Objektumok példányosítása

Gregorics Tibor

gt@inf.elte.hu

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

Objektum különböző nézőpontokból

Modellezési szempontból

- Az objektum a megoldandó probléma adott részéért felelős, önálló egyedként azonosítható része.
- Az objektum elrejti a felelősségi köréhez tartozó adatokat: azokat kizárólag a metódusai révén kezeljük (olvassuk, módosítjuk).
- Az objektumnak van életciklusa : objektum létrejöttével kezdődik, és megszűnésével fejeződik be.

Programnyelvi megközelítés

- Az objektum egy olyan memória-foglalás, ahol az objektumhoz tartozó adatokat tároljuk.
- Az objektum adattagjainak és metódusainak) láthatósági köre szabályozható, de az objektum metódusai mindig elérik azokat.
- Egy objektumnak a konstruktora foglal memóriát (példányosítás), és a destruktora törli.

Feladat

Töltsünk fel egy tömböt 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 egész számok.

Objektumok:

- sokszögek
- síkbeli pontok (sokszögek csúcsai, súlypontjai, eltolás helyvektorának végpontja)
- tömb a sokszögek tárolásához

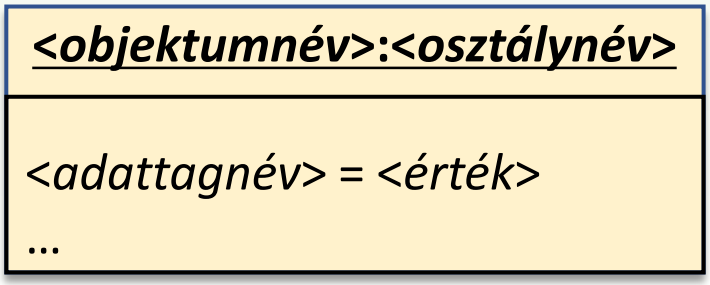
Single responsibility
O
L
I
D

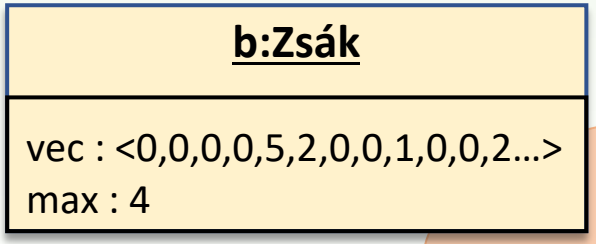
Objektumok felelősségi köre:

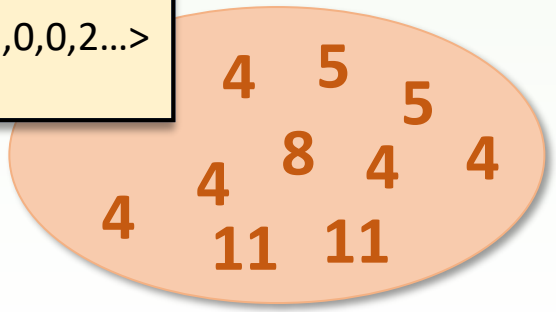
- sokszög: eltolása, súlypontjának kiszámítása, oldalszámának megadása, adott csúcspontjának lekérdezése és módosítása
- síkbeli pont: eltolása, koordinátáinak lekérdezése és módosítása
- tömb: adott indexű elem elérése, hosszának lekérdezése

Objektum UML jelölése

- Egy objektumot meghatároz
  - az osztálya, amely az ugyanolyan adattagokkal és metódusokkal rendelkező objektumokat írja le.
  - a neve (amit nem kötelező megadni),
  - az állapota (amit az adattagjainak értékei jelölnek ki).

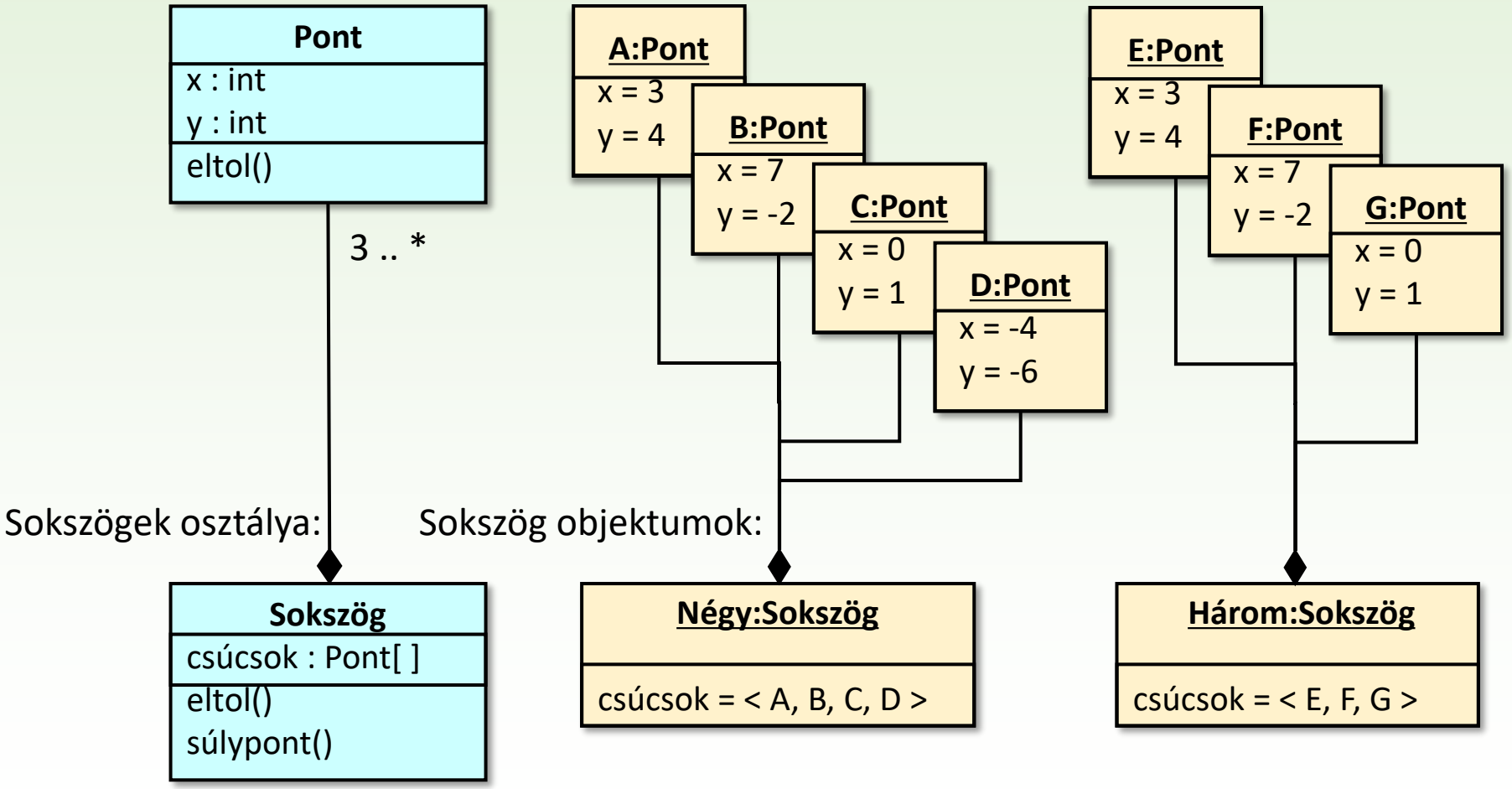






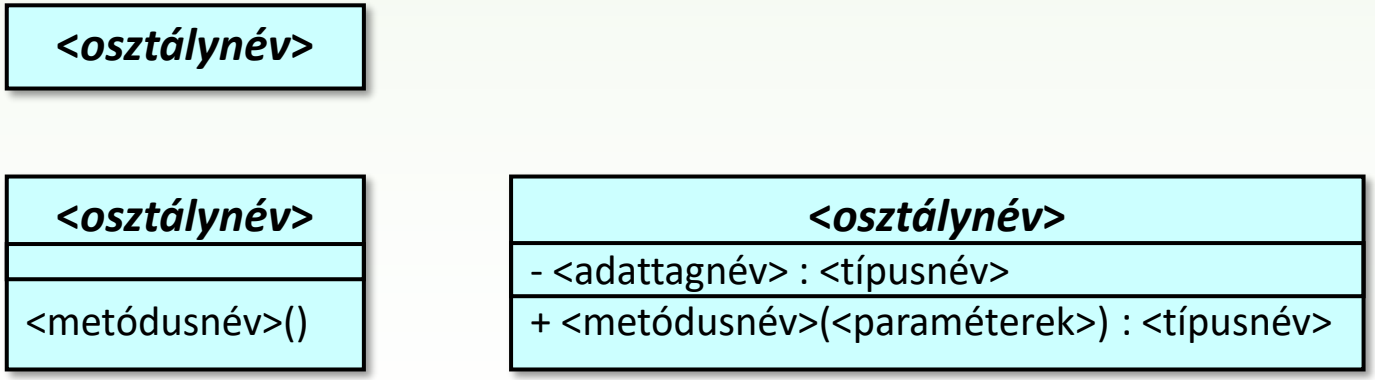
Objektumok egy populációja

Pontok osztálya: Pont objektumok:



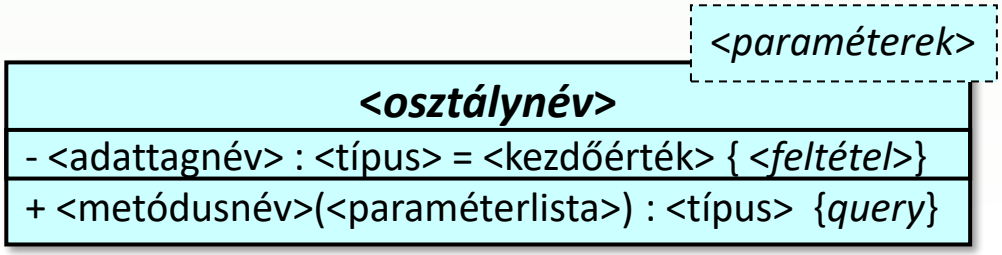
Osztály-leírás részletezettsége

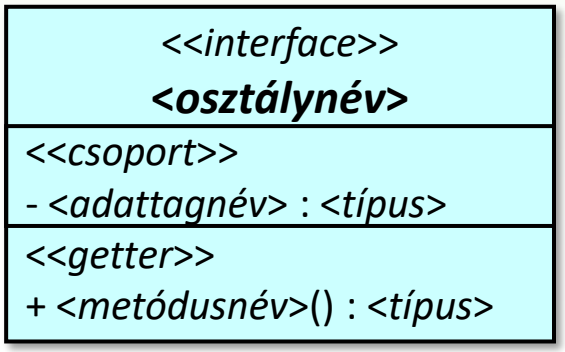
- Az osztály az objektum típusa: az ugyanolyan adattagokkal és metódusokkal rendelkező objektumokat jellemzi.
- Az osztály-leírás a modellezés során fokozatosan alakul ki, ezért a modellezés adott szintjén bizonyos részletei még hiányozhatnak.
  - Hiányozhatnak belőle az attribútumok és/vagy a metódusok.
  - Hiányozhat az adattagok típusa, a metódusok paraméterezése és a visszatérési típusa.
  - Hiányozhat a láthatósági jelölések feltüntetése.



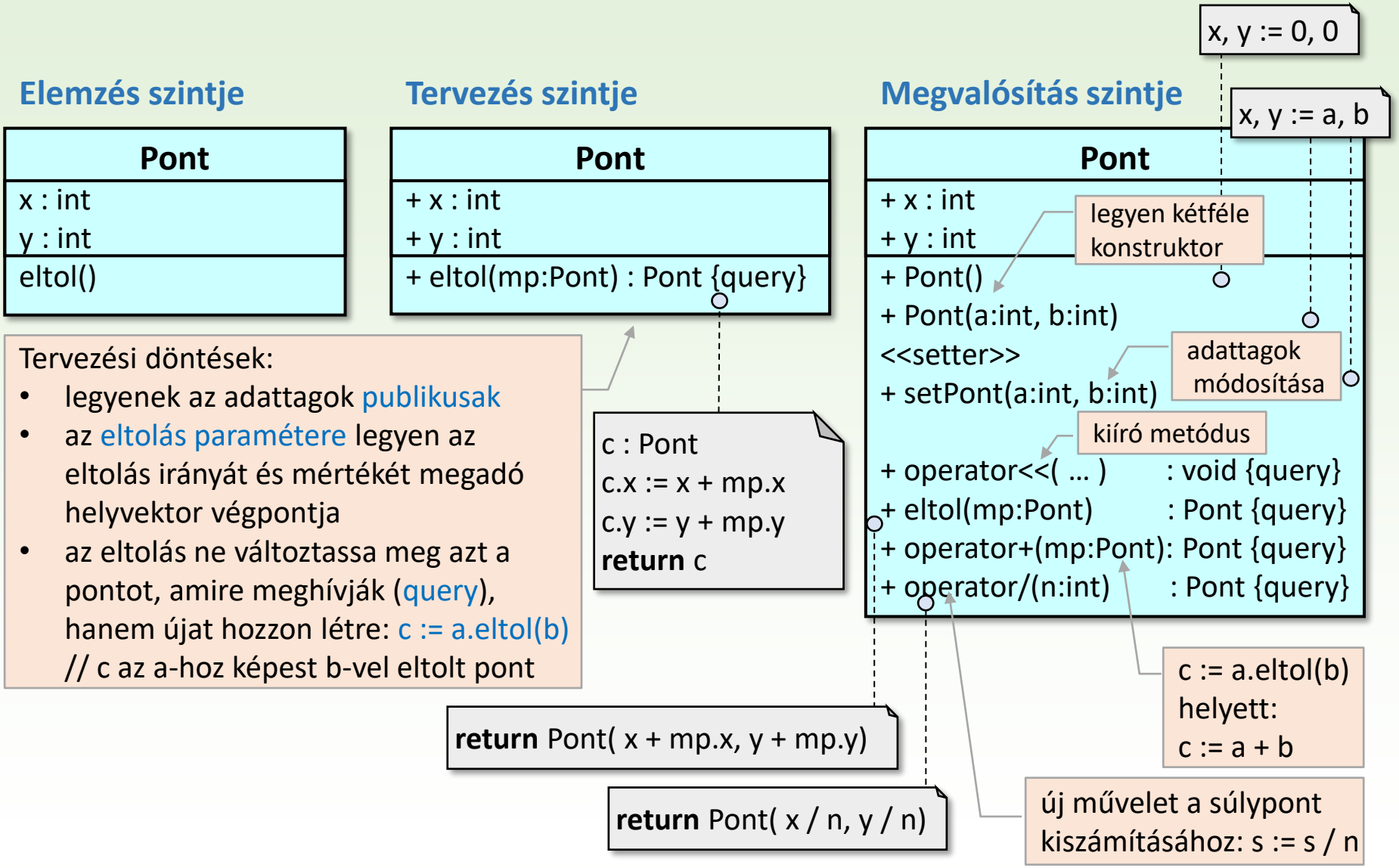
Osztályleírás kiegészítései

- Az osztályok a példányaik jellegzetes viselkedése alapján különféle kategóriákba (sztereotípiákba) sorolhatók, amelyet <<…>> jelzés között írhatunk le (pl. <<interface>> , <<enumeration>>).
- Egy osztály leírása később megadandó paraméterek megadásával (osztály-sablon) általánosítható (pl. típust helyettesítő paraméterek).
- Az adattagokhoz kezdőértéket rendelhetünk (ezt a konstruktor állítja majd be), és a {…} jelzésben megszorításokat (típus invariáns) tehetünk.
- A metódusok működését egyenként is jellemezhetjük. Pl. a {query} jelzi, hogy egy metódus az adattagokat nem módosítja. A <<getter>> az adattagok értékét lekérdező műveletek csoportját, a <<setter>> a felülíró műveletek csoportját vezeti be.





Pont fogalmának modellezése



Pont osztály C++ kódja

Pont
+ x : int + y : int
+ Pont() + Pont(a:int, b:int) <<setter>> + setPont(a:int, b:int) + operator<<(…) : void {query} + eltol(mp:Pont) : Pont {query} + operator+(mp:Pont): Pont {query} + operator/(n:int) : Pont {query}

```
class Point
{
public:
        int _x, _y;
public:
        Point(int x = 0, int y = 0): _x(x), _y(y) { }

        void setPoint(int x, int y) { _x = x; _y = y; }

        Point move(const Point &mp) const
                { return Point(_x + mp._x, _y + mp._y ); }

        Point operator+(const Point &mp) const
                { return Point(_x + mp._x, _y + mp._y); }

        Point operator/(int n) const
                { return Point(_x / n, _y / n ); }
};
```

paraméterváltozók default értéke:

```
Point a;
Point b(3);
Point c(-4, 8);
```

konstans referencia

query

operátor felüldefiniálás:
p + q kifejezés meghívja a p objektumra az operator+()-t, a q paraméterre az mp hivatkozik, és egy pontot ad vissza.

A kiíró operátor nem a Point osztály metódusa, hisz cout<<… formában kell a cout-ra meghívni, nem p<<… alakkal. Ezért külső operátorként vezetjük be.

```
std::ostream& operator<<(std::ostream &o, const Point &p)
{
        o << "(" << p._x << "," << p._y << ")";
        return o;
}
```

Objektum default pointere

```
class Point{
public:
        Point(int x=0, int y=0):_x(x),_y(y) {}

        void setPoint(int x, int y) { _x = x; _y = y; }
        void setPoint(int x, int y) { this->_x = x; this->_y = y; }
        ...

public:
        int _x, _y;
};
```

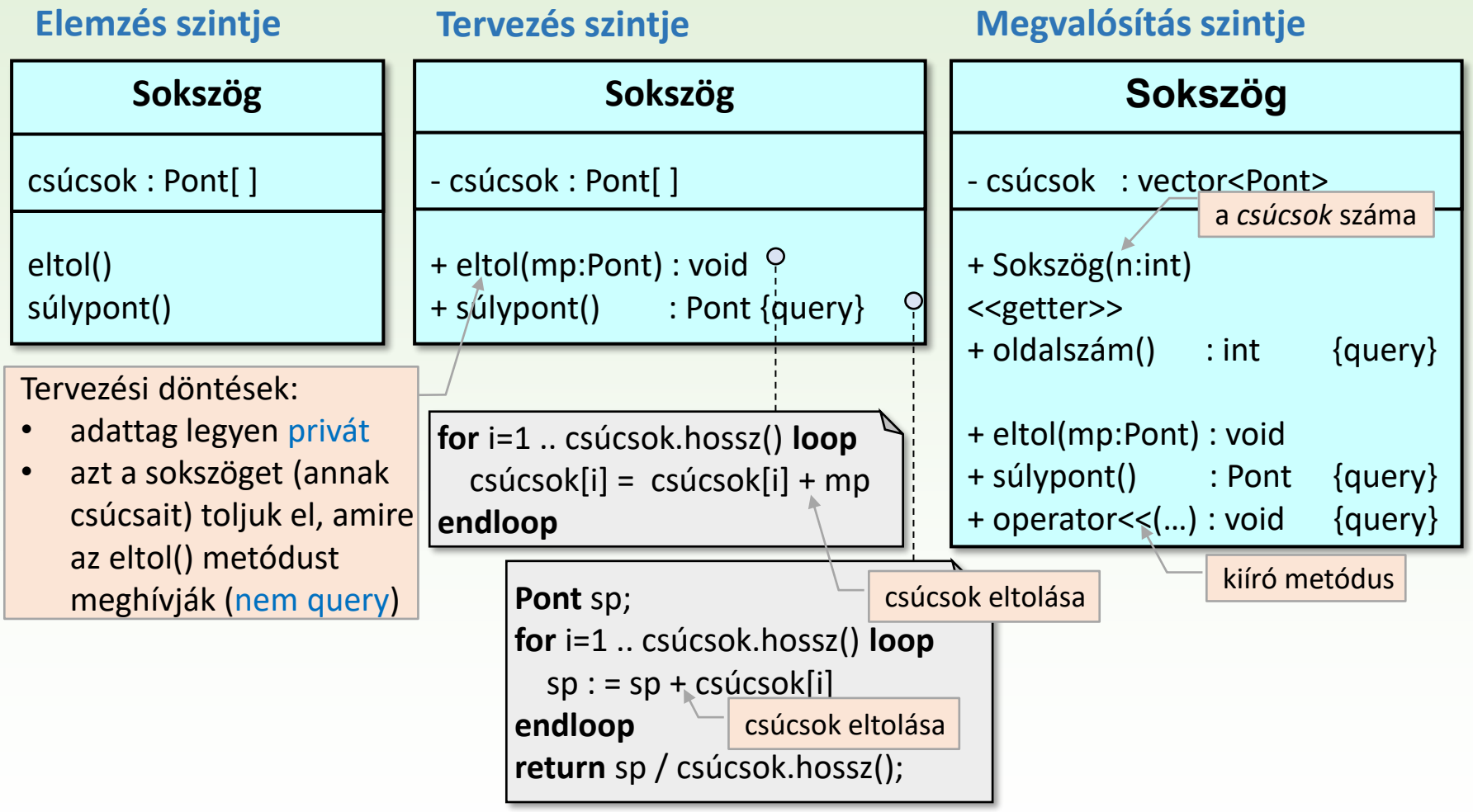
lehetnének az adattagok x és y

A **this** egy alapértelmezett módon létező pointerváltozó, amely azon objektumnak a memóriacímét tartalmazza (arra mutat), amelyik objektumra a metódust meghívták: Pl. a p.setPoint(3, -2) hívás esetén a this a p címe.

3

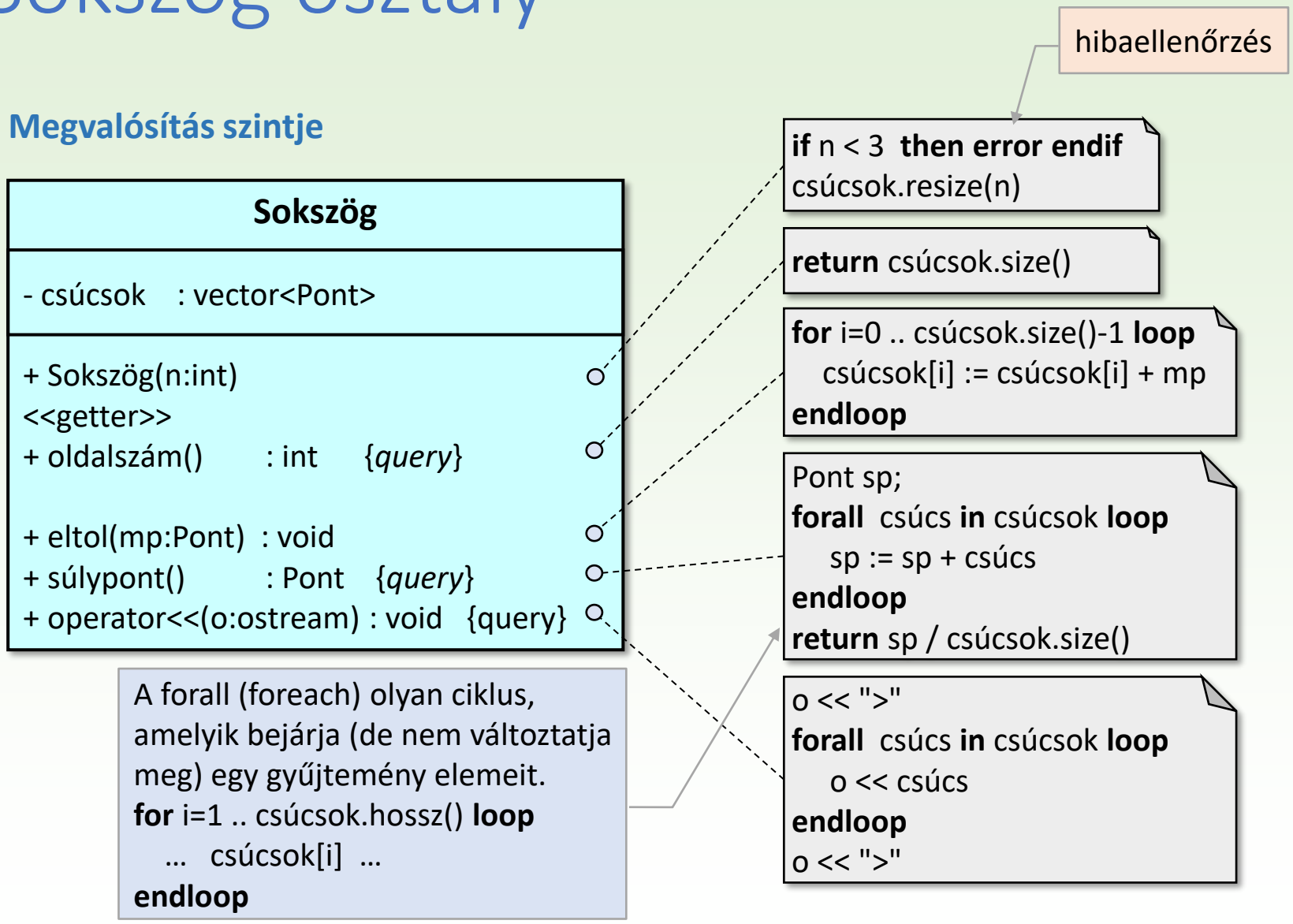
Az objektum orientált nyelvek további ismérve a nyílt rekurzió:
az objektum mindig látja saját magát, eléri saját műveleteit és adattagjait.

Sokszög fogalmának modellezése



Sokszög osztály

Megvalósítás szintje



Sokszög osztály C++ kódja

```
class Polygon
{
private:
      std::vector<Point> _vertices;
public:
      enum Errors{FEW_VERTICES};

      Polygon(int n) : _vertices(n) {
            if (n < 3) { throw FEW_VERTICES; }
      }

      int  sides() const { return _vertices.size(); }

      void  move(const Point &mp);

      Point center() const;

      friend std::ostream& operator<<(std::ostream &o, const Polygon &p);
};
```

hibaesetek (kivételek) definiálása

Sokszög
- csúcsok : vector<Pont>
+ Sokszög(n:int) <<getter>> + oldalszám() : int {*query*} + eltol(mp:Pont) : void + súlypont() : Pont {*query*} + operator<<(o:ostream) : void {query}

tesztelésnél:
CHECK_THROWS(Polygon())

A hibát csak jelezzük (kivétel dobása), de a hiba kezelése nem itt történik.

olyan külső operátor, amelyik hozzáfér a privát tagokhoz

Sokszög metódusainak C++ kódja

```
void Polygon::move(const Point &mp)
{
	for(unsigned int i=0; i<_vertices.size(); ++i) {
		_vertices[i] = _vertices[i] + mp;
	}
}

Point Polygon::center() const
{
	Point center;
	for(const Point& vertex : _vertices) { center = center + vertex; }
	return center / sides();
}

std::ostream& operator<<(std::ostream &o, const Polygon &p)
{
	o << "<";
	for( const Point& vertex : p._vertices ) { o << vertex; }
	o << ">";
	return o;
}
```

a **forall** (foreach) ciklus C++ kódja

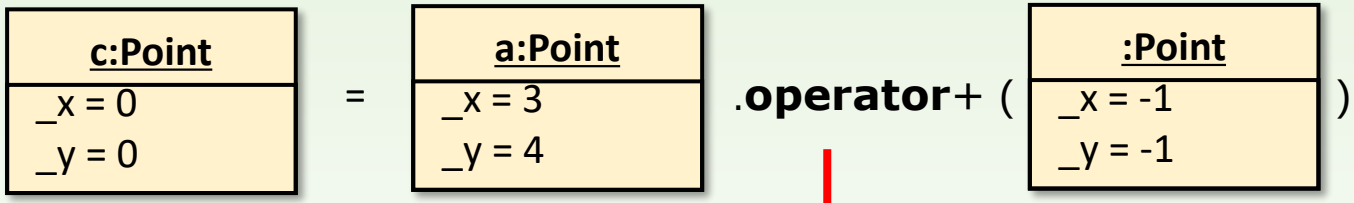
a friend kapcsolat miatt hivatkozhat a privát adattagra

Síkbeli pontok példányosítása

```
Point a(3,4);
Point c;
c = a + Point(-1,-1);
```

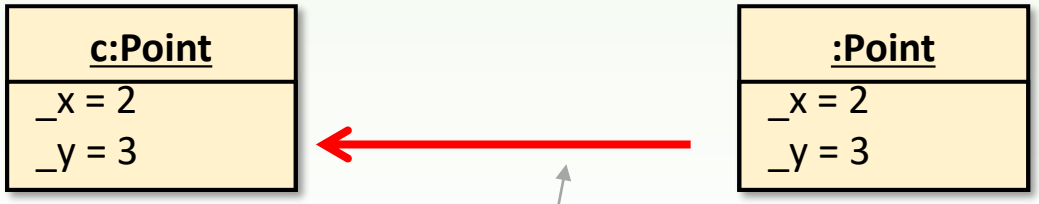
Erre a pontra hivatkozik a **operator**+() metódus mp paraméterváltozója.

operator+() hívása előtt:



operator+() hívása közben:

operator+() hívása után:



Ez az **operator**+()-ban létrehozott pont.

Minden objektum rendelkezik értékadás operátorral, amely – ha nem változtatjuk meg – lemásolja az adattagok értékét.

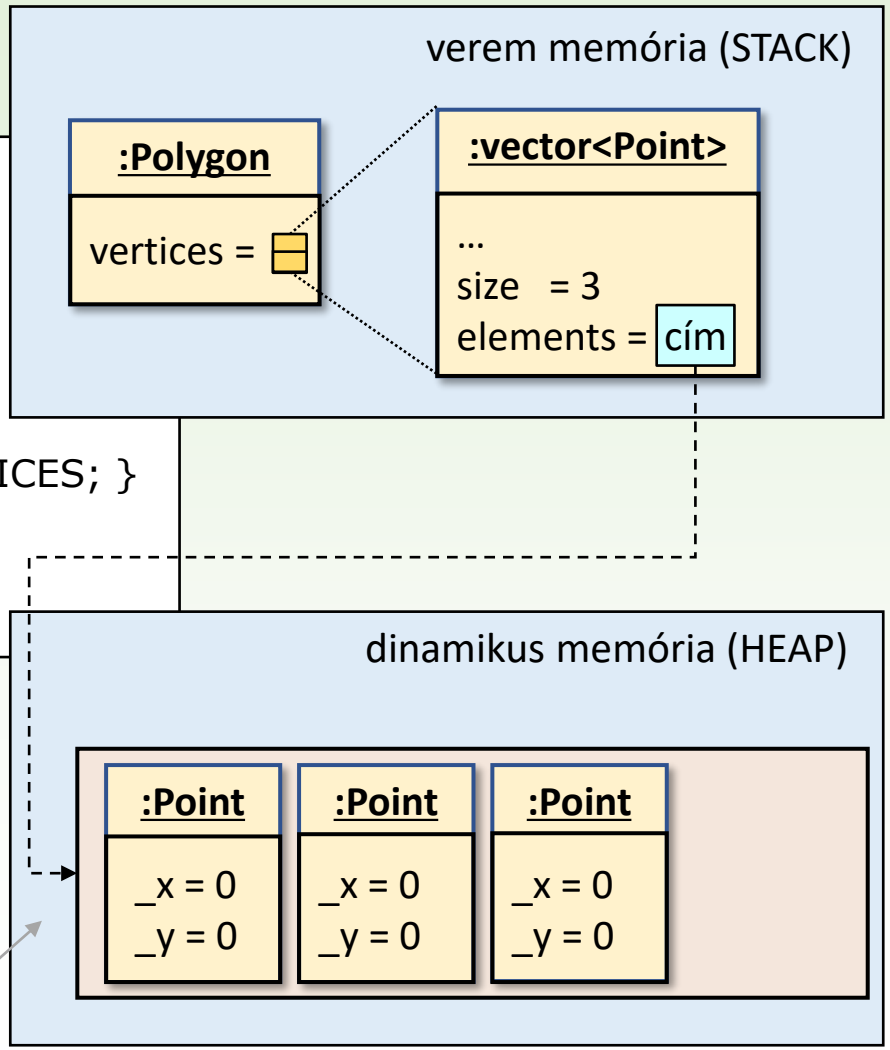
```
Point operator+(const Point &mp) const {
        return Point(_x + mp._x, _y + mp._y );
}
```

Egy sokszög példányosítása

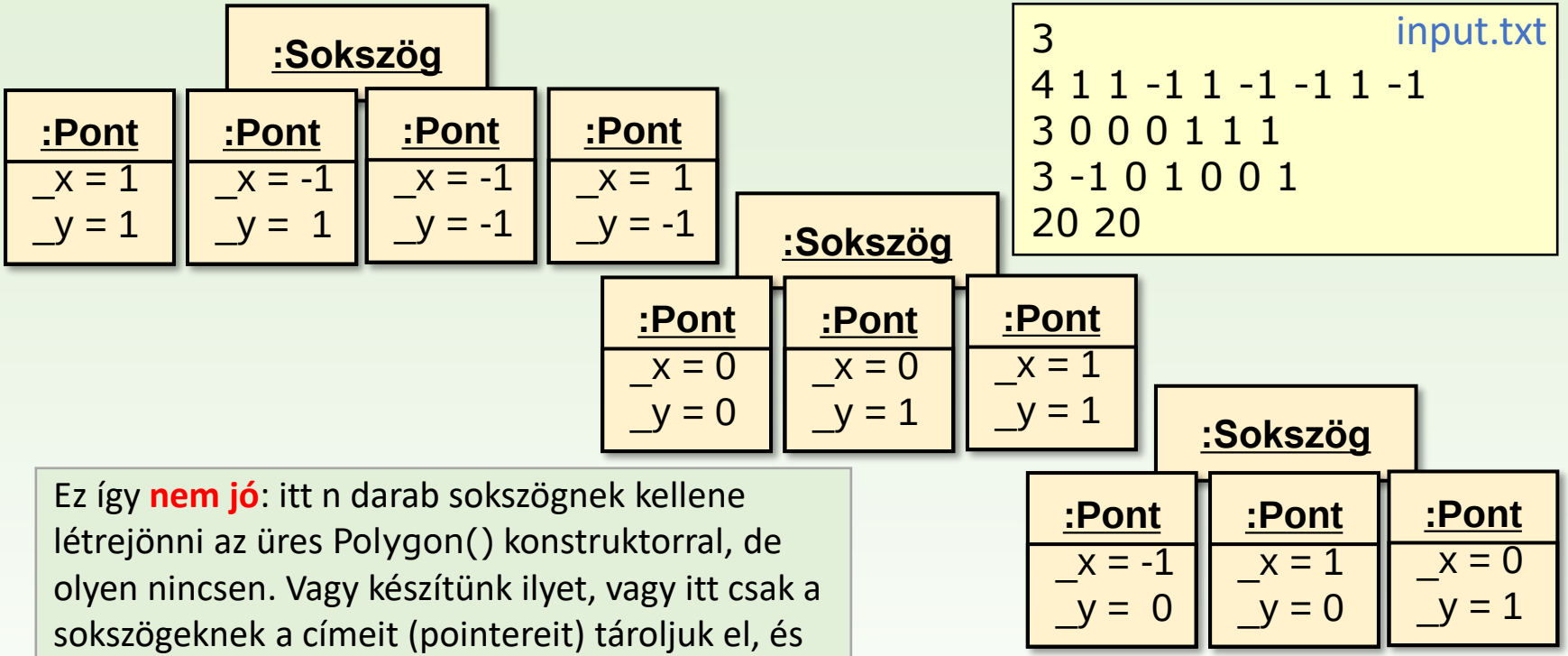
```
class Polygon {
private:
      vector<Point> _vertices;
public:
      Polygon(int n) : _vertices(n)
      {
            if (n < 3) throw { FEW_VERTICES; }
      }
      …
};
```

```
Polygon p(3);
```

A Polygon konstruktora meghívja a vector<Point>(n) konstruktort, amely a Point üres konstruktorát hívja, és ezért minden pont az origóba kerül. Ezeket módosítani kellene tudnunk.



A feladat felpopulálása



input.txt

```
3
4 1 1 -1 1 -1 -1 1 -1
3 0 0 0 1 1 1
3 -1 0 1 0 0 1
20 20
```

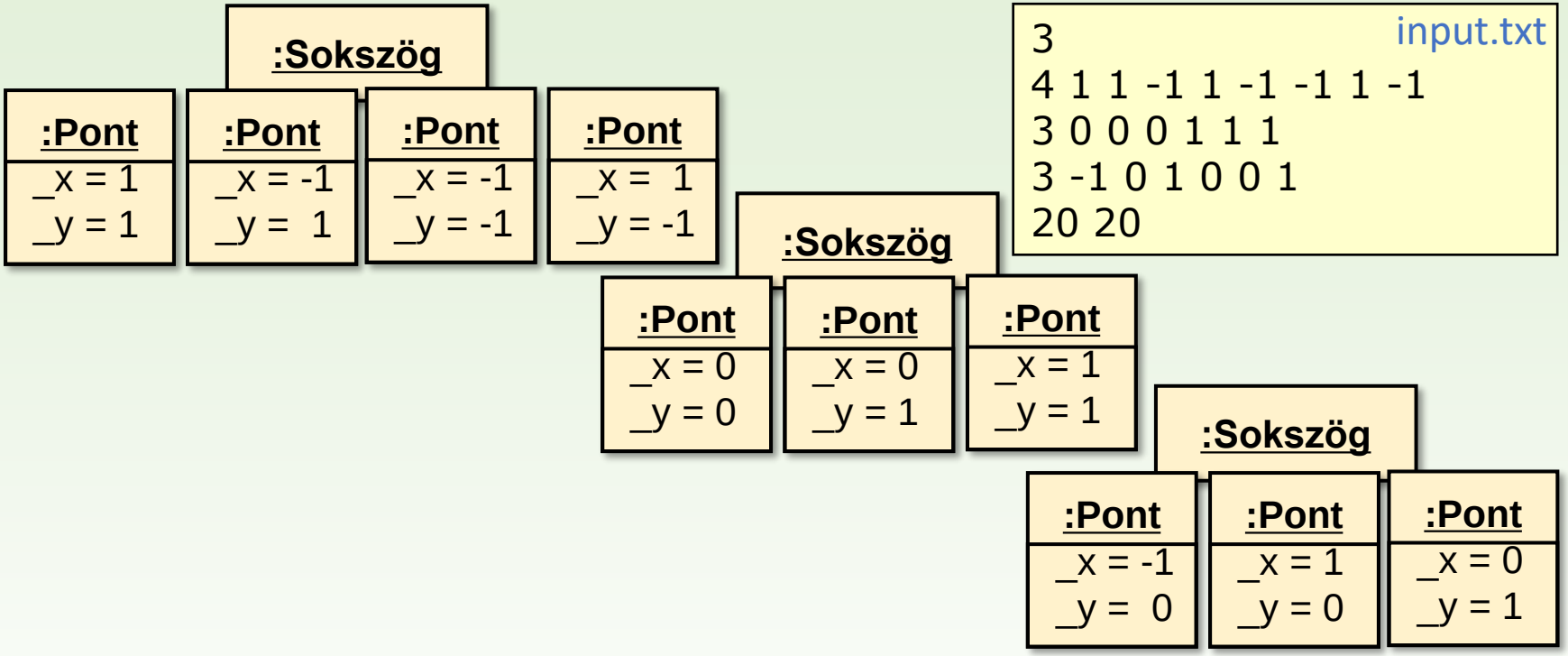
Ez így **nem jó**: itt n darab sokszögnek kellene létrejönni az üres Polygon() konstruktorral, de olyen nincsen. Vagy készítünk ilyet, vagy itt csak a sokszögeknek a címeit (pointereit) tároljuk el, és később hozzuk létre a sokszögeket.

```
ifstream inp("input.txt");
…
int n; inp >> n;
vector<Polygon> t(n);

for( int i=0; i<n; ++i ) { t[i].set(inp); }
```

Ilyen metódusa még nincs a sokszögnek: ennek egy szöveges állomány soron következő sora alapján kell beállítania a sokszög oldalainak számát, és a sokszög csúcsainak koordinátáit.

A feladat felpopulálása újra



input.txt

```
3
4 1 1 -1 1 -1 -1 1 -1
3 0 0 0 1 1 1
3 -1 0 1 0 0 1
20 20
```

```
ifstream inp("input.txt");
…
int n; inp >> n;
vector<Polygon*> t(n);

for( int i=0; i<n; ++i ) { t[i] = create(inp); }
```

Kell egy olyan függvény, amely egy szöveges állomány soron következő sora alapján hoz létre (példányosít) egy sokszöget, majd visszaadja annak a címét.

Sokszög dinamikus példányosítása

input.txt

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

```
Polygon* create(ifstream &inp) {
        Polygon *p;

        int sides;
        inp >> sides;
        p = new Polygon(sides);
        for(int i=0; i < sides; ++i) {
                int x, y; inp >> x >> y;
                p->_vertices[i].setPoint(x,y);
        }
        return p;
}
```

létrehozza a sokszöget

beállítja a sokszög csúcsainak koordinátáit

NEM JÓ!
Mivel a create() nem a metódusa a Polygon osztálynak, így nem fér hozzá annak privát _vertices adattagjához.

Sokszöget legyártó függvény

```
Polygon* Polygon::create(ifstream &inp) {
        Polygon *p;
        try {
                int sides;
                inp >> sides;
                p = new Polygon(sides);
                for(int i=0; i < sides; ++i) {
                        int x, y; inp >> x >> y;
                        p->_vertices[i].setPoint(x,y);
                }
        } catch(Polygon::Errors e){
                if(e==Polygon::FEW_VERTICES) {
                        cout << "cannot create the polygon.\n";
                }
        }
        return p;
}
```

Legyen a create() a Polygon osztály metódusa.
De melyik sokszög objektumra kellene meghívni ezt?
Éppen neki kellene létrehoznia egy ilyen objektumokat.

hiba figyelés

hibakezelés

osztályszintű metódus hívása

```
t[i] = Polygon::create(inp);
```

```
class Polygon {
public:
        enum Errors { … };
        Polygon(int n);
        …
        static Polygon* create(std::ifstream &inp);
private:
        vector<Point*> _vertices;
};
```

Legyen osztályszintű a create(): ne egy sokszög objektumhoz, hanem a sokszögek osztályához tartozzon. Ez lehetővé teszi, hogy lássa a rejtett elemeket is.

Főprogram

Tömbbeli sokszögek eltolása ugyanazon irányban és mértékkel, majd súlypontjaiknak kiszámolása.

$A = (t : \text{Sokszög}^n , \ mp : \text{Pont} , \ cout : \text{Pont}^n)$

$Ef = (t = t_0 \ \wedge \ mp = mp_0)$

sorozatok összefűzésének jele

$Uf = (mp = mp_0 \ \wedge \ t = \bigoplus_{i=1}^{n} < \text{eltol}(t_0[i], mp) > \wedge \ cout = \bigoplus_{i=1}^{n} < \text{súlypont}(t[i]) >)$

Két összegzés (összefűzés):

a két összegzés közös ciklusba vonható össze

$i \in [m..n]$	~	$i \in [1 .. n]$	$i \in [1 .. n]$
$f(i)$	~	$<\text{eltol}(t'[i], mp)>$	$<\text{súlypont}(t[i])>$
s	~	t	$cout$
$H,+,0$	~	$\text{Sokszög}^*, \oplus, <>$	$\text{Pont}^*, \oplus, <>$

az összefűzés helyett elég a már létező t elemeit egyenként megváltoztatni: t[i] := eltol(t[i], mp)

r := <>

i = 1 .. n

t[i] := eltol(t[i], mp)

cout := cout ⊕ súlypont(t[i])

t[i].eltol(mp)

t[i].súlypont()

```
for( Polygon *p : t ) {
      p->move(mp); cout << *p << endl;
      cout << p->center() << endl;
}
```

a cout-hoz történő hozzáfűzés valójában egy kiírás

Típus-orientált procedurális kód

```
#include "polygon.h"
#include "point.h"
using namespace std;

int main()
{
        cout << "file name: "; string fn; cin>> fn;
        ifstream inp(fn);
        if(inp.fail()) { cout << "File open error\n"; return 1;}
        int n; inp >> n;
        vector<Polygon*> t(n);
        for ( int i=0; i<n; ++i ) t[i] = Polygon::create(inp);
        int x, y; inp >> x >> y;
        Point mp(x, y);
        for ( Polygon* p : t ) {
                p->move(mp); cout << *p << endl;
                cout << p->center() << endl;
        }
        for ( Polygon* p : t ) delete p;
        return 0;
}
```

#include <iostream>
#include <fstream>
#include <vector>

populálás

számolás

törlés

megszünteti a dinamikus memóriában tett helyfoglalásainkat

Objektum-orientált kód

```
int main(){
        Application a;
        a.run();
        return 0;
}
```

```
class Application{
public:
        Application();
        void run();
        ~Application();
private:
        std::vector<Polygon*> t;
        Point mp;
};
```

populálás

számolás

törlés

```
Application::Application(){
        cout << "file name: "; string fn; cin>> fn;
        ifstream inp(fn);
        if(inp.fail()) { cout << "File open error\n"; exit(1);}
        int n; inp >> n;
        t.resize(n);
        for( int i=0; i<n; ++i) t[i] = Polygon::create(inp);
        int x, y; inp >> x >> y;
        mp.setPoint(x, y);
}
```

#include <cstdlib>

```
void Application::run(){
        for ( Polygon* p : t ) {
                p->move(mp); cout << *p << endl;
                cout << p->center() << endl;
        }
}
```

A destruktor akkor hívódik meg, amikor az objektum megszűnik.

```
Application::~Application(){
        for ( Polygon* p : t ) delete p;
}
```

Menüvezérelt objektum-orientált kód

```
int main()
{
	Menu a;
	a.run();
	return 0;
}
```

```
void Menu::run()
{
	int v = 0;
	do{
		menuWrite();
		cin >> v;  // ellenőrzés!
		switch(v){
			case 1: case1(); break;
			case 2: case2(); break;
			case 3: case3(); break;
			case 4: case4(); break;
		}
	}while(v != 0);
}
```

```
class Menu{
public:
	Menu(){s = nullptr;}
	void run();
	~Menu(){ if(s!=nullptr) delete s;}
private:
	Polygon* s;
	void menuWrite();
	void case1();
	void case2();
	void case3();
	void case4();
};
```

egy sokszöget létrehozó, kiíró, eltoló, súlypontját kiszámoló metódusok

```
void Menu::menuWrite(){
	cout << "0 – exit\n";
	cout << "1 – create\n";
	cout << "2 – write\n";
	cout << "3 – move\n";
	cout << "4 – center\n";
}
```

Menüpontok

input1.txt

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

input2.txt

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

```
void Menu::case1(){ // create
      if(s!=nullptr) delete s;
      cout << "file name: "; string fn; cin>> fn;
      ifstream inp(fn);
      if(inp.fail()) { cout << "File open error\n"; return;}
      s = Polygon::create(inp);
}
```

```
void Menu::case2(){  // write
      if(s==nullptr) { cout << "There is no polygon!\n"; return;}
      cout << *s << endl;
}
```

```
void Menu::case3(){  // move
      if(s==nullptr) { cout << "There is no polygon!\n"; return;}
      int x, y;
      cout << "x = "; cin >> x;
      cout << "y = "; cin >> y;
      Point mp(x, y);
      s->move(mp);
}
```

```
void Menu::case4(){  // center
      if(s==nullptr) { cout << "There is no polygon!\n"; return;}
       cout << s->center() << endl;
}
```