Objektumorientált szoftvertervezés - 3. előadás
Refaktorálás

Mi a refaktorálás?

- definíció:
  - **refaktorálás**: a külsőleg megfigyelhető viselkedés megtartásával a szoftver belső struktúrájában végrehajtott olyan változtatás, amely által a szoftver kódja könnyebben érthetővé és olcsóbban módosíthatóvá válik
  - **refaktorálni**: szoftvert átstruktúrálni a külsőleg megfigyelhető viselkedés módosítása nélkül!
- mikor érdemes refaktorálni?
  - új funkció hozzáadásakor, bug javításakor, kódszemle alkalmával
- refaktorálás szabályai:
  - legyen alapos tesztkészlet (külső viselkedés nem változott e), kis lépésenként történjen, merjünk változtatni

Refaktorálás lépései

1. Legyen alapos tesztkészlet
2. A régi kód menjen át a teszteken
3. Apró változtatás végrehajtása
4. Az új kód is menjen át a teszteken
5. Ismétlés az 1. lépéstől

Refaktorálás tulajdonságai

- előnyök: jobb tervek, olvashatóbb és érthetőbb kód, kevesebb bug, gyorsabb fejlesztés
- nehéz refaktorálni: adatbázis, interfészek (stabilak, ritkán változnak, sokak függenek tőlük)

Code smell (Büdös kód)

- A **code smell** egy olyan felületi tünet, amely valamely mélyebb tervezési problémát jelez (Martin Fowler)
- a code smell NEM bug, de nehezíti a további fejlesztést

Tünetek:

1. **Duplikált kód**: ugyanaz vagy nagyon hasonló kód ismétlődik több helyen
   - refaktorálás: külön függvénybe/osztályba kiemelés
2. **Hosszú metódus**: függvény kódja túl hosszú: túl sok feltételes ág illetve ciklus
   - refaktorálás: feldarabolás kommentek, feltétel elágazások szerint
3. **Hosszú paraméterlista**: a függvényne ktúl sok (háromnál több) paramétere van
   - refaktorálás: több függvényre bontás, paraméterek egy objektumba vonása
4. **Nagy osztály**: az osztálynak túl sok metódusa van (SRP és ISP sérül)
   - refaktorálás: több osztályra darabolás, ISP használata, öröklési hierarchia készítése

5. **Divergent change**: technológiánként más és más osztály
    - refaktorálás: közös részek stabil absztrakt csomagba szervezése
6. **Shotgun surgery**: egy változtatás sok más osztályban apró változtatásokat indukál (pl konstansok beégetésekor, DRY sérül)
    - refaktorálás: ezek elnevezése, egy helyre gyűjtése, akár erőforrás fájlba kiszervezése (így többnyelvűség támogatása pl)
7. **Feature envy**: egy osztály túlságosan érdeklődik egy másik iránt (túl nagy a csatolás köztük)
    - refaktorálás: szerverből átrakjuk a gyakran hívott függvényeket a kliensbe (nagy kohézió, kis csatolás)
8. **Data clumps**: ugyanaz a paramétercsoport ismétlődik több metódushíváson keresztül
    - refaktorálás: nem OO alapon terveztünk, ezeket osztály mezőibe építsük be
9. **Primitive obsession**: adatok primitív típusokban vannak tárolva osztályok helyett
    - refaktorálás: nem OO alapon terveztünk, ezeket szervezzük osztályokba (akár öröklést is bevethetünk)
10. **Switch statements**: túl sok elágazás (sok if-else, null ellenőrzés, nagy switch)
    - refakotorálás: polimorf viselkedés, Null Object tervezési minta
11. **Öröklési hirearchiák**: ha egy osztályból új leszármazott készül, akkor egy másikból is kell leszármazottat készíteni (shotgun surgery egy példája)
    - refaktorálás: egy osztályba vonjuk
12. **Lusta osztály (lazy class)**: az osztály túl kevés dolgot csinál
    - refaktorálás: az ilyen osztályt érdemes törölni
13. **Spekulatív általánosság**: az osztály feleslegesen túl nehézsúlyú (túl bonyolult dolgokra készültünk, YAGNI elv sértése)
    - refaktorálás: eliminájuk a felesleges dolgokat
14. **Temporarí field (ideiglenes mező)**: egy attribútum csak bizonyos esetekben van használva (pl globális változók, alacsony osztályon belüli kohézió, objektum egy szálon használható csak)
    - refaktorálás: külön osztályba szervezzük ki ezeket (így nagy lesz a kohéziója)
15. **Message chains**: hosszú metódushívási láncok (demeter törvényt sérti (LoD))
    - refaktorálás: függvények osztályok közti átmozgatása (felelősségek újragondolása), lánc maradékát delegáló függvénybe rejtjük (csak közvetlen szomszédokat ismerik, de elszaporodhatnak a delegálók)
16. **Middle man**: túl sok delegálás egy másik osztály felé (az adott osztálynak csak közvetítő szerepe van)
    - refaktorálás: ezen osztály eliminálása, hívjuk közvetlenül a célosztályt
17. **Inappropriate intimacy**: egy másik osztály privát (nempublikus) tagjainak közvetlen elérése (egységbezárás sérül, magas lesz a csatolás)
    - refaktorálás: felelősségek újragondolása, metódusok átrendezése a két osztály közt
18. **Alternative classes with different interfaces**: ugyanarra a feladatra különböző interfészű osztályok
    - refaktorálás: egyes függvények átnevezése, közös interfészre jutás (Bridge tervezési minta hasznos)
19. **Incomplete library class**: egy felhasznált könyvtárbeli osztályt nem tudunk módosítani
    - refaktorálás: könyvtárbeli osztályt saját osztályba csomagoljuk, delegálással új funkciót adunk neki
20. **Data class**: egy osztály csak adatokat tárol
    - refaktoráls: nem OO módon terveztünk, vigyünk felelősségeket az adatosztályba

21. **Refused bequest (elutasított örökség)**: egy leszármazottnak nincs szüksége az ős viselkedésére (ősben definiált viselkedés túl magasra került a hierarchiában, vagy üresen definiálja felül a leszármazott az ős metódusát (LSP megsértése), gyerekben egyáltalán nem releváns az ős interfésze)
    - refaktorálás: függvényt és hozzátartozó attribútumokat vigyük lejjebb az öröklési hierarchiában, vagy átrendezzük az öröklési hierarchiát (ős és leszármazott felcserélése), vagy öröklődés újragondolása (esetleg delegálással helyettesítjük)
22. **Kommentek**: túl sok magyarázó komment a kódban
    - refaktorálás: miért csinálja azt amit (ne azt írja le, hogy MIT csinál hanem hogy HOGYAN)
23. **Downcasting**: típuskasztolás és típusellenőrzés (nem használjuk ki a polimorfizmust, OCP sérül, TDA-t is sérthetjük)
    - refaktorálás: polimorfizmussal megoldhatjuk

Refaktorálási technikák

1. **Extract method**: kód egy részéből függvény
2. **Inline method**: függvény törzsének beépítése a hívó oldalán
3. **Inline temporary**: egyszer használt ideiglenes változó értékének beégetése a használat helyére
4. **Replace temporary with query**: ideiglenes változó helyett függvény bevezetése
5. **Introduce explaining variable**: bonyolultabb kifejezés helyettesítése olvasható nevű változóval
6. **Split temporary variabla**: egy ideiglenes változó több különböző célra használatakor az ideiglenes változó szétválasztása 2 változóvá találó névvel
7. **Remove assignments to parameters**: ideiglenes változó bevezetése paraméternek való értékadás helyett
8. **Replace method with method object**: metódusból osztályt készítünk, ahol a lokális változókból mezők lesznek
9. **Substitute algorithm**: algoritmus tisztább változatra cserélése
10. **Move method**: metódust másik osztályba mozgatunk
11. **Move field**: mezőt másik osztályba mozgatunk
12. **Extract class**: osztály néhány metódusából és mezőjéből egy másik osztályt képzünk
13. **Inline class**: egy osztályt készítünk több osztályból (extract inverze)
14. **Hide delegate**: távoli objektumhoz intézett hívást delegációval intézünk
15. **Remove middle man**: delegációt egy direkt hívásra cseréljük le
16. **Introduce foraign method**: kliens osztályban új metódust készítünk, aminek a szerver paramétere (kliensből, a szerver által nem támogatott funkció adása a szervernek)
17. **Introduce local extension**: szerver bővítése leszármazott létrehozásával
18. **Self encapsulate field**: mezőt getter-setter metódusokba vagy property-be csomagolunk
19. **Replace data value with object**: egyszerű primitív adatokat tároló változók helyett osztály bevezetése
20. **Change value to reference**: értékegyenlőség helyett referenciaegyenlőséget vezet be
21. **Change reference to value**: referenciaegyenlőség helyett értékegyenlőséget vezet be
22. **Replace array with object**: tömb helyett osztály használata az összetartozó elemek csoportosítására
23. **Duplicate observed data**: GUI-ban ne tároljunk a modell számára fontos adatokat (GUI függjön a modelltől és ne fordítva)
24. **Change unidirectional association to bidirectional**: egyirányú asszociáció helyett kétirányú asszociáció
25. **Changed bidirectional association to unidirectional**: kétirányú asszociáció helyett egyirányú asszociáció

26. **Replace magic numbers with symbolic constant**: kódban szétszórt számok és string iterálok helyett nevesített konstansok használata
27. **Encapsulate field**: nem privát attribútumból privát készítése, és getter-setter vagy property készítése hozzá
28. **Encapsulate collection**: gyűjtemény típusú attribútum elrejtése, kifele csak olvashatóvá tétel, módosítás függvényeken keresztül
29. **Replace record with data class**: nem OO könyvtárhoz csatlakozás (rekordok helyett adat osztályok használata)
30. **Replace type code with class**: típusok intként vagy enumként tárolása helyett osztályokat használjunk
31. **Replace type code with subclasses** típusok intként vagy enumként tárolása helyett osztályokat és öröklődést is használjunk
32. **Replace type code with state/strategy**: objektum dinamikus viselkedésének kiszervezése state vagy strategy mintával
33. **Replace subclass with fields**: ha a leszármazottak semmi hasznosat nem csinálnak, így mezővel helyettesíthetőek
34. **Decompose conditional**: feltételes elágazások ágainak külön függvényként reprezentálása
35. **Consolidate conditional expression**: egy feltétel többszöri ellenőrzése helyett egyszer legyen kiértékelve és letesztelve
36. **Consolidate duplicate conditional fragments**: egy részkifejezés többszöri ellenőrzése helyett egyszer legyen kiértékelve és letesztelve
37. **Remove control flag**: kilépést jelző változó break vagy return utasításra
38. **Replace nested conditional with guard clauses**: egymásba ágyazott feltételek helyett lapos feltételsorozat (egymás utáni if-ek inkább)
39. **Replace conditional with polymorphism**: típusellenőrzés helyettesítése polimorfizmussal
40. **Introduce null object**: ne null értékeket tároljunk, hanem Null Object mintát
41. **Introduce assertion**: feltételellenőrzés a kód egy adott pontján, ami debug módban hibát dob ha a feltétel sérül (debuggolásra)
42. **Rename method**: függvény találóbb névre nevezése
43. **Add parameter**: új paraméter a függvénynek
44. **Remove paramtere**: nem használt paraméter törlése
45. **Separate query from modifier**: 2 metódus készítése, egy lekérdezésre, egy módosításra
46. **Parametrize method**: két nagyon hasonló függvény kombinálása eggyé
47. **Replace parameter with explicit methods**: egy függvényből több függvény készítése
48. **Preserve whole object**: teljes objektum átadása paraméterként, nem csak bizonyos részeit
49. **Replace parameter with method**: nem egy metódushívás eredményét adjuk át paraméterként, hanem hagyjuk, hogy ezt a hívást a függvény saját maga intézze el
50. **Introduce parameter object**: sok csoportosult paraméter helyettesítése önálló osztállyal
51. **Remove setting method**: attribútumok értékét konstruktorban állítjuk be, settereket nem bzitosítunk
52. **Hide method**: senki által nem használt publikus függvényt tegyük priváttá
53. **Replace constructor with factory method**: objektum létrehozása és inicializálása bonyolultabb egy egyszerű konstruktor hívásnál
54. **Encapsulate downcast**: metódus eredményét a klienseknek folyton át kell cast-olniuk, akkor ezt a típus kasztolást tegyük át a metódusba és a metódus térjen vissza a helyes típussal
55. **Replace error code with exception**: hibakód és speciális visszatérés helyett használjunk kivételeket
56. **Replace exception with test**: kivételek elkapása helyett ellenőrizzük a beadott paramétereket
57. **Pull up field**: leszármazottakban levő közös attribútumokat a közös ősbe visszük át
58. **Pull up method**: leszármazottakban levő közös metódusokat a közös ősbe visszük át

59. **Pull up constructor body**: leszármazottak konstruktorainak közös részeit viszik fel a közös ős konstruktorába
60. **Push down method**: a csak a leszármazottak egy részére érvényes metódust lejjebb visszük az öröklési hierarchiában
61. **Push down field**: a csak a leszármazottak egy részére érvényes mezőt lejjebb visszük az öröklési hierarchiában
62. **Extract subclass**: azokat a viselkedéseket amik csak a leszármazottak egy részére érvényesek, egy önálló osztályba visszük ki, az eredeti leszármazottak tőle fognak mostantól leszármazni
63. **Extract superclass**: ha néhány leszármazott egy közös viselkedéssel rendelkezik, ezt a viselkedést egy önálló közös ősbe lehet kiszervezni
64. **Extract interface**: néhány osztály publikus interfészében vannak közös részek, akkor az a közös halmaz egy közös interfészbe kiszervezhető
65. **Collapse hierarchy**: ha egy leszármazott osztály nem sokat ad az ős viselkedéséhez, akkor azt összevonjuk az őssel
66. **Form template method**: ha néhány osztály hasonló metódusokat tartalmaz hasonló lépésekkel azonos sorrendben, akkor a template method mintával vigyük fel őket az ősbe
67. **Replace inheritance with delegation**: ha egy leszármazottnak nem minden ősből örökölt dologra van szüksége, akkor használjunk inkább delegációt
68. **Replace delegation with inheritance**: ha egy osztály csak delegál egy hasonló viselkedésű osztályhoz, akkor használjuk inkább öröklődést
69. **Tease apart inheritance**: ha egy öröklési hierarchia 2 dolgot csinál egyszerre, vágjuk 2 részre
70. **Convert procedural design to objects**: adatosztályokhoz felelősségeket (metódusokat) rendelünk
71. **Separate domain from presentation**: üzleti logika kivétele a GUI-ból, hozzunk létre önálló csak GUI-val foglalkozó osztályokat
72. **Extract hierarchy**: bonyolult osztály feldarabolása komplex öröklési hierarchiára