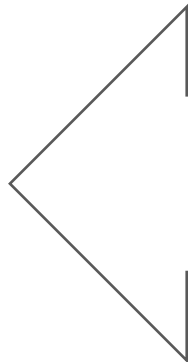


Gráf alapfogalmak,
szélességi
és mélységi keresés,
Dijkstra algoritmus
gyakorlat

Mirő lesz szó?

Gráfrepresentáció
elkészítése Javaban

Szélességi, mélységi és
Dijkstra algoritmusok
megvalósítása Javaban



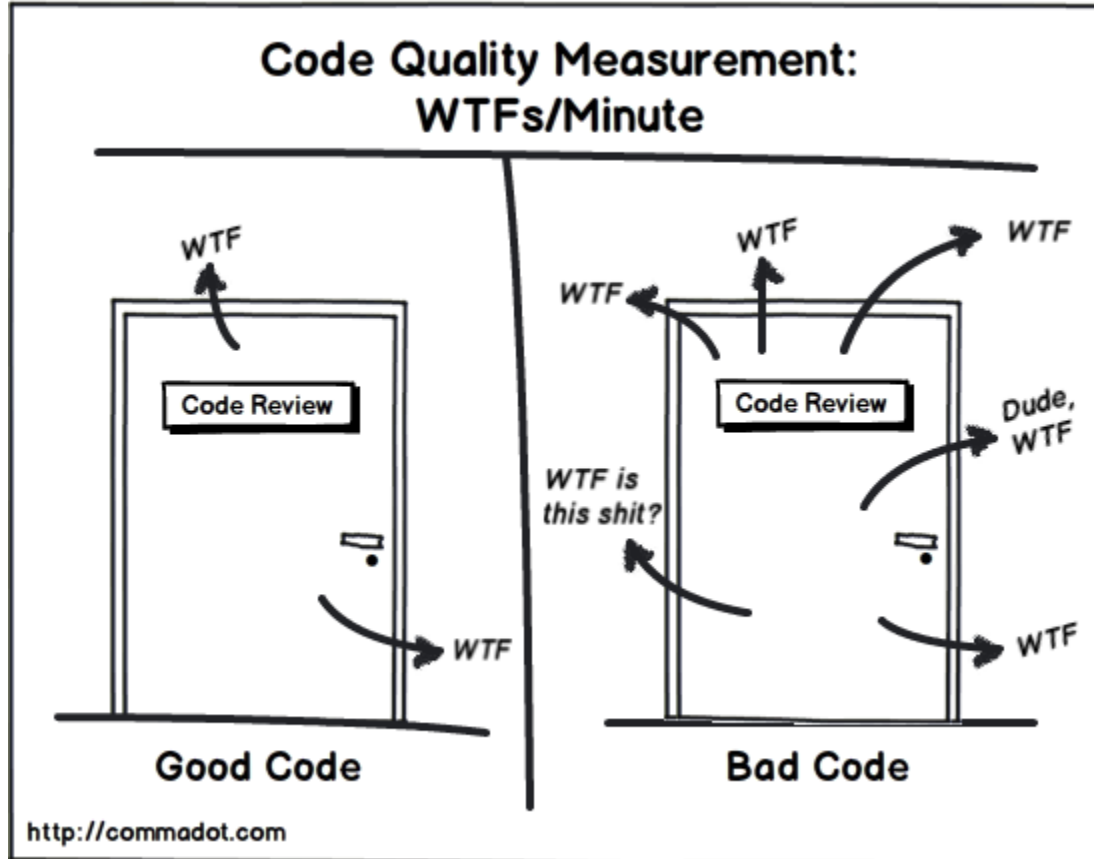
**Tiszta kóddal
&&
patternekkel**

**az igazán jó
kódért**

Mitől is jó egy kód?

- Mi az hogy “jósági tényező”?
- Számos metrika van a kód “jóságának” mérésére, melyek rengeteg tapasztalattal alakultak ki és sok paraméterrel dolgoznak (==nem a legegyszerűbb módja a kód jóságának megfogására)
- Fejlesztői szemmel, dióhéjban :
 - A kód akkor jó, ha olyan mint egy mese, amit csak olvasni kell.

A kód jóságának mértékegysége a WTFs/min.



Miért hasznos jó kódot írni?

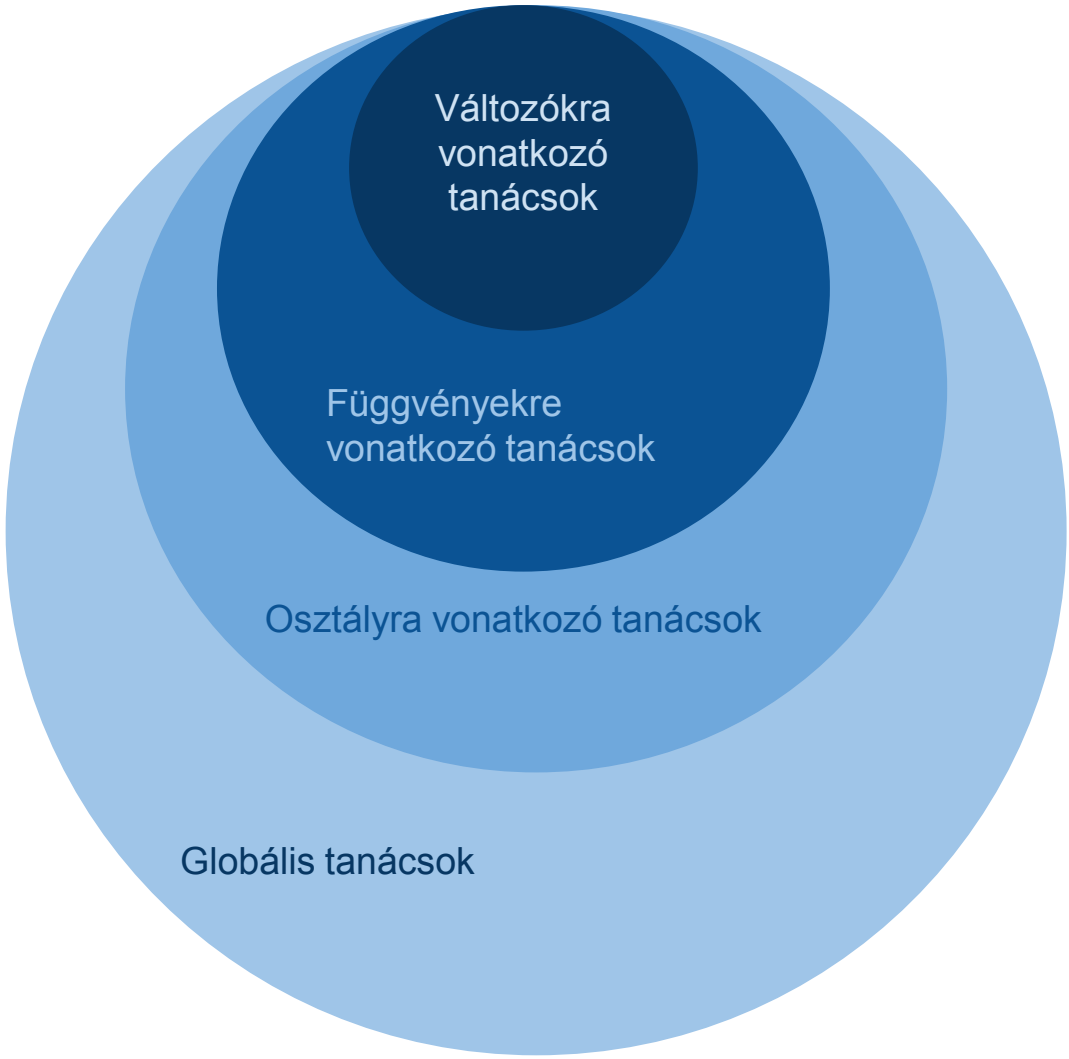
- karbantartható
- nem túlbonyolított
- könnyen továbbfejleszthető
- egységei újrafelhasználhatóak
- jól olvasható
- a rossz kód, hosszú távon a teljes projekt bukását okozhatja

Hogyan lehet jó kódot írni ?

- ezt is tanulni kell!
- él a köztudatban számos irányelv, melyet érdemes nekünk is követnünk:
 - Coding Principles (ld. később)
 - Design Patterns
 - szinte kötelező olvasmányok:
 - **Robert C. Martin: Clean code** (<https://cleancoders.com/landing>)
 - **Joshua Bloch: Effective Java**

Coding principles,
vagyis
kódolási etikett





Változókra
vonatkozó
tanácsok

Függvényekre
vonatkozó tanácsok

Osztályra vonatkozó tanácsok

Globális tanácsok

Példa

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

```
public list<int[]>  
    getFlaggedCells () {  
  
        List<int[]> flaggedCells= new  
ArrayList<int[]>();  
  
        for (int[]cell :gameBoard)  
  
            if (cell[STATUS_VALUE]  
                ==FLAGGED)  
  
                flaggedCells.add(cell);  
  
        return flaggedCells;  
  
}
```

Változókra vonatkozó tanácsok

- következetes, beszédes, logikus (pl: negáció negálása: bad smell), rövid de értelmes (nem rövidítés), kereshető, nem rokonértelmű nevek főnevekből
- fontosabb az olvashatóság, mint a takarékoság
- valódi megkülönböztetés az elnevezésekben (+változó elfedést kerüljük)
- a legkisebb még megfelelő típust használjuk (long helyett int is általában elég pl: egy számlálóhoz)
- osztályváltozók deklarációja legyen felül (javában lehet előbb használni, mint deklarálni, de kusza lesz)
- magic numberból konstanst (nagybetű)

Függvényekre vonatkozó tanácsok

- nevük igékből álló kifejezés
- túl hosszú, bonyolult, többször szereplő kifejezés: legyen függvény (azért ne változó, mert a fv értéke mindig friss)
- olvasás felülről lefelé
- kevés paraméter (vagy paraméter objektum, varargs)
- egy függvény max 5 sor és egy feladat végrehajtása (ha több, biztos kettéosztható)
- get,set,is
- paramétert, ha lehet ne módosítsunk (final ==inmutable)
- nullt ne adjunk vissza (exceptionok nagy része nullptrexc, java 8-optional)

Osztályokra vonatkozó tanácsok

- **Single responsibility:** egyetlen, jól szeparálható feladatkörre, funkcióra összpontosító osztályok
- **„for the sake of DRYness”:** modularitás, újrafelhasználhatóság
- osztályváltozó, vagy paraméter?:
 - a. ha több helyen van függvényben használva, érdemes az osztályváltozó
 - b. Immutable objektumok (értékük nem fog változni)
 - c. Belső változóink (classfields) védelme (private láthatóságú +getter hozzá)
- Konstruktor
 - a. ha nincs megírva, akkor is generálódik default
 - b. a többféleképpen definiált szükséges-e? (overloading= túlterhelés típus alapján)
 - c. super hívása

Osztályokra vonatkozó tanácsok

- Öröklődés:
 1. Öröklődés és enkapszuláció közötti különbség
 2. Ősosztály, az absztrakció célja: közös tulajdonságok és viselkedés egy helyre mozgatása
 3. diamond öröklődés veszélyei
 4. Absztrakt osztály (leszármaztatott osztály extendálja)
 5. Interfész osztály (egy osztály implementálhat többet is)

Globális vonatkozású tanácsok

- Egy probléma amibe belefutunk, valószínű más is belefutott már: leggyorsabb, legbiztosabb megoldás a neten utánakeresés
- For ciklus helyett foreach vagy stream (Iterable interface megvalósítása miatt lehet)
- Autoboxing és unboxing
- Egyenlőségek(érték illetve referencia alapján: == vs. equals)
- Array->raw ArrayList->generic ArrayList<> ->HashSet

Coding Principles mozaikszavak

S.O.L.I.D.

Single responsibility principle: egy osztály egy dolgot csinál

Open close principle: bővítésre nyitott, módosításra zárt

Liskow substitution principle: egy fajta objektum a leszármazott példányával helyettesíthető kell legyen

Interface segregation principle: sok kliens-specifikus interfész jobb, mint egy nagy általános

Dependency inversion principle: ősz osztályok határozzák meg a fő funkcionalitást és nem pedig fordítva
(gyerek osztály csak specializációra való)

További elterjedt mozaikszavak:

P.O.L.S.: **P**rinciple **O**f **L**east **S**urprise: kiszámítható kód (főleg az elnevezések esetén fontos)

K.I.S.S.: **K**ee**P** **I**t **S**hort, **S**imple: Próbáld mindig a legegyszerűbb megvalósítást választani, ne készíts direkt rejtvényeket, vagy dicsekedj a kódod felesleges bonyolultságával (, hogy te képes vagy ilyen bonyolultan is megírni a megoldást). Ha túlbonyolítjuk, akkor ugyanaz a funkcionalitás nehezebben lefedhető tesztekkel, nem karbantartható, nehezebben bővíthető és kevésbé olvasható

D.R.Y.: **D**on't **R**epeat **Y**ourself : moduláris, újrafelhasználható kódok készítése a cél: pl:private metódusokba kiszervezés, ha ugyanaz a kódrészlet többször is előfordul. DE! mikor éri meg? (lehet hogy egyetlen sor miatt már nem)

T.M.T.O.W.T.D.I.: **T**here's **M**ore **T**han **O**ne **W**ay **T**o **D**o **I**t: egy problémát több oldalról meg lehet közelíteni és ugyanolyan jó megoldást lehet rá találni. Egymás kódjának átvizsgálásával sokat lehet fejleszteni szemléletmódunkon.

Szorgalmi plusz pontért a repóban!

Köszönjük a figyelmet!