Vonatkozások szétválasztása (3)

- A vonatkozás valami olyan, ami érdekes vagy fontos egy érintett vagy érintettek egy csoportja számára.
  - Például: teljesítmény, adott funkció biztosítása, karbantarthatóság, …
- A rendszerkövetelményeket tükrözik.

Vonatkozások szétválasztása (4)

- Vonatkozások fajtái:
  - **Alapvető vonatkozások** (***core concerns***): a rendszer elsődleges céljához kötődő funkcionális vonatkozások.
  - **Másodlagos vonatkozások** (***secondary concerns***): például a rendszer nem funkcionális követelményeinek kielégítéséhez szükséges funkcionális vonatkozások.
  - **Átszövő vonatkozások** (***cross-cutting concerns***): alapvető rendszerkövetelményeket tükröző rendszerszintű vonatkozások.
    - A másodlagos vonatkozások lehetnek átszövőek is, bár nem minden esetben szövik át az egész rendszert.
    - Például: biztonság, naplózás.

Vonatkozások szétválasztása (5)

- Kapcsolódó programozási paradigma: aspektus-orientált programozás (AOP – *aspect-oriented programming*)
  - Például: *AspectJ* https://eclipse.org/aspectj/
    - A Java programozási nyelv aspektus-orientált kiterjesztése.

GoF alapelvek (1)

- Felhasznált irodalom:
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.
  - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Programtervezési minták: Újrahasznosítható elemek objektumközpontú programokhoz*. Kiskapu, 2004.

GoF alapelvek (2)

- **Interfészre programozzunk, ne implementációra!**
  - *„Program to an interface, not an implementation.”*
- Lásd a létrehozási mintákat!

GoF alapelvek (3)

- **Részesítsük előnyben az objektum-összetételt az öröklődéssel szemben!**
  - *„Favor object composition over class inheritance.”*
- A két leggyakoribb módszer az újrafelhasználásra az objektumorientált rendszerekben:
  - Öröklődés (fehér dobozos újrafelhasználás)
  - Objektum-összetétel (fekete dobozos újrafelhasználás)
- A fehér/fekete dobozos jelző a láthatóságra utal.

GoF alapelvek (4)

- A tervezési minták két fajtája hatókör szerint:
  - **Osztályminták**: az osztályok és alosztályaik közötti viszonyokkal foglalkoznak.
    - Ezek a kapcsolatok öröklés lévén jönnek létre, ezért statikusak, azaz fordításkor kerülnek rögzítésre.
  - **Objektumminták**: az objektumok közötti kapcsolatokkal foglalkoznak, melyek futásidőben változhatnak, ezért dinamikusabbak.

GoF alapelvek (5)

- A tervezési minták két fajtája hatókör szerint (folytatás):

Hatókör/Cél	Létrehozási	Szerkezeti	Viselkedési
Osztály	• Gyártó metódus	• (Osztály) adapter	• Értelmező • Sablonfüggvény
Objektum	• Elvont gyár • Építő • Prototípus • Egyke	• (Objektum) adapter • Híd • Összetétel • Díszítő • Homlokzat • Pehelysúlyú • Helyettes	• Felelősséglánc • Parancs • Bejáró • Közvetítő • Emlékeztető • Megfigyelő • Állapot • Stratégia • Látogató

GoF alapelvek (6)

- Az öröklődés előnyei:
  - Statikusan, fordítási időben történik, és használata egyszerű, mivel a programozási nyelv közvetlenül támogatja.
  - Az öröklődés továbbá könnyebbé teszi az újrafelhasznált megvalósítás módosítását is.
    - Ha egy alosztály felülírja a műveletek némelyikét, de nem mindet, akkor a leszármazottak műveleteit is megváltoztathatja, feltételezve, hogy azok a felülírt műveleteket hívják.

GoF alapelvek (7)

- Az öröklődés hátrányai:
  - Először is, a szülőosztályoktól örökölt megvalósításokat futásidőben nem változtathatjuk meg, mivel az öröklődés már fordításkor eldől.
  - Másodszor – és ez általában rosszabb –, a szülőosztályok gyakran alosztályaik fizikai ábrázolását is meghatározzák, legalább részben.
    - Mivel az öröklődés betekintést enged egy alosztálynak a szülője megvalósításába, gyakran mondják, hogy az öröklődés megszegi az egységbe zárás szabályát.
    - Az alosztály megvalósítása annyira kötődik a szülőosztály megvalósításához, hogy a szülő megvalósításában a legkisebb változtatás is az alosztály változását vonja maga után.
  - Az implementációs függőségek gondot okozhatnak az alosztályok újrafelhasználásánál.
    - Ha az örökölt megvalósítás bármely szempontból nem felel meg az új feladatnak, arra kényszerülünk, hogy újraírjuk, vagy valami megfelelőbbel helyettesítsük a szülőosztályt. Ez a függőség korlátozza a rugalmasságot, és végül az újrafelhasználhatóságot.

GoF alapelvek (8)

- Az objektum-összetétel dinamikusan, futásidőben történik, olyan objektumokon keresztül, amelyek hivatkozásokat szereznek más objektumokra.
- Az összetételhez szükséges, hogy az objektumok figyelembe vegyék egymás interfészét, amihez gondosan megtervezett interfészek kellenek, amelyek lehetővé teszik, hogy az objektumokat sok másikkal együtt használjuk.

GoF alapelvek (9)

- Az objektum összetétel előnyei:
  - Mivel az objektumokat csak az interfészükön keresztül érhetjük el, nem szegjük meg az egységbe zárás elvét.
  - Bármely objektumot lecserélhetünk egy másikra futásidőben, amíg a típusaik egyeznek.
  - Továbbá, mivel az objektumok megvalósítása interfészek segítségével épül fel, sokkal kevesebb lesz a megvalósítási függőség.
  - Az öröklődéssel szemben segít az osztályok egységbe zárásában és abban, hogy azok egy feladatra összpontosíthassanak.
  - Az osztályok és osztályhierarchiák kicsik maradnak, és kevésbé valószínű, hogy kezelhetetlen szörnyekké duzzadnak.

GoF alapelvek (10)

- Az objektum összetétel hátrányai:
  - Másrészről az objektum-összetételen alapuló tervezés alkalmazása során több objektumunk lesz (még ha osztályunk kevesebb is), és a rendszer viselkedése ezek kapcsolataitól függ majd, nem pedig egyetlen osztály határozza meg.

SOLID (1)

- Robert C. Martin („Bob bácsi”) által megfogalmazott/rendszerezett/népszerűsített objektumorientált programozási és tervezési alapelvek.
  - Blog: https://blog.cleancoder.com/
  - https://github.com/unclebob
  - Uncle Bob. *Getting a SOLID start*. 2009. https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start
- Felhasznált irodalom:
  - Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Pearson Education, 2002.
    - C++ és Java nyelvű programkódok.
  - Robert C. Martin, Micah Martin. *Agile Principles, Patterns, and Practices in C#*. Prentice Hall, 2006.

SOLID (2)

- ***S****ingle Responsibility Principle* (SRP) – Egyszeres felelősség elve
- ***O****pen/Closed Principle* (OCP) – Nyitva zárt elv
- ***L****iskov Substitution Principle* (LSP) – Liskov-féle helyettesítési elv
- ***I****nterface Segregation Principle* (ISP) – Interfész szétválasztási elv
- ***D****ependency Inversion Principle* (DIP) – Függőség megfordítási elv

SOLID – egyszeres felelősség elve (1)

- Robert C. Martin által megfogalmazott elv:
  - *„A class should have only one reason to change.”*
  - Egy osztálynak csak egy oka legyen a változásra.
- Kapcsolódó tervezési minták: díszítő, felelősséglánc