Dependency Injection(DI) Nedir? Nasıl Uygulanır?

**Kaydet**

Paylaştır[**22**](https://www.gencayyildiz.com/blog/dependency-injection-nedir-nasil-uygulanir/)

Merhaba,

Bu yazımızda Martin Fowler’in ortaya attığı Dependency Injection(DI) – Bağımlılık Enjeksiyonu kavramını inceleyecek ve nasıl uygulandığını örneklendireceğiz. Öncelikle “Nedir bu Dependency Injection?” sorusuyla başlayalım.

Dependency Injection’ı özetle anlatmak gerekirse; bağımlılık oluşturacak parçaların ayrılıp, bunların dışardan verilmesiyle sistem içerisindeki bağımlılığı minimize etme işlemidir. EEee yani!

Yani, temel olarak oluşturacağınız bir sınıf içerisinde başka bir sınıfın nesnesini kullanacaksanız new anahtar sözcüğüyle oluşturmamanız gerektiğini söyleyen bir yaklaşımdır. Gereken nesnenin ya Constructor’dan ya da Setter metoduyla parametre olarak alınması gerektiğini vurgulamaktadır. Böylece iki sınıfı birbirinden izole etmiş olduğumuzu savunmaktadır. Ha doğru mudur? dibine kadar doğrudur…

Daha da özetlememiz gerekirse yazılımı oluşturan yapılar kaçınılmaz olarak birbirleri ile ilişkilidir. Lakin bu ilişkinin bir bağa ve sınırlandırmaya sebep olmaması için mümkün mertebe ilişkiyi gevşek tutmak önemlidir. Biz buna Loosely Coupled yani Gevşek Bağlılık diyoruz.

Bundan dolayı yazılımı oluşturan yapıların birbirleri ile olan sıkı bağ azalacağı için uygulamaya yeni özellikler eklenip çıkartılabilmesi kolay hale gelecektir. Bu durumu şöyle örneklendirelim.

Elimizde “A” ve “B” sınıfı olsun. “A” sınıfında “B” nesnesi üretildiğini düşünün. Yazımızın devamında da örneklendireceğimiz gibi “A” sınıfı “B” sınıfına bağlı hale gelmiştir. Gün geldi “B” nesnesinde bir değişiklik olduğu zaman mecbur gidip “A” sınıfında da bu değişiklik üzerine çalışma yapmamız gerekecektir. Tamam, eğer burada konuştuğumuz gibi projeniz iki sınıftan ibaretse hiç Dependency Injection mevzusuna girmeden gidip “A” sınıfında değişikliği yapmanız mantıklıdır. Ama yüzlerce class’lardan oluşan bir proje için bunu denemenizi pek tavsiye etmem. Eee, peki ne yapacağız? diye sorarsanız eğer, “A” nın “B” ye olan bağımlılığını minumuma indirgeyeceğiz. Tabi bunun nasıl yapıldığını yazının devamında irdeleyeceğiz.

İşte bu tarz bir durumda Dependency Injection tekniğini uygulayarak uygulama içerisinde değiştirilmesi, müdahale edilmesi gereken yerler minumuma inecektir.

***Dependency Injection, bağımlılıkları soyutlamak demektir.***

Dependency Injection’ın teferruatlarına girmeden önce dikkat etmeniz gereken husus şudur ki; Dependency Injection çoğu zaman Dependency Inversion ile karıştırılır. Fakat Dependency Inversion problem çözmeye yarayan bir prensip iken Dependency Injection ise bu prensibi uygulayan bir Design Pattern’dir.

Şimdi Dependency Injection’ın tasarımını uygulama durumlarını konuşalım…

DI, aşağıdaki iki teknikle uygulanabilmektedir.

* Constructor Injection(Constructor Based Dependecy Injection)
* Setter Injection(Setter Based Dependency Injection)

Sizlere iki yöntemle örnek durumlarıda göstereceğim. Şimdi DI desenini uygulayabileceğimiz durumları pratik olarak inceleyelim.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Yukarıdaki kod bloğunu incelerseniz eğer yazının başında da anlattığım gibi “Vasita” sınıfı “Araba” sınıfına bağlı bir vaziyet arz etmektedir. Yani ben ne zaman “Araba” sınıfında bir değişiklik yapsam gelip “Vasita” sınıfında da o değişikliğe göre çalışma gerçekleştirmem gerekebilir. Haliyle onlarca sınıf söz konusu olduğu durumlarda bu pek mümkün olmayacaktır.

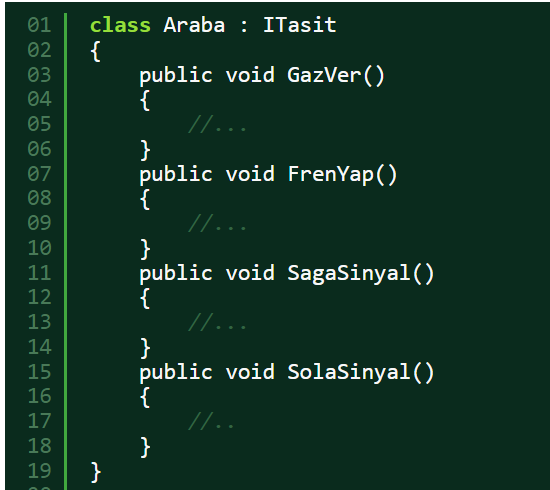
Şimdi düşünün ki, “Vasita” sınıfına “Araba” yerine “Otobus” classı vermek durumunda kalırsam eğer gelip burada ki komutları güncellemek zorunda kalacağım. Haliyle sabahtan beri bahsettiğim onlarca sınıf durumunda bağımlılık arz eden sınıflarda güncelleme yapmak hiç yazılımcı işi değildir.

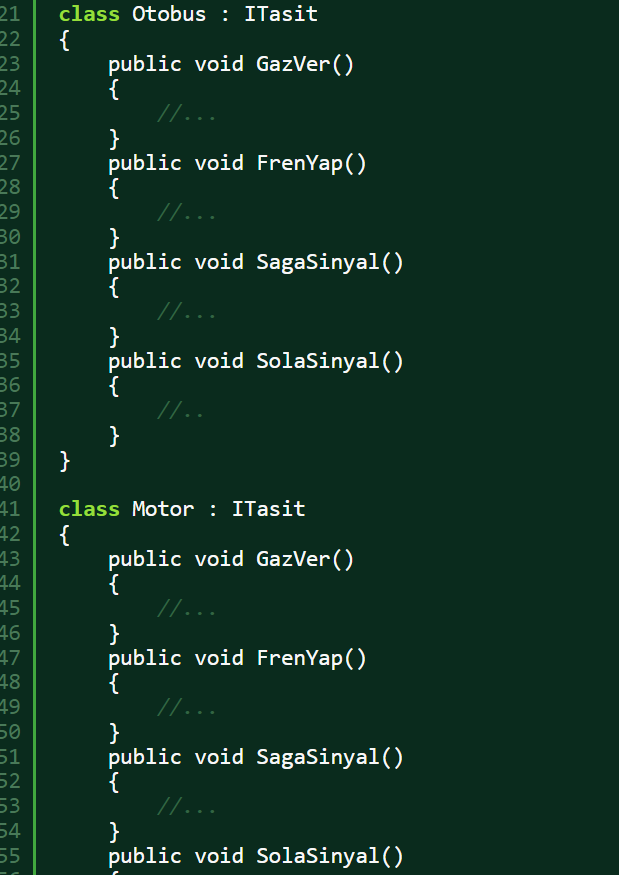
İşte… “Araba sınıfı istediği kadar değişsin ama Vasita sınıfının bundan haberi olmasın. Haberi olmasın ki Vasita sınıfıyla uğraşmak zorunda kalmayayım” diyorsak eğer Dependency Injection(DI) tasarımını uygulayacağız.

Şimdi DI desenini uygulamak için bir interface tanımlayacağız. Tanımladığımız bu Interface’den kalıtım alan tüm sınıflar doğal olarak Interface’in kalıbını, şartını sağlayacaktır.

|  |  |
| --- | --- |
|  |  |

Evet, gördüğünüz gibi Interface’imizi oluşturduk. Şimdi “Araba”, “Otobus”, “Motor” vs. gibi taşıtlarımızı bu Interface’den türeteceğiz.





|  |  |
| --- | --- |
|  |  |

Şimdi vakit “Vasita” sınıfımıza DI desenini uygulamaya geldi.

|  |  |
| --- | --- |
|  |  |

Yukarıdaki kod bloğunu incelerseniz eğer, “ITasit” Interface’inden kalıtım alan herhangi bir sınıf “Vasita” sınıfına bir taşıt olabilir. Haliyle siz proje sürecinde “Vasita” sınıfına ister “Araba”, isterseniz “Otobus” yahut “Motor” verebilirsiniz. Nede olsa hepsi aynı metod ve işlevleri barındırdığı için hangi nesneyi verirseniz verin Interface referansı üzerinden ilgili nesne çalıştırılacaktır.

Olaya “Vasita” sınıfı açısından bakarsak eğer, kendisine verilen nesnenin ne olduğuyla ilgilenmemekte ve bağlı olduğu bir sınıf bulunmamaktadır. Yani olay araba, otobüs veyahut motor da olsa bu elemanların çalışma ve işleyişiyle ilgilenmemektedir. Nihayetinde araba ve otobüste gaz vermek için pedala basılırken, motorda sağ kolu çevirmek lazımdır. İşte Dependency Injection sayesinde “Vasita” sınıfı bu işlevlerin nasıl yapıldığıyla ilgilenmemekte, sadece ve sadece gelen araç hangisi olursa olsun gaz vermekte, frene basmakta ve sağa sola sinyal çakmaktadır. Eğer DI uygulanmazsa, “Vasita” gaz verme işleminde araba ve otobüs için pedalla, motor içinse sağ kolla ilgilenecektir.

Anlayacağınız “Vasita” sınıfı Dependency Injection sayesinde üzümü yiyor bağını sormuyor, farklı bir sınıfa olan bağını minimize etmiş oluyor.

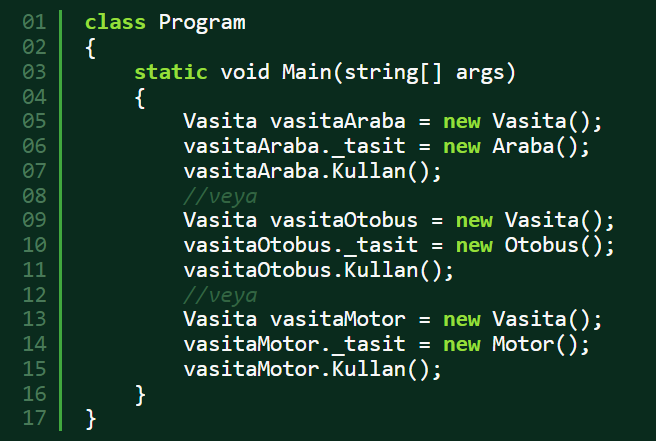
Bu tasarım neticesinde aşağıdaki şekilde kullanım gerçekleştirilebilir.

|  |  |
| --- | --- |
|  |  |

Şuana kadar sizlere Dependency Injection’ı Constructor Injection(Constructor Based Dependecy Injection) yöntemi ile göstermiş oldum. Setter Injection(Setter Based Dependency Injection) yöntemi ile DI’yı uygulamak istiyorsanız eğer sadece “Vasita” sınıfında(yani projenizde DI’yı uyguladığınız sınıfta) aşağıdaki değişikliği yapmanız yeterlidir.

|  |  |
| --- | --- |
|  |  |

Yukarıdaki gibi Setter Injection yöntemi uyarlandıysa kullanımı aşağıdaki gibi yapmalısınız.



Gördüğünüz gibi Dependency Injection kodunuzu özgürleştirmekte, sınıflar arasındaki bağı minimize etmektedir. Sınıflar çalıştırılırken üzümü yiyen amma bağını sormayan olgunlukta olurlar. Ama bunun yanında Visual Studio’nun “Go To Definiton” özelliğinide boşa çıkaran bir yapı olduğunu söyleyebilirim. Çünkü DI uyguladığınız sınıfta ilgili Interface’e “Go To Definiton” dediğiniz zaman sizi Interface’e gönderecektir. Eee haliyle hangi sınıfa bu Interface’in uygulandığını bulabilmeniz bazen işgenceye dönebilir. Hele hele bir başkası tarafından tasarlanmış proje üzerinde çalışma gerçekleştiriyorsanız….

Hepinize iyi çalışmalar dilerim…

Sonraki yazılarımda görüşmek üzere…