1. Az alkatrészek kapcsolatos feladat módosítása

Egészítsük ki az előző órán megismert rendszert két új szolgáltatással:

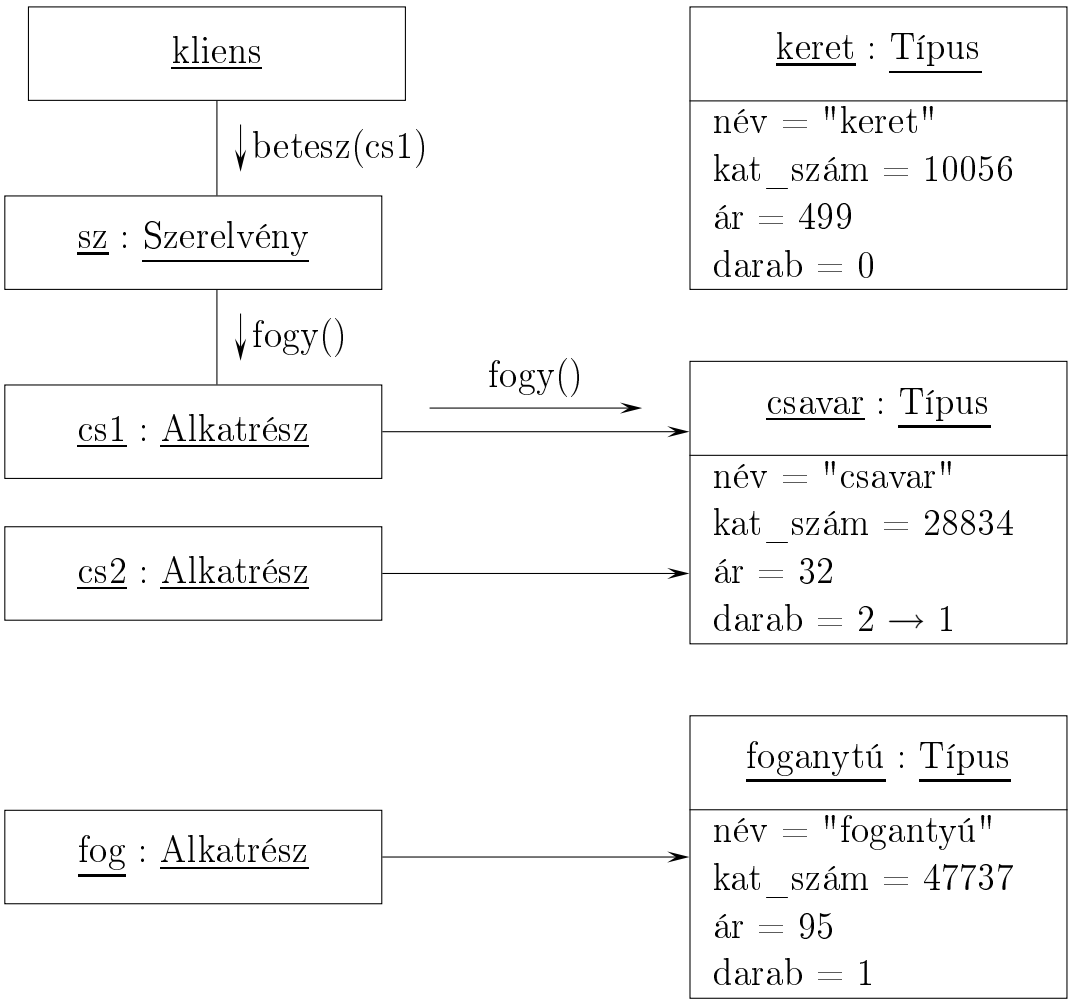
1. nyilván kell tartani azon alkatrészek számát, amelyek nincsenek beépítve valamilyen szerelvénybe;
2. egy szerelvényben szereplő összes alkatrészt ki kell listázni.

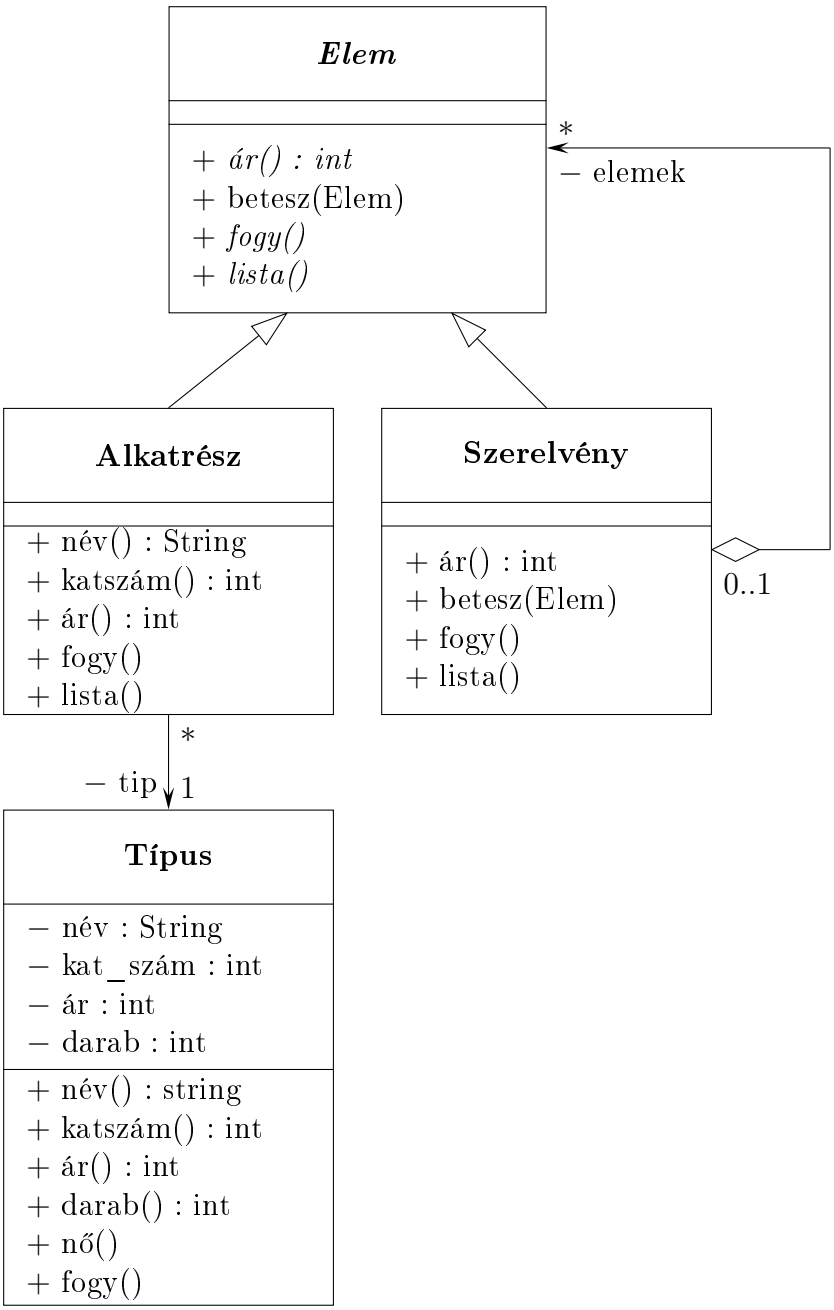
- Határozzuk meg, hogy milyen adatot és hol kell nyilvántartani!
- Adjuk meg milyen változtatások szükségesek a kódban!

A darabszám tárolása egyetlen objektumban, a típusban lehetséges, mivel egyedül az tartalmaz általános információkat a hozzátartozó egyedi alkatrészekről. Egy alkatrész beépítése esetén a darabszámot kell módosítani, amit megtehetünk, hiszen az alkatrész felől elérhető a típus. Ennek az új adattagnak a neve legyen `darab`, típusa `int`.

Az objektumdiagramban az üzenteküldést egy kliens indukálja azzal, hogy betesz egy alkatrészt egy szerelvénybe (`betesz()`). A szerelvény ezután egy csökkentő üzenetet (`fogy()`) küld az alkatrésznek, ami ezt továbbítja a típusnak. Ott ennek hatására eggyel csökken a `darab` értéke.

```
Típus      keret("keret", 10056, 499);
Típus      csavar("csavar", 28834, 32);
Típus      fogantyú("fogantyú", 47737, 95);
Alkatrész  cs1(csavar);
Alkatrész  cs2(csavar);
Alkatrész  fog(fogantyú);
Szerelvény sz;
sz.Betesz(cs1);
```

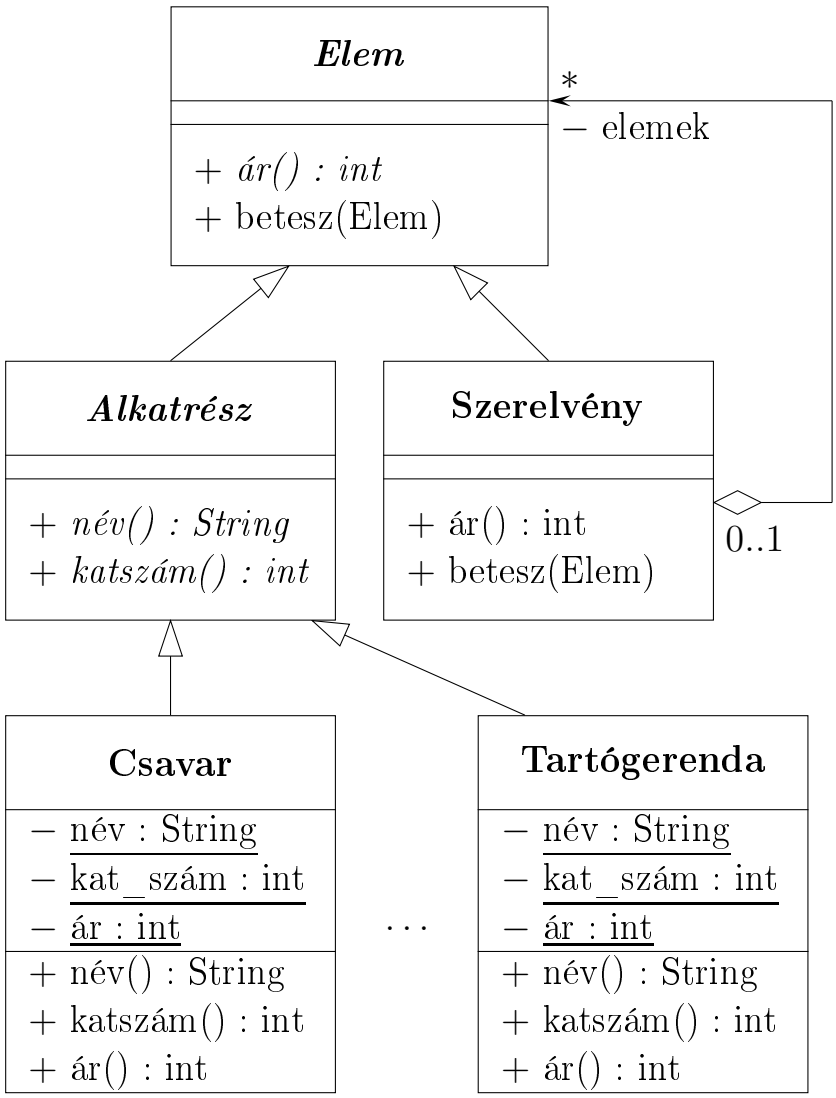

kliens
betesz(cs1)
sz : Szerelvény
fogy()
cs1 : Alkatrész
fogy()
cs2 : Alkatrész
fog : Alkatrész
keret : Típus
név = "keret"
kat_szám = 10056
ár = 499
darab = 0
csavar : Típus
név = "csavar"
kat_szám = 28834
ár = 32
darab = 2 → 1
foganytú : Típus
név = "fogantyú"
kat_szám = 47737
ár = 95
darab = 1


Elem
+ ár() : int
+ betesz(Elem)
+ fogy()
+ lista()
*
− elemek
Alkatrész
+ név() : String
+ katszám() : int
+ ár() : int
+ fogy()
+ lista()
Szerelvény
+ ár() : int
+ betesz(Elem)
+ fogy()
+ lista()
0..1
*
− tip
1
Típus
− név : String
− kat_szám : int
− ár : int
− darab : int
+ név() : string
+ katszám() : int
+ ár() : int
+ darab() : int
+ nő()
+ fogy()

Java program: `Raktár2` projekt

2. Alternatív terv

Egy másik tervezési megközelítése ugyanennek a problémának, ha elhagyjuk az alkatrészek és a típusok osztályát, és helyettük minden különböző típusú alkatrészhez külön-külön osztályt rendelünk. Így például lenne külön `Csavar`, `Tartógerenda` osztály. Minden osztályban a név, a katalógusi szám és az ár statikus adattag lenne, és az egyedi alkatrészek a megfelelő osztály példányai lennének.


Elem
+ ár() : int
+ betesz(Elem)
*
− elemek
Alkatrész
+ név() : String
+ katszám() : int
Szerelvény
+ ár() : int
+ betesz(Elem)
0..1
Csavar
− név : String
− kat_szám : int
− ár : int
+ név() : String
+ katszám() : int
+ ár() : int
...
Tartógerenda
− név : String
− kat_szám : int
− ár : int
+ név() : String
+ katszám() : int
+ ár() : int

Java progam: `Raktár3` projekt

A terv elemzése, összehasonlítása az eredeti tervvel:

Adattárolás: Az új terv szerint egy alkatrésztípus attribútumai egy osztály statikus adattagjaiként, tehát csak egy helyen kerülnek tárolásra. Minden alkatrész objektum a megfelelő osztály egy példánya, és nem tárolunk benne semmilyen adatot.

Ennek megfelelően az egyes alkatrésztípusok leírása ugyanakkora területet igényel mindkét esetben, viszont az új megközelítésben az egyes alkatrész objektumok egy mutatóval kevesebb helyet foglalnak el. Így az új terv valamivel kevesebb területet használ adattárolásra.

Osztályszerkezet: Az absztrakt `Elem` osztály ebben az esetben is szükséges, hiszen csak ennek használatával tudjuk elérni, hogy egy szerelvény részszerelvényeket is tartalmazhasson. Ennek megfelelően a `Szerelvény` osztály nem változik.

Ugyanakkor a különböző alkatrésztípusoknak megfelelő osztályokat az `Elem` osztályból kell származtatni, hogy egy szerelvénybe bekerülhessenek. Az alkatrészek egységes kezelése miatt vezettük be az absztrakt `Alkatrész` osztályt.

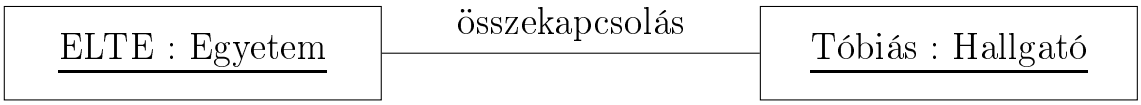
Új alkatrésztípus megjelenése: Egy új alkatrésztípus megjelenésekor az eredeti terv esetén egy új típus objektumot kell létrehozni a megfelelő értékekkel. Ezt akár egy kliens is megteheti futási időben. Az új terv esetén a programkódot ki kell egészíteni egy új osztály definiálásával, és ezután újra kell fordítani.

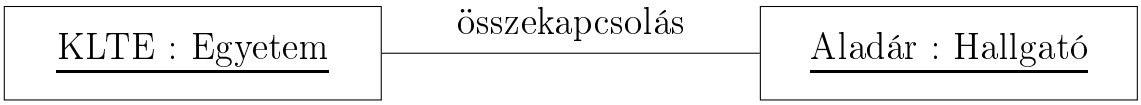
Melyiket használjuk: Egy program módosítása mindig új hibák bevezetésének forrása lehet, ezért ezt lehetőleg el kell kerülni. Ennek megfelelően az új terv csak abban az esetben elfogadható, ha az alkatrésztípusok előre ismertek, új alkatrésztípus felvételének a valószínűsége elenyésző; ugyanakkor a program futási környezetének adattároló kapacitása szűkös. Minden más esetben az eredeti tervet javasoljuk.

3. Asszociáció

Ez a legáltatalánosabb reláció két osztály között. Az asszociáció két osztály közötti absztrakt reláció, amely kétirányú társítást fejez ki. A reláció absztrakt volta azt jelenti, hogy a reláció konkretizálása osztályok objektumainak összekapcsolásában valósul meg.

Konkrét esetben összekapcsolás.





Absztrakt esetben társítás.

3.1. Az asszociáció informális definíciója

1. Asszociáció: két vagy több osztály objektumainak valamilyen relációval történő *összekapcsolása*, azaz:

$$\begin{aligned} \text{asszociáció}(C_1, C_2) \;=\; & \{(obj_1, obj_2) \mid \\ & obj_1 \in C_1 \wedge obj_2 \in C_2 \wedge rel(obj_1, obj_2)\}. \end{aligned}$$

2. Az asszociáció lehet *reflexív*, azaz azonos osztályon belüli objektumok összekapcsolását is megengedi.

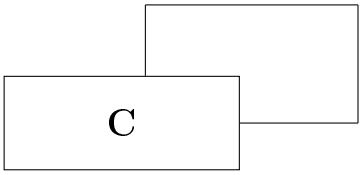
3. Az asszociációhoz társulhat annak neve, *azonosítója*.

4. Az asszociáció részleteinek leírása a hozzá *társult osztályban* kaphat helyet.

5. Az összekapcsolt objektumoknak lehet *multiplicitása* is:
   - pontosan i, jele: i;
   - i és j közötti, jele: $i..j$;
   - 0 vagy több, azaz valamennyi, jele: $*$;
   - legalább i, jele: $i..*$.
6. Az asszociációban részt vevő objektumnak lehet *szerepe* is:
   - névvel azonosított szerep,
   - kiemelt szerep,
   - sorrendiségi szerep.
7. Az asszociációhoz *minősítő* társulhat, amelynek értékei az osztály objektumait a társítás szempontjából diszjunkt partíciókhoz rendelik.
8. Az asszociáció esetén megadhatjuk a *navigálhatóságot*. Előfordulhat, hogy a társított osztályok objektumai nem ismerik egymást kölcsönösen, csak az egyik osztály objektumai érhetik el a másik osztályba tartozó objektumokat. Ezt fejezhetjük ki ezzel a tulajdonsággal. Ha nem tüntetjük fel, akkor kölcsönös elérhetőséget tételezünk fel.

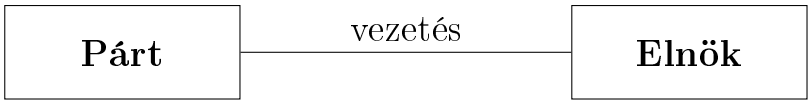
3.2. Az asszociáció jelölése:

A reflexív asszociáció jelölése:



Példa:

Tekintsük például a pártokat, amelyeket egy osztályba szervezünk, és a pártok elnökeit, akiket egy másik osztályba szervezünk! A két osztály objektumait a *vezetés* reláció kapcsolja össze. Feltesszük, hogy minden pártnak pontosan egy elnöke van, és egyvalaki csak egy pártnak az elnöke lehet. Az osztálydiagram:



3.3. A multiplicitás jelölése

A relációban az osztályok objektumai közül általában több példány is részt vehet, azaz:

$$\text{asszociáció}(C_1^m, C_2^n), \qquad m \geq 0 \wedge n \geq 0 \ .$$

A lehetséges eseteket és a megfelelő jelöléseket vizsgáljuk egy-egy példán keresztül a következőkben. A példában a *személy* és a *telefonszám* osztályokat kapcsoljuk össze asszociációval, azaz C_1 a személyek osztálya, C_2 pedig a telefonszámoké.

$m = 1 \wedge n = 1$:

$(m = 0 \vee m = 1) \wedge n = 1$:

$m \geq 0 \wedge n = 1$:

$m \geq i \wedge n = 1$:

$m = i \wedge n = 1$

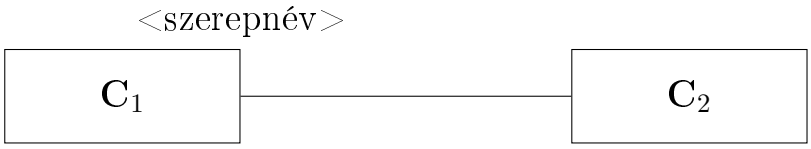
$i \leq m \leq j \wedge n = 1$:

3.4. Az objektum szerepének jelölése

Az informális definíció során említettük, hogy az asszociációval összekapcsolt osztályok objektumai különféle szerepeket tölthetnek be. A szereppel kapcsolatban az alábbi fogalmak jelölését kell bemutatni:

- a szerep megnevezése,
- kiemelt szerep,
- sorrendiségi szerep,
- több szerep megnevezése.

A szerep megnevezése:

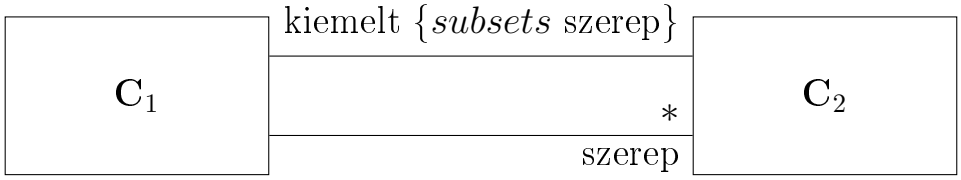


Tekintsük példaként a cégek és a személyek kapcsolatát! A két osztály a *cég* és a *személy*, amelyek asszociációban állnak egymással aszerint, hogy mely cég kit alkalmaz. Az *alkalmaz* relációban a cégek az *alkamazó*, a személyek pedig az *alkalmazott* szerepet töltik be. A multiplicitástól tekintsünk most el!

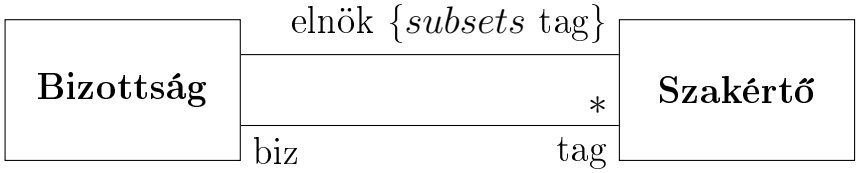
```
public class Cég
{
   ...
   private Személy alkalmazott;
   ...
}

public class Személy
{
   ...
   private Cég alkalmazó;
   ...
}
```

Kiemelt szerep:

Ebben az esetben az osztály egy vagy több objektuma a relációban, a többitől eltérő módon, más szerepet is betölt.



Konkrét példaként tekintsük a szakértői bizottságokat! Ekkor a bizottságok osztálya kapcsolatba hozható a szakértők osztályával. A szakértők szerepe ebben a relációban a tagság, de van minden bizottságnak egy olyan tagja, aki egyben elnök is.



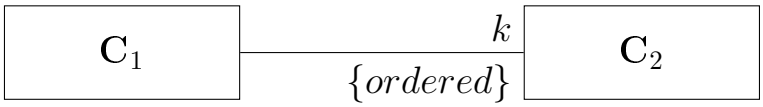
```
public class Szakértő
{
   ...
   private Bizottság biz;
   ...
}

public class Bizottság
{
   ...
   private java.util.Vector<Szakértő> tag;
   private Szakértő                   elnök;
   ...
}
```

vagy

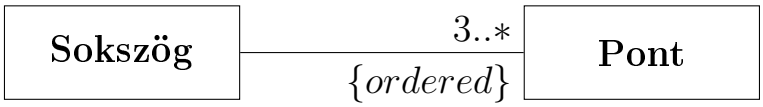
```
public class Bizottság
{
   ...
   private java.util.Vector<Szakértő> tag;
   private int                        elnök;  // index a tag sorozatban
   ...
}
```

Sorrendiségi szerep

Ebben az esetben megadjuk, hogy hány objektum veszt részt a relációban, és az *ordered* alapszó feltüntetésével jelezzük, hogy ezek kötött sorrendben vesznek abban részt.

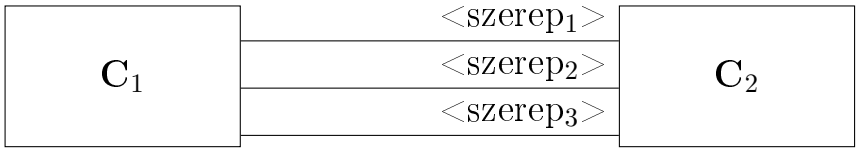


Tekintsük példaként a sokszögeket és a pontokat! A két osztály objektumai közötti kapcsolat kifejezhető, ha a sokszög csúcsait valamilyen sorrendben (óramutató járásával ellenkező irányban) adjuk meg. Egy sokszöghöz legalább három csúcspont tartozik.



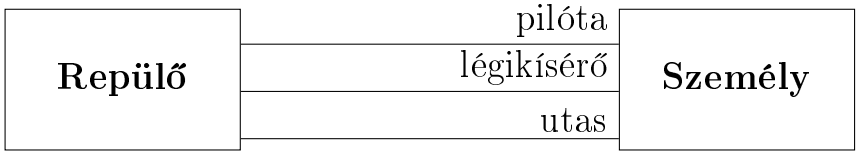
Több szerep megjelölése

Ebben az esetben a két osztály közötti asszociációt több vonallal szemléltetjük, és az egyes vonalakra ráírjuk a megfelelő szerepeket.

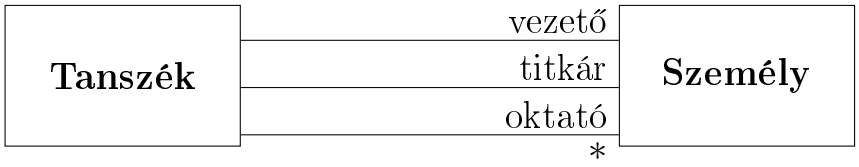


Ha az áttekinthetőség úgy kívánja, másik elrendezést is használhatunk.

Tekintsük a repülők és a személyek osztályát! A személyek többféleképpen is kapcsolatba hozhatók egy repülővel: lehet köztük pilóta, légikísérő vagy utas.



Egy másik példában tekintsük az egyetemi tanszékeket! Egy tanszéknek egy vezetője és egy titkára van, továbbá több oktató dolgozik ott, de ezek mindegyike személy.

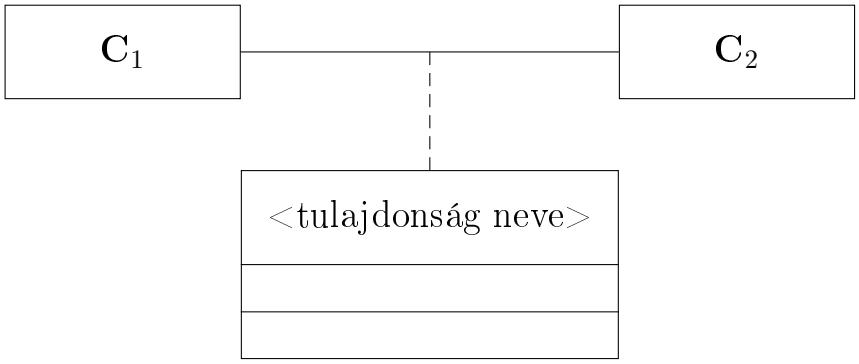


```
public class Tanszék
{
   ...
   private Személy                   titkár;
   private Személy                   vezető;
   private java.lang.Vector<Személy> oktató;
   ...
}

public class Személy
{
   ...
   private Tanszék     hely;
   ...
}
```

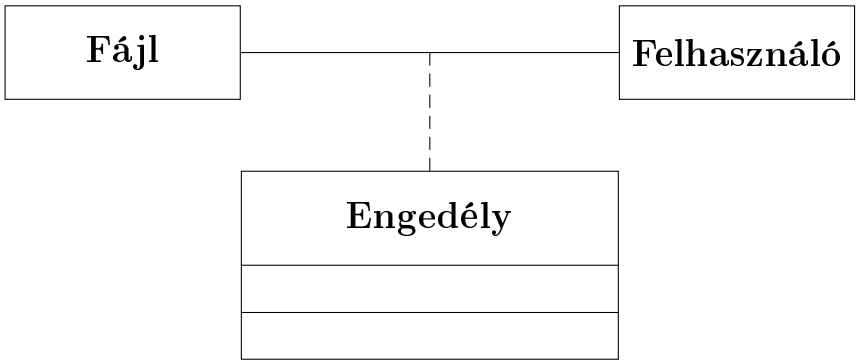
3.5. Az asszociációhoz kapcsolódó további jelölések

A reláció tulajdonságainak megadása

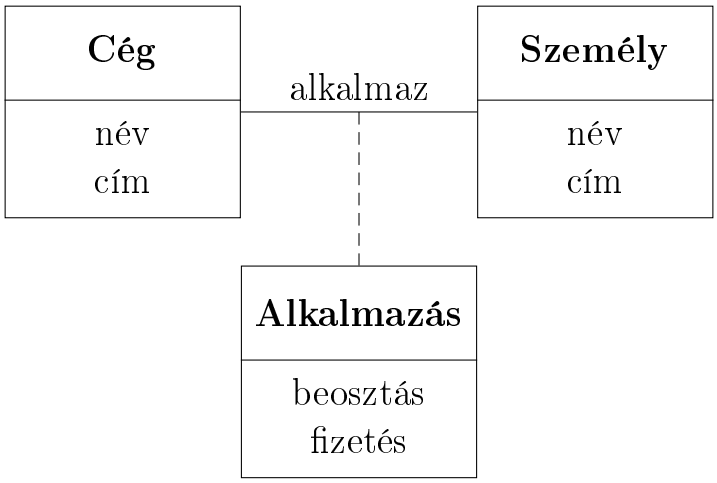
A reláció tulajdonságait, a relációra vonatkozó korlátozásokat rendszerint nem tudjuk elhelyezni a reláció mellett, noha rendkívül fontos lenne a feltüntetésük, jelölésük. Ezeket ilyenkor egy külön osztályban adjuk meg.



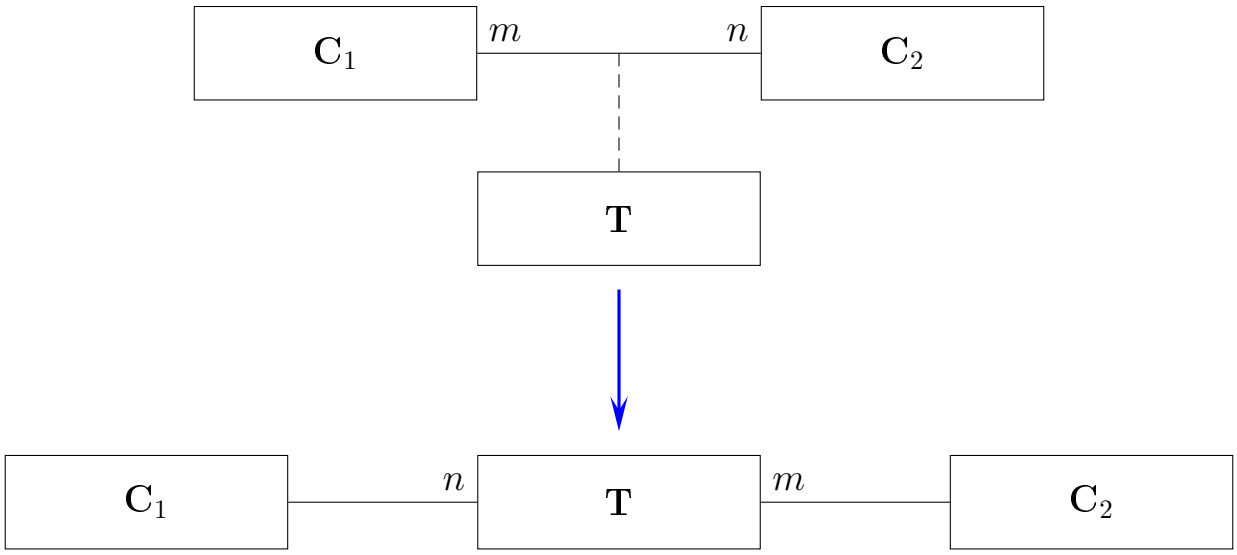
Példaként tekintsük a fájlokat és a felhasználókat! A két osztály objektumai közötti kapcsolat fontos jellemzője az engedély, amely a felhasználók hozzáférését szabályozza a fájlokhoz.



Az osztály bizonyos attribútumait, amelyek a relációhoz köthetők, a reláció tulajdonságaként tüntethetjük fel egy osztály keretében. Így mentesítjük az osztályok közötti kapcsolatot a terjedelmes felsorolástól. Erre példa a cégeknél dolgozó alkalmazottak alkalmazásukhoz köthető tulajdonságainak, adatainak, külön osztályban történő megadása.

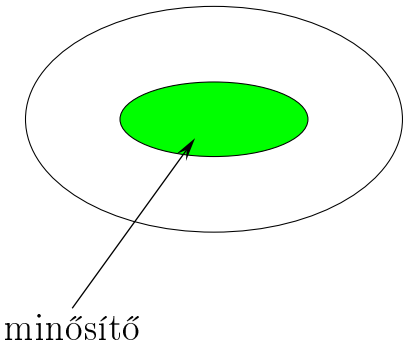


Társult osztály kiküszöbölése hagyományos módon (megvalósítás is):

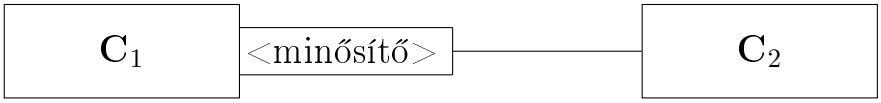


A reláció minősítése

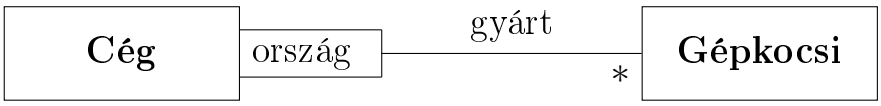
A minősítő konkrét értékei a relációban részt vevő konkrét objektumok egy példányát, részhalmazát azonosítják.



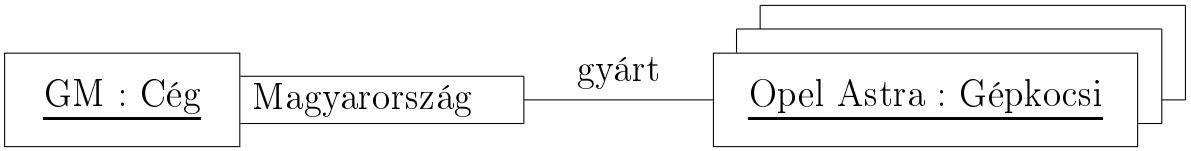
A diagramokban alkalmazott jelölés:



Példaként a multinacionális cégeket említjük meg, ahol a minősítő azt az országot azonosítja a relációban, amelyben a szóban forgó cégek működnek. Ilyen példa a gépkocsik és multinacionális gyártóik közötti reláció

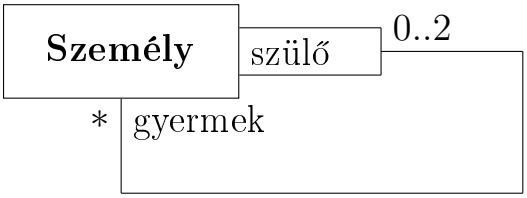


Ennek egy konkrét esete:



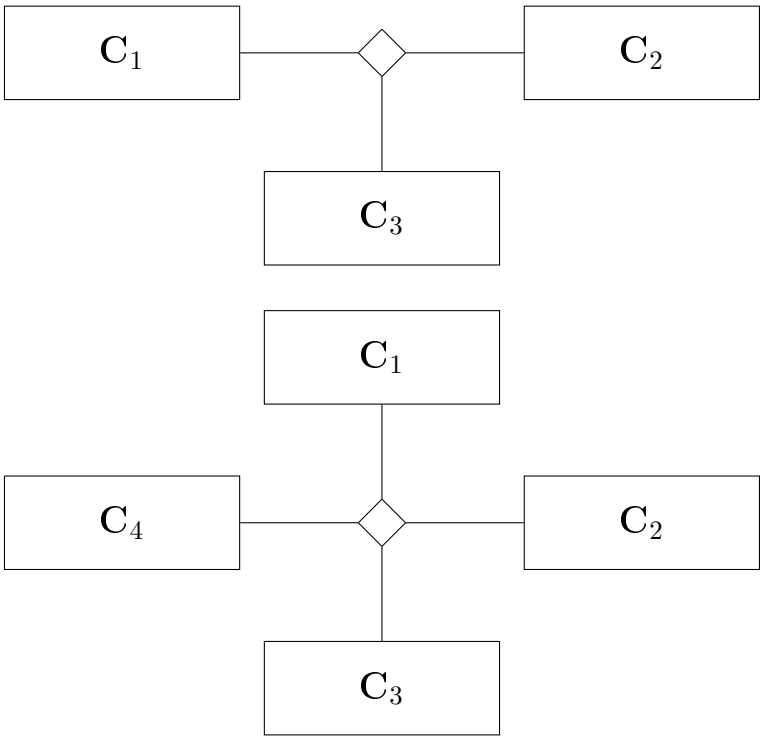
Reflexivitás

Példa reflexív asszociációra:

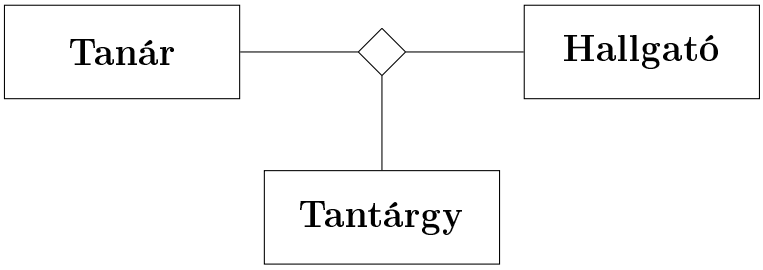


Több osztály között fennálló asszociáció

Az asszociáció nem feltétlen két osztályt kapcsol össze. Előfordulhat, hogy több osztályt hozunk kapcsolatba. Ekkor ezt a vonalak metszéspontjában elhelyezkedő rombusszal szemléltetjük.



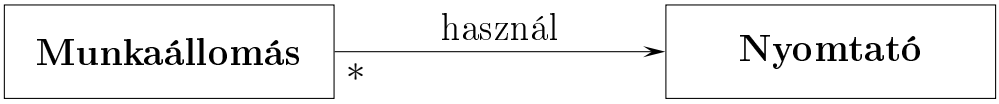
Példa három osztály közötti asszociációra a tanárok, hallgatók és tantárgyak között fennálló kapcsolat.



Navigálhatóság

Említettük, hogy a társított osztályok objektumai nem feltétlen ismerik egymást kölcsönösen. Ha az ismeret csak egyirányú, akkor lehetőségünk van ennek kifejezésére, amivel az implementáció számára is útmutatást adunk.

Tegyük fel, hogy egy rendszerben több munkaállomásról lehet egy nyomtatót használni! Ekkor a nyomtatónak nem feltétlen kell ismernie a munkaállomásokat, elég a munkaállomásoknak elérniük a nyomtatót, hiszen így ki tudják nyomtatni a kívánt dokumentumokat.



3.6. Példa: Étkező filozófusok

A problém a következő: n filozófus ül egy kör alakú asztal körül, és van n villa az asztalon, bármely két szomszédos filozófus között pontosan egy. Minden filozófus egy bizonyos ideig gondolkodik; ezután megpróbálja felvenni a tőle balra és jobbra eső villát, és ha ez sikerül, eszik, majd leteszi a villákat.

Az eddig elmondottak alapján két osztályt azonosíthatunk:

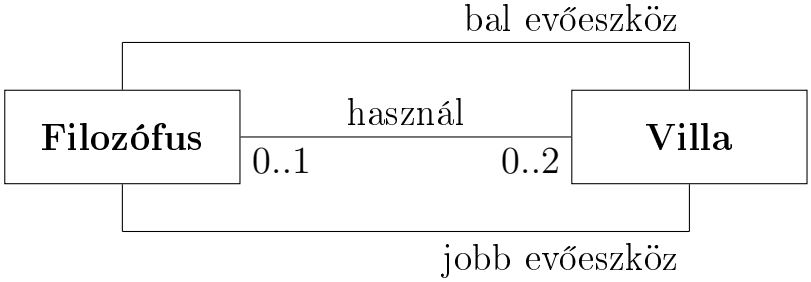
- a *filozófusok* és
- a *villák* osztályát.

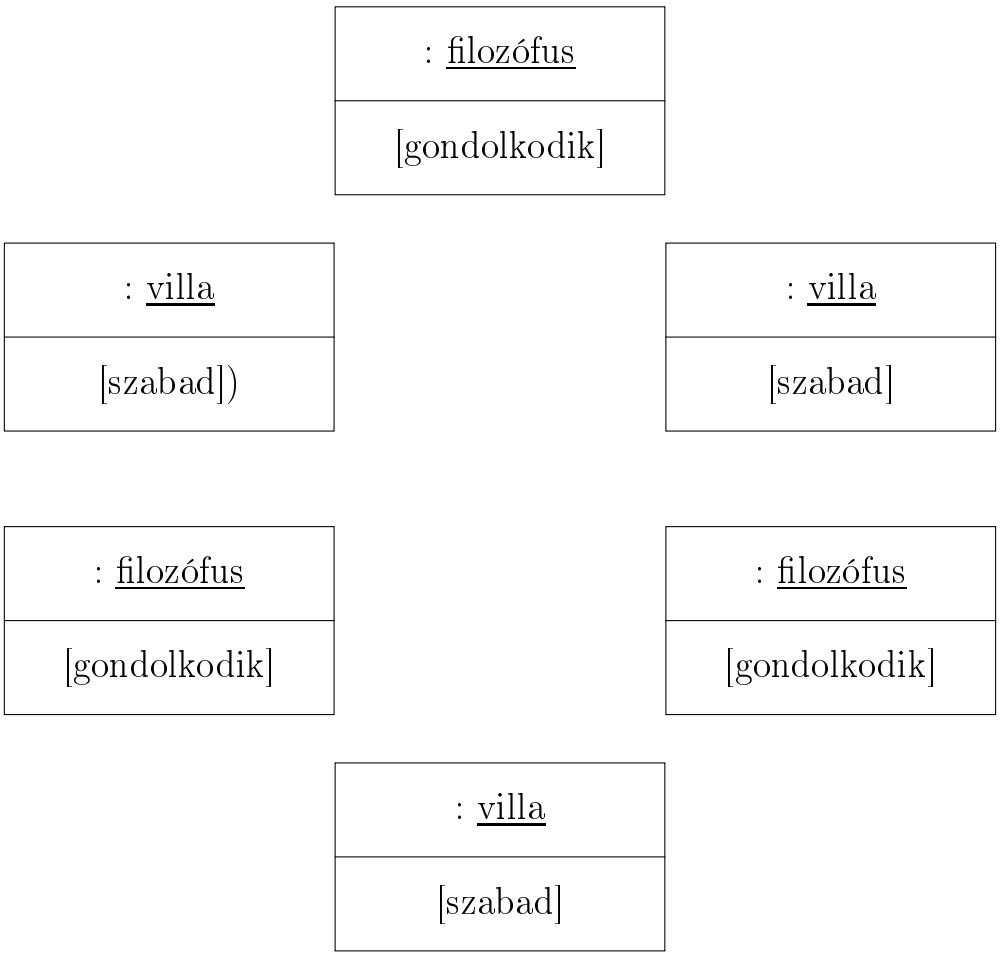
A köztük fennálló reláció: a filozófus *használja* a villát.

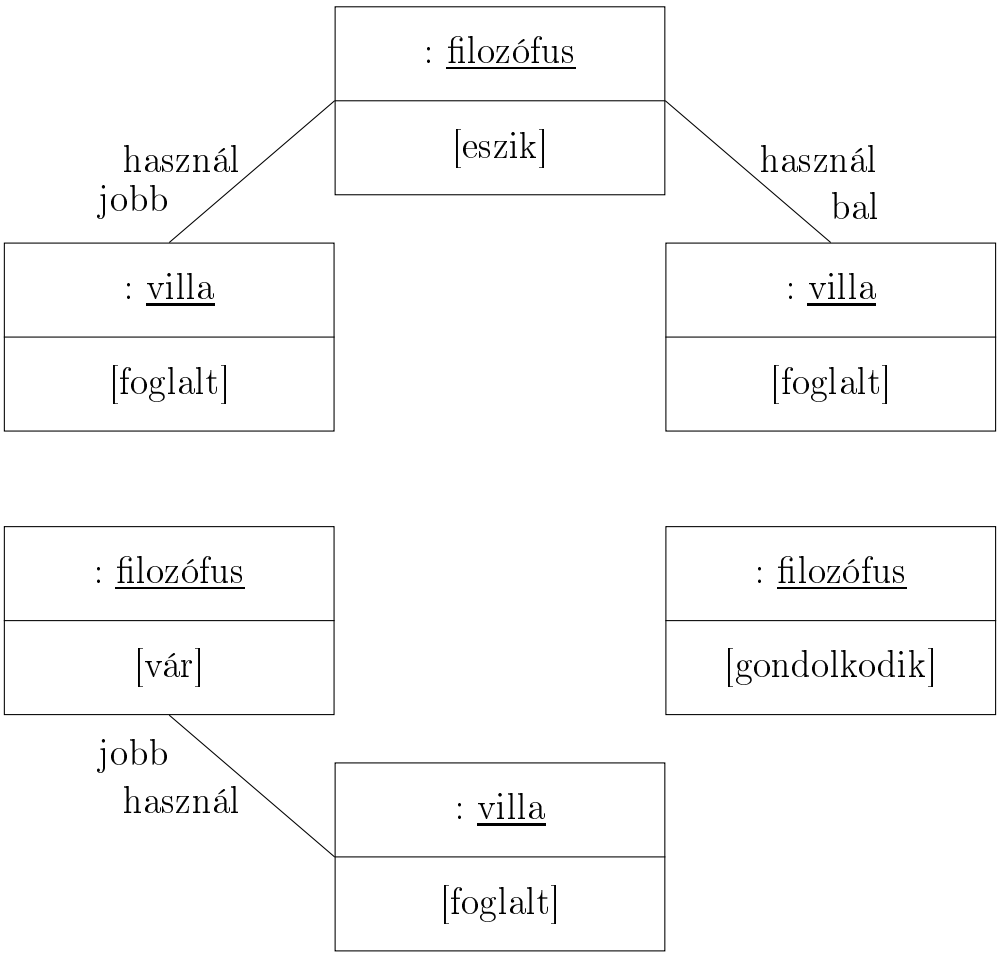
Vizsgáljuk meg a multiplicitás kérdését! Ezt megtehetjük, ha megvizsgáljuk a filozófus használatában levő villák számát a tevékenységei során:

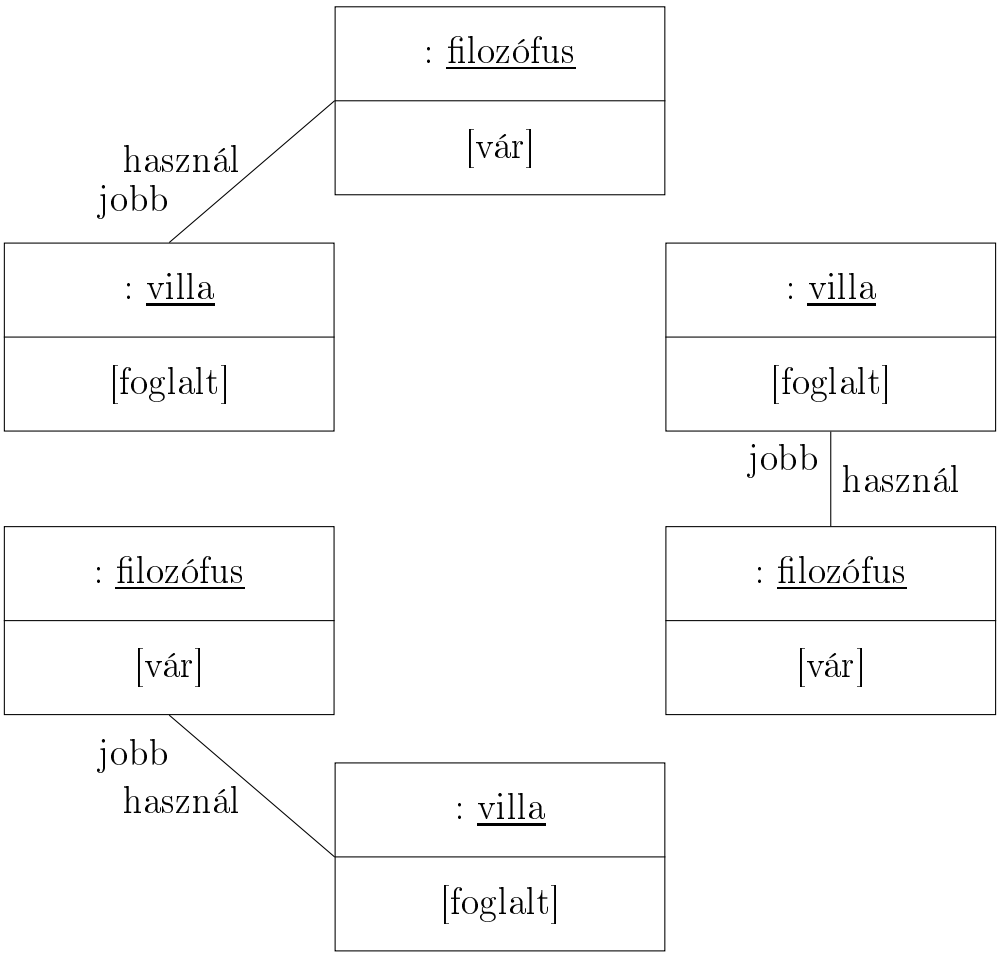
tevékenység	villák száma
gondolkodik	0
egyik villát felvette (vár)	1
mindkét villát felvette (eszik)	2

Tehát a filozófus objektumok közül 0 vagy 1, a villa objektumok közül 0, 1, vagy 2 vesz részt a kapcsolatban. Az ennek megfelelő osztálydiagram:




: filozófus
[gondolkodik]
: villa
[szabad])
: villa
[szabad]
: filozófus
[gondolkodik]
: filozófus
[gondolkodik]
: villa
[szabad]


: filozófus
[eszik]
használ
jobb
használ
bal
: villa
[foglalt]
: villa
[foglalt]
: filozófus
[vár]
: filozófus
[gondolkodik]
jobb
használ
: villa
[foglalt]


: filozófus
[vár]
használ
jobb
: villa
[foglalt]
: villa
[foglalt]
jobb
használ
: filozófus
[vár]
: filozófus
[vár]
jobb
használ
: villa
[foglalt]

4. Aggregáció

Az asszociáció az osztály objektumainak egymáshoz rendelését fejezi ki. Az egymáshoz rendelés különböző erősségű kapcsolódást jelenthet. Az asszociációt általában egymástól független osztályok társításának a kifejezésére használjuk.

Az aggregáció ennél erősebb kapcsolat, amely olyan jellegű kapcsolatokat fejez ki, mint:

- egész és annak részei,
- felépítmény és annak komponensei.

4.1. Az aggregáció informális definíciója

1. Az aggregáció egy *speciális asszociáció.*

2. Az aggregációs reláció azt fejezi ki, hogy az egyik osztály objektumai részét képezik egy másik osztály objektumainak:

$$A \text{ is an aggregation of } B \;=\; \{(a,b) \mid \\ a \in A, b \in B, b \text{ is a part of } a\}.$$

3. Az aggregáció *tranzitív*:

$$A \text{ is an aggregation of } B \wedge B \text{ is an aggregation of } C \rightarrow \\ A \text{ is an aggregation of } C.$$

4. Az aggregáció *aszimmetrikus*:

$$A \text{ is an aggregation of } B \rightarrow \neg(B \text{ is an aggregation of } A).$$

5. Az aggregáció lehet reflexív.

6. Ha A és B osztályok között aggregációs kapcsolat áll fenn, akkor az A osztály objektumai és a B osztály objektumai *egymástól függetlenül is létezhetnek.*

7. Különböző aggregátumoknak lehetnek közös komponensei.

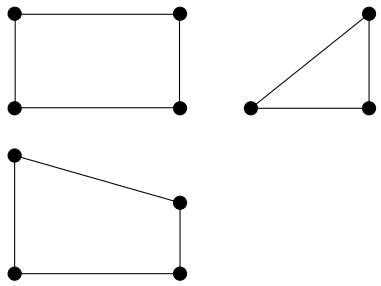
4.2. Az aggregáció jelölése

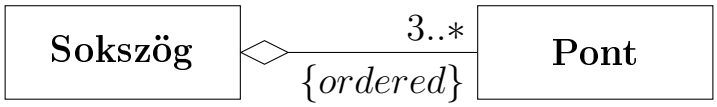
4.3. Aggregáció megvalósítása

A megvalósítás ugyanúgy hivatkozások, mutatók segítségével lehetséges, mint asszociáció esetén.

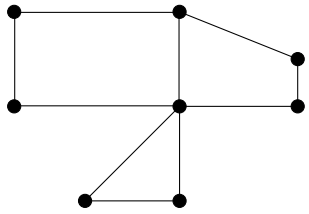
4.4. Példa: Sokszögek

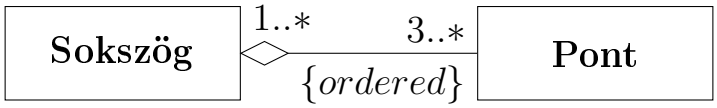
Vizsgáljuk meg a síkbeli sokszögek és a sík pontjainak viszonyát! A sokszögek csúcsai pontok méghozzá megfelelő sorrendben, így a két osztály között aggregációs kapcsolat áll fenn. Tegyük fel, hogy egy síkbeli pont csak egy sokszöghöz tartozhat, és csak ilyen pontokat veszünk figyelembe.





Tekintsünk most el az előbbi egyszerűsítéstől, azaz most már megengedjük, hogy egy pont akárhány sokszöghöz tartozzon.





5. Kompozíció

Az aggregációs társítások között a legerősebb kapcsolat a kompozíciós kapcsolat.

A kompozíció informális definíciója:

1. A kompozíció egy *speciális aggregáció*.
2. A kompozíciós kapcsolat azt fejezi ki, hogy az egyik osztály objektumai a másik osztály objektumait *fizikailag tartalmazzák*:

$$A \text{ composition of } B = \{(a, b) \mid a \in A, b \in B, a \text{ contains } b\}.$$

3. A kompozíciós kapcsolat és az attribútum jellegű kapcsolat két objektum között szemantikailag azonos, csupán grafikai megjelenítésük különböző. Az attribútum jelölése valójában a kompozíció jelölésének egy leegyszerűsített formája.
4. Egy komponens objektum legfeljebb csak egy aggregációs objektumhoz tartozhat.

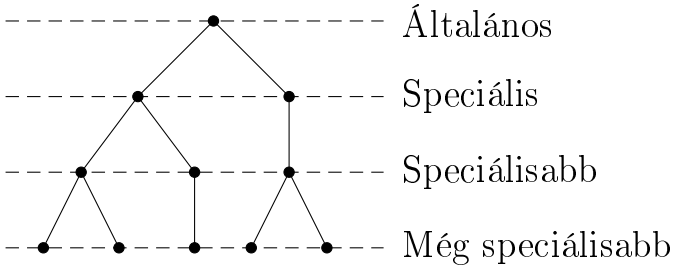
5. A kompozíciónak tetszőleges számú kompozíciója lehet.

6. A kompozíciós objektum és annak komponensei azonos életciklusban léteznek, azaz egyszerre jönnek létre és egyszerre szűnnek meg. (Az összetett objektum felel a beágyazott objektum létrehozásáért és megszüntetéséért, csak az összetett objektum kezelheti azt.)

5.1. A kompozíció jelölése

6. Általánosítás és specializáció (öröklődés)

Az általánosítás a modellalkotásban az általános tulajdonságokkal rendelkező dolog (superclass, ősosztály) és a kevésbé általános, speciálisabb dolog (subclass, származtatott osztály) között fennálló reláció. Az általánosítás egy klasszifikációs megközelítés. Ennek az a lényege, hogy először létrehozzuk az általános tulajdonságokkal felruházott osztályt, majd annak a tulajdonságait átvéve származtatjuk a speciálisabb tulajdonságokkal rendelkező osztályt. Ezt az eljárást tovább folytatva az osztályokat egy hierarchikus szerkezetbe rendezzük. Többszörös öröklődést is megengedve az osztályokat egy fastruktúrába rendezhetjük. A fában a gyökérelemtől távolodva bonyolultsági szinteket hozhatunk létre.



6.1. Az öröklődési reláció definíciója

1. A reláció egy általános konstrukció és egy speciális konstrukció közötti kapcsolatot fejez ki.

2. A reláció azt fejezi ki, hogy a speciális osztály az általános osztályból *származtatással* jön létre.

3. A származtatásnál a származtatott speciális osztály:

   - átveszi az általános osztály tulajdonságait (név, attribútum, operáció, asszociáció);
   - az átvett jellemzőkhöz, attribútumokhoz, operációkhoz stb. bővítésként új jellemzőket vezethet be;
   - az átvett jellemzőket újrafogalmazhatja, de úgy, hogy a speciális osztály objektumai az általános osztály helyére is behelyettesíthetők lehessenek:

     $$b \in B \wedge B \text{ is a specialization of } A \rightarrow b \in A \ .$$

4. A származtatás nem szimmetrikus.

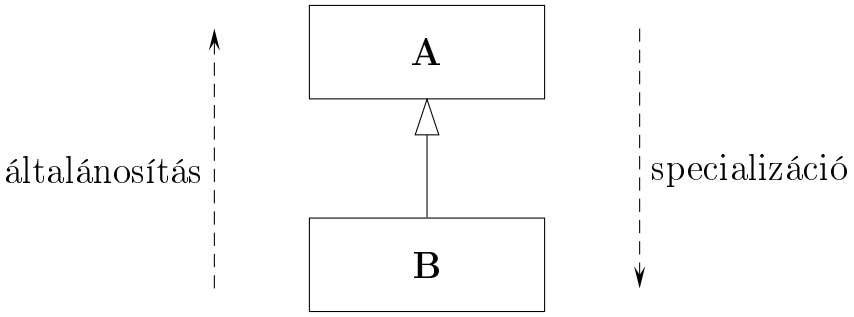
5. A származtatás nem lehet reflexív.

6. Az általánosítás és specializáció új osztályok létrehozásának egy technikája, amely absztrakt és konkrét osztályok létrehozására egyaránt alkalmas.

7. A specializáció lehet többszörös, ekkor egy általánosításból több származtatott jön létre.

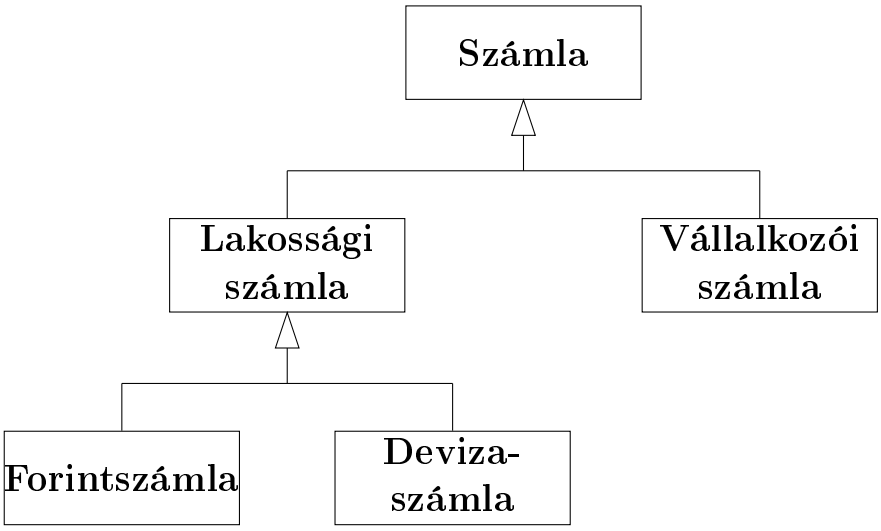
8. Az általánosítás is lehet többszörös, amikor a származtatás több általánosításból történik.

6.2. Az átalánosítás és a specializáció jelölése



6.3. Példa: Számlák

Tekintsük a bankszámlákat, röviden számlákat! Ezeket tovább lehet bontani lakossági és vállalkozói számlákra. A lakossági számlákon belül megkülönböztethetünk forint és deviza számlákat.



7. Feladat

Készítsünk programot, amellyel testek térfogatát határozhatjuk meg, illetve megadhatjuk azt is, hogy az egyes testfajtákból hány objektum létezik! A lehetséges fajták:

- szabályos sokszögek: gömb, kocka, tetraéder, oktaéder;
- hasáb jellegű testek: henger, négyzet alapú hasáb, szabályos háromszög alapú hasáb;
- gúla jellegű testek: kúp, négyzetes gúla.

7.1. A megoldás menete

A feladatot az öröklődés felhasználásával oldjuk meg. Az absztrakt `Test` osztály, adja meg a testek közös jellemzőit.

Ebből származtatjuk az absztrakt `Szabályos` osztályt, amely a szabályos testek tulajdonságait tartalmazza. Ennek konkrét esetei lesznek a `Gömb`, `Kocka`, `Tetraéder`, `Oktaéder` osztályok.

A `Hasáb` absztrakt osztály szintén a `Test` osztály leszármazottja, és az alapterület és magasság szorzatával kiszámítható térfogatú testeket írja le. Ennek speciális esete a `Gúla`, amikor a szorzatot $\frac{1}{3}$-dal kell megszorozni.

A `Hasáb` osztály konkrét esetei: `Henger`, `Négyzetes` (négyzet alapú hasáb), `Háromszöges` (szabályos háromszög alapú hasáb).

A `Gúla` osztály konkrét esetei: `Kúp` és `NGúla` (négyzet alapú gúla).

7.2. Közös tulajdonságok

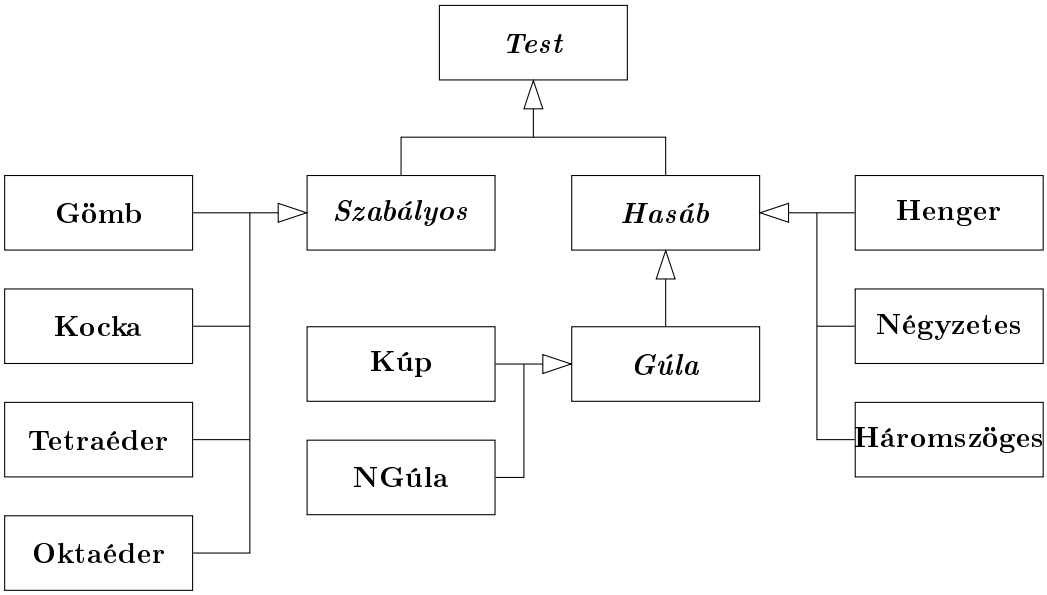
Minden test jellemzője egy *méret*, ami szabályos sokszögek esetén a sugarat, illetve az oldal hosszát jelenti. Egyéb esetben (hasábok) az alapot meghatározó szabályos síkidom meghatározó adata, amit ekkor a *magassággal* kell kiegészítenünk.

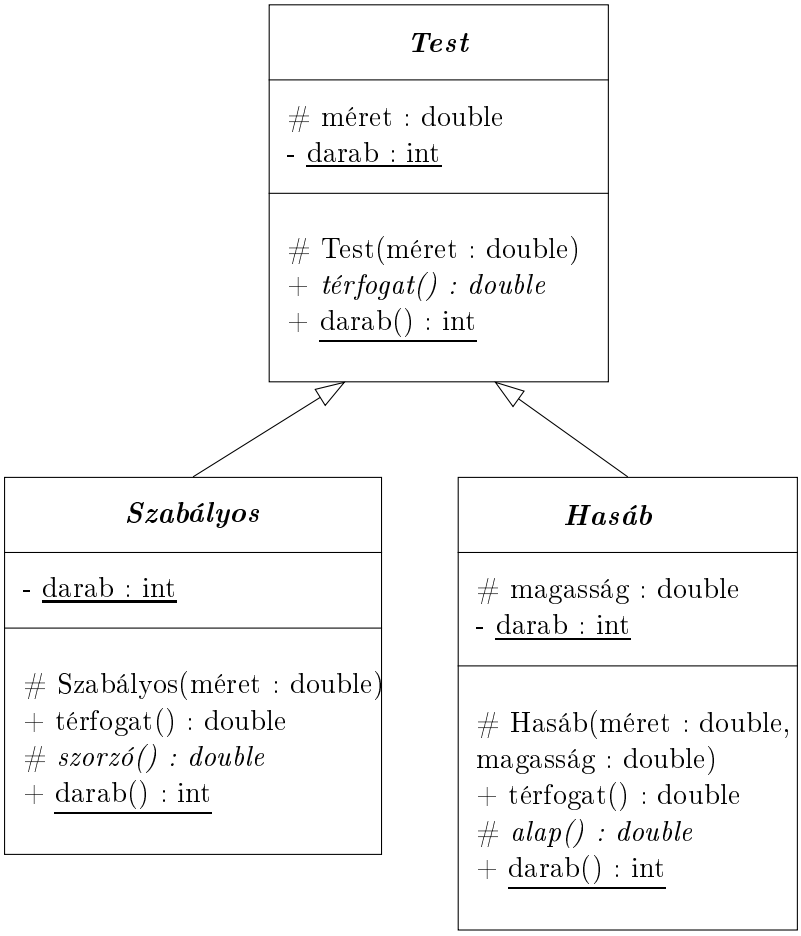
Ezen kívül minden osztályban nyilván kell tartanunk az osztály objektumainak a számát. Ez egy osztályszintű adattag lesz.

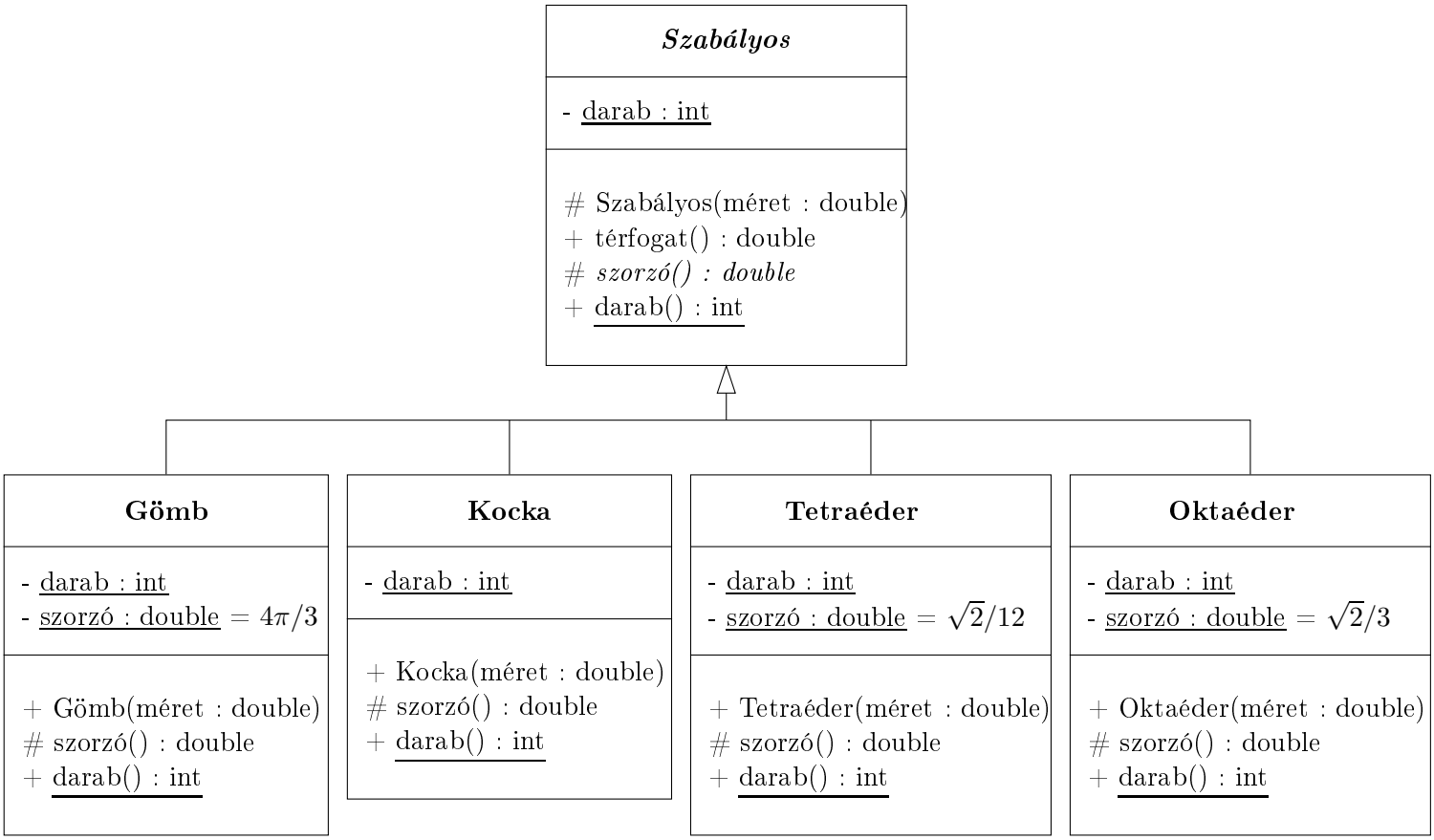
A közös művelet a térfogat kiszámítására szolgáló függvény (`térfogat`), illetve az objektumok számát lekérdező osztályszintű `darab` függvény.

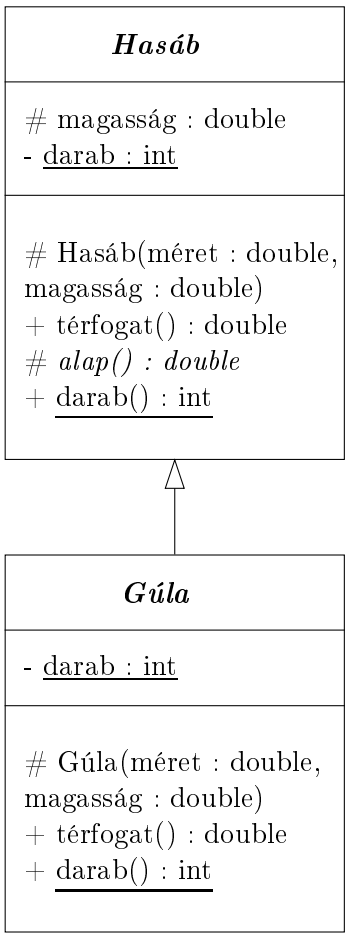
Az objektumok számát az egyes konstruktorokban kell növelni. Egy származtatott osztályban explicit meg kell hívnunk az ősosztály paraméteres konstruktorát. Ez egyben garantálja, hogy az ősosztály „objektumainak" száma is nő.

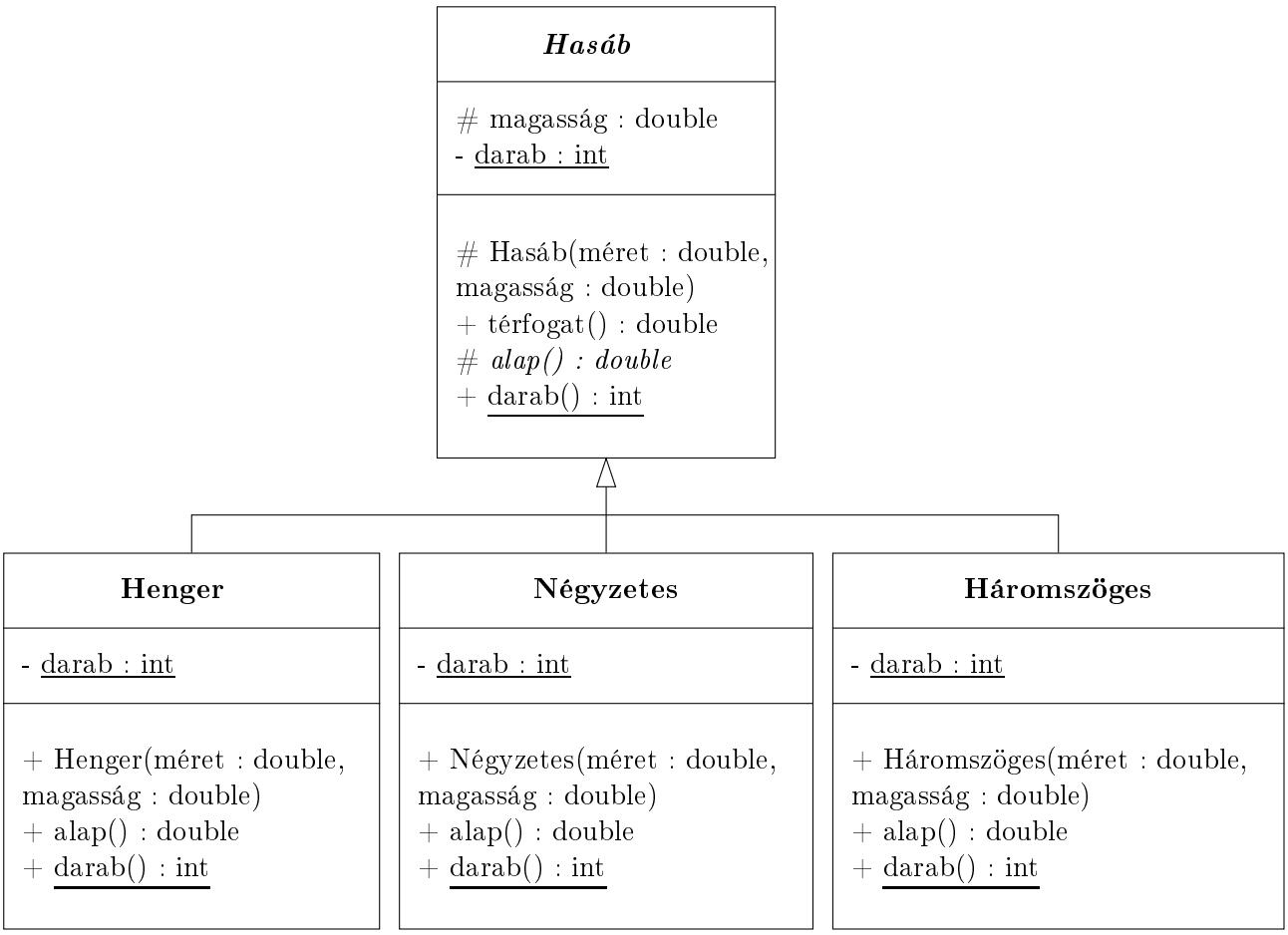
7.3. Osztálydiagram

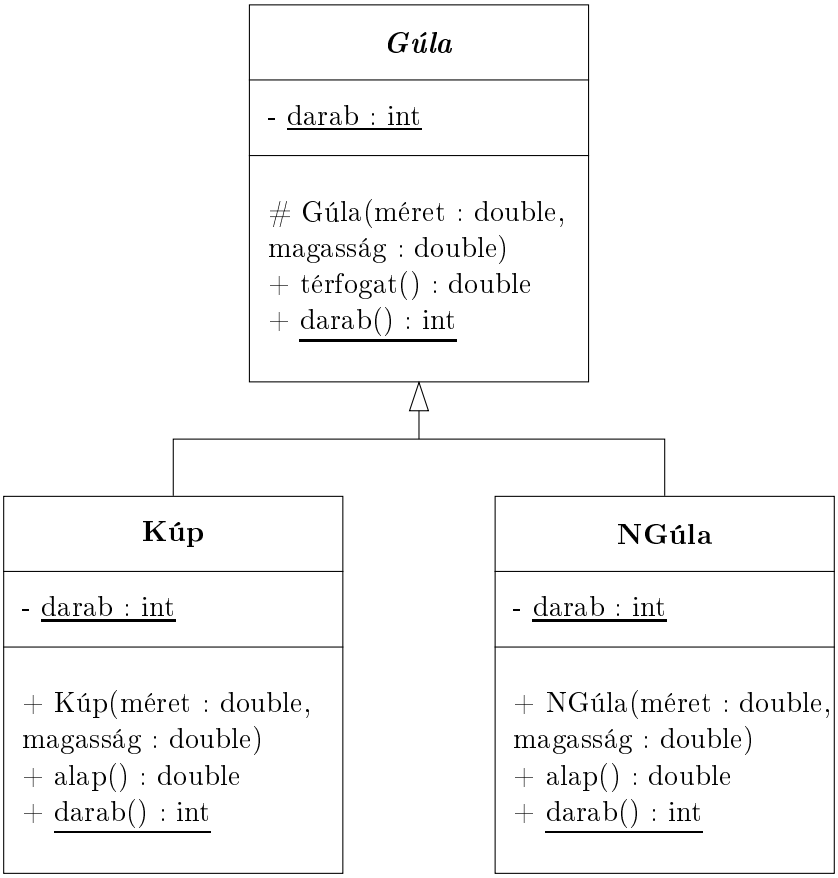



Test
méret : double
- darab : int
Test(méret : double)
+ térfogat() : double
+ darab() : int
Szabályos
- darab : int
Szabályos(méret : double)
+ térfogat() : double
# szorzó() : double
+ darab() : int
Hasáb
magasság : double
- darab : int
Hasáb(méret : double,
magasság : double)
+ térfogat() : double
# alap() : double
+ darab() : int


Szabályos
- darab : int
Szabályos(méret : double)
+ térfogat() : double
# szorzó() : double
+ darab() : int
Gömb
- darab : int
- szorzó : double = 4π/3
+ Gömb(méret : double)
# szorzó() : double
+ darab() : int
Kocka
- darab : int
+ Kocka(méret : double)
# szorzó() : double
+ darab() : int
Tetraéder
- darab : int
- szorzó : double = √2/12
+ Tetraéder(méret : double)
# szorzó() : double
+ darab() : int
Oktaéder
- darab : int
- szorzó : double = √2/3
+ Oktaéder(méret : double)
# szorzó() : double
+ darab() : int


Hasáb
magasság : double
- darab : int
Hasáb(méret : double,
magasság : double)
+ térfogat() : double
# alap() : double
+ darab() : int
Gúla
- darab : int
Gúla(méret : double,
magasság : double)
+ térfogat() : double
+ darab() : int


Hasáb
magasság : double
- darab : int
Hasáb(méret : double, magasság : double)
+ térfogat() : double
# alap() : double
+ darab() : int
Henger
- darab : int
+ Henger(méret : double, magasság : double)
+ alap() : double
+ darab() : int
Négyzetes
- darab : int
+ Négyzetes(méret : double, magasság : double)
+ alap() : double
+ darab() : int
Háromszöges
- darab : int
+ Háromszöges(méret : double, magasság : double)
+ alap() : double
+ darab() : int


Gúla
- darab : int
Gúla(méret : double,
magasság : double)
+ térfogat() : double
+ darab() : int
Kúp
- darab : int
+ Kúp(méret : double,
magasság : double)
+ alap() : double
+ darab() : int
NGúla
- darab : int
+ NGúla(méret : double,
magasság : double)
+ alap() : double
+ darab() : int

Java program: `Testek` projekt