Tervezési minták I.

1.rész

Testek térfogata (sablonfüggvény, stratégia, egyke)

Gregorics Tibor

gt@inf.elte.hu

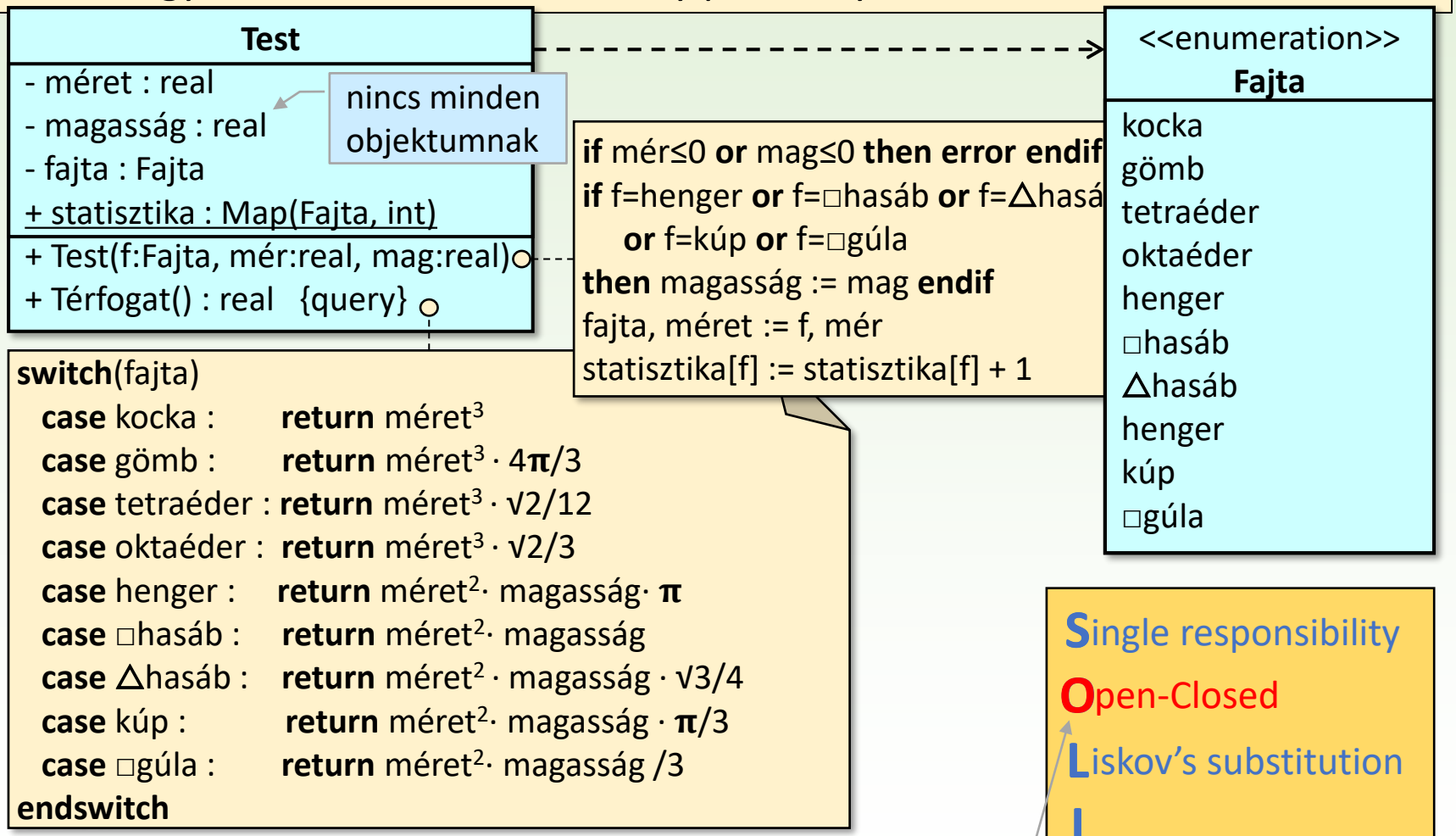
http://people.inf.elte.hu/gt/oep

Programozási minták

- A programozási feladatok megoldási folyamata gyorsabb, az előállított program biztonságosabb, ha a megoldást korábbi, hasonló feladatok megoldásainál bevált minták alapján állítjuk elő.
- A szoftvertechnológiában programozási mintáknak számos csoportja jött létre. Ilyenek például
  - az algoritmus minták (programozási tételek),
  - a tervezési minták (tervminták, design patterns).
- A tervezési minták az objektumelvű modellezést támogató minták, amelyeket az osztály diagram tervezése során alkalmazunk azért, hogy a modell újrafelhasználható, könnyen módosítható, biztonságosan működő, és hatékony legyen, valamint nem utolsó sorban megfeleljen a SOLID elveknek.

Feladat és ügyetlen megoldása

Készítsünk programot, amellyel különféle testeket hozhatunk létre azért, hogy kiszámolhassuk a térfogatukat. Eközben bármikor készíthessünk statisztikát arról, hogy a különféle testekből hány példánnyal rendelkezünk már.



Single responsibility

Open-Closed

Liskov's substitution

I

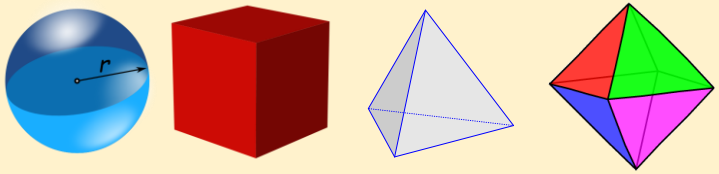
D

egy szoftver legyen nyitott új elemekkel (osztály, metódus) való bővítésre, de legyen zárt a meglevő részek módosítására

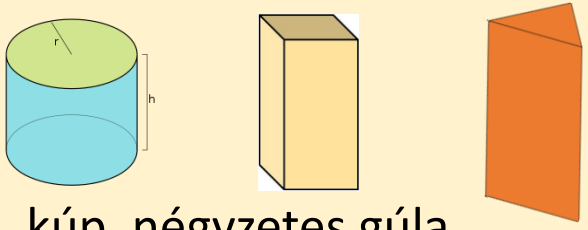
Objektum hierarchia

A testeket az alábbi módon csoportosíthatjuk:

- szabályos testek: gömb, kocka, tetraéder, oktaéder;



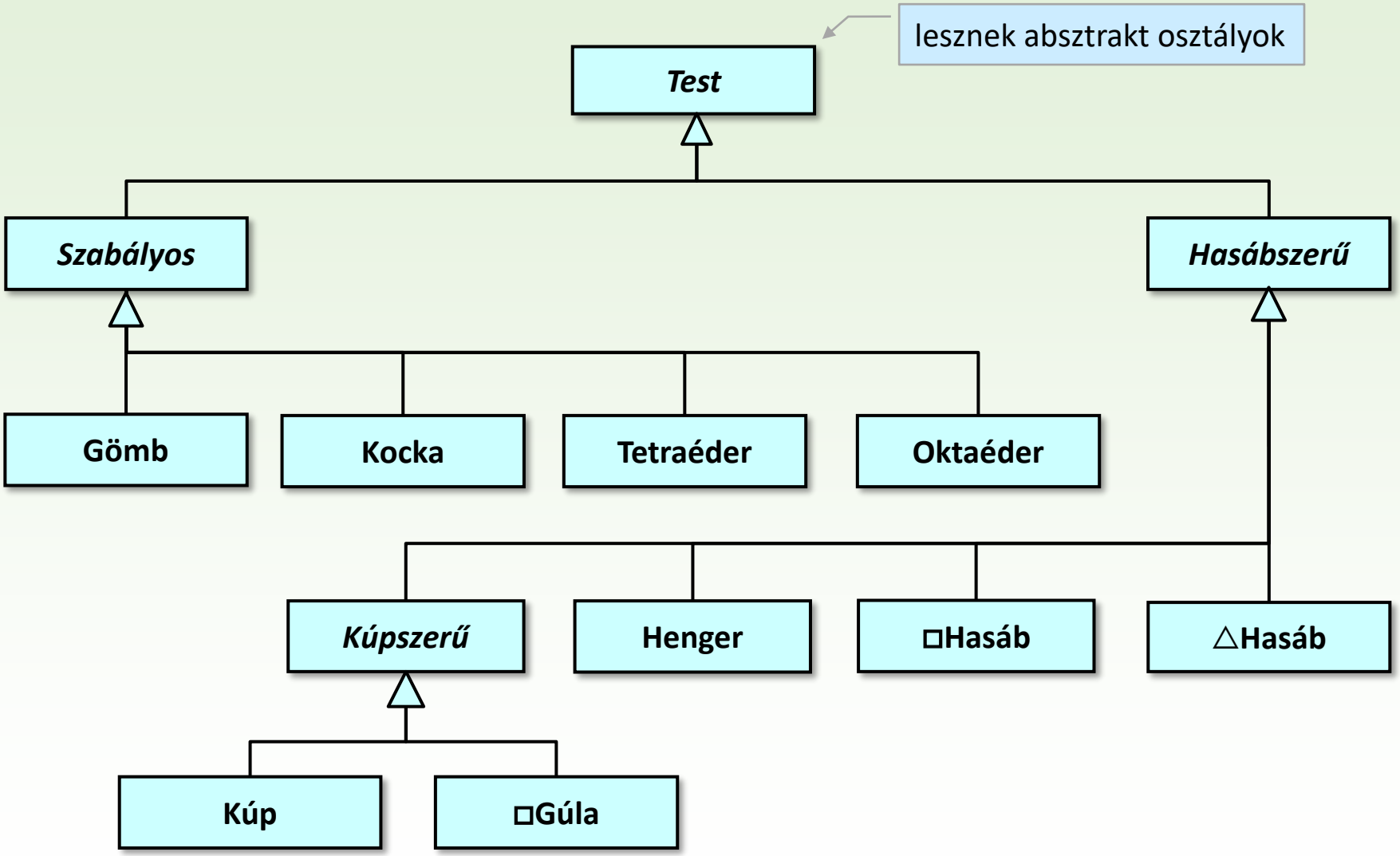
- hasáb jellegű testek: henger, négyzet és szabályos háromszög alapú hasáb;



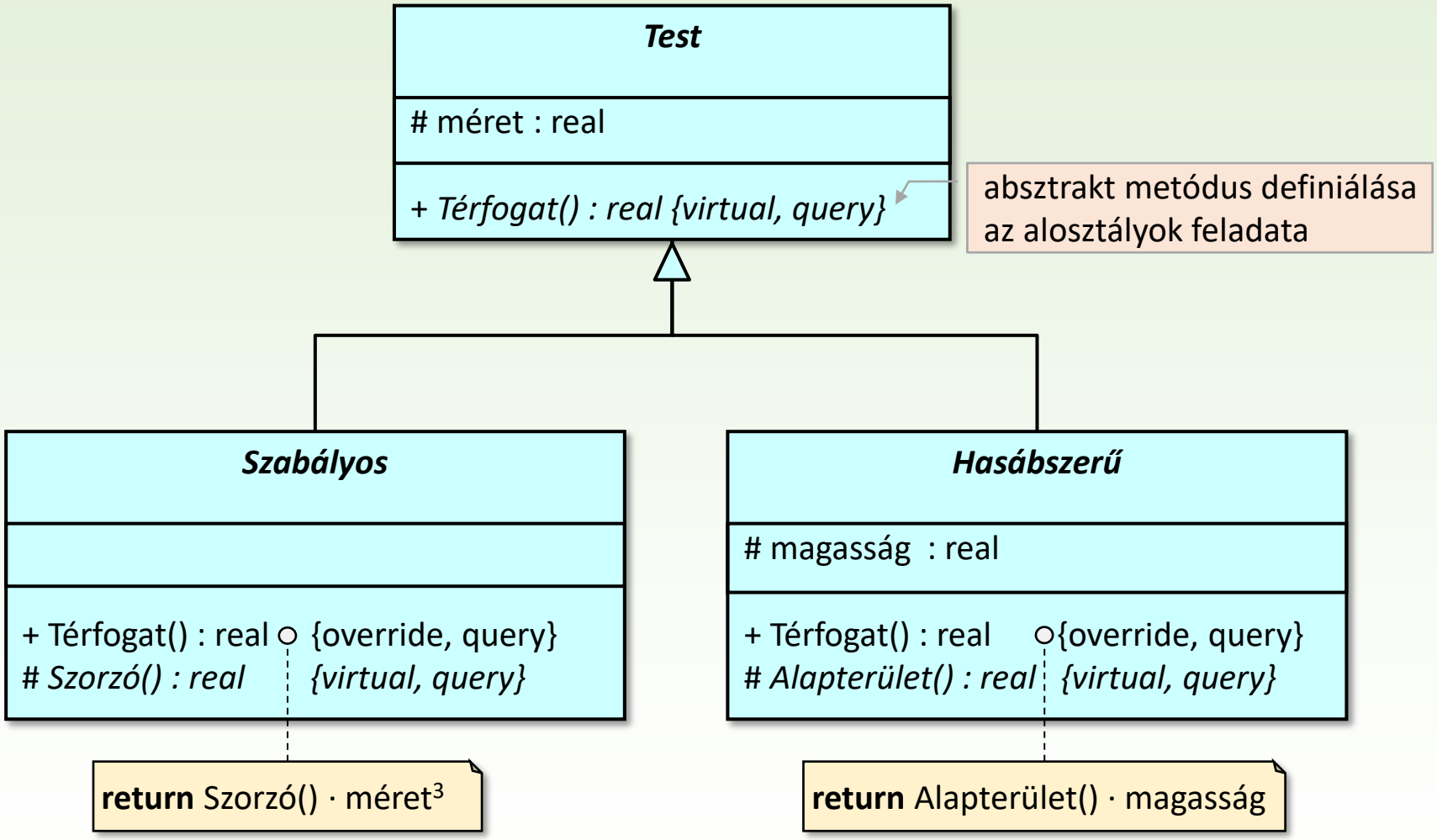
- gúla jellegű testek: kúp, négyzetes gúla.

Alkalmazzunk származtatást, de az ősosztályokból (pl. szabályos test) ne lehessen objektumokat példányosítani.

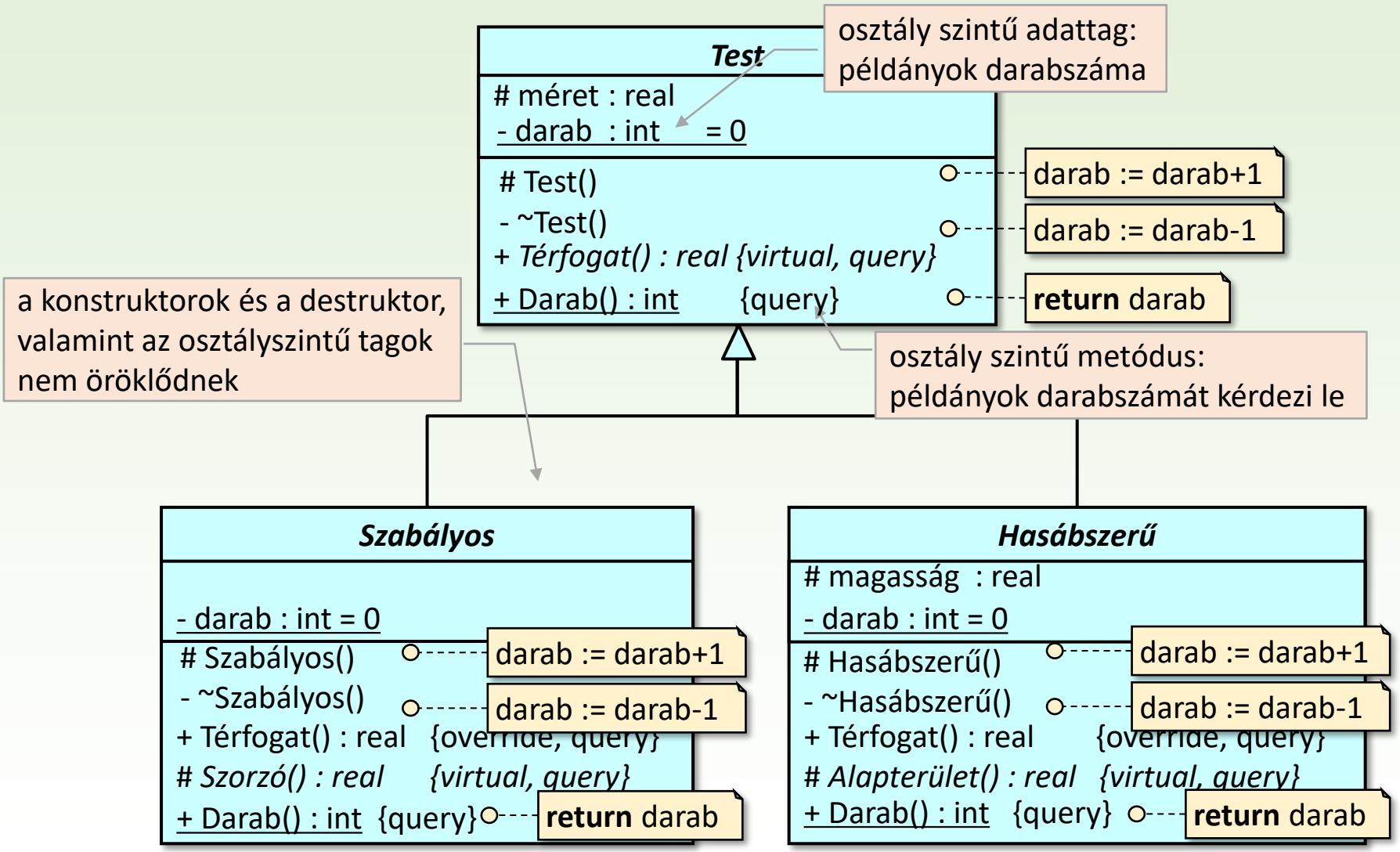
Osztály hierarchia



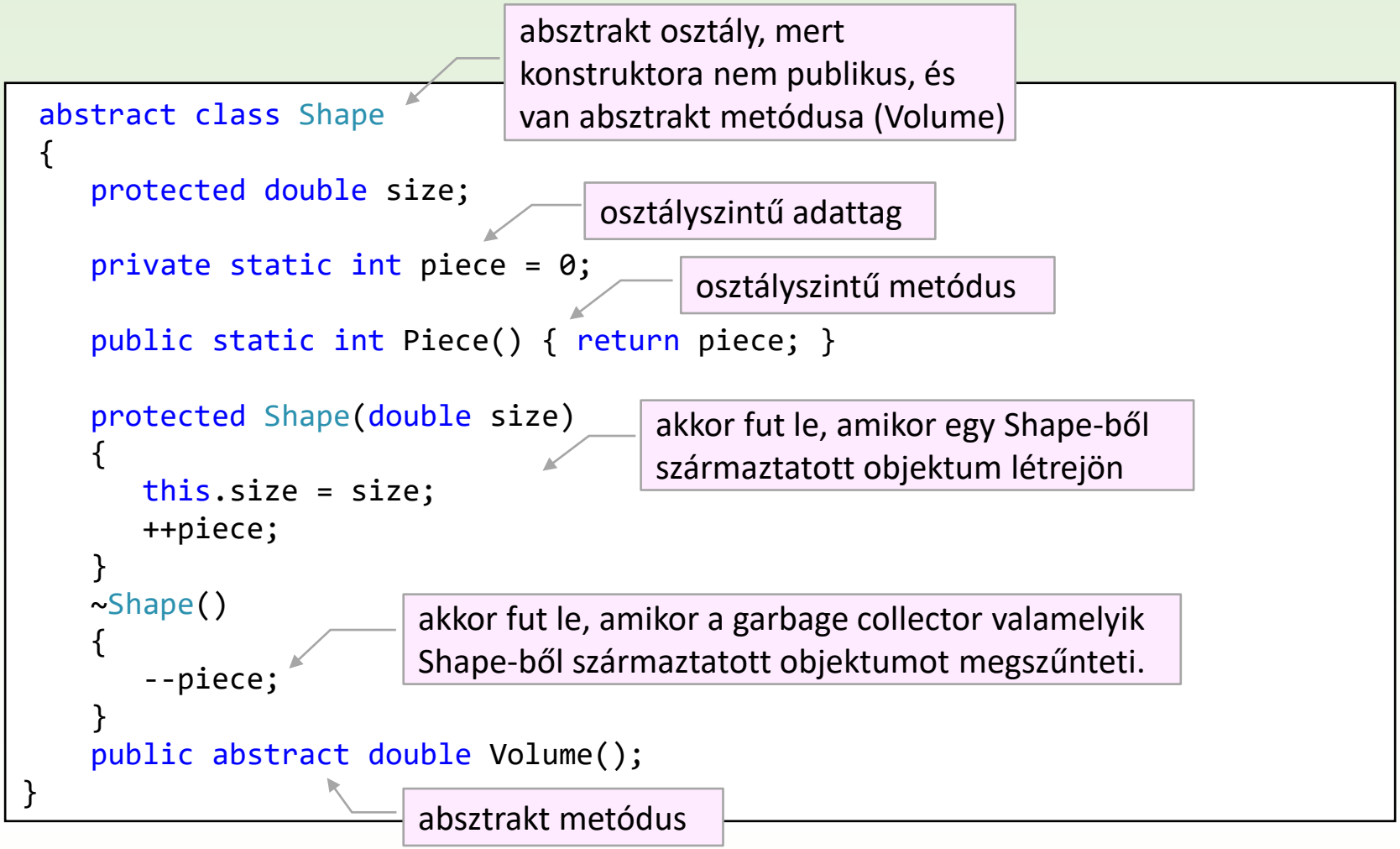
Absztrakt testek osztályai



Osztályszintű tagok



A testek ősosztálya




```
abstract class Shape
{
   protected double size;

   private static int piece = 0;

   public static int Piece() { return piece; }

   protected Shape(double size)
   {
      this.size = size;
      ++piece;
   }
   ~Shape()
   {
      --piece;
   }
   public abstract double Volume();
}
```

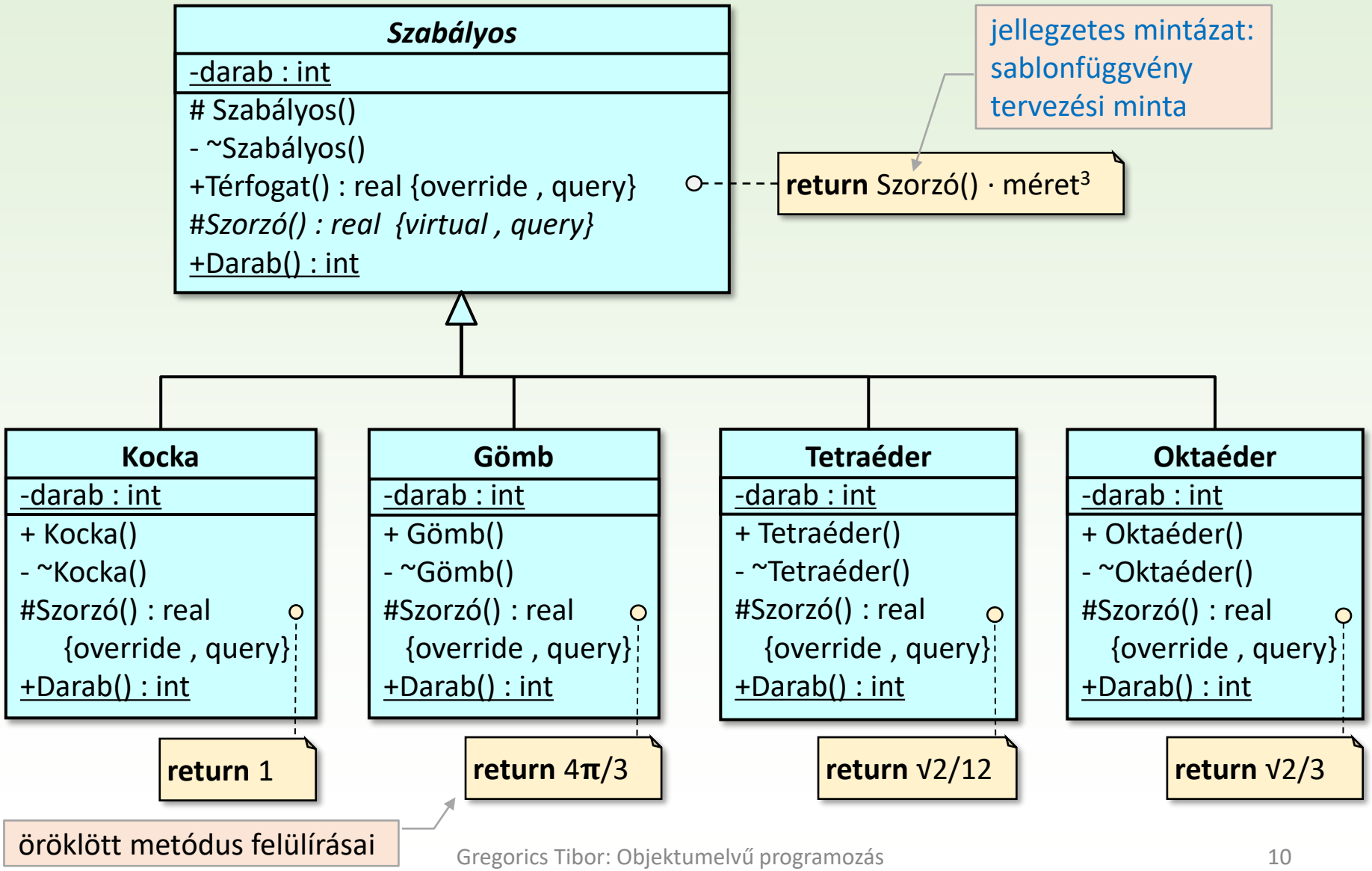
Szabályos absztrakt testek osztálya

```
abstract class Regular : Shape
{
   protected Regular(double size) : base(size) { ++piece; }

   ~Regular() { --piece; }

   public override double Volume()
   {
      return size * size * size * Multiplier();
   }

   protected abstract double Multiplier();

   private static int piece;

   public static new int Piece() { return piece; }
}
```

ősosztály adott paraméterezésű konstruktorát hívja

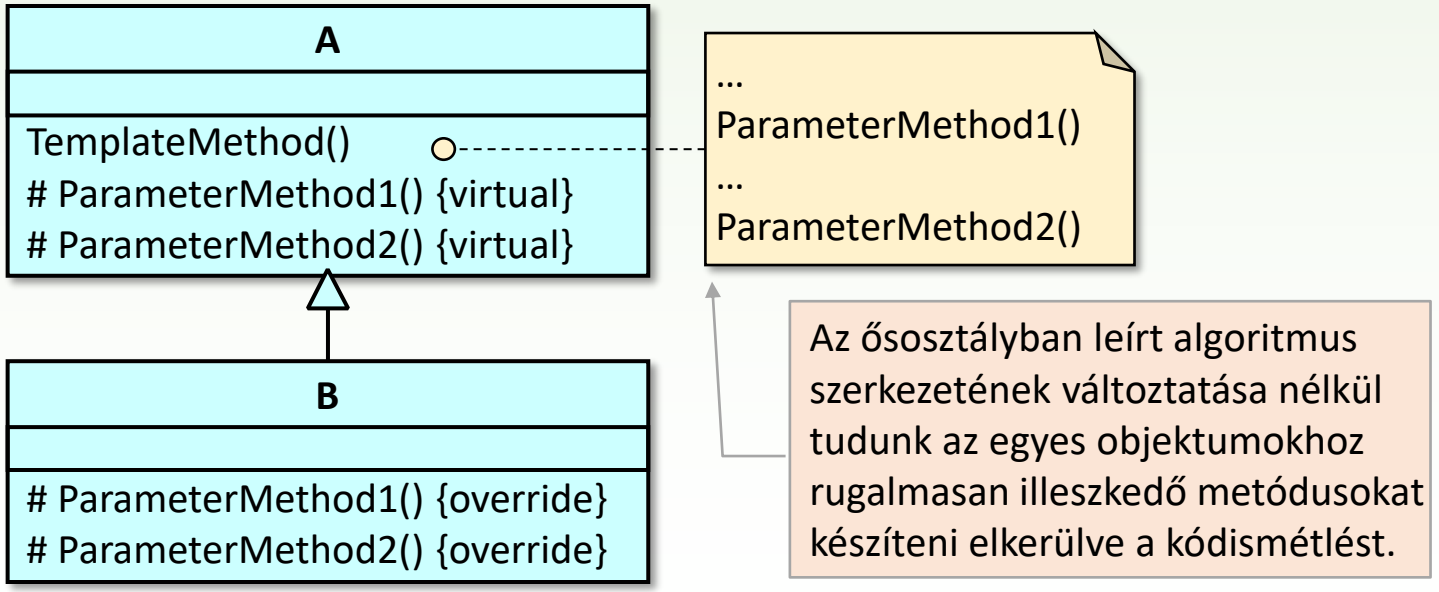
explicit módon jelzi, hogy az ősosztálybeli ilyen nevű tagot írjuk felül

Szabályos testek



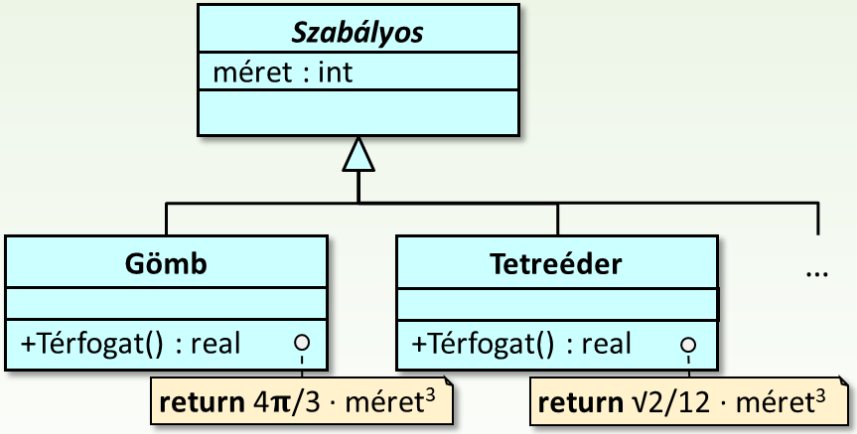
Sablonfüggvény tervezési minta (template method)

- Egy tevékenységet (sablonfüggvényt) egy ősosztálybeli metódusban vezetünk be úgy, hogy a speciális, azon objektum típusától függő részeit, amely objektumra e tevékenységet végrehajtanánk, csak virtuális metódus-hívások jelzik, és ezen metódusokat felüldefiniáljuk az adott objektum osztályában (ami egy alosztály).

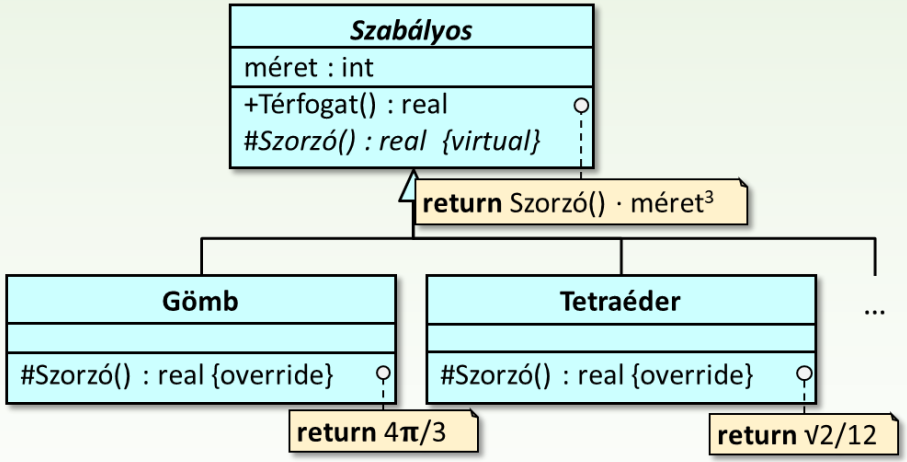


Sablonfüggvény tervminta hatása

ilyen volt



ilyen lett

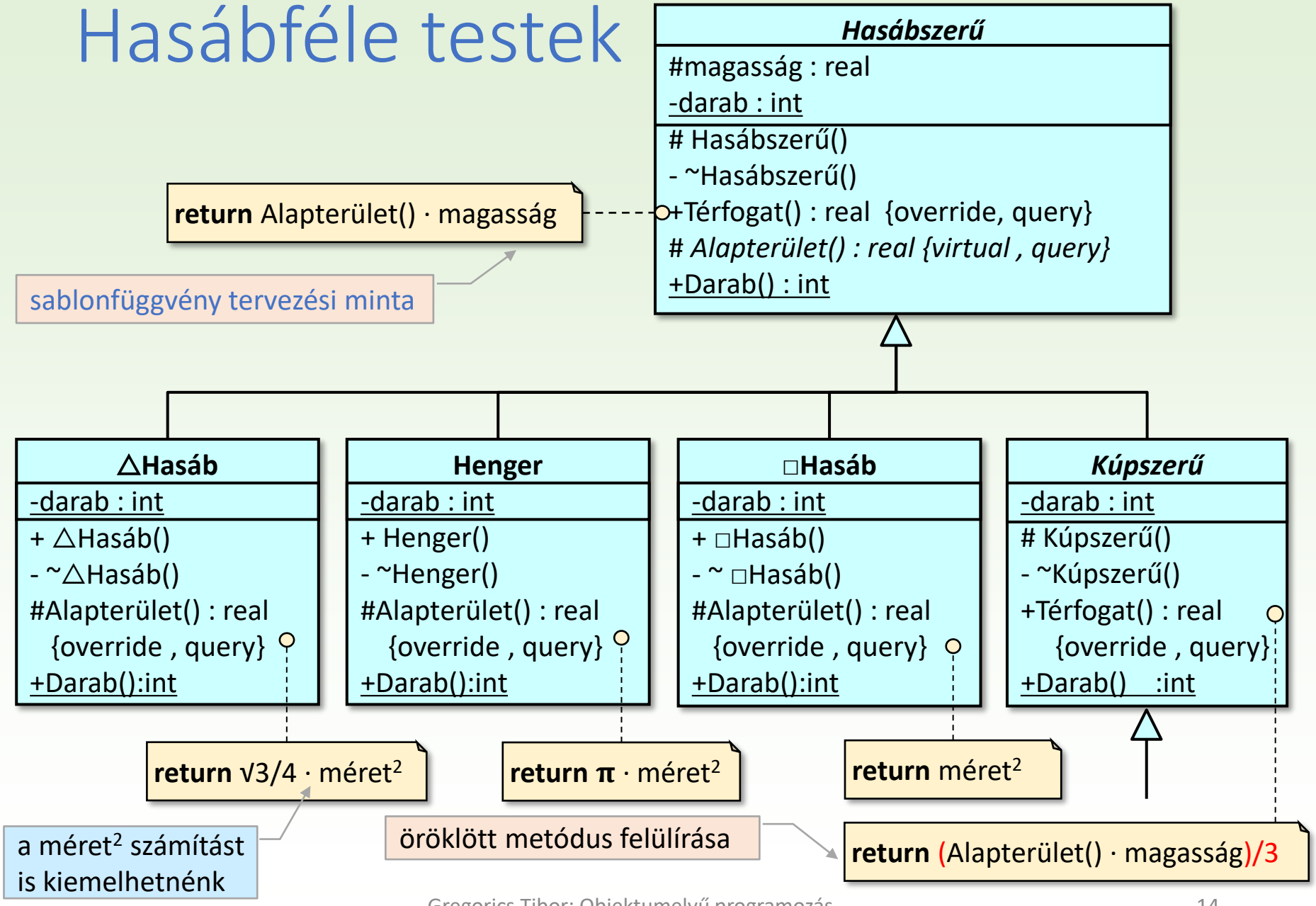


megszűnik a kódredundancia

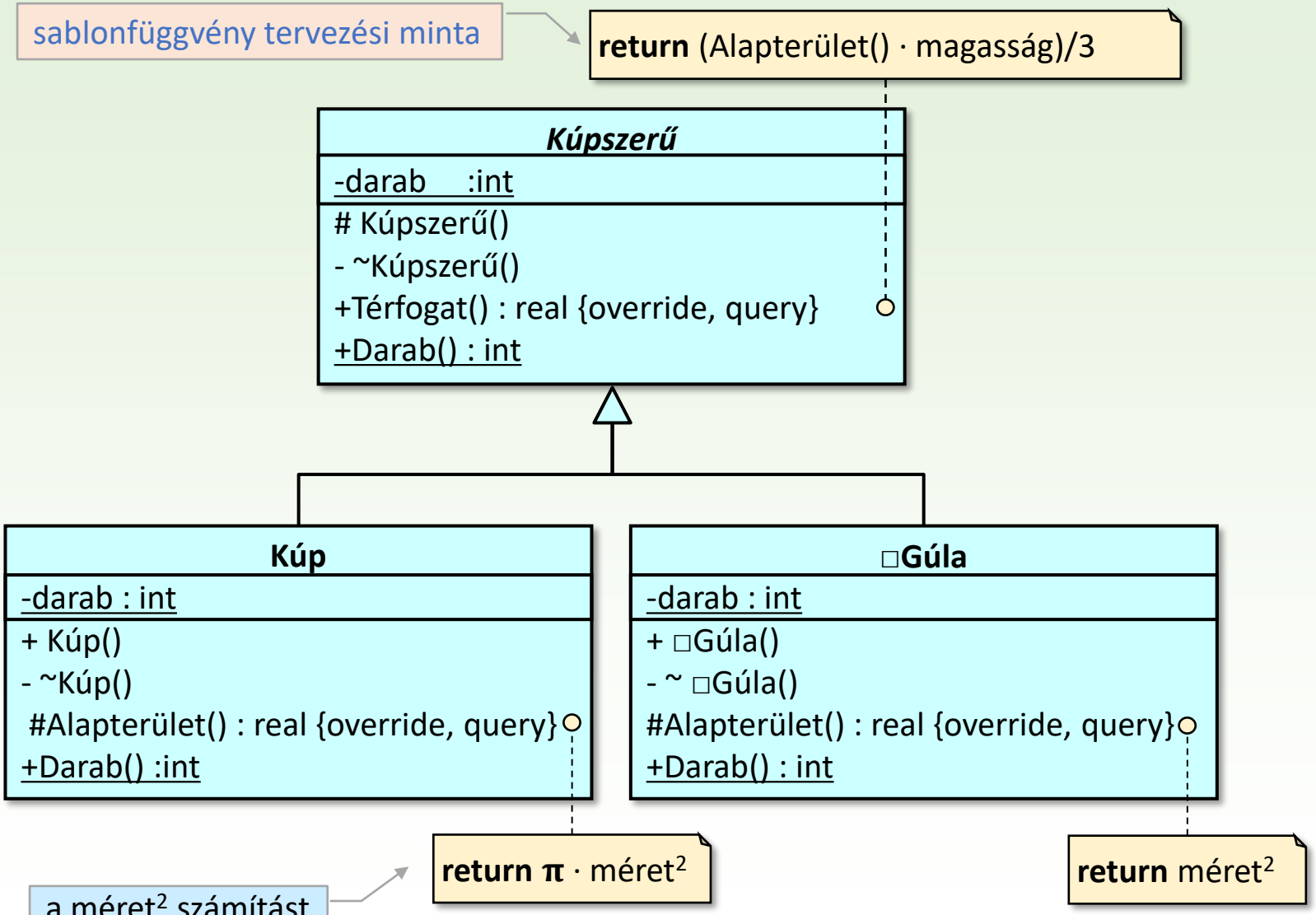
Gömb

```
class Sphere : Regular
{
   public Sphere(double size) : base(size) { ++piece; }

   ~Sphere() { --piece; }

   protected override double Multiplier()
   {
      return 4.0 * 3.14159 / 3.0;
   }

   private static int piece = 0;

   public static new int Piece() { return piece; }
}
```

Hasábféle testek



Kúpszerű testek



Főprogram

shapes.txt

```
Cube 5.0
Cylinder 3.0 8.0
Cylinder 1.0 10.0
Tetrahedron 4.0
SquarePyramid 3.0 10.0
Octahedron 1.0
Cube 2.0
SquarePyramid 2.0 10.0
```

```
static void Main()
{
   TextFileReader reader = new ("shapes.txt");

   List<Shape> shapes = new ();

   while (Shape.Create(ref reader, out Shape sh) )
   {
      shapes.Add(sh);
   }

   Console.WriteLine("Volumes:");
   foreach (Shape shape in shapes)
   {
      Console.WriteLine($"{shape.ToString().Substring(7)} : "
                        + $"{shape.Volume():f2}");
   }

   Statistics();
}
```

Különféle testek példányosítása a szöveges állomány sorai alapján

a testek térfogatának kiírása

a típus neve

A futási idejű polimorfizmus miatt ez itt a shape által hivatkozott konkrét testnek a Volume() metódusa, nem a Shape ősosztályé.

a külön fajta testek számának kiírása

Test példányosítása

shapes.txt

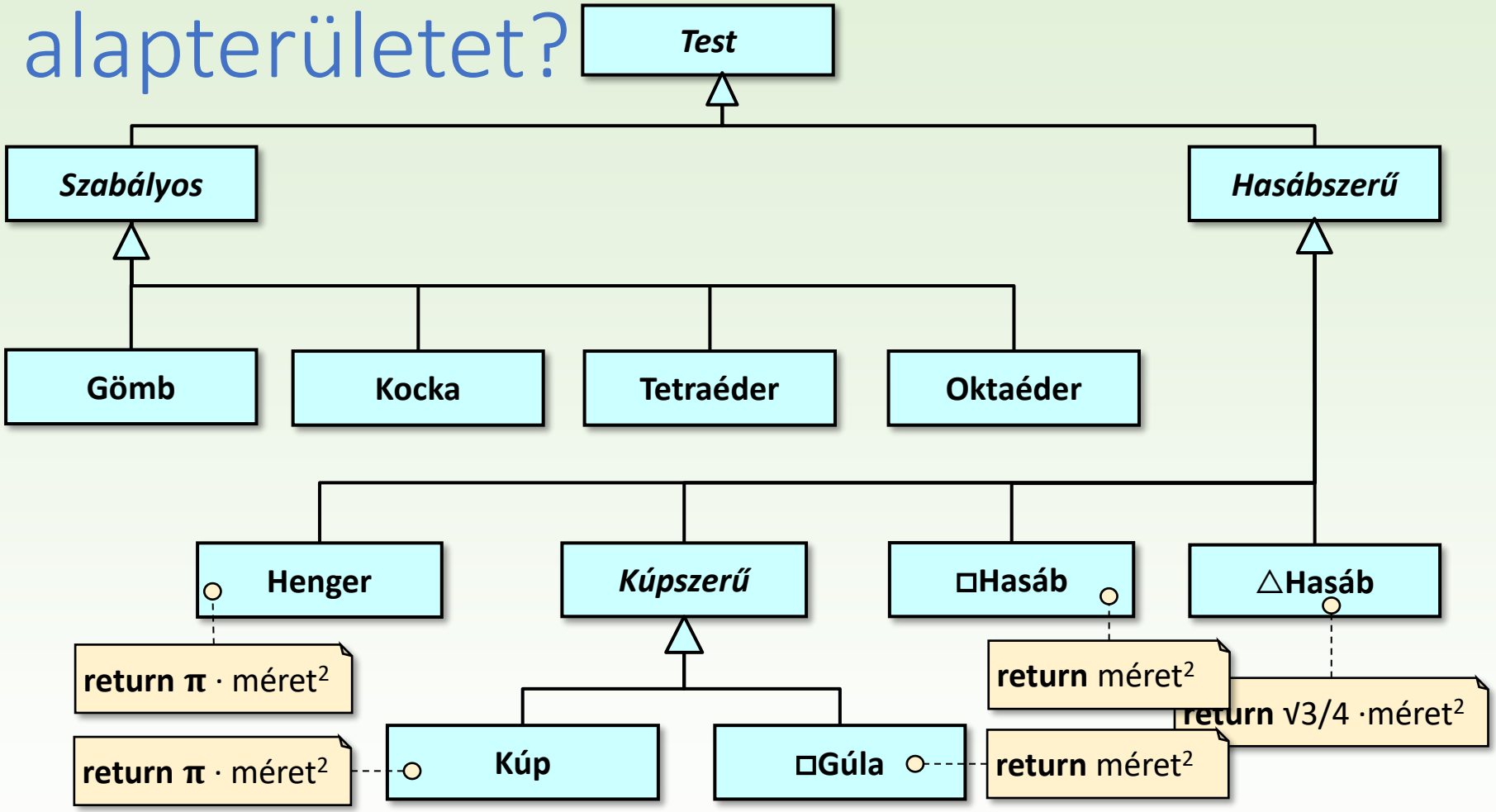
```
Cube 5.0
Cylinder 3.0 8.0
Cylinder 1.0 10.0
Tetrahedron 4.0
SquarePyramid 3.0 10.0
                    .0
```

```
public static bool Create(ref TextFileReader reader, out Shape sh)
{
   sh = null;
   if (reader.ReadString(out string type))
   {
      reader.ReadDouble(out double size);
      switch (type)
      {
         case "Cube":         sh = new Cube(size); break;
         case "Sphere":       sh = new Sphere(size); break;
         case "Tetrahedron": sh = new Tetrahedron(size); break;
         case "Octahedron":  sh = new Octahedron(size); break;
         case "Cylinder":    reader.ReadDouble(out double height);
                             sh = new Cylinder(size, height); break;
         ...
         default: throw new UnknownShapeException();
      }
      return true;
   }
   else return false;
}
```

osztályszintű gyártó függvény

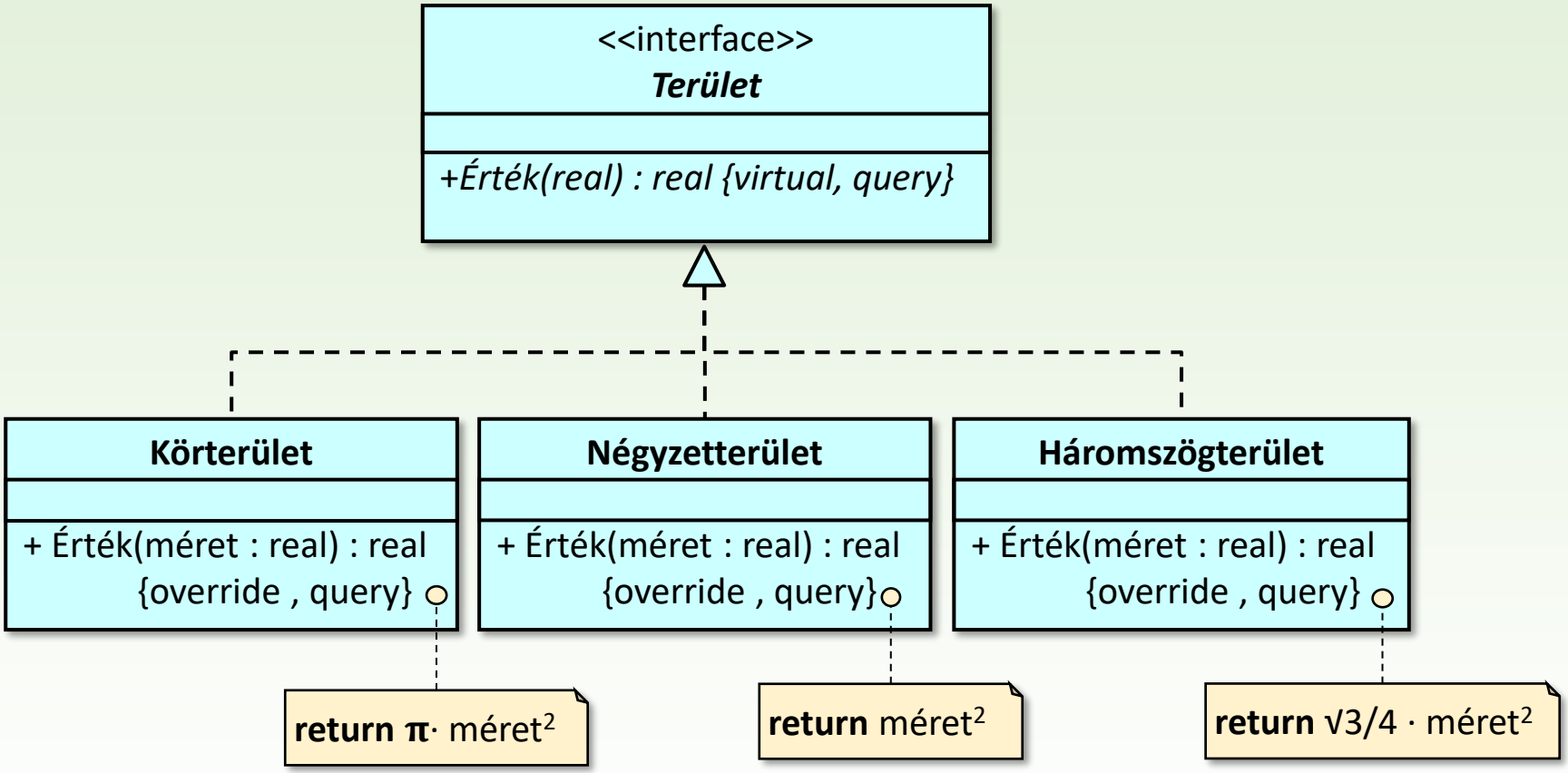
A származtatás miatt lehet értékül adni egy Shape típusú változónak egy SquarePrism típusú értéket (hivatkozást.

Hol mindenhol számol a modell alapterületet?

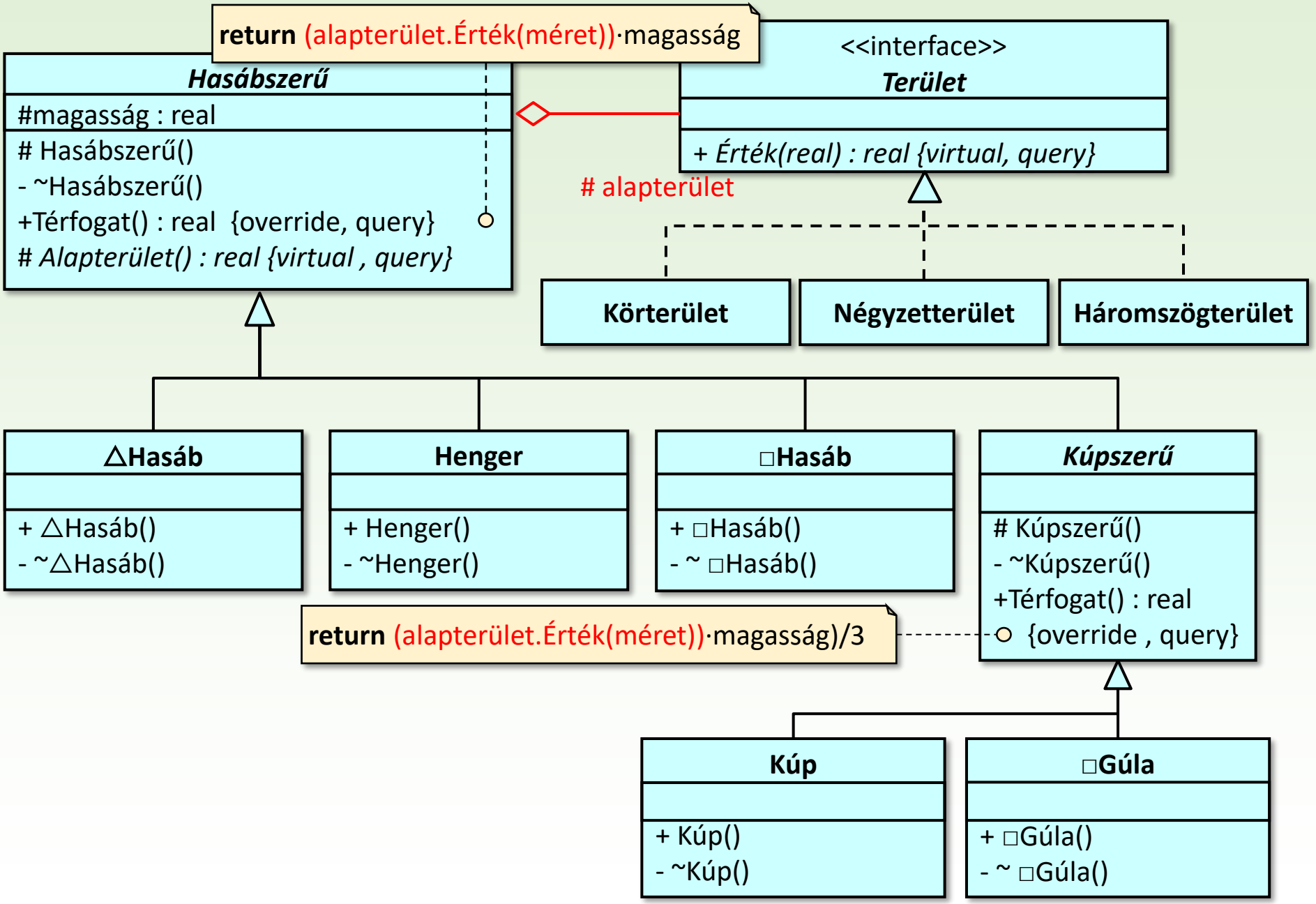


Kritika a modellről: redundancia (kódismétlődés) jelent meg a modellben. Ugyanazon alapterület kiszámolása több osztályban is szerepel: a köré a kúp és a henger osztályaiban, négyzeté a négyzetalapú hasáb és gúla osztályaiban.

Területet számoló objektumok



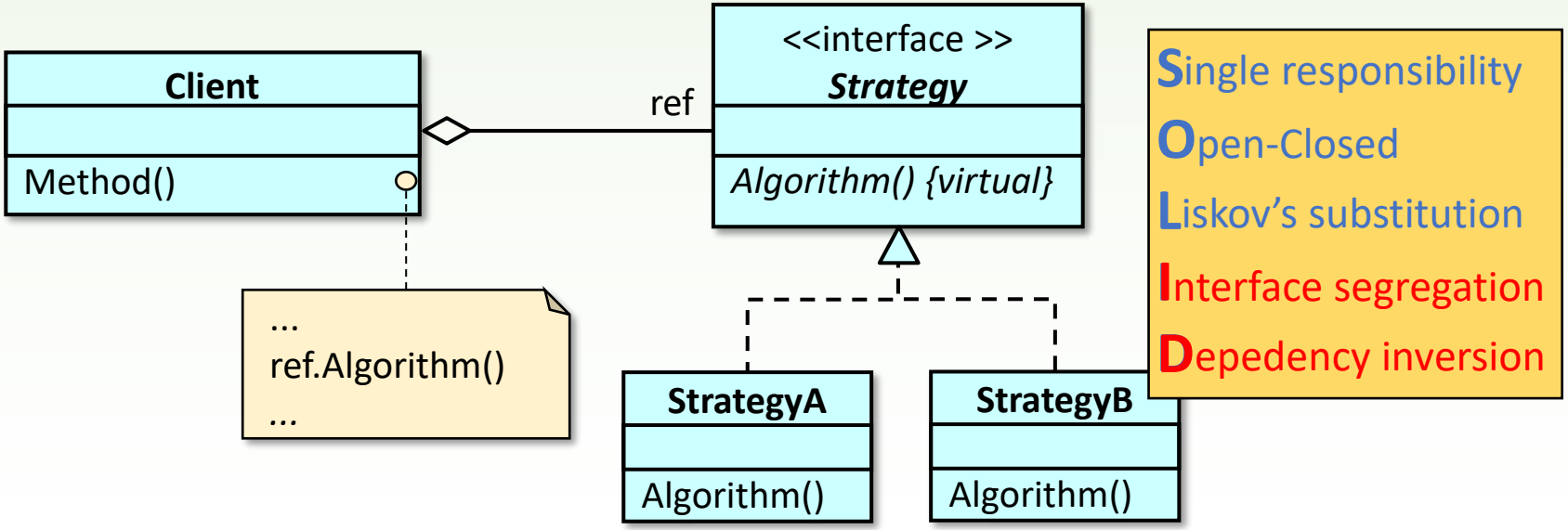
Itt, és csak itt kell az alapterületeket definiálni – szűnőben a redundancia. Ki lehetne még a méret^2 számolását is emelni, hogy az alosztályok csak az együtthatókat definiálnák. (Ekkor a Terület nem interfész lenne, hanem egy absztrakt osztály, és a sablonfüggvény tervmintát alkalmaznánk.)


return (alapterület.Érték(méret))·magasság
Hasábszerű
#magasság : real
Hasábszerű()
- ~Hasábszerű()
+Térfogat() : real {override, query}
Alapterület() : real {virtual , query}
<<interface>>
Terület
+ Érték(real) : real {virtual, query}
# alapterület
Körterület
Négyzetterület
Háromszögterület
△Hasáb
+ △Hasáb()
- ~△Hasáb()
Henger
+ Henger()
- ~Henger()
□Hasáb
+ □Hasáb()
- ~ □Hasáb()
Kúpszerű
Kúpszerű()
- ~Kúpszerű()
+Térfogat() : real
{override , query}
return (alapterület.Érték(méret))·magasság)/3
Kúp
+ Kúp()
- ~Kúp()
□Gúla
+ □Gúla()
- ~ □Gúla()

Stratégia (strategy) tervezési minta

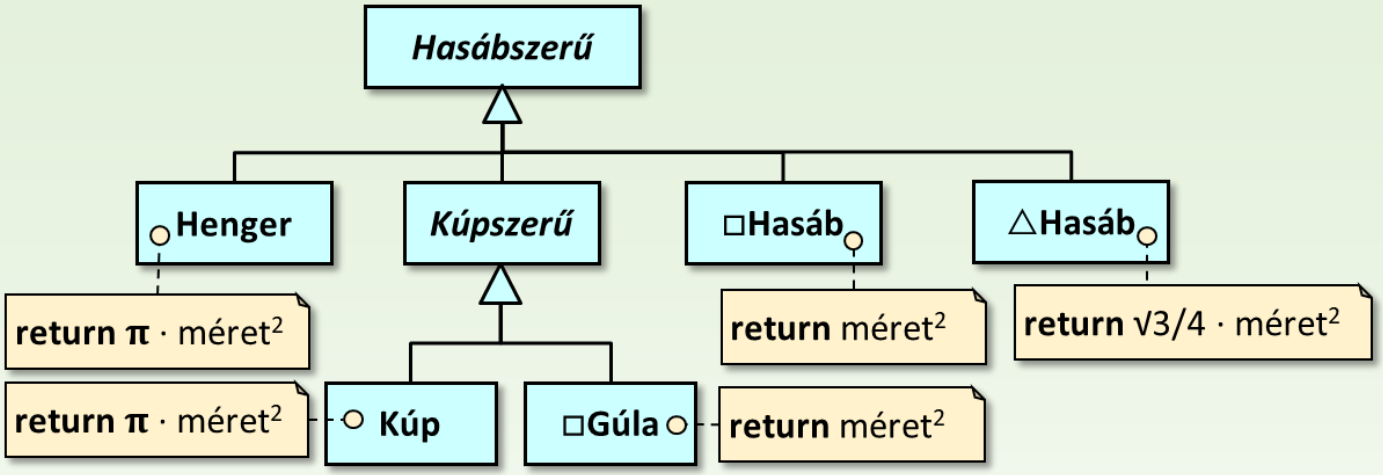
- Ahhoz, hogy futási időben dőljön el, hogy egy kliens objektum (metódusa) egy tevékenység család melyik elemét használja, e tevékenységeket egy interfészt megvalósító osztálycsalád azonos nevű metódusaiban írjuk le. A kliens úgy fér hozzá a megfelelő tevékenységhez, hogy az osztálycsalád valamelyikének objektumát „befecskendezzük” a kliens objektumba.

aggregációval, vagy a kliens metódusának paramétereként



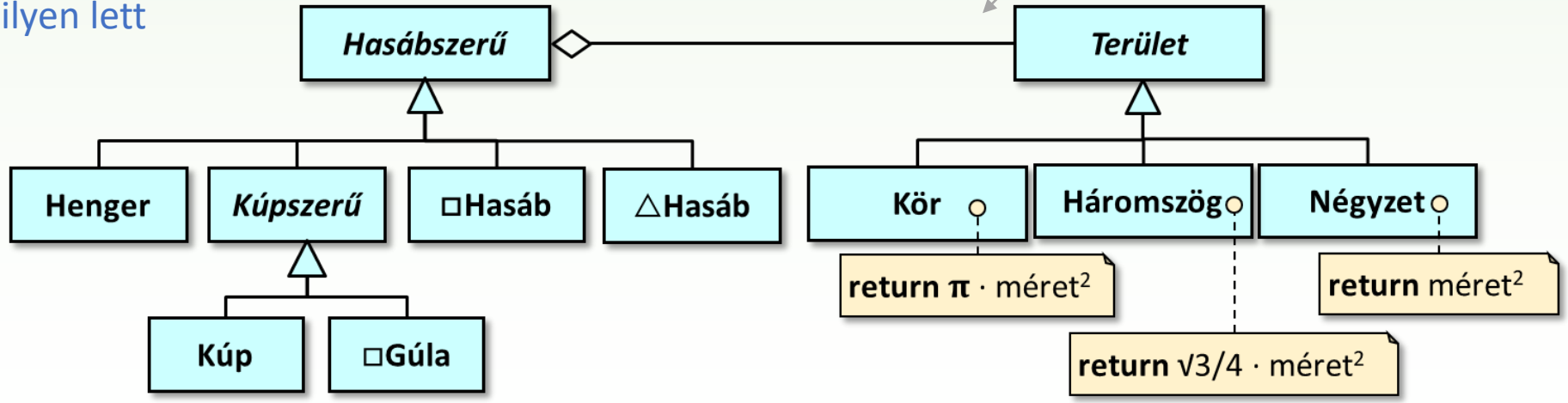
Stratégia tervminta hatása

ilyen volt



ilyen lett

szűnőben a kódredundancia



Alapterület objektum használata

```
abstract class Prismatic : Shape
{
   protected IBaseArea baseArea;
   protected double height;
   protected Prismatic(double size, double height) : base(size)
   {
      this.height = height;
      ++piece;
   }
   public override double Volume()
   {
      return baseArea.Value(size) * height;
   }
   private static int piece = 0;
   public static new int Piece() { return piece; }
}
```

az alapterület objektum hivatkozásának tárolására alkalmas adattag

az alapterület objektum területszámító metódusa

Alapterület objektum aggregációja

```
class Cylinder : Prismatic
{
   public Cylinder(double size, double height) : base(size, height)
   {
      baseArea = new CircleArea();
      ++piece;
   }
   private static int piece = 0;
   public static new int Piece() { return piece; }
}
```

az alapterület adattagot a konstruktor állítja be

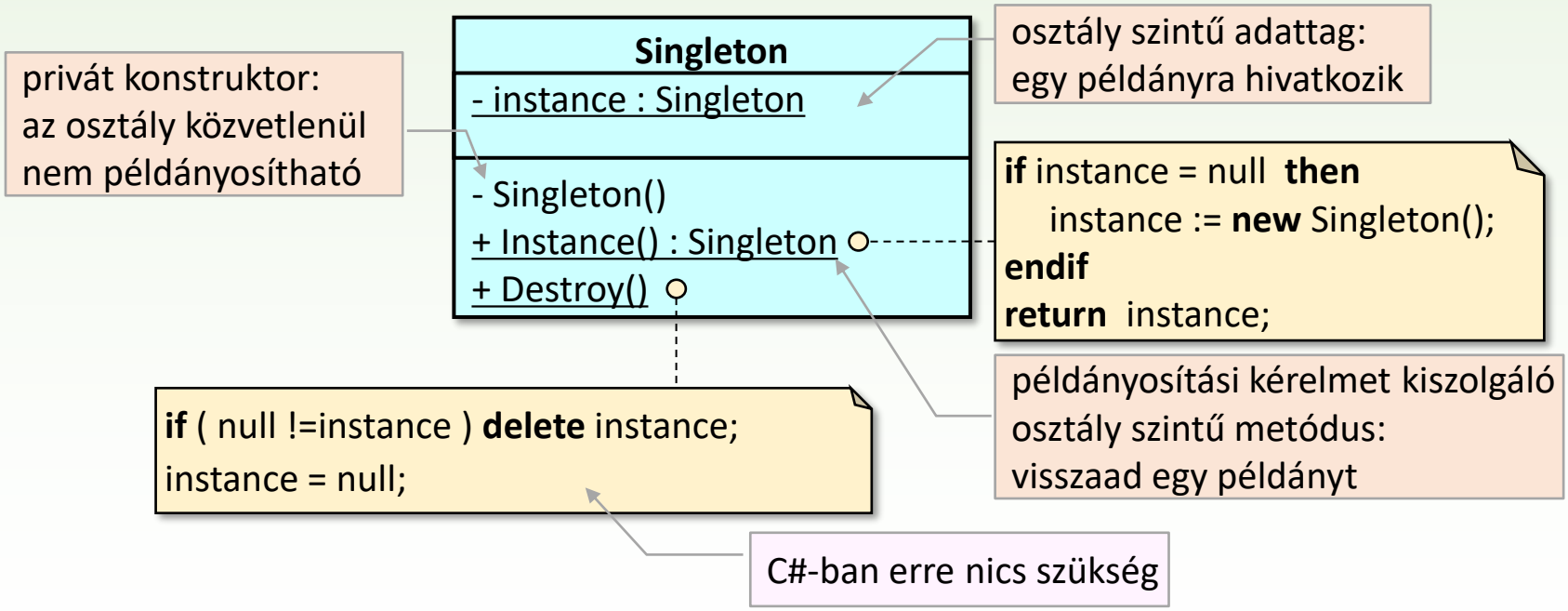
Kritika a hatékonyságról:
A kód-redundancia megszűnt, de memória-pazarlást okoz: például 5 henger és 3 kúp példányosítása során összesen 8 darab körterületet számoló objektum jön létre, pedig ebből egy is elég lenne.

```
class Cone : Pyramidal
{
   public Cone(double size, double height) : base(size, height)
   {
      baseArea = new CircleArea();
      ++piece;
   }
   private static int piece = 0;
   public static new int Piece() { return piece; }
}
```

az alapterület adattagot a konstruktor állítja be

Egyke (singleton) tervezési minta

- Amikor egy osztályhoz legfeljebb egy objektumot kell példányosítani függetlenül a példányosítási kérelmek számától.



Alapterület egykék

```
class CircleArea : IArea
{
   private CircleArea() { }

   double IArea.Area(double r)
   {
      return 3.14159 * r * r;
   }

   private static CircleArea instance = null;

   public static CircleArea Instance()
   {
      if (instance == null) instance = new CircleArea();
      return instance;
   }
}
```

Tervezési minták I.

2.rész

Lények túlélési versenye (állapot, látogató)

Gregorics Tibor

gt@inf.elte.hu

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

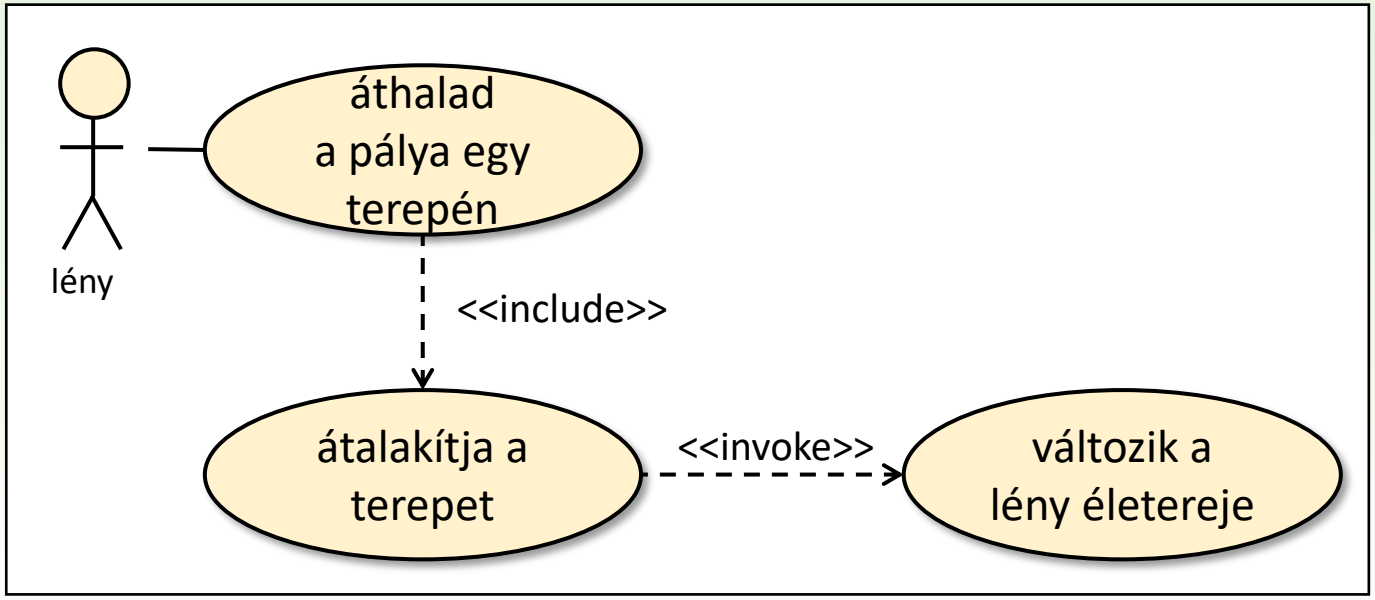
Feladat

Szimuláljuk különféle lények túlélési versenyét.

A lények három faj (zöldikék, buckabogarak, tocsogók) valamelyikéhez tartoznak. Van nevük, ismert a fajuk, és az aktuális életerejük (egész szám). A versenyen induló lények sorban egymás után egy olyan pályán haladnak végig, ahol három féle (homokos, füves, mocsaras) terep váltakozik. Amikor egy lény keresztül halad egy terepen, akkor a fajától (és az adott tereptől is) függően átalakítja a terepet, miközben változik az életereje. Ha az életereje elfogy, a lény elpusztul. Adjuk meg a pálya végéig eljutó, azaz életben maradt lények neveit!

- ***Buckabogár**: füvön az ereje kettővel csökken, homokon hárommal nő, mocsárban néggyel csökken; a füvet homokká, a mocsarat fűvé alakítja, de a homokot nem változtatja meg.*
- ***Tocsogó**: füvön az életereje kettővel, homokon öttel csökken, mocsárban hattal nő; a füvet mocsárrá alakítja, a másik két fajta terepet nem változtatja meg.*
- ***Zöldike**: füvön az életereje eggyel nő, homokon kettővel csökken, mocsárban eggyel csökken; a mocsaras terepet fűvé alakítja, a másik két terep fajtát nem változtatja meg.*

Egy lény egy lépése



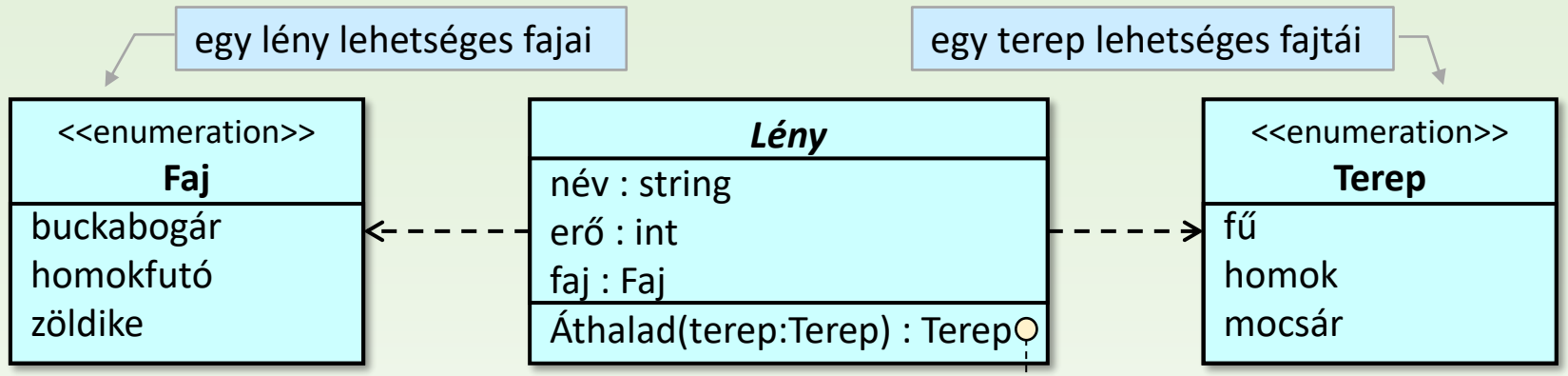
Mi történik, amikor egy lény áthalad egy terepen?

buckabogarak	életerő változás	terepváltozás
fű	-2	homok
homok	+3	-
mocsár	-4	fű

tocsogók	életerő változás	terepváltozás
fű	-2	mocsár
homok	-5	-
mocsár	+6	-

zöldikék	életerő változás	terepváltozás
fű	+1	-
homok	-2	-
mocsár	-1	fű

Osztály diagram: 1. próbálkozás

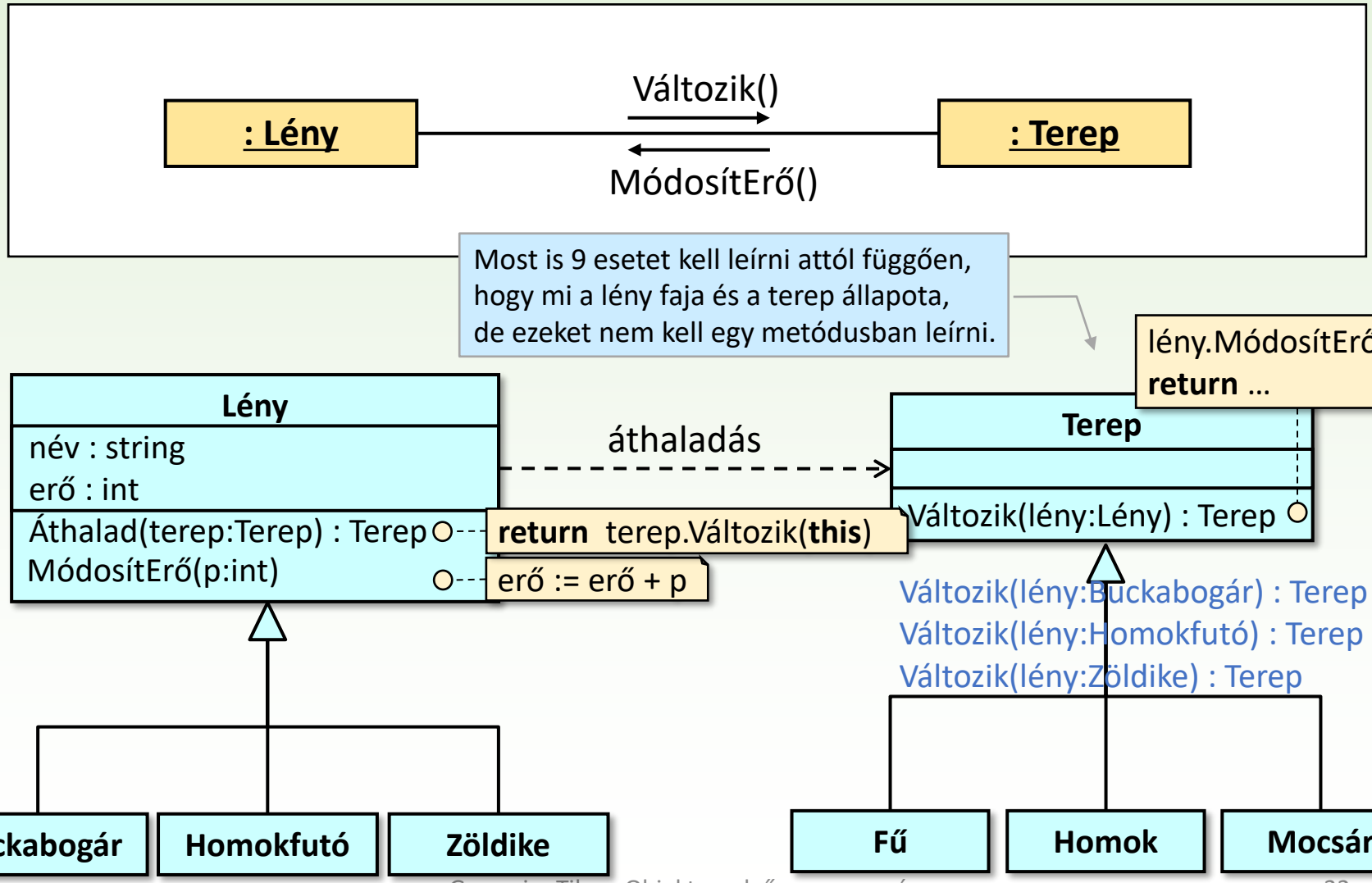



```
if      faj=buckabogár and  terep = fű       then erő:=erő+3; return terep;
elseif faj=buckabogár and terep = homok then erő:=erő–2; return homok;
elseif faj=buckabogár and terep=mocsár  then erő:=erő–4; return fű;
elseif faj=Tcsogó       and terep = fű       then erő:=erő–5; return terep;
elseif faj=Tocsogó      and terep = homok then erő:=erő–2; return mocsár;
elseif faj=Tocsogó      and terep=mocsár  then erő=erő+6; return terep;
elseif faj=Zöldike      and terep = fű       then erő:=erő–2; return terep;
elseif faj=Zöldike      and terep = homok then erő:=erő+1; return terep;
elseif faj=Zöldike      and terep=mocsár  then erő:=erő–1; return fű;
endif
```

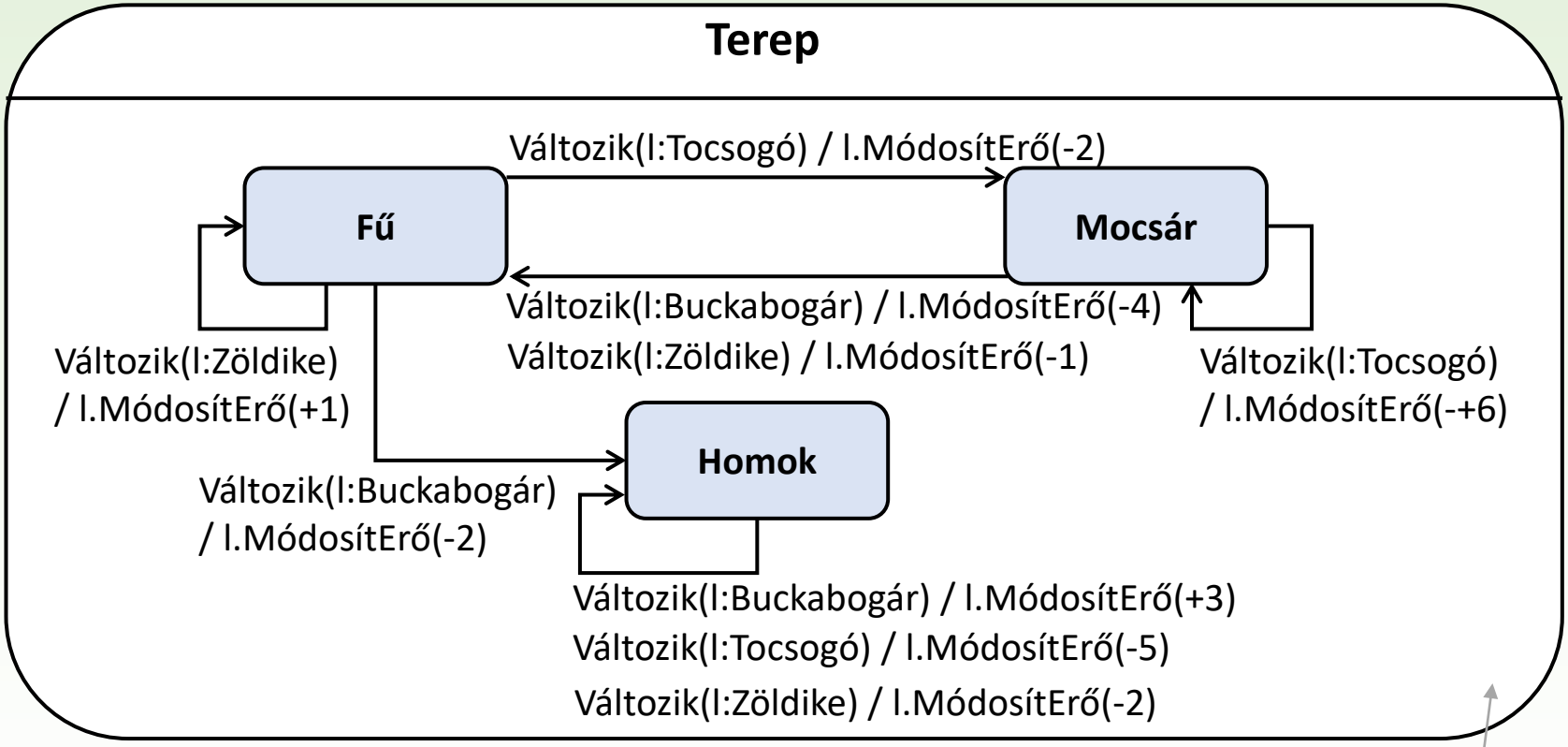
9 esetet kell leírni a lény és a terep állapotától függően

Kritika: sérül az Open-Closed elv
újabb terepfajta bevezetése esetén az összes elágazást ki kell bővíteni, azaz meglévő kódon kell változtatni

Osztály diagram: 2. próbálkozás

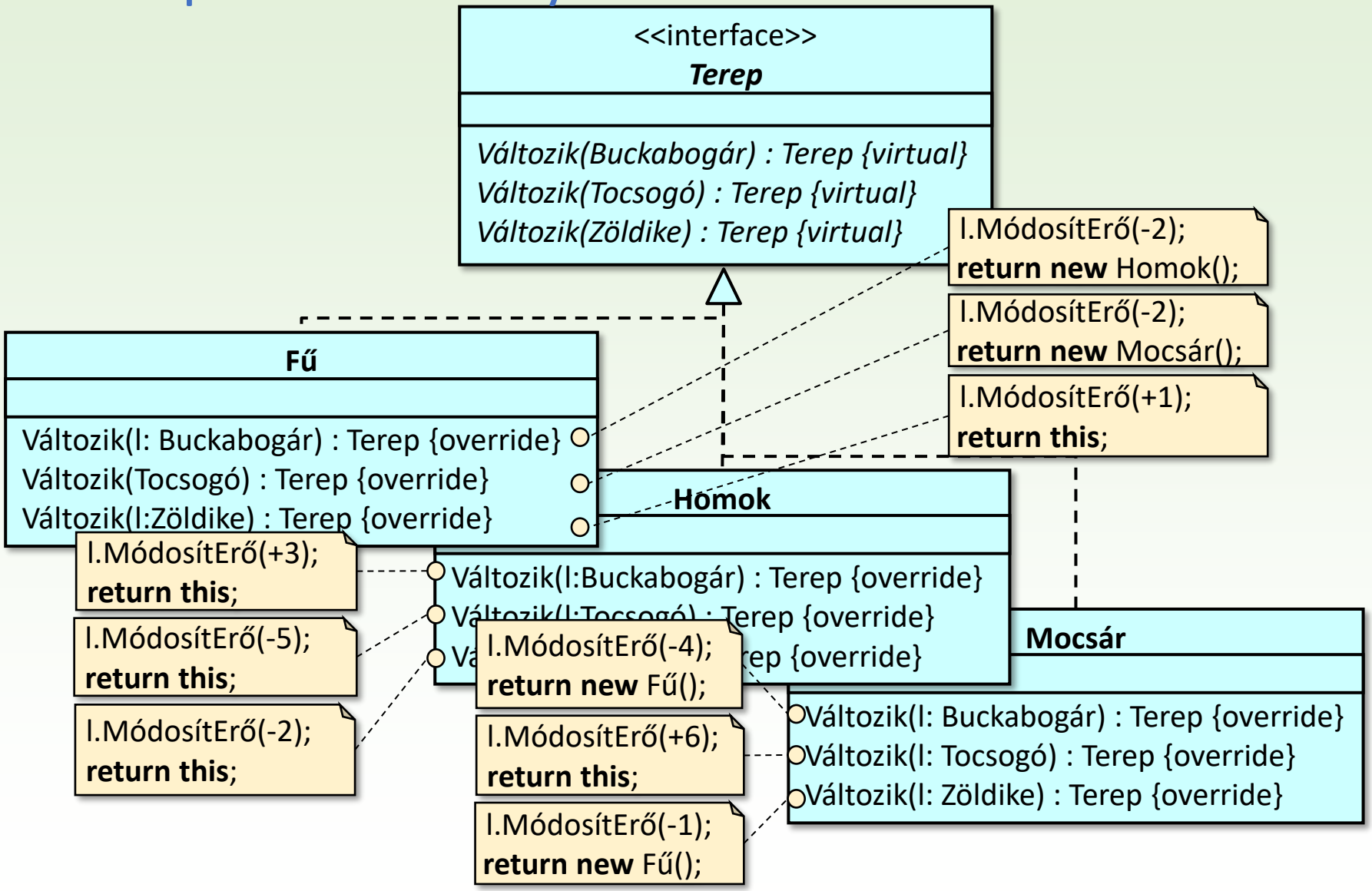


Egy terep állapotának változásai



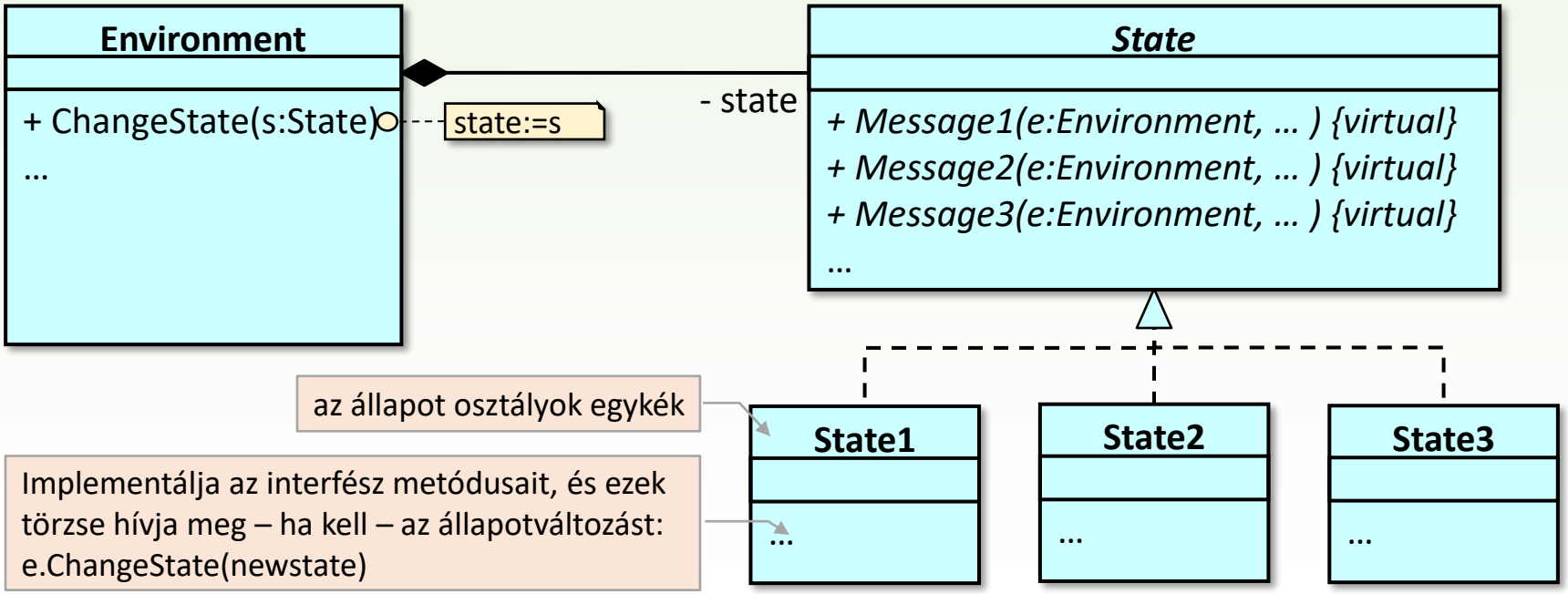
Az állapotgép megvalósításánál a Terep osztály Változik() metódusa egy 9 ágú elágazást tartalmazna. Ez nem felel meg az Open-Closed elvnek. Jobb lenne a Terep osztály alosztályaiban bevezetni 3-3 Változik() metódust, amelyeket az különböztetné meg, hogy eltérő szignatúrájuk van, hiszen eltérő fajú (típusú) lényt kapnak paraméterként.

Terepek osztályai



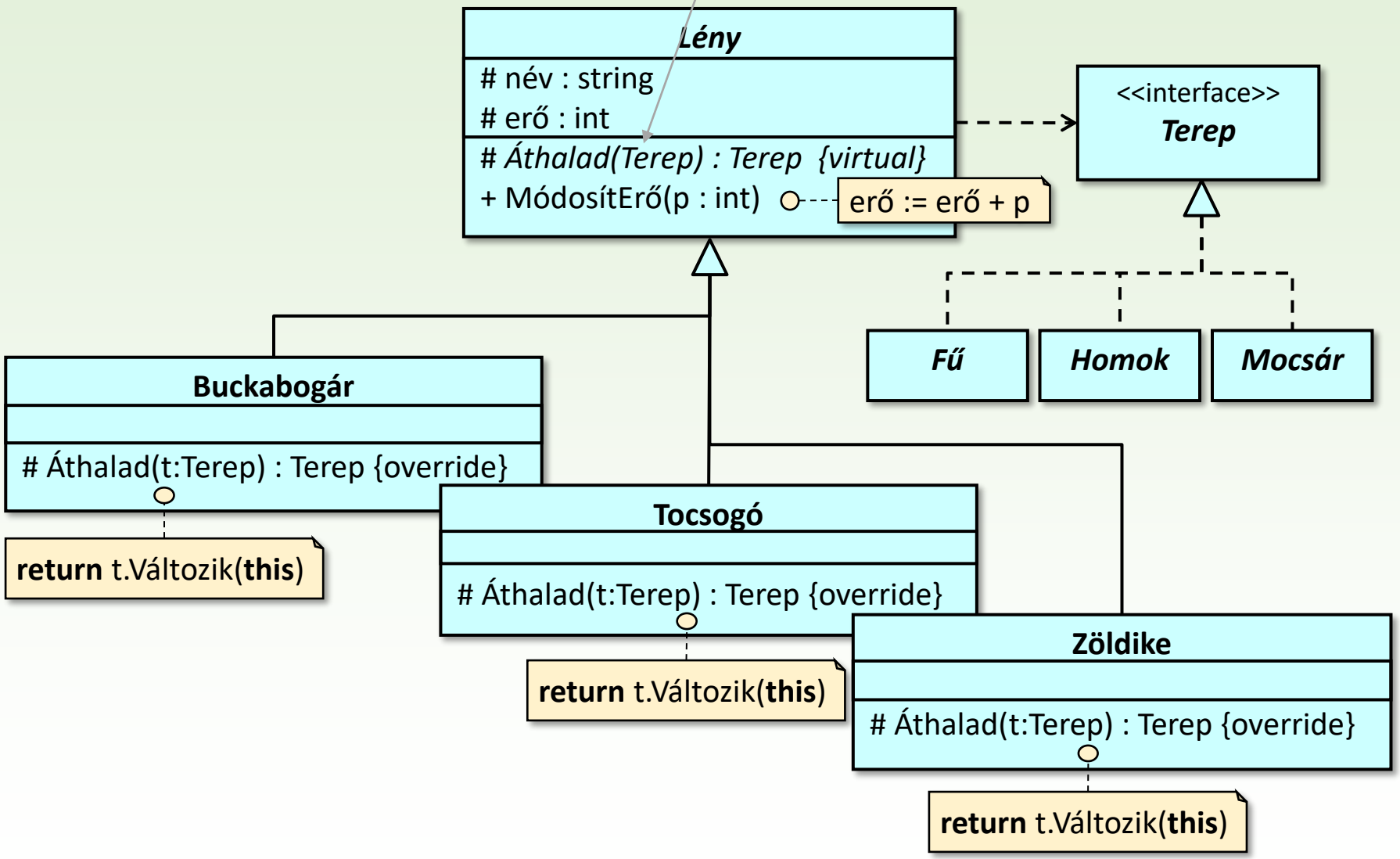
Állapot (state) tervezési minta

- Elkülönítjük egy objektumnak az egyes állapotaihoz köthető viselkedését úgy, hogy minden állapotot egy önálló objektum képviseljen. Az ilyen objektumok metódusai a környezeti objektum üzenetei, amelyek az adott állapottól függő reakciót (pl. állapotváltozást) írják le. Ez a modell nyitott újabb állapotok és az újabb üzenetek bevezetésére.



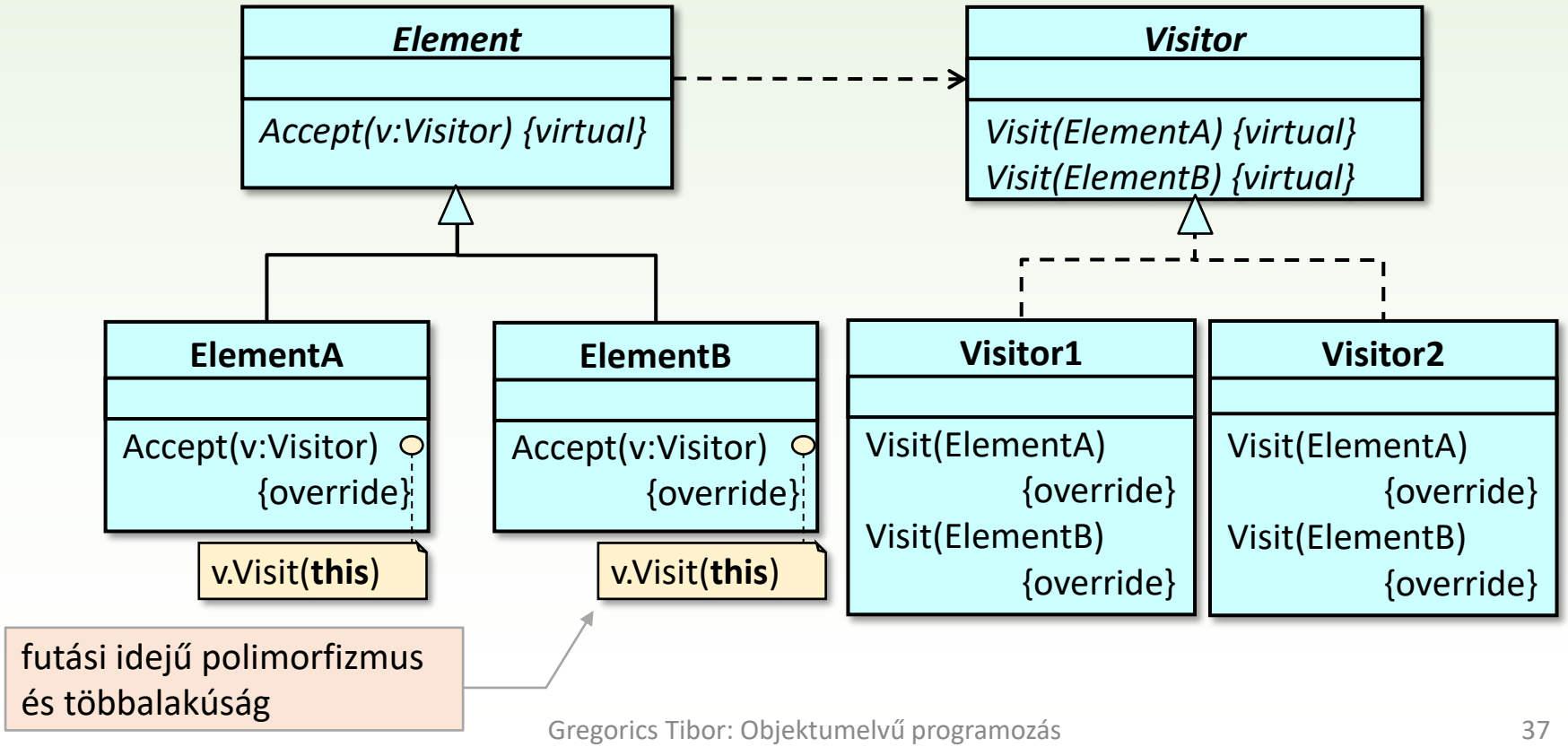
Lények osztályai

specializálni kell az Áthalad() metódust, hogy kiolvasható legyen belőle a **this** típusa, hiszen csak konkrét lény típusokkal hívható a Változik()



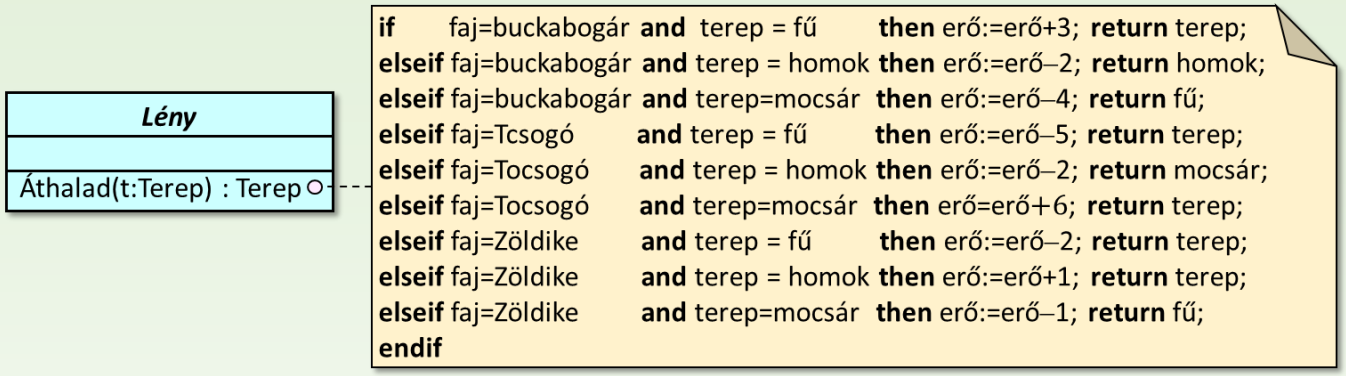
Látogató (visitor) tervezési minta

- Amikor egy metódus (Accept()) működése a saját osztályán kívül attól is függ, hogy a paramétere egy osztály melyik alosztályának objektumára hivatkozik, de nem akarjuk, hogy ez a függőség megjelenjen kódban.

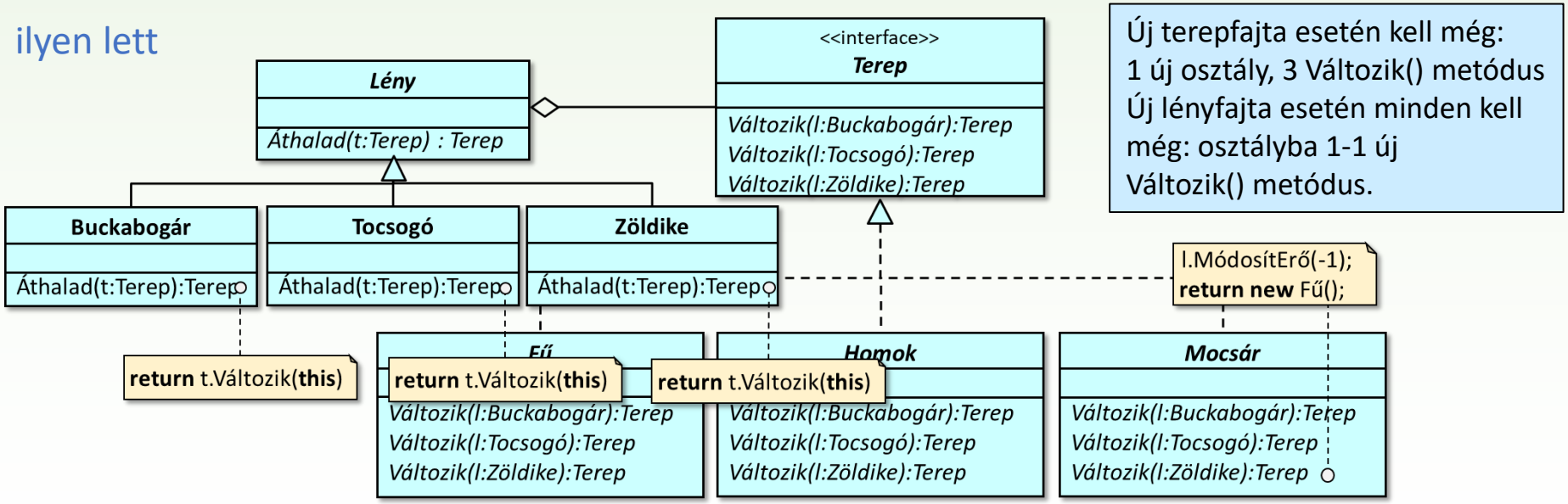


Látogató tervminta hatása

ilyen volt



ilyen lett



Új terepfajta esetén kell még:
1 új osztály, 3 Változik() metódus
Új lényfajta esetén minden kell még: osztályba 1-1 új Változik() metódus.

a modell megfelel az Open-Closed, az Interface Segregation és a Dependency Inversion elveknek

Terepek osztályai

```
class Grass : IGround
{
   Iground Change(DuneBeetle c) { c.ModifyPower(-2); return new Sand(); }
   IGround Change(Squelchy c)   { c.ModifyPower(-2); return new Marsh(); }
   IGround Change(Greenfinch c) { c.ModifyPower(1);  return this; }
}
```

```
class Sand : IGround
{
   Iground Change(DuneBeetle c) { c.ModifyPower(3);  return this; }
   IGround Change(Squelchy c)   { c.ModifyPower(-5); return this; }
   IGround Change(Greenfinch c) { c.Modifyower(-2);  return this; }
}
```

```
class Marsh : IGround
{
   Iground Change(DuneBeetle c) { c.ModifyPower(-4); return new Grass(); }
   IGround Change(Squelchy c)   { c.ModifyPower(6);  return this; }
   IGround Change(Greenfinch c) { c.ModifyPower(-1); return new Grass(); }
}
```

Kritika a hatékonyságról:
A Change() mindig új terep objektumot példányosít, valahányszor egy terep változik, pedig elég lenne minden terep-típushoz egyetlen objektum, amelyre aztán több helyről is hivatkozhatunk.
A terepek legyenek egykék!

interfész

```
interface IGround
{
   Iground Change(DuneBeetle c);
   IGround Change(Squelchy c);
   IGround Change(Greenfinch c);
}
```

Terepek osztályai egykék

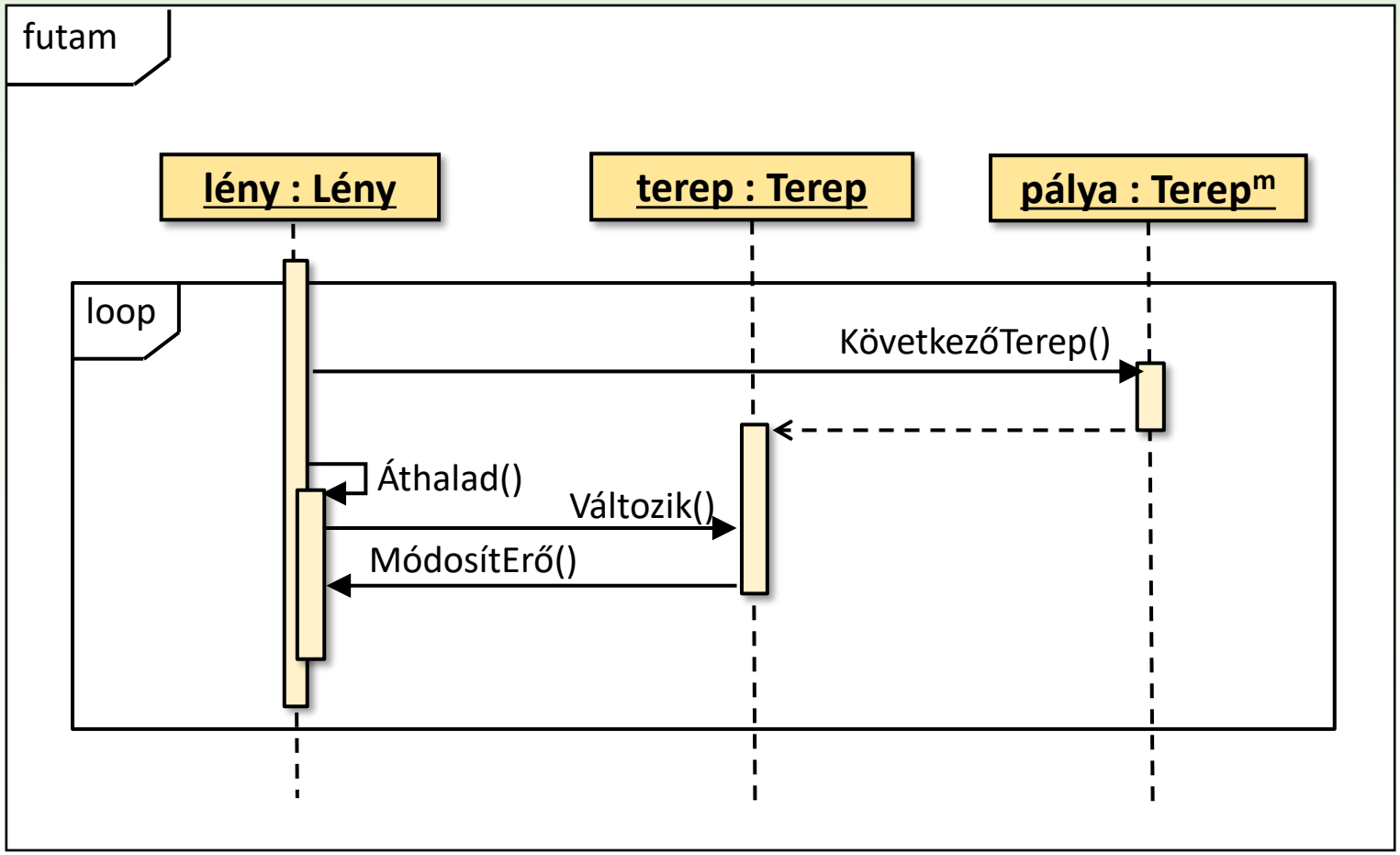
```
class Grass : IGround
{
   Iground Change(DuneBeetle c) { c.ModifyPower(-2); return Sand.Instance(); }
   IGround Change(Squelchy c)   { c.ModifyPower(-2); return Marsh.Instance(); }
   IGround Change(Greenfinch c) { c.ModifyPower(1);  return this; }
}
```

```
class Sand : IGround
{
   Iground Change(DuneBeetle c) { c.changePower(3);  return this; }
   IGround Change(Squelchy c)   { c.changePower(-5); return this; }
   IGround Change(Greenfinch c) { c.ChangePower(-2); return this; }
}
```

```
class Marsh : IGround
{
   Iground Change(DuneBeetle c) { c.changePower(-4); return Grass.Instance(); }
   IGround Change(Squelchy c)   { c.changePower(6);  return this; }
   IGround Change(Greenfinch c) { c.ChangePow

   private Grass() { }
   private static Marsh instance = null;
   public static Marsh Instance()
   {
      if (instance == null) instance = new Marsh()
      return instance;
   }
}
```

```
interface IGround
{
   Iground Change(DuneBeetle c);
   IGround Change(Greenfinch c);
   IGround Change(Squelchy c);
}
```

Egy lény futama



Egy lény futama

pálya' ~ a pálya állapota a lény futama előtt
$lény_0$ ~ a lény állapota a lény futama előtt
$lény_{j-1}$ ~ a lény a *j*-dik terepen való áthaladás előtt
$lény_j$ ~ a lény a *j*-dik terepen való áthaladás után

Egy lény (amíg él) a pálya egyes terepein sorban áthalad: minden lépése megváltoztathatja az adott terepet, miközben a lény maga is átalakul.

A = (pálya: $Terep^m$, lény: Lény)

Ef = (pálya = pálya' $\wedge$ lény = $lény_0$)

Uf = (lény = $lény_m$ $\wedge$
$\forall j \in [1..m]$: Él($lény_{j-1}$) $\rightarrow$ ($lény_j$, pálya[j]) = Áthalad($lény_{j-1}$, pálya'[j]) $\wedge$
$\neg$Él($lény_{j-1}$) $\rightarrow$ ($lény_j$, pálya[j]) = ($lény_{j-1}$, pálya'[j]))

Sorozatos átalakítás: lény = $\ominus_{j=1..m}$ $lény_j$
ahol $\ominus$: Lény × Lény $\rightarrow$ Lény
és új := régi $\ominus$ új
Összefűzés: pálya = $\oplus_{j=1..m}$ <pálya[j]>

futam - dupla összegzés

t:enor(E) ~ j = 1 .. m (terepek felsorolása)
H,+,0 ~ Lény×$Terep^m$, ($\ominus$, $\oplus$), ($lény_0$, <>)
f(e) ~ Áthalad(lény, pálya[j]) ha lény.Él()
(lény, pálya[j]) különben

- a két összegzés közös ciklusba vonható össze, amely korábban is leállhat, ha a lény már nem él
- az új pálya terepeit összefűzés helyett a régi pálya mezőinek változtatásával alakítjuk ki

lény.Futam(pálya)

lény, pálya := Futam(lény, pálya)

j := 1

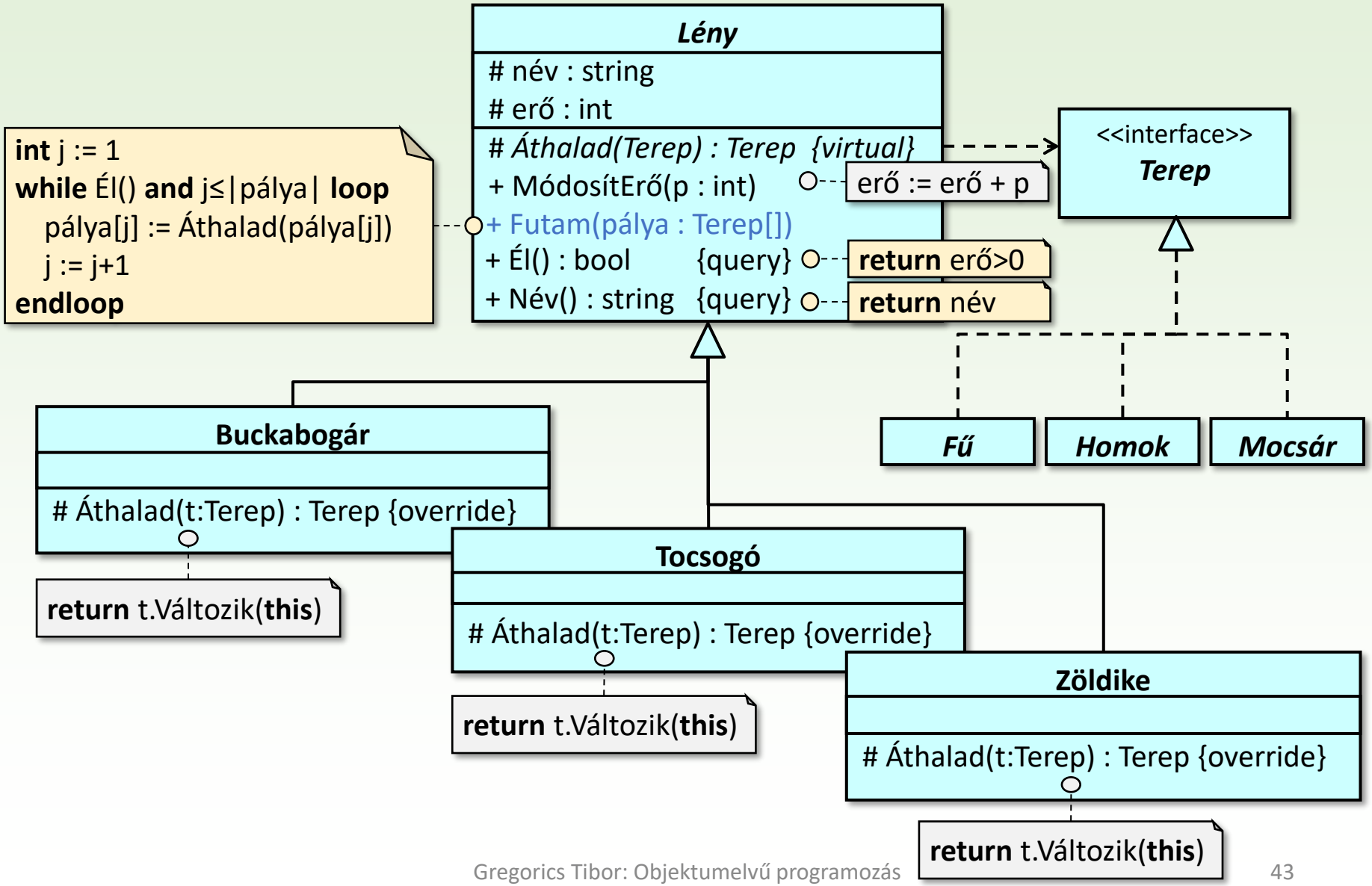
lény.Él() $\wedge$ j ≤ m

lény, pálya[j] := Áthalad(lény, pálya[j])

j := j+1

pálya[j] := lény.Áthalad(pálya[j])

Újabb metódusok a Lény osztályban



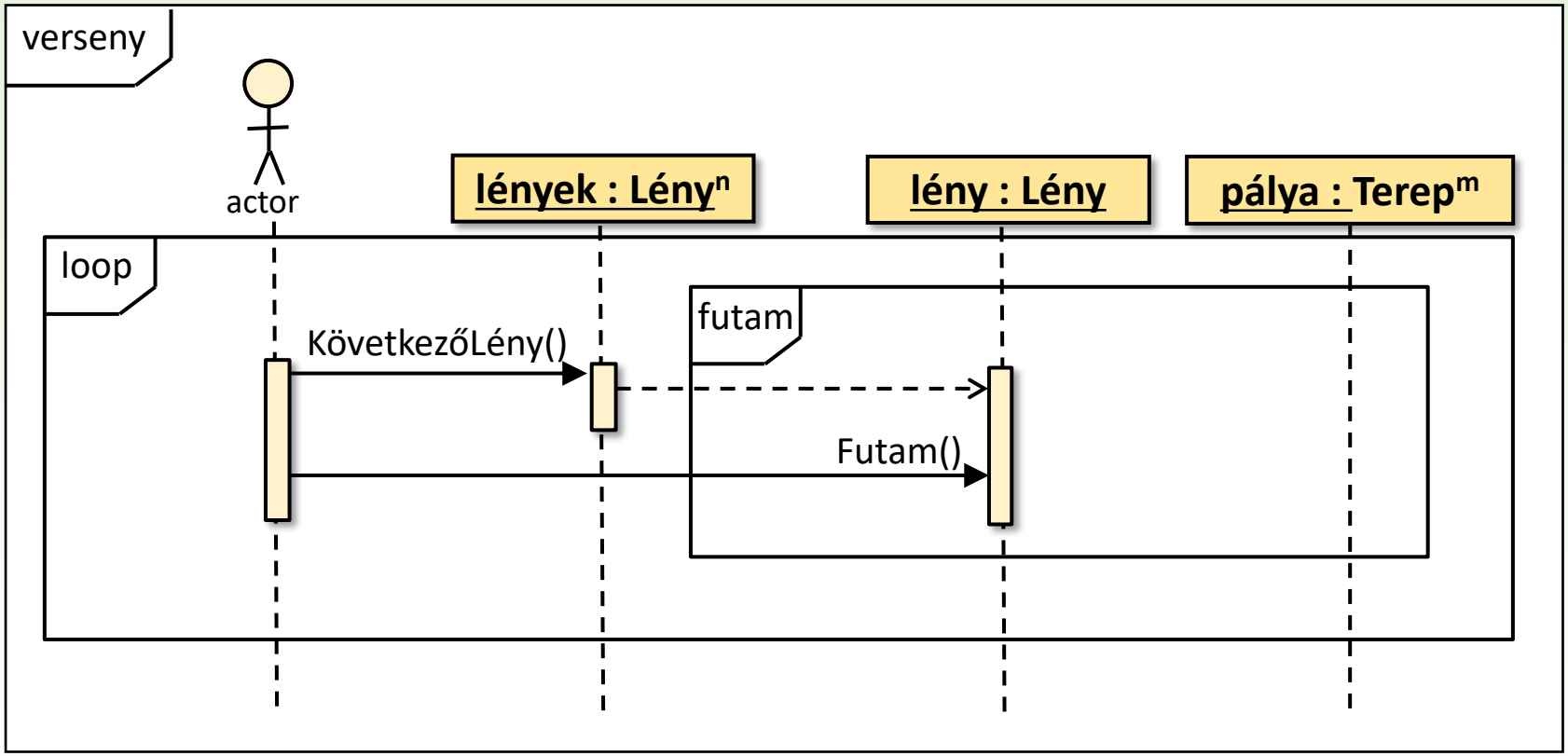
```
abstract class Creature
{
    public string Name { get; }
    protected int power;
    public void ModifyPower(int e) { power += e; }
    public bool Alive()  { return power > 0; }
    protected Creature(string str, int e = 0)  { name = str; power = e; }

    protected abstract IGround Traverse(IGround court);
    public void Race(ref List<IGround> courts)
    {
        for(int j = 0; Alive() && j<courts.Count; ++j) courts[j] = Traverse(courts[j]);
    }
}
```

```
class DuneBeetle : Creature
 {
    public DuneBeetle(string str, int e = 0) : base(str, e) { }
    protected override IGround Traverse(IGround court)
    { return court.Change(this); }
 }
```

```
class Squelchy : Creature
 {
    public Squelchy(string str, int e = 0) : base(str, e) { }
    protected override IGround Traverse(IGround court)
    { return court.Change(this); }
}
```

```
class Greenfinch : Creature
 {
    public Greenfinch(string str, int e = 0) : base(str, e) { }
    protected override IGround Traverse(IGround court)
    { return court.Change(this); }
 }
```

Lények versenye



Lények versenye

lények$_0$ ~ a lények állapota a verseny előtt
pálya$_0$ ~ a pálya állapota a verseny előtt
pálya$_{i-1}$ ~ a pálya az *i*-dik lény áthaladása előtt
pálya$_i$ ~ a pálya az *i*-dik lény áthaladása után

A lényeket egymás után elindítjuk a pályán;
miközben átalakítják a pályát, maguk is változnak.
Ezután kiválogatjuk a túlélő lények neveit.

A = (lények: Lény^n, pálya: Terep^m, túlélők: String*)

Ef = (lények = lények$_0$ $\wedge$ pálya = pálya$_0$)

Uf = (pálya = pálya$_n$ $\wedge$ $\forall i \in [1..n]$: (lények[i], pálya$_i$) = Futam(lények$_0$[i], pálya$_{i-1}$) $\wedge$

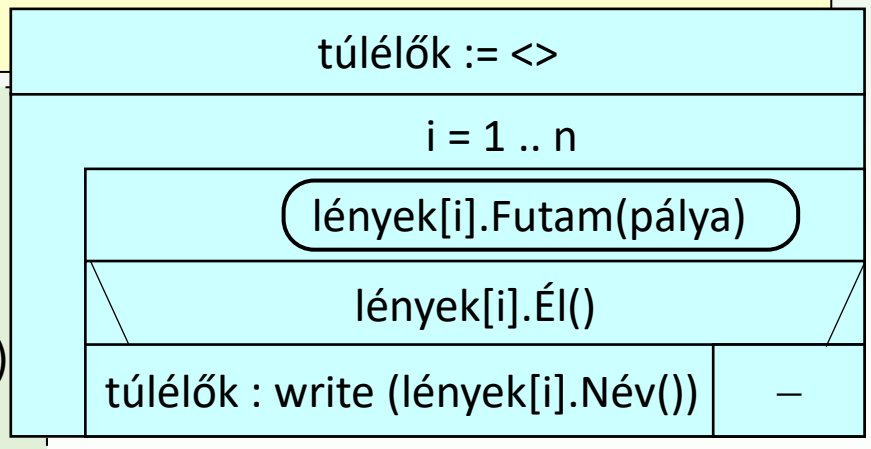
$\wedge$ túlélők = $\bigoplus_{i=1..n}$ <lények[i].név()>)

lények[i].él()

Összefűzés: lények = $\bigoplus_{i=1..n}$ <lények[j]>
Sorozatos átalakítás: pálya = $\ominus_{i=1..n}$ pálya$_i$
ahol $\ominus$: Pálya × Pálya → Pálya
és új := régi $\ominus$ új

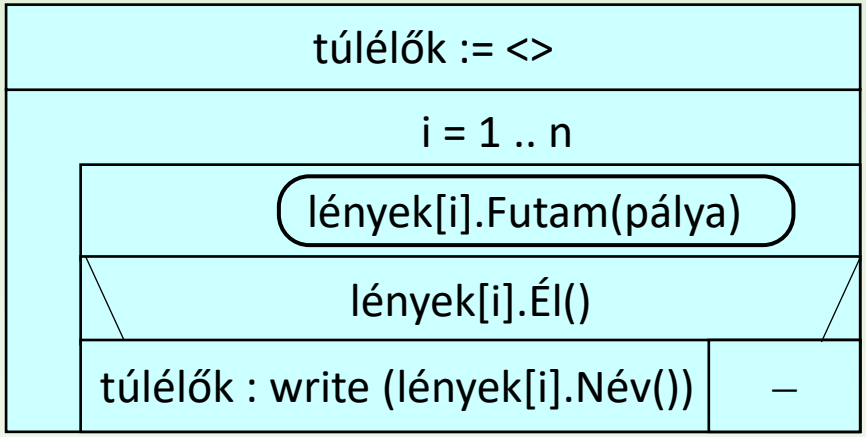
Kiválogatás

t:enor(E) ~ i = 1 .. n (lények felsorolása)
verseny - dupla összegzés
összefűzés és sorozatos átalakítás:
f(e) ~ Futam(lények[i], pálya)
H,+,0 ~ Lény^n×Terep^m, ($\bigoplus$, $\ominus$), (<>, pálya$_0$)
kiválogatás - összegzés
f(e) ~ <lények[i].Név()> ha lények[i].él()
H,+,0 ~ String*, $\bigoplus$, <>



- a három összegzés közös ciklusba vonható össze
- a megváltozott lények tömbjét összefűzés helyett a régi tömb megváltoztatásával alakítjuk ki

Főprogram




```
// populating creatures
 ...
// populating courts
 ...
// competition

for (int i = 0; i < n; ++i)
{
   creatures[i].Race(ref courts);
   if (creatures[i].Alive()) Console.WriteLine(creatures[i].Name);
}
```

Itt is jól jön a futási idejű polimorfizmus:
a Race() működése attól függ, hogy a creatures[i] milyen fajtájú lény, azaz mi az osztálya.

Lények létrehozása

input.txt

```
4
S plash 20
G greenish 10
D bug 15
S sponge 20
10
gmsgmgsgsm
```

```
// populating creatures
reader.ReadInt(out int n);  // number of creatures
List<Creature> creatures = new ();
for (int i = 0; i < n; ++i)
{
   char[] separators = new char[] { ' ', '\t' };
   Creature creature = null;
   if (reader.ReadLine(out line))
   {
      string[] tokens = line.Split(separators,
                                   StringSplitOptions.RemoveEmptyEntries);
      char ch = char.Parse(tokens[0]);
      string name = tokens[1];
      int p = int.Parse(tokens[2]);
      switch (ch)
      {
         case 'G': creature = new Greenfinch(name, p); break;
         case 'D': creature = new DuneBeetle(name, p); break;
         case 'S': creature = new Squelchy(name, p);   break;
      }
   }
   creatures.Add(creature);
}
```