SOLID (1)

- Robert C. Martin („Bob bácsi”) által megfogalmazott/rendszerezett/népszerűsített objektumorientált programozási és tervezési alapelvek.
  - Bob bácsi honlapja: http://cleancoder.com/
  - Uncle Bob. *Getting a SOLID start*. 2009. https://sites.google.com/site/unclebobconsultingllc/getting-a-solid-start
- 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)

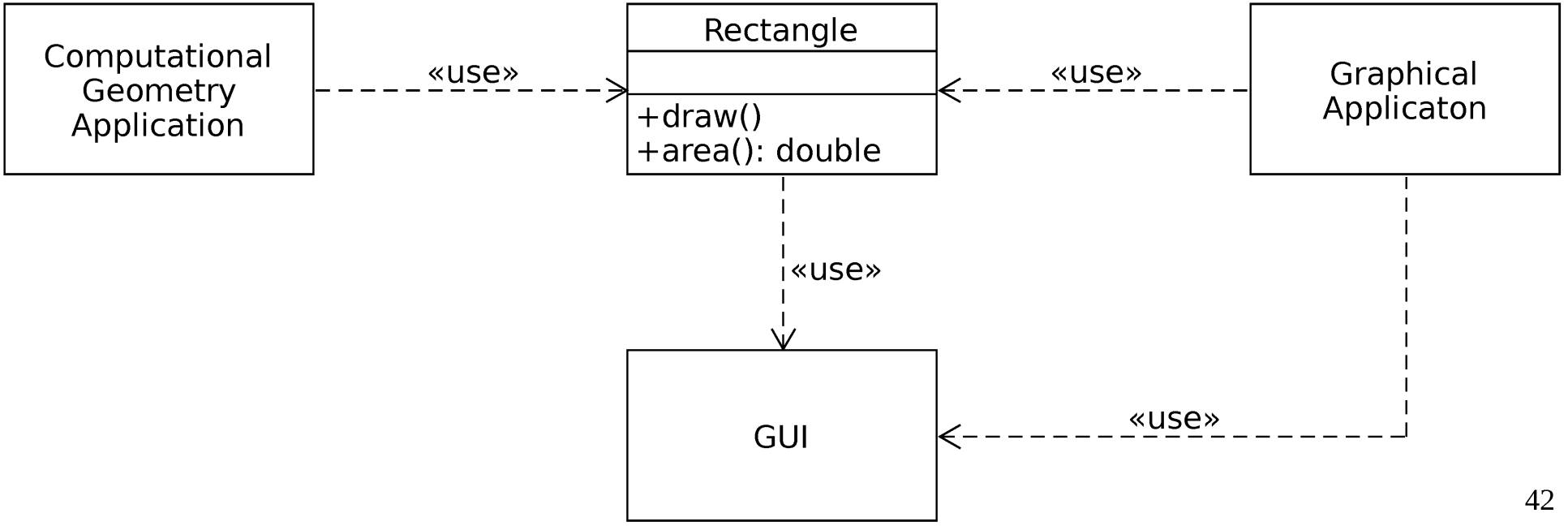
- Robert C. Martin által megfogalmazott elv:
  - *„A class should have only one reason to change.”*
  - Egy osztálynak csak egy oka legyen a változásra.
- Kapcsolódó tervezési minták: díszítő, felelősséglánc

SOLID – egyszeres felelősség elve (2)

- Egy felelősség egy ok a változásra.
- Minden felelősség a változás egy tengelye. Amikor a követelmények változnak, a változás a felelősségben történő változásként nyilvánul meg.
- Ha egy osztálynak egynél több felelőssége van, akkor egynél több oka van a változásra.
- Egynél több felelősség esetén a felelősségek csatolttá válnak. Egy felelősségben történő változások gyengíthetik vagy gátolhatják az osztály azon képességét, hogy eleget tegyen a többi felelősségének.

SOLID – egyszeres felelősség elve (3)

- Példa az elv megsértésére:
  - A Rectangle osztály két felelőssége:
    - Egy téglalap geometriájának matematikai modellezése.
    - Téglalap megjelenítése a grafikus felhasználói felületen.

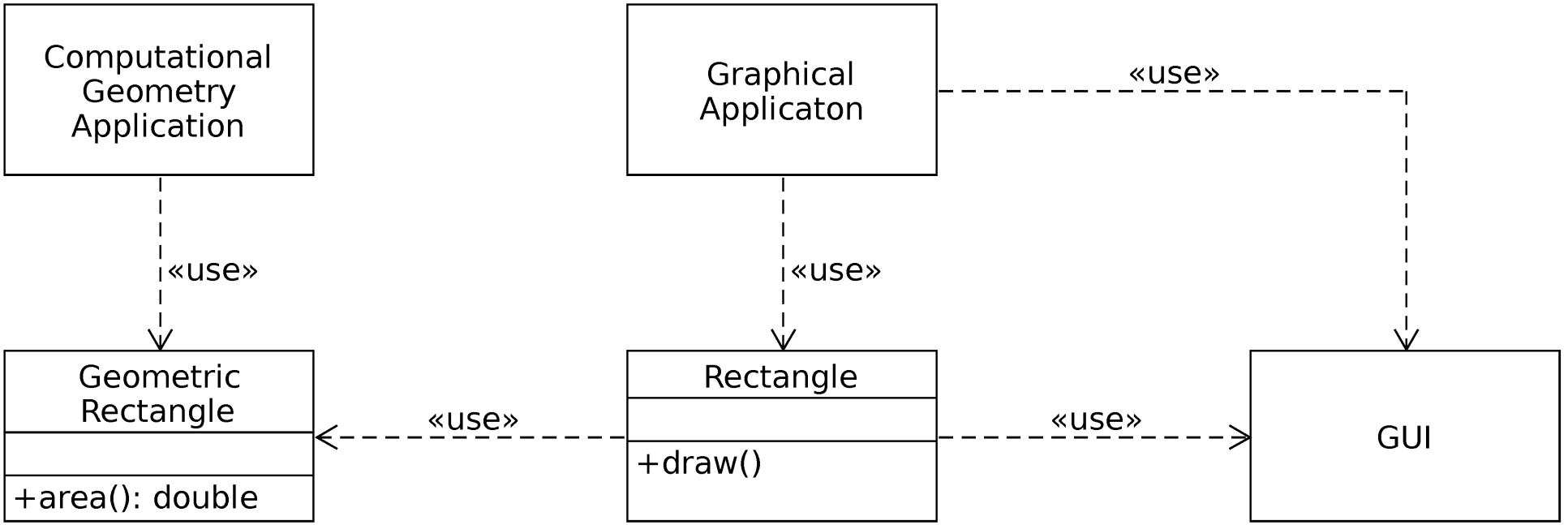


SOLID – egyszeres felelősség elve (4)

- Példa az elv megsértésére: (folytatás)
  - A számítógépes geometriai alkalmazásnak tartalmaznia kell a grafikus felhasználói felületet.
  - Ha a grafikus alkalmazás miatt változik a `Rectangle` osztály, az szükségessé teheti a számítógépes geometriai alkalmazás összeállításának, tesztelésének és telepítésének megismétlését (*rebuild*, *retest*, *redeploy*).

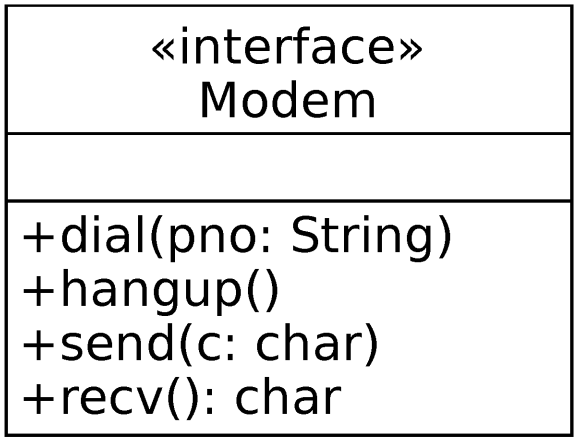
SOLID – egyszeres felelősség elve (5)

- Az előbbi példa az elvnek megfelelő változata:



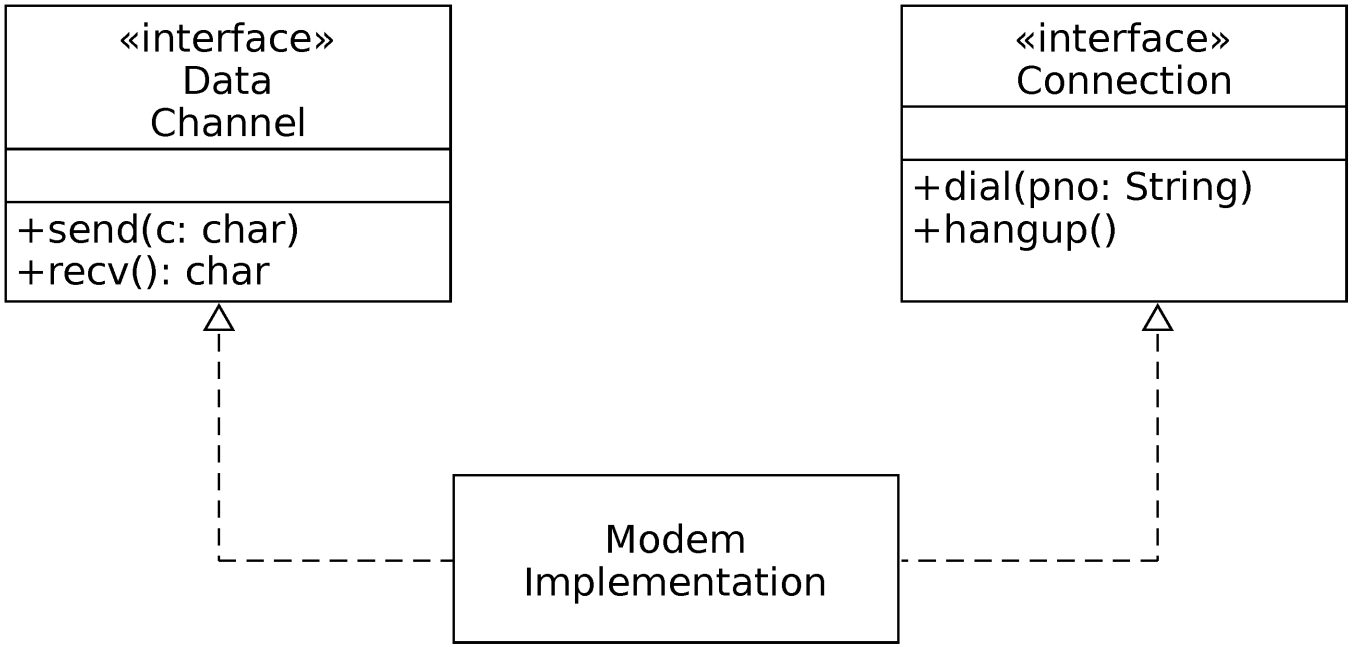
SOLID – egyszeres felelősség elve (6)

- Példa: Mi a felelősség?
  - Az alábbi `Modem` interfésznél két felelősség állapítható meg: a kapcsolatkezelés és az adatkommunikáció.
  - Hogy érdemes-e őket szétválasztani, az attól függ, hogyan változik az alkalmazás.



SOLID – egyszeres felelősség elve (7)

- Példa: Mi a felelősség? (folytatás)
  - Ha például úgy változik az alkalmazás, hogy az hatással van a kapcsolatkezelő függvények szignatúrájára, akkor a két felelősséget szét kell választani.



SOLID – egyszeres felelősség elve (8)

- Az elv megfogalmazásának finomodása:
  - *„A **class** should have only one reason to change.”*
    - Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Pearson Education, 2002. p. 95.
  - *„… a **class or module** should have one, and only one, reason to change.”*
    - Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008. p. 138.

SOLID – egyszeres felelősség elve (9)

- A vonatkozások szétválasztásának elve és az egyszeres felelősség elve szorosan összefügg. Így a felelősségek befoglaló halmazát alkotják a vonatkozások.
- Ideális esetben minden vonatkozás egy felelősségből áll, mégpedig a fő funkció felelősségéből. Azonban egy felelősségben gyakran több vonatkozás is keveredik.
- A vonatkozások szétválasztásának elve azt nem mondja ki, hogy egy felelősség csak egy vonatkozásból állhat, hanem csak annyit követel meg, hogy a vonatkozásokat el kell különíteni egymástól, vagyis tisztán felismerhetőnek kell lennie, ha több vonatkozás is jelen van.

SOLID – egyszeres felelősség elve (10)

- Példa az egyszeres felelősség elvének megfelelő, de vonatkozások szétválasztásának elvét megsértő kódra:
  - Artur Trosin. *Separation of Concern vs Single Responsibility Principle (SoC vs SRP)*. 2009. https://weblogs.asp.net/arturtrosin/separation-of-concern-vs-single-responsibility-principle-soc-vs-srp

SOLID – nyitva zárt elv (1)

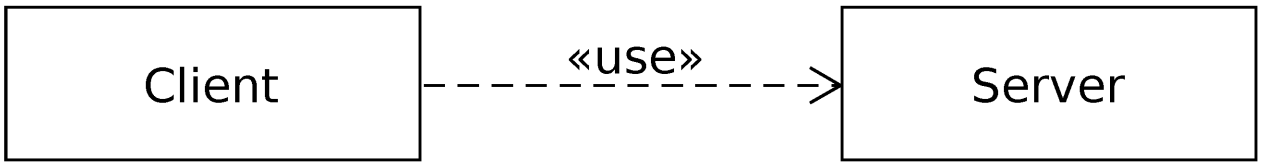
- Bertrand Meyer által megfogalmazott alapelv.
  - Bertrand Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1988.
- A szoftver entitások (osztályok, modulok, függvények, …) legyenek nyitottak a bővítésre, de zártak a módosításra.
- Kapcsolódó tervezési minták: gyártó metódus, helyettes, stratégia, sablonfüggvény, látogató

SOLID – nyitva zárt elv (2)

- Az elvnek megfelelő modulnak két fő jellemzője van:
  - Nyitott a bővítésre: azt jelenti, hogy a modul viselkedése kiterjeszthető.
  - Zárt a módosításra: azt jelenti, hogy a modul viselkedésének kiterjesztése nem eredményezi a modul forrás- vagy bináris kódjának változását.

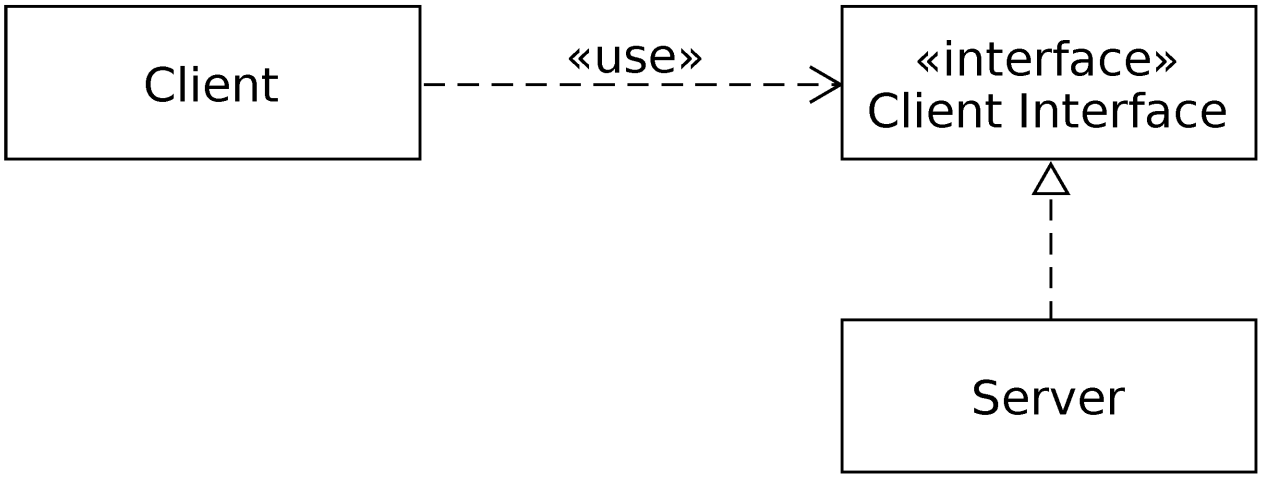
SOLID – nyitva zárt elv (3)

- Példa az elv megsértésére:
  - A `Client` és a `Server` konkrét osztályok. A `Client` osztály a `Server` osztályt használja. Ha azt szeretnénk, hogy egy `Client` objektum egy különböző szerver objektumot használjon, a `Client` osztályban meg kell változtatni a szerver osztály nevét.



SOLID – nyitva zárt elv (4)

- Az előbbi példa az elvnek megfelelő változata:



SOLID – Liskov-féle helyettesítési elv

- Barbara Liskov által megfogalmazott elv.
  - Barbara Liskov. *Keynote Address – Data Abstraction and Hierarchy*. 1987.
- Ha az *S* típus a *T* típus altípusa, nem változhat meg egy program működése, ha benne a *T* típusú objektumokat *S* típusú objektumokkal helyettesítjük.

SOLID – interfész szétválasztási elv (1)

- Robert C. Martin által megfogalmazott elv:
  - *„Classes should not be forced to depend on methods they do not use.”*
  - Nem szabad arra kényszeríteni az osztályokat, hogy olyan metódusoktól függjenek, melyeket nem használnak.

SOLID – interfész szétválasztási elv (2)

- **Vastag interfész** (***fat interface***) (Bjarne Stroustrup)
  http://www.stroustrup.com/glossary.html#Gfat-interface
  - *„An interface with more member functions and friends than are logically necessary.”*
  - Az ésszerűen szükségesnél több tagfüggvénnyel és baráttal rendelkező interfész.

SOLID – interfész szétválasztási elv (3)

- Az interfész szétválasztási elv a vastag interfészekkel foglalkozik.
- A vastag interfészekkel rendelkező osztályok interfészei nem koherensek, melyekben a metódusokat olyan csoportokra lehet felosztani, melyek különböző klienseket szolgálnak ki.
- Az ISP elismeri azt, hogy vannak olyan objektumok, melyekhez nem koherens interfészek szükségesek, de azt javasolja, hogy a kliensek ne egyetlen osztályként ismerjék őket.

SOLID – interfész szétválasztási elv (4)

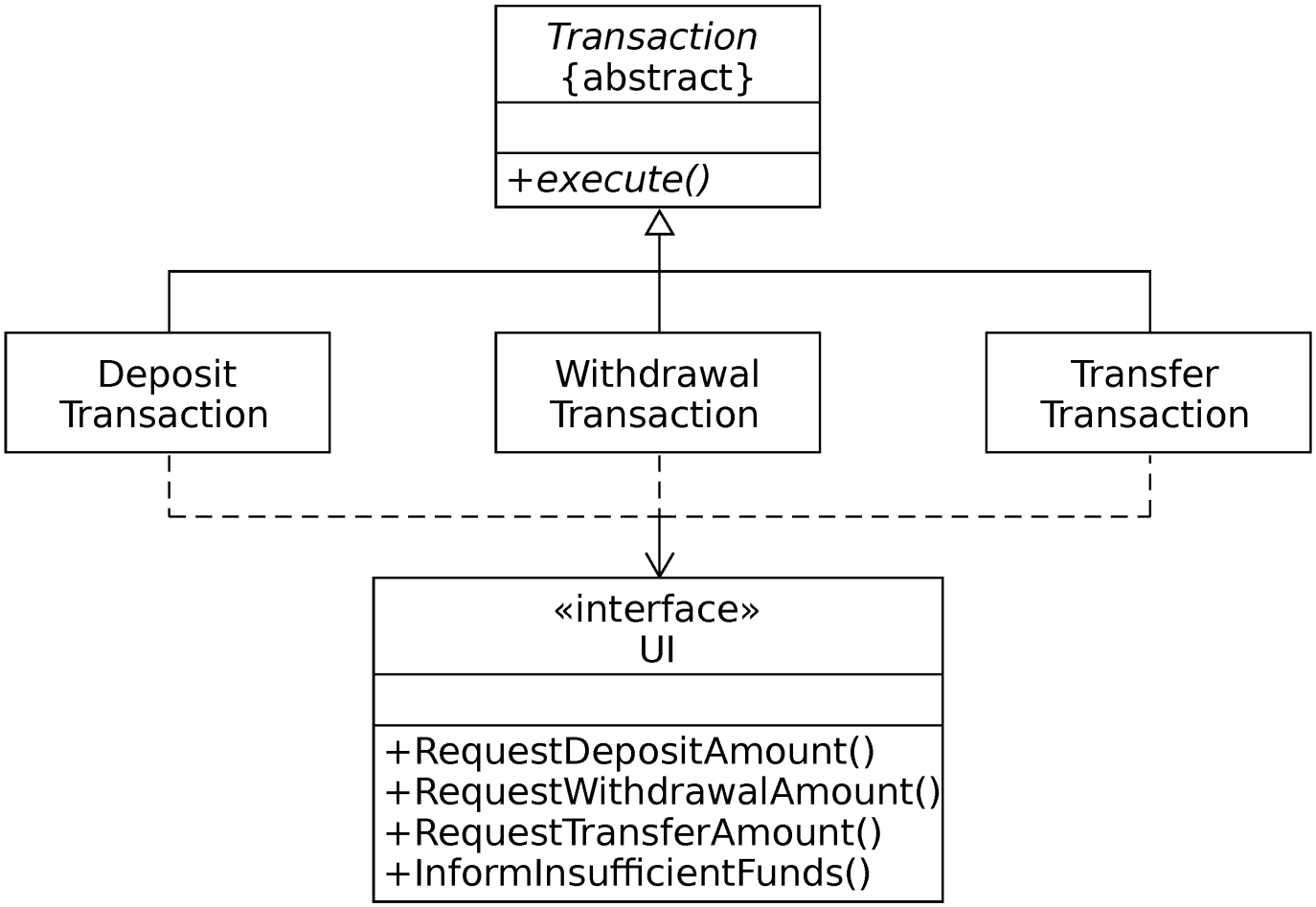
- **Interfész szennyezés** (***interface pollution***):
  - Egy interfész szennyezése szükségtelen metódusokkal.

SOLID – interfész szétválasztási elv (5)

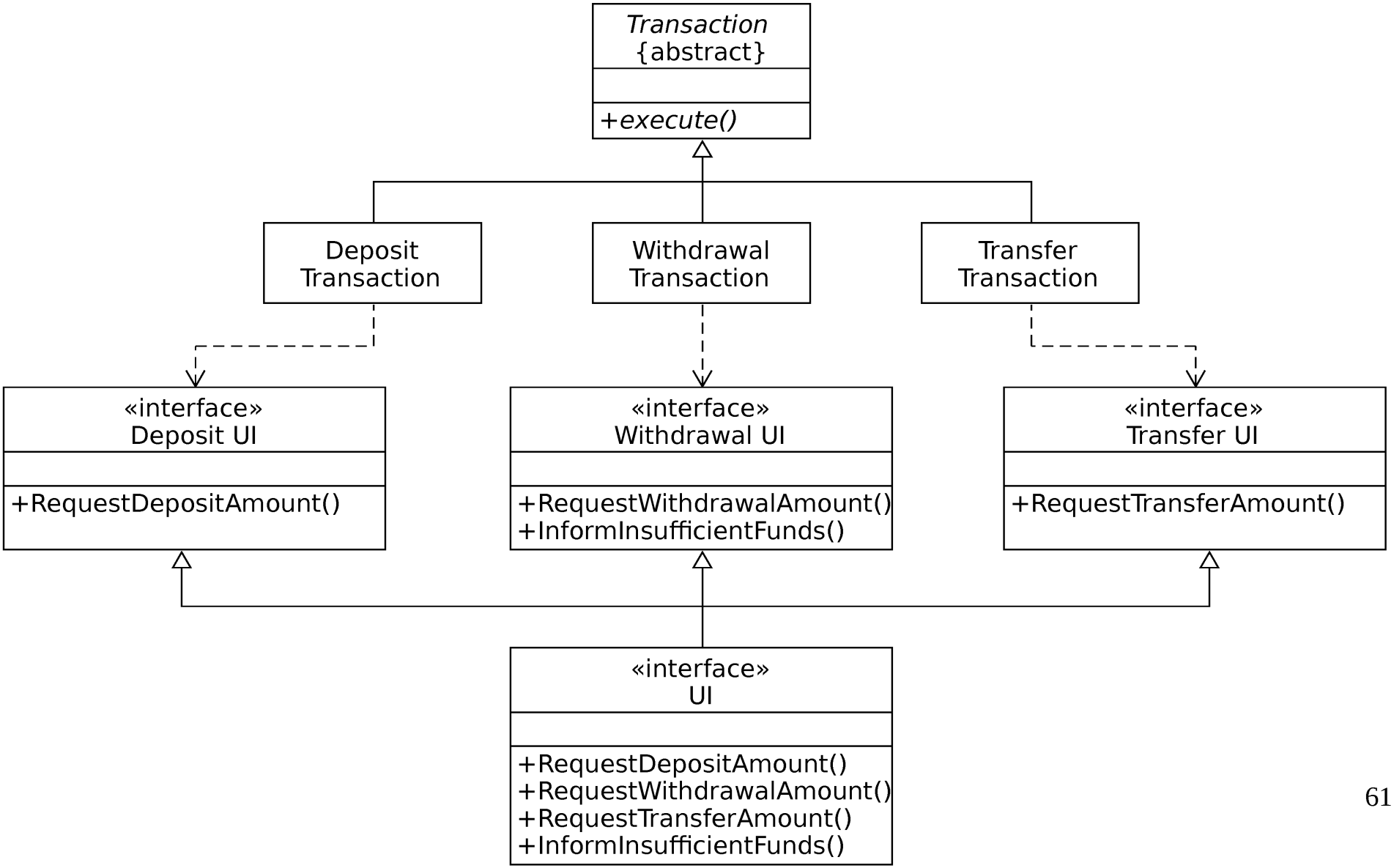
- Amikor egy kliens egy olyan osztálytól függ, melynek vannak olyan metódusai, melyeket a kliens nem használ, más kliensek azonban igen, akkor a többi kliens által az osztályra kényszerített változások hatással lesznek arra a kliense is.
- Ez a kliensek közötti nem szándékos csatoltságot eredményez.

SOLID – interfész szétválasztási elv (6)

- Példa: ATM (Robert C. Martin)



SOLID – interfész szétválasztási elv (7)



SOLID – függőség megfordítási elv (1)

- Robert C. Martin által megfogalmazott elv:
  - Magas szintű modulok ne függjenek alacsony szintű moduloktól. Mindkettő absztrakcióktól függjön.
  - Az absztrakciók ne függjenek a részletektől. A részletek függjenek az absztrakcióktól.

SOLID – függőség megfordítási elv (2)

- Az elnevezés onnan jön, hogy a hagyományos szoftverfejlesztési módszerek hajlamosak olyan felépítésű szoftvereket létrehozni, melyekben a magas szintű modulok függenek az alacsony szintű moduloktól.
- Kapcsolódó tervezési minta: illesztő

SOLID – függőség megfordítási elv (3)

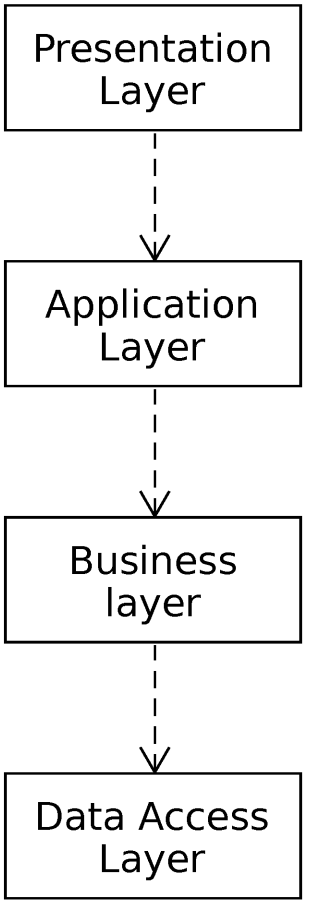
- A magas szintű modulok tartalmazzák az alkalmazás üzleti logikáját, ők adják az alkalmazás identitását. Ha ezek a modulok alacsony szintű moduloktól függenek, akkor az alacsony szintű modulokban történő változásoknak közvetlen hatása lehet a magas szintű modulokra, szükségessé tehetik azok változását is.
- Ez abszurd! A magas szintű modulok azok, melyek meg kellene, hogy határozzák az alacsony szintű modulokat.

SOLID – függőség megfordítási elv (4)

- A magas szintű modulokat szeretnénk újrafelhasználni. Az alacsony szintű modulok újrafelhasználására elég jó megoldást jelentenek a programkönyvtárak.
- Ha magas szintű modulok alacsony szintű moduloktól függenek, akkor nagyon nehéz az újrafelhasználásuk különféle helyzetekben.
- Ha azonban a magas szintű modulok függetlenek az alacsony szintű moduloktól, akkor elég egyszerűen újrafelhasználhatók.

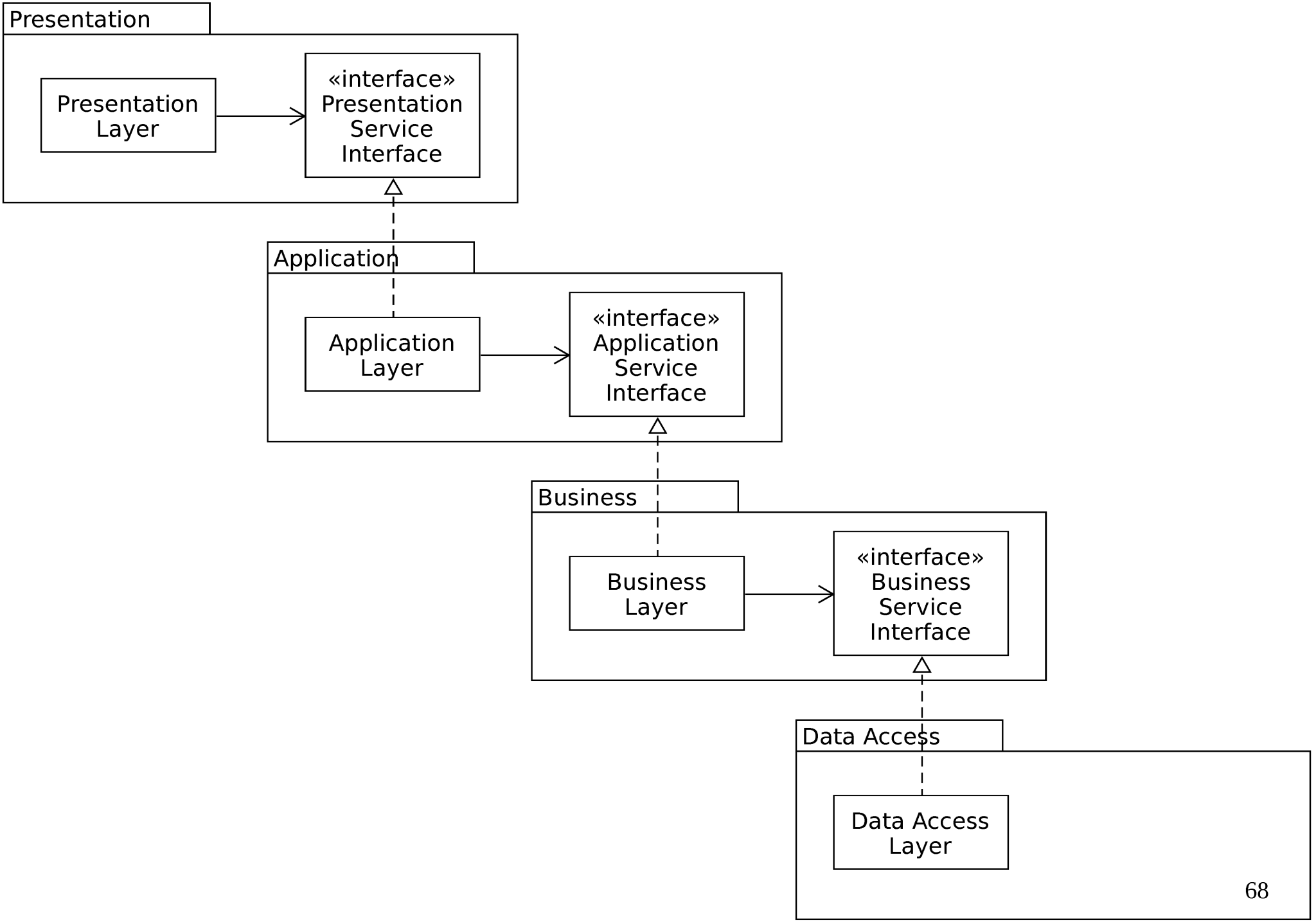
SOLID – függőség megfordítási elv (5)

- Példa a rétegek architekturális minta hagyományos alkalmazására:



SOLID – függőség megfordítási elv (6)

- Az előbbi példa az elvnek megfelelő változata:
  - Minden egyes magasabb szintű interfész deklarál az általa igényelt szolgáltatásokhoz egy interfészt.
  - Az alacsonyabb szintű rétegek realizálása ezekből az interfészekből történik.
  - Ilyen módon a felsőbb rétegek nem függenek az alsóbb rétegektől, hanem pont fordítva.


Presentation
Presentation Layer
«interface» Presentation Service Interface
Application
Application Layer
«interface» Application Service Interface
Business
Business Layer
«interface» Business Service Interface
Data Access
Data Access Layer

SOLID – függőség megfordítási elv (8)

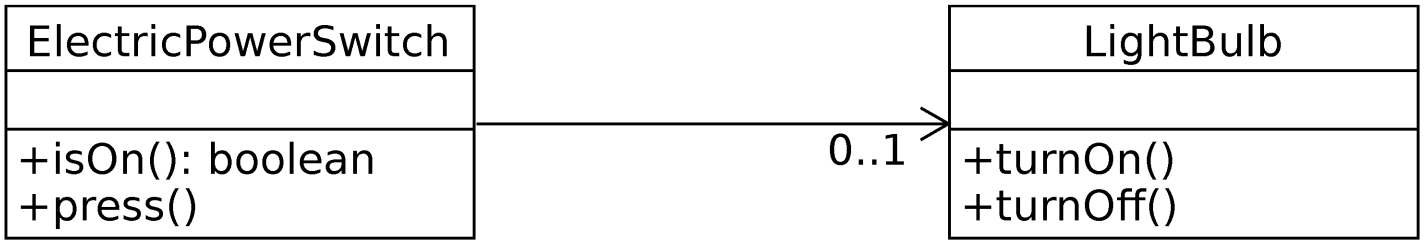
- Az előbbi példa az elvnek megfelelő változata: (folytatás)
  - Nem csupán a függőségek kerültek megfordításra, hanem az interfész tulajdonlás is (*inversion of ownership*).
    - Hollywood elv: Ne hívj, majd mi hívunk. (*Don't call us, we'll call you.*)

SOLID – függőség megfordítási elv (9)

- Függés absztrakcióktól:
  - Ne függjön a program konkrét osztályoktól, hanem inkább csak absztrakt osztályoktól és interfészektől.
    - Egyetlen változó se hivatkozzon konkrét osztályra.
    - Egyetlen osztály se származzon konkrét osztályból.
    - Egyetlen metódus se írjon felül valamely ősosztályában implementált metódust.
  - A fenti heurisztikát a legtöbb program legalább egyszer megsérti.
  - Nem túl gyakran változó konkrét osztályok esetén (például `String`) megengedhető a függés.

SOLID – függőség megfordítási elv (10)

- Példa az elv megsértésére:
  - Forrás: https://springframework.guru/principles-of-object-oriented-design/dependency-inversion-principle/



SOLID – függőség megfordítási elv (11)

- Az előbbi példa az elvnek megfelelő változata: