# Programozási nyelvek Java

*Kozsik Tamás előadása alapján*
*Készítette: Nagy Krisztián*

## 6. előadás

**Objektum-orientált szemlélet**

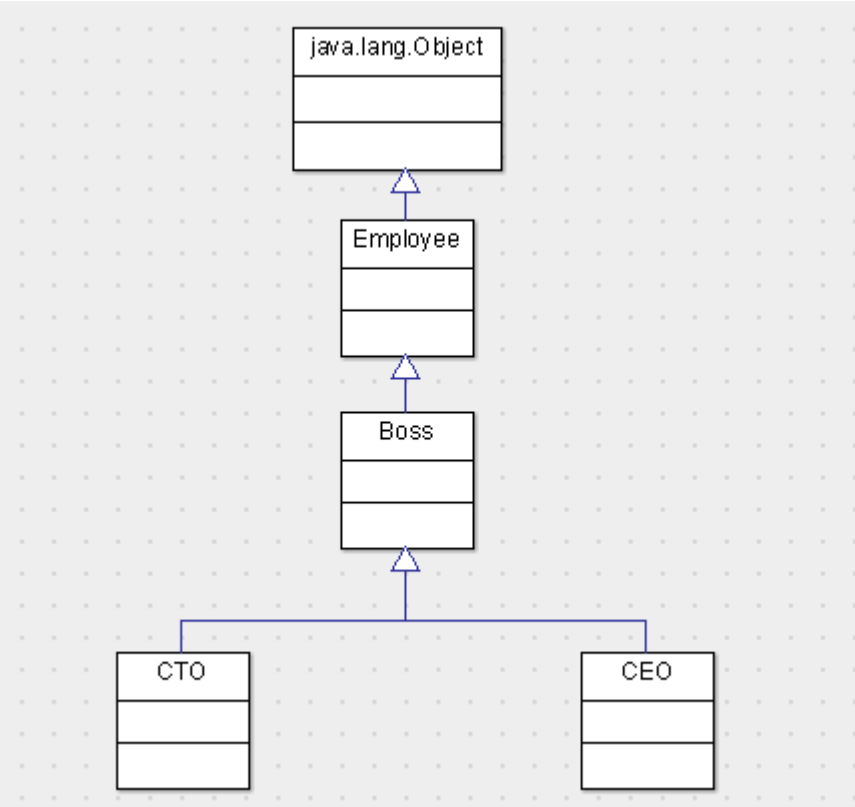
- Egységbe zárás (incapsulation)
- Információ elrejtés
- Öröklődés
  altípusosság
  dinamikus kötés

**Öröklődés**

- kiterjesztem, kibővítem, megváltoztatom
- extends kulcsszó

```
class Employee {…}
class Boss extends Employee {…}
```
→ különbség az Employee-hoz képest

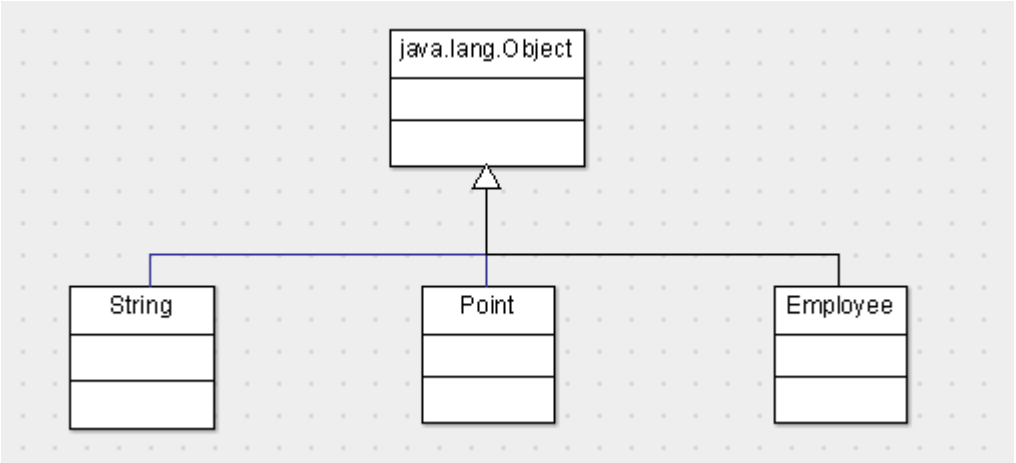
A különbséget kell programozni.

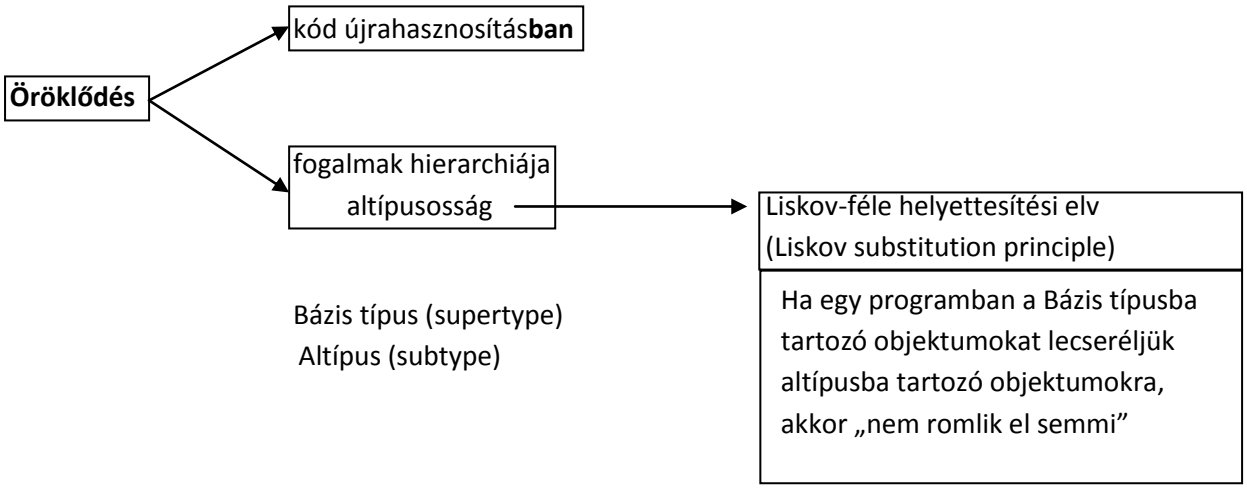



Parciális rendezési reláció (Reflexív, Antiszimmetrikus, Tranzitív)

Kört nem tartalmazhat.
Egy szülő osztálynak lehet több gyermek osztálya, de JAVA-ban többszörös öröklődés nincsen.
(Azaz egy gyermeknek nem lehet kettő szülője)

Minden nem származtatott osztály a java.lang.Object –ből származik.




**Polimorfizmusok (”több alakúság”)**

- Altípusos polimorfizmus (”több típussal való együttműködés”)
  - bázis típusok**hoz** megírt kódot használjuk altípusbeli értékekre
- Egyebek… (több féle polimorfizmus létezik)

**Felüldefiniálás (Override / Redefine)**

Egy megörökölt művelet implementációját lecseréljük (JAVA-ban)

hatékonysági szempontból
(Pl.: Téglalap kerülete, Négyzet kerülete)

Más eredmény (teljesen más vagy kiegészítés)
(Pl.: Dolgozók fizetése – Főnök fizetése)

```
class Employee{
      int salary(){…}
}

class Boss extends Emp{
      // @Override – annotáció, opcionális a kiírása, de jó ha van
      @Override
      int salary(){[Emp salary-től eltérő]}
}
```

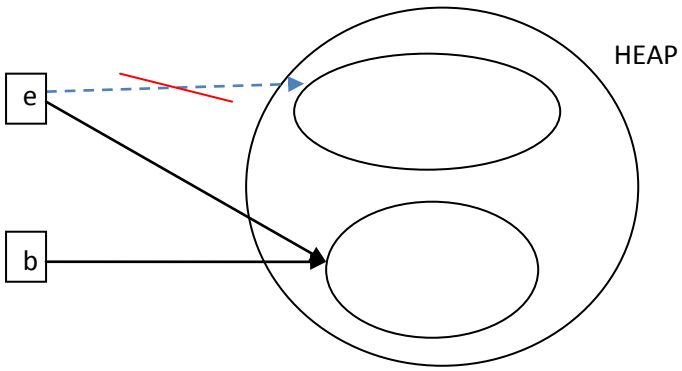
super.salary() + supplement()

bázis osztálybeli megfelelője

Boss osztályban:
- this.salary()  Boss-os salary()
- super.salary()  Emp-es salary()

```
Emp e = new Emp();
Boss b = new Boss();
e.salary();      // Emp-es salary()
b.salary();      // Boss-os salary
e = b;
```

(kompatibilitás (altípusra)) – Legális!
Boss  Emp
e.salary()  Boss salary lesz (felüldefiniált változat)  [DINAMIKUS KÖTÉS]



**Dinamikus kötés (Késői kötés [Late binding])**

Az objektum létrehozó osztályához felülről legközelebb álló implementáció végrehajtása a felüldefiniált implementációk közül.

```
Emp e = new Emp();
Boss b = new Boss();
Emp s = new Boss();
Object o = new Boss();
```

Helytelen: String str = new Boss(); , mivel a String nem bázis típusa a Bossnak.

**Statikus típus**: referencia deklarált típusa
**Dinamikus típus**: a hivatkozott objektum osztálya

A statikus típus bázis típusa a dinamikus típusnak, esetleg meg is egyeznek.

**Dinamikus típus szerint** Emp ⊇ Boss

Statikus típus

- nem változik
- fordítási időben (static time) ismert

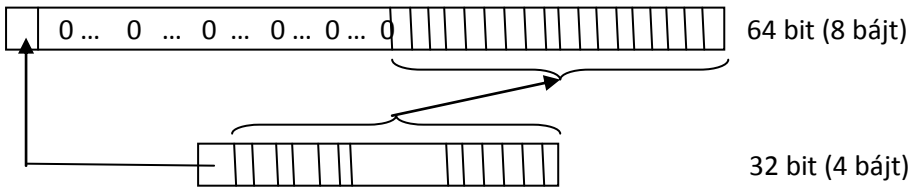
Dinamikus típus

- változékony (pl.: értékadás)
- csak futási időben (dinamic time) ismert

dinamikus kötés : runtime overhead (Kevésbé hatékony az OOP Java, mint a procedurális C)
Hatékonyság: C > C++ > JAVA

**Primitív típusok kompatibilitása**

long l = 13L; - 8 bájt: kettes komplemens
int i = 13; - 4 bájt: kettes komplemens
l = i; e = 12; - utasításokat elfogadja, pedig típus hibás.
Ebben az esetben egy adatkonverzió és egy értékadás is történik




long : $\mathbb{Z}|_{-2^{31}\ldots 2^{31}-1}$ ⊇ int : $\mathbb{Z}|_{-2^{15}\ldots 2^{15}-1}$
Kompatibilitás JAVAban (<:)

bool <: short <: int <: long <: float <: double

Megjegyzés: Tisztább az osztályok közötti, mint a primitív típusok közötti.

**Konstruktorok**

- Minden osztályt önmagában kell vizsgálni ilyen szemszögből
- Nem öröklődik

```
class Emp {…}                    ←——— default
class Boss extends Emp {…}       ←——— default
```

```
class Emp{
      Emp(String name){…}
}
class Boss extends Emp {…}
```

Ebben az esetben egy Boss(){ super();} konstruktor hívódik meg, ami fordítási hibát okoz! (Emp-ben nincs paraméter nélküli konstruktor!)

Javítása:

```
class Boss extends Emp{
      Boss(String name){
            super(name);
      }
}
```

Az alosztály-beli konstruktorból egy bázisosztály-beli konstruktort hívunk meg kivéve, ha van a bázisosztályban paraméter nélküli konstruktor. (Utóbbinál ugyan is a bázisosztályban található paraméter nélküli konstruktor fog meghívódni, amennyiben az alosztályban nem definiáltunk konstruktort.)