

# 1. UML

## Ví am UML?

(Object Management Group)

OMG egyetemes modellrádió nyelv

Szabványrendszer modelljeinek felülvétel + tanzelesek - részletek leírását, megjelenítését, dokumentálását segítő nyelv

Előzmény => Booch, OMT, COSE => Ezek fejlesztői kincsén coincidens az UML-t

### Object Constraint Model

Modellész nyelv - formális nyelv kifejtéséhez leírásra  
UML modelljeinek

Nem programnyelv! (nem használható csak direktan) Típusok, ennek kiválasztásának

### XMI - XML Metadata Interchange

XMI formátum metadatok alkalmazásukhoz kiálti szerepében  
UML modellök szereje

## Modell - Metamodell

Modell - Egy rendszer leírása => Rendszerek közötti felülvizsgálat

Rendszert egy bármelyes módon felülírható információ, emiatt elérhető  
egy bármelyes szolgáltatásnak, alkotóelemek szinten

Teljes => minden rendszert, esetleg többet is, amelyeknek  
bármely alkotóelemre, amiből lehetséges a cél  
teljesítése

# Metamodell

Működő modellje

UML-ben a metamodell egy olyan modell, mely önmagát modelleni

- eis modellet is metamodellek modelllezésre is használható

MOF egy metamodell

UML alkotrodott szintaxisát UML metamodellel határoznak meg

## Szabványt -specifikus nyelvek

DSL - Domain Specific Language

Egy terminus fejlesztés problémára koncentrált szemantikus nyelv.  
Például: Van általános célú nyelv.

- BibTeX, LaTeX, CSS, Node, Plant UML, SQL

## Szöveges - elvileg nem létező

### MOF - Meta Object Facility

Nyílt, platformfüggetlen metadat "rendszer" Generálásra, és többet kínálhat szolgáltatásokat birtokba => Működő - is metadat rendszer fejlesztése  
Metamodellök definícióival szolgáló DSL

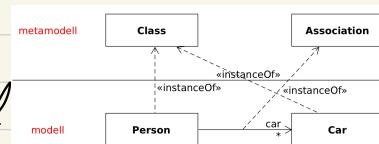
UML metamodell => MOF az alapja

### Hatóterülett metamodell architektúra

Nyelv Specifikáció - Metamodell

Felhasználói Specifikáció - Működő

Működő implementáció



# Négy rétegű Metamodell architektúra

## Meta-metamodell

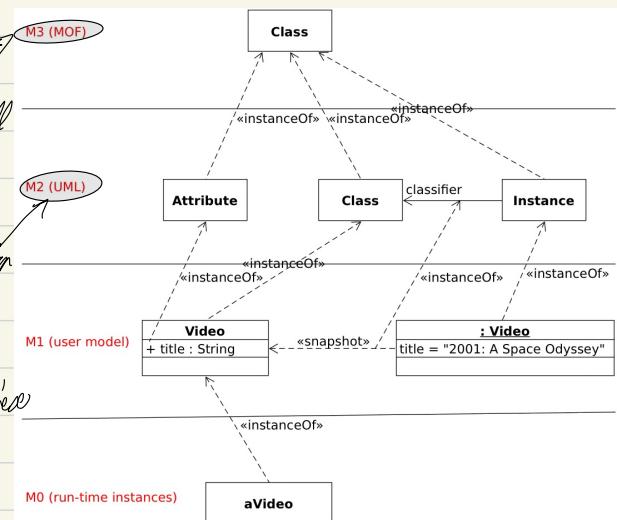
- nyíltan definíálható metamodell megadására
- tömörítő működésben általában érhető metamodell

## Metamodell

- egyetlen részlegű a meta-metamodellnek
- felhasználó nyíltan definíálható modell megadására

## Rodell

- egyetlen részlegű
  - felhasználó részlegű problémamegoldók modellben
- Felhasználó részlegű



## UML Modellelementek

### Contályozás

Olyan modellelemben mely hivatkozás füllennélküli rendelkezésre állásával rendelkező részlegű az egyetlen részlegű kontályozás.

Hierarchikus, összetett részlegű

### Specializáció

- Datatype
  - Association
  - Interface
  - Class
- (mint a kontályozás)

### UML Diagramm Tippek

#### Szerkezeti Diagramm

- Objektumok statikus felépítését mutatja
- Sötétből független elemek

#### Viselkedési diagramm

- Objektumok dinamikus viselkedést mutatja
- (együttműködés, várakozás, állapotváltozás)

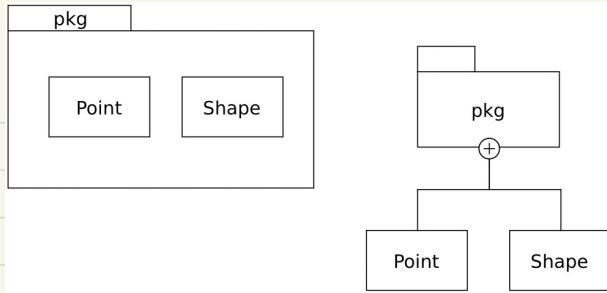
- Sötétből valóságosak sorrendje lehet "nem keverni!"

# Comagoo

Modellelerek csapatosítása  
szolgáltó funkciókkal

Névtelen határon meg

Hivatalos: `Phg::Point, Phg::Shape`



# Stukkónevek

UML felülvésmód fejlesztői szerei

Szöveges annotációkban jelölés meg vagy UML grafikus eljárás használata  
(előre írt műfajt) azon használata (respectively)

Lekérdezni a grafikus felületen UML felülvésmód megadásának következetét  
Egy modellelemre több is vonathozhat => elszorcsás

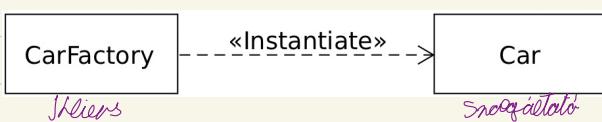
Példa: <<interface>>

francia idézések (red)

# Megjegyzés

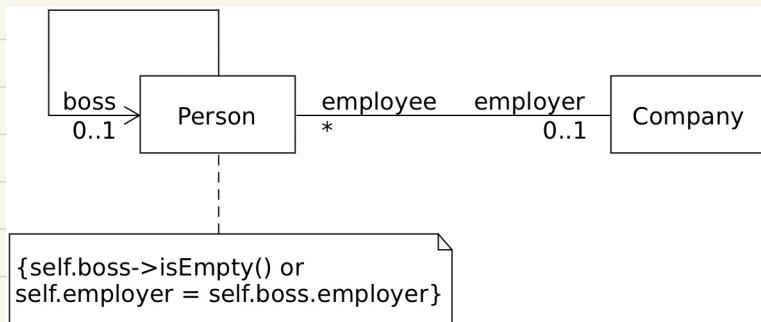
Nincs jelentére, csak potenciális hozzájárulás a modell olvashatóságához

# Függőségek



Modellelek előtti szolgáltató - helyes megközelítés  
Szolgáltató modellítésben hatással lehet a beliévre  
Megszüntetés

- { [név:] logikai-kifejezés }



# Ontologydiagram

Olyan funkciókat írja le, és a köztük fennálló hálózatot  
származtatják a kódolásra.

Leírások az ontologyak tulajdonságait, műveleteit, megvalósításait

(objektumok konstruktorai)

## Fajtái

### Elméleti

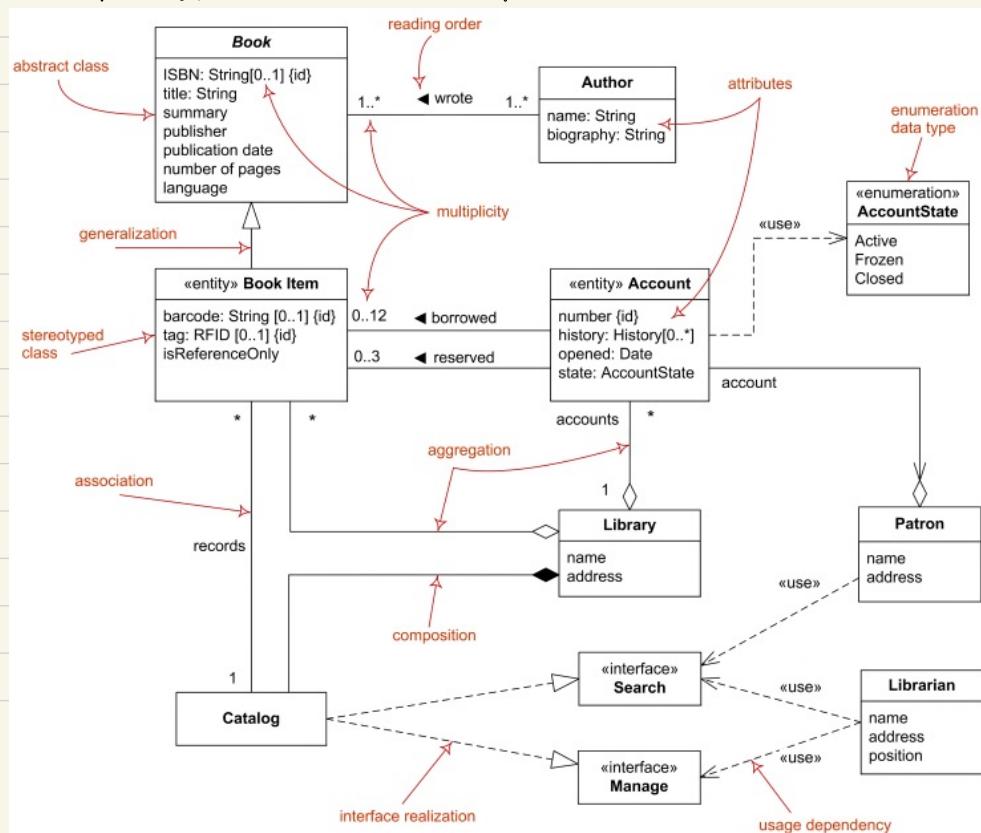
Elméleti szerkezet az ontologyoknak szolgálókhoz, ontologydiagramnak a szabványos felépítése

### Térítési

Végfelhasználók számára megvalósítható technikai szempontból

### Implementáció

Az ontologyok egy implementációjának konkrétni részletei



# Osztályos Fejlesztés

## Jelölésekkel Általánosítás

Név
Attribútumok
Műveletek

- + public
- private
- # protected
- ~ package

## Szaincrság

Leggyorítóis hozzájáró elnevezésére

[alsó\_korlát ..] felső\_korlát

\* => korlátos (renegatív egész)

Ha ezt kiszorítja, elég az kiírni

## Tulajdonságok

Egy attribútumot, vagyis, csomocciacionról általános

Örökölt Szinonimával elhelyezik a default

[^] [láthatóság] [/] név [: típus] [[ számosság ]]  
[= alapérték] [{ módosító [, módosító}\* }]

readonly, ordered, unique ...

## Paraméter

Paraméterlista  
paraméter [, paraméter]\*

In default, out, inout, return  
[irány] név : típus [[ számosság ]]  
[= alapérték] [{ tulajdonság [, tulajdonság}\* }]

ordered, unique ....

## Növekedés

[^] [láthatóság] név ([paraméterlista]  
[: típus] [[ számosság ]]  
[{ tulajdonság [, tulajdonság}\* }])

ordered, nonunique ...  
+ merev => felbontás, nem vállalhatja a rendszer állapotát

## Person

-title: String[0..1] => 0 vagy 1 titulus  
-name: String  
-birthDate: Date => megnövelte  
-/age: int {age >= 0}

## Attribútumok

In növekedés

+Person(title: String, name: String, birthDate: Date)  
+Person(name: String, birthDate: Date)  
+getTitle(): String {query}  
+setTitle(title: String) const ordered  
+getName(): String {query}  
+setName(name: String)  
+getBirthDate(): Date {query}  
+setBirthDate(birthDate: Date)  
+getAge(): int {query}

## Növekedés

Singleton
-instance: Singleton
-Singleton()
+getInstance(): Singleton

Statikus attribútumok és "unireltetők"  
alakíthatók

## Abstrakt kontígyák

Nem részlegesítethető kontígyák

kontígyázók

Nem lehet többel is vagy  
nem utor/alkot adható meg  
az abstract 3 móvegezés,  
annáluk

UN 2.5.1-ben nincs

Elág.

Shape
-x: int
-y: int
#Shape(x: int, y: int)
+getX(): int
+getY(): int
+moveTo(newX: int, newY: int)
+getArea(): double
+draw()

## Associációk

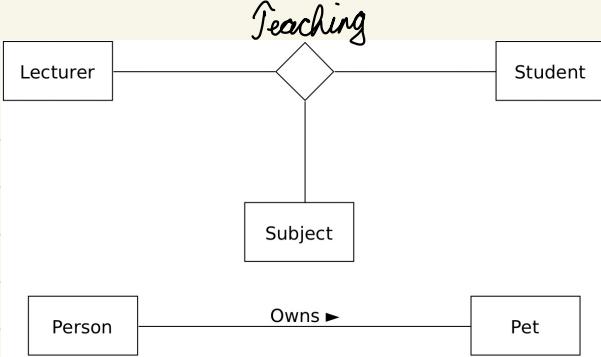
Szemantikus viszony kontígyációk jelölésével leírók  
tárolását célnak kívánjuk. Egyike megfelelően ragy  
implementálásban az associação kifejezés

Legalább 2 tól több van  $\Rightarrow$  Ha van: bináris, associação  
egy encsoportban (verb) egy associaçãoi relációja

Jelölés  $\Rightarrow$  Círcusára alkotott névhez, melyet poligonokkal  
szabhatunk össze a végi tipusoknak kontígyációjával  
bináris associação

Elág.

Hét kontígyárt csinálható  
poligonokat szabhat

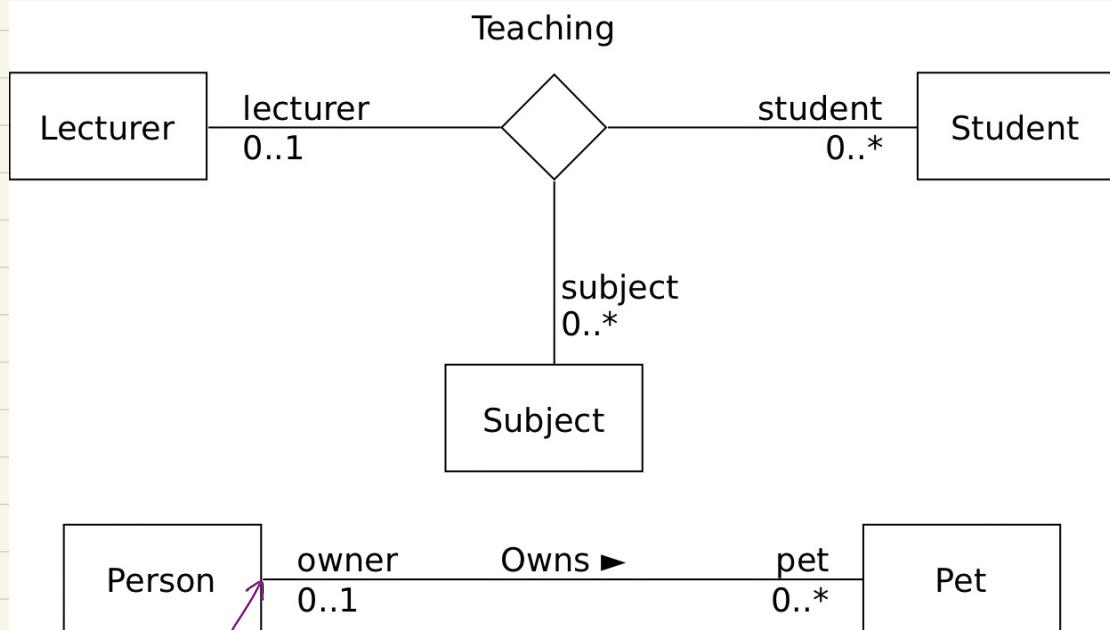


Associációi vég

Associációt ábrázoló vonal, is egy kontolyon általánosítva használata  
Vonal végeinél összeférhetők lesznek:  
 - Név / Szerelembőr  
 - Számoság  
 - Körüljárás  
 - Sárhatalom

## Navigálhatóság

Vonal vezér előre nyílt azeljük hogy navigálható, X ha nem  
Tetézi időben hatékonyan elérhető - e egységes a művek



Szélelhetőségi szabályok

Az iratitásnál ha a modell törlésnél egy tulajdonsgát, melynek törlesztése a port címét erősített kontolyon ábrázolja, õ a tulajdonig a másik véger előre kontolyozottakhoz törleszt

# Egész - Rész Kompozíció

Bérciés asszociáció egész - rész kompozícióval kifejező fogalmi

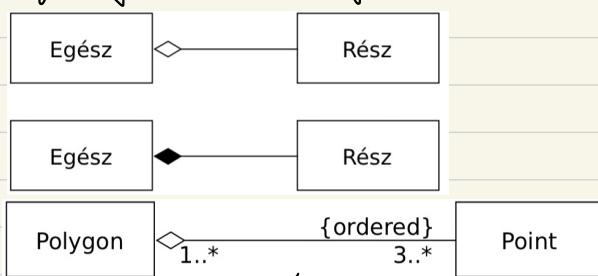
## - Aggregáció

Ha néhány objektum szükséges több aggregációs objektumhoz is tartozhat a részek és az aggregációs objektum számtól függően is elválthatók

## - Kompozíció

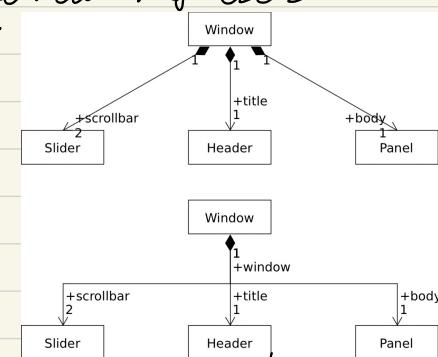
Aggregáció önmagában formálja. Ha néhány objektum szükséges több aggregációs objektumhoz tartozhat, akkor kompozíciót objektumhoz törlésekben van szükség néhány objektumhoz valamit.

Bérciés asszociáció egyik vége jelölik meg csak aggregációból vagy kompozícióból



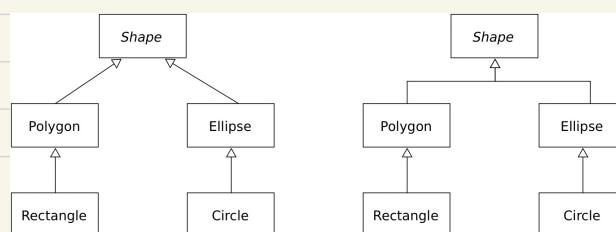
Aggregáció

Általánosítás



Kompozíció

Általánosítás - specifikáció önmagával kontolgyancsban kívül Spec kontolgyancs önmagával közös általános kontolgyancsval

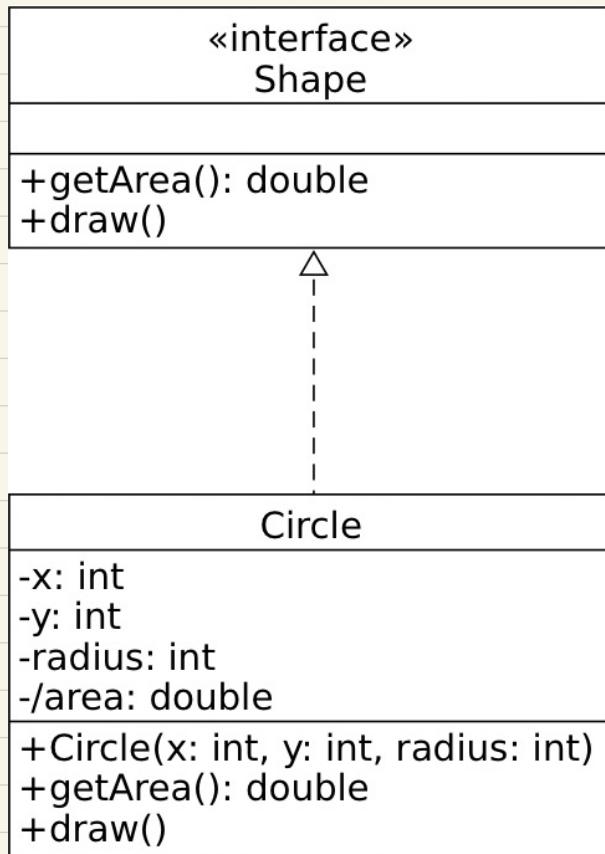


Összefüggő részterülete minden általánosításnak részterülete  
Specifikus kontolgyancs öröklíti az általános bennanyes területet

# Interfaces

Működési jellemzőt is hűtőszabályt definiáló kód blokk, amelynek eljárásai nem több mint az implementálásban

Nem részlegesítethetők, csaklyanás implementálás vagy realizálás az interfész specificációt



## 7. Szoftvertesztelés

### Mi a Szoftvertesztelés?

Program vizelkedésének tesztelése a várt kiiratellel összefüggően  
Tesztelés véges időn belül történik

Szoftver környezetének megállapítására meghatározott  
Szoftver használata előtti rész, melyben meghatározott működés.

### Verifikáció és Validáció

**Verifikáció:** Szoftver megfelel-e a céle szerinten  
támwartott követelményeknek  
(funkcionális, és nem funkcionális)

**Validáció:** Szoftver megfelel-e az ügyfel elvárásainak

### Hibát leíró szabályozások

**Jelzők / Error:**rossz eredményt adó esetek leírása

**Hiba / bug:** tökéletlenseg/kialányodás egy-egy leírásban  
melynek nem teljesül a követelményi előírások

**Neglázásnak fejlődő:** sejten, errevel egy kapcsolts vagy  
rendben nem lesz egy meghövöllett funkció  
a meghatározott hatások között

# Tesztelési Magyarázat

- ① A tesztelés kilalkás jelentéletét mutatja meg, nem kiüríteni
- ② Lehetetlen a teljes / finomított tesztelés
- ③ Thorai tesztelés időt, részt takarít meg
- ④ A kilalkás csapottcukorral
- ⑤ Ávahodj a hirtelen poradószínű
- ⑥ A tesztelés horogszemfűző
- ⑦ Lilámenteség egy törököt

## Tesztelés / Tesztadat

### Tesztelés

Berendezés, végrehajtásra feltételek → előírt eredmények  
egy halmona melyeket horogni céllal borítva meghatározva

↓  
Program végrehajtása, működési hibák rögzítése ellencímzések

Tevént feltelelek elajánlás meghatározott előírt feltelelek, berendezési rendszeregek, általános eredmények és utánpótlások rendszere

### Nagyszerű tesztelés

Működést előírt feltelelekkel, berendezési rendszerekkel előírt eredményekkel és leírásokkal rendelkezik

### Nagyobb mennyű tesztelés

Ugyancsak, csak nagyobb mértékben dolgozik

### Tesztadatok

Tevént végrehajtásban előírásokat adnak

# Fentelisi Sintez

## Egyégeontelis

Függetlenül testtelhető önmagasságra összehoztak  
Harmonikus változtatásra után érdemes futtatni  
Lehet TDD-t is tolpi  
Harmonikusnak szemben fejlesztési részről

## Integrációs testtelis

Harmonikus vagy rendszerek közötti kommunikációt testeli  
le integrációra koncentrikusnak, nem a harmonikus hűtő - hűtőr  
működésére

Jel figyelem: Harmonikus  
Fejlesztés felélegysége

Rendszerek  
Testtelis felélegysége

## Rendszertesttelis

A rendszer egészének viselkedését testeli  
Független testtelis részről

## Elégadási testtelis

Rendszer része - e a telepítésre és az egyfél általi  
használatra

Egyfél rendszerré szemelhető felélegysége  
Működési testtelis

Felhasználók  
és fejlesztés  
együtt

Egyfél és felhasználók  
Felhasználók helyér történik

Fejlesztő fejlesztés

# Tesztírások

## Funkcionális tesztelés

Irrational tesztelés: nem a rendszer esemény  
Rendber tesztelés: minden esemény

## Nem funkcionális tesztelés

Hosszúállhatóság, teljesíterem, biztonság értékelés  
Amely a tesztelés a rendszer működésétől törni a dolgjait

## Fehér dolgok

Rendszerrel "felülről vagy implementációján  
alapuló tesztelés

Hosszú, architektúra, rendszerekben belüli meghajlásokat

## Váltással kapcsolatos

Rendszer endocitikusai, vagy funkcionális  
mediotípusai / hosszadásiak

Itt figyelem:

Regressív => eredeti hiba mi lett javítva

Progressív => Váltással által előzetesen  
alacsony működési kritériumok elérésére

## fő egységektesztelés részletei fogya

Fast => gyorsan futnak le

Independent => függetlenek egymástól

Reusable => Bármiely könyvtárból regisztrálhatók

Self - Validating => Logikai hibákat vagy átmegy vagy megoldás

Timely => Ideális időben kell megírni, hónapokkal a  
tesztelendő hónál előbb

# Egysegtesítés Szereise - ATA

**Amraige** => Törlettel rendelkezésre álló szolgáltatások  
ellenőrzésére

**Act** => Törlettel rendelkezésre álló szolgáltatások megújítása,  
előzetes törlesztésére

**Dessert** => Hivatalos ellenőrzés

## J Unit

### Tesztkontrolály - Törletemelkedés

**Tesztkontrolály** => felügyeleti kontroll, statikus tagoztatály  
vagy @ festettség kontroll, melyben  
legfeljebb egy tesztreléjes van

### Törletemelkedés

@\* test \* ameliorációval elérhető részleges törletemelkedés

Nem minden public legyen, de nem lehet privat

Lehet kontrolljáról belül funkcionális dokumentáció, örököshelyes  
Nem lehet alapvető, nem ad vissza eredményt

Lehetnek konzumálás => Dependency Injection

### Teszt vegrehajtási életciklus

Kincs, törletemelkedés és törletemelkedés közötti kölcsönhatás

Megvalósítható => @Test felsorolás(Lifecycle.PER\_CLASS)

Sorrend determinálható, de nem sorba

# Tesztteredmények

Siker  $\Rightarrow$  Passz

Buñdes  $\Rightarrow$  Van egyenlőség az esetben a vártoal

Hiba  $\Rightarrow$  Nem fut le a teszt  $\Rightarrow$  Errorless

## Irodalomszerzési Vérfelvétel

Utazás lefektetéséig  $\Rightarrow$  Végrejelzett elutazásra

Összes utazásra

Sor lefektetéséig  $\Rightarrow$  Végrejelzett hibákra

Összes sorra

Ig lefektetéséig  $\Rightarrow$  Végrejelzett ágak

Összes igra

||  
H/Switch sorra

Miben érzi?

Né a magas lefektetéséig legy a cél de nem rövid be az  
fő negatív, de rövid pozitívindikátor

### 3. OOP tervezési alapelvek

#### Statikus Modellekben

Programunk elemeihez a hozzájuk közelítési mód

Statikus modellben körönök => autonómus elemek

Java: Checkstyle  
ErrorProne  
Null away  
SpotBugs

JS: ESLint/Flint

Python: Prorector  
PyLint

C#: StyleCop  
Infer Short

C/C++: CPCheck

PMD => Statikus Modellekben

#### DRY - Don't Repeat Yourself

Né legyen felülböges ikerelő => Ellenőrzésje WET

Write Everything Twice

#### Ikerelősi Tájatás

Hagyományos => Minős és valóság, a kiemelést megkövételi

Nem öröklődés => Nem emeli ki az a fejlesztők

Ikerelőtől => Futtatásának fárad, így hosszú ideig tölcsér

Fejlesztői öröktől => Nagy vagy több csatorna információ elszállás

Ikerelőtől => Nagy Posta, ennek folyamán minden mely többetől előfordul  
Nem mindig információ ikerelő

DRY el megsérítés van meadis öröklési

DRY információ van elérési szüksége

```
class Line {  
    Point start;  
    Point end;  
    double length; // DRY violation  
}
```

=>

```
class Line {  
    Point start;  
    Point end;  
  
    double length() {  
        return start.distanceTo(end);  
    }  
}
```

Jelentősen egyszerűbb megírásba

Indenes elválasztási

Representációs vanilles elválasztási

KISS - Keep it simple stupid

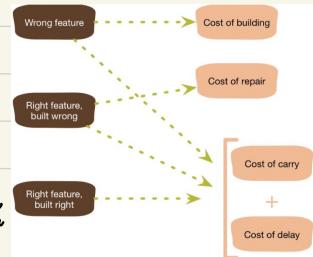
Vinclar alyan egyszerű kell legyen csempészni céltérig, de semmivel sem egyszerűbb

YAGNI - You aren't gonna need it  
Egyet nem programozás egyszerű alkalmazása

Kinekik ahol implementálj valamit amikor kell, nincs ahol másikra szántott helyet kell

Nem vonathozik a szolgálat működtetését  
Öröklítő törekedésre

Oda a hibák rövidítésére



# Csatlakozás

Egy szoftvermodul függvényei mértékben egy másik szoftvermodultól.

Rendezési szabványokról: tan/izmár

Nem jó a szerves csatlakozás  $\Rightarrow$  konfliktusok, rekesz  
modellhatáros horizontális  
szövetségi és ügyfelháználkülső

## Szervi csatlakozás

- Nyilto port által megfelelő kód incs  $\Rightarrow$  Hibajavításig
- Portbanban fejlesztés

Horizontális

## GOF alapelvek

- Interfície programozás ne implementációra  $\Rightarrow$  feltérkészítés minták
- Kompozitum over Inheritance  $\Rightarrow$  Sebés/Sebés dolgozó leírásai

Hét leggyakoribb meccsen az ügyfelháználkülső

## Öröklődés

### Előnyei

- Szárhármas, konditív időben
- Cégesnek
- Környelők ar ügyfelháználkülső implementáció módosításra

Túlcsökkentette a fejlesztési időt

Szükségteljes maghatározás alkotályai a funkciós ábrázolását

- megengi az objektivitásnak szereplőit

- Szűkű vállalkozásnak alkotályai a vállalkozás

Implementációs függőségek gondolatvezetésben alkalmazás  
ügyfelszolgáltatóknál

## Thamposicid - Objektum összetétel

Dinamikus, futási időben történő, objektumok  
helyesül valóban más objektumokra hivatkozás

Objektumok legyelvezető részei egymás interfejszel  
" " →  
Gondozott megtervezett  
Interfejszek van azoknál

### Elényei

Interfejsen keresztül ejtés el az ontológiáról → egységhatáról nem számít meg  
Túlcsík időben lecsereálhatók objektumok, amig létrehozva megágyazni  
Segít egységhatárának → Nincs ontológiája feladatra összpontosít  
Kisebb kontolók, kontolýhierarchia → használhatóbb lesz

### Hátrányai

Több objektum, és a rendszer viselkedése nem egyszerűsítőleg függ  
" " →  
Nem egyszerű kontoló határozza meg

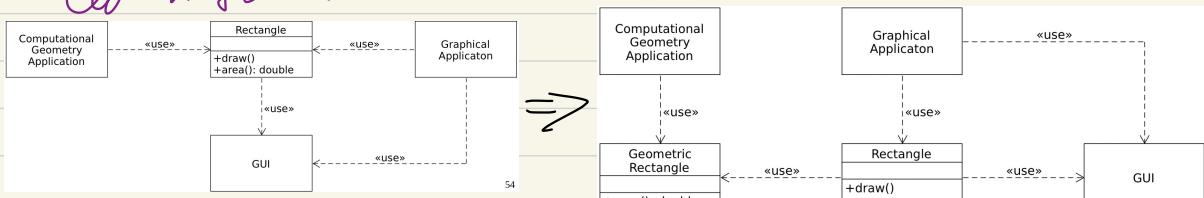
## SOLID原則

### Single Responsibility Principle - SRP

Egy kontolónak csak egy olyan felelősségre vonatkozásra → így felelőssége  
Nincs felelőssége a viselkedésről, ha a következőkkel vonatkozik  
a változás a felelősségről történő viselkedést nyilvánít meg

Cágnél több felelősség esetén a felelősségek osztalhatók  
válnak. Egy felelősség változásra illeszkor módosíthatja a többet.

## Ölv megszerzése



- A Rectangle felelőssége a táglatba geometriai címek modellenése
- és a grafikus megjelenítés

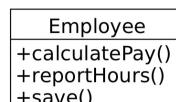
Set van en die een model is een view

Ölv finanszírozva  $\Rightarrow$  Ossztalható VAGY modulnál leghyoggabb felelőssége  
Ovat egy "alternatív" törleszter felelősséggel egy modul

## Hosszúelidőű tervezési minták

- dekorátor
- felelősségös

- calculatePay(): a bérörszínű határozza meg
- reportHours(): a munkaügyi osztály határozza meg
- save(): az adatbázis adminisztrátorok határozzák meg



## Open-Closed Principle - OCP (Nyitott szint)

Szoftver entitások legyenek nyitottak a körültekre, de zártak a modernizációra

### Nyitott a körültekre

### Zárt a modernizációra

Modul viselkedésének hibajavítása

Modul viselkedésének hibajavítása nem mindenkorán  
vagy modul/tervezés hibajavítás valószínűsége

- gyártó minták
- kezelés
- stratégia
- szablonfigurák
- kategória



A független csatát az a szervír hozzájárلja,  
mert lehet egyszerűbb lecserelni

## Liskov - Substitutionicor Principle - LSP

- Barbara Liskov által
- Ha S típus a T típus alkalmazása, nem változtat a program működését, ha a verne a T típusi objektumokat S típusi objektumokkal helyettesítjük

## Interface Segregation Principle - ISP

Néha bonyolultsági szinten több kölcsönösen nem kölcsönösen működő részre van szükség

### Varázslat / Fat Interface

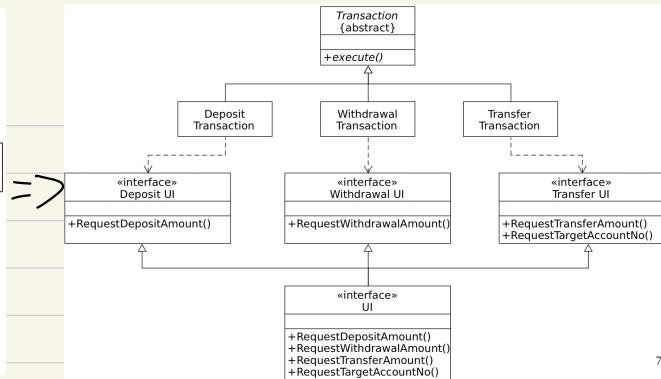
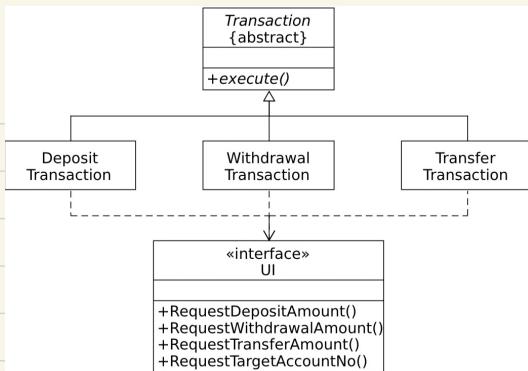
Egyetlen szélesített több tagfüggvényel rendelkező interface

Ennek felhalmozásakor interfész szélesebbé válik, de minden interfész - elhelyezkedésben kontingenčiálisan - megfelelő, melyben lehet mindenhol használni.

ISP elvben, néha szélesítve ennek

### Interfész Szemantikai → Függőségek megtávolítása

Ha egy interfész lelt kontingenčiálisan minden mindenhol használja valamelyik → Van mindenhol csak olyan



## Dependency Inversion Principle - DIP

Nagysági szintű modulokról re függjünk abstrakcyós szintű modulokról

"Kiválható" abstraktióktól függ

Abstraktiók nincsenek rögzítéssel, a rögzítés függ a konkrétnak megadott funkciókkal történő összhangban

- illesztő

Hagyományos međodunkat használva az abstrakt kifejezést érthetjük

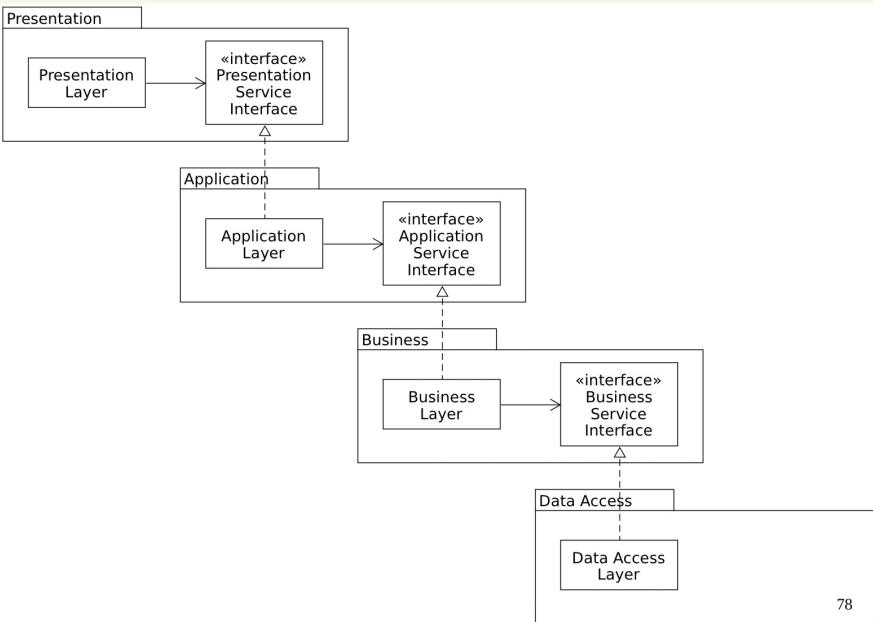
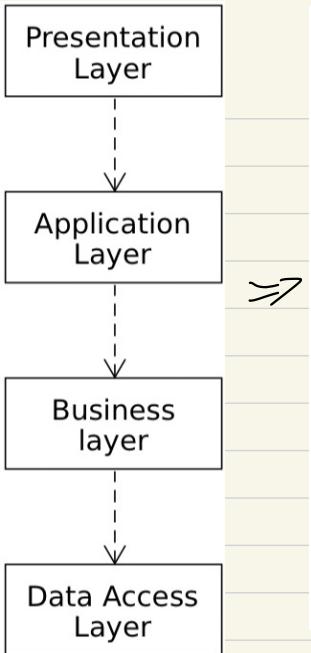
Nagysági szintű modulok => ületi logikai, modell

Vállalkozásnak nincs maga => re függjünk az implementációtól  
Ületi logikát

Nagysági szintű moduloknál sorrendben alkalmazhatunk

Abstrakcyós szintűkönve ott van a programrendszerünkben

Ha nagysági szintű modulok függhetnek a konkrétnak szántakról, könyvelni alkalmazhatunk



78

Minden Interface-en keresztül tökéletes  
a függőségek és az interfész telepítendői is megfordult

## Függés az abstrakciótól

Nem függjön a program hozzájáról csatolynakról, hanem  
abstrakti csatolynakról és interfész -en tőle

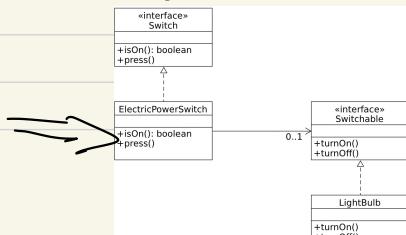
- vállaló ne kiválasztva hozzájáról csatolynak

- csatoly ne visszavonva hozzájáról csatolynak

- egyetlen metódus is jogos felül csatolynakban implementálható metódust

Ezt egyszerű program logikára egyszer megírni

Nem-túl gyakran váltanó hozzájáról csatolynak esetén megengedhető a függés



*Note:* SRP es ISP konkurálnak, de nem minden funkció redundáns

Not really

Imagine a stack class with both push and pop. Imagine a client that only pushes. If that client depends upon the stack interface, it depends upon pop, which it does not need. SRP would not separate push from pop; ISP would.

## Dependency Injection

Inversion of Control architecture alkalmazásánál szükséges erre

Címen megfertőzési elvek és minták összefüggése melyeket minden csatolt hirdet fejlesztők török alkotnak

Hitelesítettség  $\Rightarrow$  Körlátozott hatóság

Egy objektum egy címen megfogalmazás, melyet más objektumok elismernek használva

Objektumok követlik a hosszú-sorozatok önmagukat  
függések reverensi  $\Rightarrow$  Csak tranzitív

Függőség

Ihlető objektum igényelt megfogalmazás

Függő

Ihlető objektum, melynek egy függősége van előtte

Objektum Graf

Függő objektumok és függőségeik összefüggése

Injector

Egy ihlető függőségeinek megadása

Dependency  
Injector  
Inversion of  
Control  
II

## DI/IoC Frontiers

Dependency Injector funkcionális részletek programozásban  
Nem megyek DI-Ban DI-frontiera

Fizikai DI  $\Rightarrow$  DI-frontier részletek Dependency Injection

A DI objektum gyártó kábelny a többi objektussal, emellett mindenivel is legjobb gyakorlatban fogalmazni

DI frontendben "nincs" hűtő elérhetőségek között függő  
szereinél törekedni kell a visszatérítésre

Nincs DI

```
public interface SpellChecker {  
    public boolean check(String text);  
}  
  
public class TextEditor {  
    private SpellChecker spellChecker;  
    public TextEditor() {  
        spellChecker = new HungarianSpellChecker();  
    }  
    // ...  
}
```

DI frontendben

```
public class TextEditor {  
    private SpellChecker spellChecker;  
    public TextEditor(SpellChecker spellChecker) {  
        this.spellChecker = spellChecker;  
    }  
    // ...  
}
```

DI Setterrel

```
public class TextEditor {  
    private SpellChecker spellChecker;  
    public TextEditor() {}  
    public void setSpellChecker(SpellChecker spellChecker) {  
        this.spellChecker = spellChecker;  
    }  
    // ...  
}
```

Interface DI

DI előnyei

- Interfacekkel
- Herediterhetőség
- Invertált kötöttség
- Fejleszhető (Unitest)

```
public interface SpellCheckerSetter {  
    void setSpellChecker(SpellChecker spellChecker);  
}  
  
public class TextEditor implements SpellCheckerSetter {  
    private SpellChecker spellChecker;  
    public TextEditor() {}  
  
    @Override  
    public void setSpellChecker(SpellChecker spellChecker) {  
        this.spellChecker = spellChecker;  
    }  
    // ...  
}
```

## 4. Tértervezési minták

### Mi a minta?

Ügyre és ügyre felbukkanó problémáit is, és használja megoldásainakat

Felirat névből álló szöveg, kiírás, grafikus és megoldás hozzájárulhatnak.

Ötlet mely gyakorlati környezetben már használtan van, és könnyen más környezetben is használható.

Gyakori probléma általános megoldásainak összefoglalása

### Architektonikai minták, MVC

Szoftverrendszerrel alkotott felületekre vonatkozó minták  
Ahol elérhetők az adatszolgáltatók, esetleg a felületszolgáltatók, szolgáltatók, irányítók a modell kezelésére

### MVC

Modell - Kérő - Terület

Környezet

Programszerkezeti - gép felüleettel rendelkező interaktív alkalmazások  
Probléma

Gyakori igény a UI valósítására  $\Rightarrow$  UI környezetet kell kezdeni a modell kezelésére nélküli

Megoldás

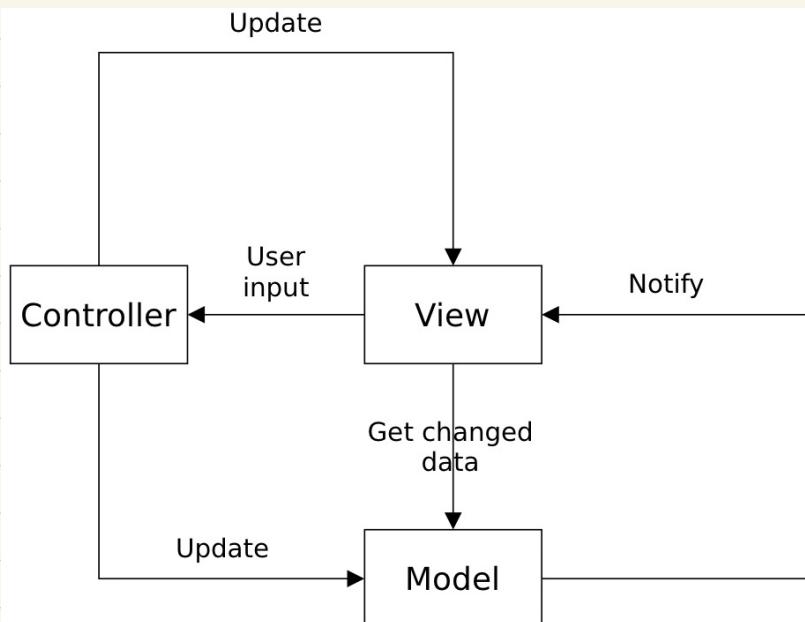
Interaktív alkalmazások körül nézve

- Modell  $\Rightarrow$  üres szöveg, szöveg mutatók  $\Rightarrow$  fogadás a megoldásban

- Kérő  $\Rightarrow$  fogadás

- Terület  $\Rightarrow$  fogadás inputok a résztvevők, modell alapján keletkezik

Módellek változtatása  $\Rightarrow$  Föld rész megváltozásai és  
Vonj el változtatás a "modellben" a tervezéshez használt formához



## Tervezési Mérkőzések

Hosszú értelemben nézve, minden építkezés mint a műszaki

Vírus hatásra eggy szoftverrendszer alapvető felületénél, de az alkendők felületénél is.

Függetlenül megrendezési nyelvből / paradigmával

GOF

# Nincs öröklési című szerint GOT

Létrehozói Minta $\Rightarrow$  Objektumok létrehozása

Szerkeszti Minta $\Rightarrow$  Hozzon alkotható öröklést és objektumot napellenőrökkel

Visszaolvási Minta $\Rightarrow$  Öröklés vezet objektumok egymára hatással, és felülvizsgálja lemezükre fogalmazik

## Létrehozású Minta

Címe

az objektumok felételeit függőlegesen az öröklési sorrendben, így ugyancsak a polycapsulation öröklési öröklésben különösen használunk

\* Builder\*, StringJoiner, Lambda

```
var s = new StringJoiner(", ", "[", "]")
    .add("John")
    .add("Paul")
    .add("George")
    .add("Ringo")
    .toString(); // "[John, Paul, George, Ringo]"
```

Egyéb

Egy öröklési vezet objektumt engedélyez és elérhetővé tenni konkréten a Java.lang.Runtime

## Díszítő

Ír objektumokhoz díszítésre való felhasználatra rendelhető  
Rugalmas alternatívája alacsonyabb elosztási költségekkel  
jár. I.e. Input/Output Stream

## Viselkedési Minta

### Bejárás

(folyékony eljárások alkalmából)

Összetett objektumok elemeinek soros eléréset biztosító módszer  
java · util · Iterator / Enumeration

## Sablonfigurák

Egy előre meghatározott szerűt látogatja el,  
melynek egyes lépései alacsonyabbak lehetnek, mint  
alacsonyabbak felülbirálhatók az algoritmus egyes lépései  
algoritmus szerkezetének megközelítése nélküli  
jár. I.e. Input/Output Stream, java · util · abstract \*

## Negligálás

Objektumok hozzájárhatók egy soh-soh figyelemi kapcsolat  
láthatóvá, így ha egy változó, több tud vala, frissül

# Programozási idiomok, Implementációs minták

Idiomok egy programozási nyelvre jellemző akarcsa szintű minták  
Idiomok a leggyakoribb orientáció minták.

Leírja hogyan valósítuk meg funkciókat és kapcsolatokat  
szemelhelyes adathalmazban

Legtöbb nyelvspecifikus

## Implementációs - Blok, Spaghetti code

Elvettető részletek, bármely szinten megjelenhetnek

Gyakran, összetévesztendő elosztású megoldások, melyek szerint

Hőszigetelés  $\Rightarrow$  Softverfejlesztés, Softver Architekturális  
Softverprojekt vezetési - Implementáció

Design Small / Térítési Snag  $\Rightarrow$  Térítési elvök megoldások - feladat

## Klassz - The Class - The Good Class

Címkefelülvilág: Metaprogrammázás

Ügyfélcímkei megoldásra: felhasználók címkefelülvilág

Ügyfélcímkei megoldásra: Softver

Kivállók: Létrehozás, törlés

Kiegészítőkönnyel: funkcionális teljesítmény és környezettség sorának

Architecturális részletek: "En az örököli az architekturális rész"

Felkölönést szegyél meg a csövély sajátítja ki  
szegyél konfolyt vezérlel csövély  
Ítéltőlök procedurális, lehet OOP  
Isteni kívánt részletek csövénél  
-> nélkülözhető coincidens  
-területi és lokális

### Típusok

Kura sor előre is metódus, melyet nem feltételez  
Paraméterek számához  
Hölgége memória által elérhető  
Sír konfolyt szerelelműekben vagy tervezésben

### Fizikai objektum

- OOP, illesztés bármelyik architektúra,
- Architektúra felhasználásával hozza

### Hersteller

Legacy Grid  
ejracsunogolásba

### Lijragyárak

Kodifizáció  
szereles

### Spaghetti kód

Csövökkel: alholónás  
Lijragyárakhoz: függőségek, hibatartás  
Lijragyárakhoz: szoftver  
Hersteller: tudatoság, lusterag  
Kiegészítőkkel: konfolytosság, vállalás hosszú  
Architecturesz: könyelbői részletek mint moduláris  
Seghinesellő címletek

## Altalánosan

Struktúrálatlak, néha címkékkel megjelölve,  
OOP esetén bárhol konték, metódusok implementációja  
nagyon hosszú

## Típusok

- Funkcionális OOP típusai
- Nincs mentőállás / hibakürtösvagalás

- Nincs funkciók implementációja
- Fejlesztésben együtt dolgozva

## Hibák

Az interfészeti hibák körülbeszélhetők, csak az implementáció szükséges

## Megoldás

Shédiárazszerű  
analízis, lásdoldyák

## 5. Tionta Gácl

# Erteletes reakciók

We leggen int x;  $\Rightarrow$  int elapsedTimeDays;

Not this

```
public List<int[]> getThem() {
    List<int[]> list1 = new ArrayList<int[][]>();
    for (int[][] x : theList)
        if (x[0] == 4)
            list1.add(x);
    return list1;
}
```

Vi kan nu tilføje en variable som er legget over

Leggenet a neek trijkhetsels, harschetsels

Hlossa heterosticha bigejör

Batályaló nevű fövvér, fövví hifjánével

Récoches de ravi-igels, igel Dépôts de

Leggiuntis hinc et hoc est sol, ne jocofaci hunc judicem

Helyonell a redact kontekstusla, nem mindighezhegy

11

State  
(allegat)

cedarState  
(állom)

We leggende felolges frontatusc th

# Függvények

Legyenek többek, soha 100 soros, nincs 20 soros  
általában 2-4 soros

Utasítások (for, if, switch) egy sorról töredékre -  
anek, általában függvénykés  $\Rightarrow$  Indent Level max 2

Egy dolgot minősítesz, azt jár

Argументumok  $\Rightarrow$  0-án ideális

- Niladic : 1
- Monadic : 1
- polyadic : 3+
- Dicadic : 2
- Tricadic : 3

Rinél több arg, amiket mindenki elvethető, megelőző  
a testtel

Né ha minél jobb felülvizsgálatot

3+ argumentum esetén minden lehet kontináltsági terület

Függvények legyenek mellékhatásmentesek, csak az  
minősítési rész igényel

Hozzá kell adni argumentumokat

Vagy minősítésre valódnak, vagy ugyanúgy viszont valódnak,  
egyébként a hozzájuk

Hilfreichek helyett fizetések, my catch előzetes rövid  
függvények

# Megjegyzések

Legjobb esetben is szükséges működés

Ha elég teljesen lesz a hálózatban minden, van lehet mégis előrelépések

Mert nem mindenki, mert nem mindenki nincs nem mindenhol, de minden

Hálózatban megjegyzést nem tudják használni

Pontatlan megjegyzés működés  
mint az nem lesz

Erdelzetesebb rendszert a hálózatban, mint megjegyzésekben

Törökoldalra arra, hogy ne legyen mégis előrelépések

## Jó megjegyzések

- Jogi megjegyzések
- Informatív megjegyzések
- Szándékot magyarázó megjegyzés
- Tisztázó megjegyzés
- Következményekre figyelmeztető megjegyzés
- TODO megjegyzés
- Megerősítő megjegyzés
- Javadoc megjegyzés nyilvános API-ban

## Nincs megjegyzés

- Szerző neve megjegyzésben
- Megjegyzésbe tett kód
- HTML megjegyzés
- Nem lokális megjegyzés
- Túl sok információt tartalmazó megjegyzés
- A kódhoz nem nyilvánvalóan kapcsolódó megjegyzés
- Javadoc megjegyzés nem nyilvános kódban

Redundancia M.  $\Rightarrow$  Nincs megjegyzés  
de mi a hálózat

Hatókörön M.  $\Rightarrow$  Kétféle módon megfordítva, ezt ezzel elkerülheti

Narancs M.  $\Rightarrow$  Működtetést nem tesznek le  $\Rightarrow$  Vérni hosszú rendszerek miatt fellesleges

# Forrásról formálás

formálás

Lépésről lépésre kell lépni  $\Rightarrow$  Cígyomű előállítása  
nincs hozzájárhatóság

## Függőleges formálás - újragondolás

Olyan mintegy újragondolás

- révén egyszerű, kezdődés
- elején magas szintű fogalmak, algoritmusok
- Lefelé haladva egyszerűsül, legutol a fogalmazásban több szint van

Egy összessor felül minden így fogalmazott

Szorosan összefüggő sorok  
nemtől nem legyen szüksége

Szorosan összefüggő fogalmak  
legyenek egymáshoz közel

Váltószókat alkalmazhat  
ahol használjuk által

Példányváltásokon keresztül eljárunk

ha egy függvényhez egy másikat, legyenek közötti összefüggések kétik a két funkció a két vállal

# Véronikai Formánk

Milyen hosszú legyen egy programozás? < 120 karakter

Gyengér összetörő elemek nincs száma  
(+, -, ... operaterek)

Nem minden alkalmazásban, esetlegesít, feljegyzést visszaírni igényeli

Fizelésű megfelelő leírásra

## Hilchenkélez

Hilchenkélez visszaadása helyett kiírások  
ellenőrzési nem ellenőrzési kiírások

Nem adjunk át vissza nullát

### Ellenorzött kiírások

Melodikus alkalmazásban, melyek behovethetők  
a végrehajtásra során  $\Rightarrow$  fordítás előtt ellenőrzi az, hogy  
a kiírások minden karaktere  
tartozik minden az ellenőrzők kiírásokhoz  
Íme négyeszerűsített programozás, hogy foglalkozzon a  
kiírásokkal, ha jo el ötöt  $\Rightarrow$  nincs megfelelőként  
írtíthetünk programhanyatlalnak címkell

### Nem ellenőrzött kiírások

Bárhol dolgozhatunk melódustársaság  
jártalmazási, ennek vonatkozásában

az ellenorölli kivételek megsérhetők az egyéghenőcist

↓

Ha egy elektromos nyílú metóduson horáciumbusz  
a hossz növekszik, az összesen ami itt kívja is  
horác zöll adja a migrációját