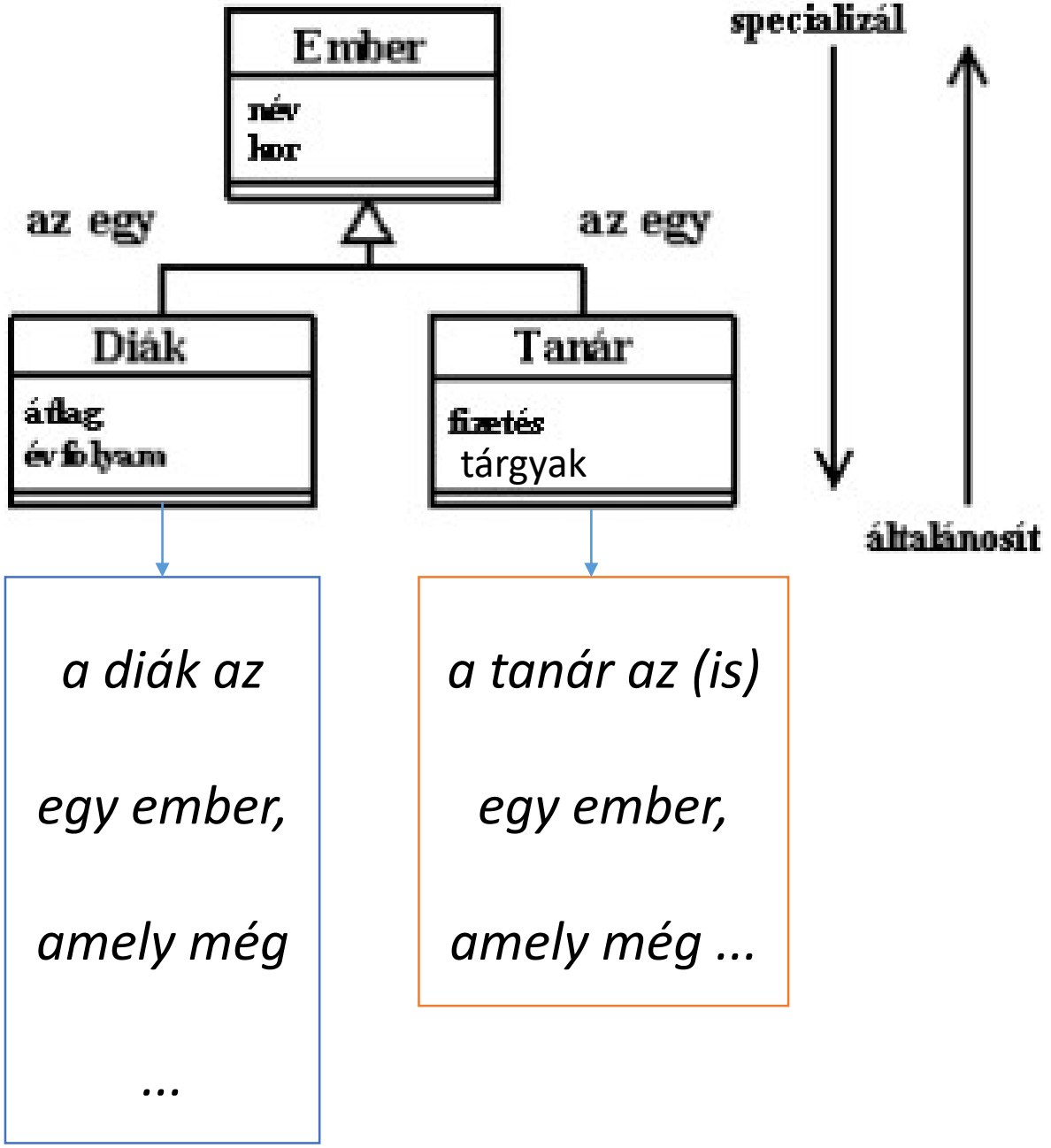
Sokszor találkozhatunk olyan, valós problémával, hogy egy objektumot néhány tulajdonság teljesen jól leír, de egy bonyolultabb változatához szükség volna még pár jellemzőre. Ezt az öröklődés nélkül úgy tenné az ember, hogy létrehozza az egyszerűbb osztályt, majd átmásolja a kódot az összetett osztályba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

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

Ember **Diák** **Tanár**

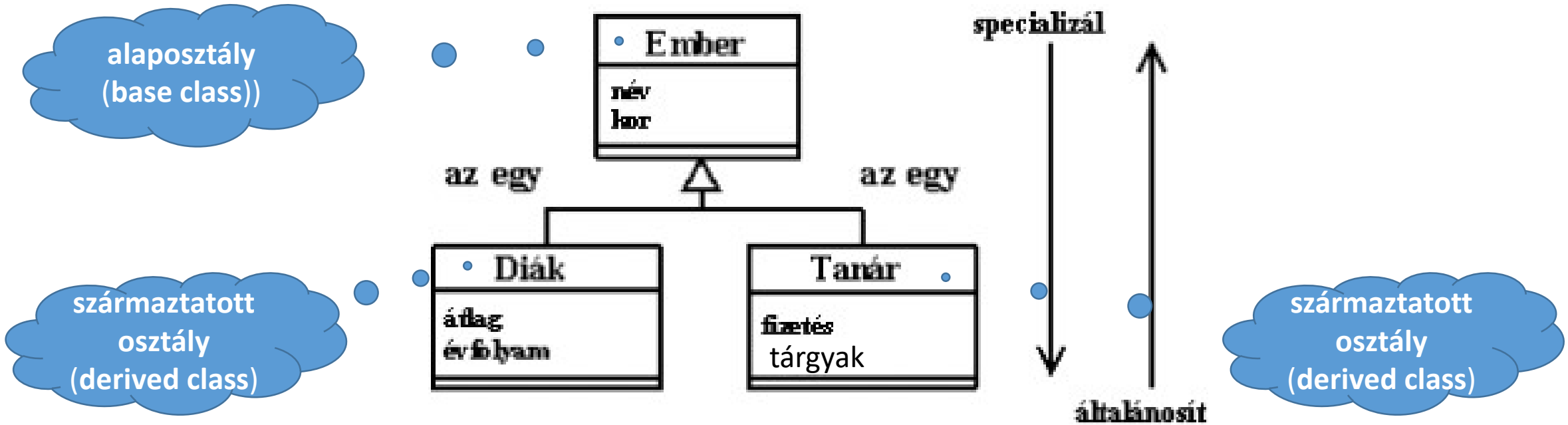
mi a közös bennük
(ha van közös – márpedig van)

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 (adatok, műveletek)

#emper.py

```python
class Ember:
    def __init__(self):
        self.Nev: str = ""
        self.kor: int = 0
```

#diak.py

```python
from ember import Ember

class Diak(Ember):
    def __init__(self):

        Ember.__init__(self)

        self.atlag: float = 0
        self.evfolyam: int = 0
```

ős osztály konstruktor meghívása

#tanar.py

```python
from typing import *

from ember import Ember

class Tanar(Ember):
    def __init__(self):

        Ember.__init__(self)

        self.fizetes: float = 0
        self.targyak: [List[str]] = []
```

main.py | ember.py | diak.py | tanar.py

main.py > ...

```python
from ember import Ember
from diak import Diak
from tanar import Tanar

ember:Ember = Ember()
diak: Diak = Diak()
tanar: Tanar = Tanar()

ember.kiVagyok()

diak.
tanar
```

- Nev
- atlag
- evfolyam
- kiVagyok
- kor
- mro
- __base__
- __bases__
- __class__
- __delattr__
- __dict__
- __dir__

main.py | ember.py | diak.py | tanar.py

main.py > ...

```python
from ember import Ember
from diak import Diak
from tanar import Tanar

ember:Ember = Ember()
diak: Diak = Diak()
tanar: Tanar = Tanar()

ember.kiVagyok()

diak.kiVagyok()

tanar.
```

- Nev
- fizetes
- kiVagyok
- kor
- mro
- targyak
- __base__
- __bases__
- __class__
- __delattr__
- __dict__
- __dir__

Tudjuk, hogy az osztályok tartalmazhatnak függvényekete is. De mi van akkor, ha az ős osztályban lévő függvény nem felel meg a céljainknak, vagyis az öröklött osztályban másként használnánk?

```
class Ember:
    def __init__(self):
        self.Nev: str = ""
        self.kor: int = 0

    def kiVagyok(self) -> None:
        print(f"Ember vagyok!")
```

Mint a példákon is látható *(**kiVagyok** függvény)*, ilyenkor a Python egyszerűen felüldefiniálja az öröklött osztályban **(override)**.

```
from ember import Ember

class Diak(Ember):
    def __init__(self):

        Ember.__init__(self)

        self.atlag: float = 0
        self.evfolyam: int = 0

    def kiVagyok(self) -> None:
        print(f"Diak vagyok!")
```

```
from typing import *
from ember import Ember

class Tanar(Ember):
    def __init__(self):

        Ember.__init__(self)

        self.fizetes: float = 0
        self.targyak: [List[str]] = []

    def kiVagyok(self) -> None:
        print(f"Tanar vagyok!")
```

```python
#main.py

from ember import Ember
from diak import Diak
from tanar import Tanar

ember:Ember = Ember()
diak: Diak = Diak()
tanar: Tanar = Tanar()

ember.kiVagyok()

diak.kiVagyok()

tanar.kiVagyok()
```

```
#output

e:/Desktop/Python/main.py
Ember vagyok!
Diak vagyok!
Tanar vagyok!
```

Öröklés szabályai:

- Az **öröklés**, de főleg a **többszörös öröklés (multiple inheritance)** kényes téma a Python-ban, annál is inkább, mivel forradalmi változások zajlanak az objektumok táján, és ezek a kódokra óriási hatással lesznek. Mindenesetre annyit el lehet mondani, hogy a Python a többszörös öröklődés útját követi mint a c++ (nem pedig a c#-ban és Java-ban használatos interface-re épülő megoldásokat).

Az öröklődés lehet több szintű is.

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

```
class B(A)
{
  // az osztaly
}
```

```
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, de lehetőség van meghívni az ősosztály konstruktorát a „*{öröklött osztály}*.`__init__(self)`” utasítás segítségével. Ha az ős osztály konstruktora paramétert vár, azt meg kell adni)