Objektumok kapcsolatai

Gregorics Tibor

gt@inf.elte.hu

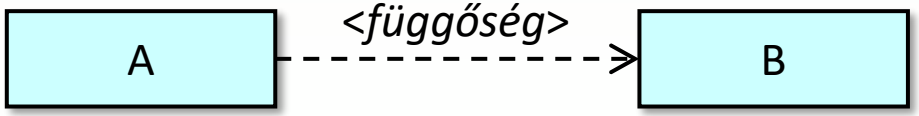
http://people.inf.elte.hu/gt/oep

Objektum-kapcsolatok fajtái

- Amikor objektumok egymással kommunikálnak (szinkron vagy aszinkron módon egymás metódusait hívják, egyik a másiknak szignált küld, esetleg közvetlenül a másik adattagjain végeznek műveletet), vagy csak hasonlítanak egymásra, akkor kapcsolat alakul ki közöttük.
- A kapcsolat fajtája lehet:
  - Függőség (*dependency*)
  - Asszociáció (*association*) vagy társítás
  - Aggregáció (*shared aggregation*) vagy tartalmazás
  - Kompozíció (*composite aggregation*) vagy szigorú tartalmazás
  - Származtatás vagy öröklődés (*inheritence*)

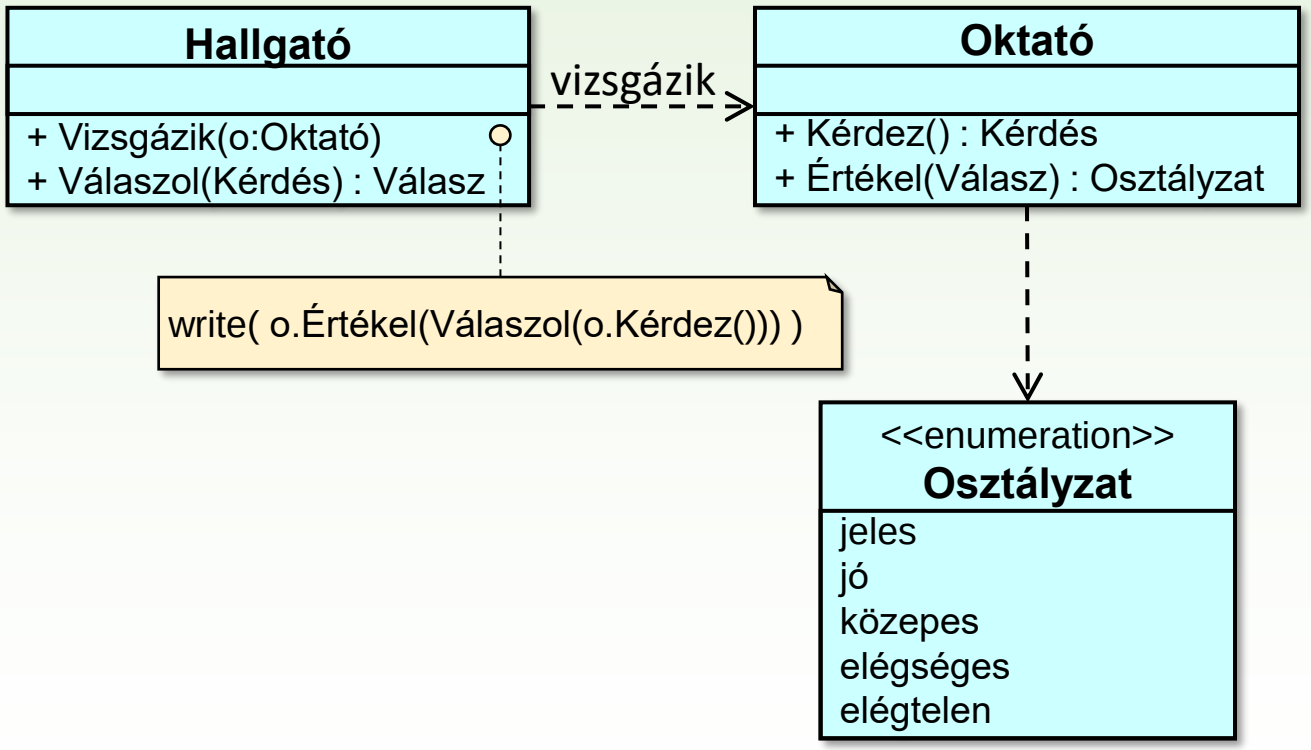
Függőség

- Az A osztály függ a B osztálytól, ha
  - a két osztály egy-egy objektuma ideiglenesen (rövid időre) kerül kapcsolatba úgy, hogy az A osztály egy objektumának egy metódusa vagy paraméterként kapja meg, vagy helyben példányosítja a B osztály egy objektumát abból a célból, hogy
    - meghívja annak egy metódusát, vagy
    - szignált (üzenetet) küldjön neki, vagy
    - továbbadja a hivatkozását (visszatérési értékként vagy akár kivétel-dobásba ágyazva).
  - az A osztály egyik osztályszintű metódusa a B osztály egy osztályszintű metódusát hívja. (nem objektumok közötti kapcsolat)
  - az A osztály egy objektuma használja egy típusértékét a B osztállyal leírt típusnak, ami lehet az A osztály egy objektumához tartozó
    - adattagnak az értéke
    - metódusnak a visszatérési értéke



Példa függőségekre

- Egy hallgató vizsgázik egy oktatónál, aki kérdést tesz fel neki, majd értékeli a válaszát.



1.rész
Asszociáció

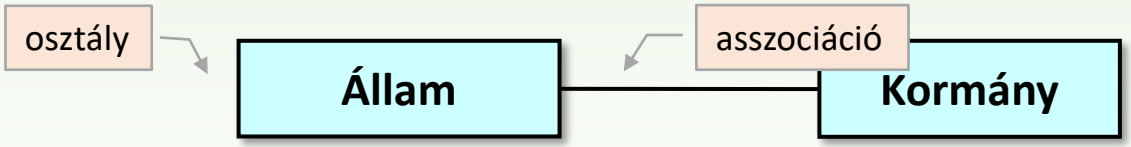
Gregorics Tibor

gt@inf.elte.hu

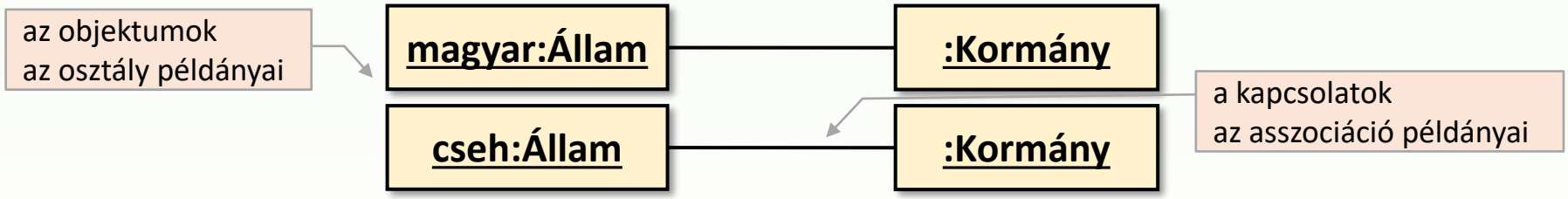
http://people.inf.elte.hu/gt/oep

Asszociáció

- Az asszociáció az objektumok között hosszabb időszakon keresztül fennálló kapcsolat, amely mentén az objektumok sorozatosan küldenek egymásnak üzeneteket.
- Matematikai értelemben az asszociáció egy reláció az adott osztályok példányait (objektumait) tartalmazó halmazok direkt szorzatán:
  - egy asszociáció több objektum-kapcsolatot ír le, és ugyanaz az objektum egy asszociáció több kapcsolatában is szerepelhet, de megjelenhet különböző asszociációk kapcsolataiban is.

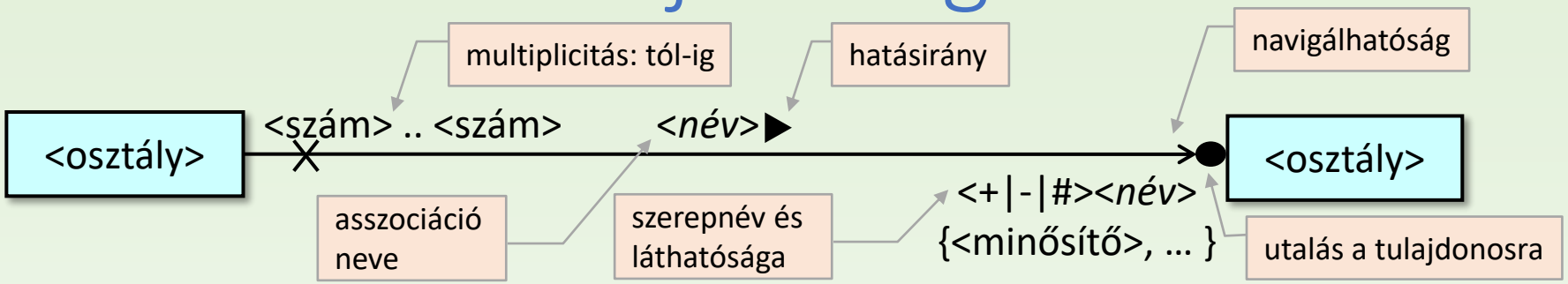


Az osztálydiagram egy lehetséges példányosítása (felpopulálása):



4 Az objektum-orientált nyelvek ismérve a fogalmi szintű absztrakció. Az osztály az objektumok, az asszociáció a kapcsolatok absztrakt leírására szolgál.

Asszociáció tulajdonságai



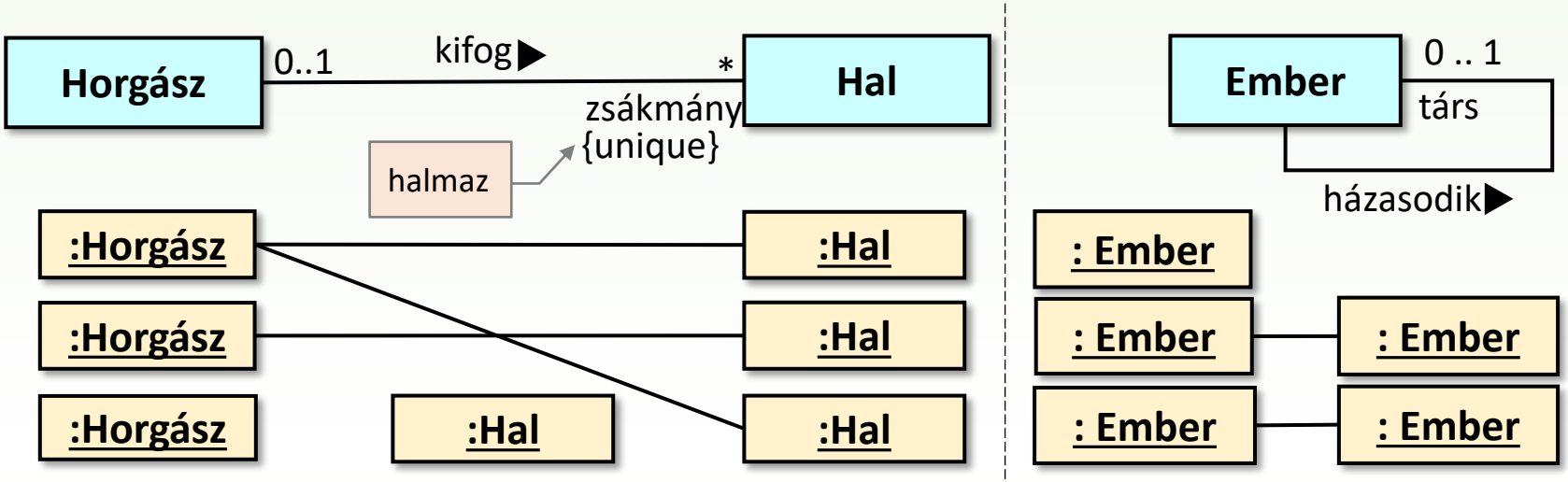
- Az asszociáció tulajdonságai az általa leírt kapcsolatokat jellemzik:
  - név: a kapcsolatok közös megnevezése
  - hatásirány: kapcsolódó objektumok közti fogalmi viszony
  - multiplicitás: egy objektumhoz kapcsolható objektumok száma
  - aritás: az egyes kapcsolatokban résztvevő objektumok száma
  - navigálhatóság: a kapcsolatban melyik objektumot kell gyorsan elérni
  - asszociációvég nevek: a kapcsolatban álló objektumok hivatkozási nevei
  - asszociációvégek neveinek láthatósága
  - asszociációvégek neveinek tulajdonosa
  - asszociációvég nevével jellemzett gyűjtemény minősítése

Nevek és a hatásirány

- Egy objektumelvű modell asszociációi és a megoldandó feladat szöveges leírásának mondatai között szoros kapcsolat áll fenn. Ez alapján az asszociációnak, és annak kapcsolataiban megjelenő objektumoknak neveket lehet adni:
  - az asszociáció neve a mondat állítmánya (néha tárgya), amely gyakran az érintett osztályok egyikének metódusneve is.
  - a mondat többi (nem állítmány, nem jelző) szavai adják az asszociáció-végek neveit (szerepnevek), amelyek arra szolgálnak, hogy a kapcsolatban levő objektumok egymásra hivatkozhassanak.
- A bináris (két objektum kapcsolatát leíró) asszociációk neve mellé rajzolt fekete háromszög hegye az asszociáció hatásiránya
  - ez mindig az asszociációt jellemző mondat alanyát adó objektum felől mutat a másik irányába (ami sokszor a mondat tárgya).

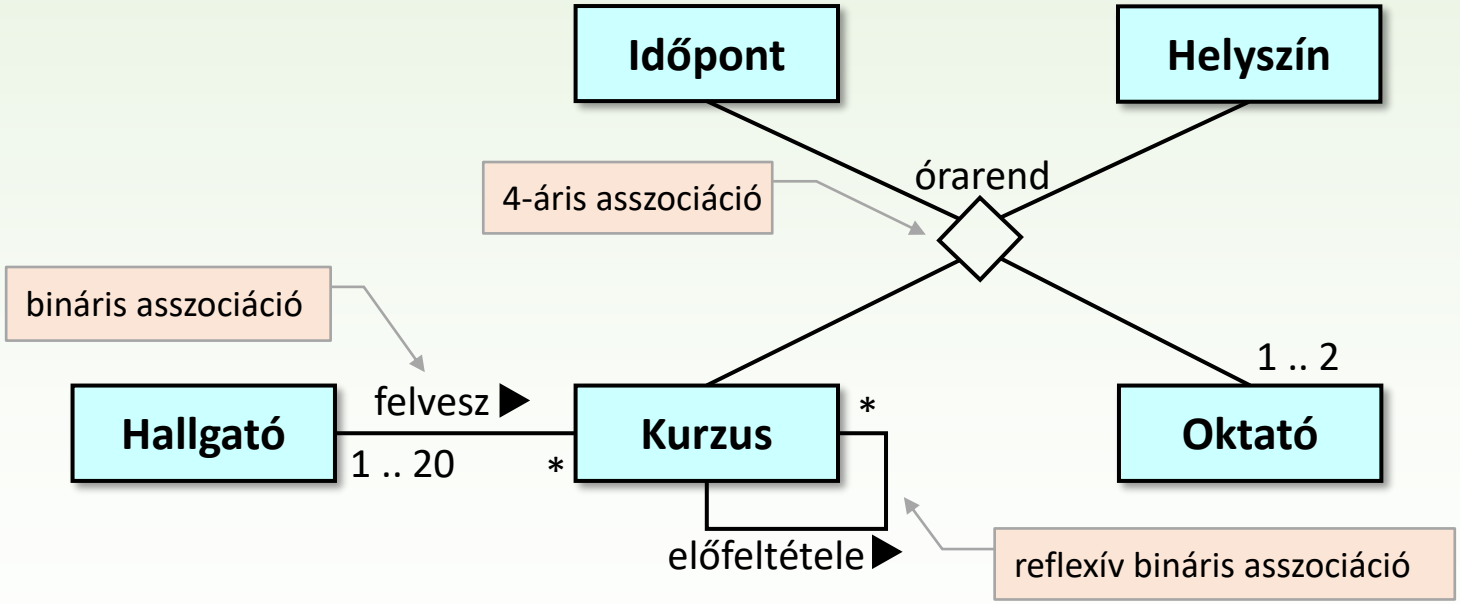
Multiplicitás

- Az asszociáció multiplicitása azt mutatja, hogy az asszociáció alapján egy objektum hány másik objektummal létesíthet egyszerre kapcsolatot. Ez lehet egy természetes szám (alapértelmezés szerint 1), vagy természetes számok min..max intervalluma. (Ha a multiplicitás szám helyén * áll, akkor az az adott érték tetszőleges voltára utal.)
- Előírható, hogy egy adott asszociációban egy objektumhoz kapcsolódó objektumok
  - mind különbözzenek egymástól {unique}
  - megadott sorrendben legyenek felsorolhatók {ordered}



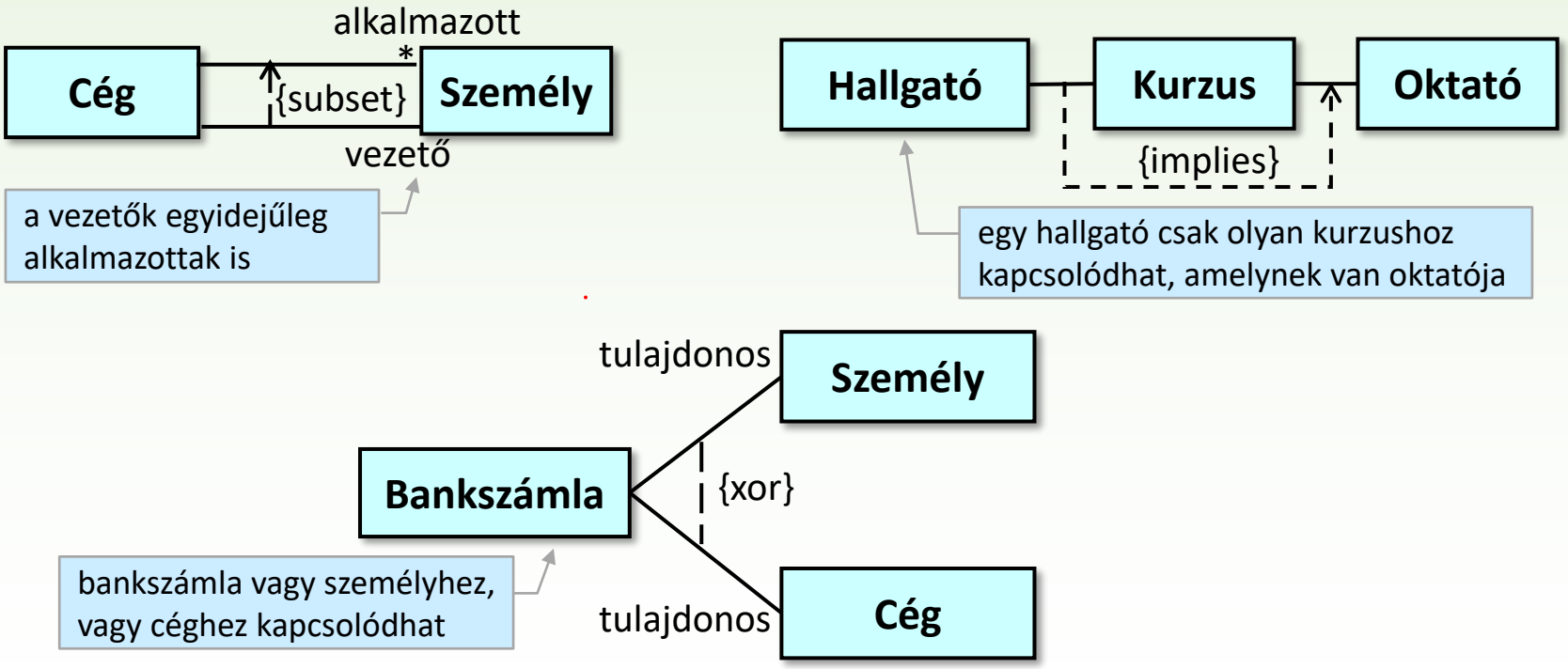
Aritás

- Az asszociáció aritása arra utal, hogy az asszociáció egyetlen kapcsolata hány objektumot köthet össze. (Eddig csak bináris asszociációkra láttunk példákat, ahol egy kapcsolat mindig két objektum között jött létre.)



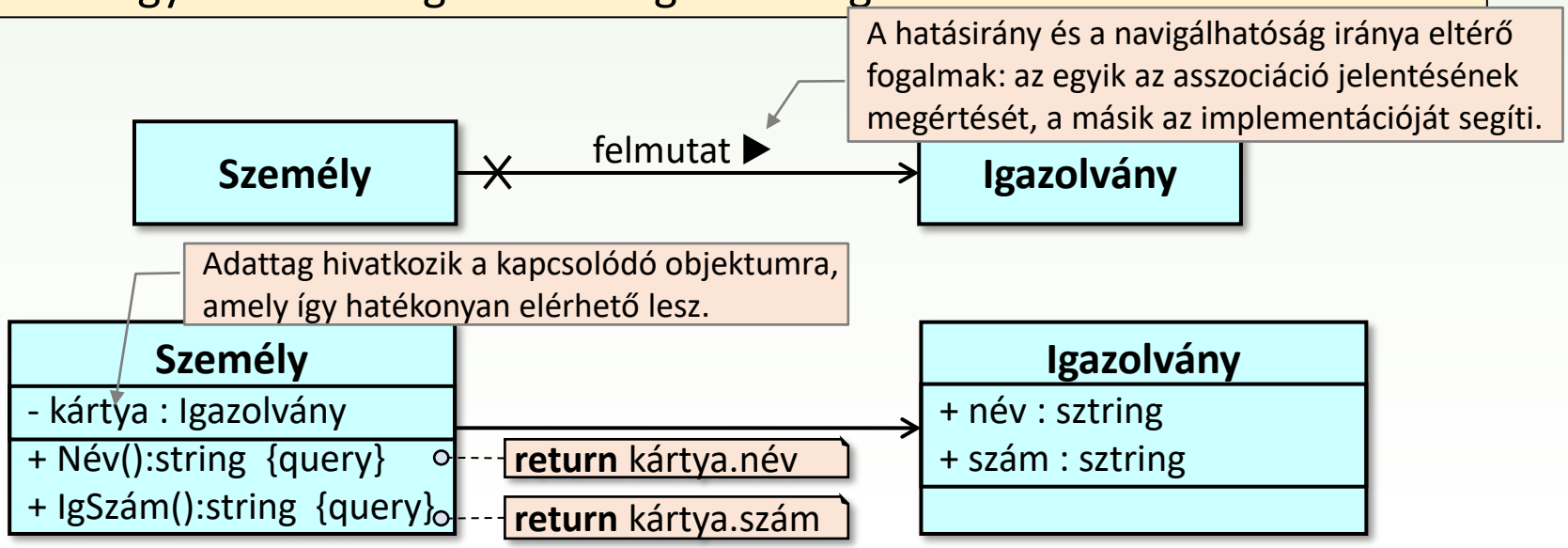
Asszociációk közti megszorítások

- Megadhatunk az asszociációk között logikai feltételeket (subset, and, or, xor, implies, …), amelyek a megadott asszociációk kapcsolatai közötti korlátozásokat fogalmazzák meg.



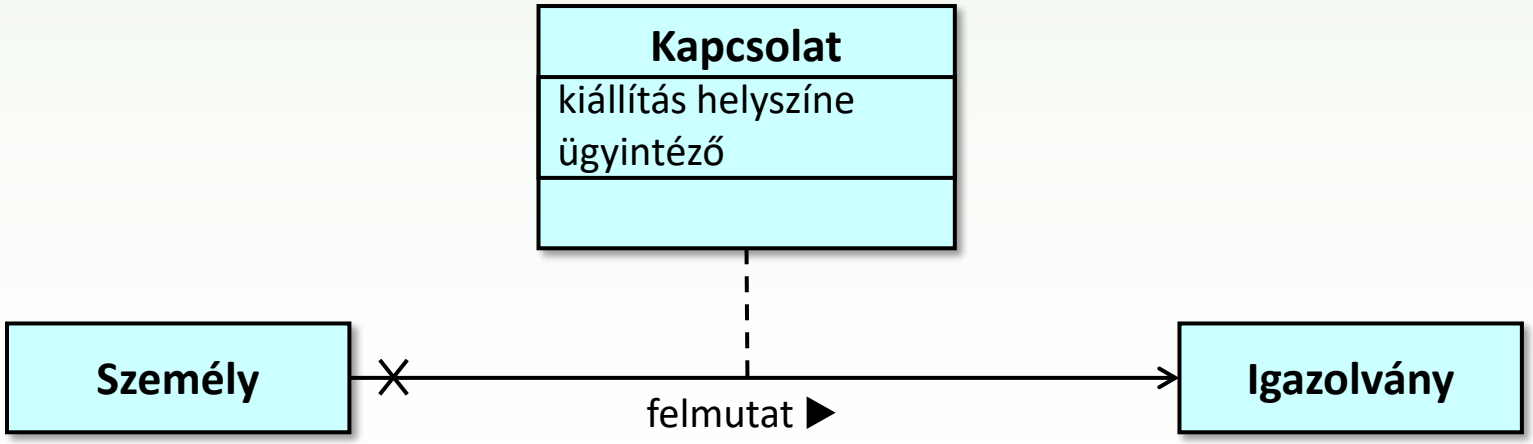
Navigálhatóság

- A navigálhatóság azt jelzi, hogy egy asszociáció kapcsolataiban melyik osztály objektumát kell a kapcsolat másik (többi) objektumának hatékonyan elérni.
  - A nyíl az asszociáció végén hatékonyan elérendő objektum osztályára mutat.
  - Az „×” jel az asszociáció végén azon osztály mellett van, amely objektumát a kapcsolatokban nem kell (hatékonyan) elérni.
  - A jelöletlen asszociáció vég arra utal, hogy ott még nincs eldöntve, hogy kell-e támogatni a navigálhatóságoz.



Asszociációs osztály

- Egy kapcsolatra olyan önálló egységekként tekinthetünk, amely tárolja az összekötött objektumok hivatkozásait, emellett rendelkezhet egyéb adattagokkal (sőt akár metódusokkal is). Egy ilyen asszociáció kapcsolatainak leírására szolgál az asszociációs osztály.
- A programozási nyelvek többsége azonban nem kezeli önálló entitásként a kapcsolatokat, nem foglal le számukra külön memóriaterületet, ezért az asszociációs osztály fogalmát sem ismeri. Így ennek megvalósítása igazi kihívás.



2.rész
Asszociációk részletezése egy megvalósítás szintű osztály diagramban

Gregorics Tibor

gt@inf.elte.hu

http://people.inf.elte.hu/gt/oep

Szerepnév tulajdonosa

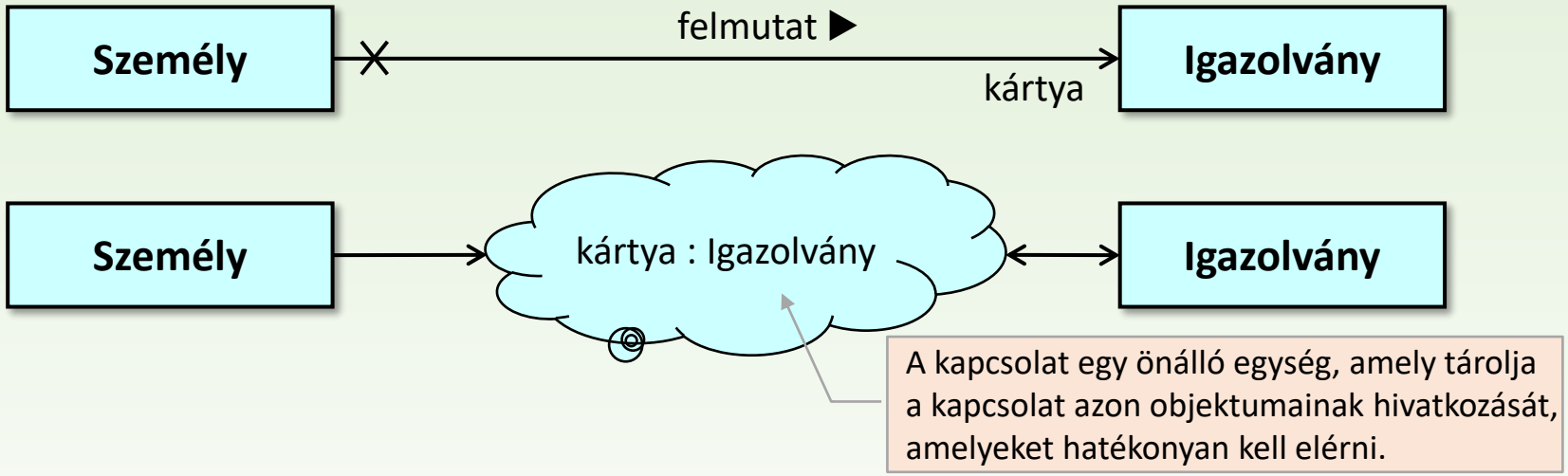
- Láttuk, hogy a szerepnevek bevezetése egyfelől a modellezés fogalomalkotási folyamatában hasznosak, másfelől ezek segítségével tudunk egy kapcsolatban az objektumokra hivatkozni.
- Ha egy kapcsolatban egy objektumhoz hatékonyan kell elnavigálni, akkor ismerni kell az objektum hivatkozását. Ezt az adott szerepnévvel azonosítva tároljuk. De hol? Ki legyen a szerepnév tulajdonosa?
  - maga a kapcsolat, feltéve, hogy a kapcsolat olyan önálló tárhellyel rendelkezik, amelyet a kapcsolat objektumai alapértelmezés szerint elérnek. A szerepnév ilyenkor az asszociációt leíró asszociációs osztály adattagjaként jelenik meg.
  - a kapcsolat azon másik (többi) objektuma, ahonnan hatékonyan kell elérni a szerepnévvel jelölt objektumot. A szerepnév ilyenkor ezen másik objektumnak lesz az adattagja, ott foglal tárhelyet.

ez az elterjedtebb

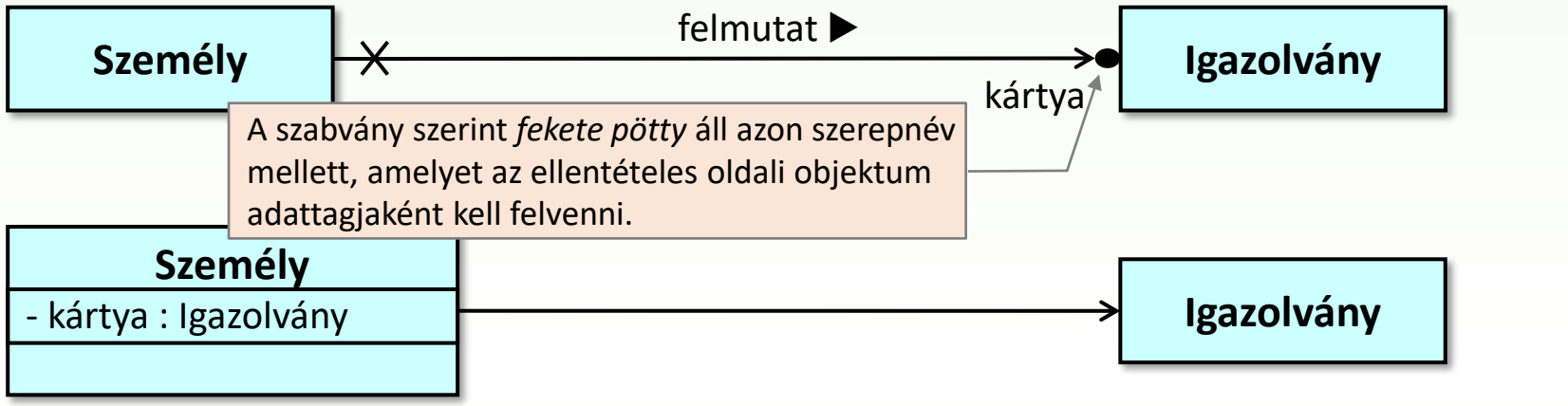
- A szerepnevek láthatóságát az adattagokéhoz hasonló módon lehet szabályozni.

Szerepnév tulajdonosának jelölése

Maga a kapcsolat a szerepnév tulajdonosa

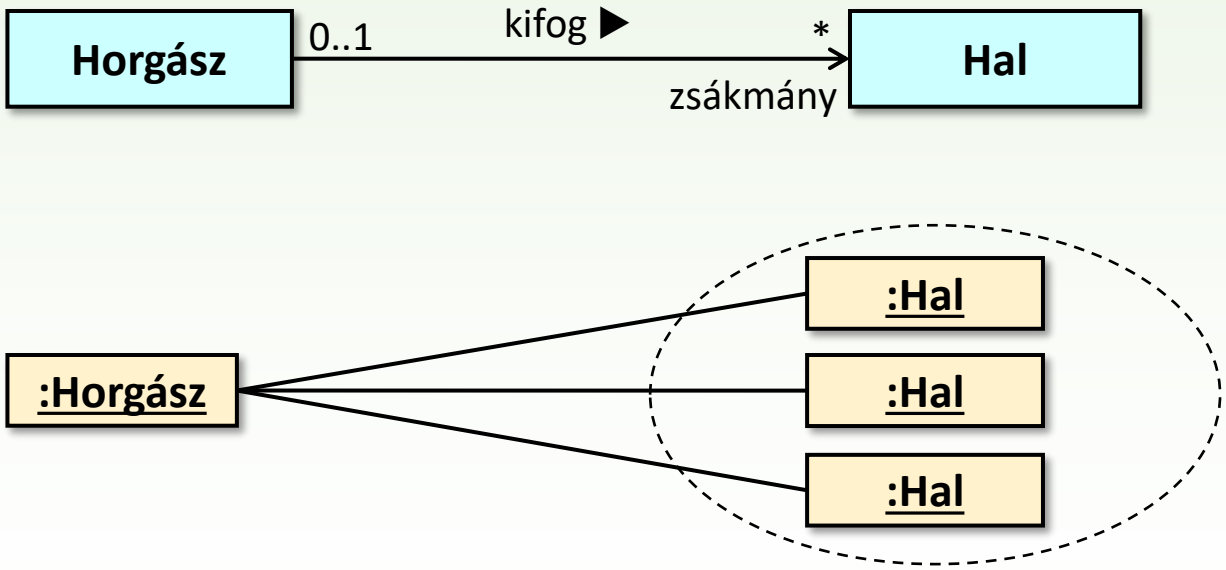


A kapcsolat másik objektuma a szerepnév tulajdonosa



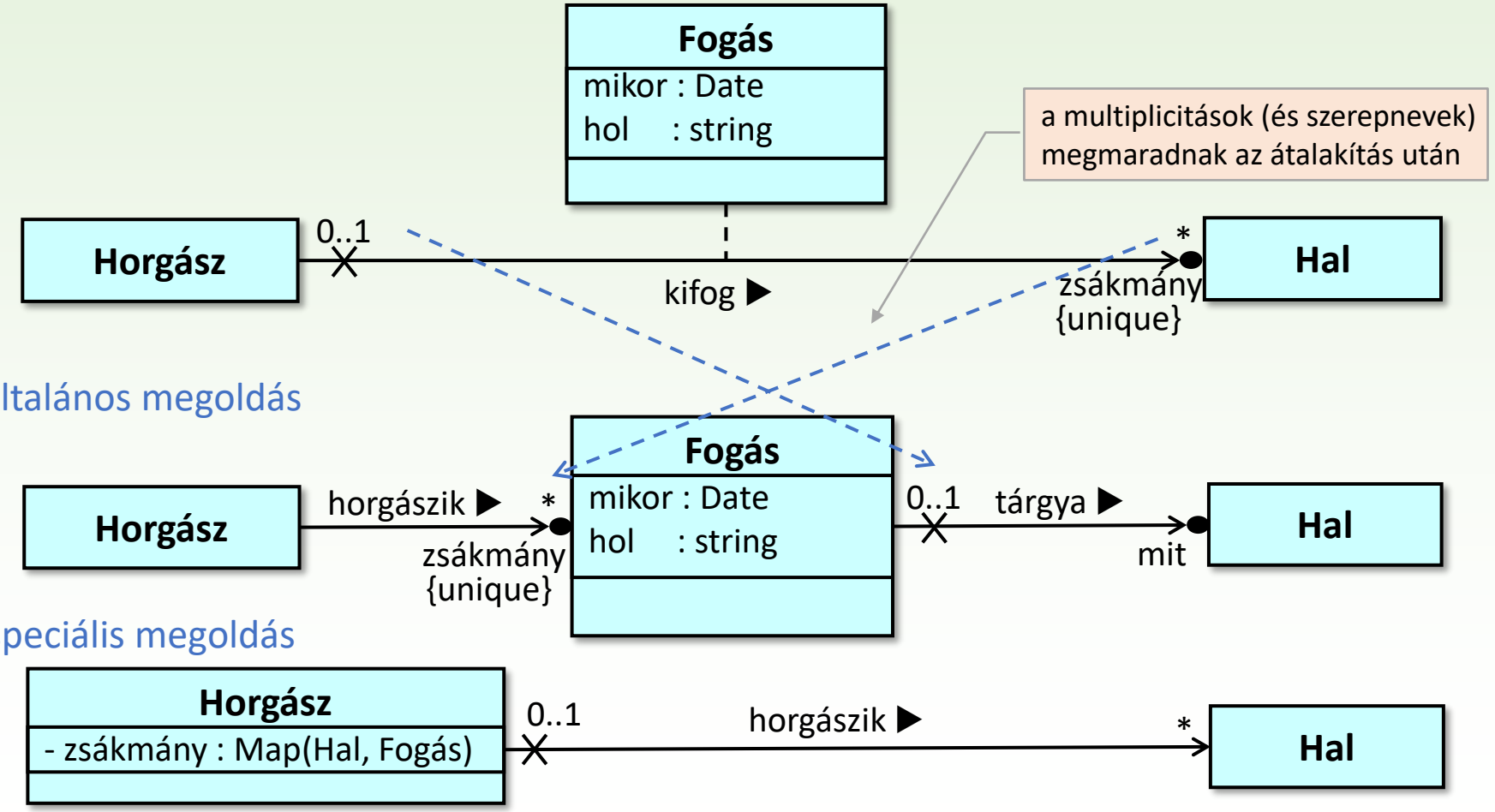
Gyűjteményt jelölő szerepnév

- Amikor egy asszociáció egyik szerepneve mellett többszörös multiplicitás látható, akkor a szerepnév az általa jelzett osztály azon objektumainak a gyűjteményét azonosítja, amelyek egy adott asszociációban a másik (többi) objektumhoz kapcsolódnak.



Asszociációs osztály kiküszöbölése

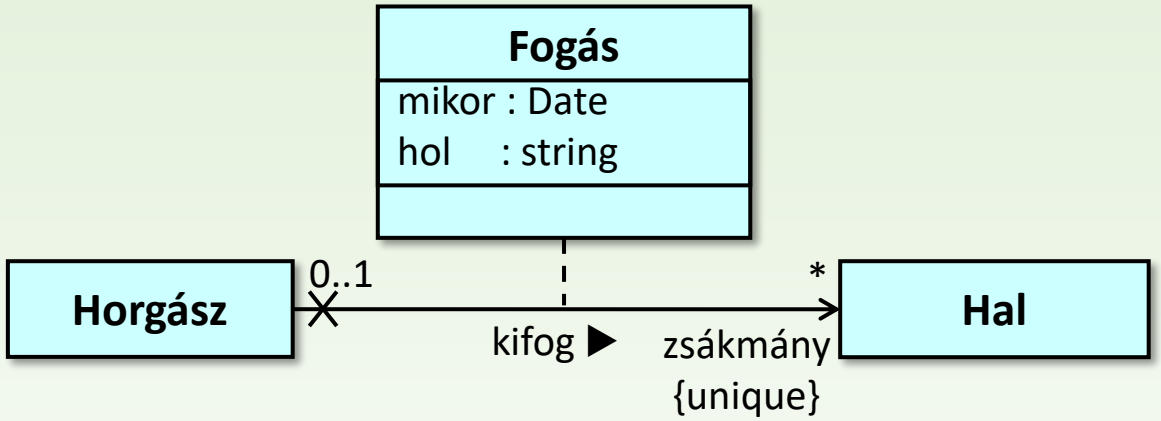
- Ha egy programozási nyelv nem támogatja az asszociációs osztályok kezelését, akkor a kapcsolatoknak nem lesz önálló tárhelye, kivéve, ha az asszociációs osztályt közönséges osztállyal nem helyettesítjük.



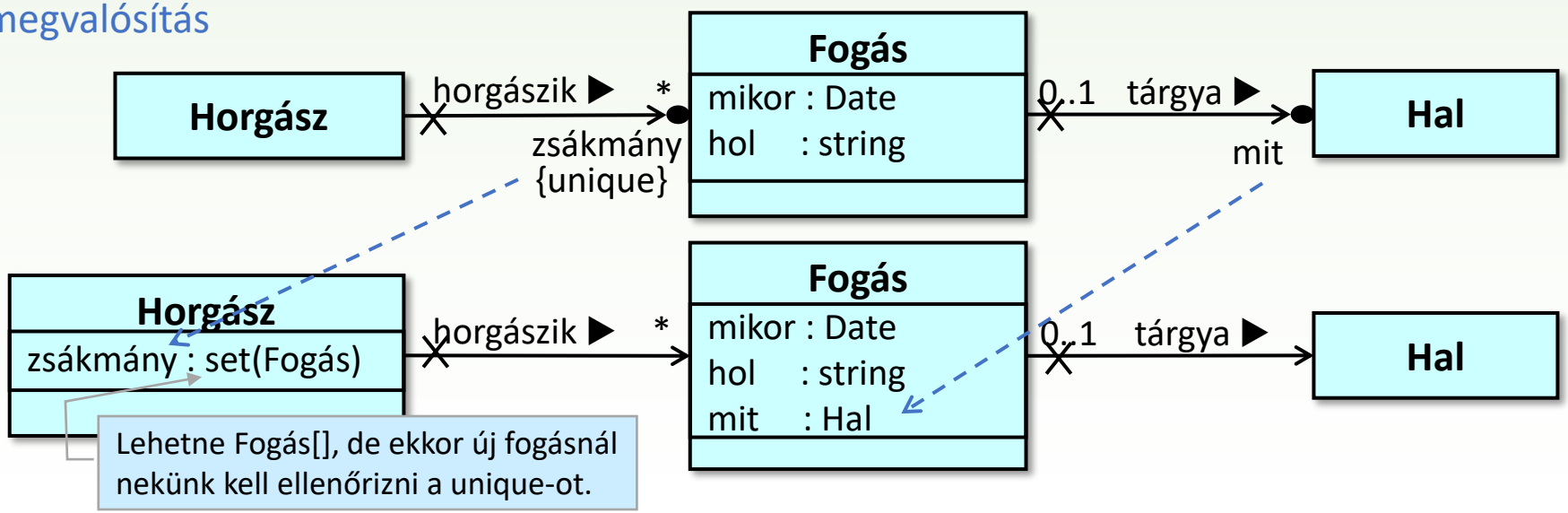
Modellezési szintek

elemzés

tervezés



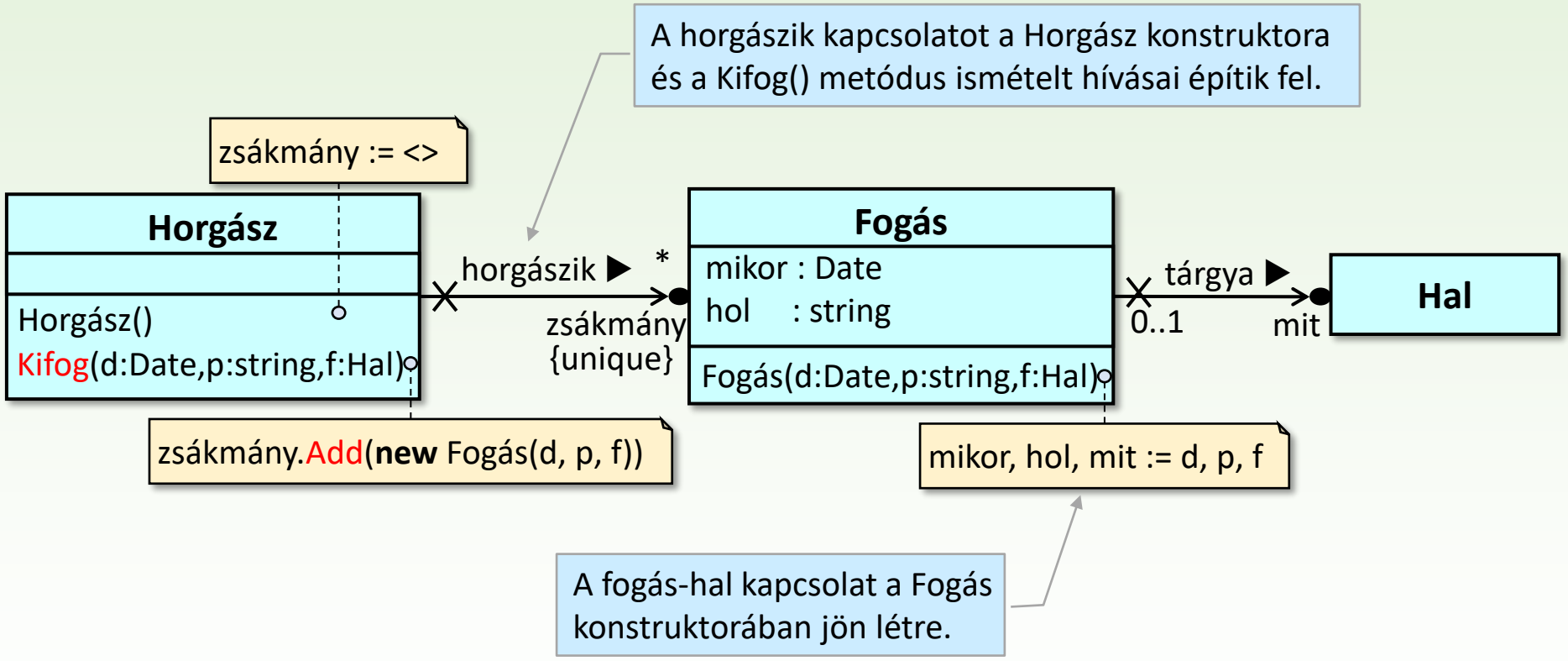
megvalósítás



Asszociáció kapcsolatainak létrehozása

- Egy asszociációhoz mindig társul annak kapcsolatait létrehozó metódus, amely számos esetben egy konstruktor.
- Egy kapcsolatot létrehozó metódus vagy maga hozza létre az összekapcsolandó objektumokat, vagy paraméterként kapja meg azok (egy részének) hivatkozását, és ezeket a megfelelő szerepneveknek adja értékül.
- A kapcsolat létrehozáskor ügyelni kell:
  - az asszociáció multiplicitásaiból adódó korlátokra
  - a szerepnevek korlátozásaira (pl. unique)
  - más asszociációkhoz való viszonyokra (pl. implies)

Példa kapcsolatok létrehozására



3.rész
Aggregáció és kompozíció

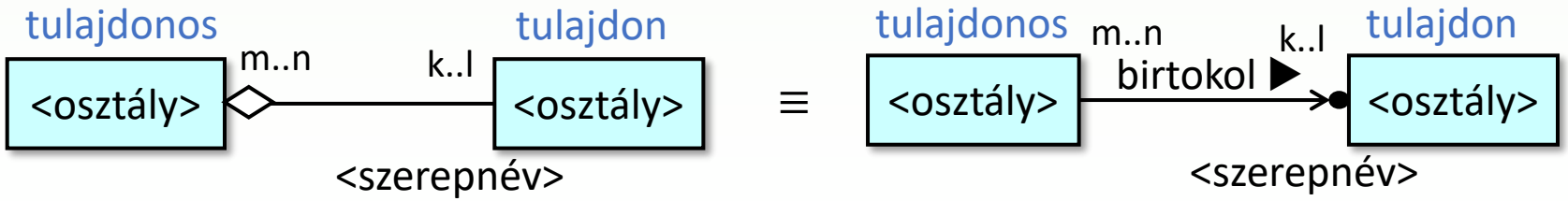
Gregorics Tibor

gt@inf.elte.hu

http://people.inf.elte.hu/gt/oep

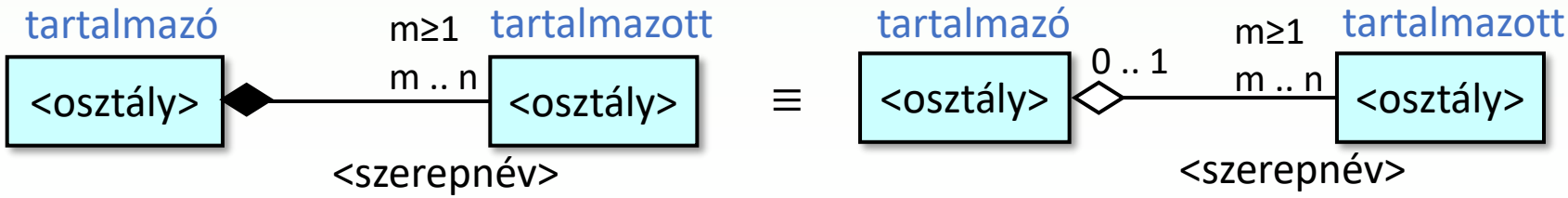
Aggregáció

- Birtoklási kapcsolat (tulajdonos-tulajdon viszony) kifejezésére szolgáló bináris asszociáció, ahol a kapcsolatok nem alkothatnak irányított kört (egy objektum még közvetett módon sem lehet önmaga tulajdona).
- Megengedi, hogy
  - egy tulajdonos objektumnak ne legyen tulajdon objektuma;
  - egy tulajdon objektum önmagában is létezzen
  - egy tulajdon objektum egyszerre több tulajdonos objektumhoz is tartozzon.
- A kapcsolat a tulajdon objektum irányába hatékonyan navigálható úgy, hogy a tulajdonos objektum tárolja el a tulajdonának szerepnevét.
- Az aggregációs kapcsolat létrehozása a tulajdonos objektum felelőssége: ennek egy metódusa (lehet a konstruktora is) helyezi el a tulajdon objektum hivatkozását a tulajdonos erre szolgáló adattagjában.

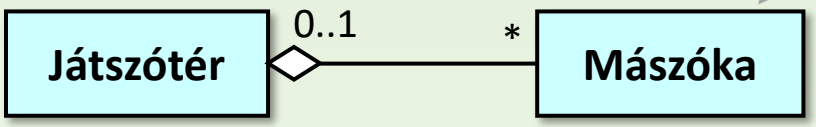


Kompozíció

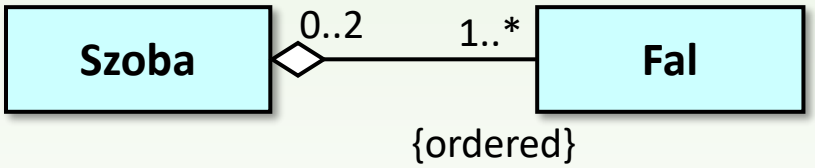
- Speciális aggregáció, ahol a tulajdonlást egy tartalmazó-tartalmazott (egész-rész) viszonynak tekintjük, és ahol
  - a tartalmazó objektum nem létezhet tartalmazott objektum nélkül,
  - a tartalmazott objektum egyszerre csak egy objektum része lehet.
- A kompozíció fogalmához sokan az alábbi, a tartalmazott objektumra tett egyre szigorodó megszorítások valamelyikét is hozzáértik:
  - van tartalmazója: a tartalmazott önmagában nem létezhet.
  - a tartalmazója nem változik: a tartalmazott objektumot a tartalmazó objektum metódusai példányosítják, és szűntetik meg.
  - élettartama azonos a tartalmazójáéval: a tartalmazott objektumot a tartalmazó objektum konstruktora példányosítja, destruktora törli.
- A tartalmazó objektum konstruktora építi fel a kapcsolatot, de a tartalmazott objektumot egy közönséges metódus lecserélheti másikra.



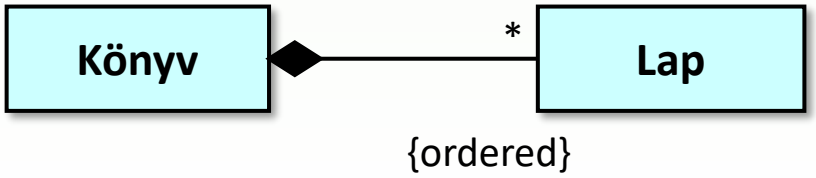
Példák: aggregáció vs. kompozíció



A mászókák a játszótér részei, de a játszótér mászóka nélkül is játszótér marad. (A döntést zavarhatja, hogy egy mászóka egyszerre csak egy játszótér tartozéka lehet, habár átvihető másikra is, és önmagában is használható.)

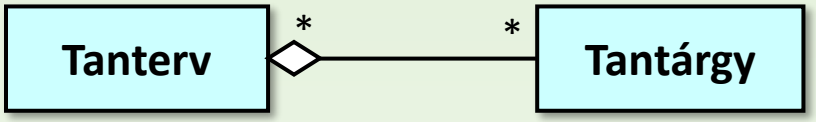


A falak a szoba részei, de ugyanaz a fal két szobához is tartozhat egyszerre. (A döntést zavarhatja, hogy falak nélkül nincs szoba.)

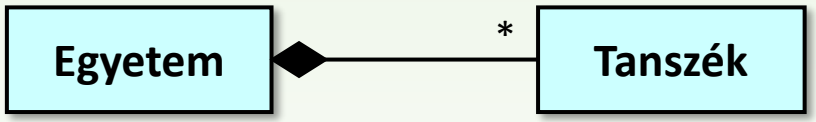


Egy könyv adott számú lapból áll, és a lapjai mindig csak egy adott könyvhöz tartozhatnak. (A döntést nem befolyásolja, hogy egy könyv egy-egy lapja önmagában is használható.)

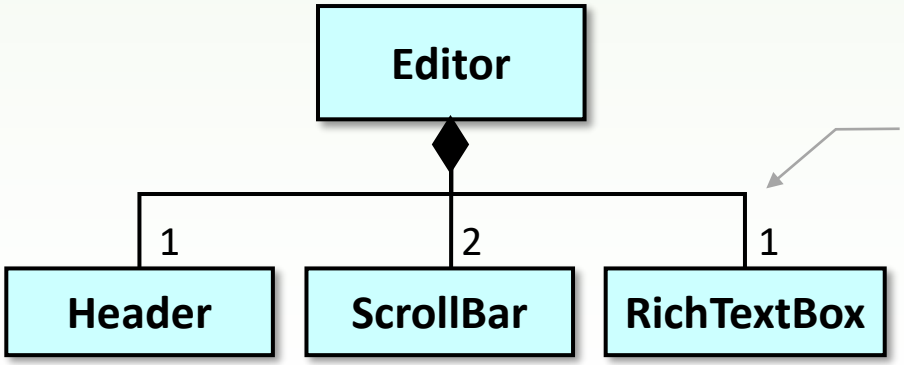
Példák: aggregáció vs. kompozíció



Tanterv ugyan nincs tantárgyak nélkül, de egy tantárgy egyszerre több tantervnek is része lehet (sőt tantervhez sem kell tartoznia).



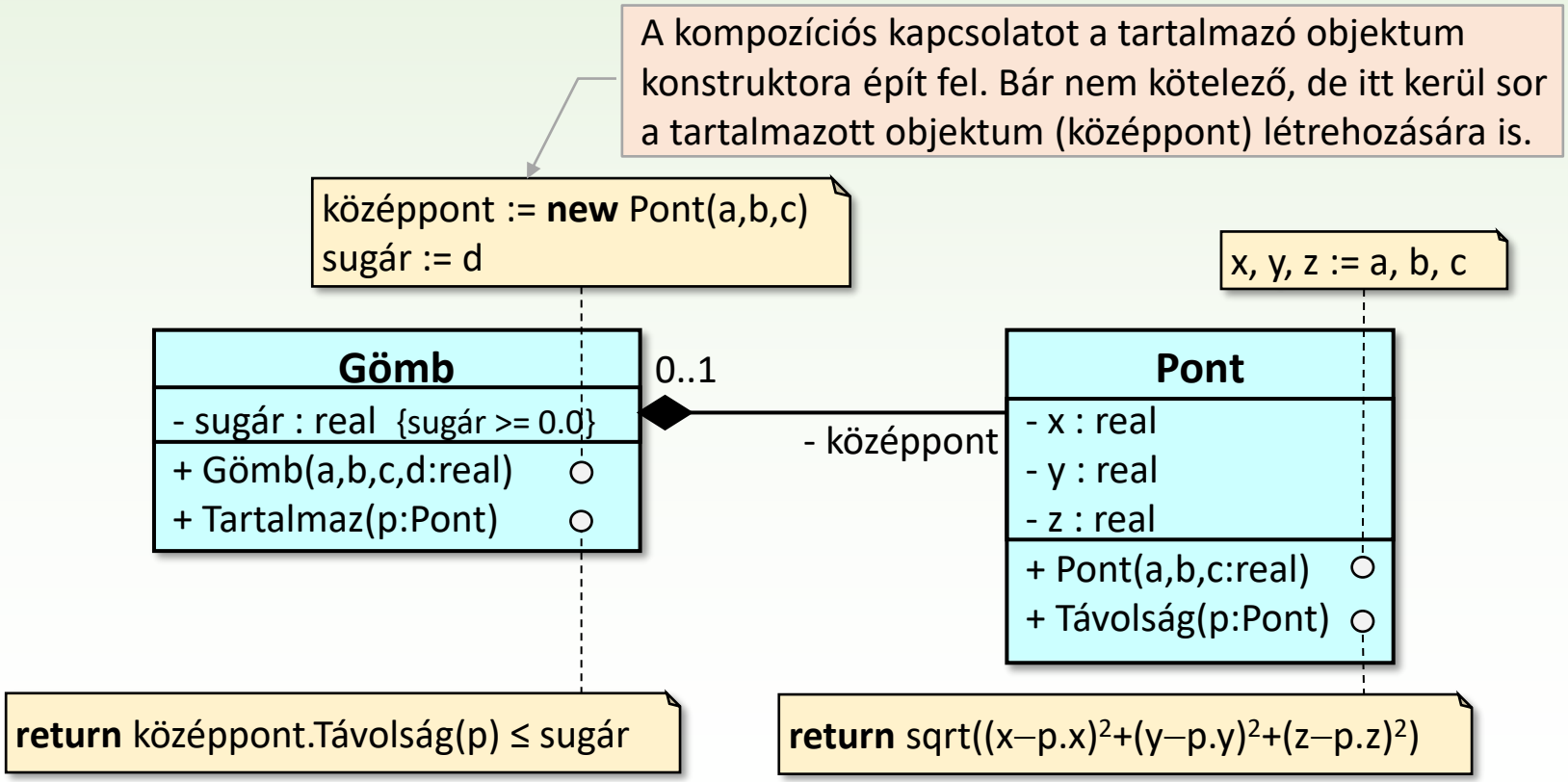
Nincs egyetem tanszék nélkül (de tanszék sem egyetem nélkül), már alapítása idején is van tanszéke (de tanszék később is létrejöhet, és meg is szűnhet). Egy tanszék nem lehet több egyetem tanszéke egyszerre (nem kerülhet át másik egyetemre; ha bezárják az egyetemet, akkor megszűnik).



A szerkesztő ablak létrehozásakor egyszerre jönnek létre annak kizárólagos részei: fejléc, gördítősávok, és szerkesztő felület. Ezek majd az ablak megszűnésével együtt szűnnek meg.

Példa: gömbben a pont

- Modellezzük a térbeli pontok és a gömbök fogalmait egy olyan feladat megoldásának részeként, amelyben meg kell számolnunk, hogy adott térbeli pontok közül hány található egy adott gömbben.
- Ötlet: a gömb középpontja legyen a gömb része.



Példa: gömbben a pont

```
class Point
{
   private readonly double x, y, z;

   public Point(double a, double b, double c) { x = a; y = b; z = c; }
   public double Distance(Point p)
   {
      return Math.Sqrt(Math.Pow(x-p.x,2)+Math.Pow(y-p.y,2)+Math.Pow(z-p.z,2));
   }
}
```

matematikai függvények névtere

```
class Sphere
{
   class IllegalRadiusException : Exception { }
   private readonly Point centre;
   private readonly double radius;
   public Sphere(double a, double b, double c, double r)
   {
      if (r < 0.0) throw new IllegalRadiusException();
      centre = new Point(a,b,c); radius = r;
   }
   public bool Contains(Point p) { return centre.Distance(p) <= radius; }
}
```

4.rész
Származtatás

Gregorics Tibor

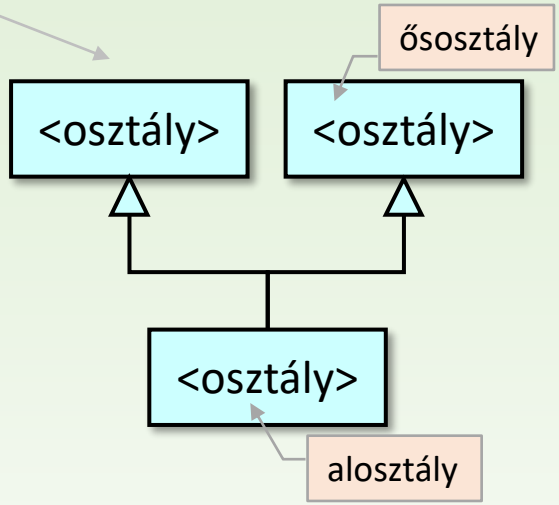
gt@inf.elte.hu

http://people.inf.elte.hu/gt/oep

Származtatás, öröklődés

C#-ban ha több ősosztályból is származtatnánk, akkor ezek egy kivételével csak ún. interfészek lehetnek

- Ha egy objektum más objektumokra hasonlít, azaz azokkal megegyező adattagjai és metódusai vannak, tehát örököli azok tulajdonságait (de azokat akár módosíthatja is, ki is egészítheti), akkor az osztálya a vele hasonló objektumok osztályainak mintájára alakítható ki, azaz belőlük származtatható.

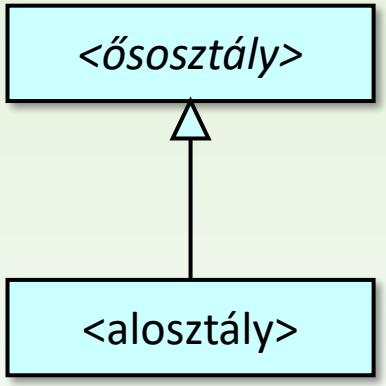


- A modellezés során kétféle okból használunk származtatást:
  - Általánosítás: már meglévő, egymáshoz hasonló osztályoknak a közös tulajdonságait leíró ősosztályát (szuperosztály) hozzuk létre.
  - Specializálás: egy osztályból származtatással hozunk létre egy alosztályt.

5

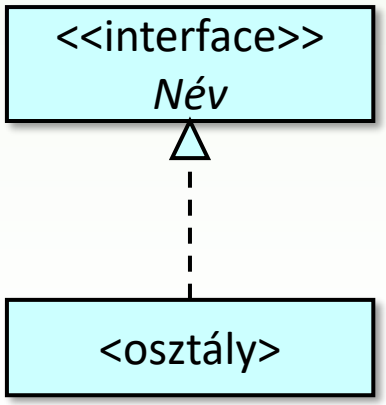
Az objektum-orientált nyelvek támogatják az öröklést: osztályok már meglévő osztályokból származtathatók.
Ősosztály változójának értékül adható az alosztályának objektuma.

Absztrakt osztály, interfész



- Absztrakt (*abstract*) osztály az, amelyből nem példányosítunk objektumokat, hanem kizárólag ősosztályként szolgál a származtatásokhoz. (UML-ben dőlt betűvel írjuk a nevét)
- Egy osztály attól lesz absztrakt, hogy
  - konstruktorai nem publikusak, vagy
  - legalább egy metódusa absztrakt (UML-ben ezt dőlt betű jelzi), azaz nem rendelkezik törzzsel, ezt a származtatás során kell majd megadni.

- Interfésznek a tisztán absztrakt (*pure abstract*) osztályt hívjuk. Ennek nincsenek adattagjai, és egyetlen metódusának sincs törzse.
- Egy interfészből származtatott konkrét osztálynak minden absztrakt metódust implementálni kell: meg kell valósítania az interfészt.



Származtatás és láthatóság

- Egy alosztály az ősosztály minden tagját örökli, de csak az ősosztály publikus és védett tagjaira hivatkozhat, a privát tagjaira nem, azokhoz csak indirekt módon, az ősosztálytól örökölt nem-privát metódusokkal férhet hozzá.
- A származtatás módja maga is lehet

  C# nyelvben csak ez van

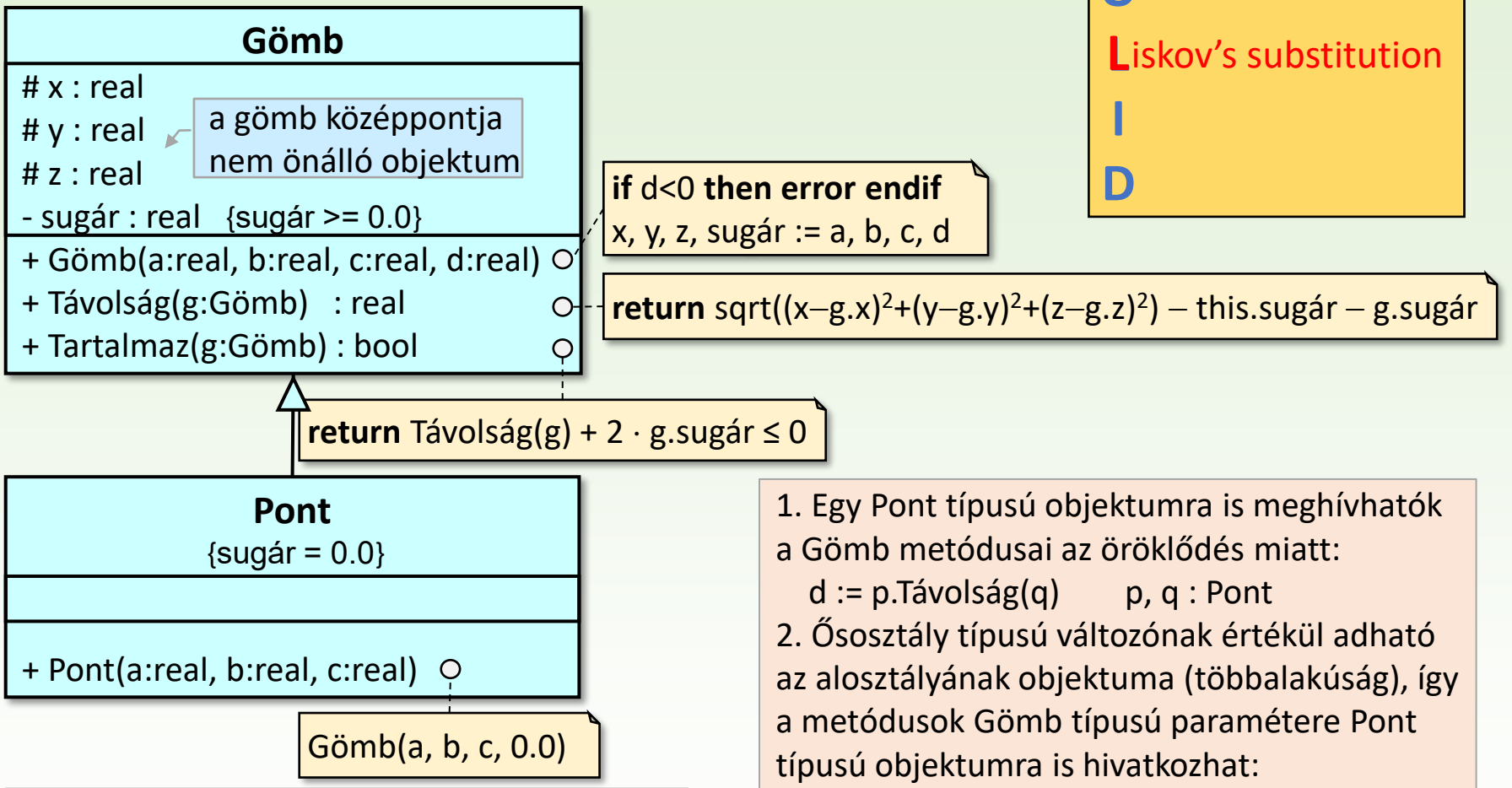
  - publikus (*public*): ekkor az ősosztály publikus és védett tagjai az ősosztályban definiált láthatóságukkal együtt öröklődnek az alosztályra. (UML szerint ez az alapértelmezett)
  - védett (*protected*): ekkor az ősosztály publikus és védett tagjai mind védettek lesznek az alosztályban.

    C++ nyelvben ez az alapértelmezett

  - privát (*private*): ekkor az ősosztály publikus és védett tagjai mind privátok lesznek az alosztályban.

Példa: gömbből pont

Single responsibility
O
Liskov's substitution
I
D



Kívánatos, hogy egy felüldefiniált metódus, akár az ősosztály, akár alosztályának egy objektumára hívják is meg, a modellezett feladat elvárásainak megfelelően működjön.

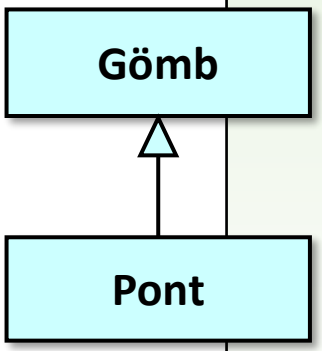
1. Egy Pont típusú objektumra is meghívhatók a Gömb metódusai az öröklődés miatt:
   d := p.Távolság(q) p, q : Pont
2. Ősosztály típusú változónak értékül adható az alosztályának objektuma (többalakúság), így a metódusok Gömb típusú paramétere Pont típusú objektumra is hivatkozhat:
   l := a.Tartalmaz(p) a : Gömb, p : Pont

C# : gömbből pont

```
Point p = new (0, 0, 0);
Sphere g = new (1, 1, 1, 1);
Console.WriteLine("p.Distance(p): {0}", p.Distance(p));
Console.WriteLine("g.Distance(p): {0}", g.Distance(p));
Console.WriteLine("p.Distance(g): {0}", p.Distance(g));
Console.WriteLine("g.Distance(g): {0}", g.Distance(g));
Console.WriteLine("g.Contains(p): {0}", g.Contains(p));
Console.WriteLine("g.Contains(g): {0}", g.Contains(g));
```

védett (protected)

```
class Sphere
{
   class IllegalRadiusExceptic
   protected readonly double x, y, z;
   private readonly double radius;
   public Sphere(double a, double b, double c, double d = 0.0)
   {
      x = a; y = b; z = c; radius = d;
      if (radius < 0.0) throw new IllegalRadiusException();
   }
   public double Distance(Sphere g)
   {
      return Math.Sqrt(Math.Pow((x-g.x),2) + Math.Pow((y-g.y),2)
           + Math.Pow((z-g.z),2)) - radius - g.radius;
   }
   public bool Contains(Sphere g) { return Distance(g) + 2*g.radius<=0; }
}
```

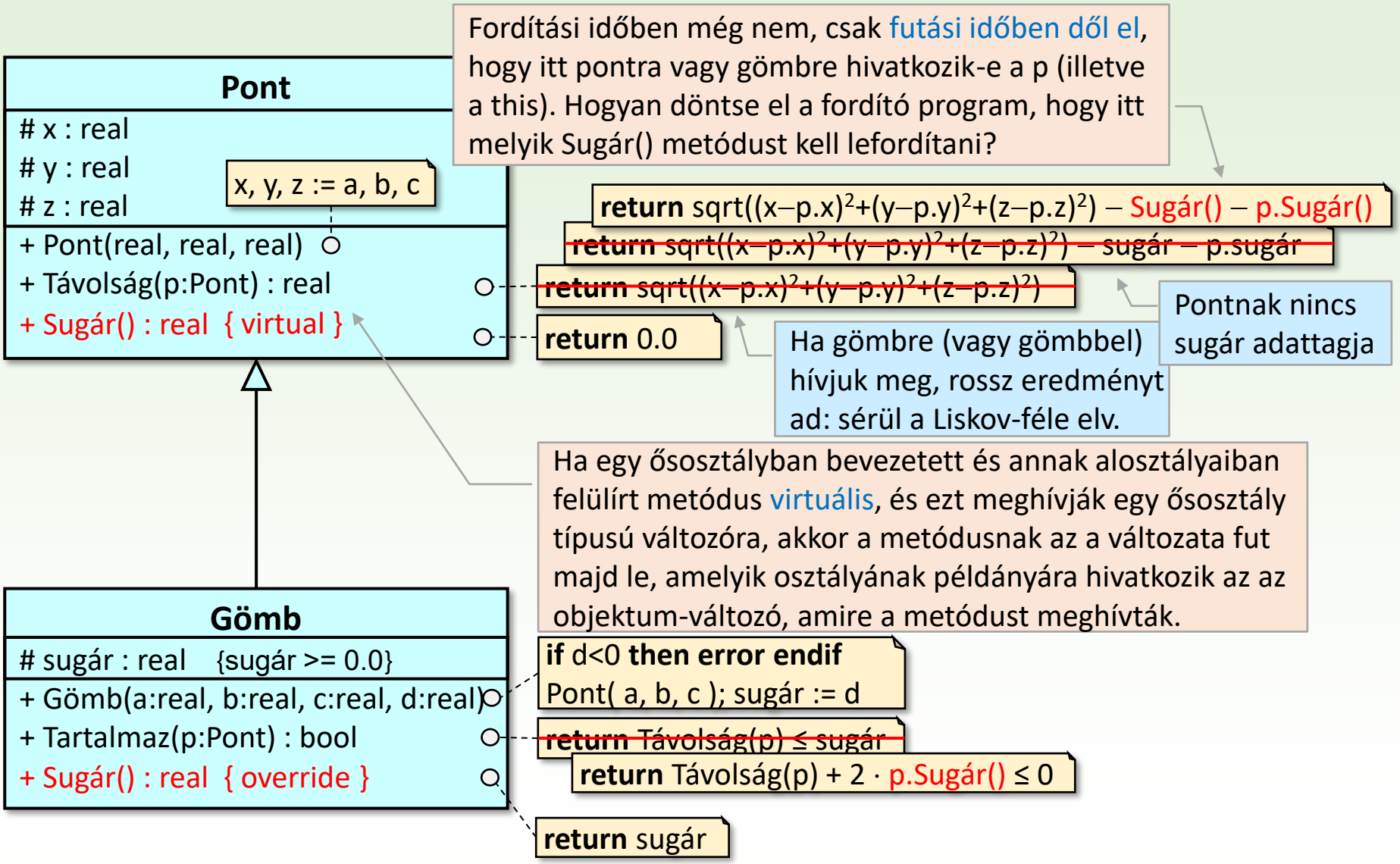


származtatás

```
class Point : Sphere
{
   public Point(double a, double b, double c) : base(a, b, c, 0.0) { }
}
```

az ős konstruktorának hívása helyett közvetlenül is lehetne értéket adni az öröklött védett tagoknak

Példa: pontból gömb



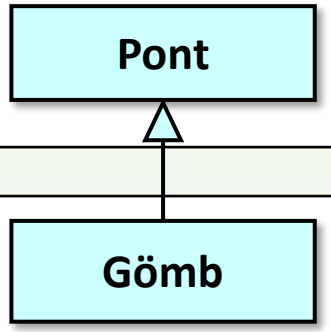
C# : pontból gömb

```
Point  p = new Point(0,0,0);
Sphere g = new Sphere(1,1,1,1);
Console.WriteLine( p.Distance(p) );
Console.WriteLine( g.Distance(p) );
Console.WriteLine( p.Distance(g) );
Console.WriteLine( g.Distance(g) );
Console.WriteLine( g.Contains(p) );
Console.WriteLine( g.Contains(g) );
```

```
class Point
{
   private readonly double x, y, z;
   public Point(double a, double b, double c)
   { x = a; y = b; z = c; }
   public double Distance(Point p)
   {
       return Math.Sqrt(Math.Pow(x - p.x, 2) + Math.Pow(y - p.y, 2)
                      + Math.Pow(z - p.z, 2) - Radius() - p.Radius();
   }
   public virtual double Radius() { return 0.0; }
}
```

virtuális metódus

this.Radius()




```
class Sphere : Point
{
   class IllegalRadiusException : Exception { }
   private readonly double radius;
   public Sphere(double a, double b, double c, double d) : base(a, b, c)
   {
      radius = d;
      if (radius < 0.0) throw new IllegalRadiusException();
   }
   public bool Contains(Point p) { return Distance(p) + 2*p.Radius() <= 0; }
   public override double Radius()  { return radius; }
}
```

felülírt metódus

Dinamikus altípusos polimorfizmus

- Ha egy ősosztály metódusát a leszármazott osztályban felülírjuk (override), akkor ez a metódus több alakkal is rendelkezik (polimorf).
- Mivel egy ősosztály típusú változónak mindig értékül adható alosztálya példányának hivatkozása, csak futási időben derülhet az ki, hogy egy ilyen a változó az ősosztálynak egy példányára vagy alosztályának egy példányára hivatkozik. (késői vagy futási idejű vagy dinamikus kötés).
- Ha egy ősosztály típusú változóra egy polimorf virtuális metódust hívunk meg, akkor e metódusnak azon osztályban definiált változata fut majd le, amilyen osztályú példányra hivatkozik a változó (dinamikus altípusos polimorfizmus).

C++ esetén referencia- vagy pointerváltozó

6 Az objektum-orientált nyelvek ismérve a dinamikus altípusos polimorfizmus.