**Dependency Injection Mekanizmaları**

Dependency Injection (DI), yazılım geliştirmede kullanılan bir tasarım deseni olup, bir sınıfın bağımlılıklarını (örneğin, başka sınıfları veya servisleri) doğrudan kendisi yaratmak yerine dışarıdan almasına olanak tanır. Bu yaklaşım, yazılımın modülerliğini, test edilebilirliğini ve bakımını kolaylaştırır.

Dependency Injection, bir sınıfın ihtiyaç duyduğu bağımlılıkların dışarıdan sağlanmasını, böylece sınıfın bu bağımlılıkları kendisinin oluşturma zorunluluğunun ortadan kalkmasını sağlar. Bu yöntem, yazılımın esnekliğini, modülerliğini ve test edilebilirliğini artırır.

**Temel Kavramlar**

* **Dependency (Bağımlılık):** Bir sınıfın işlevlerini yerine getirmek için ihtiyaç duyduğu başka bir sınıftır.
* **Injection (Enjeksiyon):** Bağımlılığın, ihtiyaç duyan sınıfa dışarıdan sağlanmasıdır.

**Dependency Injection Mekanizmaları**

**1-Constructor Injection:**

* + Bağımlılıklar, sınıfın kurucusu (constructor) aracılığıyla sağlanır.

1. public class Service
2. {
3. private readonly IRepository \_repository;
4. public Service(IRepository repository)
5. {
6. \_repository = repository;
7. }
8. }

**2-Setter Injection (Property Injection):**

* Bağımlılıklar, sınıfın özellikleri (properties) aracılığıyla sağlanır.
* public class Service
* {
* public IRepository Repository { get; set; }
* public void DoSomething()
* {
* Repository?.Save();
* }
* }

**3-Method Injection:**

* Bağımlılıklar, bir metodun parametresi olarak sağlanır.
* public class Service
* {
* public void DoSomething(IRepository repository)
* {
* repository.Save();
* }
* }

**4-Interface Injection:**

* Sınıf, bağımlılıklarını bir arayüz aracılığıyla alır. Bu yöntem pek yaygın değildir.
* public interface IInjectable
* {
* void InjectDependency(IRepository repository);
* }
* public class Service : IInjectable
* {
* private IRepository \_repository;
* public void InjectDependency(IRepository repository)
* {
* \_repository = repository;
* }
* }

**DI Frameworkleri**

Birçok modern framework, DI'ı destekleyen mekanizmalar sunar. Örneğin:

* **ASP.NET Core:** Microsoft.Extensions.DependencyInjection
* **Spring (Java):** Spring Framework
* **Angular (JavaScript):** Angular Dependency Injection

Bu frameworkler, bağımlılıkları otomatik olarak enjekte eder ve yazılımcının manuel olarak bu işlemi yapmasını gereksiz kılar. Bu sayede kod daha temiz ve test edilebilir hale gelir.

**Yazdığım Kodlarda Dependency Injection**

Kullanıcı tarafından verilen bilgiye dayanarak, ASP.NET Core'da bir UserController içerisindeki API kodunu başka bir sınıfa taşıyıp, bu sınıfın objesini dependency injection kullanarak nasıl kullanacağımı açıklayacağım.

**1. Kodun Yapısı**

Önce UserController içinde API kodu yazıldı. Bu kod, bir iş sınıfına taşınacak ve UserController üzerinden bu iş sınıfına erişilecektir.

**2. Dependency Injection Kullanarak Kodun Ayrılması**

Bu örnekte, UserController içerisindeki API kodunu UserService sınıfına taşıdım ve UserController'ın constructor'ında bu servisi dependency injection ile aldım. Şimdi, dependency injection kullanımını detaylandıracağım.

**a. Constructor Injection**

Bu, en yaygın kullanılan DI yöntemidir. UserController'ın constructor'ında IUserService arayüzü kullanılarak UserService sınıfına bağımlılık eklenmiştir.

**b. Setter Injection (Property Injection)**

Setter Injection, genellikle constructor injection yerine daha az tercih edilir.

**Startup.cs veya Program.cs Dosyasında Konfigürasyon**

Setter injection için özel bir yapılandırma gerekebilir ve genellikle daha az tercih edilir. Constructor injection kullanımı önerilir.

**c. Method Injection**

Method Injection, bağımlılıkları doğrudan metoda parametre olarak geçmemizi sağlar. Ancak, bu yöntem ASP.NET Core'da genellikle kullanılmaz çünkü controller'larda constructor injection tercih edilir.

**d. Interface Injection**

Interface Injection, bağımlılıkları arayüzler aracılığıyla sağlamak için kullanılır. Bu yöntem pek yaygın değildir ve genellikle constructor injection veya property injection kullanılır.

**Sonuç**

Yukarıdaki örneklerde, UserController sınıfı ve UserService sınıfı arasındaki bağımlılık constructor injection kullanılarak sağlanmıştır. Bu, ASP.NET Core'da DI kullanımının standart yoludur. Startup.cs veya Program.cs dosyasında, UserService sınıfını IUserService arayüzü ile eşleştirerek DI konteynerine kaydettim.

Bu yapının amacı, UserController ve UserService arasındaki bağımlılıkları daha yönetilebilir ve test edilebilir hale getirmektir. DI, yazılımın modülerliğini ve esnekliğini artırır.