# Osztály öröklődés

Öröklődés és polimorfizmus

## Szoftvertervezés és -fejlesztés II. előadás

http://nik.uni-obuda.hu/sztf2

Szénási Sándor
szenasi.sandor@nik.uni-obuda.hu

Óbudai Egyetem,Neumann János Informatikai Kar

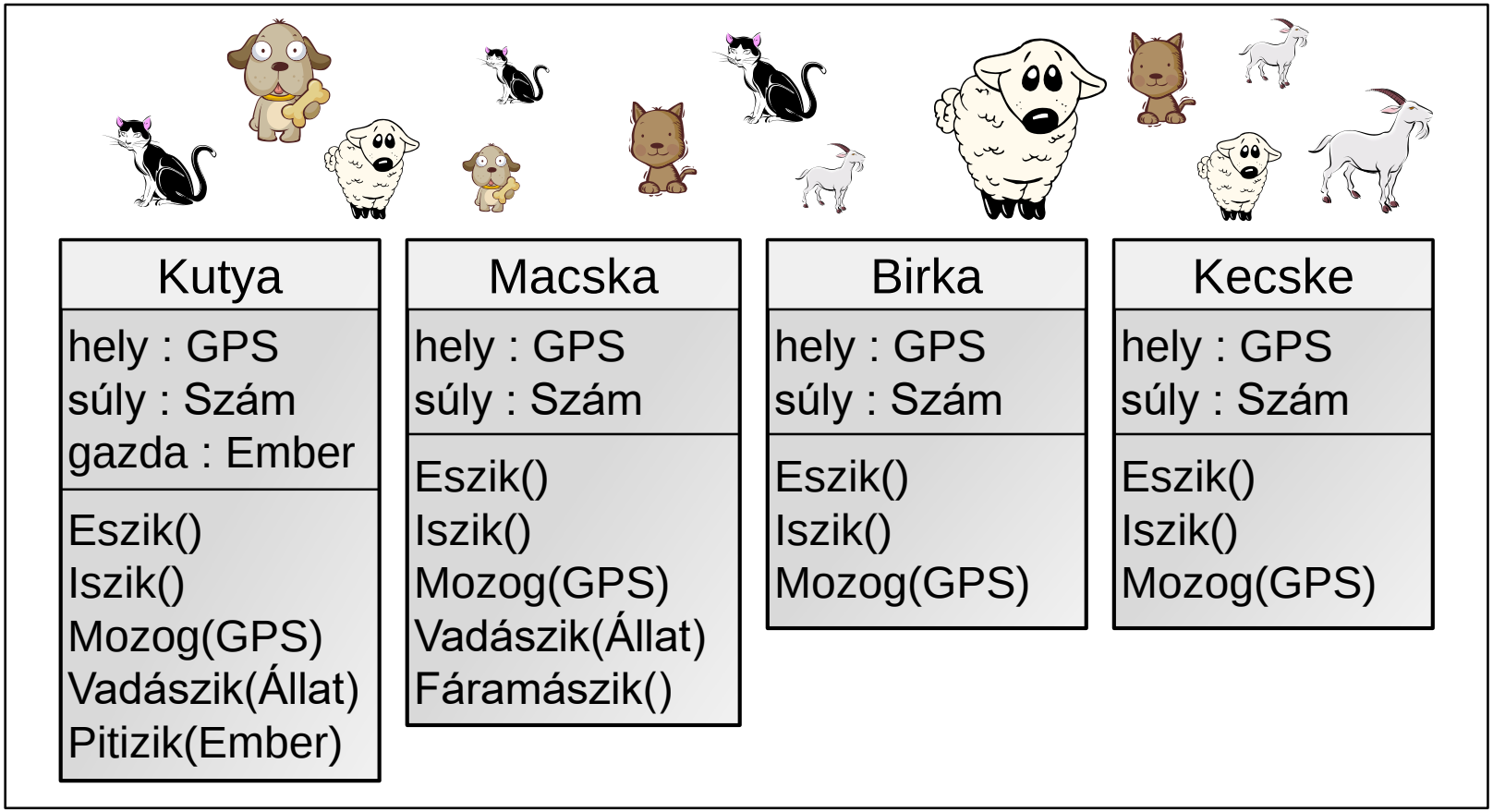
# Osztály öröklődés

**Öröklődés**

Polimorfizmus

Osztályok közötti kapcsolatok

- Modellezés lépései
  - A probléma leírásában megkeressük az objektumokat
  - Objektumcsoportok közös tulajdonságait osztályokkal írjuk le
  - Megkeressük az osztályok kapcsolatait

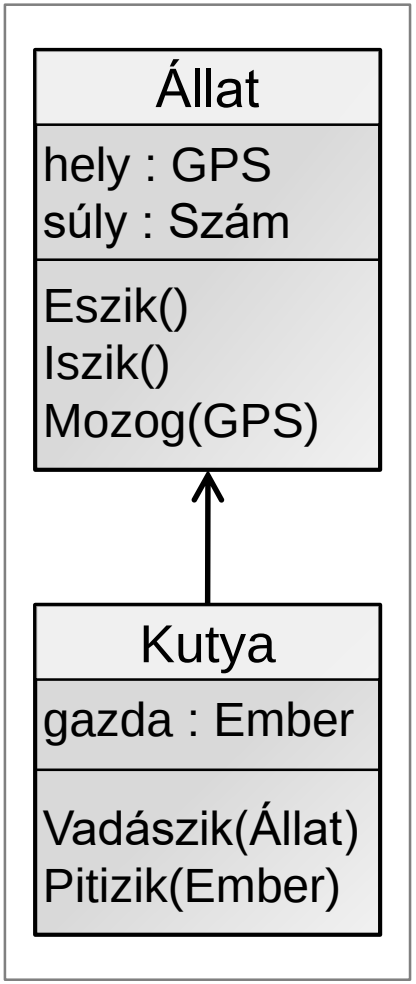


# Hasonló osztályok kezelése

- Hasonló osztályok kezelése
  - Redundáns kódot nem engedhetjük meg
    - Felesleges kódolási munka
    - Nehezen módosítható
    - Kód áttekinthetetlen
  - Következmény: amit lehet, ki kell emelni
- Alacsonyabb szinten a kiemelés
  - Konstansok az ismétlődő értékek helyett
  - Ciklus az ismétlődő utasítások helyett
  - Függvények/metódusok a gyakori kódrészletek helyett
- Különféle technikai lehetőségek
  - Makrók
  - Technikai osztályok
  - Stb.
- Osztályok szintjén a kiemelés: öröklődés

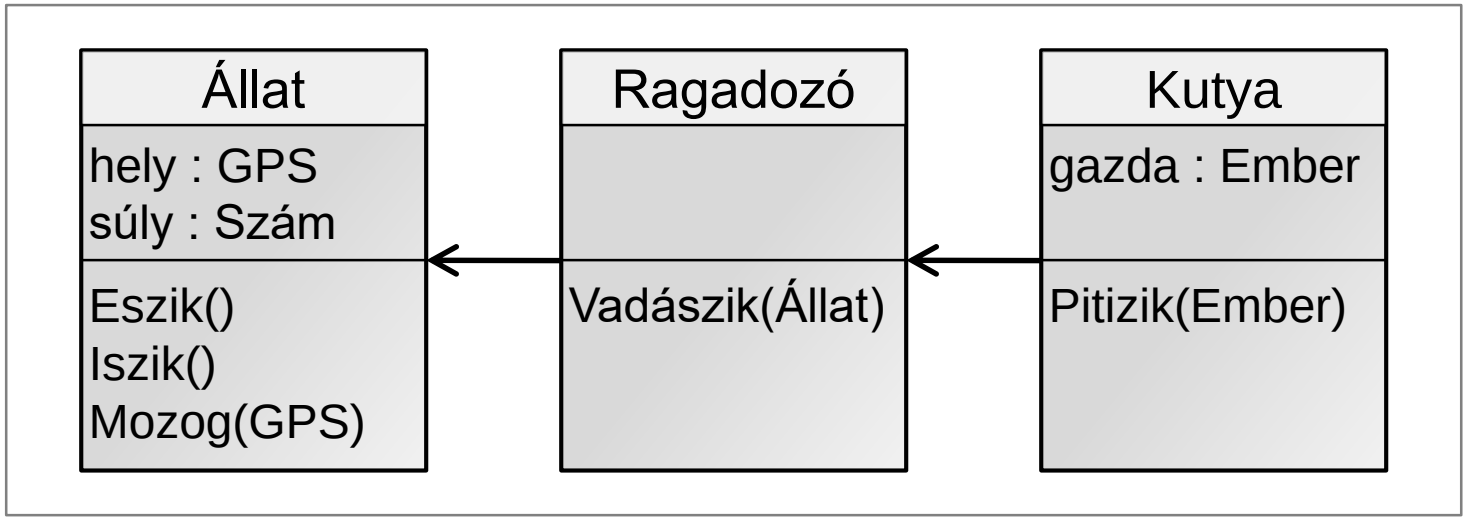
# Öröklés definíciója

- Az öröklés (inheritance) olyan implementációs és modellezési eszköz, amelyik lehetővé teszi, hogy egy osztályból olyan újabb osztályokat származtassunk, amelyek rendelkeznek az eredeti osztályban már definiált tulajdonságokkal, szerkezettel és viselkedéssel
- Úgy lehet létrehozni további osztályokat, hogy csak a változásokat kell megírni
- Egy specializációs művelet, a leszármazottak bővíthetik, vagy szűkíthetik az őstípus állapotterét, működését



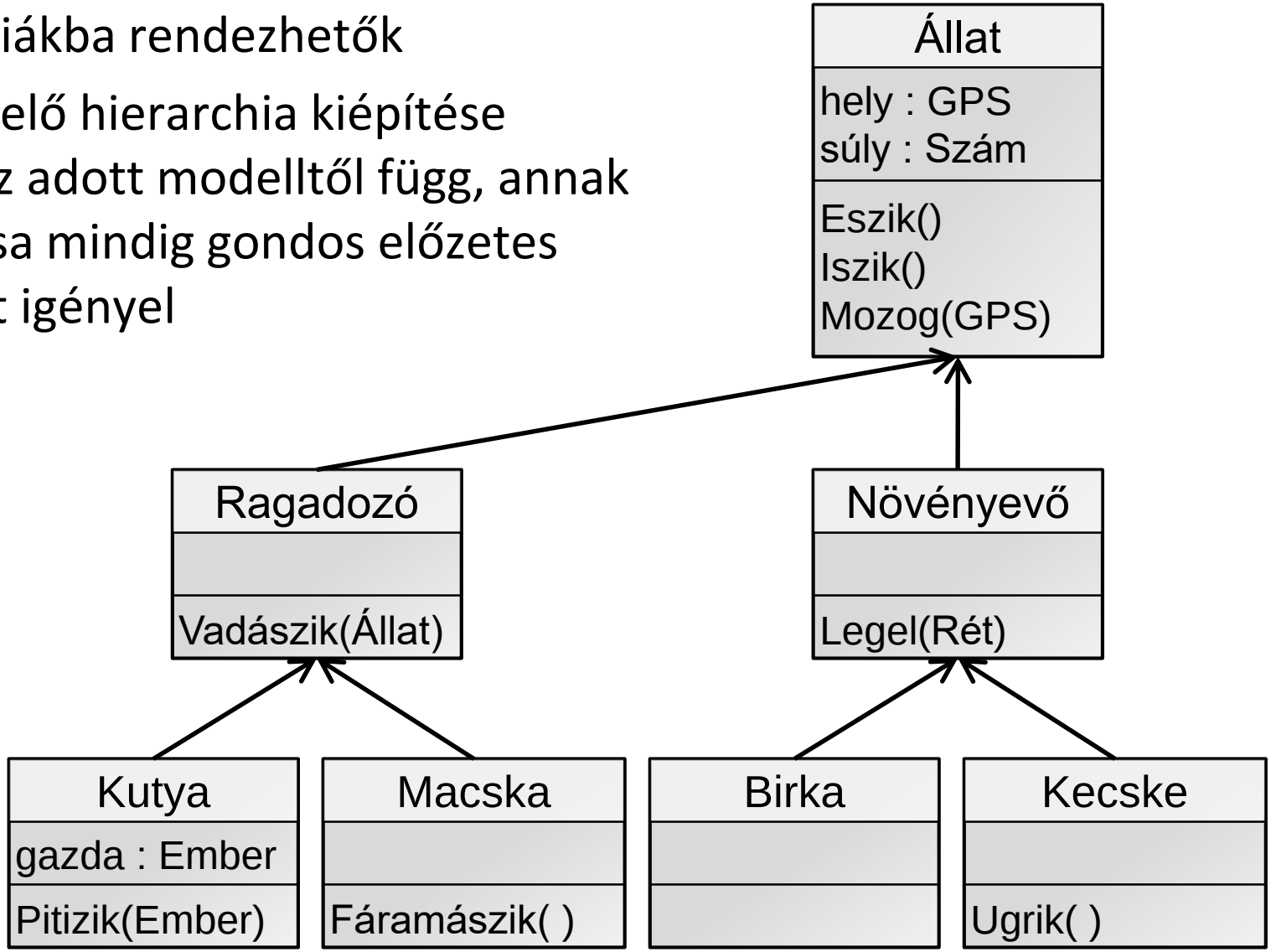
# Osztályhierarchia

- Az öröklés többszöri felhasználásával teljes leszármazási hierarchiákat is létrehozhatunk
- Alapfogalmak
  - Osztályhierarchia: egymásból leszármazó osztályok hierarchikus rendszere (származási fa)
  - Közvetlen ős: az osztályhierarchiában egy szinttel magasabban lévő osztály
  - Ős: a származási fa magasabb szintű eleme
  - Leszármazott: a származási fa alacsonyabb szintű eleme



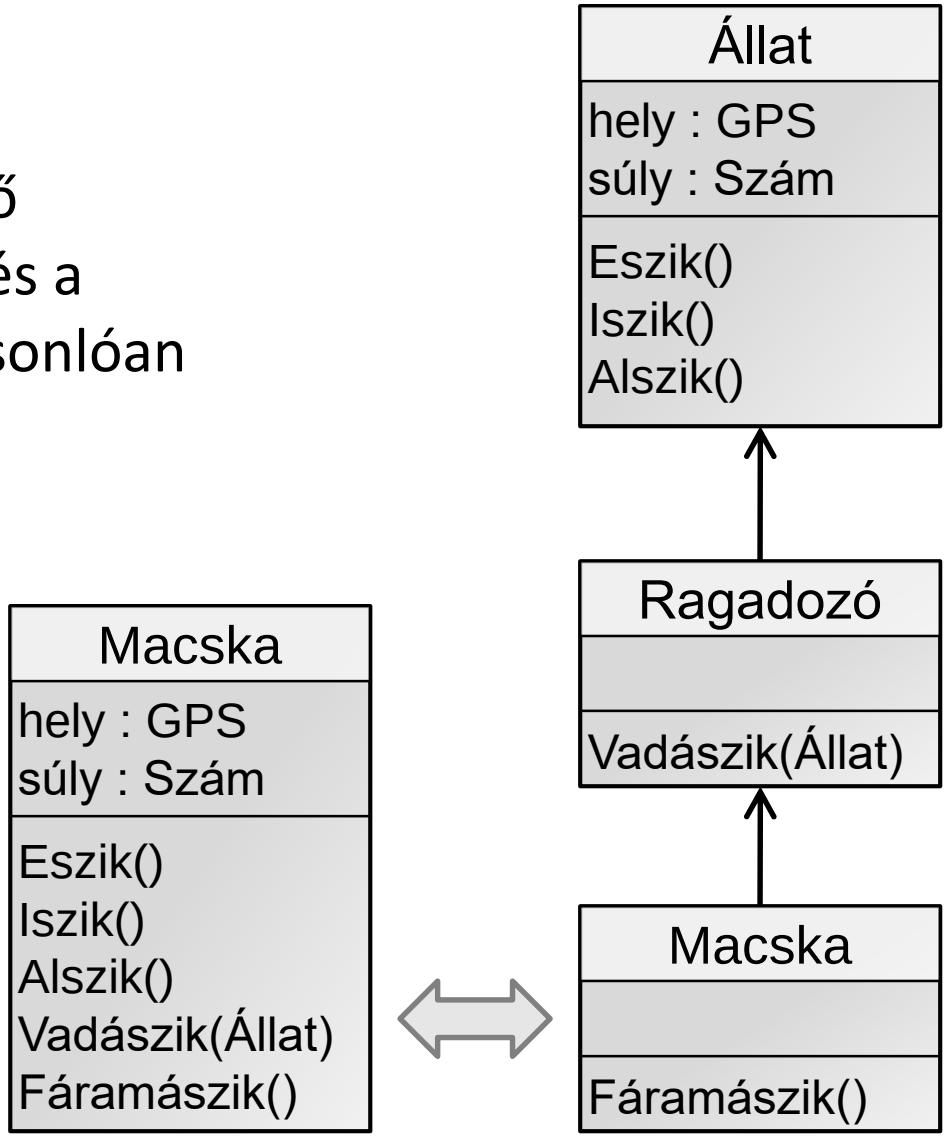
# Osztályhierarchia felépítése

- Megadott osztályok különféle hierarchiákba rendezhetők
- A megfelelő hierarchia kiépítése mindig az adott modelltől függ, annak kialakítása mindig gondos előzetes tervezést igényel



# Hierarchia értelmezése

- A leszármazottak mind rendelkeznek őseik összes adatmezőjével és metódusával
- Az osztály használatakor (külső szemlélő számára) az örökölt és a saját mezők és metódusok hasonlóan működnek

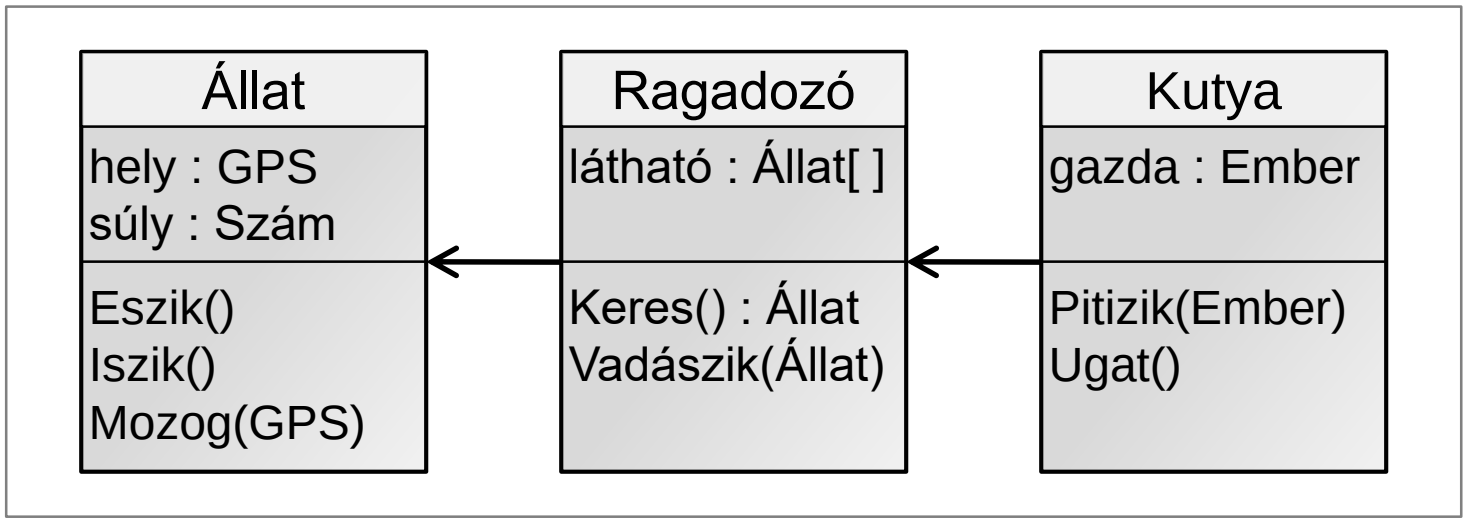


# Új mezők és metódusok

- A leszármazottnak lehetősége van módosítani az ős osztályt az alábbiak segítségével:
  - Új mezők felvétele
  - Új metódusok felvétele
  - Felüldefiniálás: egy már meglévő metódus újraírása
- Új mezők felvétele
  - Nevük nem lehet azonos egy már örökölt mezőével
  - Nincs lehetőség a típus megváltoztatására
  - Nincs lehetőség az örökölt mező hozzáférési szintjének szigorítására
- Új metódus felvétele
  - Esetleg lehet lehetőség egy örökölttel azonos nevű metódus létrehozására (elrejtés)

# Új mezők és metódusok

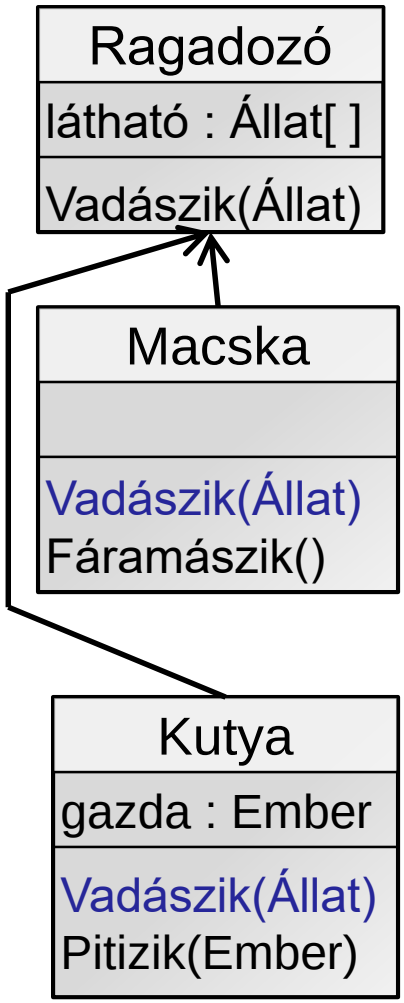
- Új és örökölt mezők és metódusok teljesen azonos módon kezelhetők
- Az ős nem férhet hozzá a leszármazottban definiált mezőkhöz vagy metódusokhoz



# Metódus felüldefiniálás

- A leszármazott felüldefiniálhatja az ős egy metódusát. Technikailag ez általában azt jelenti, hogy tartalmaz egy azonos nevű és szignatúrájú metódust mint az ős
- Segítségével megoldható, hogy a leszármazottak ugyanazt a funkciót biztosítsák, mégis más-más megvalósításban
- Erre akkor lehet szükség, ha az őstől örökölt metódus megvalósítása a leszármazott számára már nem megfelelő
  - Tipikusan a leszármazott specializált mivoltából adódó változások esetén szükséges
    - Új mezők használata
    - Új metódusok meghívása
- Átalában elérhetők az ős műveletei és ezeket célszerű is használni

# Metódus felüldefiniálás




```
eljárás Ragadozó.Vadászik(Állat zs)
  Mozog(zs.hely)
  <gyenge idegzetűek miatt nem implementált>
eljárás vége
```

```
eljárás Macska.Vadászik(Állat zs)
  Ha (zs.hely.magasság > 0) Akkor
    Fáramászik()
  eljárás vége
  Ragadozó.Vadászik(zs)
eljárás vége
```

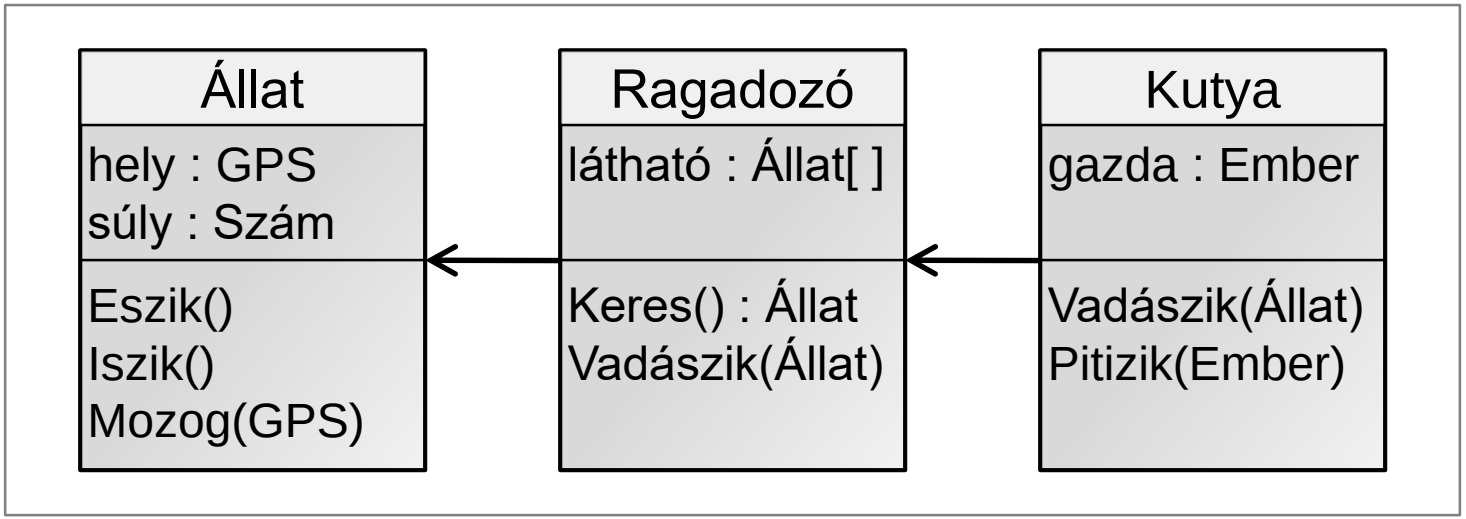
```
eljárás Kutya.Vadászik(Állat zs)
  Ha (zs.hely.táv < gazda.hely.táv) Akkor
    Ragadozó.Vadászik(zs)
  Különben
    Pitizik(gazda)
  eljárás vége
eljárás vége
```

# Speciális metódusok öröklődése

- Konstruktor és destruktor öröklődése
  - A konstruktorok és a desktruktorok nem öröklődnek
  - A leszármazottaknak is lehetnek konstruktorai, ezek nem írják felül az ősökben található konstruktort
  - Célszerűen a hierarchia minden szintjén csak az ott kezelt adatokat létrehozni/megszüntetni
- Ős konstruktorának meghívása
  - A leszármazottakból elérhető az ős konstruktora
  - Általában kötelező meghívni az ős valamelyik konstruktorát
    - Explicit konstruktorhívás
      - A leszármazott konstruktorából meg lehet hívni az ős konstruktorát, hogy megtörténjen a mezők inicializálása
      - A paraméterlista határozza meg, hogy melyik konstruktor fusson le
    - Automatikus konstruktorhívás
      - Amennyiben nincs explicit konstruktorhívás, az ős konstruktorának meghívása megtörténhet automatikusan
      - Tipikusan az ős paraméter nélküli konstruktora hívódik meg

# Speciális metódusok hívási sorrendje

- Konstruktor hívási sorrend
  - Származási hierarchia esetén minden ős konstruktora lefut
  - Ezek a származás sorrendjében futnak le, hiszen a leszármazott már hivatkozhat az ős adataira
- Destruktor hívás sorrend
  - Származási hierarchia esetén minden ős destruktor lefut
  - Ezek a származással ellentétes sorrendjében futnak le, hiszen az utódok a saját destruktorukba még hivatkozhatnak az ős által kezelt objektumokra



# Láthatósági szintek

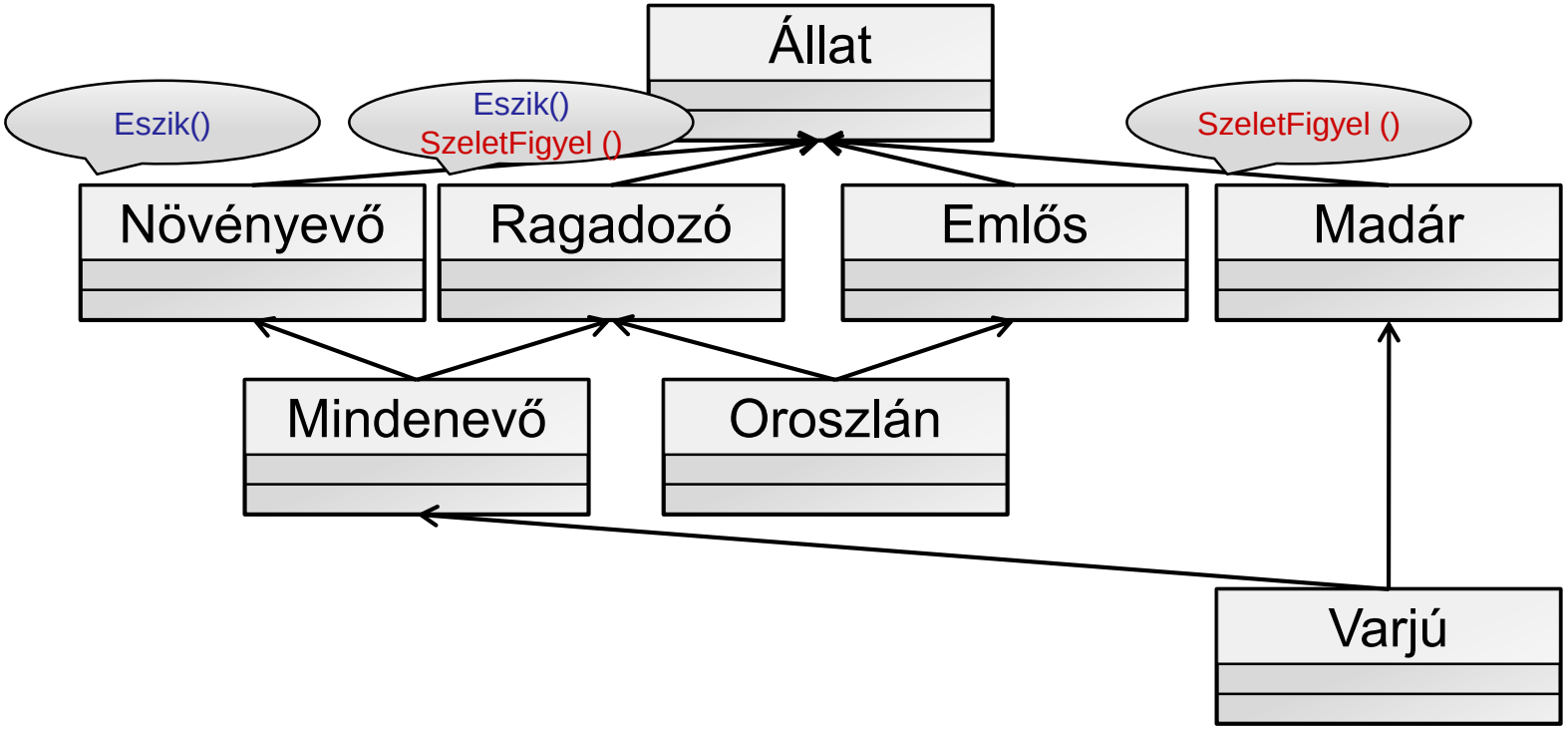
- Minden mező és metódus rendelkezik egy saját láthatósági szinttel
  - Cél az egységbezárás elvének megvalósítása
  - Segítségével megadott tagokhoz csak megadott osztályok férhetnek hozzá, ezzel csökkentve a programozói hibákat
  - A mezők írása és olvasása nincs megkülönböztetve
- Ezek az alábbiak lehetnek:
  - Privát (private)
    - Csak a megadott osztály metódusaiból érhető el
    - Az osztály belső funkcionalitását valósítja meg
  - Védett (protected)
    - Megadott osztály saját metódusai és az osztály leszármazottainak minden metódusa hozzáférhet
  - Nyilvános (public)
    - Minden osztály minden metódusa hozzáférhet
    - Az osztály külvilág által látható felülete
  - Egyéb, nyelvfüggő lehetőségek (internal, protected internal)

# Többszörös öröklés

- Többszörös öröklés esetén egy osztálynak több őse van
- Értelemszerűen minden ős minden mezőjét és metódusát öröklik a leszármazottak
- Nem minden programozási nyelv támogatja
  - Támogatja: C++
  - Nem támogatja: C#, Java
- Előnyei
  - Egyszerre több ősosztály képességei örökölhetők
  - Több ős tulajdonságait ötvöző vegyes osztályok hozhatók létre
  - Nagy rugalmasságot biztosít
- Hátrányai
  - Nehéz megvalósítani
  - Nagyobb rendszereknél áttekinthetetlen kódot eredményezhet
  - Bonyolítja a programok tervezését és implementálását
- Gyakran csak a polimorfizmus miatt van rá szükség, ilyenkor az interfészekkel is jól lehet helyettesíteni

# Többszörös öröklés tipikus problémái

- A leszármazottak több ős esetén több azonos nevű metódust és mezőt is örökölhetnek
- Ha egy osztály több ősének is van egy korábbi közös őse, akkor ennek metódusait a végső leszármazott több irányból is megkaphatja különböző megvalósításokkal



# Egyéb tulajdonságok

- Statikus (osztály szintű) tagok és az öröklés
  - A statikus mezők és metódusok mindig egy adott osztályhoz tartoznak
  - Ennek megfelelően az öröklléstől ezek teljesen függetlenek
  - Minden szinten létre lehet hozni azonos nevű statikus mezőket és metódusokat, ezek azonban egymástól teljesen függetlenek

# Osztály öröklődés

Öröklődés

**Polimorfizmus**

Osztályok közötti kapcsolatok

# Típus fogalma

- Adat típusa
  - Mit ábrázolunk?
  - Milyen belső ábrázolással?
  - Milyen külső ábrázolással?
  - Milyen műveleteket végezhetünk vele?
- Primitív típusok – referencia típusok
  - Primitív típusok esetében a változó neve közvetlenül az adatok tárolására szolgáló rekeszt azonosítja, így a deklarált változó típusa megegyezik az eltárolt adat típusával
  - OOP környezetben az objektumokra egy referencián keresztül hivatkozunk, az objektum típusa mellett megkülönböztethetjük a referencia típusát is
  - A referencia típusa fontos a fordítóprogram számára, mivel ez alapján dönti el, hogy egy művelet végrehajtható-e vagy sem
  - Eddigi gyakorlatunkban ez a két típus mindig azonos volt, értelmetlen is lenne egy objektumra egy egészen más típusú referenciával hivatkozni

# Öröklés hatása a típus fogalomra

- Az öröklésnek köszönhetően lehetőségünk van egy objektumra egy más típusú referenciával hivatkozni
- Minden T típusú objektumra hivatkozhatunk T típusú vagy T bármelyik őse típusú referenciával

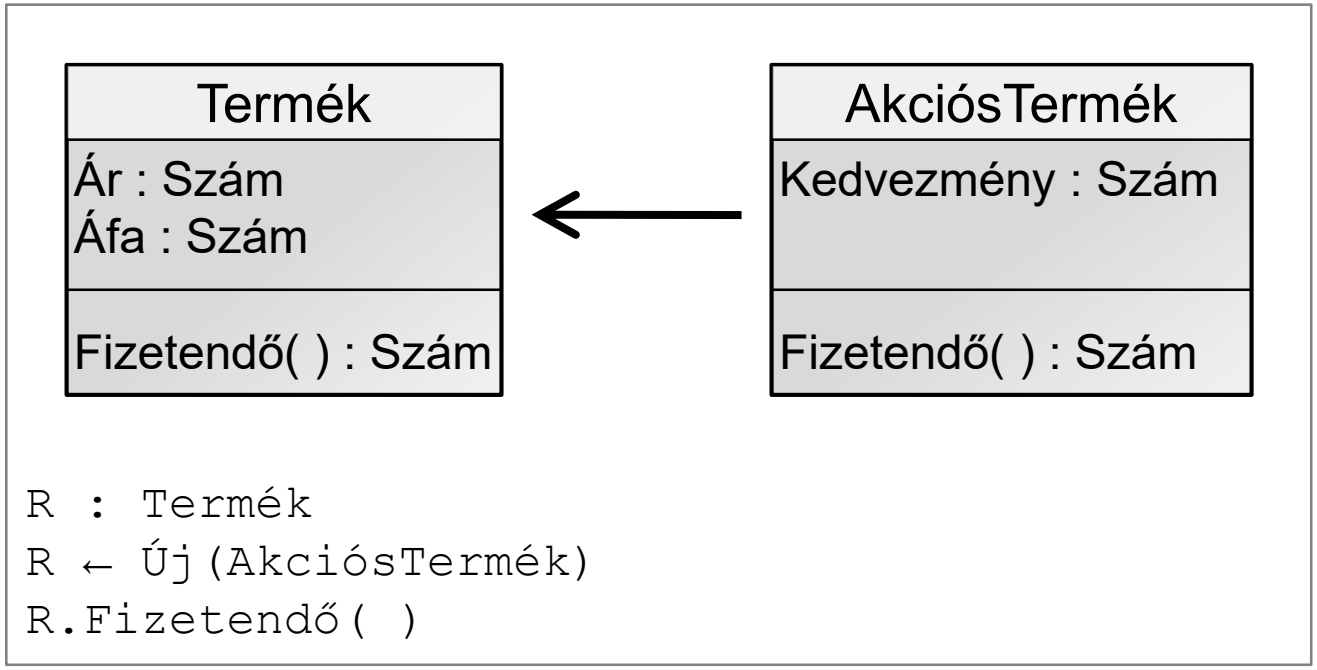
```
A : Termék
A ← Új(Termék)
A.ÁrNövel()

B : AkciósTermék
B ← Új(AkciósTermék)
B.ÁrNövel()
B.KedvezményNövel()

C : Termék
C ← Új(AkciósTermék)
C.ÁrNövel()

D : AkciósTermék
D ← Új(Termék)
```

# Új mezők és új metódusok használata

- A leszármazottakban felvett új mezők és metódusok a leszármazott referencián keresztül elérhetők
- Az ős referenciáján keresztül ezek nem érhetők el még akkor sem, ha a referencia egy leszármazott objektumra hivatkozik
- A fordítóprogramnak már a fordításkor garantálnia kell, hogy egy referencia se hivatkozhasson olyan objektumra, amelyik nem rendelkezik a szükséges tagokkal
  - Új objektum létrehozás
  - Értékadás

# Felüldefiniált metódusok használata

- Mi történjen ha az ős egyik metódusát a leszármazott felüldefiniálta, azonban ezt a metódust az ős referenciáján keresztül hívjuk meg




- Melyik osztály metódusa fut le?
  - Ős – hiszen a referencia típusa alapján ezt várnánk
  - Leszármazott – hiszen az objektum típusa alapján ezt várnánk

# Korai kötés

- Korai kötés
  - Már fordításkor eldől, hogy a meghívott metódushoz melyik megvalósítás tartozik
  - Mivel a fordító nem tudhatja biztosan, hogy a referencia milyen objektumra fog mutatni futás közben, ezért a referencia típusát veszi alapul
  - Mivel a referencia típusa nem változhat futás közben így magát a meghívást megvalósító gépi kódot már a fordításkor elkészítheti (statikus kötés)
- Jellemzői
  - Előnyei
    - Egyszerű megvalósítás
    - Nincs futásidejű teljesítményveszteség
  - Hátrányai
    - Nem alkalmas polimorfizmus megvalósítására
- Az előző példában ennek megfelelően az ősben megvalósított metódus fut le

# Késői kötés

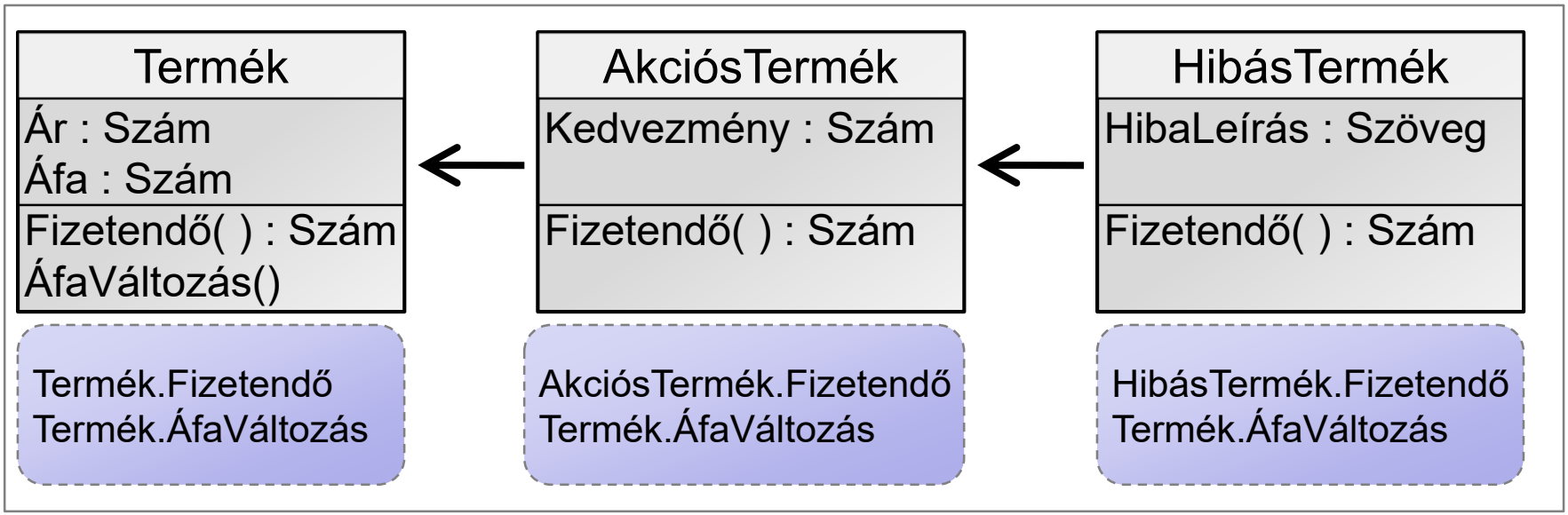
- Késői kötés
  - A metódus meghívásakor a referencia által hivatkozott objektum valódi típusának megfelelő osztály metódusa hívódik meg
  - A fordító nem tudhatja, hogy a meghívás pillanatában milyen típusú objektumra fog hivatkozni a referencia, így olyan kódot generál, ami futás közben dönti el, hogy melyik metódust hívja meg (dinamikus kötés)
- Jellemzői
  - Előnyei
    - Rugalmasság
    - Polimorfizmus lehetősége
  - Hátrányai
    - Teljesítményveszteség
    - Nagyobb tárigény
    - Bonyolultabb megvalósítás
- Az előző példában ennek megfelelően a leszármazottban megvalósított metódus fut le

# Nemvirtuális - virtuális metódusok

- A kötési mód nyelvtől függően választható
  - Java – csak késői kötés van
  - C#, Delphi – választható a metódus megvalósításakor
- A kötési módot nem meghíváskor kell megadnunk, hanem a metódus megvalósításakor
  - Nemvirtuális metódus
    - Minden meghívásakor korai kötést fog használni
    - Leszármazottakban nem lehet felüldefiniálni, csak esetleg elrejteni egy azonos nevű metódussal
  - Virtuális metódus
    - Minden meghívásakor késői kötést fog használni
    - Leszármazottakban felül lehet definiálni, ilyenkor a késői kötésnek megfelelően fog a megfelelő metódus lefutni
    - Egy következő leszármazottban újra felüldefiniálható
- A virtuális metódus a polimorfizmuson keresztül az OOP egyik legfontosabb alapköve

# VMT - Virtuális Metódus Tábla

- A késői kötés támogatásának egyik lehetséges megvalósítása a Virtuális Metódus Tábla
  - Egy objektum VMT táblázata tartalmazza a dinamikusan kötött metódusok címeit
  - Egy metódus meghívásakor a VMT alapján dönthető el (futásidőben), hogy melyik megvalósításnak kell lefutnia
  - Azonos osztályba tartozó objektumoknál ez mindig azonos, ezt tipikusan meg szokták osztani, és minden objektum csak egy mutatót tárol erre a táblázatra



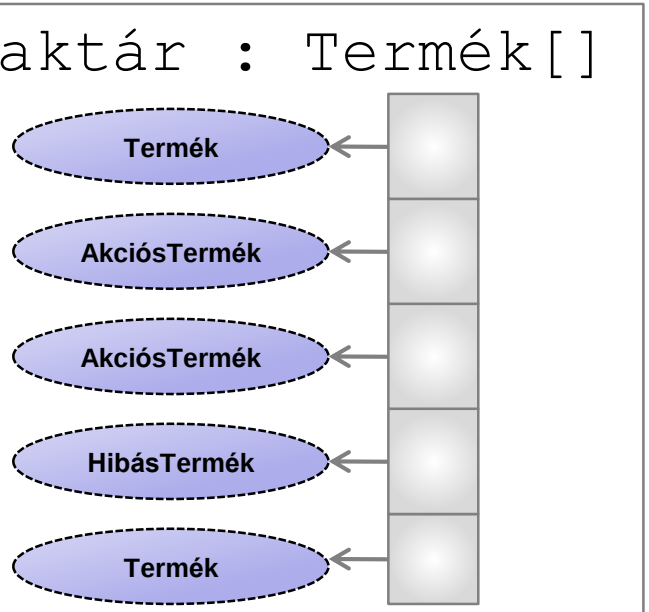
# Polimorfizmus

- Virtuális műveletek a gyakorlatban
  - Minden olyan helyzetben és funkcióban, ahol az őstípus szerepelhet, annak bármely leszármazottja szerepelhet
  - Mezők és nemvirtuális metódusok esetén a leszármazott pontosan ugyanúgy viselkedik mint az ős
  - Virtuális metódusok esetén van rá lehetőség, hogy a leszármazott ugyanarra a metódushívásra másképp reagáljon
- Polimorfizmus (többalakúság) azt jelenti, hogy ugyanarra az üzenetre különböző objektumok különbözőképpen reagálhatnak
  - Módszer polimorfizmus: egy leszármazott osztály egy örökölt módszert újraimplementálhat
  - Objektum polimorfizmus: minden egyes objektum szerepelhet minden olyan szituációban, ahol az ősosztály objektuma szerepelhet, nem csak saját példányaként használható
    - Változó értékadásnál
    - Paraméter átadásnál

# Polimorfizmus változó értékadásnál

- Egy őstípus referenciájával lehet hivatkozni bármelyik leszármazottjából példányosított objektumra
  - Jól alkalmazható sokféle objektum tárolásánál, ilyenkor nem kell minden típushoz külön tömböt létrehozni, elég egy ős típusú
  - A tároláson túl az egyes objektumok metódusait is meg lehet hívni az ős referencián keresztül (ami az ősben definiált)
  - Virtuális metódusok esetében ezek mindig az adott objektum példánynak megfelelő megvalósítást hajtják végre

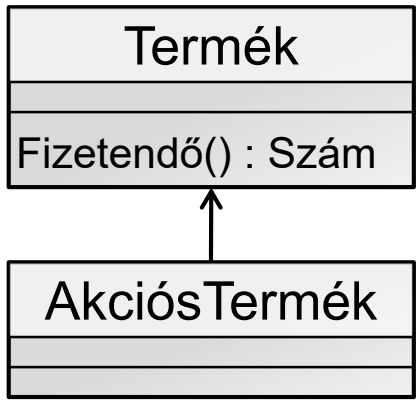
```
Raktár[1] ← Új(Termék)
Raktár[2] ← Új(AkciósTermék)
...
Raktár[1].Ár = ...
Raktár[2].Áfa = ...
Raktár[3].ÁfaVáltozás()
...
ciklus i ← 1-től N-ig
  KI: Raktár[i].Fizetendő()
ciklus vége
```



# Polimorfizmus paraméter átadásnál

- A változó értékadáshoz hasonlóan paraméterátadásnál is alkalmazhatók az előbbi lehetőségek
  - Egy T típusú paramétert váró függvénynek átadhatunk T típusú vagy T bármelyik leszármazottja típusú objektumot paraméterként
  - Egy T típusú visszatérési értékkel rendelkező függvény eredményeként visszaadhat T típusú vagy T bármelyik leszármazottja típusú objektumot
  - Erre célszerű számítani is, ha mi magunk hívjuk a függvényt, elképzelhető, hogy nem pontosan a megadott típust kapjuk paraméterként

```
eljárás Kosárba(Termék t)
  KI: t.Fizetendő()
eljárás vége

Termék nt ← Új(Termék)
Kosárba(nt)
AkciósTermék at ← Új(AkciósTermék)
Kosárba(at)
```



# Polimorfizmus mezők esetében

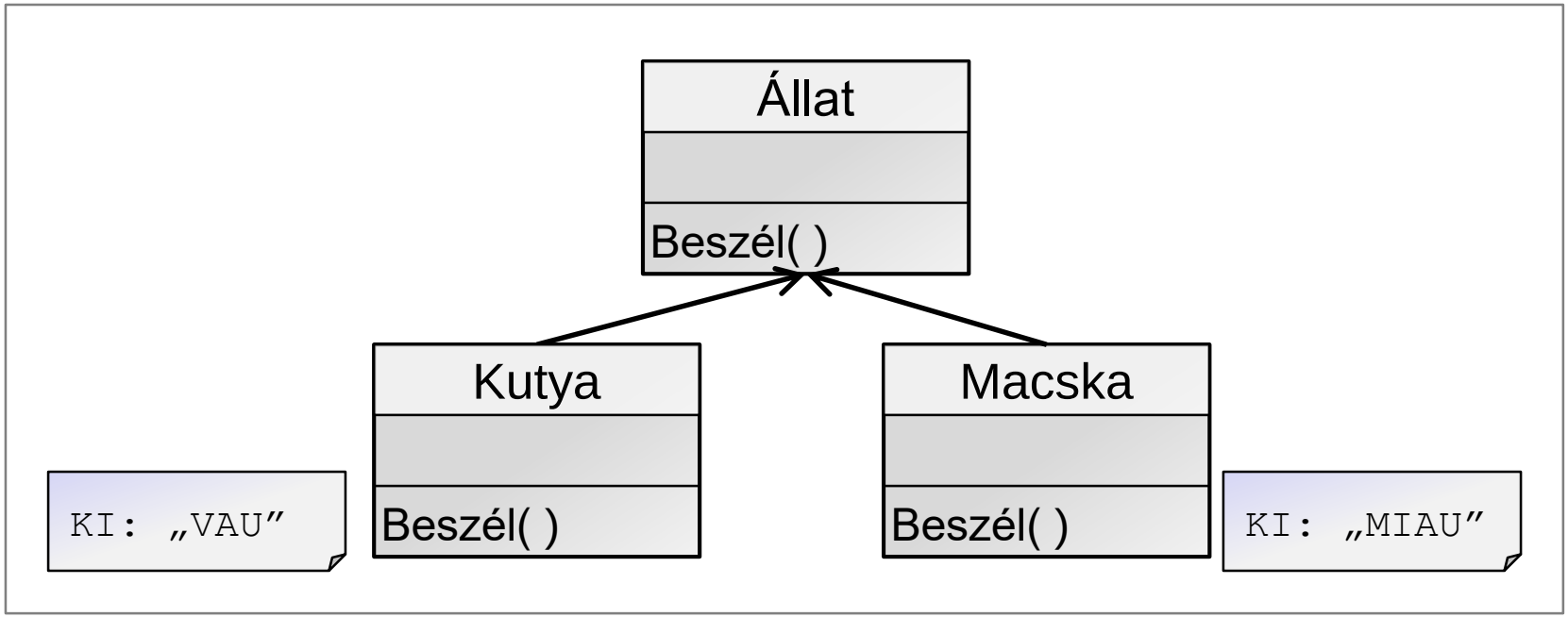
- Az örökölt mezők a leszármazottakban is elérhetőek
  - Az elérést az adott mező hozzáférési szintje szabályozza
- A mezők statikusan viselkednek
  - Típusuk, nevük nem változtatható meg az öröklés során
  - Minden leszármazott tartalmazza az ősének minden mezőjét, nincs lehetőség azok szűkítésére
  - A leszármazott felvehet új mezőket, de azokhoz az ős referenciákon keresztül nem férhetünk hozzá
  - A mezők így nem képesek a polimorfizmusra
- Ez nem keverendő össze a tulajdonság fogalommal, amely funkciójában a mezőkhöz, de megvalósításában inkább a metódusokhoz hasonlít
  - Tulajdonság olvasása – olvasó metódus
  - Tulajdonság írása – író metódus

# Absztrakt osztályok

- Absztrakt osztály jellemzői
  - Absztrakt metódus: deklarált, de meg nem valósított virtuális metódus
  - Absztrakt osztály: legalább egy absztrakt metódust tartalmazó osztály
- Az absztrakt osztály célja az öröklés kikényszerítése
  - Absztrakt osztály nem példányosítható
  - A leszármazottak kötelesek megvalósítani az összes „örökölt” absztrakt metódust
  - Amennyiben nem tudják implementálni a hozzá tartozó műveletet, a leszármazott osztálynak is absztraktnak kell lennie
- Segítségükkel általános funkcionalitás írható elő az öröklési hierarchia felsőbb szintjein
  - A példányosítható leszármazottak már biztosan meg fogják valósítani az absztrakt metódusokat, így azokra lehet számítani

# Absztrakt osztály a gyakorlatban

- Ős referenciával hivatkozva leszármazottakra, a polimorfizmus segítségével csak a már az ősben is létező metódusokat tudjuk meghívni
- Néha emiatt szükség lehet az ősosztályban felvenni olyan metódusokat, amelyeket azon a szinten valójában még nem tudunk megvalósítani
- Erre alkalmasak az absztrakt metódusok és osztályok



# Lezárt osztályok

- Lezárt osztályok és lezárt metódusok lehetőséget teremtenek az öröklés megakadályozására
  - Lezárt metódus: leszármazottban nem definiálható felül
  - Lezárt osztály: nem származtatható belőle új osztály
- Használatának tipikus okai
  - Tervezési megfontolások
  - Biztonsági megfontolások
  - Teljesítmény optimalizálás
- Használatával a polimorfizmus megakadályozható
  - Tervezési döntés, hogy ennek használata indokolt vagy sem

# Osztály öröklődés

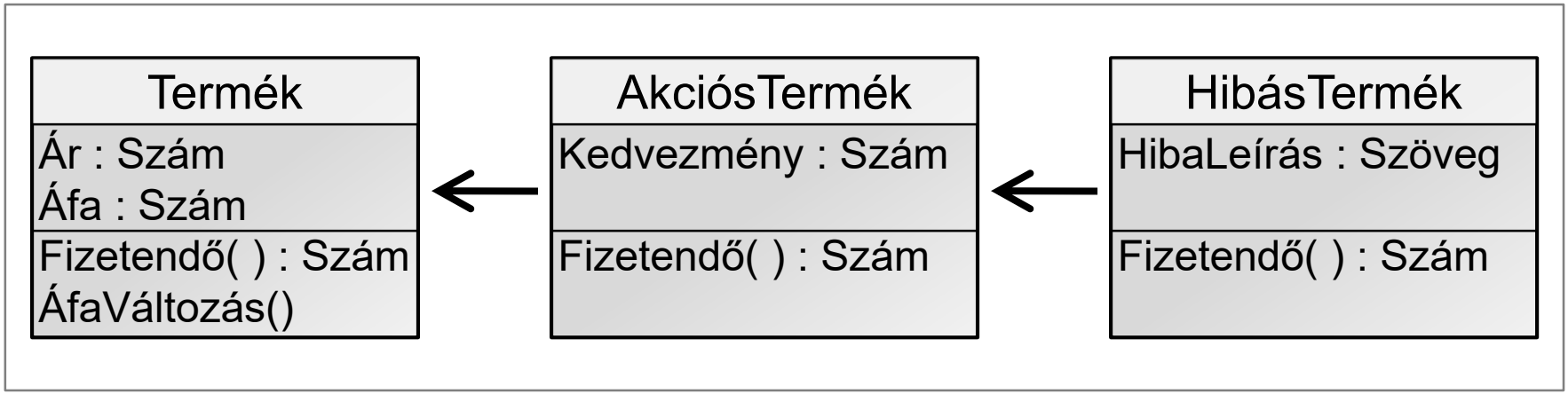
Öröklődés

Polimorfizmus

**Osztályok közötti kapcsolatok**

# Leszármazás

- Generalizáció – specializáció
- A hasonló tulajdonságokkal rendelkező osztályok közötti kapcsolatot írja le
- Lásd előző diák

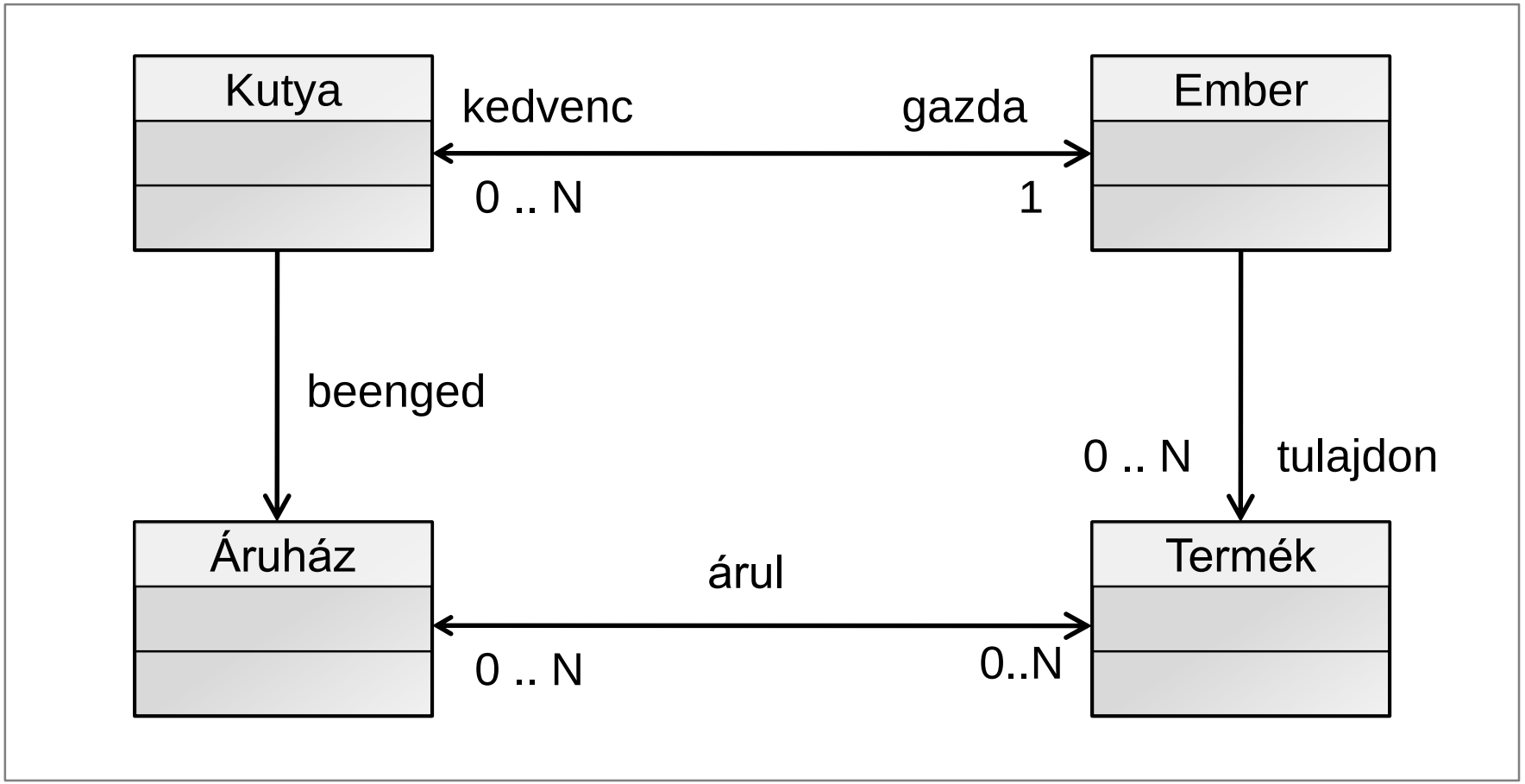


# Asszociáció

- Az asszociáció osztályok közötti tetszőleges viszony
  - Általában a neve jellemzi
  - Végpontoknál szerepek: az osztályok miként látják egymást
- Multiplicitása: a résztvevő objektumok száma
  - Egy-egy kapcsolat
  - Egy-több kapcsolat
  - Több-több kapcsolat
- Irányultsága: a kommunikáció irányát jelzi
  - Egyirányú kommunikáció
  - Kétirányú kommunikáció
- Asszociációs kapcsolat áll fenn két osztály között, ha az egyiknek a saját helyes működéséhez ismernie kell a másikat
  - Az egyik használja a másikat
  - Az egyik tartalmazza a másikat
  - Stb.

# Asszociáció példa

- Az asszociáció osztályok közötti tetszőleges viszony

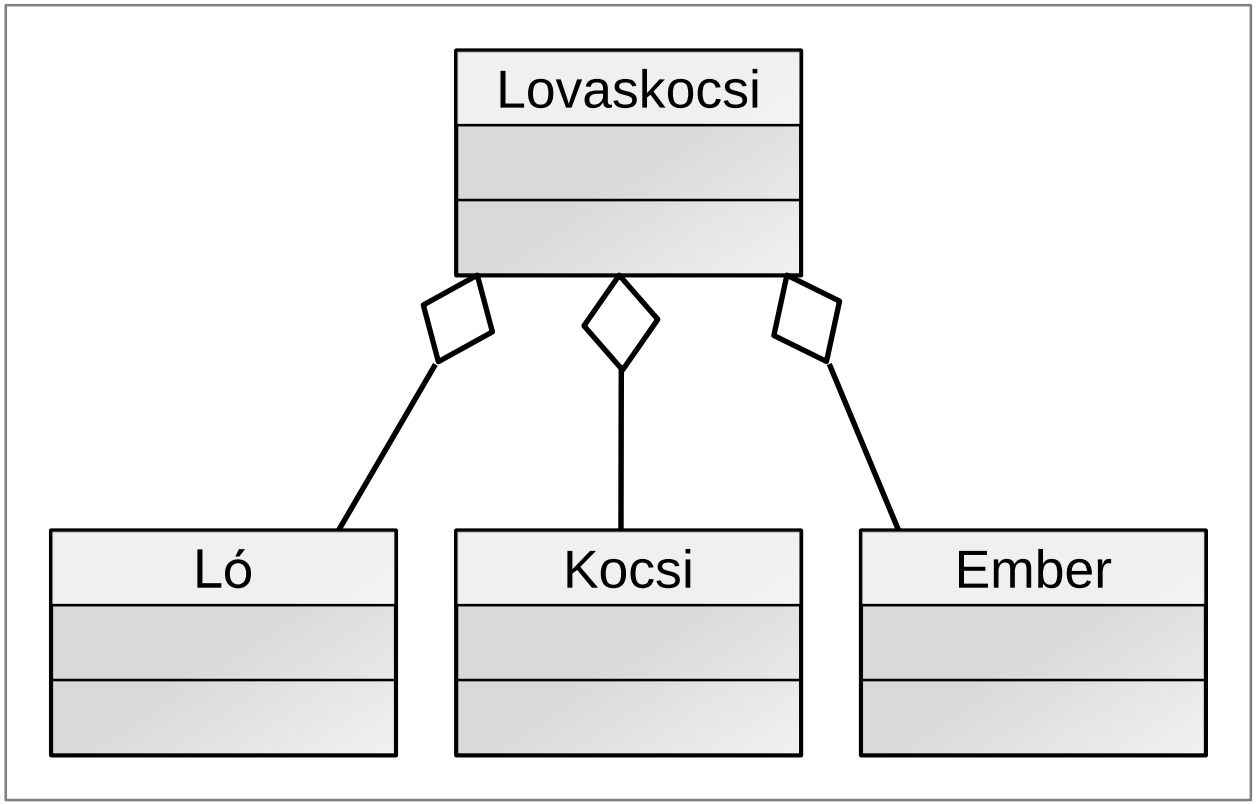


# Aggregáció

- Az aggregáció az asszociáció speciális esete: tartalmazási kapcsolat
  - A tartalmazó osztály példányai magukba foglalják a tartalmazott osztály egy vagy több példányát (rész-egész kapcsolat)
  - A tartalmazó és a tartalmazott osztály egymástól függetlenül létezhetnek
  - A tartalmazás lehet erős illetve gyenge
- A tartalmazó gyakran átveszi a tartalmazott egyes jellemzőit
  - Módosítva vagy módosítás nélkül
  - Bizonyos szempontokból az örökléshez hasonló állapot állítható elő az aggregáció segítségével
  - Azonban ez nem keverendő össze az örökléssel (technikai megvalósítás, láthatóság, polimorfizmus stb.)
  - Gyakran tervezési döntést igényel, hogy öröklés vagy aggregáció szükséges

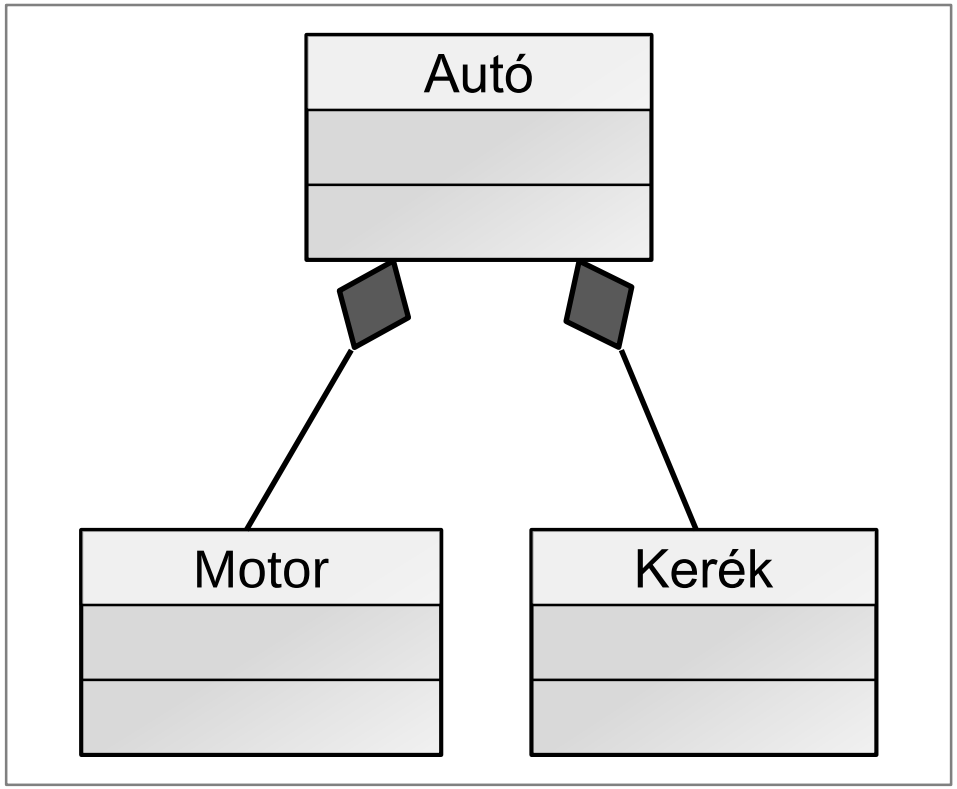
# Aggregáció példa

- Speciális asszociáció: tartalmazás



# Kompozíció

- A kompozíció az aggregáció speciális esete: szigorú tartalmazási kapcsolat
  - Egy rész objektum csak egy egészhez tartozhat
  - A tartalmazó és a tartalmazott életciklusa közös



# Függőség

- Két elem között akkor áll fenn, ha az egyik elem változása hatással van a másik elemre
- Ez nem feltétlenül jelen asszociációt
  - függvény paraméterek
  - Lokális változók
  - Statikus tagokra való hivatkozások

# Irodalomjegyzék

- Javasolt/felhasznált irodalom
  - Sergyán Szabolcs, Vámossy Zoltán és Miklós Árpád diasorozatai, Óbudai Egyetem, AAO tárgy
  - Dr. Kondorosi Károly, Dr. László Zoltán, Dr. Szirmay-Kalos László: Objektumorientált szoftverfejlesztés, ComputerBooks, 2006, Budapest
  - http://en.wikipedia.org/wiki/Virtual_method_table