Programozási Ismeretek

Dr. Gergely Tamás
Dr. Ferenc Rudolf, Dr. Jász Judit, Dr. Kiss Ákos

Szegedi Tudományegyetem
Informatikai Intézet
Szoftverfejlesztés Tanszék

2024

(v0214)

Tartalom

Tartalom

Vizuális modellezés

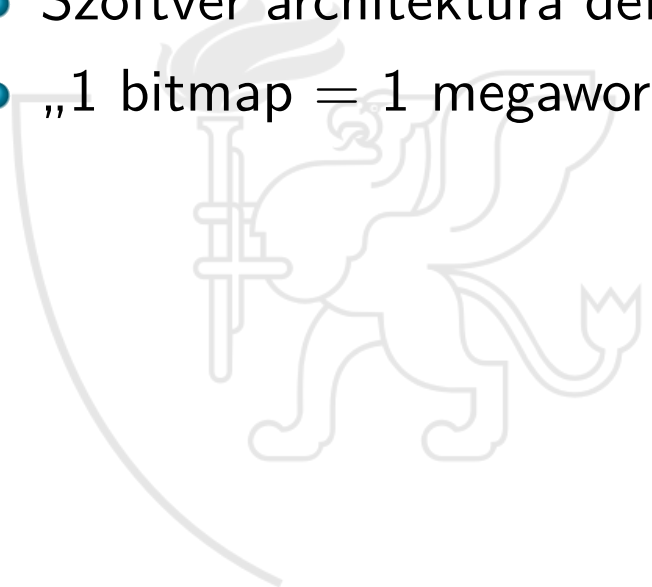
Szoftverfejlesztés folyamata

- A programozás fázisai
  - **Követelmény-specifikáció**
  - **Tervezés**
  - Megvalósítás
  - Ellenőrzés
  - Fenntartás
  - *Folyamatos dokumentáció*
- A követelmény-specifikáció és tervezés fázisokban modellezünk
- Modellezés
  - A probléma leírása a valós világból vett ötletekkel
  - A rendszernek a probléma szempontjából lényeges részeit vizsgálhatjuk
  - Vizuális: szabványos grafikai eszközökkel

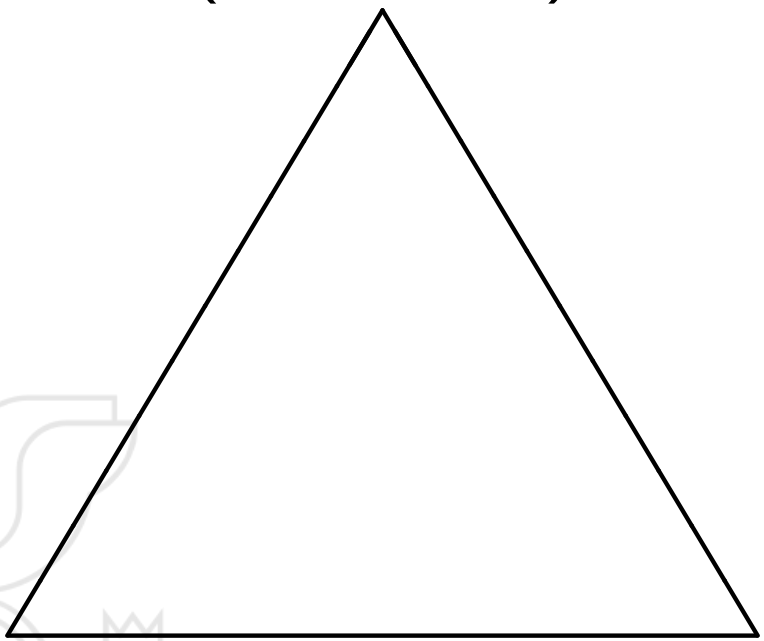
Vizuális modellezés

Miért modellezünk?

- Alkalmas üzleti folyamatok leírására
- Esettanulmányok a felhasználó szempontjából
- Kommunikációs eszköz
- Komplexitás kezelése
- Fejlesztési idő és rizikó csökkentése
- Szoftver architektúra definiálása
- „1 bitmap = 1 megaword” (ismeretlen szerző)



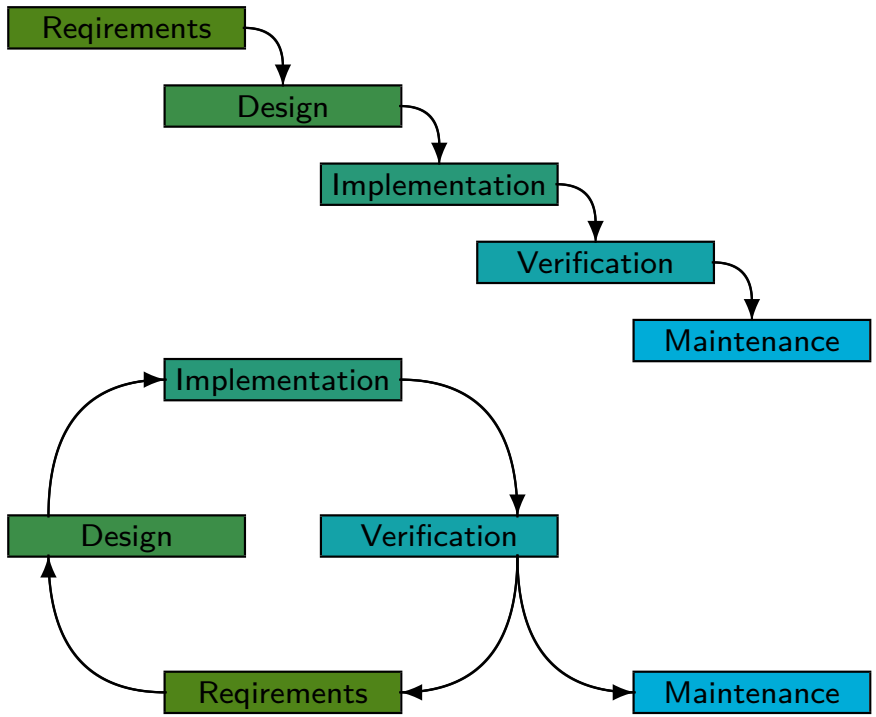
A „siker háromszöge”



A „siker háromszöge”

1. Folyamat

- Fejlesztési életciklust irányítja (folyamat leírása)
- Végrehajtandó lépések és a végrehajtás sorrendje
- Általában iteratív, inkrementális életciklust ír elő
- Minden iterációban egy mini vízesés modell:
  - követelmények gyűjtése
  - analízis
  - tervezés
  - implementáció
  - tesztelés
- Példák
  - Agile: Scrum, Kanban, eXtreme Programming (XP)
  - Iterative: Rational Unified Process (RUP), Rapid Application Development (RAD)



A „siker háromszöge”

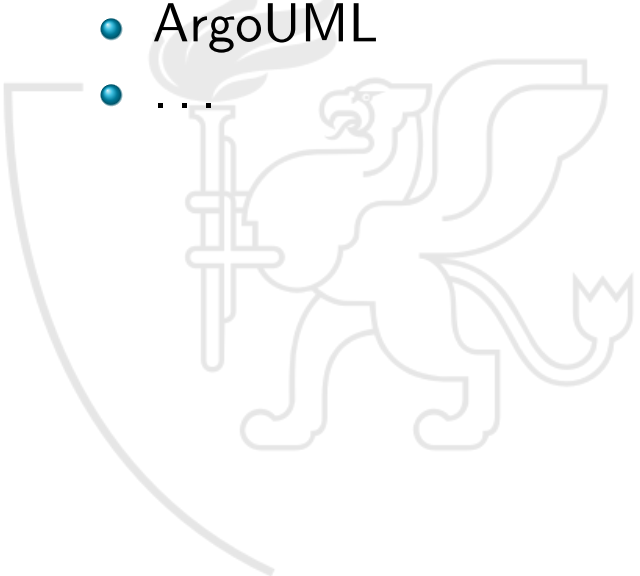
2. Jelölésrendszer: UML

- Modellezéskor a rendszer architektúráját különböző nézetekkel írjuk le
  - logikai, komponens (implementációs), folyamat, telepítési (feladat-kiosztási)
- Használati eset nézet (esettanulmány)
  - az egészet összefogja és közös kommunikációs platformot létesít a résztvevő felek között
- Nem minden rendszer igényli az összes nézetet
- Új nézeteket is lehet definiálni
  - pl. adat, biztonság

A „siker háromszöge”

3. Eszköz

- Ősi eszköz: papír és ceruza
- Nagyon sok UML eszköz van: egyszerű rajzoló programtól kifinomult objektum-modellező eszközig. Pl.
  - Rational Rose: teljes vizuális modellezés, kliens/szerver, osztott, valósidejű rendszerekhez
  - Borland Together ControlCenter
  - Microsoft Visio
  - ArgoUML
  - ...



Tartalom

UML

- Unified Modeling Language
  (Egységesített Modellező Nyelv)
- Egy nyelv: szintaktikai és szemantikai szabályok összessége
- Szoftverrendszer elemeinek
  - vizualizálására
  - specifikálására
  - létrehozására
  - dokumentálására
- Teljes UML dokumentáció
  - `http://www.uml.org`

A UML előnyei

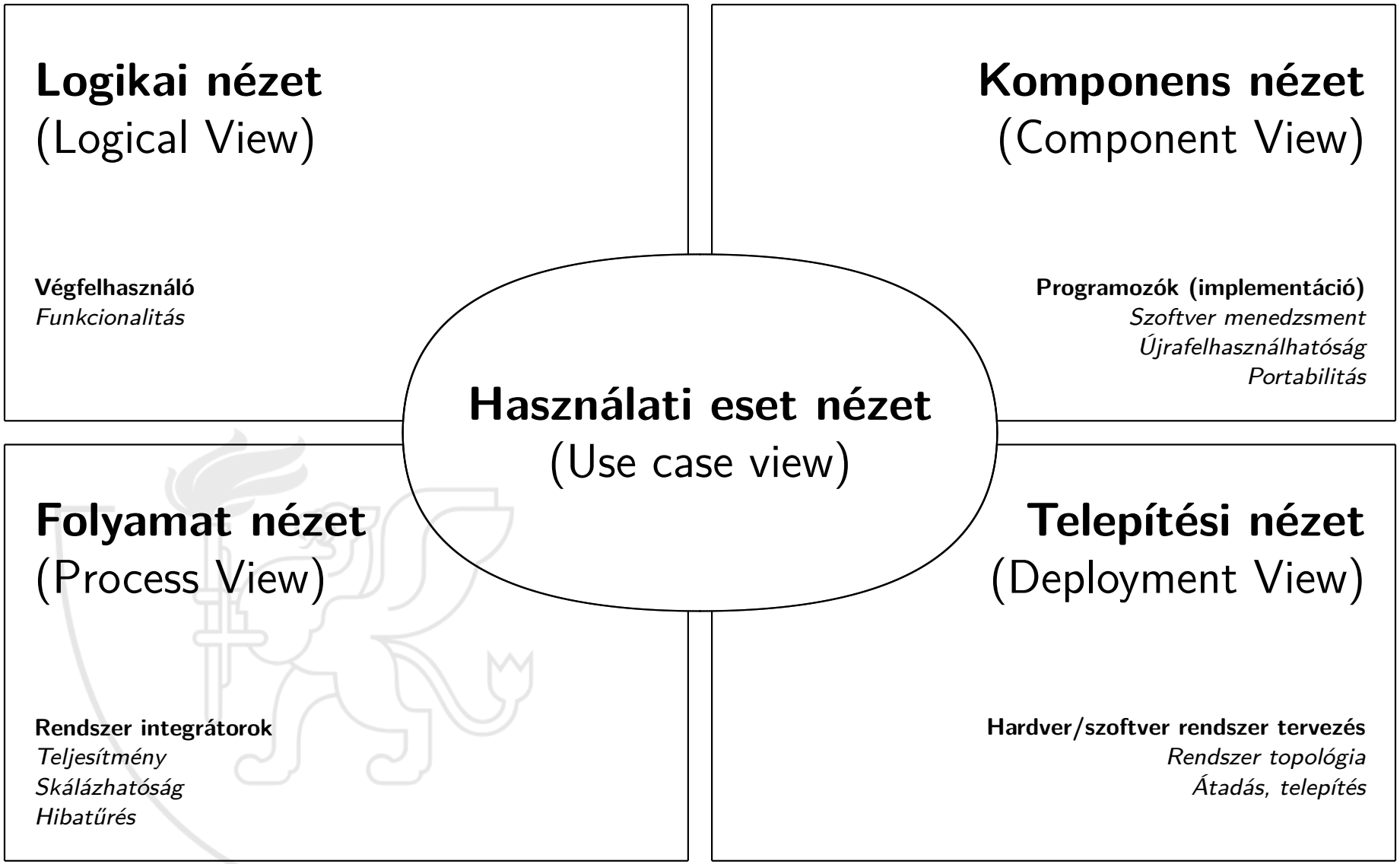
- Nyílt szabvány (Object Management Group – OMG által)
- Teljes szoftverfejlesztési életciklust támogatja
- Különböző alkalmazási területekre alkalmazható
- Hatalmas tapasztalati tudásra épít
- Sok eszköz támogatja
  - Rose, MSVC, Visio, GDPro, Together, dia, Eclipse, NetBeans, Visual Paradigm, ...
  - `https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools`
- Támogatók:
  - Rational, HP, IBM, Microsoft, Oracle, Platinum, TI, Sun, DEC, Compaq, ...

Tartalom

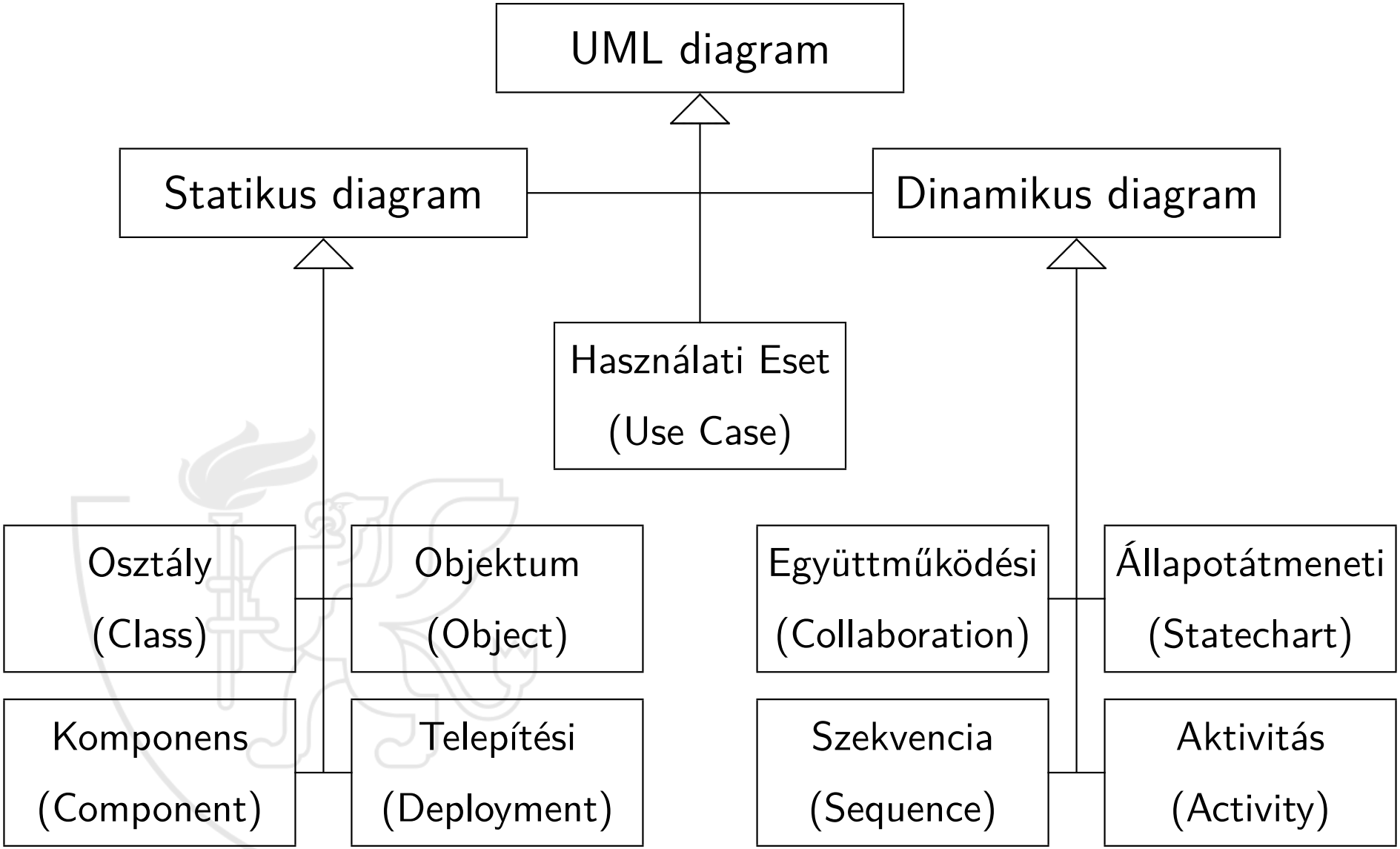
Modell, nézet és diagram

- **Modell**: a rendszer teljes leírása
  - „adatbázis”
- **Nézet**: a modell különböző vetületei
  - valamely résztvevő szemszögéből (megrendelő, fejlesztő, üzemeltető, ...)
  - részleges reprezentációja a rendszernek
  - szemantikailag konzisztens a többi nézettel
- **Diagram**: a nézet ábrázolása
  - egy nézet többféle diagrammal is ábrázolható
  - részleges reprezentációja a rendszernek
  - szemantikailag konzisztensek egymással

Az architektúra 4+1 nézete



Diagramok


UML diagram
Statikus diagram
Dinamikus diagram
Használati Eset
(Use Case)
Osztály
(Class)
Objektum
(Object)
Együttműködési
(Collaboration)
Állapotátmeneti
(Statechart)
Komponens
(Component)
Telepítési
(Deployment)
Szekvencia
(Sequence)
Aktivitás
(Activity)

Objektum- és osztálydiagram

- Objektumok és osztályok vizuális reprezentációja
- Objektum orientált tervezéshez nélkülözhetetlen
- Jó kommunikációs eszköz

Tartalom

Objektum fogalma

- Egy (valós világból vagy elvonatkoztatott) entitás ábrázolása, pl.:
  - személy adatai
  - poligon
  - kurzus
- Minden objektum rendelkezik:
  - állapottal
  - viselkedéssel
  - identitással

Objektum állapota

- Egy a lehetséges létezési lehetőségek közül
- Időben változó
- *Attribútumok* határozzák meg
- Példa:
  - egy *Kurzus* típusú objektum állapota nyitott, ha kevesebb, mint 20 jelentkező van, egyébként zárt

programozas: Kurzus
nyitott: bool = true

algoritmusok: Kurzus
nyitott: bool = false

Objektum állapota

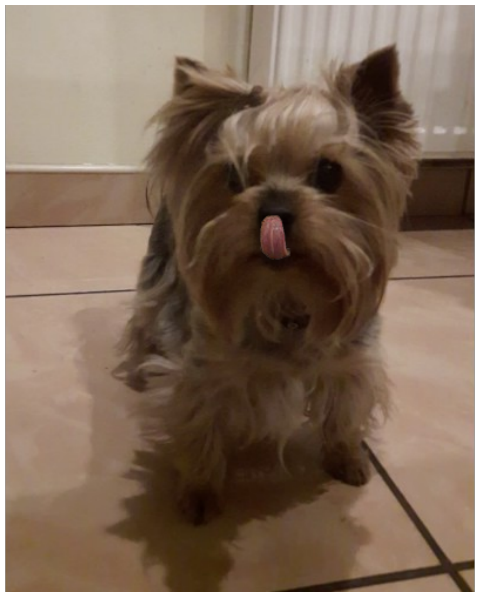
kutyus: Kutya
nev: String = "Buksi"
szin: Color = BARNA
ehes: bool = true

06:15:12

$\rightarrow$

kutyus: Kutya
nev: String = "Buksi"
szin: Color = BARNA
ehes: bool = false

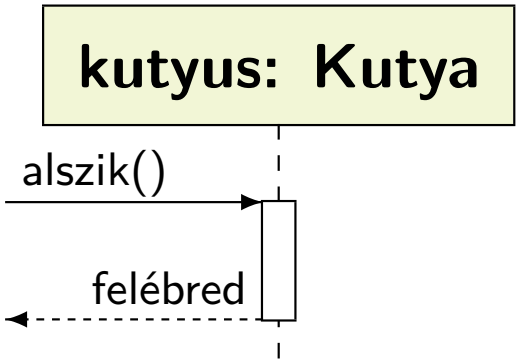
06:16:21

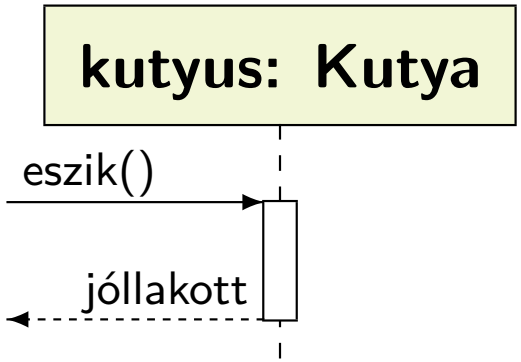


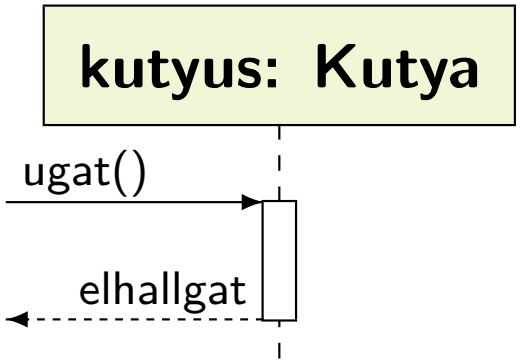
Objektum viselkedése

- Annak módja, hogyan reagál más objektumok kéréseire
- Mindent definiál, amit az objektum „csinálhat”
- *Operációk* határozzák meg
- Példa:
  - Egy *Kurzus* típusú objektumnak lehetnek *hallgatoFelvetel* és *hallgatoTorles* operációi

Objektum viselkedése





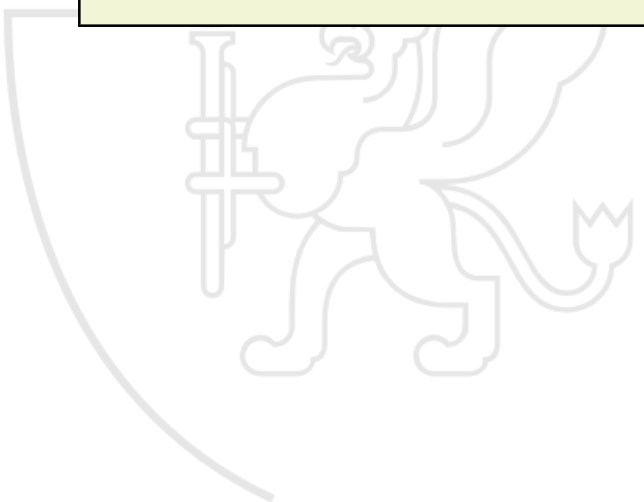


Objektum identitása

- Minden objektum egyedi
  - Akkor is, ha az állapotuk azonos
- Példa:
  - A *programozas* és *algoritmusok* különböző objektumok
  - Mindkettő *Kurzus* (ugyanabba az osztályba tartoznak)

programozas: Kurzus
nyitott: bool = false

algoritmusok: Kurzus
nyitott: bool = false



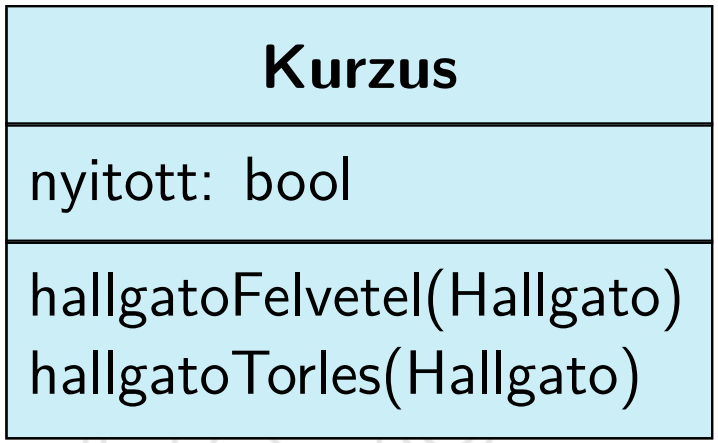
Objektum identitása

kutya: Kutya
nev: String = "Buksi"
szin: Color = FEHER
ehes: bool = false

eb: Kutya
nev: String = "Buksi"
szin: Color = FEHER
ehes: bool = false

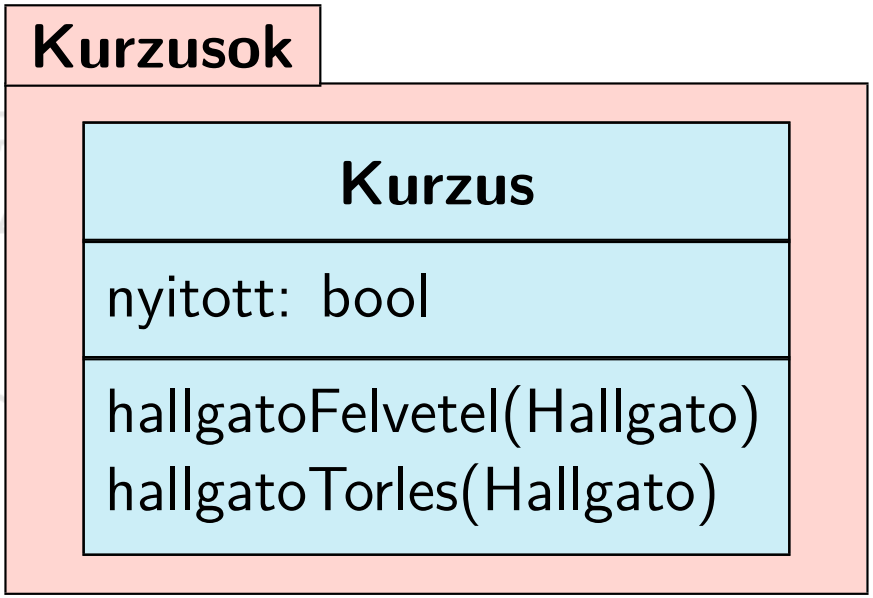
Osztály fogalma

- Leírás objektumok csoportjához, amelyeknek közösek az
  - attribútumaik, operációik,
  - más objektumokkal való kapcsolataik és
  - szemantikus viselkedésük
- Egy minta objektumok létrehozásához (példányosításához):
  - minden objektum pontosan egy osztály példánya
  - az osztály az objektum típusa



Csomagok

- Nagy rendszereknél elkerülhetetlen az osztályok csoportosítása
- Hierarchikus szerkezetet biztosít
- Magasabb szintű absztrakciót valósít meg
- Minden csomag tartalmaz(hat)
  - interfész (publikus) osztályokat és
  - implementációs (belső, privát) osztályokat

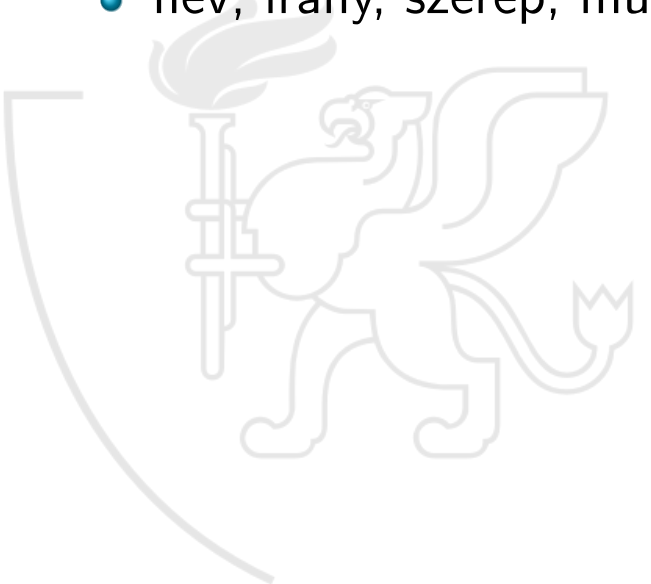


Osztálydiagramok

- Csomagokba szervezett osztályok önmagukban nem elegendőek (modell)
  - nehezen olvasható
- A modell különböző vetületei a tényleges megjelenítései a rendszernek
- Követhetik a modellt, de általában nem:
  - pl. a modell egy osztálya több különböző vetületen (diagramban) is megjelenhet
- Mindig van egy fődiagram, amely tipikusan a rendszer fő csomagjait tartalmazza
- Minden csomagnak is van saját fődiagramja, amely a publikus interfész osztályokat jeleníti meg

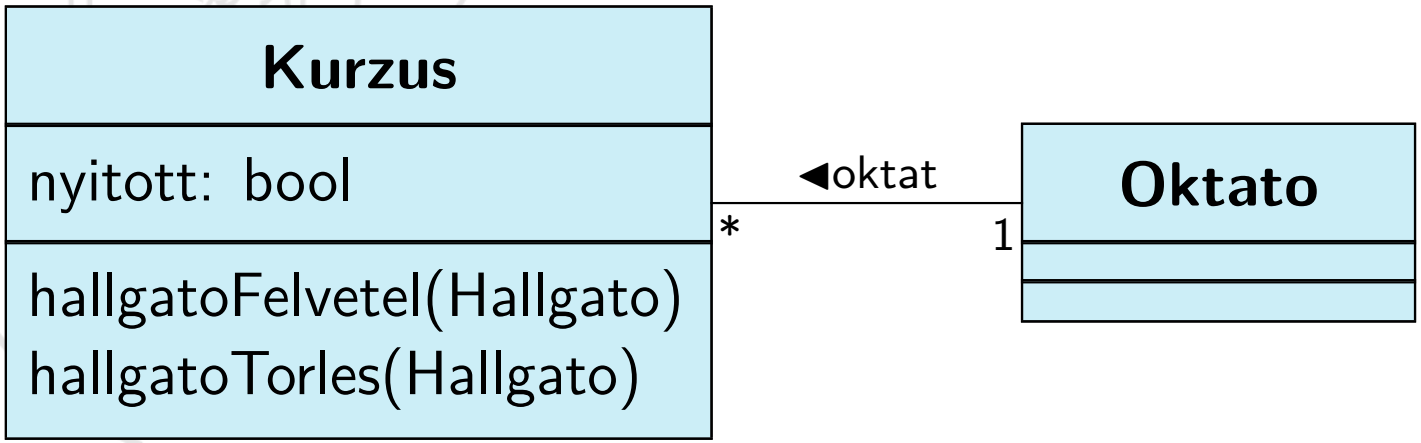
Kapcsolatok

- Objektum-kölcsönhatások megvalósulásai
  - objektum-üzeneteknek a „csatornái”
- Kapcsolatok az osztálydiagramokon:
  - asszociáció
  - aggregáció és kompozíció (rész-egész kapcsolatok)
  - öröklődés
- Kapcsolatok tulajdonságai:
  - név, irány, szerep, multiplicitás, . . .



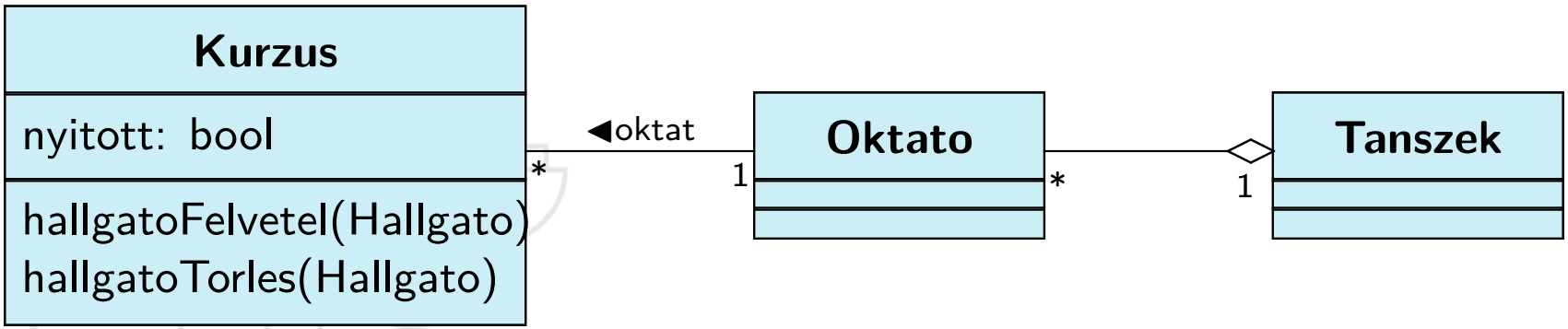
Asszociáció

- Osztályok közötti kétirányú összeköttetés (navigálási irány megadható – üzenet iránya)
- „Használati kapcsolat”, létük egymástól általában független, de legalább az egyik ismeri és/vagy használja a másikat
- Szemantikus összefüggést ábrázol:
  - nem adatfolyam (mindkét irányba lehet információ/adat továbbítás)
- Gyakorlatilag az osztályokból létrejövő objektumok között van összefüggés



Aggregáció

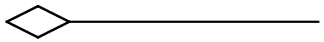
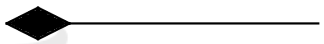
- Asszociáció egy speciális formája
- „Rész-egész” kapcsolat
  - erősebb mint az asszociáció
- Az egyik objektum fizikailag tartalmazza vagy birtokolja a másikat
- UML (rombusz a tartalmazó oldalán)



Asszociáció vagy aggregáció

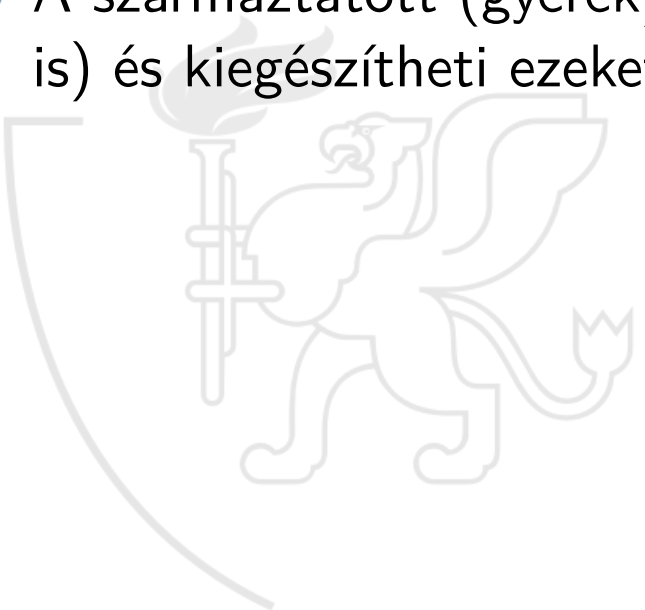
- Mikor legyen egy *asszociáció* helyett inkább *aggregáció* vagy *kompozíció*?
  - a kapcsolat leírásánál a „része” használható
  - a rész-objektum állapota része az egész-objektum állapotának
  - az egész-objektum bizonyos műveletei automatikusan a rész-objektumokra is vonatkoznak (pl. megszűnés)
  - sok esetben nem egyértelmű (ugyanaz a kapcsolat két alkalmazásban másként jelenhet meg)

Aggregáció vagy kompozíció

- Az aggregáció lehet gyenge vagy erős:
  - Aggregáció
    - 
    - gyenge tartalmazás
    - rész-objektum önállóan, őt tartalmazó egész-objektum nélkül is létezhet, tartalmazó objektum nélkül is elérhető, „megszólítható”
    - egy rész-objektumot többen is használhatnak
  - Kompozíció
    - 
    - erős tartalmazás
    - rész-objektum nem létezhet őt tartalmazó egész-objektum nélkül, csak a tartalmazón keresztül érhető el
    - egy rész-objektumnak pontosan egy gazdája van

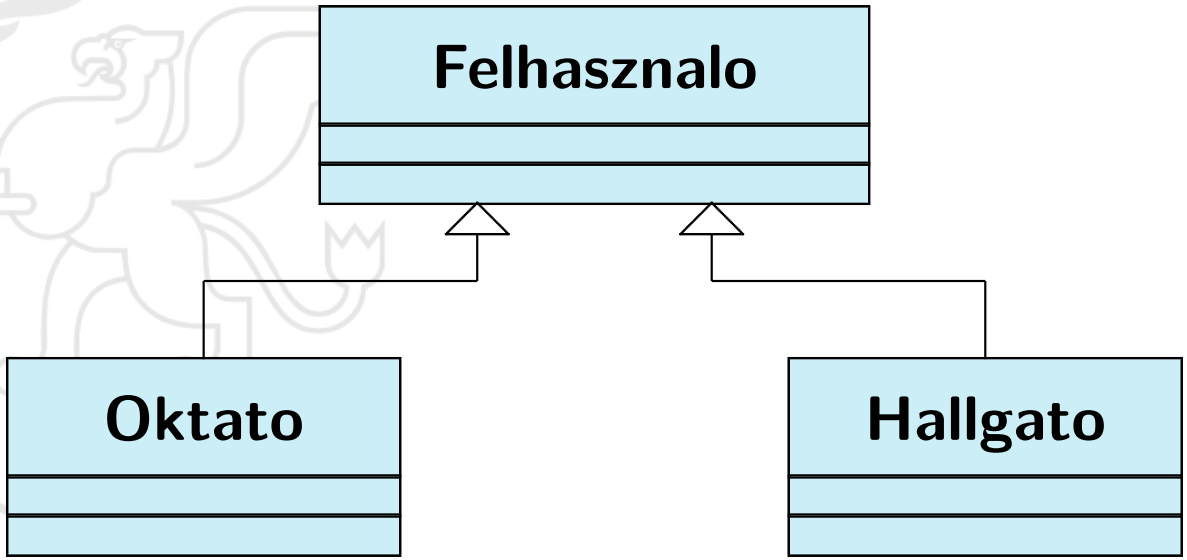
Öröklődés

- Osztályok közötti kapcsolat (reláció) ahol egy osztály megosztja a struktúráját és/vagy a viselkedését egy vagy több másik osztállyal
- Öröklődési hierarchia
  - származtatott osztály örököl az ősosztály(ok)tól
- Az attribútumokat és operációkat a lehető legfelsőbb szinten kell definiálni
- A származtatott (gyerek) osztály mindent örököl az őstől (relációkat is) és kiegészítheti ezeket sajátokkal



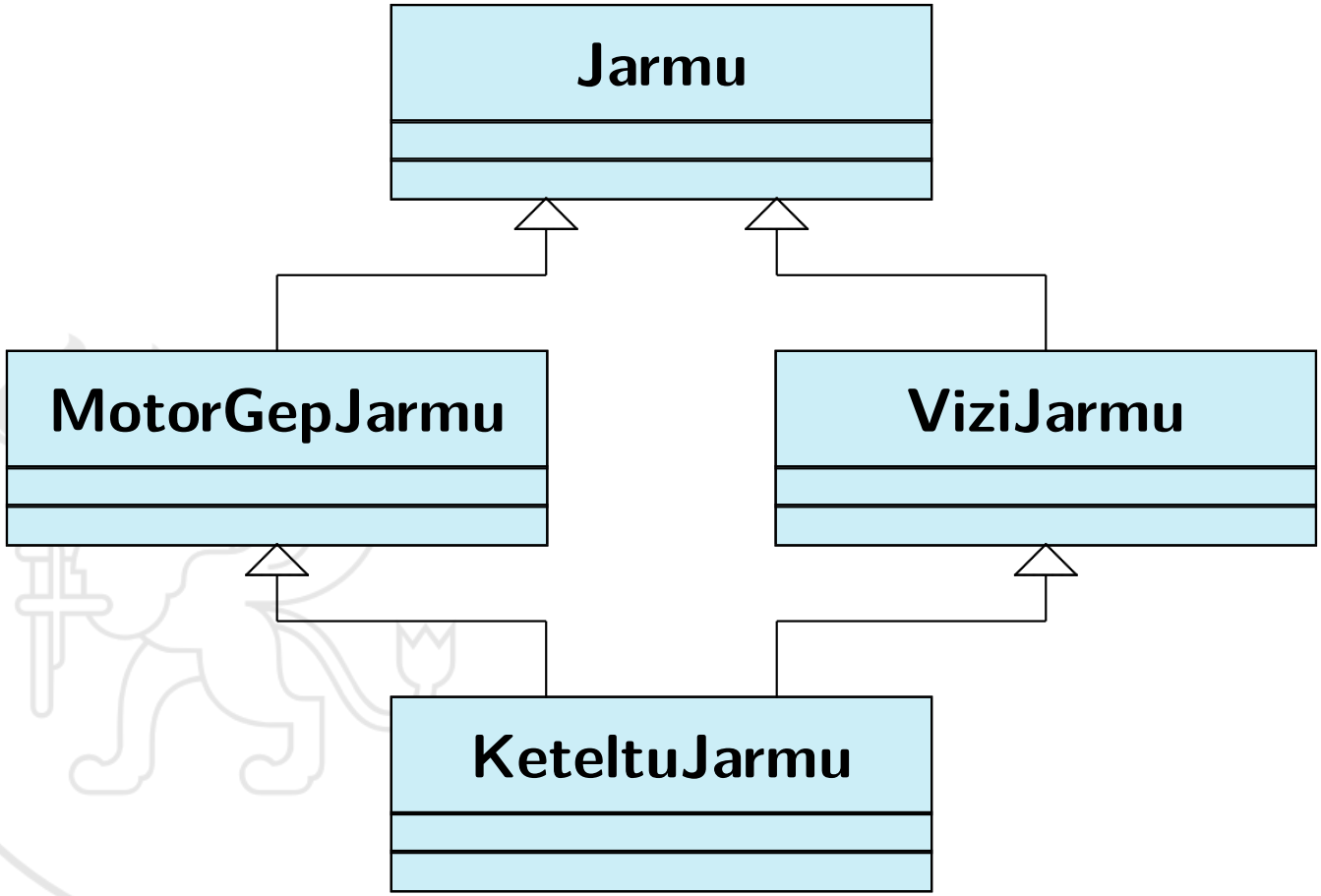
Öröklődés (folyt.)

- A származtatott osztály egy örökölt operáció saját implementációját is nyújthatja: *polimorfizmus* (felüldefiniálás, overriding)
- Az öröklődés relációnak nincs neve, multiplicitása
- Tipikus öröklődési szintek száma: 3-5
- Az újrafelhasználhatóság alapeszköze
- Öröklődés feltárása
  - általánosítás és
  - specializálás



Többszörös öröklődés

- Példa: a kétéltű jármű egy motorgépjármű (ami egy jármű) és egyben egy vízi jármű is (ami ugyancsak egy jármű)



Többszörös öröklődés

- Problémák adódhatnak, pl.
  - név ütközések
  - többszörösen örökölt operációk/attribútumok
- Megoldható: C++ virtuális öröklődés
- Kevésbé karbantartható kódhoz vezet
- Csak akkor szabad használni, ha tényleg szükséges, de akkor is csak nagy odafigyeléssel
- Java-ban nincs rá lehetőség