Mi a fene az a Dependency Injection

Vastag Atila

2024

A S.O.L.I.D. elvek kapcsán beszéltünk arról, hogy nem jó, ha a moduljaink szorosan függenek egymástól. De mégis, hogy a fenébe működik ez a gyakorlatban?

```
class MyBusinessLogic
{
  private MySQLConnection db;

  public MyBusinessLogic()
  {
    this.db = new MySQLConnection();
  }
}
```

Látszólag jó megoldás, de van néhány komoly hátulütője:

- Nem tudjuk leválasztani az osztály függőségeit teszteléshez.
- Nem tudjuk kicserélni a MySQLConnection osztályt egy másik, azonos működésű osztályra.

Magyarán szeretnénk létrehozni például egy DatabaseConnectionInterface interfacet, és elérni azt, hogy a MyBusinessLogic osztály csak és kizárólag ettől az interface-től függjön, ne pedig a konkrét megvalósítástól.

Na, de ha csak az interface-t használjuk, honnan fogja megkapni a MyBusinessLogic a megfelelő adatbázis kapcsolatot?

Az egyik megoldás, hogy bekérjük a konstruktorban például így:

```
interface IMySQLConnection
{

}

class MyBusinessLogic(IMySQLConnection db)
{

}
```

Van-e megoldás arra, ezt a konstruktort automatikusan paraméterezzünk?

Ha kicsit utána nézünk, kiderül, hogy van. A legtöbb modern nyelv rendelkezik olyan mechanizmussal, amivel le tudjuk kérni egy függvény elvárt paramétereit. Ezt a megoldást a legtöbb nyelvben *reflection*nek hívják. (Még mielőtt felhördülnél, hogy a reflection lassú, vannak megoldások, amik ezt megkerülik.)

Magyarán írhatunk egy olyan rendszert, amit egy konfigurációnak megfelelően felparaméterezi a szolgáltatásainkat. Ezt a fajta funkcionalitást *Dependency Injection Containernek*, vagy hosszabb nevén *Inversion of Control (IoC) Dependency Injection Containernek* hívják.

Ha a fenti példánál maradunk, a *Dependency Injection Contai*ner észleli, hogy az adott üzleti logikának szüksége van egy DatabaseConnectionInterface típusú implementációra (osztályra). Ezt viszont nem tudja automatikusan létrehozni, hiszen ez egy interface amit nem lehet példányosítani. Éppen ezért a DI-t fel kell paraméterezni, meg kell tanítani, hogy mely interface-t mely konkrét osztály valósítja meg.

Megvalósítás

```
public interface ICodeWriter
{
    string GetCode();
}
```

```
public class CodeGenerator : ICodeWriter
{
   private readonly Guid Guid = Guid.NewGuid();

    public string GetCode()
    {
        return $"code: {this.Guid}";
    }
}
```

```
AddSingleton<ICodeWriter, CodeGenerator>();
```

Minden egyes alkalommal (API kéréskor, osztály példányosításkor) ugyanazt az osztálypéldányt kapjuk.

```
AddScoped<ICodeWriter, CodeGenerator>();
```

Minden egyes alkalommal (API kéréskor, osztály példányosításkor) új osztály-példányt kapjuk. Ha egy API kéréskor vagy osztály példányosztáskor kettő vagy több ugyanazon függőségre van szükségünk, ugyanazt a példányt fogjuk kapni.

```
AddTransient<ICodeWriter, CodeGenerator>();
```

Minden egyes alkalommal (API kéréskor, osztály példányosításkor) új osztály-példányt kapjuk.