Programozás alapjai II.
(4. ea) **C++**

analitikus és korlátozó öröklés

Szeberényi Imre
BME IIT
<szebi@iit.bme.hu>

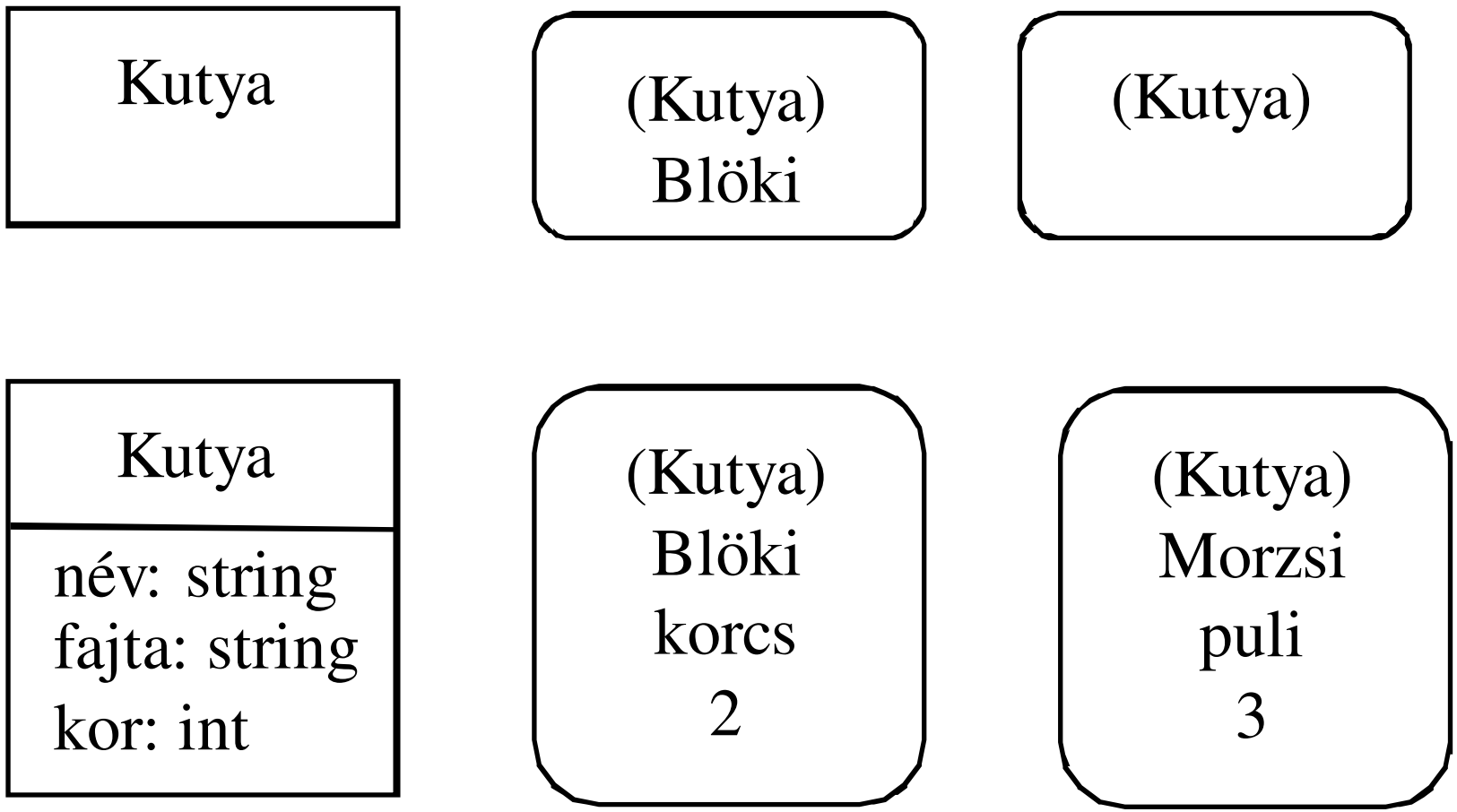
OO modellezés fogalmai újból

- 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 (behavior)
  - 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, (több osztállyal).

Osztály és típus/2

- Példaként vegyünk egy olyan komplex objektumot, amiben valós és képzetes résszel tárolunk, és vegyünk egy másikat polárkoordinátákkal.
- A kétfajta komplex megvalósítás osztály szinten különböző, de típusuk – viselkedésük – interfész szinten azonos.
- Hagyományos nyelveken a típus értékhalmazt jelöl.

Modellezés objektumokkal

- Különböző szempontok szerint modellezünk.
- Objektummodell
  - Adat szempontjából írja le a rendszer statikus tulajdonságait (osztály v. entitás-relációs diagram).
- Dinamikus modell
  - A működés időbeliségét rögzíti (állapotgráf, kommunikációs diagram).
- Funkcionális modell
  - Funkció szerint ír le (adatfolyam-ábra).

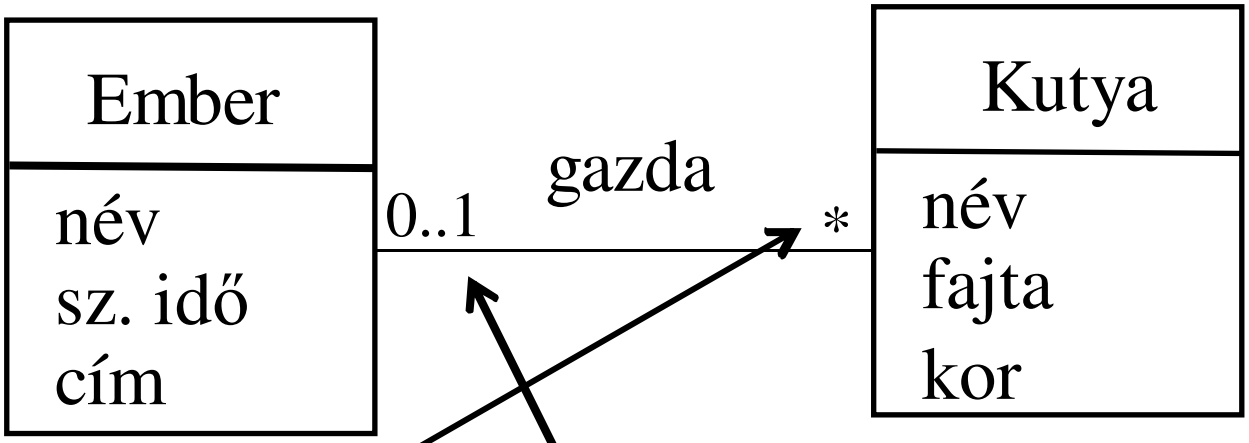
Modellezés eszközei, módszertana

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

Objektummodell

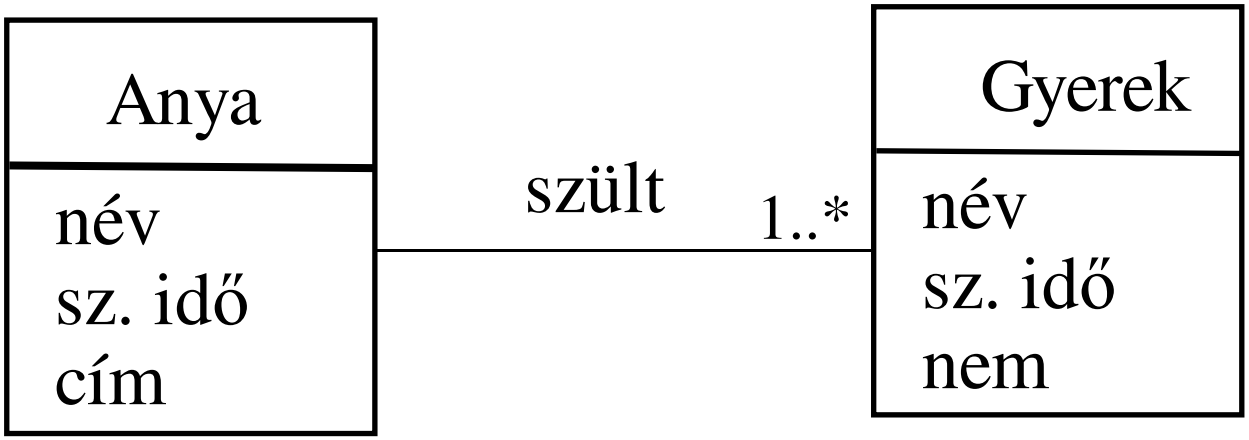
- Attribútumok leírása
  - Elnevezés típusú attribútumok. Nem vagy ritkán változnak (név, személyi szám, nem)
  - Leíró attribútumok (jövedelem, kor)
  - Referenciák. Kimutatnak az objektumból (cím).
- Kapcsolatok (relációk) leírása
  - láncolás – objektum példányok között
  - asszociáció – osztályok közötti kapcsolat
- Öröklés leírása

Példák a kapcsolatok leírására



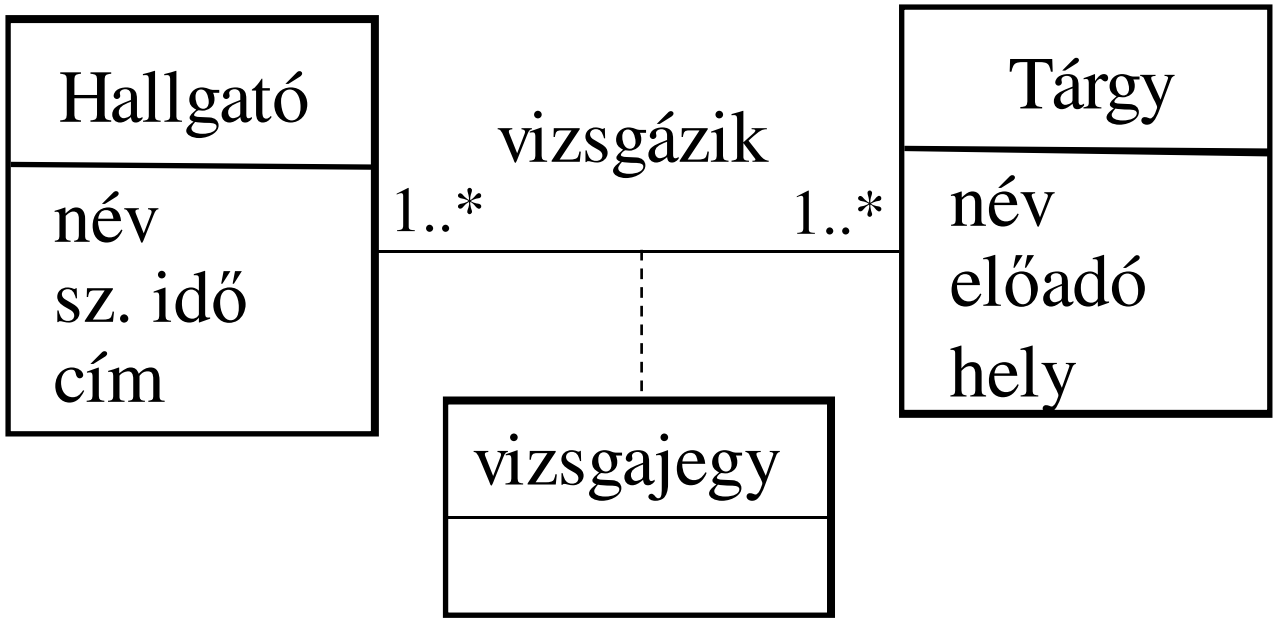
Egy ember 0 vagy több kutyának lehet gazdája.
Egy kutyának legfeljebb egy gazdája van, de lehet, hogy gazdátlan.

Egy – több kapcsolat



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

Kapcsolatok attribútumai



Egy tárgyból többen is vizsgázhatnak.
Egy hallgató több tárgyból 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 öröklésssel!

Komponens vs Agregáció

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 át is definiálhatók.

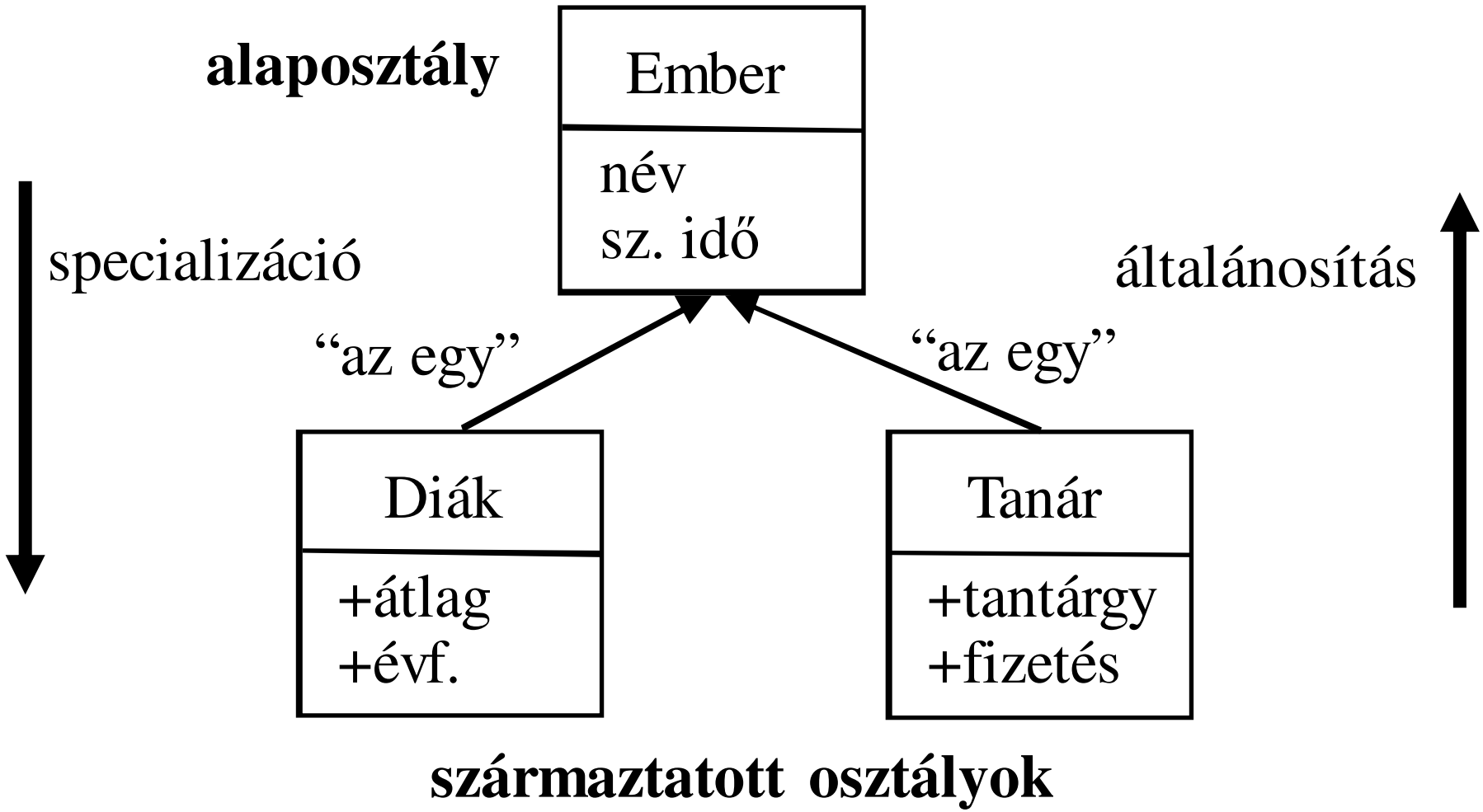
Feladat

- Diákokból, tanárokból álló rendszert szeretnénk modellezni.
  - Diák attribútumai:

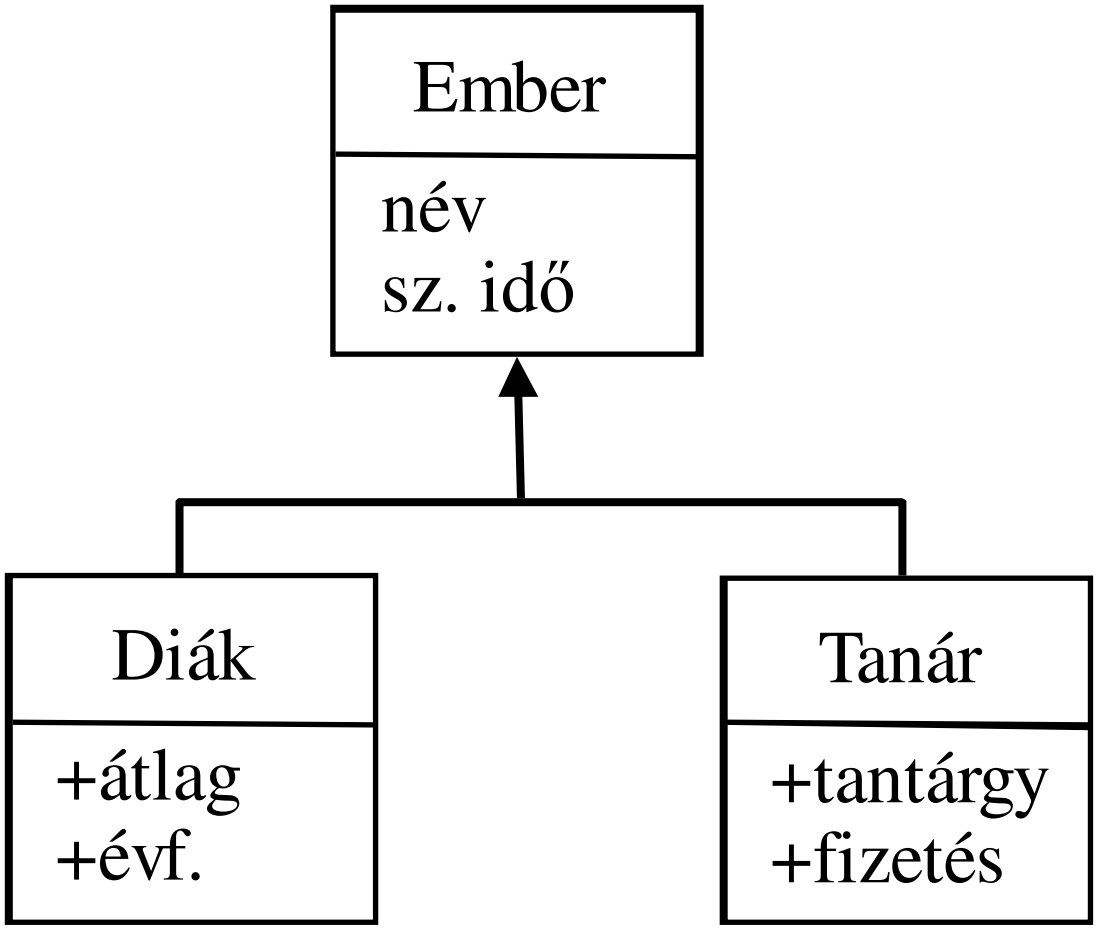
    név, sz. idő, átlag, évfolyam
  - Tanár attribútumai:

    név, sz. idő, tantárgy, fizetés
- Milyen osztályokat hozzunk létre ?
- 2 független osztály ?
  - név, sz. idő 2x, műveletek 2x, nehezen módosítható

Öröklléssel



Öröklés másként jelölve

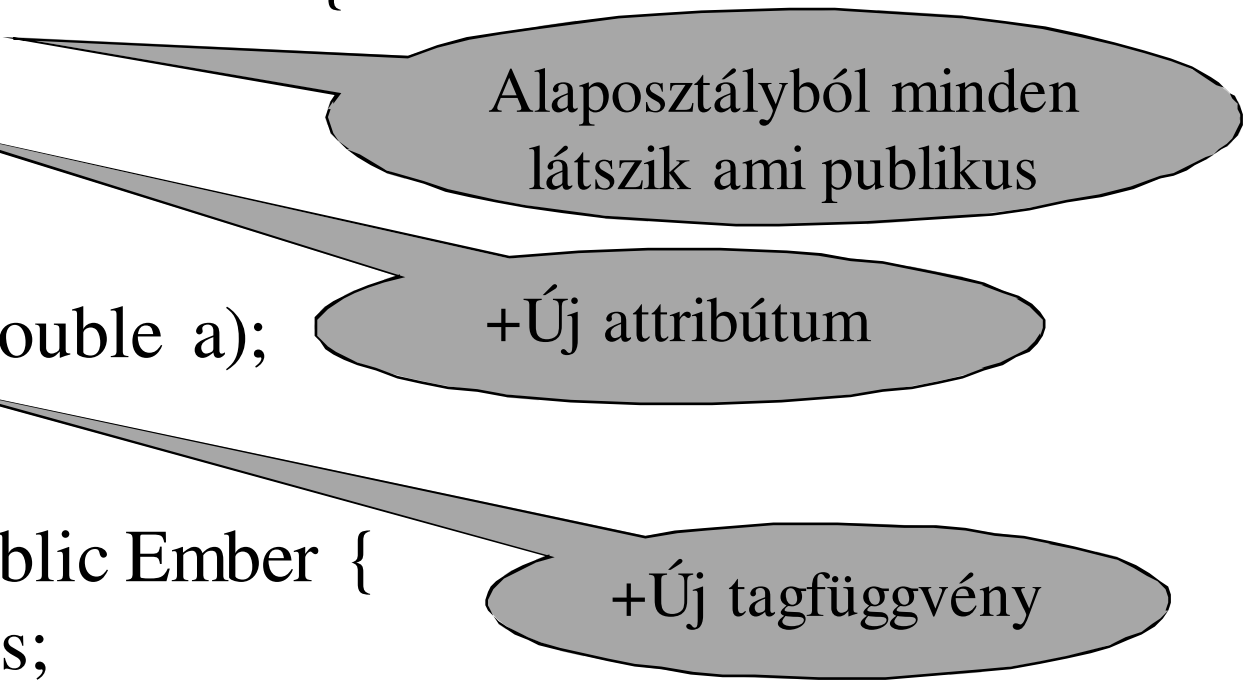


C++ jelölés

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

C++ jelölés/2

```
class Diak :public Ember {
   double atlag;
public:
   Diak();
   void setAv(double a);
   ....
};
class Tanar :public Ember {
   double fizetes;
public:
   Tanar(); ....
};
```



Öröklés előnyei

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

Öröklés fajtái

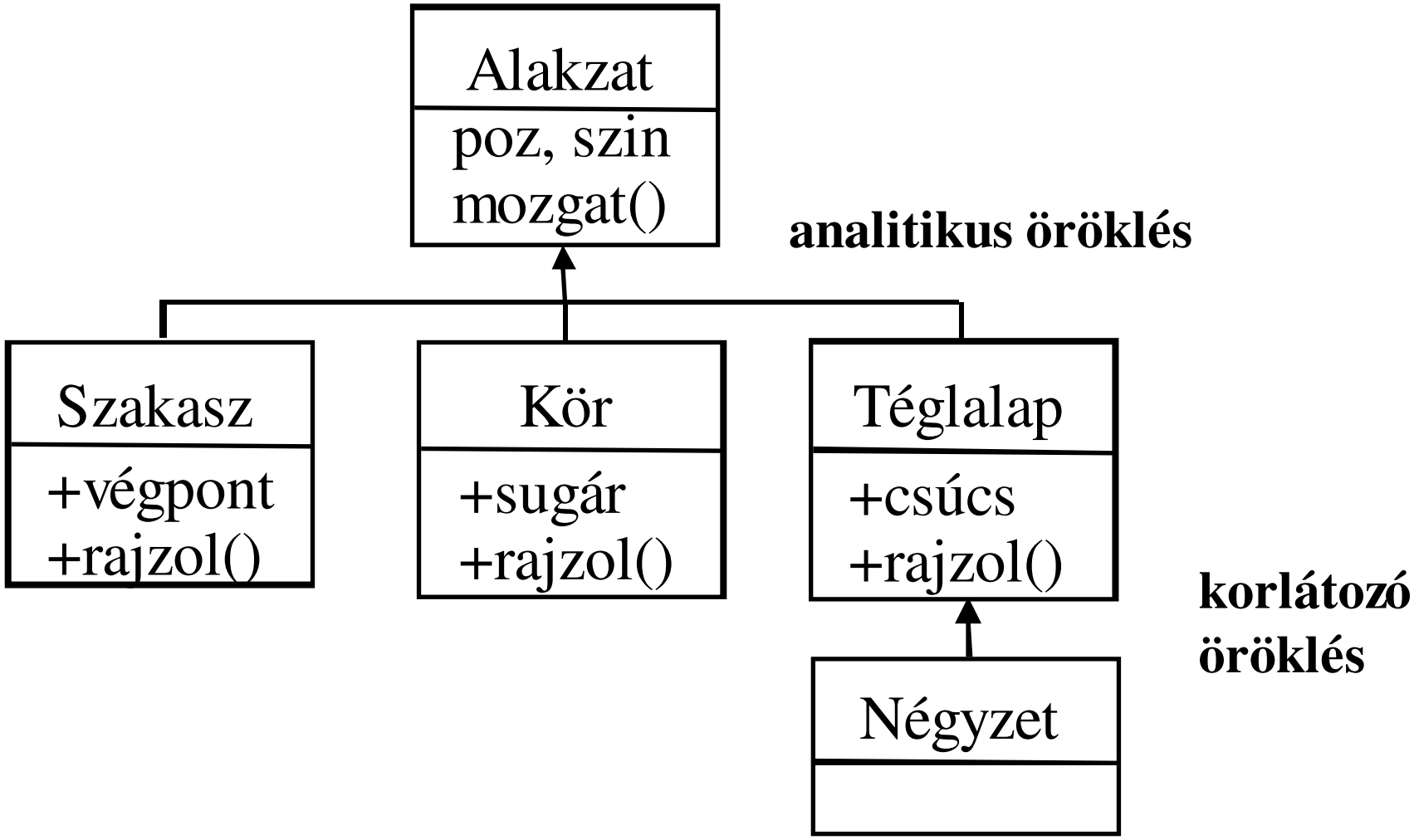
I.

- Analitikus
- Korlátozó

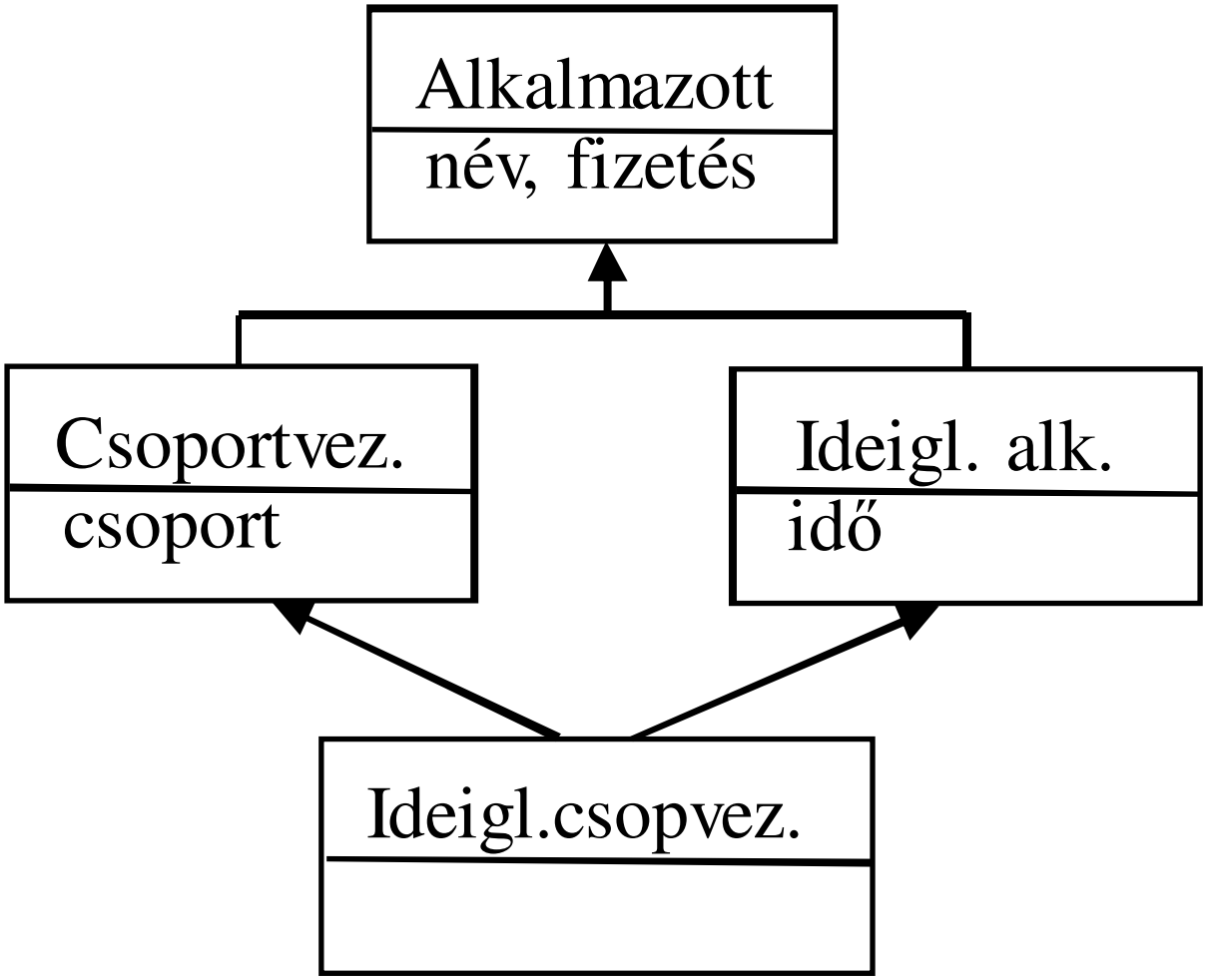
II.

- Egyszerű
- Többszörös

Analitikus és korlátozó öröklés



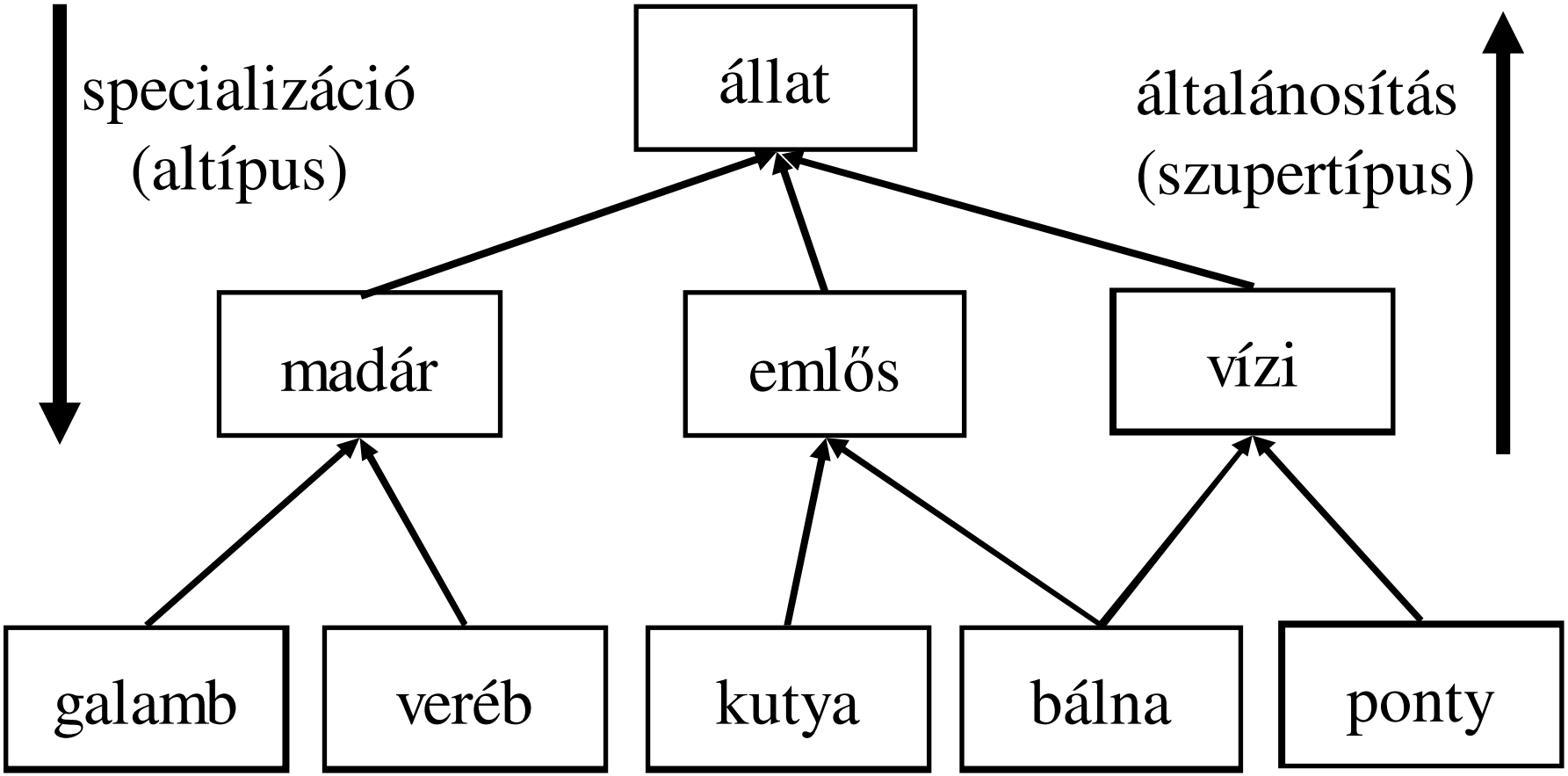
Többszörös öröklés



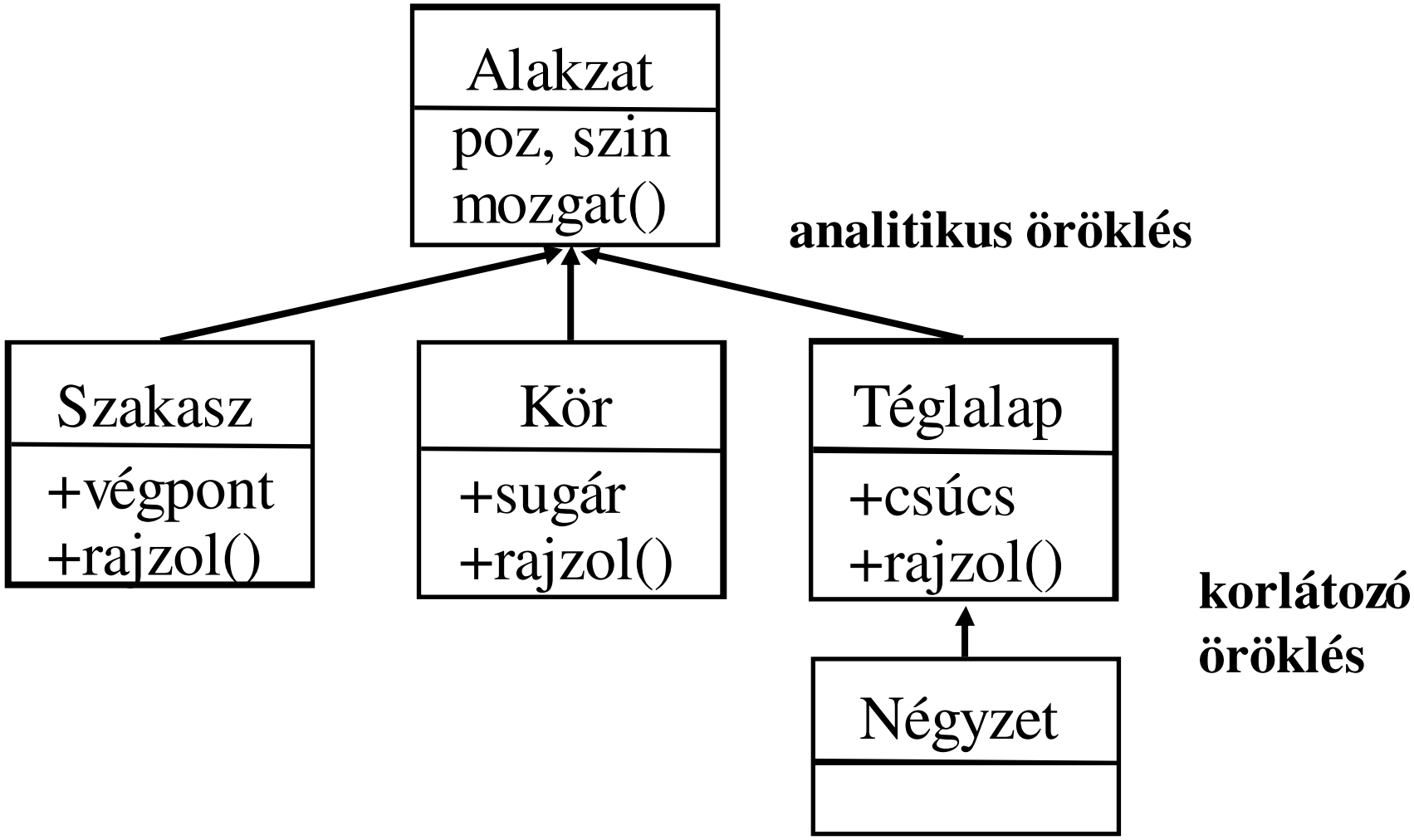
Kompatibilitás és öröklés

- A típusú objektum kompatibils B-vel, ha A típusú objektum bárhol és bármikor alkalmazható, ahol B használata megengedett.
- A reláció reflektív és tranzitív, de nem szimmetrikus.
- A kompatibilitás egy hierarchiát szab meg
  - pl: állat ← madár ← veréb

Kompatibilitás/2



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(sz) { }
// mozgat(), érezzük, hogy itt a helye, de nem
// tudjuk hogyan kell rajzolni!
// Ezért próbáljuk a származtatottba tenni, ahol
// már ismert a rajzolás menete.
};
```

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);
};
```

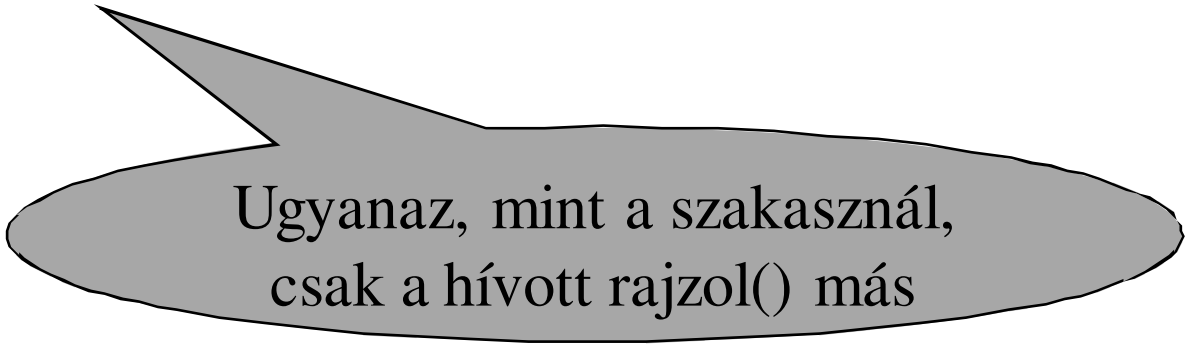
Szakasz tagfüggvényei

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

void Szakasz :: Mozgat( int dx, int dy ) {
   int sz = szin;          // tényleges rajzolási szín elmentése
   szin = BACKGND;// rajzolási szín legyen a háttér színe
   rajzol( );              // A vonal letörlése 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
}
```

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);
};
```



mozgat() helye

- Származtatott osztályokban
  - látszólag ugyanaz a függvény minden alakzatban
  - csak az általa hívott rajzol() más

- Alaposztályban
  - ha a hívott rajzol()-t egy manó le tudná cserélni mindig a megfelelő származtatott rajzol()-ra, akkor működne → **virtuális függvény**

Alakzat osztály virtuális függvénnyel

```
class Alakzat {
protected:
    int x, y;
    int szin;
public:
   Alakzat(int x0, int y0, int sz)
                        :x(x0), y(y0), szin(sz) { }
   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 itt van.

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
   Szakasz 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;
   for (int i = 0; i < 2; i++ )  ap[i] ->rajzol();
}
```

Mikor melyik rajzol() ?

	Virtuális Alakzat:: rajzol()	Nem virtualis 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:

  Absztrakt alaposztály

Absztrakt alaposztályok

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

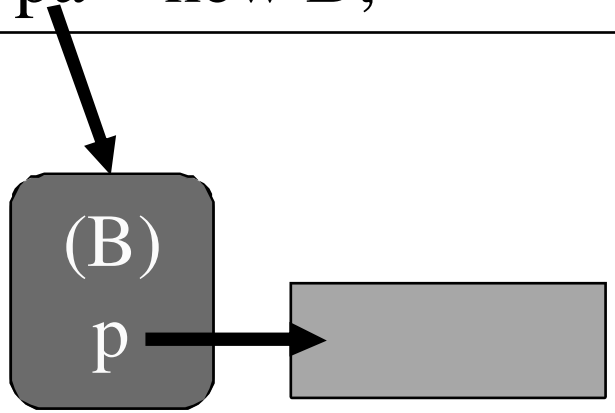
```
class Alakzat {
protected:     int x, y, szin;
public:
        Alakzat( int x0, int y0, int sz) ;
        void   mozgat( int dx, int dy );
        virtual void rajzol( ) = 0;  // tisztán virtuális
        virtual ~Alakzat() {};    // Ez meg mi ?
};
```

Nem példányosítható

Virtuális destruktor szerepe

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

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

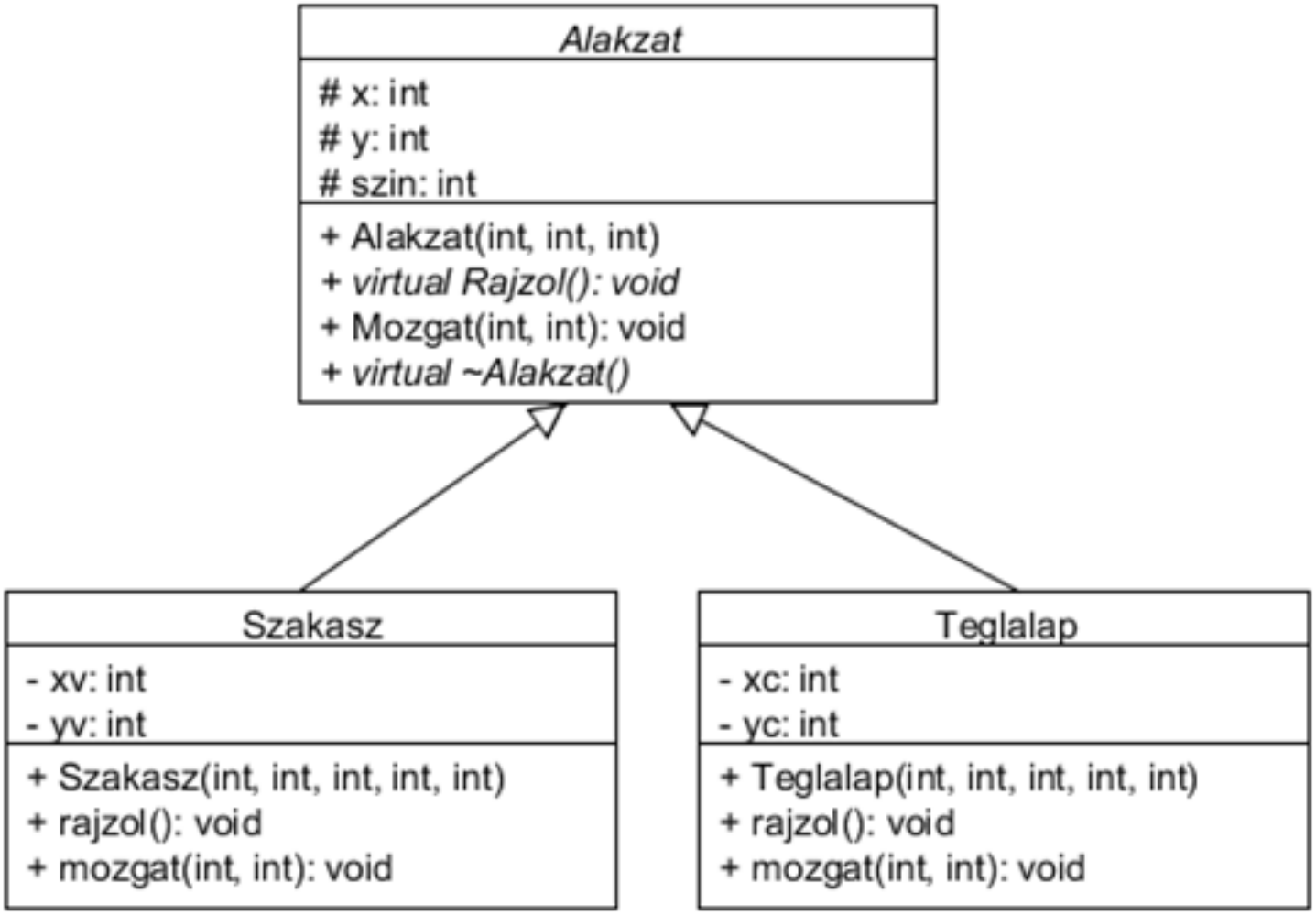



```
delete pa;
```

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

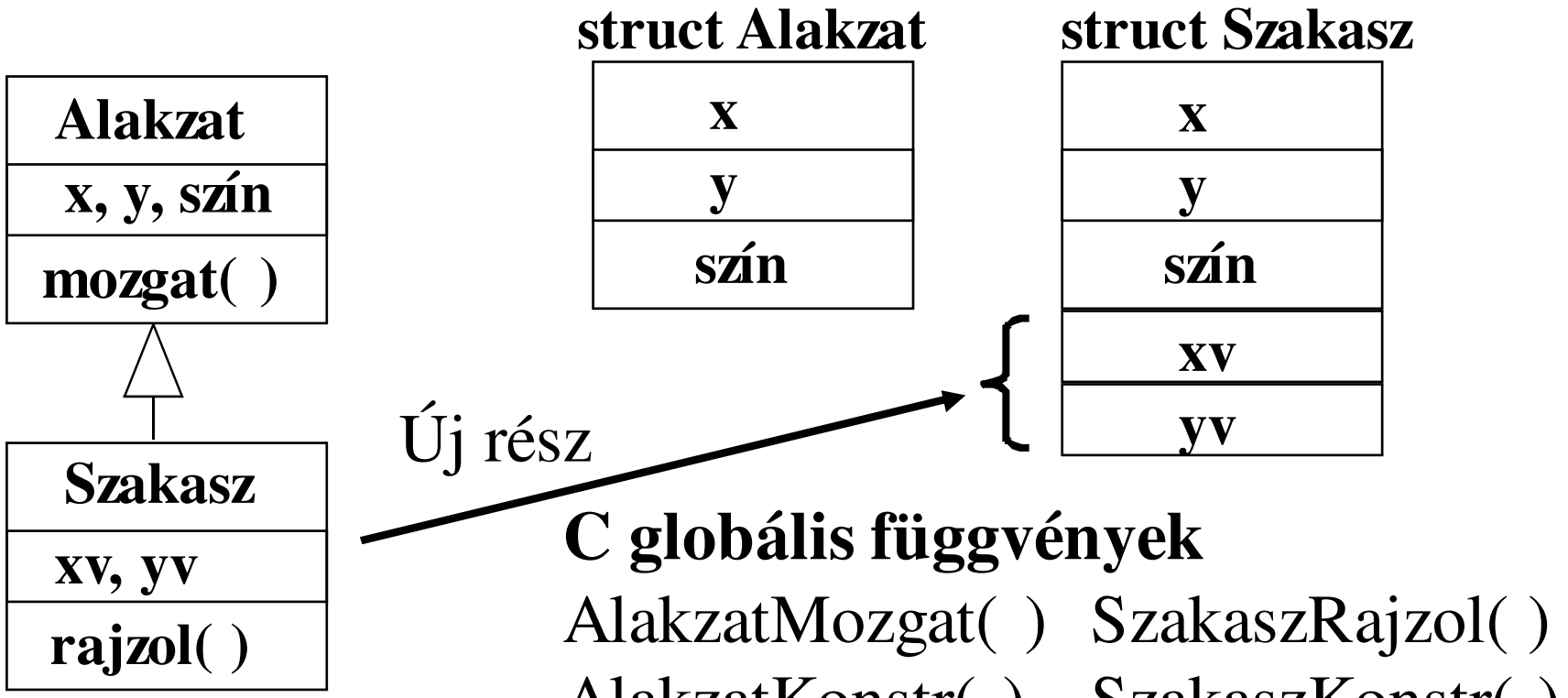
svn.iit.bme.hu/proga2/eloadas_peldak/ea_04/

Most itt tartunk

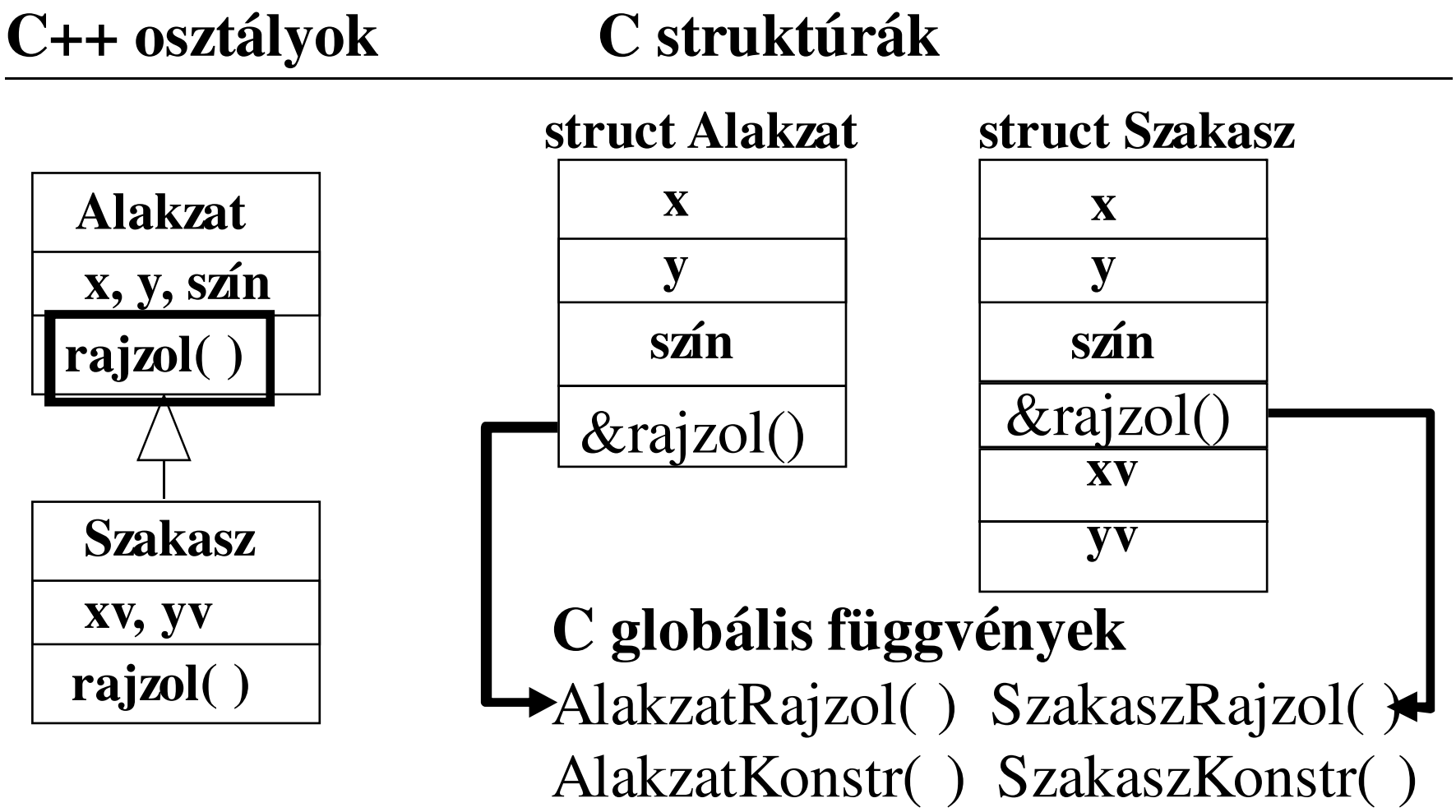


Öröklés impl., ha nincs virtuális fv.

C++ osztályok **C struktúrák**



Öröklés impl., ha a Rajzol() virtuális



Alakzat C implementációja

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

void AlakzatMozgat( struct Alakzat *this ) { }
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 ←→ általánosítás)
  - analitikus v. korlátozó
  - egyszerű v. többszörös
- C++ nyelvi eszköz:
  - analitikus → public, korlátozó → private
  - tagfüggvények átdefiniálása, protected mezők
  - virtuális tagfüggvény: alaposztály felől elérhető a származtatott osztály tagfüggvénye,
  - absztrakt alaposztály nem példányosítható

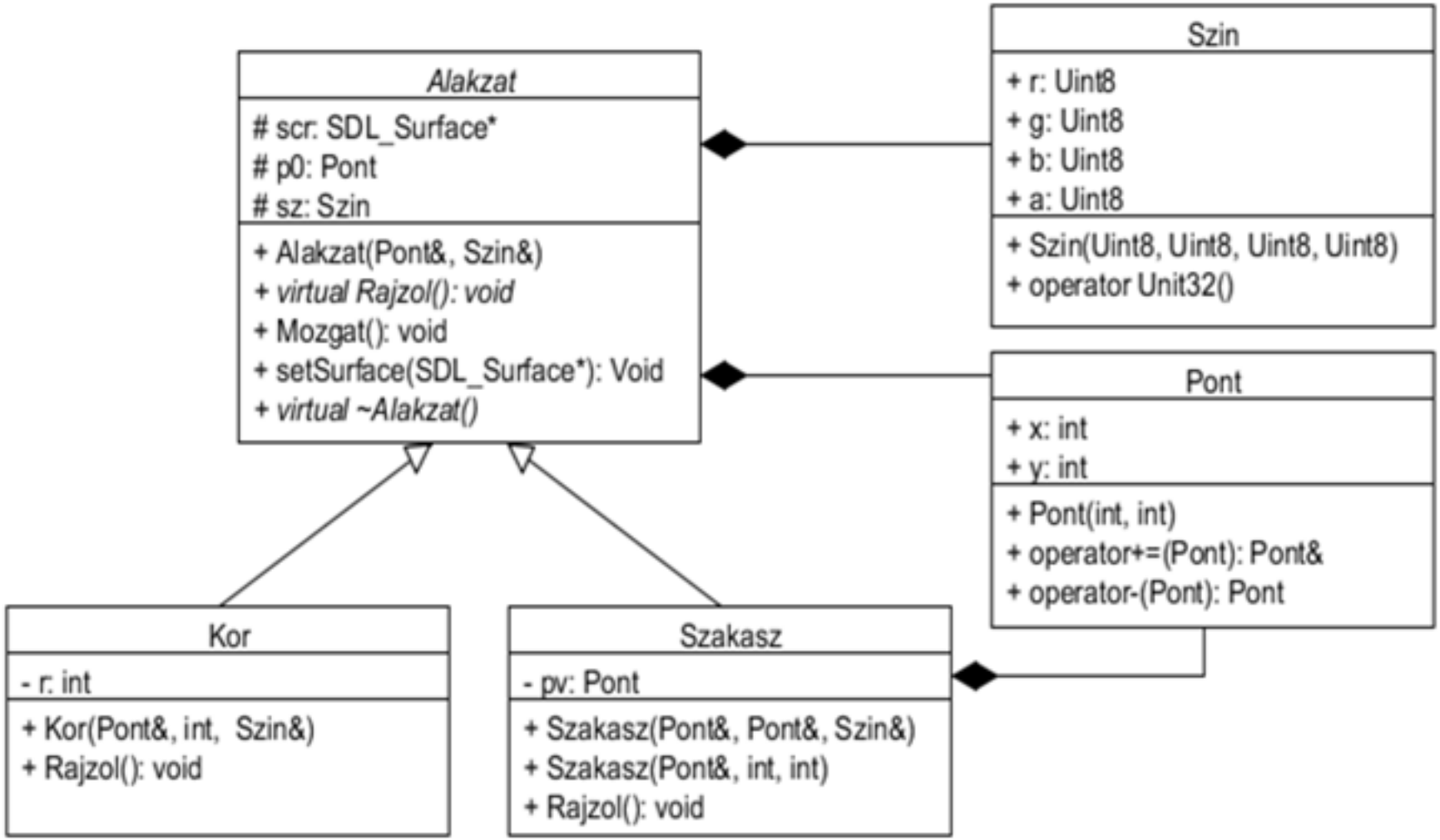
Védelem összefoglalása

	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.

Objektummodell kibővítve



Eltérések, kiegészítések

- Pont osztály bevezetése: rugalmasabb, könnyebben bővíthető (pl. 3D-re).
- Szin osztály: SDL-hez alkalmazkodik.
- Mindkettő teljesen publikus – úgysem hozunk belőlük létre önálló példányt, egyszerűbb a haszn.
- Alakzat osztályban statikus taggal rejtjük el az SDL egyik globális adatát.
- Rajzolás után mindig van SDL_Flip – nem a legjobb megoldás, de most nem a maximális felhasználói élmény elérése a cél.

Kiegészített alakzat

```
class Alakzat {
protected:
    Pont p0;                        // alakzat origója
    Szin sz;                        // alakzat színe
    static SDL_Surface *scr;   // eldugott ”globális”
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
};
```

svn.iit.bme.hu/proga2/eloadas_peldak/ea_04/