Modellezés és példányosítás
1.rész
Modellezés

Gregorics Tibor

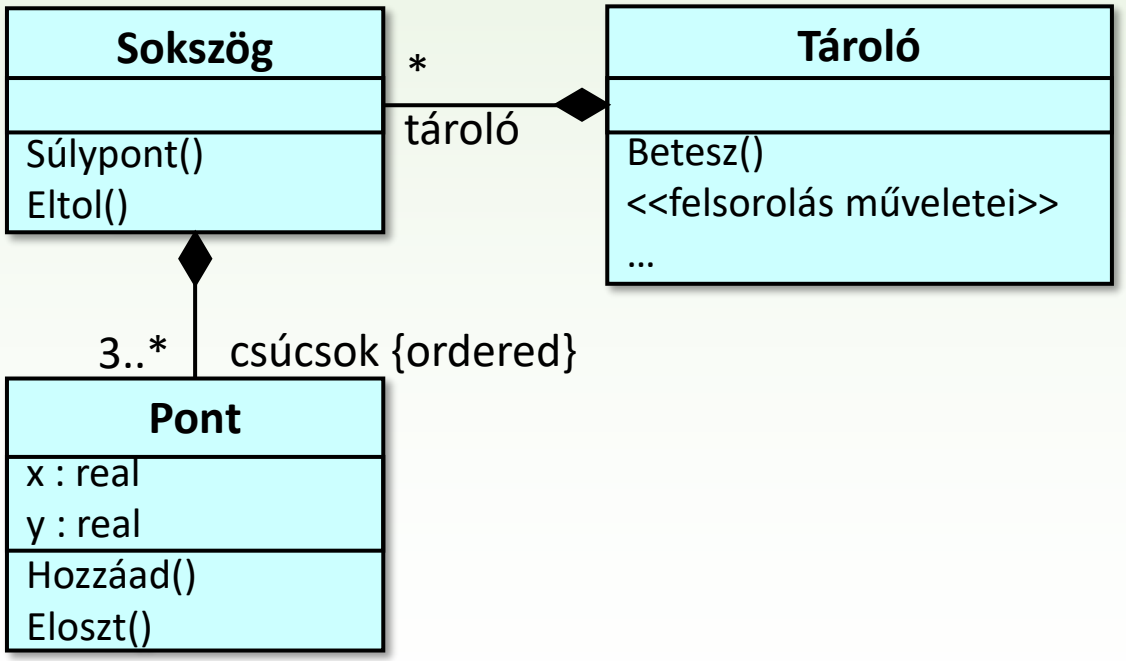
gt@inf.elte.hu

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

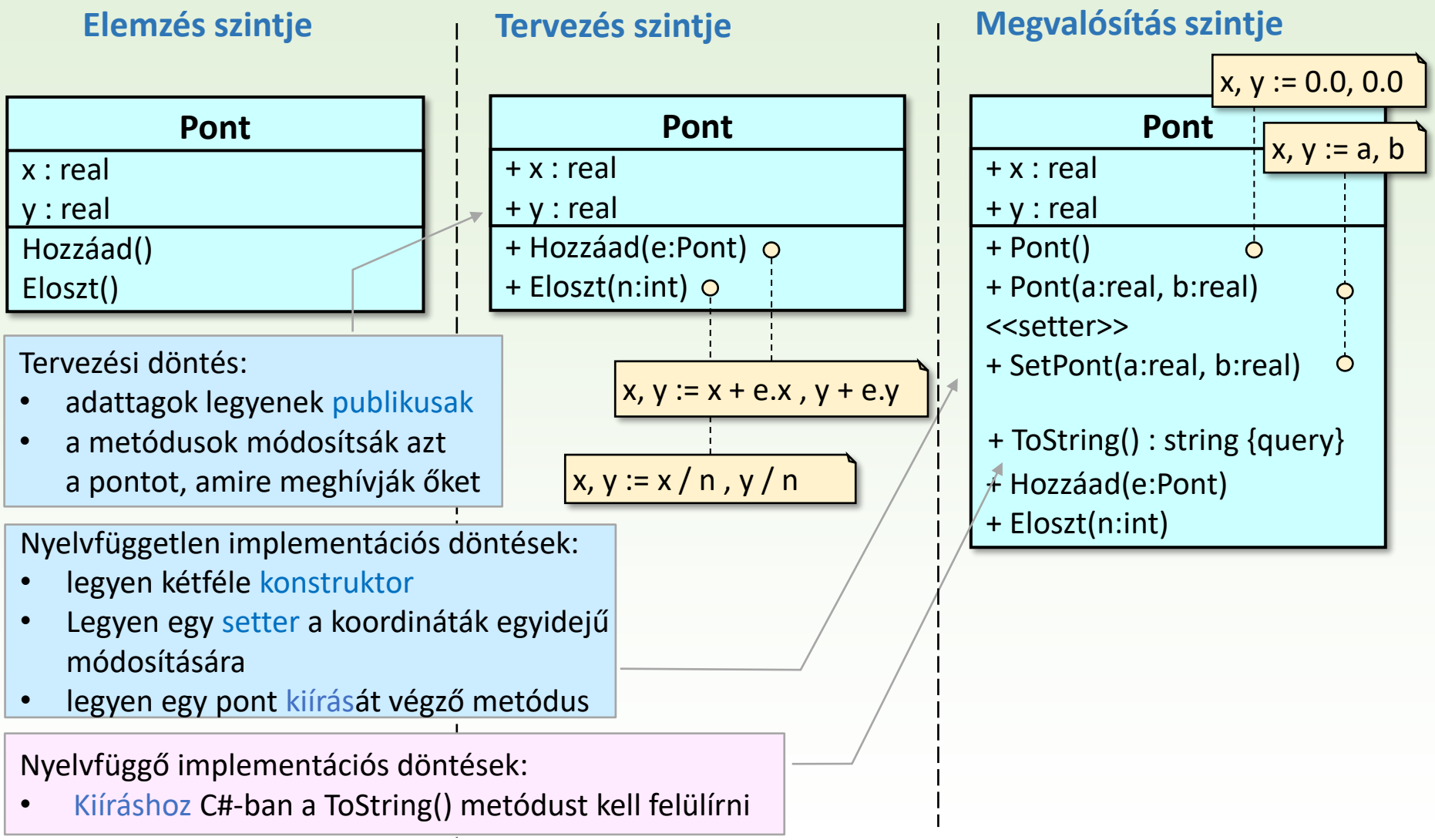
Feladat

Töltsünk fel egy tárolót különféle sokszögekkel, és mindegyiket toljuk el ugyanazon irányba és mértékkel, majd számoljuk ki az így nyert sokszögek súlypontjait. A csúcspontok és súlypontok koordinátái, sőt az eltolást leíró helyvektor végpontjának koordinátái is legyenek valós számok.

Elemzés eredménye:



Pont modellezése



Pont osztály C# kódja

```
class Point
{
   public double x, y;
   public Point(double x = 0.0, double y = 0.0)
   { this.x = x; this.y = y; }

   public void SetPoint(double x, double y)
   { this.x = x; this.y = y; }

   public override string ToString()
   { return string.Format($"({x:0.0#},{y:0.0#})"); }


   public void Add(Point e)
   { x += e.x; y += e.y; }

   public void DivideBy(int n)
   {
     if (n == 0) { throw new System.DivideByZeroException(); }
     x /= n; y /= n;
   }
}
```

default paraméter:

```
Point a = new ();
Point b = new (3.0);
Point c = new (-4.0, 8.0);
```

aktuális objektum

alapértelmezett működést írja felül

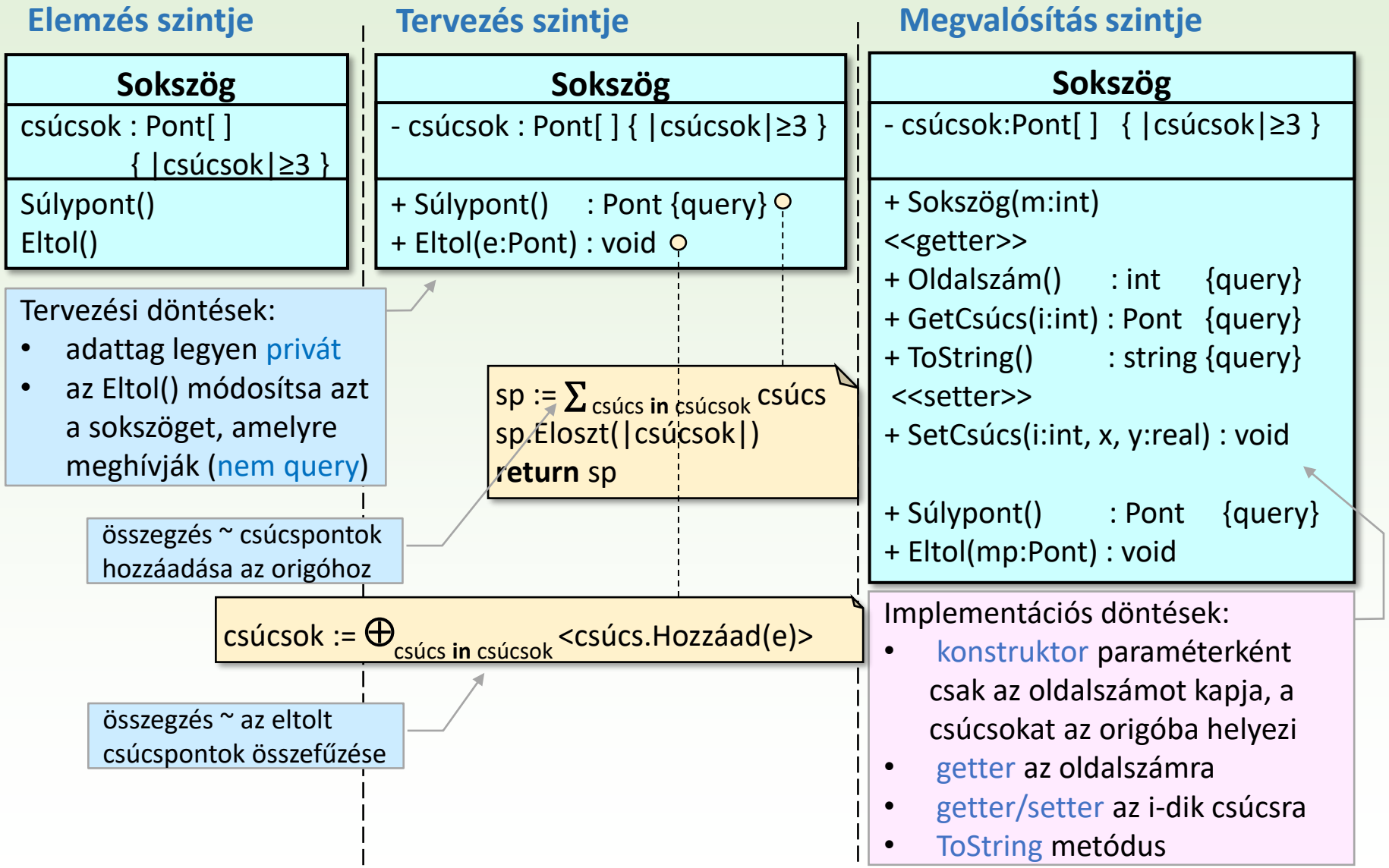
valós szám sztringgé formázása

Pont
+ x : real + y : real
+ Pont() + Pont(a:real, b:real) <<setter>> + SetPont(a:real, b:real) + ToString() : string {query} + Hozzáad(e:Pont) + Eloszt(n:int)

3

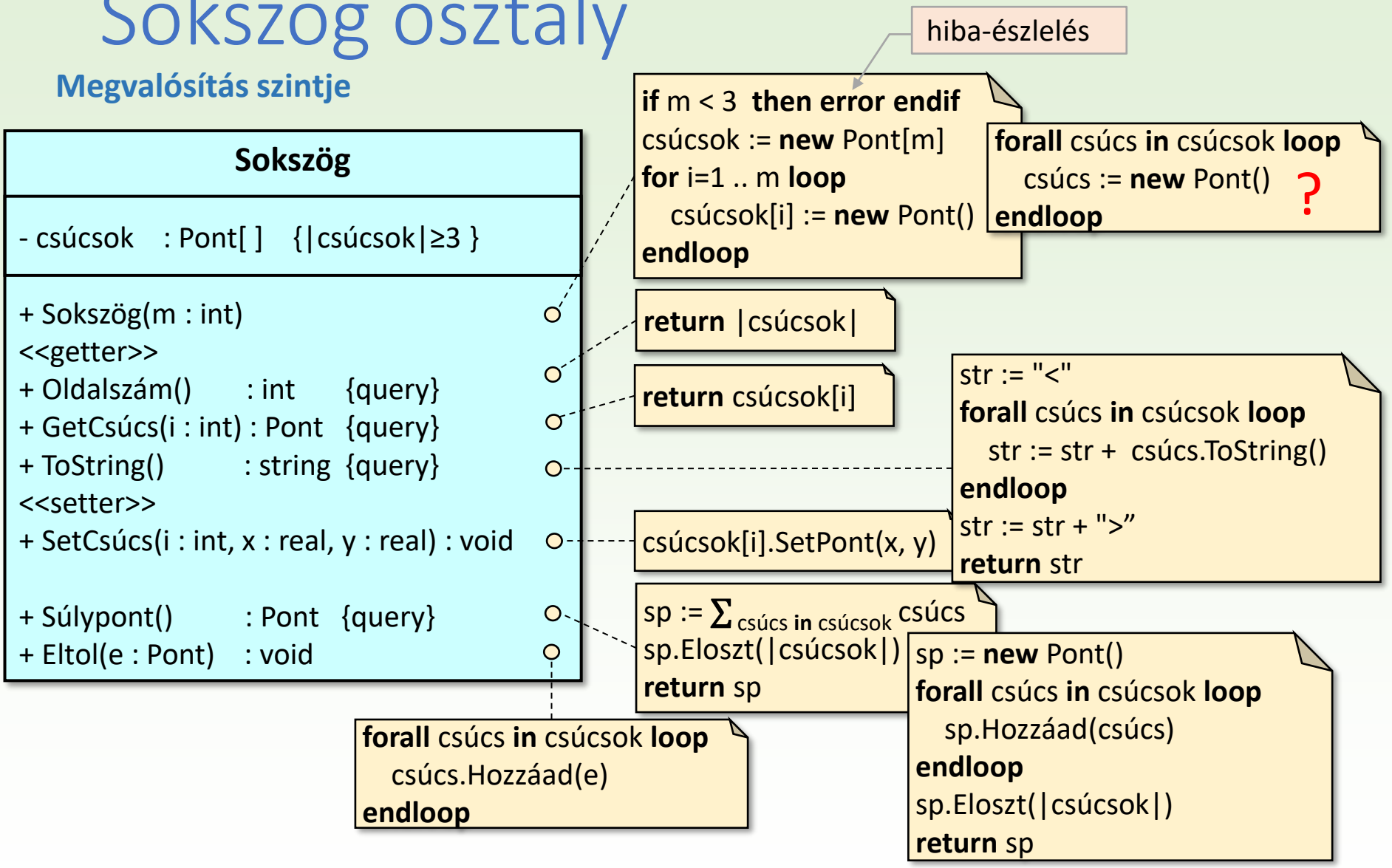
Az objektum orientált nyelvek további ismérve a nyílt rekurzió: az objektum mindig látja saját magát, eléri metódusaiban az adattagjait és metódusait.

Sokszög modellezése



Sokszög osztály

Megvalósítás szintje



Sokszög osztály C# kódja

```
class Polygon
{
  class FewVerticesException : Exception { }

  private readonly Point[] vertices;

  public Polygon(int m)
  {
    if (m < 3) throw new FewVerticesException();
    vertices = new Point[m];
    for (int i = 0; i < m; ++i) vertices[i] = new Point();
  }

  public int Sides  { … }
  public Point this[int i] { … }
  public override string ToString() { … }

  public Point Centroid() { … }
  public void Shift(Point e) { … }
}
```

a 'readonly' miatt a tömb helyfoglalása csak a konstruktorban történhet meg: a tömb memória címe (a tömb hivatkozása) nem változtatható meg, ugyanakkor a tömb elemei szabadon felülírhatók.

itt nem működik a foreach

Sokszög osztály getter-ei speciális nyelvi elemekkel

```
public int Sides
{
  get { return vertices.Length; }
}

public Point this[int i]
{
  get { return vertices[i]; }
  set { vertices[i] = value; }
}

public override string ToString()
{
  string str = "< ";
  foreach ( Point vertex in vertices ) str += vertex.ToString();
  str += " >";
  return str;
}
```

getter az oldalszámra:

```
Polygon p = new (3);
int n = p.Sides;
```

indexelő getter-setter
a GetCsúcs(i) és SetCsúcs(i) helyett:

```
Polygon p = new (3);
p[0] = new Point(23,-4);
Point q = p[1];
```

Sokszög osztály metódusai

```
public Point Centroid()
{
   Point centroid = new ();
   foreach (Point vertex in vertices)
   {
      centroid.Add(vertex);
   }
   centroid.DividedBy(Sides);
   return centroid;
}

public void Shift(Point e)
{
   foreach (Point vertex in vertices)
   {
      vertex.Add(e);
   }
}
```

Modellezés és példányosítás
2.rész
Példányosítás

Gregorics Tibor

gt@inf.elte.hu

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

Objektum különböző nézőpontokból

Modellezés

- Az objektum a megoldandó problémának egy részéért felelős, önálló egyedként kezelt elem (adatok és metódusok).
- Az objektum egységbe zárja a felelősségi köréhez tartozó adatokat és metódusokat, egy részüket elrejti, hogy azokat csak a metódusai használhassák.
- Az objektumnak van életciklusa: ez az objektum létrejöttével kezdődik, és megszűnésével fejeződik be.

Megvalósítás

- Az objektum egy memória-foglalás, ahol az objektumhoz tartozó adatokat tároljuk; ennek címét tartalmazza egy objektum-változó.
- Az objektum adattagjainak és metódusainak láthatósági köre szabályozható, de a metódusok elérik az objektum összes adattagját és a többi metódusát.
- A konstruktor foglal memóriát egy objektum számára (példányosít), amit a destruktor törlöl.

Sokszög példányosítása

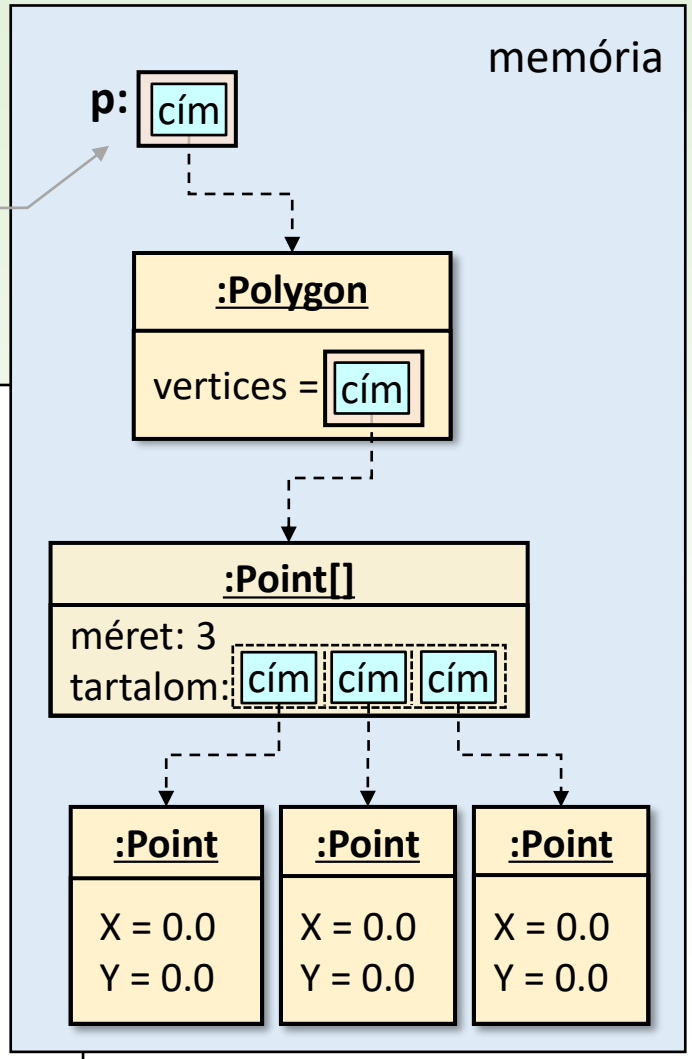
```
Polygon p = new (3);
```

A p referencia változó deklarálásakor csak egy akkora memória terület kerül lefoglalásra, amely azt a memória címet (hivatkozást) tárolja, ahol a változó típusának megfelelő adatot (esetünkben egy sokszög adattagjait) lehet eltárolni. Ezt foglalja le a memóriában a **new** parancs.

```
class Polygon
{
  private readonly Point[] vertices;

  public Polygon(int m)
  {
    if (m < 3) throw new FewVerticesException();
    vertices = new Point[m];
    for (int i = 0; i < m; ++i)
    {
      vertices[i] = new Point();
    }
  }
  ...
}
```

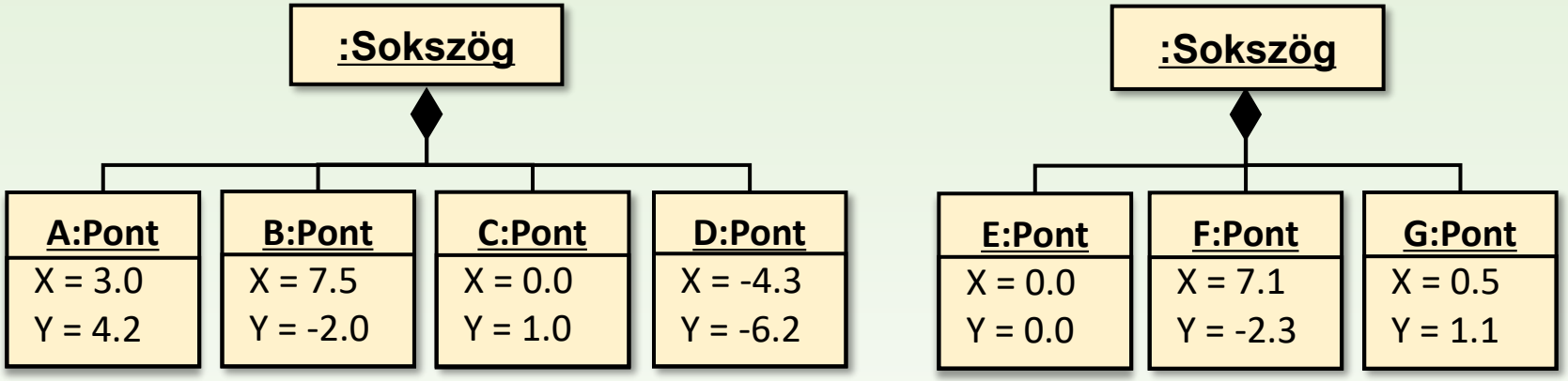
Ezeket a foglalásokat majd a garbage collector szabadítja fel.



A feladat felpopulálása

input.txt

```
4 3.0 -4.2 7.5 -2.0 0.0 1.0 -4.3 -6.2
3 0.0  0.0 7.1 -2.3 0.5 1.1
```




```
TextFileReader reader = new ("input.txt");
List<Polygon> container = new ();
while(reader.ReadInt(out int sides))
{
   container.Add(Polygon.Create(reader, sides));
}
```

sorozat típus

Kell egy metódus, ami a fájl egy sorának adataiból létrehoz egy új sokszöget. De ez nem lehet a sokszög-objektum metódusa, mert azt csak létező sokszög objektumra lehet meghívni. Nem lehet külső metódus sem, mert az nem éri el a sokszög rejtett adattagjait, nem tudná inicializálni azokat. Legyen osztály-szintű (static) metódus.

Osztályszintű gyártófüggvény

```
public static Polygon Create(TextFileReader reader, int sides)
{
  Polygon p = new (sides);

  for (int i = 0; i < sides; ++i)
  {
    reader.ReadDouble(out double x);
    reader.ReadDouble(out double y);
    p[i].SetPoint(x, y);
  }
  return p;
}
```

létrejön a sokszög az origóba pozícionált csúcsaival, vagy kivételt dob, ha sides<3

beállításra kerülnek egy csúcs koordinátái

Főprogram

A = (tároló : Sokszög*, eltolás : Pont, spk : Pont*)

Ef = (tároló = tároló$_0$ $\wedge$ eltolás = eltolás$_0$)

Uf = (eltolás = eltolás$_0$

$\wedge$ tároló = $\oplus_{\text{sokszög } \textbf{in} \text{ tároló}_0}$ <Eltol(sokszög, eltolás) >

$\wedge$ spk = $\oplus_{\text{sokszög } \textbf{in} \text{ tároló}}$ <Súlypont(sokszög) >)

itt még nem az objektumelvű modell metódusai szerepelnek:
Eltol : Sokszög×Pont→Sokszög
Súlypont : Sokszög→Pont

Két összegzés (összefűzés), ahol az első állítja elő a második felsorolását:

e in t	~	sokszög in tároló$_0$	sokszög in tároló
s	~	tároló	spk
H,+,0	~	Sokszög*, $\oplus$, <>	Pont*, $\oplus$, <>
f(e)	~	< Eltol(sokszög, eltolás) >	< Súlypont(sokszög) >

sokszög in tároló

sokszög.Eltol(eltolás)

a tároló egy sokszögének módosítása helyben

spk := < >

sokszög in tároló

spk := spk $\oplus$ < sokszög.Súlypont() >

spk := < >

sokszög in tároló

sokszög.Eltol(eltolás)

két ciklus összevonása

spk := spk $\oplus$ < sokszög.Súlypont() >

1. változat: procedurális szemlélet

```
static void Main()
{
   Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
   TextFileReader reader = new ("input.txt");
   reader.ReadDouble(out double x); reader.ReadDouble(out double y);
   Point move = new (x, y);

   List<Polygon> container = new ();
   while(reader.ReadInt(out int sides))
   {
      try
      {
         container.Add(Polygon.Create(reader, sides));
      }
      catch (Polygon.FewVerticesException)
      {
         Console.WriteLine("A polygon needs more than two vertices.");
      }
   }
```

```
using System.Globalization;
using System.Threading;
```

populálás

```
   foreach (Polygon polygon in container)
   {
      polygon.Shift(move);
      Console.WriteLine(polygon.Centroid());
   }
}
```

számolás

az eltolt polinomok súlypontjait nem tároljuk el egy sorozatban, hanem azonnal kiírjuk

2. változat: objektumelvű szemlélet

```
static void Main()
{
  Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
  Application app = new ()
  app.Run();
  return 0;
}
```

```
class Application
{
  private List<Polygon> container = new ();
  private Point move = new ();

  public Application(string fname) { ... /* Populating */ }
  public void Run()    { ... /* Computing */ }
}
```

```
public Application(string
{
  TextFileReader reader = new (fname);
  reader.ReadDouble(out double x); reader.ReadDouble(out double y);
  move.SetPoint(x,y);

  while(reader.ReadInt(out int sides))
  {
    ...
    container.Add(Polygon.Create(reader, si
    ...
  }
}
```

```
public void Run()
{
  foreach (Polygon polygon in container)
  {
    polygon.Shift(move);
    Console.WriteLine(polygon.Centroid());
  }
}
```

3. változat: menüvezérelt program

```
static void Main()
{
  Thread.CurrentThread.CurrentCulture = new CultureInfo("en-US");
  Menu menu = new ();
  menu.Run();
  return 0;
}
```

```
class Menu
{
  private Polygon polygon;
  public Menu(){polygon = null;
  public void Run() {…}

  private void MenuWrite() {…}
  private void Case1() {…}
  private void Case2() {…}
  private void Case3() {…}
  private void Case4() {…}
}
```

```
public void Run()
{
  int v = 0;
  do
  {
    MenuWrite();
    v = int.Parse(Console.ReadLine());
    switch(v)
    {
      case 1: Case1(); break;
      case 2: Case2(); break;
      case 3: Case3(); break;
      case 4: Case4(); break;
    }
  }
  while(v != 0)
}
```

```
private void MenuWrite()
{
 Console.WriteLine("0 – exit\n");
 Console.WriteLine("1 – create\n");
 Console.WriteLine("2 – write\n");
 Console.WriteLine("3 – shift\n");
 Console.WriteLine("4 – centroid\n");
}
```

sokszöget létrehozó, kiíró, eltoló, súlypontját kiszámoló metódusok

Menüpontok

input1.txt

```
4 1 1 -1 1 -1 -1 -1 1
```

input2.txt

```
3 0 0 -1 0 0 -1
```

beolvassuk a fájlnevet

```
private void Case1()
{ // create
   Console.Write("File name: ");
   string filename = Console.ReadLine();
   TextFileReader reader = new (filename);
   if(reader.ReadInt(out int sides)) polygon = Polygon.Create(reader, sides);
}
```

```
private void Case2()
{ // write
   if(polygon==null) {Console.WriteLine("There is no polygon!"); return;}
   Console.WriteLine(polygon);
}
```

```
private void Case3()
{ // shift
   if(polygon==null) {Console.WriteLine("There is no polygon!"); return;}
   ... //  reading x, y
   Point move = new (x, y);
   polygon.Shift(move);
   Console.WriteLine(polygon);
}
```

```
private void Case4()
{  // centroid
   if(polygon==null) {Console.WriteLine("There is no polygon!"); return;}
   Console.WriteLine(polygon.Centroid());
}
```

Tesztkörnyezet

- A Point és a Polygon osztályok metódusaihoz tesztelő projektet készítünk a VS-ben.

1. project reference to Polygon
2. a Polygon névtérnek a Point és Polygon osztályai publikusak legyenek

```
using Polygon;
using TextFile;

namespace TestPolygon
{
  [TestClass]
  public class UnitTestPolygon
  {
    [TestMethod]
    public void TestCreate () { ... }

    [TestMethod]
    public void TestShift () { ... }

    [TestMethod]
    public void TestCentroid () { ... }
  }
}
```

```
using Polygon;
using System.Globalization;

namespace TestPolygon
{
  [TestClass]
  public class UnitTest
  {
    [TestMethod]
    public void TestAdd () { ... }

    [TestMethod]
    public void TestDividedBy () { ... }
  }
}
```

számpárok összeadásának tesztelése:
- asszociatív: (a+b)+c = a+(b+c)
- neutrális elem: 0+a = a, a+0=a
- inverz elem: a+(-a) = 0, (-a)+a = 0
- kommutatív: a+b = b+a

számpárok természetes számmal történő osztásának tesztelése
- kerekítés
- nullával való osztás

összegzés algoritmus minta tesztesetei:
- eltérő hosszúság
- eleje és vége