OOP

Tervezés, programozás
Angster Erzsébet: Objektumorientált tervezés és programozás
Andrew Troelsen: A C# 2008 és a .NET 3.5
Kathy Sierra, Bert Bates: Agyhullám: Java

Öröklés – C#, Java

- Osztályhierarchia diagram
- Egyszeres öröklés
- Többszörös öröklés
- Öröklés szabályai:
  - Egy osztályból több osztály is származhat
  - Osztályhierarchia tetszőleges, de…
  - Tranzitív: B -> A és C -> B, akkor C -> A
- Szabályok:
  - Egyszer öröklés
  - Implicit ős: Object

HengerProgram

- Mértani hengerek
- Tömör hengerek
- Csövek (Lyukas hengerek)

  - Sugár, magasság, fajsúly, falvastagság
  - Fajsúly nem kötelező, alapértelmezett: 1
  - Minden henger adja vissza alapadatait, térfogatát, esetleges súlyát
  - toString az adott henger szöveges reprezentációja
  - Program futása közben született hengerek száma
  - Az összes henger típus átlagtérfogata
  - Csövek összes súlya

Szükséges fogalmak

- Öröklődés: Konstruktor hívási lánc
- Virtuális függvények
- Polimorfizmus
- Típus kényszerítés

Konstruktorhívási lánc

- A megkonstruáláskor az öröklési ág minden osztályának legalább 1 konstruktorát meg kell hívni, egészen Object-ig.
- Minden kstr. -ban kell legyen:
  - this a saját konstruktorok hívására
  - Vagy super | base az ős hívására
- A fenti kettő kizárja egymást, ha nincs, akkor a fordító betesz : super() | base(); //ez legyen az ősben
- A hívás előtt (java) nem lehet semmi, adatdekr. sem

Konstruktorhívás

- Az ősadatokat az ős konstruktor hívásával inicializáljuk
- Minden kstr. a saját dolgát végezze

```
class C3{
public C3(int a, int b){ }
C3(int a) : this(a, 3) { ; }
```

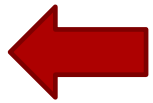
```
class C4 : C3{
C4() : base(){ ; }
C4(){  }
C4(){ base(5); }
C4() : base(5){ ; }
C4() : base(){ this(); }
```

Mi a hiba?
Miért?

C3 ban nincs C3()
C3 ban nincs C3()
base(5) nem jó helyen
Nem publikus
Kettő nem lehet

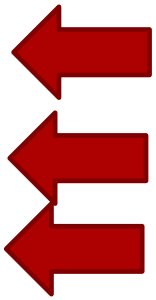
Konstruktorhívás

```
class C1{ C1(int a) {} }

class C2 : C1{
C2(int a, int b) : base(a){}
C2(int a): this(a, 1) { }
C2() : this(2) { }

class C3 : C2{
C3(int a, int b){ }
C3(int a) : this(a, 3) { }

public class KstrHivas{
   Psv Main(){ new C3(1)}
 }
```



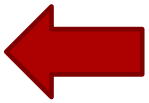
base()

Object konstruktorának hívása



Mi a sorrend?

base()



Virtuális függvények

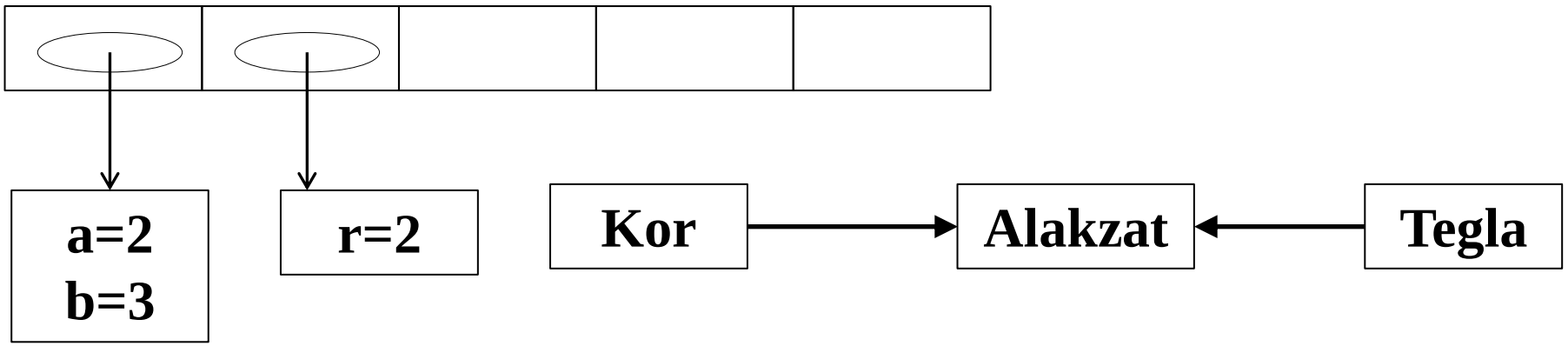
```
toString(): String
```

- Java-ban minden függvény implicit virtuális
  - `@Override`
- C#-ban explicit kell a `virtual` kulcsszó, majd
  - `new`
  - `override`

Polimorfizmus

```
Alakzat[] alakzatok = new Alakzat[5];
alakzatok[0] = new Tegla(2,3);
alakzatok[1] = new Kor(2);
```

- Egy objektum többfélének látszódhat
- Többfajta egyfélének tűnhet
- *Függvény felülírás*



Típus kényszerítés - cast

```
int szam = (int)(Math.random()*(f-a+1))+a;
```

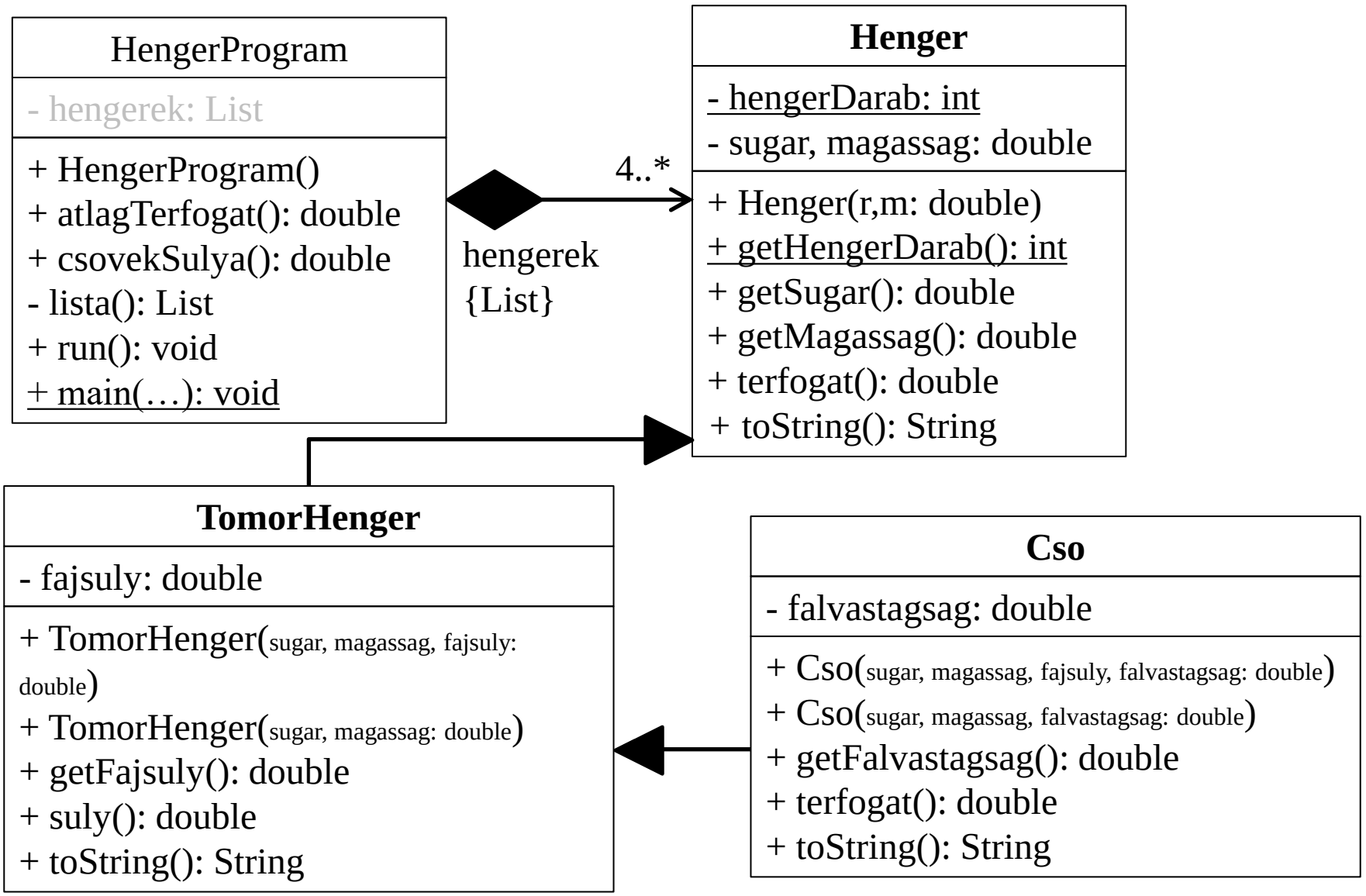
HengerProgram

- Mértani hengerek
- Tömör hengerek
- Csövek (Lyukas hengerek)

UML
Implementálás

- Sugár, magasság, fajsúly, falvastagság
- Fajsúly nem kötelező, alapértelmezett: 1
- Minden henger adja vissza alapadatait, térfogatát, esetleges súlyát
- toString az adott henger szöveges reprezentációja
- Program futása közben született hengerek száma
- Az összes henger típus átlagtérfogata
- Csövek összes súlya

UML


HengerProgram
- hengerek: List
+ HengerProgram()
+ atlagTerfogat(): double
+ csovekSulya(): double
- lista(): List
+ run(): void
+ main(…): void
4..*
hengerek
{List}
Henger
- hengerDarab: int
- sugar, magassag: double
+ Henger(r,m: double)
+ getHengerDarab(): int
+ getSugar(): double
+ getMagassag(): double
+ terfogat(): double
+ toString(): String
TomorHenger
- fajsuly: double
+ TomorHenger(sugar, magassag, fajsuly: double)
+ TomorHenger(sugar, magassag: double)
+ getFajsuly(): double
+ suly(): double
+ toString(): String
Cso
- falvastagsag: double
+ Cso(sugar, magassag, fajsuly, falvastagsag: double)
+ Cso(sugar, magassag, falvastagsag: double)
+ getFalvastagsag(): double
+ terfogat(): double
+ toString(): String