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 principle
* **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

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