Gang-of-Four tervezési minták 2

Dependency inversion módszerek

- A függőségeket ne az őket felhasználó osztály hozza létre.
- Várjuk kívülről a példányokat interfészeken keresztül.
- Példány megadására több módszer is lehetséges
    - Dependency Injection
    - Inversion of Control (IoC) container
    - Factory tervezési minta
- **Ha nem követjük, akkor**
    - Egymástól szorosan függő osztályok végtelen láncolata
    - Nem lehet modularizálni és rétegezni
    - Kód újrahasznosítás lehetetlen

Dependency Injection

- Lazán csatoltság kiterjeszthetővé teszi a kódot, a kiterjeszthetőség pedig karbantarthatóvá.
- **Probléma**
    - A kód függjön absztrakciótól, ne konkrét implementációtól.
- **Megoldás**
    - Az interfészt várjuk paraméterként a konstruktorban.
    - Setter injektálás, amikor az objektumokat setter metódusok segítségével injektáljuk.

Factory (method) (Creational pattern)

- A Factory Method lehetővé teszi, hogy az új példány létrehozását leszármazott osztályra bízzuk. (Szokás virtuális konstruktornak is nevezni.)
- **Probléma**
    - Az objektumainkat gyakran bonyolult létrehozni és a konstruktor nem elég flexibilis ehhez.
- **Megoldás**
    - Az új objektumainkat a factory method-on belül hozzuk létre, ha pedig vissza is tér ezzel az objektummal, akkor azokat product-oknak is szokták nevezni.

Factory használjuk, ha

- Egy osztály nem látja előre annak az objektumnak az osztályát, amit létre kell hoznia.
- Egy osztály azt szeretné, hogy leszármazottjai határozzák meg azt az objektumot, amit létre kell hoznia.

Factory implementálása

1. Interfész implementálása a megfelelő metódusok segítségével.
2. A creator osztályban adjunk hozzá egy üres factory method-ot, ami visszatér az interfész típusával.
3. Factory method-ban hozzuk létre az új objektumokat.
4. Creator alosztályokat hozunk létre, ami a megfelelő factory method-ot használja.
5. Ezek után a base factory method üressé válik, így ezt abstract-á tehetjük.

Factory előnyök és hátrányok

- **Előnyök**
  - Single Responsibility elv
  - Open/Closed elv
- **Hátrányok**
  - A sok alosztály miatt bonyolulttá válhat a kód.

Abstract Factory (Creational pattern)

- **Probléma**
  - Különböző feltételek alapján más és más objektumokat szeretnénk szolgáltatni.
    - Pl egy stringtől függ, hogy milyen osztályt példányosítunk.
- **Megoldás**
  - Egy ősfactory – sok leszármazott factory
  - Dictionary vagy reflexió azonosítja a paraméter függvényében a megfelelő factory-t.

Abstract factory használjuk, ha

- A rendszernek függetlennek kell lennie az általa létrehozott dolgoktól.
- A rendszernek több termékcsaláddal kell együttműködnie.

Abstract factory előnyök és hátrányok

- **Előnyök**
  - Elszigeteli a konkrét osztályokat
  - Elősegíti a termékek közötti konzisztenciát.
- **Hátrányok**
  - Nehéz új termék hozzáadása.
    - Ilyenkor az Abstract Factory egész hierarchiáját módosítani kell, mert az interfész rögzíti a létrehozható termékeket.

IoC minták

- Dependency Injection
- Observer Pattern
- Template Method

Observer (Behavioral)

- Hogyan tudják az objektumok értesíteni egymást állapotuk megváltozásáról anélkül, hogy függőség lenne a konkrét osztályaiktól.
- **Az Observer az egyik leggyakrabban használt minta!**
- **Probléma**
    - Vevő szeretne vásárolni egy új terméket, de nem szeretne mindennap meglátogatni az üzletet, ahol lehet kapni.
    - Az üzlet pedig nem szeretné feleslegesen fogyasztani az erőforrásait abból a szempontból, hogy minden egyes új termék miatt küldözget emailt, mert ez csak spam lenne.
    - Tehát a vevő pazarolja a saját idejét vagy az üzlet az erőforrásait pazarolja.
- **Megoldás**
    - Kell egy subscriber, amivel feliratkozunk valamire és az értesít.
    - Feliratkozó osztályok megvalósítanak egy ISubscriber interfészt.
    - Írjon elő egy StateChange() vagy Update() metódust.
    - A subject kezelje a feliratkozókat Subscribe(), UnSubscribe()
    - Állapotváltozáskor hívja meg az összes feliratkozó StateChange() metódusát.
    - A feliratkozók tegyék meg a frissítési lépéseket.

Observer használjuk, ha

- Egy objektum megváltoztatása maga után vonja más objektumok megváltoztatását és nem tudjuk, hogy hány objektumról van szó.
- Egy objektumnak értesítenie kell más objektumokat az értesítendő objektum szerkezetére vonatkozó feltételezések nélkül.

Observer előnyök és hátrányok

- **Előnyök**
    - Open/Closed elv
    - Az objektumok közötti kapcsolatokat futás közben is létrehozhatjuk.
- **Hátrányok**
    - A subscriber-eket véletlenszerű sorrendben értesíti.

Template (Behavioral)

- Egy műveleten belül algoritmus vázat definiál és ennek néhány lépésének implementálását a leszármazott osztályra bízza.
- **Probléma**
    - Készítünk egy olyan alkalmazást, amivel különböző dokumentumokból adatokat lehet kinyerni.
    - Egy idővel rájövünk, hogy például a PDF, CSV fájlok között viszonylag hasonló műveletek hajtódnak végre, így kód duplikáció keletkezhet.
- **Megoldás**
    - Magát az algoritmust bontsuk szét kisebb lépésekre, metódusokra.
    - Ezeket fogjuk meghívni a template method-ban.

Template használjuk, ha

- Algoritmust kisebb lépésekre szeretnénk bontani.
- Logikai hasonlóság esetén

Template implementálása

1. Kisebb részekre bontás
2. Absztrakt osztály létrehozása, ahol deklaráljuk a template method-ot.
3. Hívjuk meg az alosztályokat, a lépéseket a template method-ban.

Template előnyök és hátrányok

- **Előnyök**
    - Kód duplikáció elkerülhető vele, tehát a hierarchiában a közös kódrészeket a szülő osztályban egy helyen adjuk meg (template method), ami a különböző viselkedést megvalósító egyéb műveleteket hívja meg.
        - Leszármazott osztályban felül lehet definiálni.
- **Hátrányok**
    - Megsérthetjük a Liskov behelyettesítési elvet, ha egy alosztályon keresztül elnyomja az alapértelmezett lépés implementációját.
    - Template method-okat egy idő után nehéz karbantartani, ha sok kisebb lépést (metódusokat) tartalmaz.

IoC használata a gyakorlatban (MVVM Light / ASP.NET Core)