

Programozás alapjai II.

(5. ea) C++

Objektummodell, öröklés, virtuális tagfüggvény

Szeberényi Imre, Somogyi Péter

BME IIT

<szebi@iit.bme.hu>



M Ű E G Y E T E M 1 7 8 2

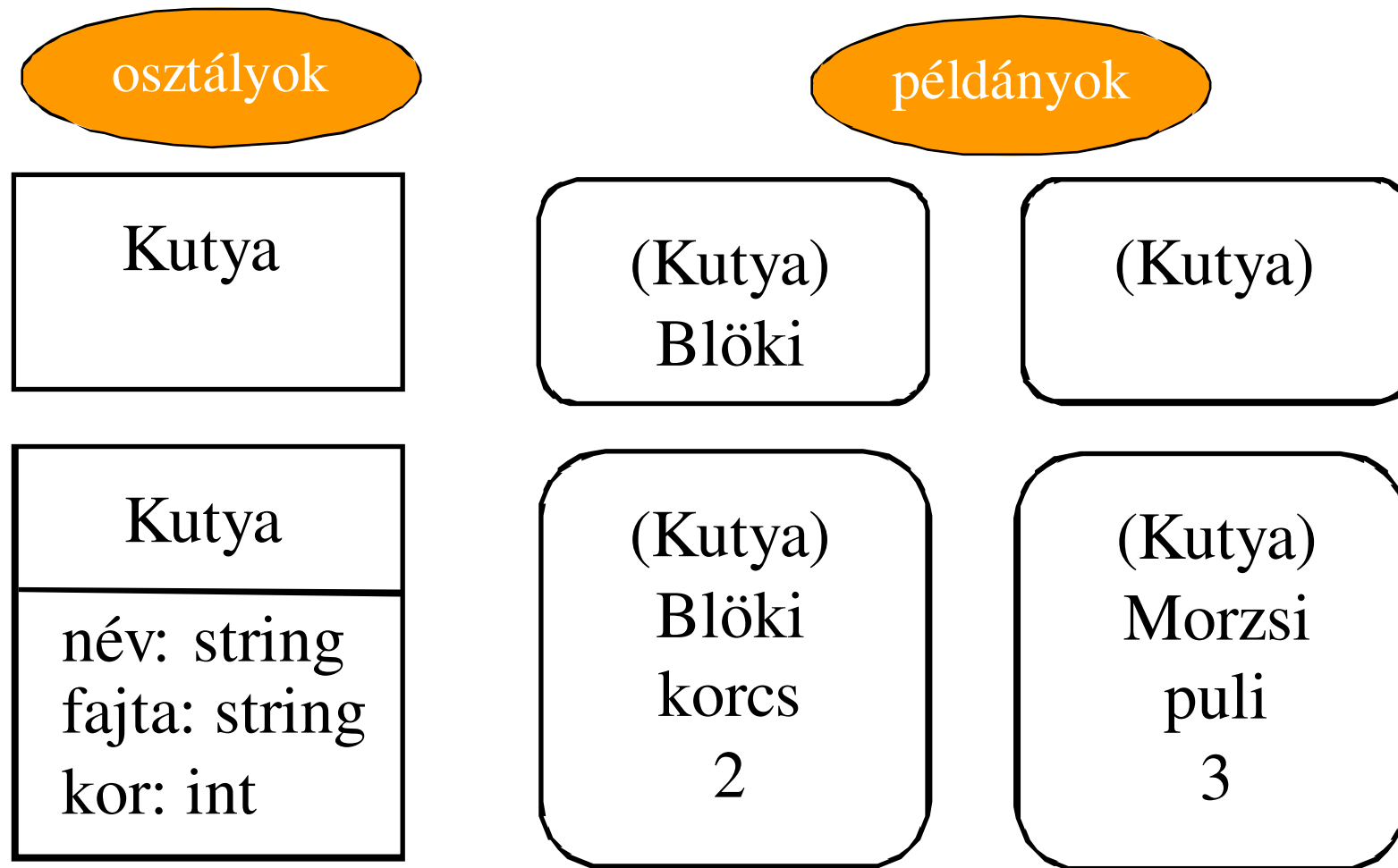
OO modellezés fogalmai újból

- Objektum
 - adat (állapot) és a rajta végezhető művelet
 - a világ egy részének egy olyan modellje, amely külső üzenetekre reagálva valahogyan viselkedik (változtatja az állapotát, újabb üzenetet küld)
 - üzenetekre (message), vagy eseményekre (event) a metódus végrehajtásával reagál, viselkedik (behaviour)
 - polimorf működés

OO modellezés fogalmai újból/2

- Objektum osztály, osztály (class)
 - megegyező viselkedésű és struktúrájú objektumok mintája, gyártási forrása. (pl, ház, ablak, kutya)
- Objektum példány, objektum (instance)
 - Minden objektum önállóan, létező egyed (Blöki, Morzsi, Bikfic)

Osztály és példány jelölése



Osztály és típus

- **int i;**
 - i nevű objektum aminek a mintája int
- Nem teljesen azonos, mert a típus egy objektum-halmaz viselkedését **specifikálja**.
- Az osztály a típus által meghatározott viselkedést **implementálja**.
- Egy adott objektumtípust többféleképpen lehet implementálni (megvalósítani).

Osztály és típus/2

- A komplex számok viselkedését például modellezhetjük olyan osztállyal, ami valós és képzetes részt tárol, de olyannal is, ami polárkoordinátákat tárol.
- A kétfajta megvalósítás osztály szinten különböző, de típusuk – viselkedésük – interfész szinten azonos.
- Ennek ellenére gyakran az osztályt (pontatlanul) típusként kezeljük.
- Hagyományos nyelveken a típus érték-, konstans- és művelethalmazt jelöl.

Modellezés objektumokkal

- Különböző szempontok szerint modellezünk.

- Objektummodell

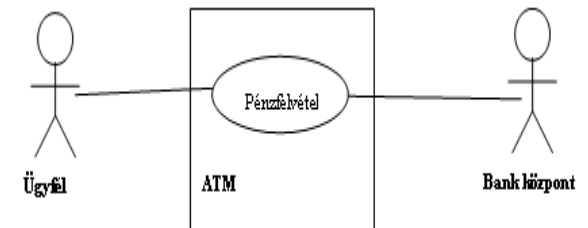
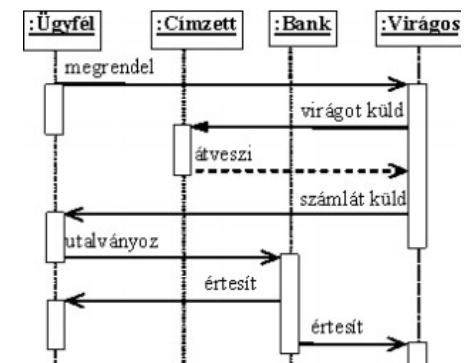
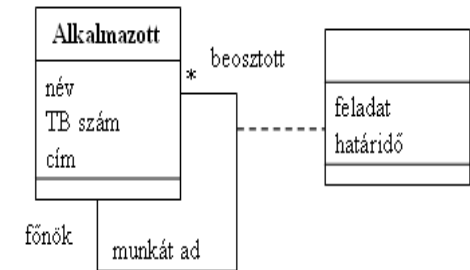
- Adat szempontjából írja le a rendszer statikus tulajdonságait (osztály vagy entitás-relációs diagram)

- Dinamikus modell

- A működés időbeliségét rögzíti (állapotgráf, kommunikációs diagram).

- Funkcionális modell

- Funkció szerint ír le (használati esetek).



Modellezés eszközei, módszertana



- Részletesen **szoftvertechnológia** c. tárgyban a következő félévben.
- Itt csak minimális alapok a nyelvi eszközök megismeréséhez, a **jelölések** használatához.

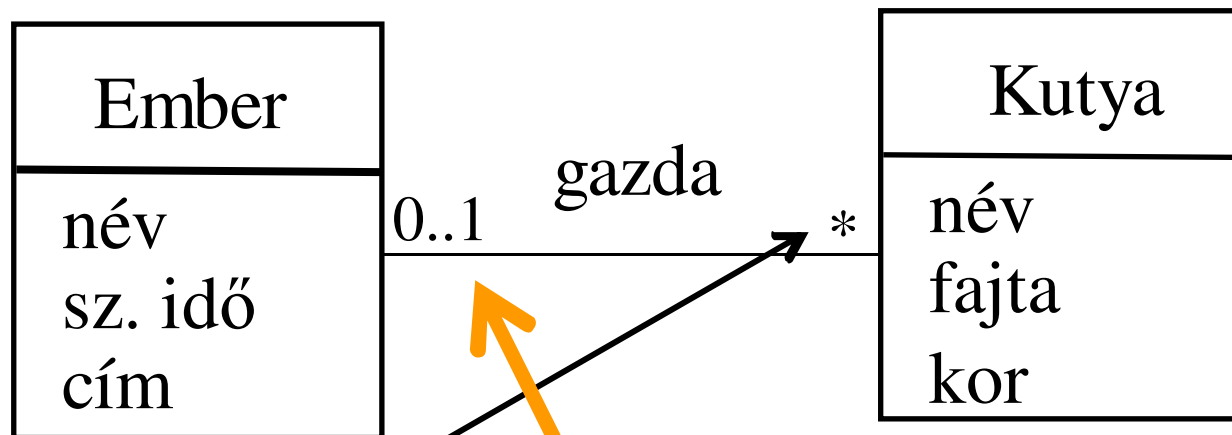
[http://www.tankonyvtar.hu/hu/tartalom/tkt/
objektum-orientalt/ch03.html](http://www.tankonyvtar.hu/hu/tartalom/tkt/objektum-orientalt/ch03.html)

Objektummodell



- Attribútumok leírása
 - Elnevezés típusú attribútumok. Nem vagy ritkán változnak (név, személyi szám, nem)
 - Leíró attribútumok (jövedelem, kor)
 - Referenciák. Kimutatnak az objektumból (cím).
- Kapcsolatok (relációk és linkek) leírása
 - asszociáció, aggregáció, komponens – osztályok közötti kapcsolat leírása
 - link (lánc): példányok közötti kapcsolat
- Öröklés leírása

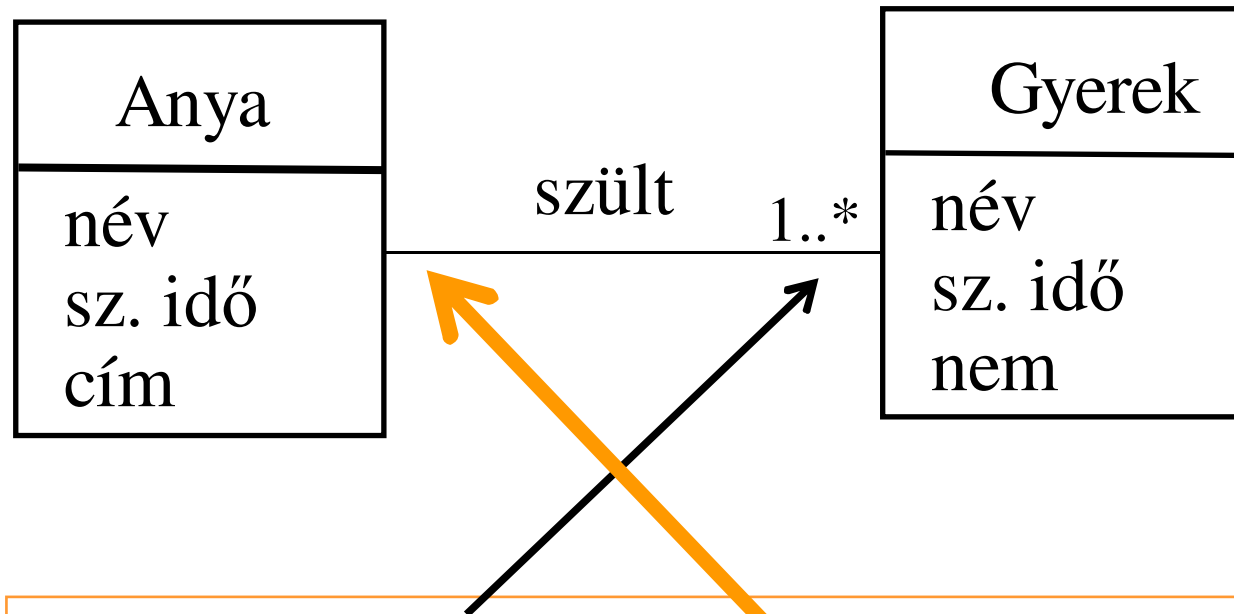
Példák a kapcsolatok leírására



Egy ember 0 vagy több kutyanak lehet gazdája.

Egy kutyanak legfeljebb egy gazdája van, de lehet, hogy gazdátlan.

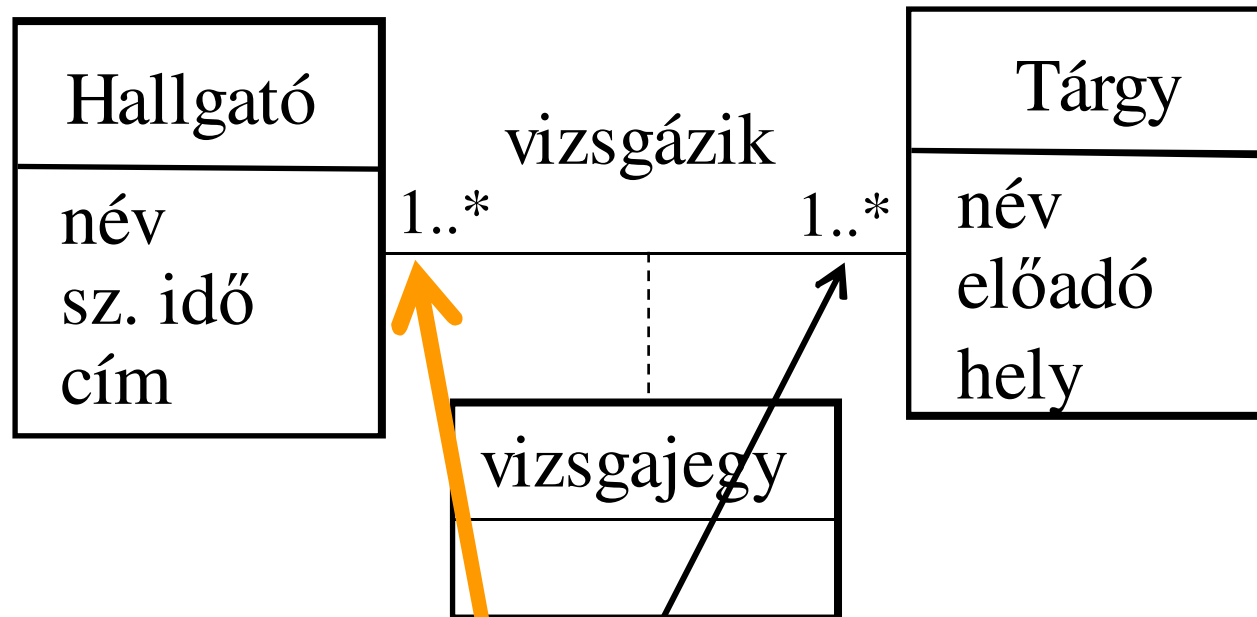
Egy – több kapcsolat



Egy anya legalább egy gyereket szült (1..*).

Egy gyereket pontosan egy anya szült.

Kapcsolatok attribútumai



Egy hallgató több tárgyból is vizsgázhat.
Egy tárgyból több hallgató is vizsgázhat.
A vizsga eredménye (attribútuma) a vizsgajegy.

Komponens reláció



A karakter része a bekezdésnek, a bekezdés része a levélnek.

Elnevezés: szülő – gyerek viszony, de nem keverendő össze az örökléssel!

Komponens vs Aggregáció



A hallgatókat nem gyilkoljuk le, ha megszűnik a tankör.
Ha az évfolyam megszűnik a tankörökre nincs szükség.

Öröklés

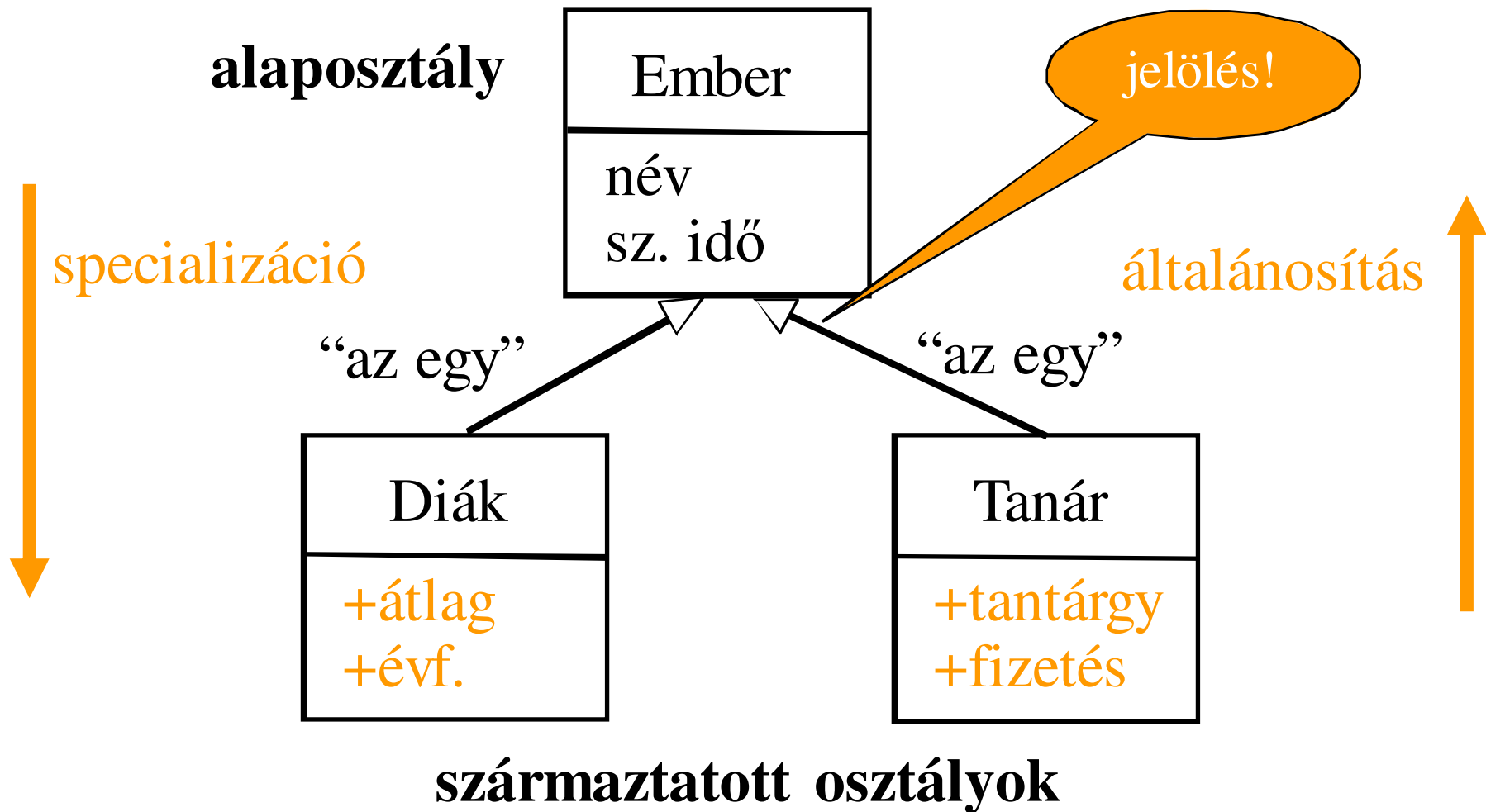


- Az öröklés olyan implementációs és modellezési eszköz, amelyik lehetővé teszi, hogy egy osztályból olyan **újabb osztályokat származtassunk**, melyek **rendelkeznek** az eredeti osztályban **már definiált** tulajdonságokkal, szerkezettel és viselkedéssel.
- Újrafelhasználhatóság szinonimája.
- Nem csak bővíthető, hanem a tagfüggvények felül is definiálhatók (override)

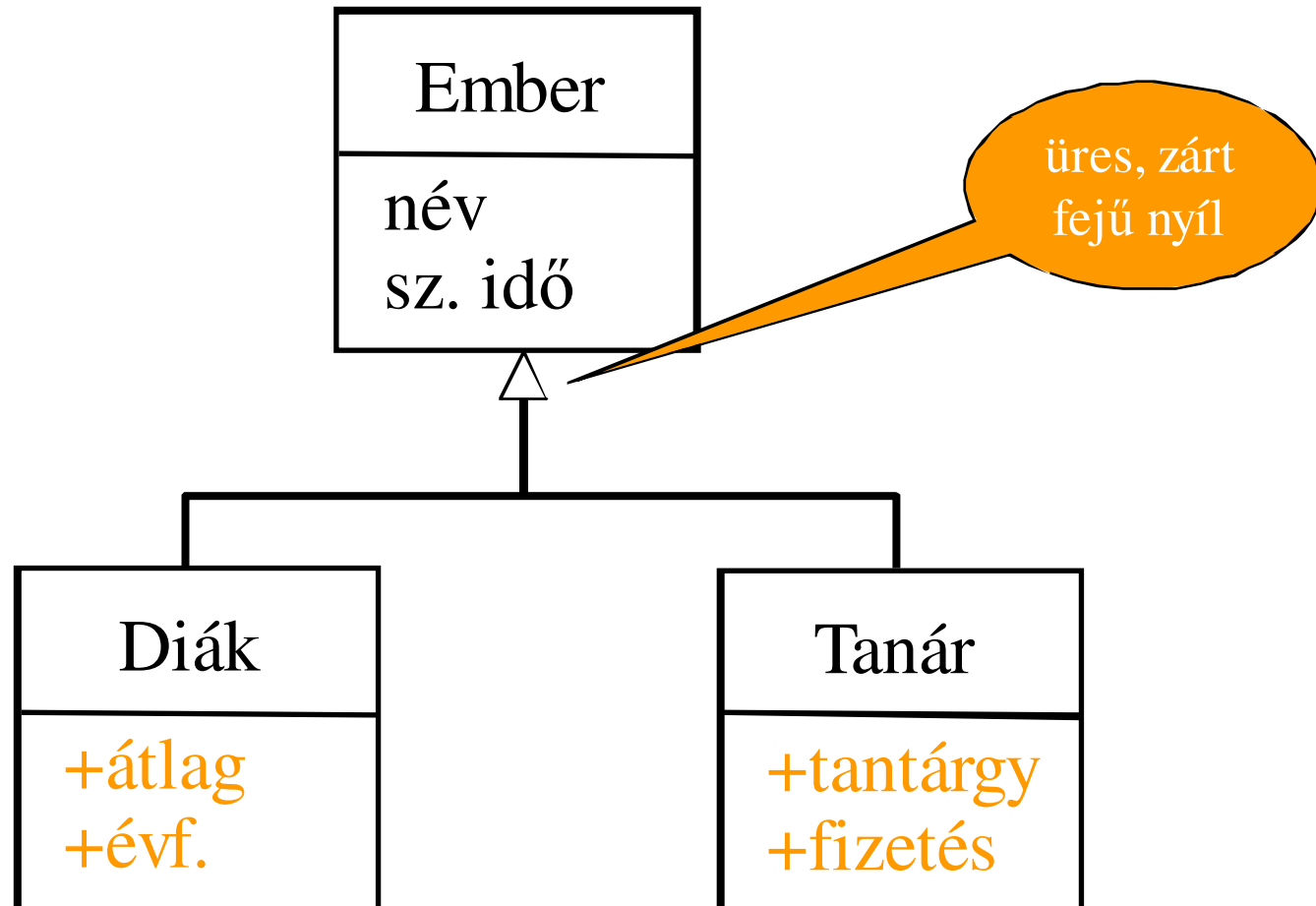
Feladat

- Diákokból, tanárokból álló rendszert szeretnénk modellezni.
 - Diák attribútumai:
név, sz. idő, átlag, évfolyam
 - Tanár attribútumai:
név, sz. idő, tantárgy, fizetés
- Milyen osztályokat hozzunk létre ?
- 2 független osztály ?
 - név, sz. idő 2x, műveletek 2x, nehezen módosítható

Örökléssel



Öröklés másként jelölve



Ember alaposztály

```
class Ember {  
    String nev;  
    Date szulDatum;  
public:  
    Ember();  
    void setDate(Date d);  
    void setName(char *n);  
    const char *getName() const;  
    ...  
};
```

Származtatott osztályok

```
class Diak :public Ember {
```

```
    double atlag;
```

```
public:
```

```
    Diak();
```

```
    void setAv(double a);
```

```
    ....
```

```
};
```

```
class Tanar :public Ember {
```

```
    double fizetes;
```

```
public:
```

```
    Tanar(); ....
```

```
};
```

Alaposztályból minden látszik ami publikus

+Új attribútum

+Új tagfüggvény

Öröklés előnyei

- Hasonlóság kiaknázása
 - Világosabb programstruktúra
- Módosíthatóság mellékhatások nélkül
 - Újabb tulajdonságok hozzáadása
- Kiterjeszthetőség
 - Újrafelhasználható

Öröklés fajtái

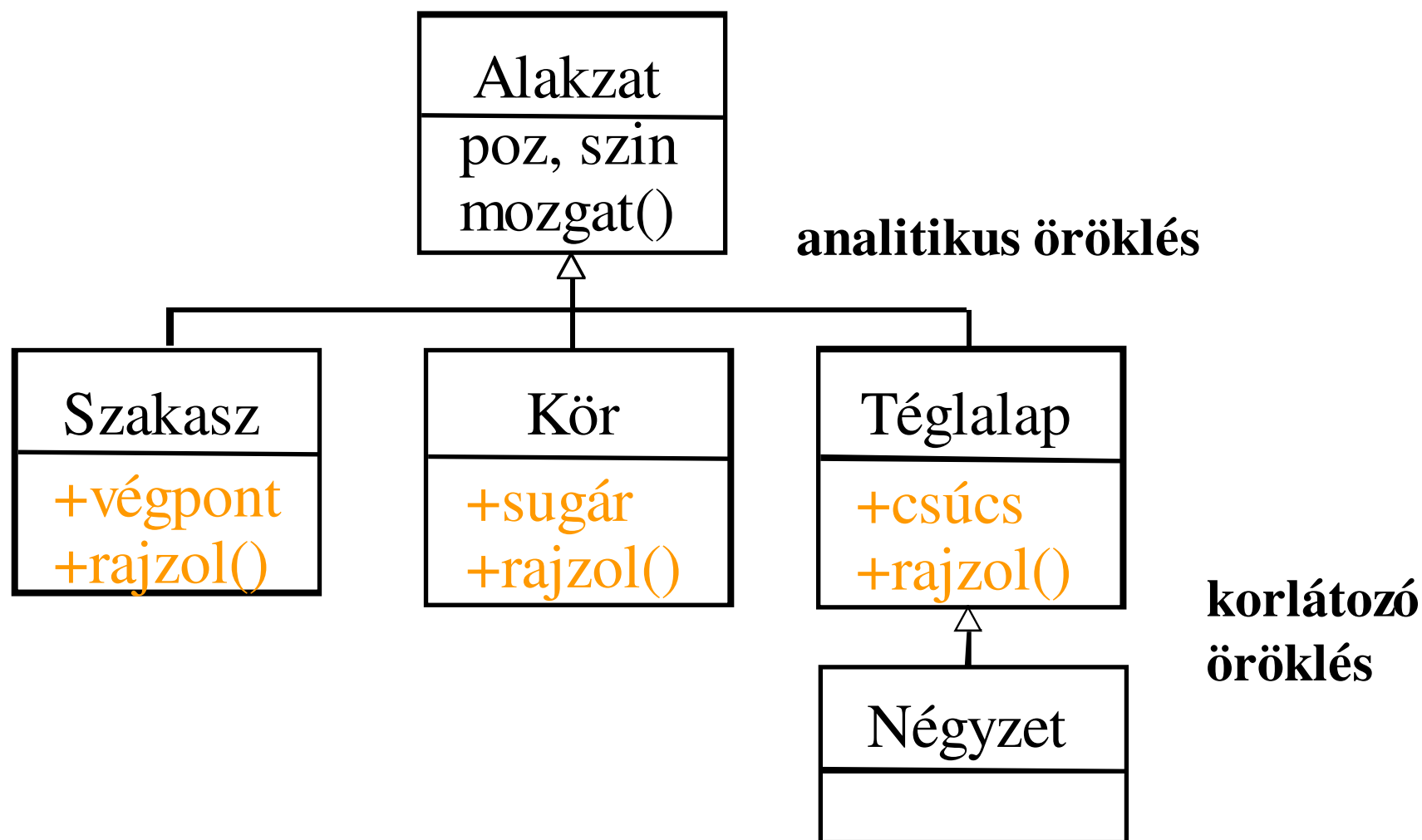
I.

- Analitikus
- Korlátozó

II.

- Egyszerű
- Többszörös

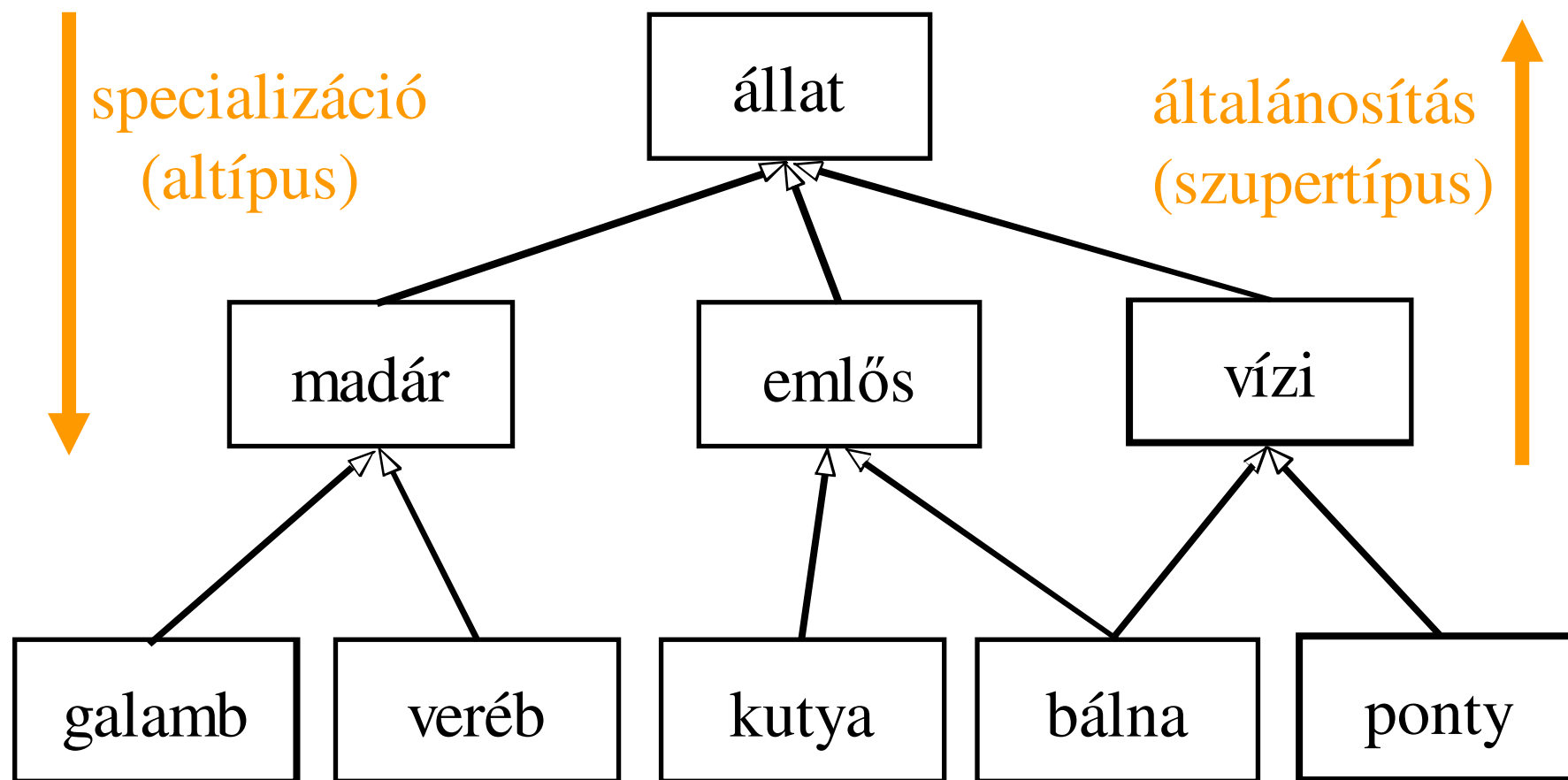
Analitikus és korlátozó öröklés



Kompatibilitás és öröklés

- *A* típusú objektum kompatibilis *B*-vel, ha *A* típusú objektum **bárhol és bármikor** alkalmazható, ahol *B* használata megengedett. (Liskov szubst. tétel)
- A reláció reflektív (*A* komp. *A*-val) és tranzitív, de nem szimmetrikus.
- A kompatibilitás egy hierarchiát szab meg
 - pl: állat \leftarrow madár \leftarrow veréb

Kompatibilitás/2



Alakzatok a rajztáblán

- Objektumok (szereplők):
 - szakasz, kör, téglalap, (Alakzatok)
 - Rajztábla
- Alakzat műveletei:
 - mozgat, rajzol
- Alakzat attribútumai:
 - pozíció, szín
- Hogyan mozgat?
 - letöröl, felrajzol

Van közös attribútum?

Vonal:

pozíció, szín

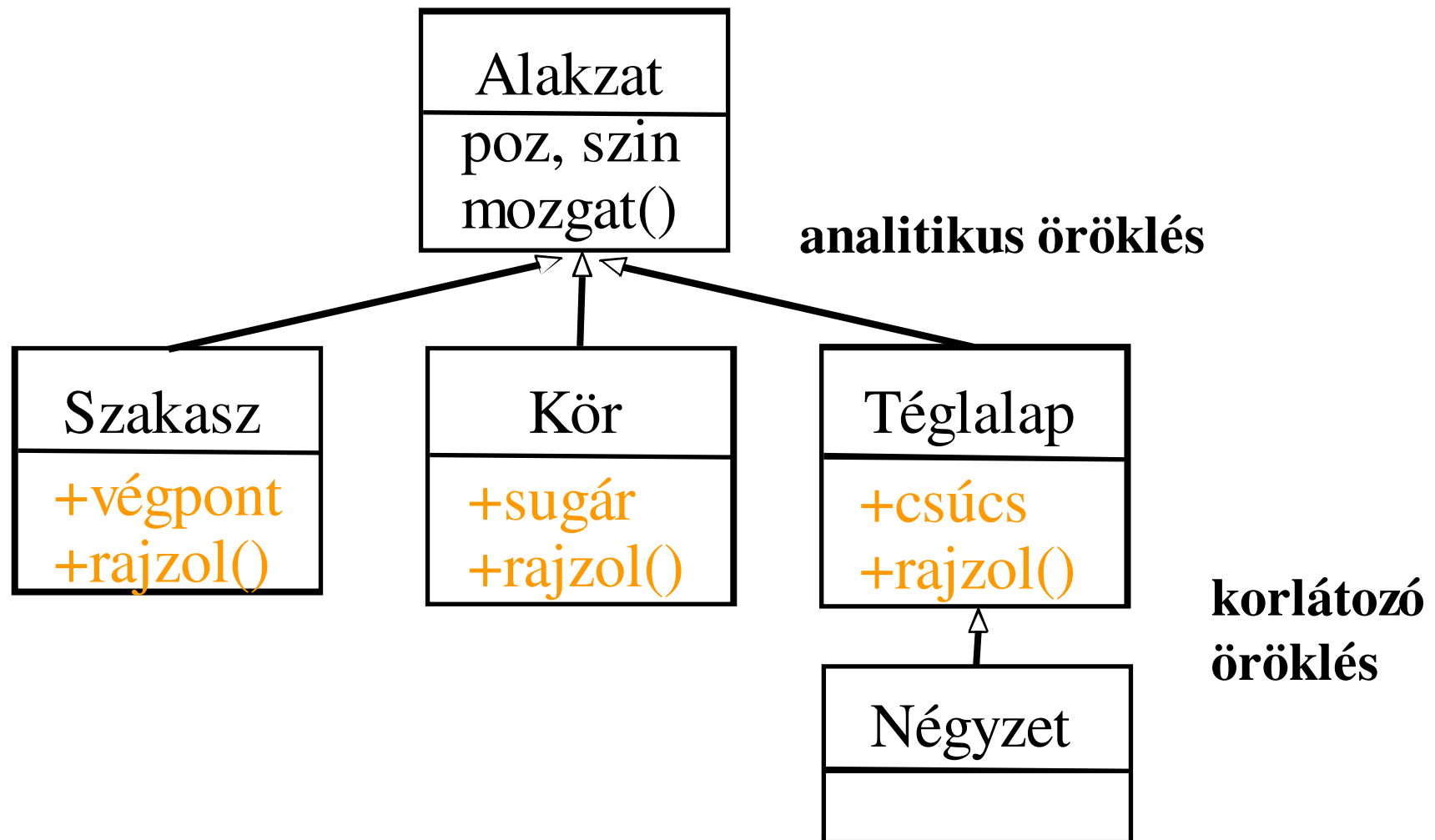
+ végpont

Kör:

pozíció, szín

+ sugár

Geometria alakzatok C++-ban



Alakzat alaposztály

```
class Alakzat {
```

```
protected:
```

```
    int x, y;
```

```
    int szin;
```

```
public:
```

```
    Alakzat(int x0, int y0, int sz)
```

```
        :x(x0), y(y0), szin(szin) { }
```

```
// mozgat(), érezzük, hogy itt a helye, de nem
```

```
// tudjuk hogyan kell megvalósítani a rajzolás részét.
```

```
// Ezért oda tesszük, ahol már ismert a
```

```
// rajzolás menete.
```

```
};
```

Védelem enyhítése a
leszármazottak felé

Szakasz osztály

```
class Szakasz : public Alakzat {  
    int xv, yv;  
public:  
    Szakasz(int x1, int y1, int x2, int y2, int sz)  
        : Alakzat(x1, y1, sz), xv(x2), yv(y2) { }  
    void rajzol( );  
    void mozgat(int dx, int dy);  
};
```

Alaposztályból minden
látszik ami publikus

Itt már tudjuk, hogyan
kell rajzolni.

Szakasz tagfüggvényei

```
void Szakasza :: Rajzol( ) {  
    .... // szakaszt rajzol  
}
```

```
void Szakasza :: mozgat( int dx, int dy ) {  
    int sz = szin;           // tényleges rajzolósi szín elmentése  
    szin = BACKGND; // rajzolósi szín legyen a háttér színe  
    rajzol( );              // A vonal letörlése az eredeti helyről  
    x += dx; y += dy; // mozgatás: a pozíció változik  
    szin = sz;             // rajzolósi szín a tényleges szín  
    rajzol( );             // A vonal felrajzolósa az új pozícióra  
}
```

Téglalap osztály

```
class Teglalap : public Alakzat {  
    int xc, yc;  
public:  
    Teglalap(int x1, int y1, int x2, int y2, int sz)  
        : Alakzat(x1, y1, sz), xc(x2), yc(y2) { }  
    void rajzol( );  
    void mozgat(int dx, int dy);  
};
```

Ugyanaz, mint a szakasznál,
csak a hívott rajzol() más

mozgat() helye

- Tegyük a származtatott osztályokba ?
 - látszólag **ugyanaz** a függvény minden alakzatban
 - csak az általa **hívott rajzol()** más
- Tegyük az alaposztályba ?
 - ha a hívott rajzol()-t egy **manó** le tudná cserélni mindig a megfelelő származtatott rajzol()-ra, akkor működne → **virtuális függvény**

Alakzat osztály virtuális függvénnnyel

```
class Alakzat {
```

```
protected:
```

```
    int x, y;
```

```
    int szin;
```

```
public:
```

```
    Alakzat(int x0, int y0, int sz)
```

```
        :x(x0), y(y0), szin(szin) { }
```

```
    virtual void rajzol( ) { }
```

```
    void mozgat(int dx, int dy);
```

```
};
```

Az öröklés során újabb jelentést kaphat, ami az alaposztályból is elérhető, így a mozgat()-ból is.

Most már ide tehetjük, mert a rajzol() is elérhető.

Alakzat mozgat() tagfüggvénye

```
void Alakzat :: Mozgat( int dx, int dy ) {  
    int sz = szin;          // tényleges rajzolási szín elmentése  
    szin = BACKGRD; // rajzolási szín legyen a háttér színe  
    rajzol( );              // A vonal letörlés az eredeti helyről  
    x += dx; y += dy; // mozgatás: a pozíció változik  
    szin = sz;              // rajzolási szín a tényleges szín  
    rajzol( );              // A vonal felrajzolása az új pozícióra  
}
```

Szakasz osztály újra

```
class Szakasz : public Alakzat {  
    int xv, yv;  
public:  
    Szakasz(int x1, int y1, int x2, int y2, int sz)  
        : Alakzat(x1, y1, sz), xv(x2), yv(y2) { }  
    void rajzol( ); // átdefiniáljuk a virt. fv-t.  
    void mozgat(int dx, int dy);  
};  
void Szakasz::rajzol( ) {  
    .... // szakaszt rajzol.  
    // Az alaposztályból hívva is ez hívódik  
}
```

Téglalap osztály újra

```
class Teglalap : public Alakzat {  
    int xc, yc;  
public:  
    Teglalap(int x1, int y1, int x2, int y2, int sz)  
        : Alakzat(x1, y1, sz), xc(x2), yc(y2) { }  
    void rajzol( ); // átdefiniáljuk a virt. fv-t.  
    void mozgat(int dx, int dy);  
};  
void Teglalap::rajzol( ) {  
    .... // téglalapot rajzol.  
    // Az alaposztályból hívva is ez hívódik  
}
```

Mintaprogram

```
main ( ) {  
    Teglalap tegla(1, 10, 2, 40, RED); // téglalap  
    Szakaszz szak(3, 6, 80, 40, BLUE); // szakasz  
    Alakzat alak(3, 4, GREEN);          // ???  
    alak.mozgat(3, 4);                  // 2 db rajzol() hívás  
    szak.rajzol( );                     // 1 db rajzol()  
    szak.mozgat(10, 10);                // 2 db rajzol() hívás  
    Alakzat *ap[10];  
    ap[0] = &szak;                      // nem kell típuskonverzió  
    ap[1] = &tegla;                     // kompatibilis  
    for (int i = 0; i < 2; i++ ) ap[i] ->rajzol();  
}
```

Mikor melyik rajzol() ?

	Virtuális Alakzat::rajzol()	Nem virtuális Alakzat::rajzol()
alak.mozgat()	Alakzat::rajzol()	Alakzat::rajzol()
szak.rajzol()	Szakasz::rajzol()	Szakasz::rajzol()
szak.mozgat	Szakasz::rajzol()	Alakzat::rajzol()
sp[0]->rajzol() Szakasz-ra mutat	Szakasz::rajzol()	Alakzat::rajzol()
sp[1]->rajzol() Teglalap-ra mutat	Teglalap::rajzol()	Alakzat::rajzol()

Alakzat önállóan ?

Alakzat alak(3, 4, GREEN); // ???

alak.mozgat(3, 4); // Mit rajzol ??

- Nem értelmes példányosítani, de lehet, mivel osztály.
- Nyelvi eszközzel tiltjuk a példányosítást:

Absztrakt alaposztály

Absztrakt alaposztályok

- Csak az öröklési hierarchia kialakításában vesznek részt, nem példányosodnak
- A virtuális függvényeknek nincs értelmes törzse:
tisztán (pure) virtuális függvény

```
class Alakzat {  
protected:  int x, y, szin;  
public:
```

Nem
példányosítható

```
    Alakzat( int x0, int y0, int sz) ;
```

```
    void  mozgat( int dx, int dy );
```

```
    virtual void rajzol( ) = 0; // tisztán virtuális
```

```
    virtual ~Alakzat() {}; // Ez meg mi ?
```

```
};
```


Virtuális függvények szerepe

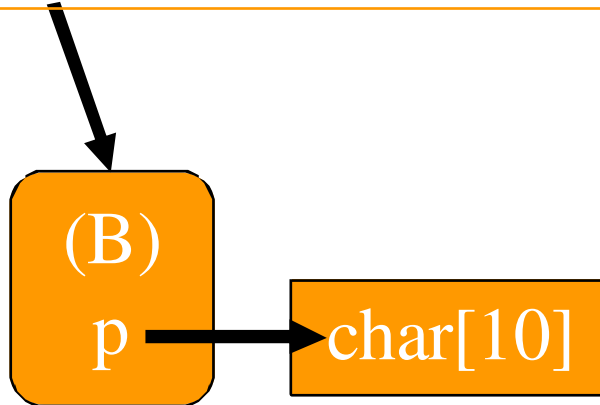
- Az alaposztály felől (alaposztály pointerét/referenciáját használva) elérhető a származtatott osztály megfelelő tagfüggvénye.
- Nagyon fontos szerepe van:
 - a heterogén kollekciókban (majd később)
 - dinamikus területet foglaló objektumok helyes megvalósíthatóságában: **virtuális destruktork**

Virtuális destruktorki hiánya

```
struct A {  
    ~A() {};  
};
```

```
A *pa = new B;
```

```
class B :public A {  
    char *p;  
public:  
    B() { p=new char[10]; }  
    ~B() { delete[] p; }  
};
```



Dinamikus
memóriaterület

Virtuális destruktor hiánya/2

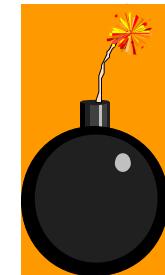
```
struct A {  
    ~A() {};  
};
```

```
A *pa = new B;  
delete pa;
```

```
class B :public A {  
    char *p;  
public:  
    B() { p=new char[10]; }  
    ~B() { delete[] p; }  
};
```

char[10]

Dinamikus
memóriaterület
nem szabadult fel,
pedig B jól kezeli!



Virtuális destruktore szerepe

```
struct A {  
    virtual ~A() {};  
};
```

```
A *pa = new B;
```

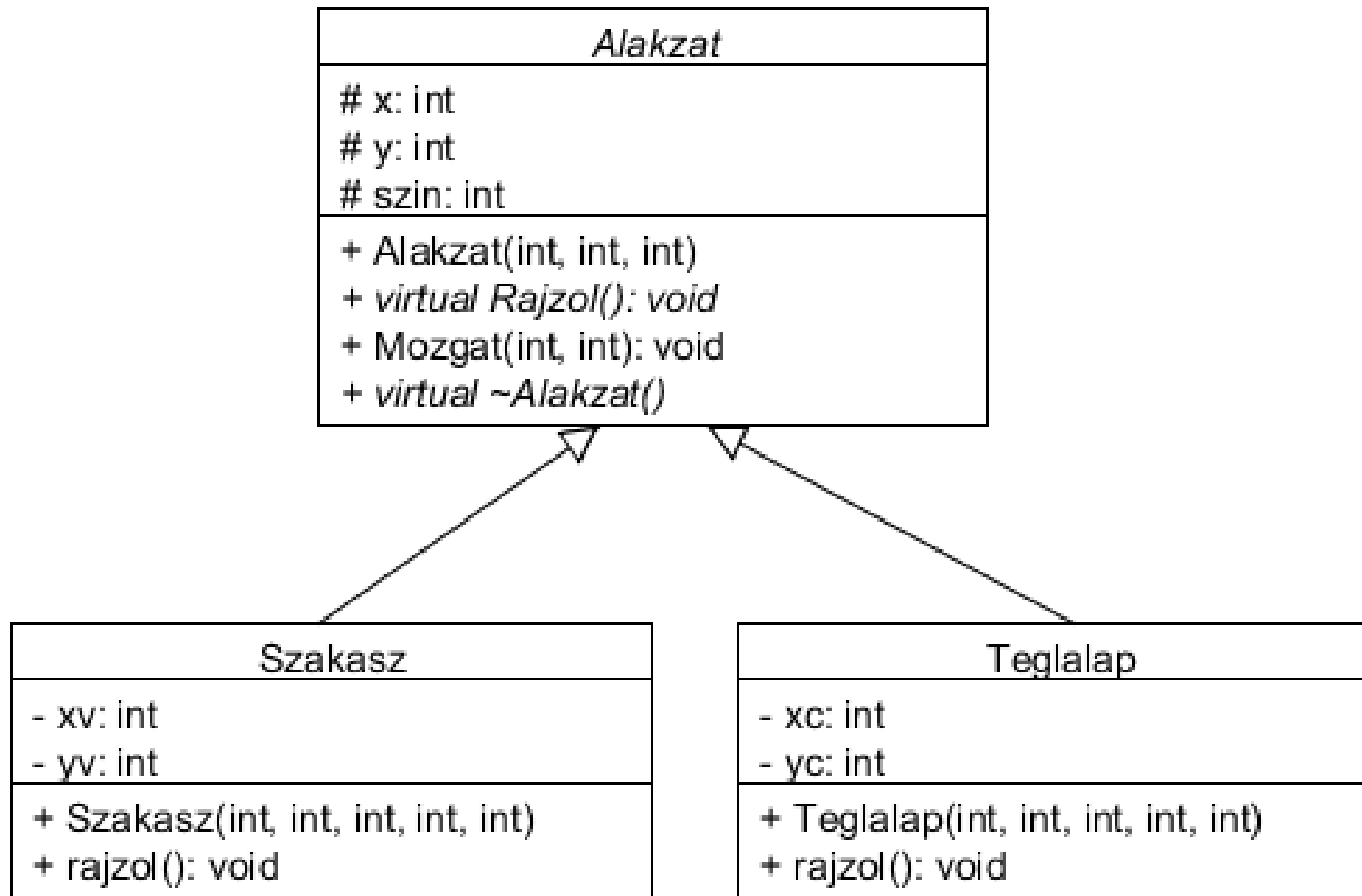
```
delete pa;
```

```
class B :public A {  
    char *p;  
public:  
    B() { p=new char[10]; }  
    ~B() { delete[] p; }  
};
```

A lefoglalt terület
felszabadult, mert a
származtatott osztály
destruktor is lefutott

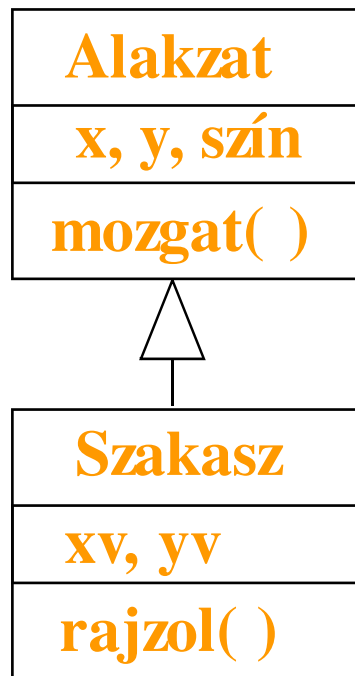
git.ik.bme.hu/Prog2/eloadas_peldak/ea_05 → virt_destruktor és otthonMegMukodott

Most itt tartunk a feladattal



Öröklés impl., ha nincs virtuális fv.

C++ osztályok



C struktúrák

struct Alakzat

x
y
szín

struct Szakasz

x
y
szín
xv
yv

Új rész

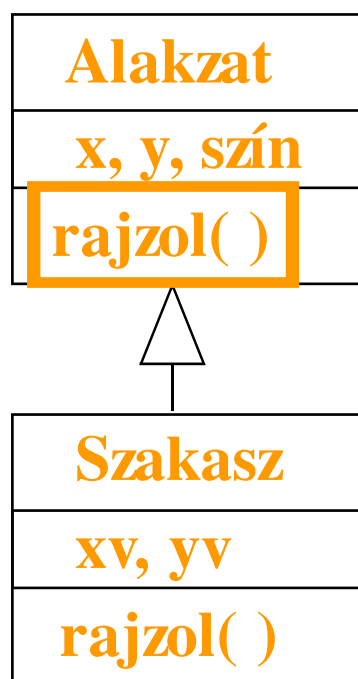
C globális függvények

AlakzatMozgat() SzakaszRajzol()

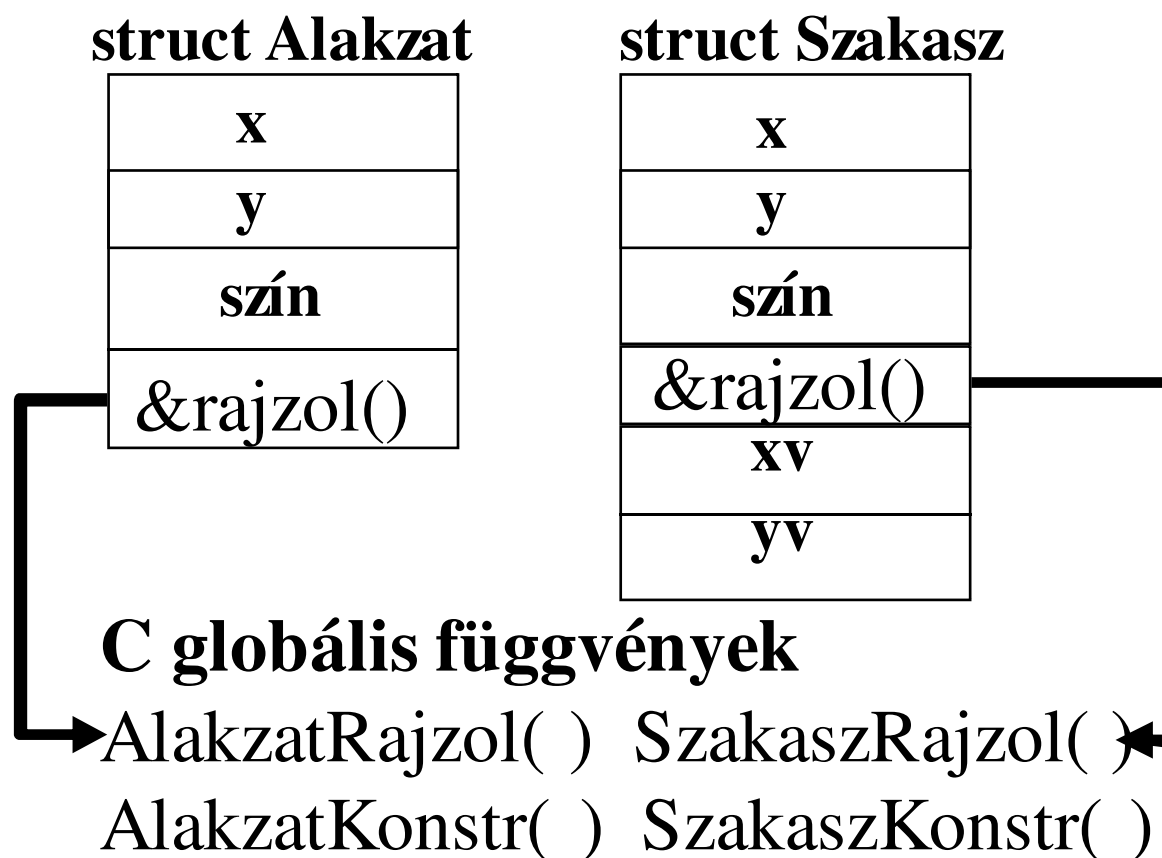
AlakzatKonstr() SzakaszKonstr()

Öröklés impl., ha a Rajzol() virtuális

C++ osztályok



C struktúrák



Alakzat C implementációja

```
struct Alakzat { int x, y, szin; void (*rajzol)( ); };
```

```
void AlakzatMozgat( struct Alakzat *this ) { }
```

```
void AlakzatKonstr(struct Alakzat *this, int x0,  
                  int y0, int sz) {
```

```
    this->rajzol = AlakzatRajzol; // manó v. fordító ?
```

```
    this->x = x0;
```

```
    this->y = y0;
```

```
    this->szin = sz;
```

```
}
```


Alakzat C implementációja/2

```
void AlakzatMozgat(struct Alakzat *this, int dx, int dy ) {  
    int sz = this->szin;  
    this->szin = BACKGND;  
    (*(this->rajzol))(this);  
    this->x += dx; this->y += dy;  
    this->szin = sz;  
    (*(this->rajzol))(this);  
}
```

Téglalap osztály újra

```
class Teglalap : public Alakzat {  
    int xc, yc;  
public:  
    Teglalap(int x1, int y1, int x2, int y2, int sz)  
        : Alakzat(x1, y1, sz), xc(x2), yc(y2) { }  
    void ujMeret(int x2, int y2)  
        { xc = x + x2; yc = y + y2; }  
    void rajzol( );  
    // mozgat() az alaposztályban  
};
```

Négyzet osztály (korlátoz)

```
class Negyzet : private Teglalap {  
public:  
    Negyzet(int x1, int y1, int s, int sz)  
        : Teglalap(x1, y1, x1+s, y1+s, sz) { }  
    void rajzol( ) { Teglalap::rajzol(); }  
    void mozgat(int dx, int dy)  
        { Teglalap::mozgat(dx, dy); }  
};
```

Eltakarja az alaposztályt

Az ujMeret() fv-t így kívülről elérhetetlenné tettük (korlátoztuk az elérését)

Összefoglalás

- Objektummodell
 - Attribútumok
 - Kapcsolatok (relációk)
- Öröklés (specializáció \leftrightarrow általánosítás)
 - analitikus v. korlátozó
 - egyszerű v. többszörös
- C++ nyelvi eszköz:
 - analitikus \rightarrow **public**, korlátozó \rightarrow **private**
 - tagfüggvények átdefiniálása, **protected** mezők
 - **virtuális tagfüggvény**: **alaposztály felől** elérhető a származtatott osztály tagfüggvénye,
 - **absztrakt** alaposztály nem példányosítható

Elérhetőség összefoglalása

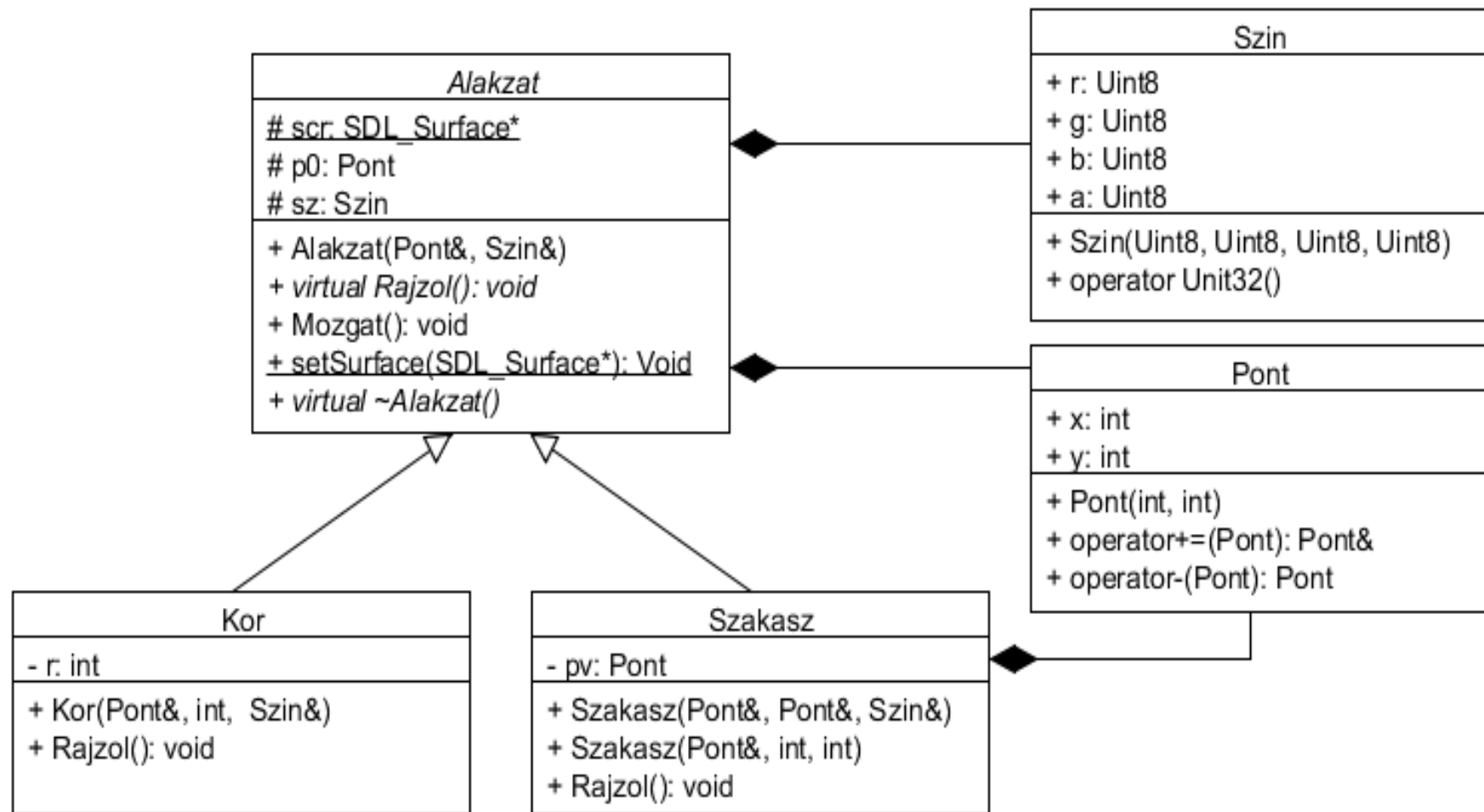
attr \ honan	külső	származtatott	tagfüggvény és barát
public:	✓	✓	✓
protected:		✓	✓
private:			✓

Példa

- Rajzoljuk ki az alakzatokat!
 - használjuk az SDL-t.
 - bővítsük az objektummodellt
- Felrajzol pár alakzatot, melyek az egérmozgással együtt mozognak.
- Csak az irányt követik, nem a mozgás nagyságát.

git.ik.bme.hu/Prog2/eloadas_peldak/ea_05 → SDL_alakzat

Objektummodell kibővítve



Előző modell kiegészítései

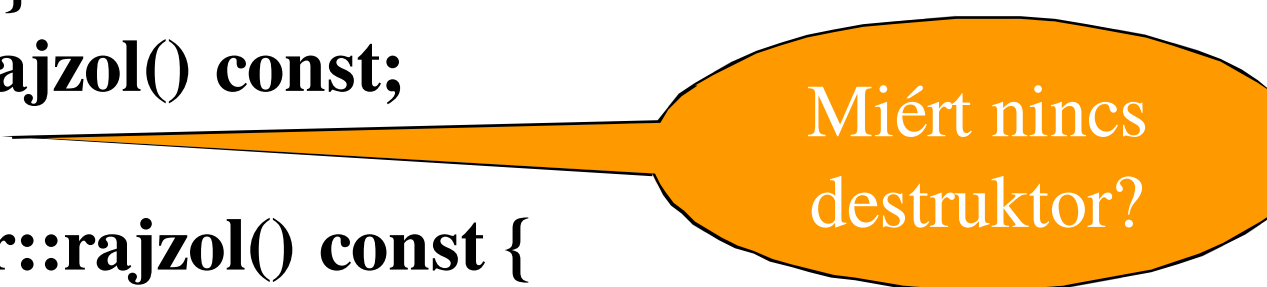
- Pont osztály bevezetése: rugalmasabb, könnyebben bővíthető (pl. 3D-re).
- Szín osztály: SDL-hez alkalmazkodik.
- Mindkettő teljesen publikus – ügysem hozunk belőlük létre önálló példányt, egyszerűbb a haszn.
- Alakzat osztályban statikus taggal rejtjük el az SDL egyik globális adatát. (surface/SDL1, renderer/SDL2)
- Minden rajzolás után van frissítés → villog, de nem a felhasználói élmény a lényeg.

Kiegészített alakzat

```
class Alakzat {  
protected:  
    Pont p0;                // alakzat origója  
    Szin sz;                // alakzat színe  
    static SDL_Surface *scr; // eldugott "globális" (SDL1)  
public:  
    Alakzat(const Pont& p0, const Szin& sz)  
        :p0(p0), sz(sz) {}  
    const Pont& getp0() const { return p0; }  
    static void setSurface(SDL_Surface* s) { scr = s; }  
    virtual void rajzol() const = 0; // tisztán virt.  
    void mozgat(const Pont& d);  
    virtual ~Alakzat() {} // fontos, ha az alap. felől szabadítunk fel  
};
```

Kor

```
class Kor : public Alakzat {  
    int r;           //< sugár  
public:  
    Kor(const Pont& p0, int r, Szin sz)  
        :Alakzat(p0, sz), r(r)    // Ős osztály inic  
    {}  
    void rajzol() const;  
};  
void Kor::rajzol() const {  
    filledCircleColor(scr, p0.x, p0.y, r, sz);  
    SDL_Flip(scr);    // képfriessítés (SDL1)  
}
```



Miért nincs destruktork?

A rajztábla most egy tömb

alakzat *idom[100]; unsigned int db = 0;

működési vázlat

```
while (VAN ESEMÉNY) {  
    if (GOBNYOMÁS) {  
        if ( db < 100-1) {  
            idom[db] = new Kor(Pont(x, y), 40, RED); // felvesz  
            idom[db]->rajzol();                      // kirajzol  
            ++db;  
        }  
    } else if (EGERMOZGÁS) {  
        for (int i = 0; i < db; i++) {  
            idom[i]->mozgat(Pont(dx, dy));  
            idom[i]->rajzol();                      // kirajzol  
        }  
    }  
}  
for (int i = 0; i < db; i++) delete idom[i]; // letöröl
```

