

Testek és lények

1.rész

Testek térfogata

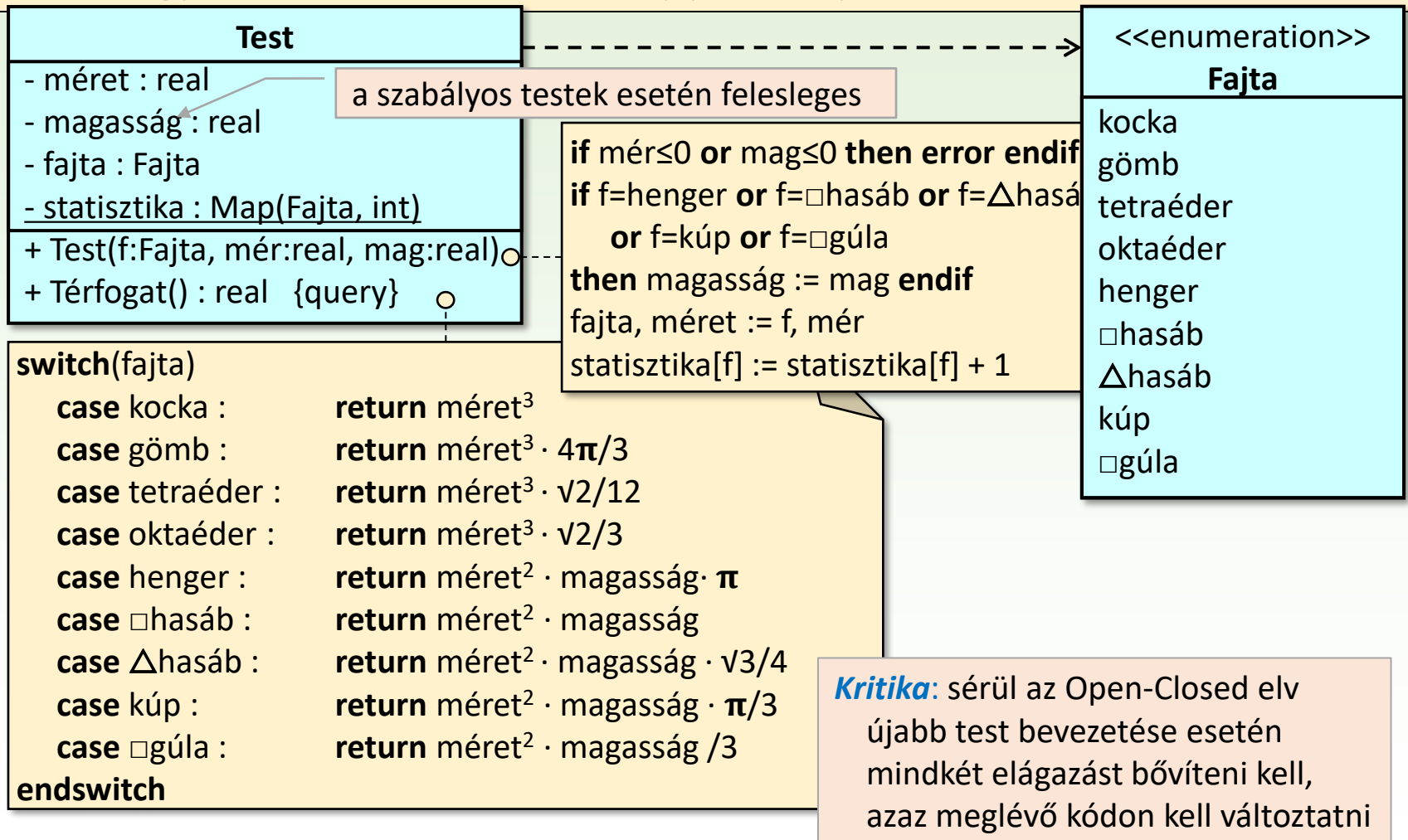
Gregorics Tibor

gt@inf.elte.hu

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

Feladat és egy ügyetlen modellje

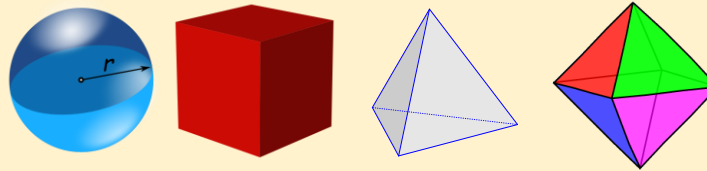
Készítsünk programot, amellyel különféle testeket hozhatunk létre azért, hogy kiszámolhassuk a térfogatukat. Eközben bármikor készíthessünk statisztikát arról, hogy a különféle testekből hány példánnyal rendelkezünk már.



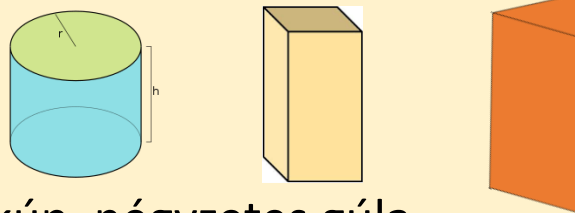
Objektum hierarchia

A testeket az alábbi módon csoportosíthatjuk:

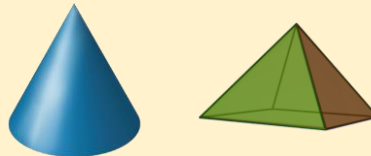
- szabályos testek: gömb, kocka, tetraéder, oktaéder;



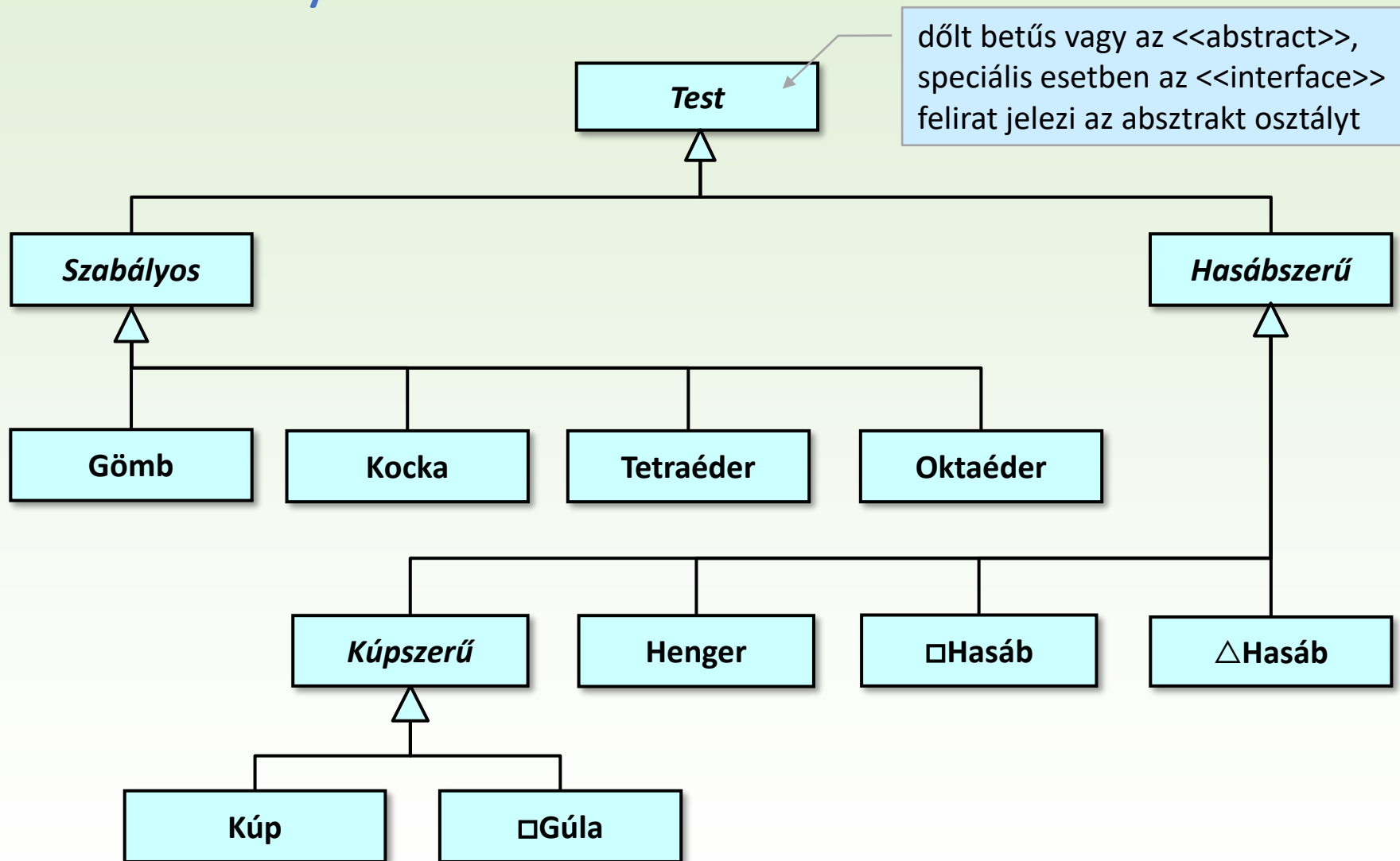
- prizmatikus testek: henger, négyzet és szabályos háromszög alapú hasáb;



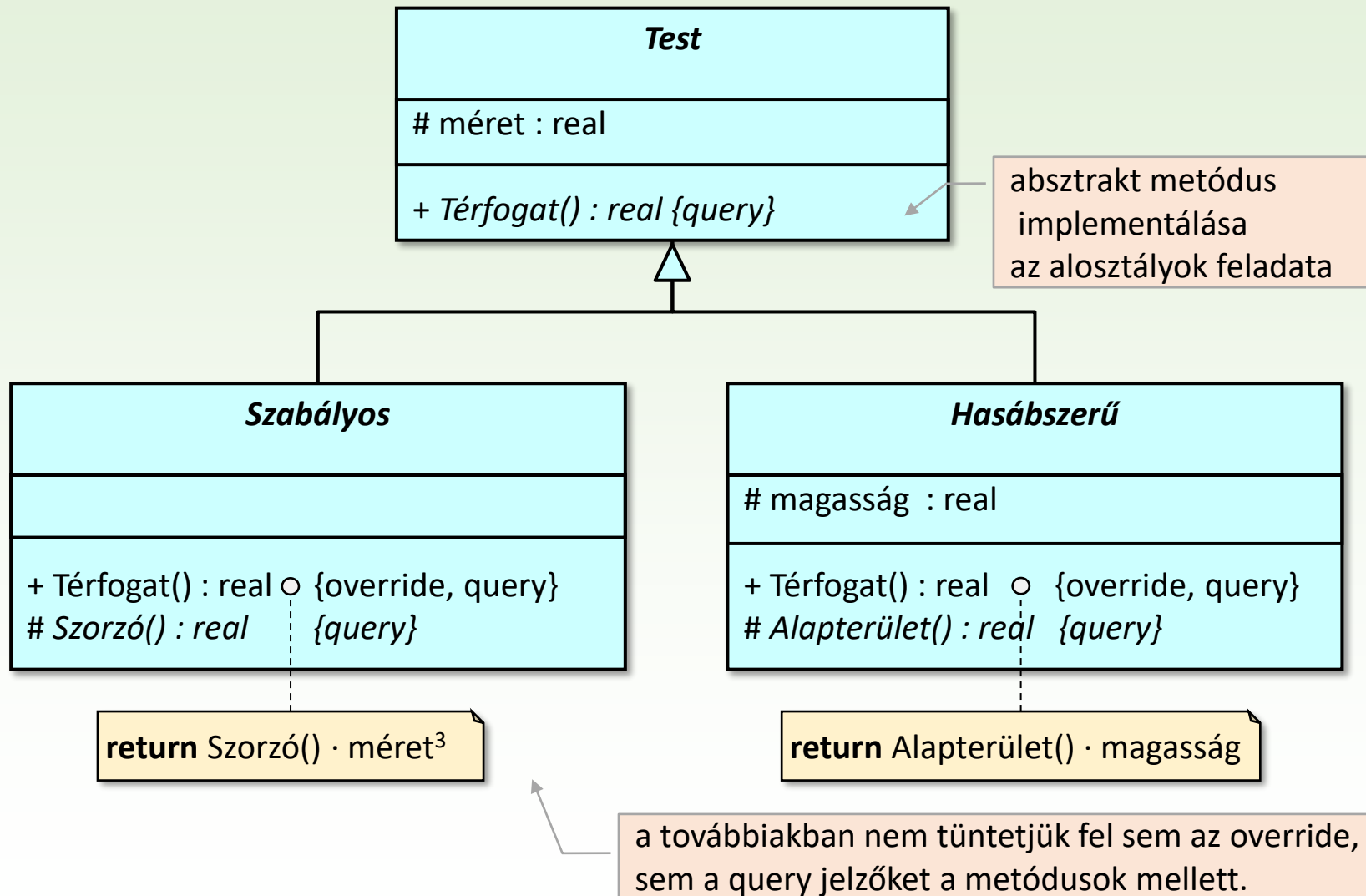
- gúlaszerű testek: kúp, négyzetes gúla.



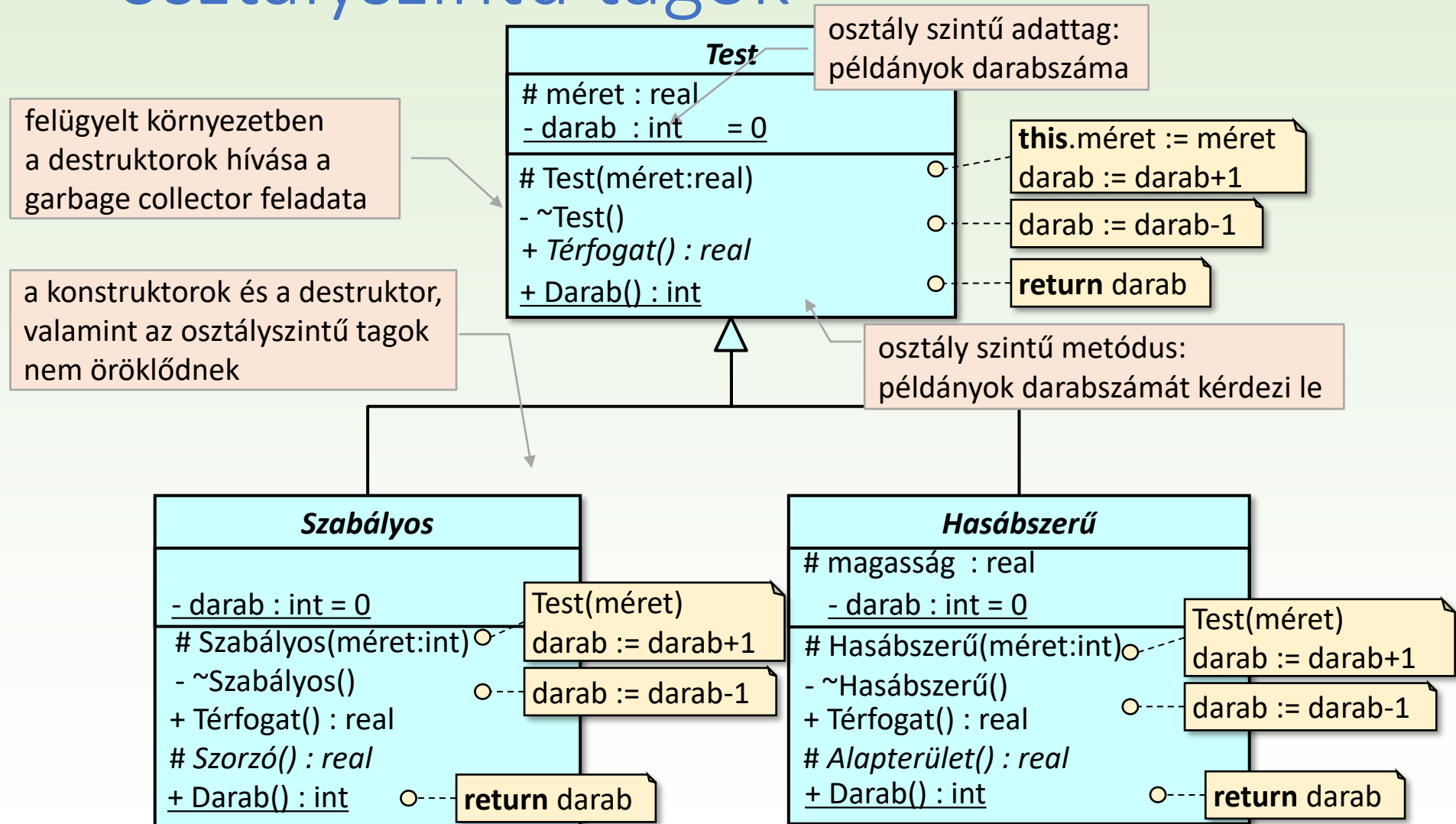
Osztály hierarchia



Absztrakt testek osztályai



Konstruktorok, destruktorkok, osztályszintű tagok



A testek őszosztálya

```
abstract class Shape
```

```
{
```

```
    protected double size;
```

```
    protected Shape(double size)
```

```
{
```

```
        this.size = size;
```

```
        ++piece;
```

```
}
```

```
    ~Shape()
```

```
{
```

```
        --piece;
```

```
}
```

```
    public abstract double Volume();
```

```
    private static int piece = 0;
```

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

```
}
```

absztrakt osztály, mert konstruktora nem publikus, és van absztrakt metódusa (Volume)

akkor fut le, amikor egy Shape-ből származtatott objektum létrejön

akkor fut le, amikor a garbage collector valamelyik Shape-ből származtatott objektumot megszünteti.

absztrakt metódus

osztályszintű adattag

osztályszintű metódus

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 * Coefficient();
    }

    protected abstract double Coefficient();

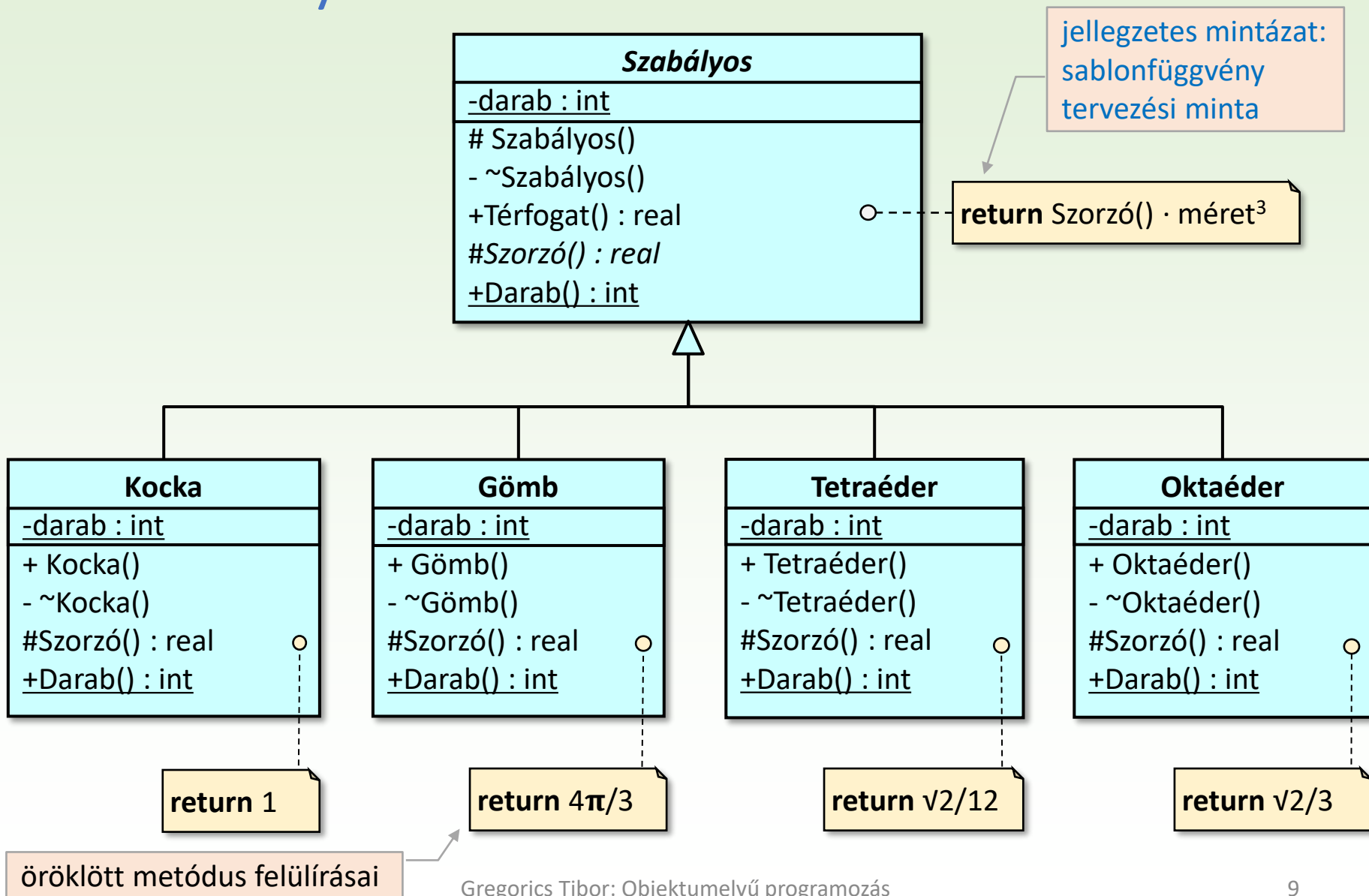
    private static int piece = 0;

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

Ősosztály adott paraméterezésű
konstruktorát hívja

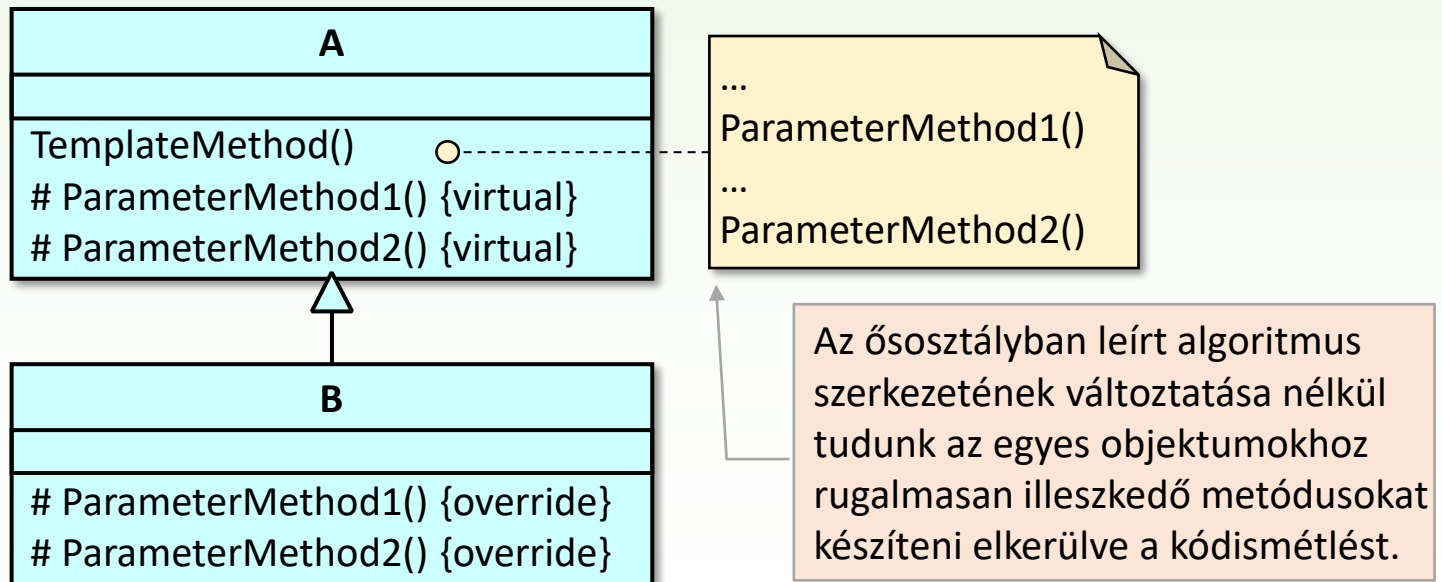
C# warning hiányolná, ha ez nem lenne

Szabályos testek



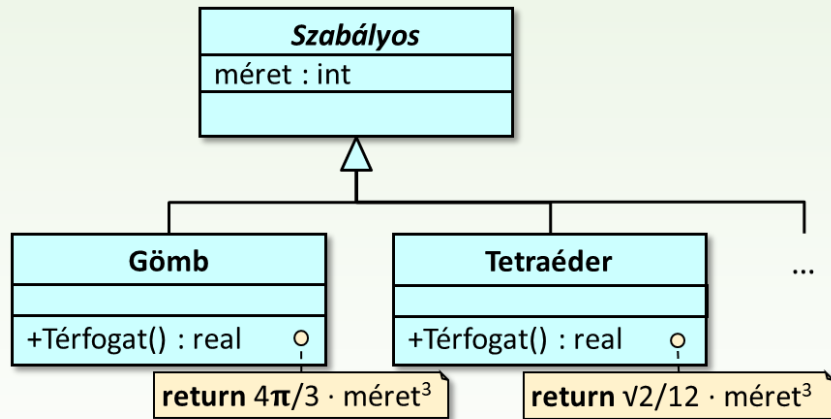
Sablonfüggvény tervezési minta (template method)

- Egy tevékenységet (sablonfüggvényt) egy őszosztály metódusaként úgy definiálunk, hogy annak speciális, altípustól függő részeit csak virtuális metódus-hívások jelzik (sablon-paraméterek), majd ezen metódusokat az alosztályokban az ott elvárt módon definiáljuk felül.

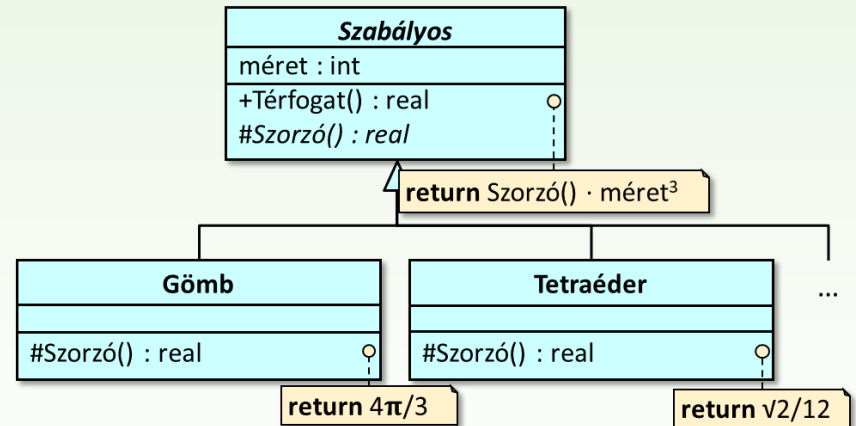


Sablontfüggvény tervminta hatása

tervminta nélkül



tervmintával



megszűnik a kódredundancia

Gömb

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


    ~Sphere() { --piece; }

    private const double coefficient = 4.0 * 3.14159 / 3.0;

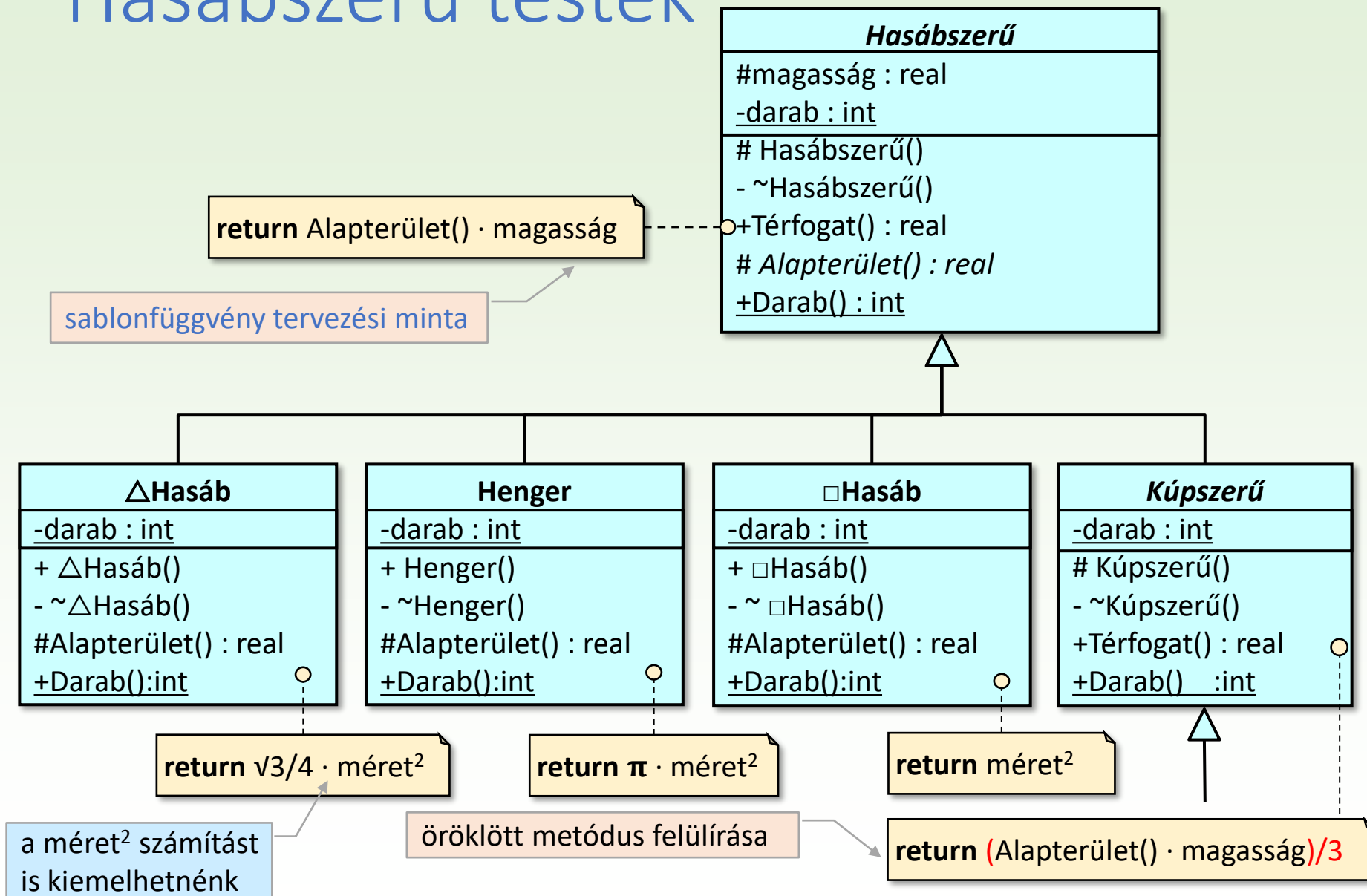
    protected override double Coefficient()
    {
        return coefficient;
    }

    private static int piece = 0;

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



Hasábszerű testek



Henger

```
abstract class Prismatic : Shape
{
    protected double height;
    public Prismatic(double size, double height) : base(size)
    { this.height = height; ++piece; }
    ~Prismatic() { --piece; }

    public override double Volume() { return Area() * height; }
    protected abstract double Area();

    private static int piece = 0;
    public static new int Piece() { return piece; }
}
```

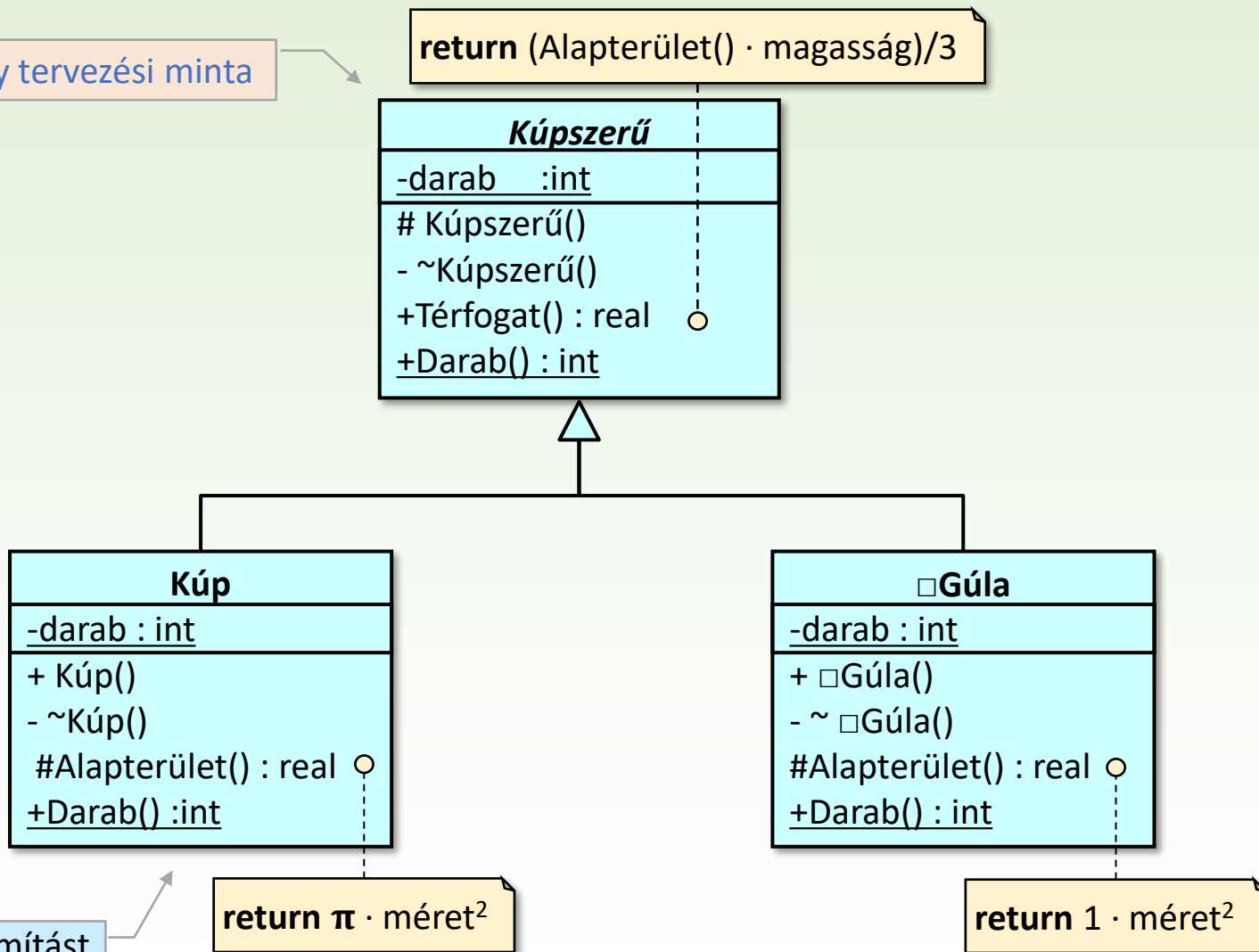
```
class Cylinder : Prismatic
{
    public Cylinder(double size, double height) : base(size, height) { ++piece; }
    ~Cylinder() { --piece; }

    protected override double Area() { return 3.14159 * size * size; }

    private static int piece = 0;
    public static new int Piece() { return piece; }
}
```

Kúpszerű testek

sablonfüggvény tervezési minta



a méret² számítást is kiemelhetnénk

Főprogram

Cube 5.0 shapes.txt
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 őosztályé.

a külön fajta testek számának kiírása

Test példányosítása

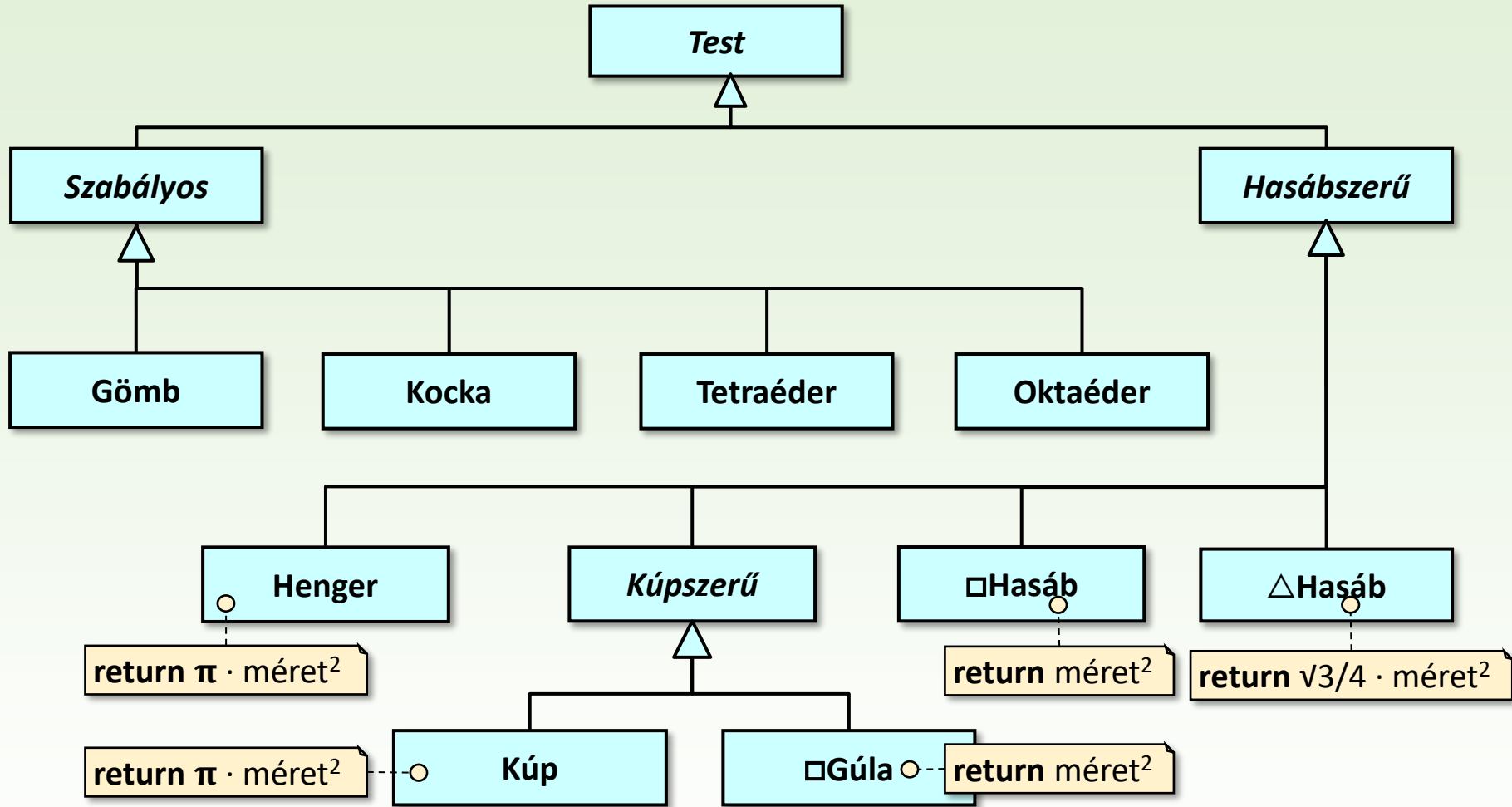
Cube 5.0 shapes.txt
Cylinder 3.0 8.0
Cylinder 1.0 10.0
Tetrahedron 4.0
SquarePyramid 3.0 10.0

```
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;
}
```

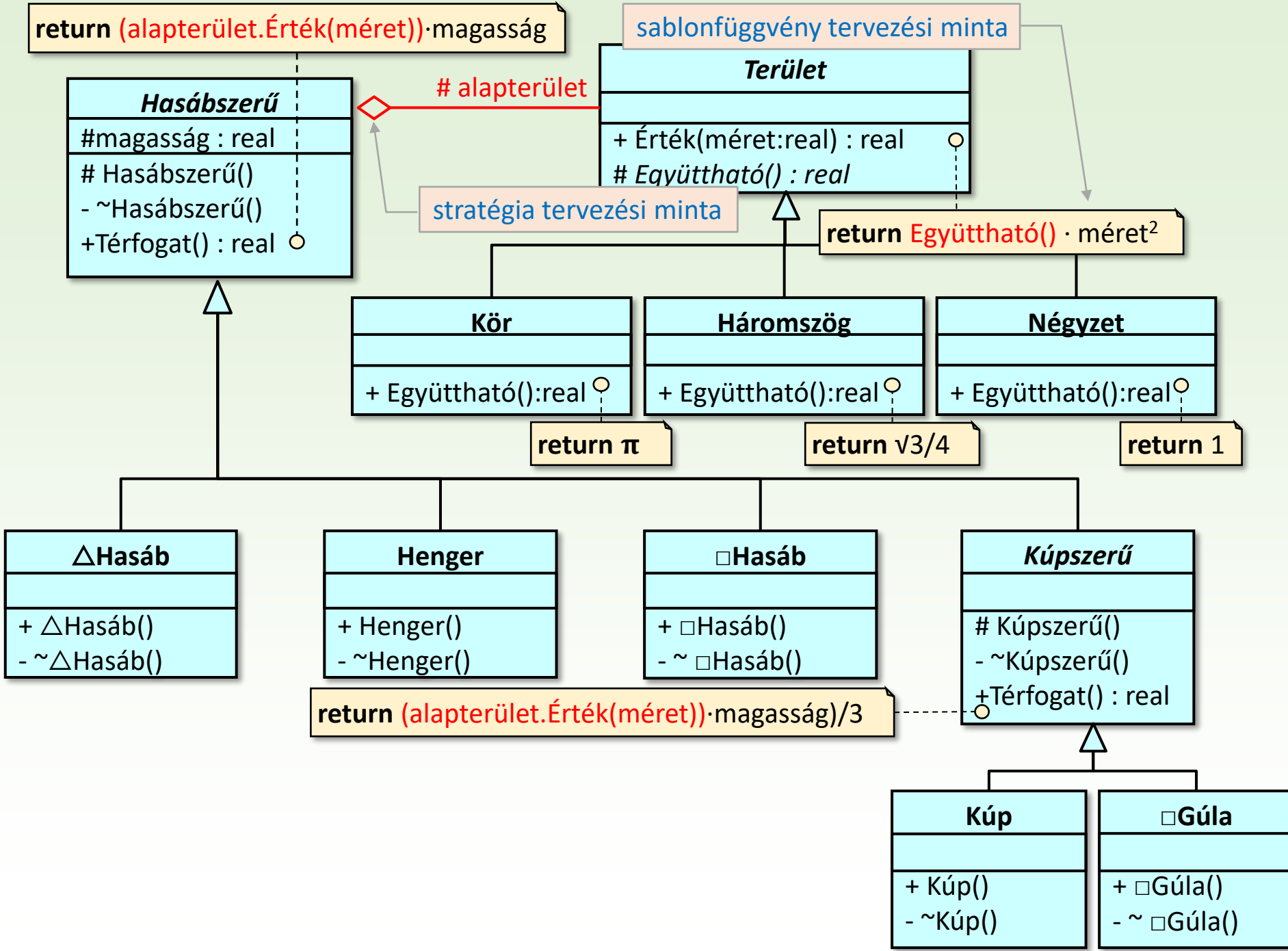
osztályszintű gyártó függvény

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

Maradt-e redundancia a modellben?



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.



Henger

```
abstract class Prismatic : Shape
```

```
{  
    protected Area area;  
    protected double height;  
    public Prismatic(double size, double height) : base(size)  
    {  
        this.height = height;  
        ++piece;  
    }  
    ~Prismatic() { --piece; }
```

az alapterület objektum hivatkozásának
tárolására alkalmas adattag

```
protected override double Volume()  
{  
    return area.Value(size) * height;  
}
```

az alapterület objektum
területszámító metódusa

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

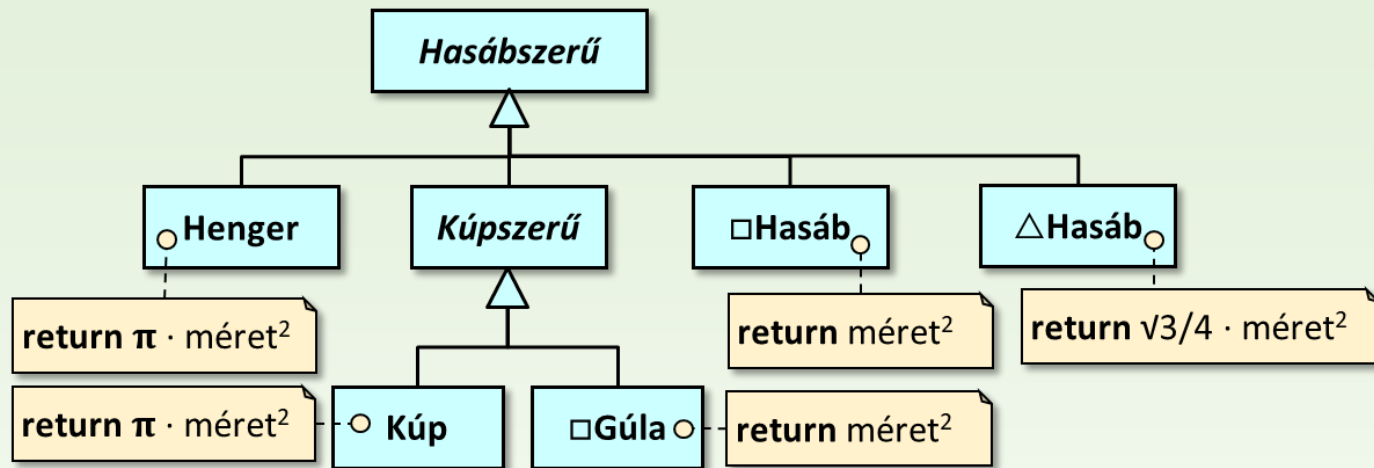
```
class Cylinder : Prismatic
```

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

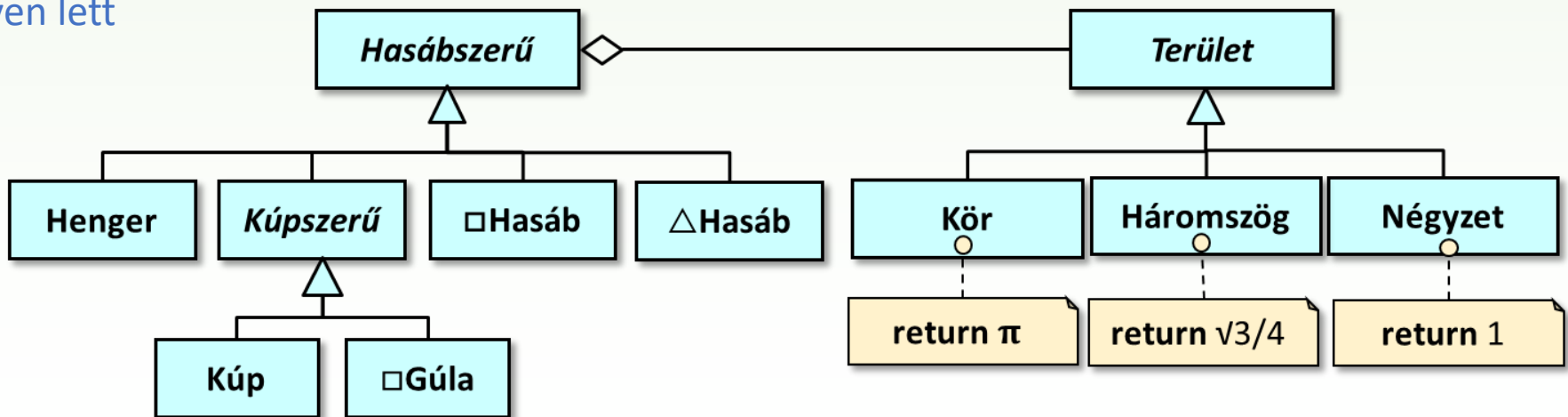
körterületet kiszámoló objektum
létrehozása és „befecskendezése”

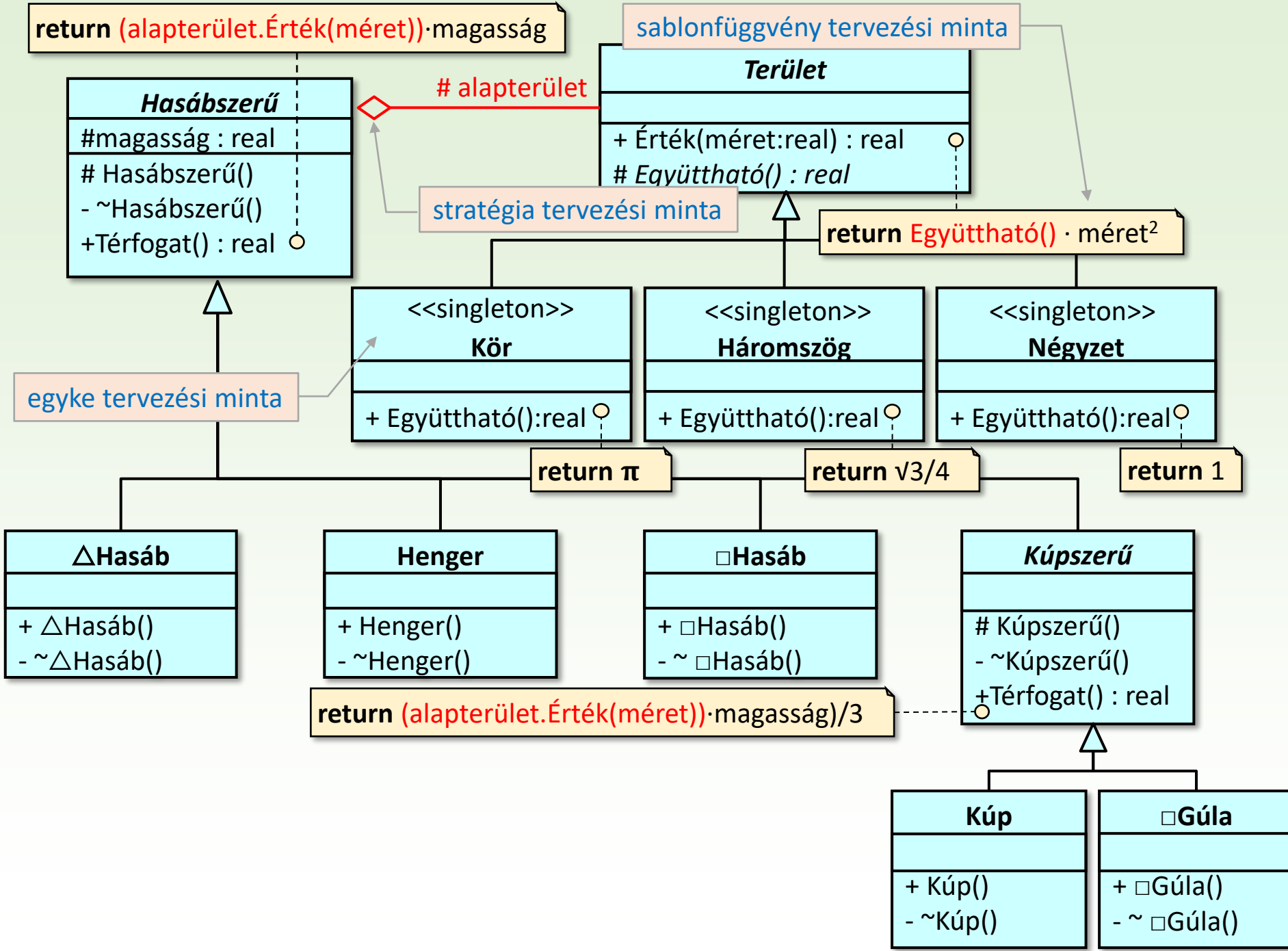
Stratégia tervminta hatása

ilyen volt



ilyen lett





Alapterület egykék

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

    protected override double Coefficient()
    {
        return 3.14159;
    }

    private static CircleArea? instance = null;

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

```
class Cylinder : Prismatic
{
    public Cylinder(double size, double height) : base(size, height)
    {
        area = CircleArea.Instance();
        ++piece;
    }
    ...
}
```

körterületet kiszámoló objektum létrehozása és „befecskendezése”

Testek és lények

2.rész

Lények túlélési versenye

Gregorics Tibor

gt@inf.elte.hu

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

Feladat

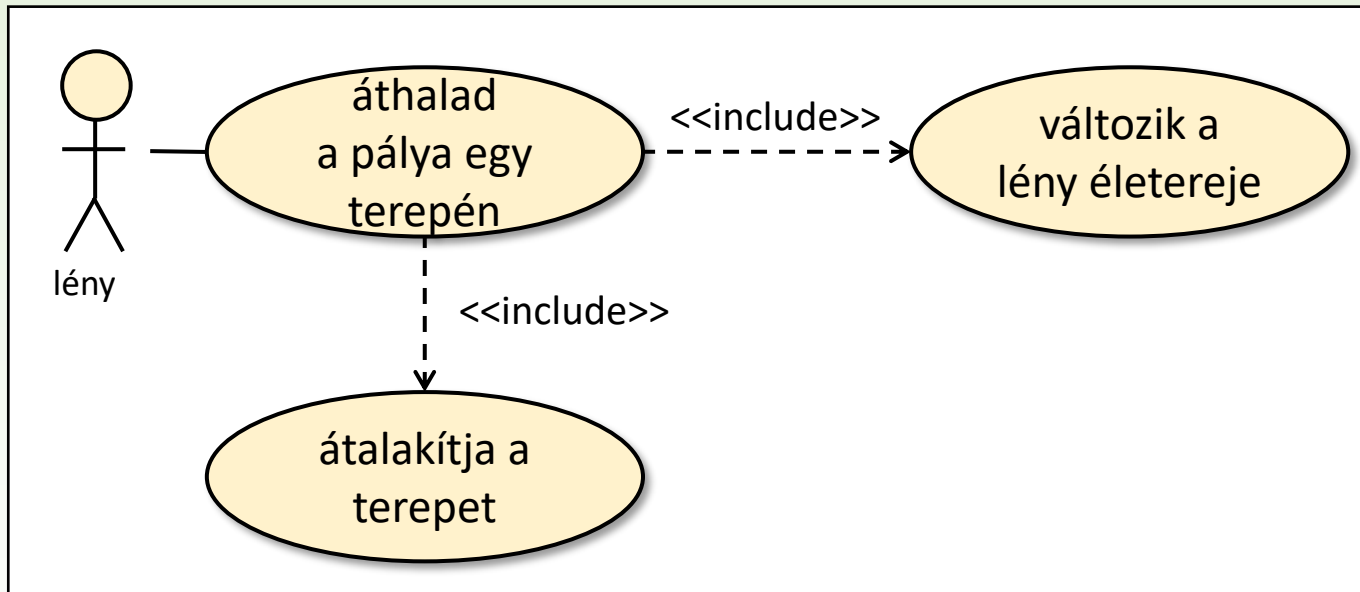
Szimuláljuk különféle **lények túlélési versenyét**.

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:** *fűvön az ereje kettővel csökken, homokon hárommal nő, mocsárban négyvel csökken; a fűvet homokká, a mocsarat fűvé alakítja, de a homokot nem változtatja meg.*
- **Tocsogó:** *fűvön az életereje kettővel, homokon öttel csökken, mocsárban hattal nő; a fűvet mocsárrá alakítja, a másik két fajta terepet nem változtatja meg.*
- **Zöldike:** *fűvön 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

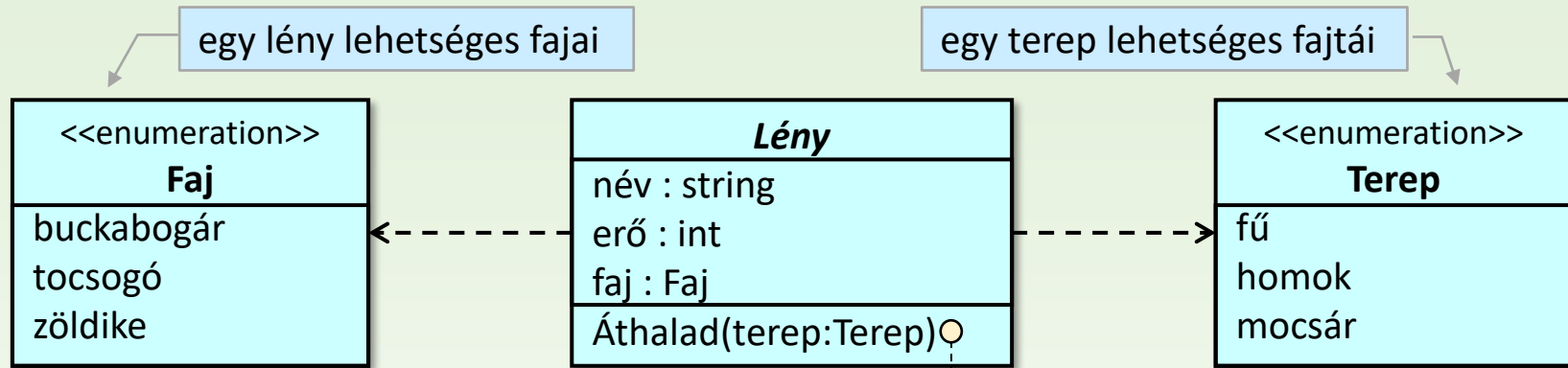


tocsogók	életerő változás	terepváltozás
fű	-2	mocsár
homok	-5	-
mocsár	+6	-



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

Osztály diagram: 1. próbálkozás

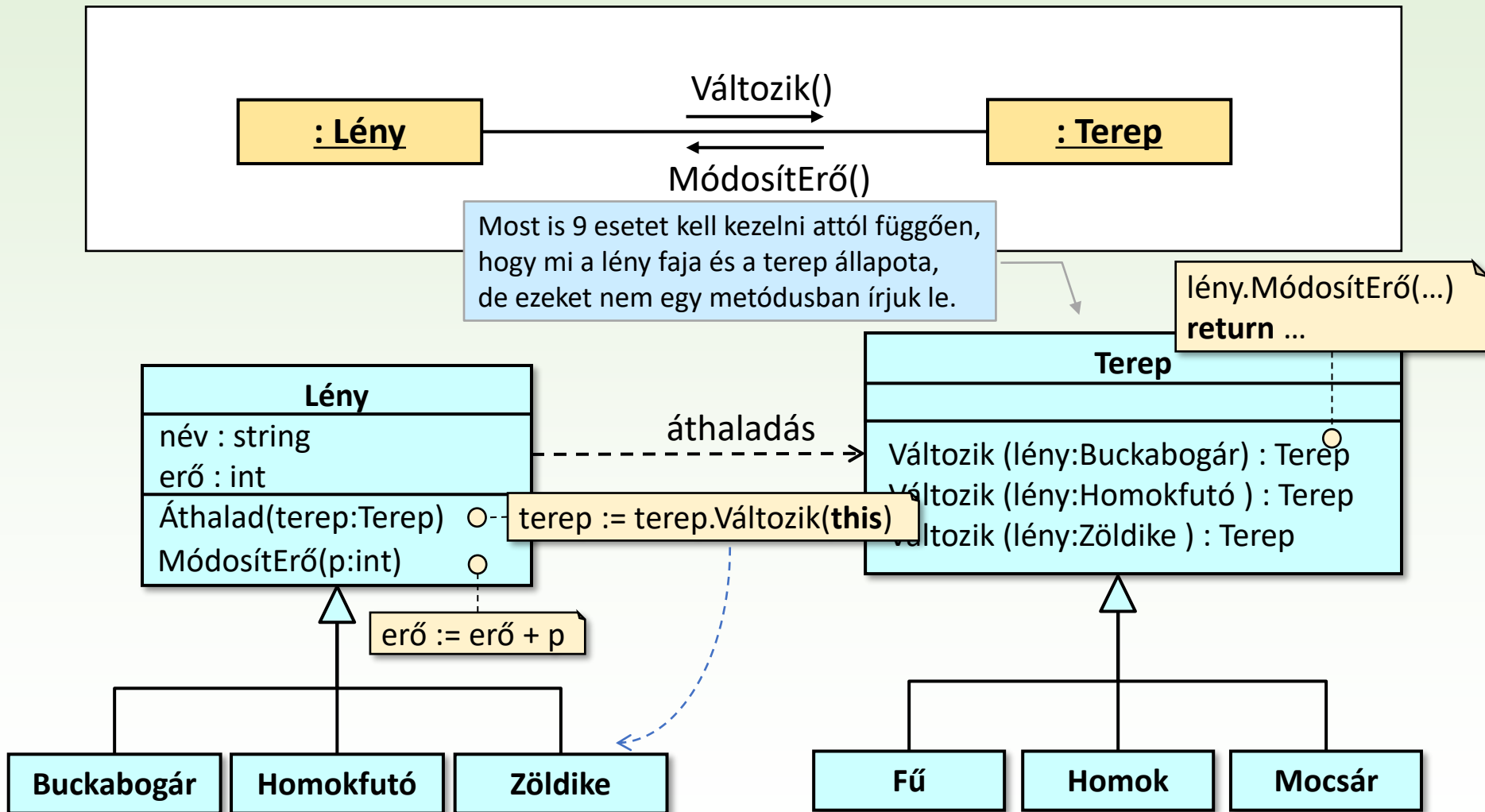


```
if faj=buckabogár and terep = fű then erő := erő-2; terep := homok;
elseif faj=buckabogár and terep = homok then erő := erő+3;
elseif faj=buckabogár and terep=mocsár then erő := erő-4; terep := fű;
elseif faj=Tocsogó and terep = fű then erő := erő-2; terep := mocsár;
elseif faj=Tocsogó and terep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

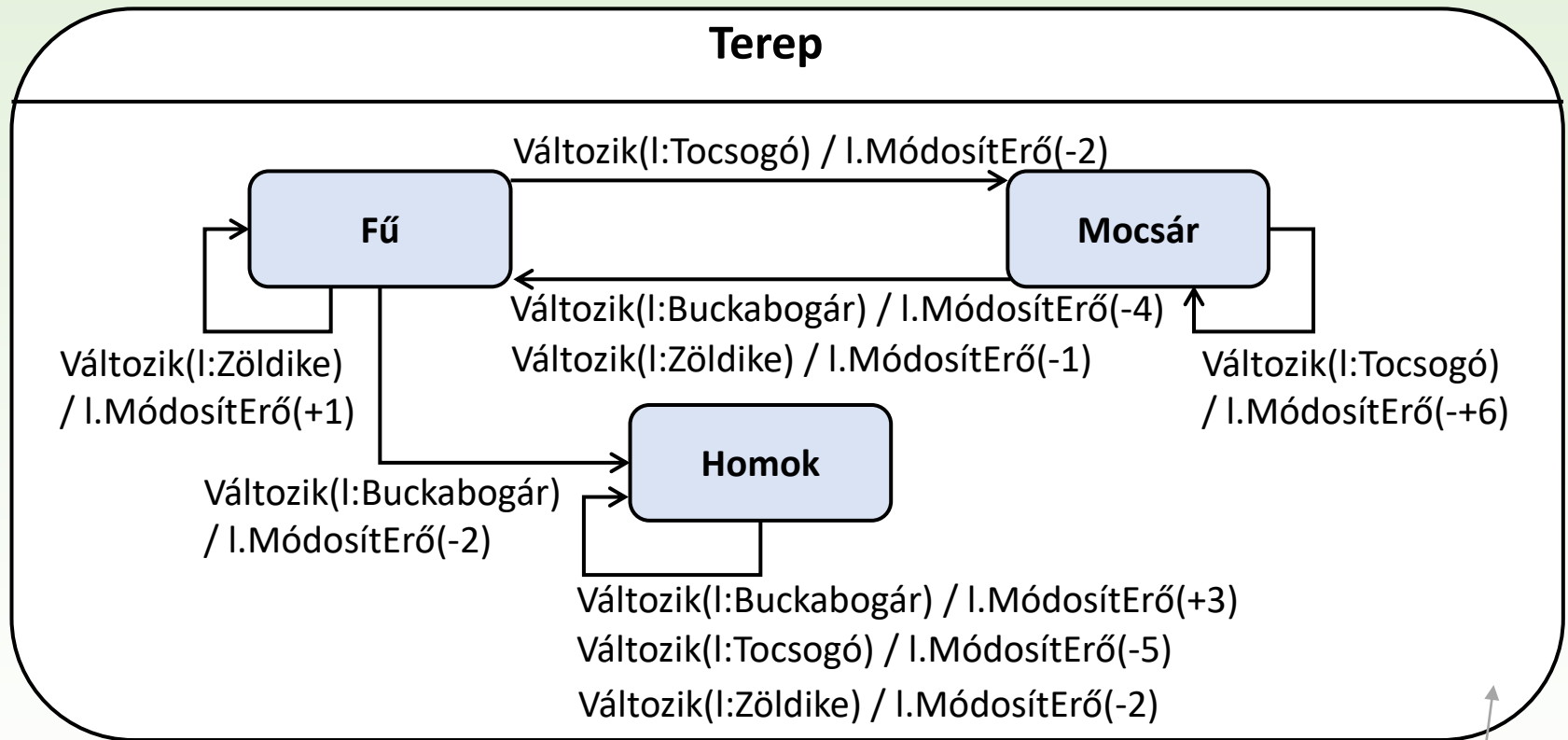
9 esetet kell leírni a lény és a terep állapotától függően

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

Osztály diagram: 2. próbálkozás

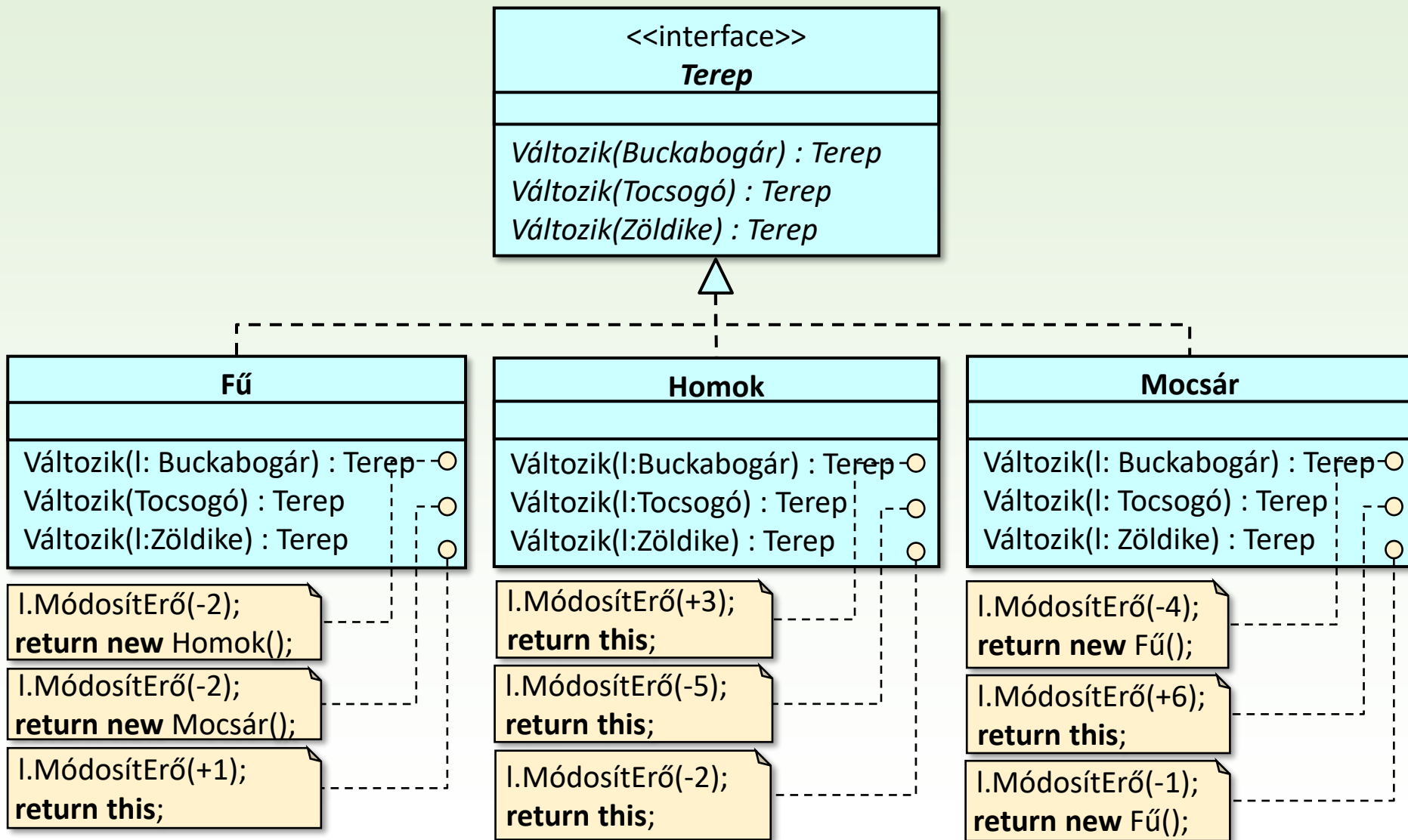


Egy terep állapotának változásai



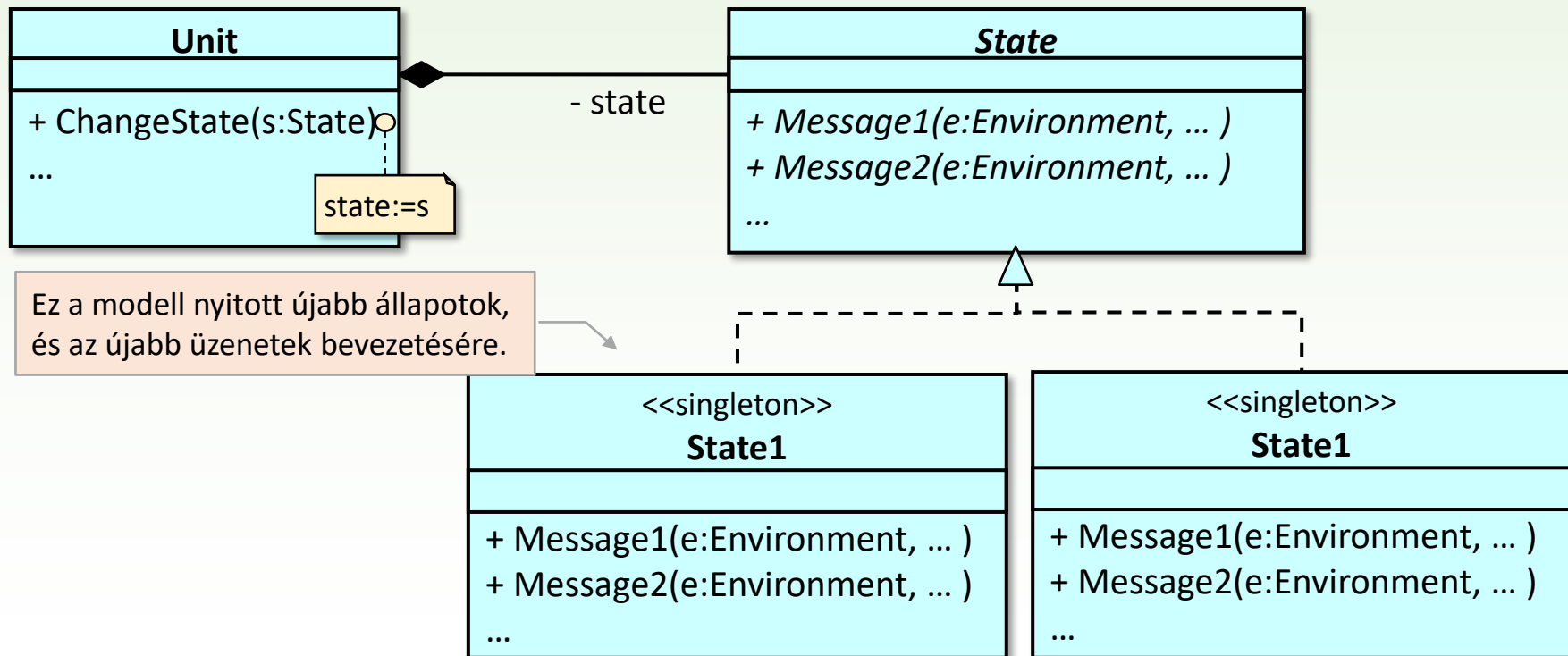
Az állapotgép megvalósításánál a Terep osztály Változik() metódusa egy 9 ágú elágazást tartalmazna. Ez nem felel meg az Open-Closed elvnek. Jobb lenne a Terep osztály alosztályaiban bevezetni 3-3 Változik() metódust, amelyeket az különböztetné meg, hogy eltérő szignatúrájuk van, hiszen eltérő fajú (típusú)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

Terepek osztályai

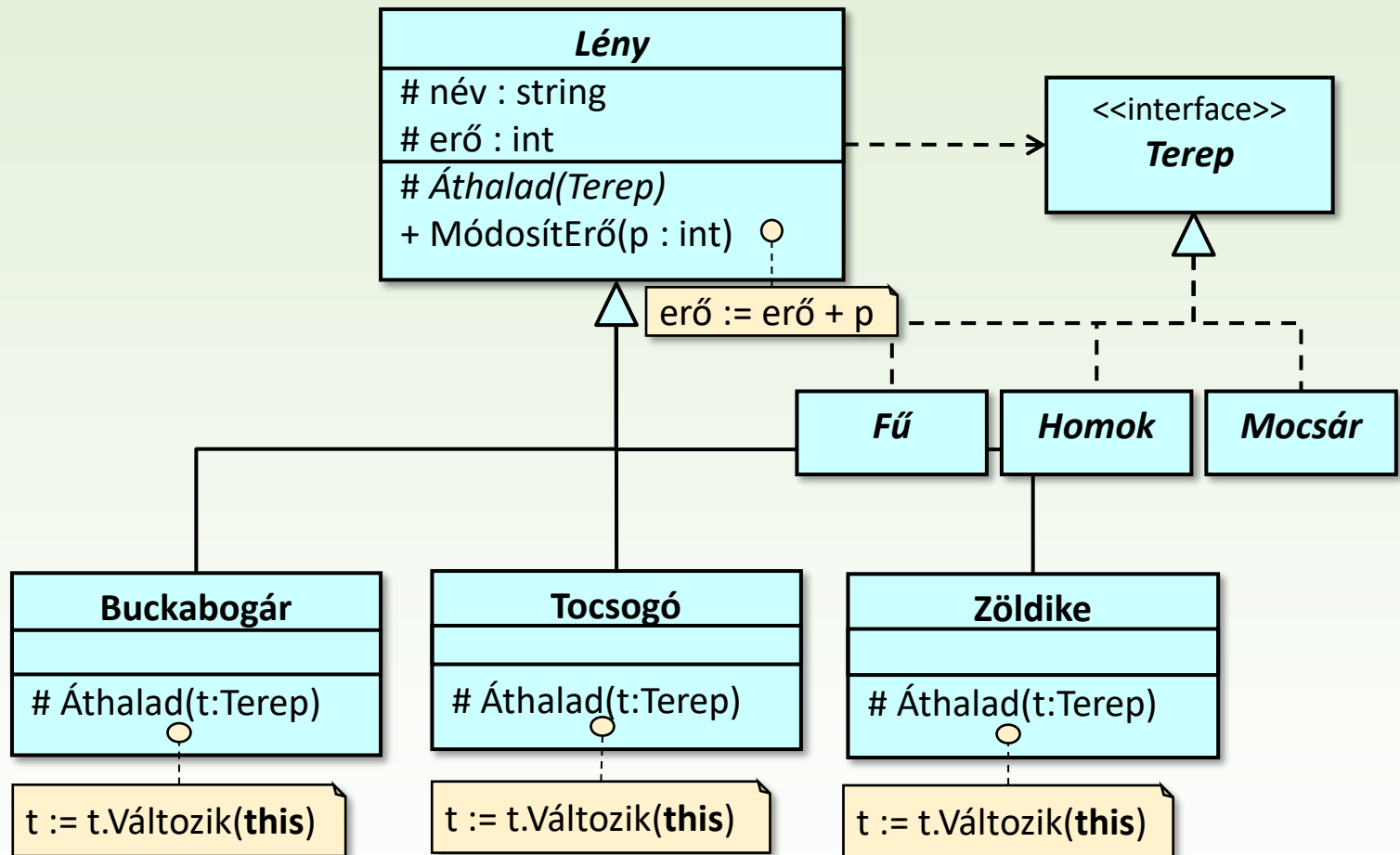


Állapot (state) tervezési minta

- Amikor egy objektum metódusai az objektum állapotától függően viselkednek, akkor – az ugyanolyan szerkezetű elágazások ismétlése helyett – az objektum állapotait külön egyke objektumokba szervezzük. Ezek közös interfésze specifikálja az állapotfüggő metódusokat, míg az eredeti objektum rendelkezik az állapotváltóztató metódussal.



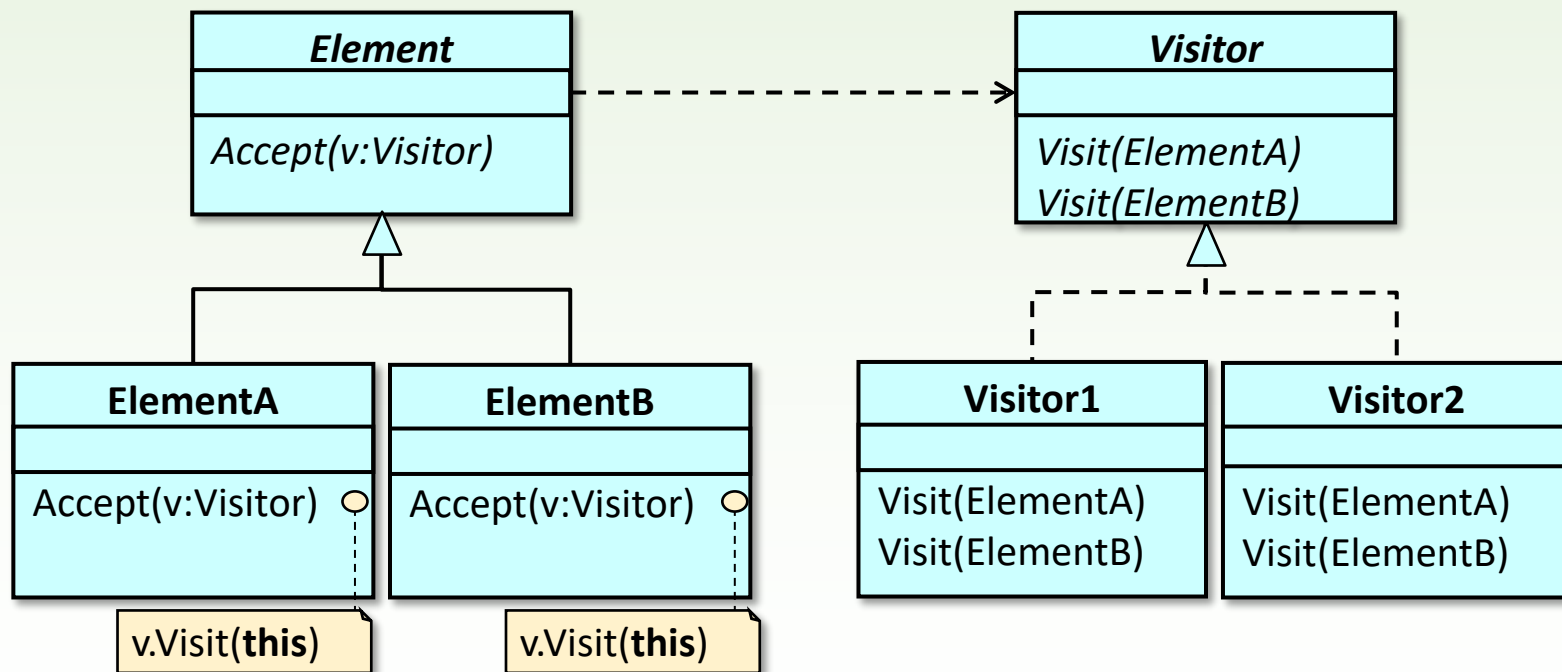
Lények osztályai



specializálni kellett az Áthalad() metódust, hogy kiolvasható legyen belőle a **this** típusa, hiszen a Változik() metódus csak konkrét lényekre hívható

Látogató (visitor) tervezési minta

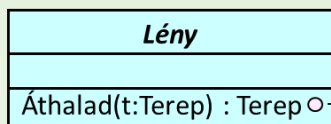
- Amikor egy metódus (`Accept()`) működése a saját osztályán kívül attól is függ, hogy a paramétere egy osztály melyik alosztályának objektumára hivatkozik, és nem akarjuk, hogy ez a függőség egy elágazás (vagy ugyanolyan szerkezetű elágazások) formájában jelenjen meg a kódban.



altípusos és parametrikus
polimorfizmus

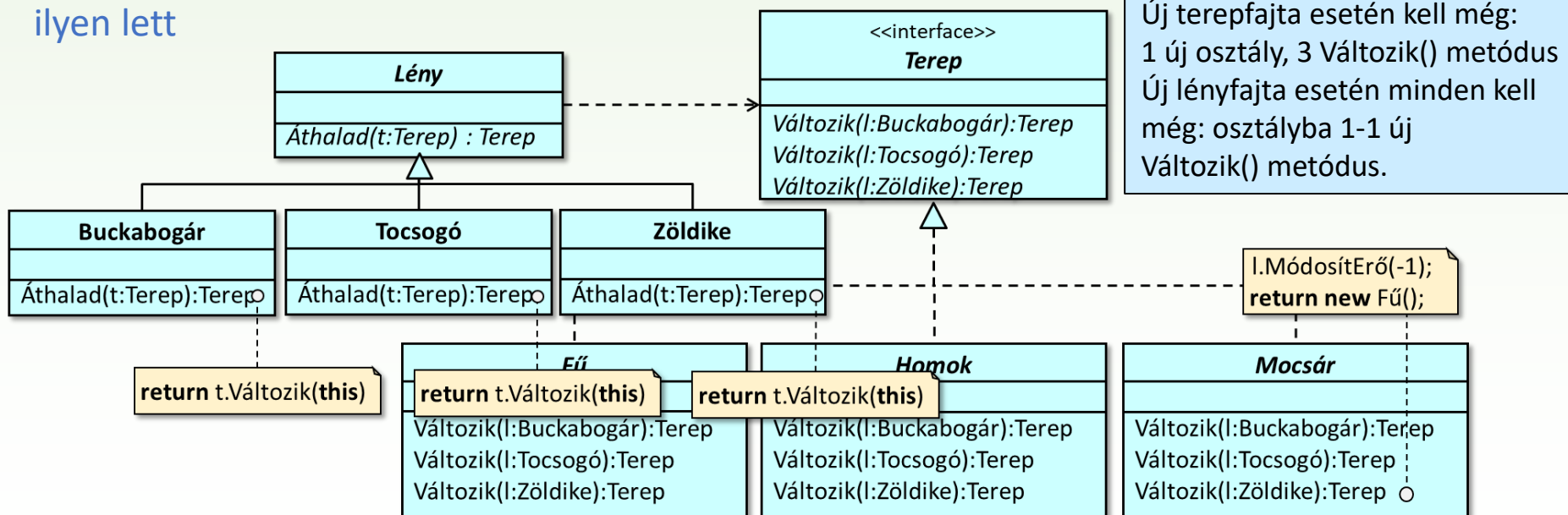
Látogató tervminta hatása

ilyen volt



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

ilyen lett



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.ModifyPower(-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.ModifyPower(3); return this; }
```

```
    IGround Change(Squelchy c) { c.ModifyPower(-5); return this; }
```

```
    IGround Change(Greenfinch c) { c.ModifyPower(-2); return this; }
```

```
class Marsh : IGround
```

```
{
```

```
    IGround Change(DuneBeetle c) { c.ModifyPower(-4); return Grass.Instance(); }
```

```
    IGround Change(Squelchy c) { c.ModifyPower(6); return this; }
```

```
    IGround Change(Greenfinch c) { c.ModifyPower(1); return this; }
```

```
    private Grass() { }
```

```
    private static Marsh instance = null;
```

```
    public static Marsh Instance()
```

```
    {
```

```
        instance ??= new Marsh();
```

```
        return instance;
```

```
    }
```

```
}
```

```
interface IGround
```

```
{
```

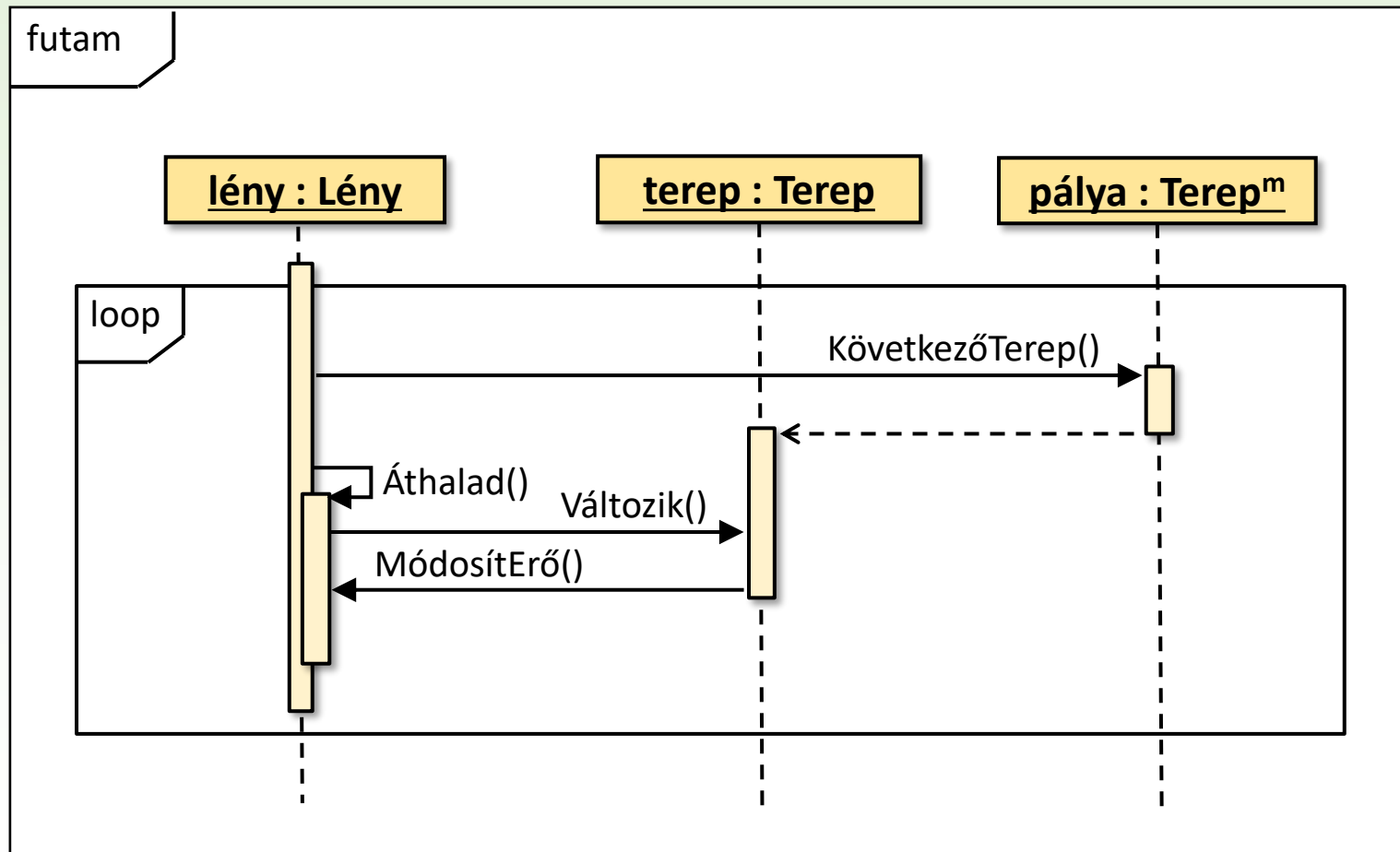
```
    IGround Change(DuneBeetle c);
```

```
    IGround Change(Greenfinch c);
```

```
    IGround Change(Squelchy c);
```

```
}
```

Egy lény futama



Egy lény futama

$pálya' \sim$ a pálya állapota a lény futama előtt
 $lény_0 \sim$ a lény állapota a lény futama előtt
 $lény_{j-1} \sim$ a lény a j -dik terepen való áthaladás előtt
 $lény_j \sim$ 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.

$A = (pálya: Terep^m, lény: Lény)$

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

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

Összefűzés: $pálya = \bigoplus_{j=1..m} \langle pálya[j] \rangle$

Sorozatos átalakítás: $lény = \bigoplus_{j=1..m} lény_j$
 ahol $\bigoplus : Lény \times Lény \rightarrow Lény$
 és $új := régi \bigoplus új$

futam - dupla összegzés

$t:enor(E) \sim j = 1 .. m$ (tereppek felsorolása)
 $H, +, 0 \sim Lény \times Terep^m, (\bigoplus, \bigoplus), (lény_0, < >)$
 $f(e) \sim \begin{cases} \text{Áthalad}(lény, pálya[j]) & \text{ha } lény.\text{Él}() \\ (lény, pálya[j]) & \text{különben} \end{cases}$

- 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

$lény.Futam(pálya)$

$lény, pálya := Futam(lény, pálya)$

$j := 1$

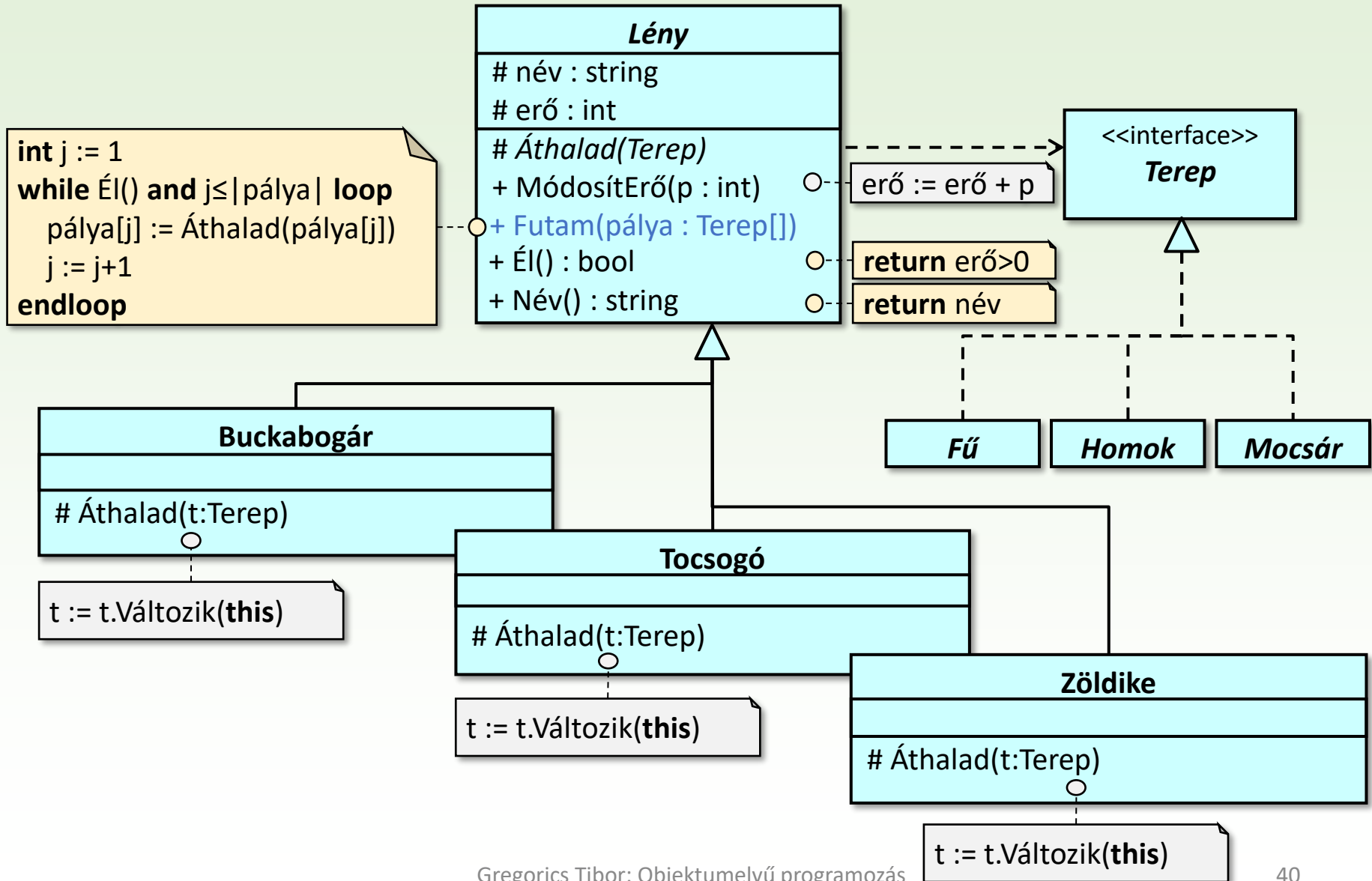
$lény.\text{Él}() \wedge j \leq m$

$lény, pálya[j] := \text{Áthalad}(lény, pálya[j])$

$j := j+1$

$lény.\text{Áthalad}(pálya[j])$

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

```

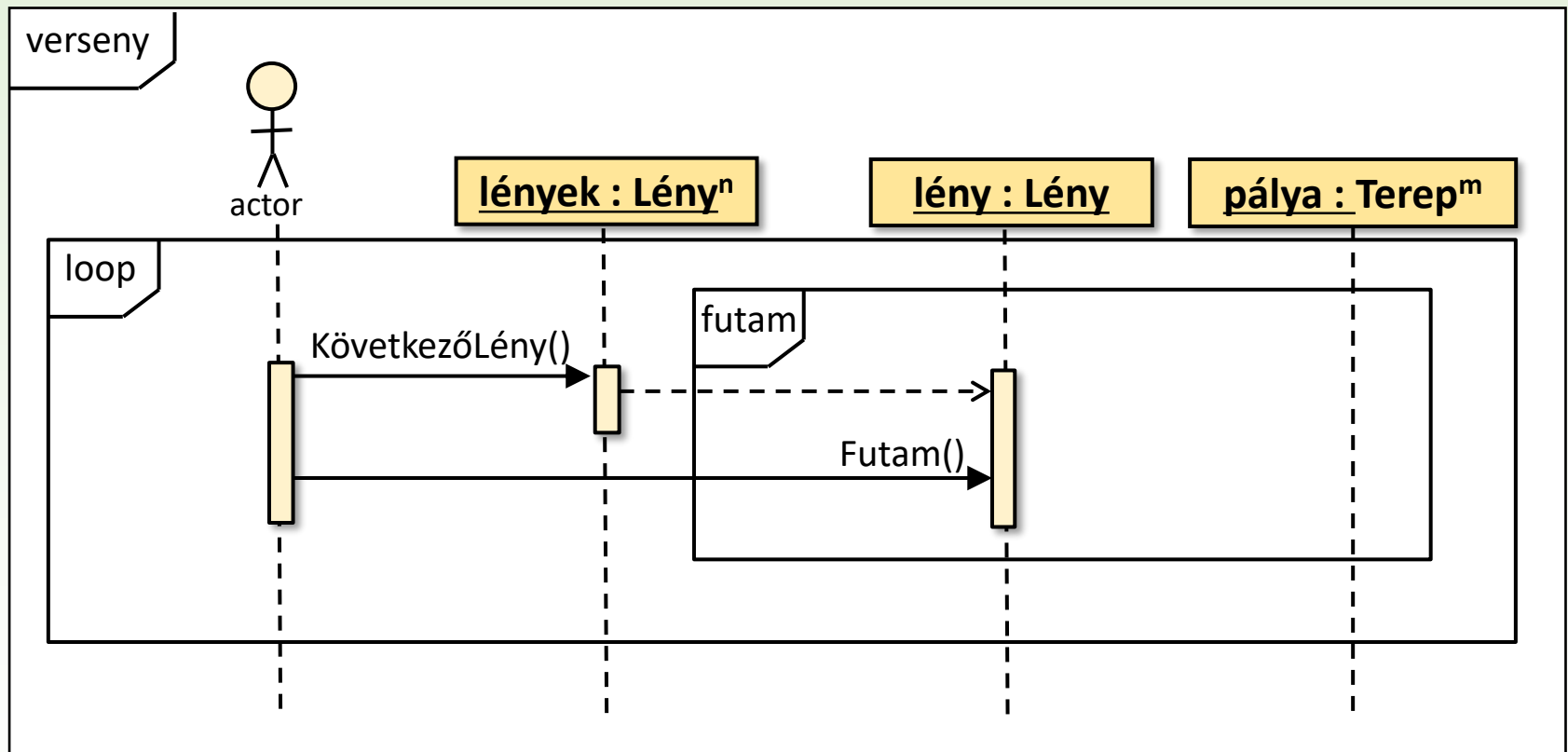
nem megy egyszerűbben:
 Traverse(ref court[j]);

```

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

```

Lények versenye



Lények versenye

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

$A = (\text{lények: Lény}^n, \text{pálya: Terep}^m, \text{túlélők: String}^*)$

$Ef = (\text{lények} = \text{lények}_0 \wedge \text{pálya} = \text{pálya}_0)$

$Uf = (\text{pálya} = \text{pálya}_n \wedge \forall i \in [1..n]: (\text{lények}[i], \text{pálya}_i) = \text{Futam}(\text{lények}_0[i], \text{pálya}_{i-1}) \wedge$
 $\wedge \text{túlélők} = \bigoplus_{i=1..n} \langle \text{lények}[i].\text{név}() \rangle)$
 $\text{lények}[i].\text{él}()$

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

Összefűzés: $\text{lények} = \bigoplus_{i=1..n} \langle \text{lények}[i] \rangle$

Sorozatos átalakítás: $\text{pálya} = \bigoplus_{i=1..n} \text{pálya}_i$
 ahol $\bigoplus : \text{Pálya} \times \text{Pálya} \rightarrow \text{Pálya}$
 és $\text{új} := \text{régi} \bigoplus \text{új}$

Kiválogatás

$t:\text{enor}(E) \sim i = 1 \dots n$ (lények felsorolása)

verseny - dupla összegzés

összefűzés és sorozatos átalakítás:

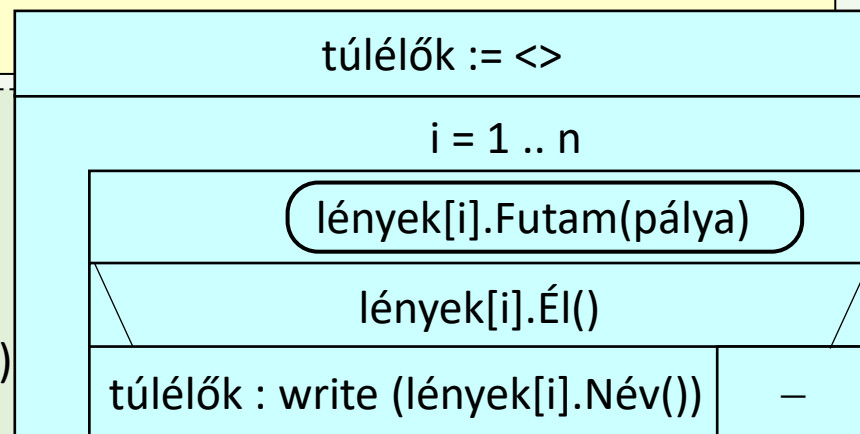
$f(e) \sim \text{Futam}(\text{lények}[i], \text{pálya})$

$H, +, 0 \sim \text{Lény}^n \times \text{Terep}^m, (\bigoplus, \bigoplus), (<>, \text{pálya}_0)$

kiválogatás - összegzés

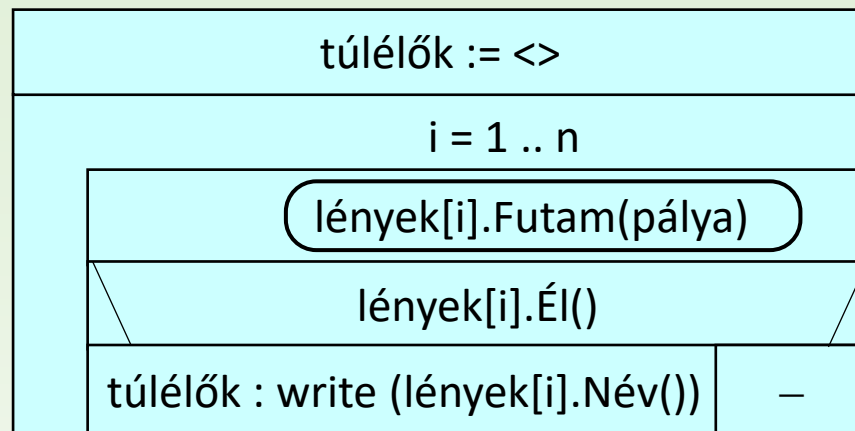
$f(e) \sim \langle \text{lények}[i].\text{Név}() \rangle$ ha $\text{lények}[i].\text{él}()$

$H, +, 0 \sim \text{String}^*, \bigoplus, <>$



- 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
```

```
foreach (Creature creature in creatures)
```

```
{
```

```
    creature.Race(ref courts);
```

```
    if (creature.Alive()) Console.WriteLine(creature.Name);
```

```
}
```

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

Lények létrehozása

```
// 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 G(name, p);
            case 'D': creature = new D(name, p);
            case 'S': creature = new S(name, p);
        }
        creatures.Add(creature);
    }
```

input.txt

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

```
// populating courts
```

```
reader.ReadLine(out line); int m = int.Parse(line);
List<IGround> courts = new ();
for (int j = 0; j < m; ++j)
```

```
{
    reader.ReadChar(out char c);
    switch (c)
    {
        case 'g': courts.Add(Grass.Instance()); break;
        case 's': courts.Add(Sand.Instance()); break;
        case 'm': courts.Add(Marsh.Instance()); break;
    }
}
```