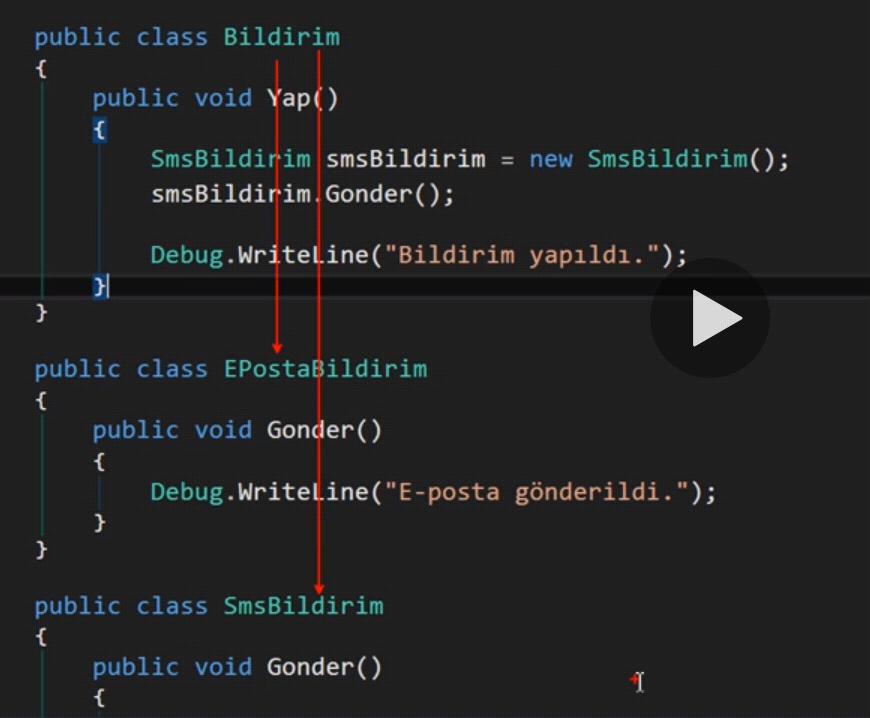
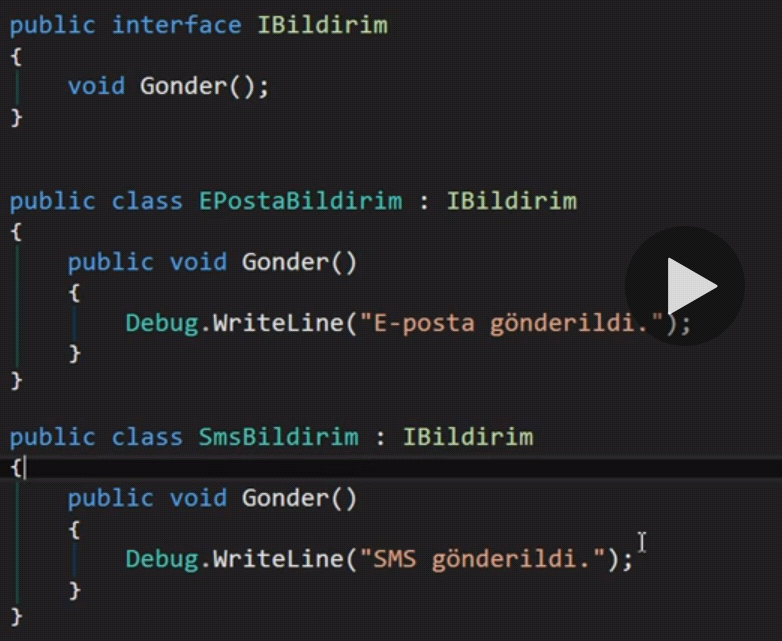
**Dependency Inversion ve Ötesi**

Temel olarak,  *‘Üst Sınıflar Alt Sınıflara Bağımlı Olmamalı’* diyoruz. Örneklerle açıklayalım

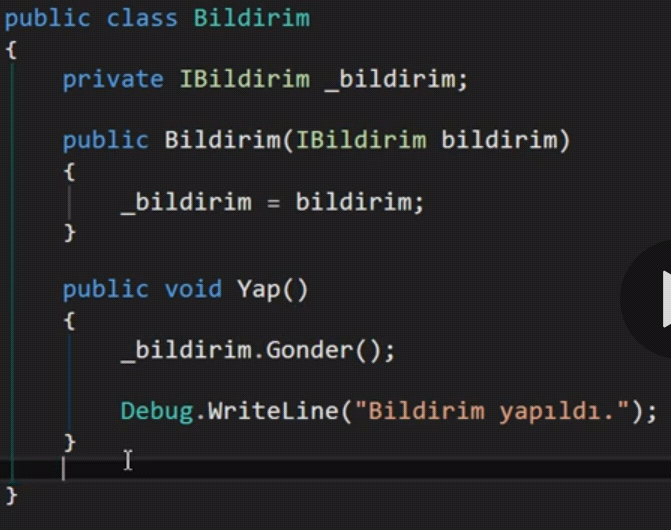
Aşağıdaki gibi Bildirim adında bir üst sınıfımız olsun. İçersindeki Yap fonksiyonunda SmsBildirim adındaki alt sınıfımızı kullanıyor olsun. Burada eposta ile bildirim yapılmak istenseydi o zaman üst sınıfımız EPostaBildirim alt sınıfımızı kullanacaktı.



Bunu dersine çevirmek için, soyutlama yaparız. Bir IBildirim interface i oluşturarak. EPostaBildirim ve SmsBildirim alt sınıflarımızı bu interfaceden türetiriz.



Bu sefer üst sınıfımızda soyutlama yaparak oluşturduğumuz interface i kullanacağız.

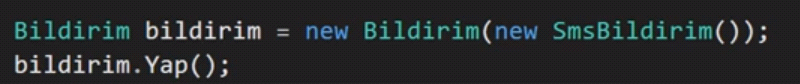


Bu soyutlama işlemi ile Dependency Inversion Prensibine uymuş oluyoruz.

Soyut sınıfın, üst sınıfımızın constructor ında parametre olarak verilmesine class seviyesinde Dependency Injection diyoruz.

Dependency Injection Patterni ile Dependency Inversion Prensibine uymuş oluyoruz.

Son olarak Main methodunda nasıl kullanıldığına bakalım:



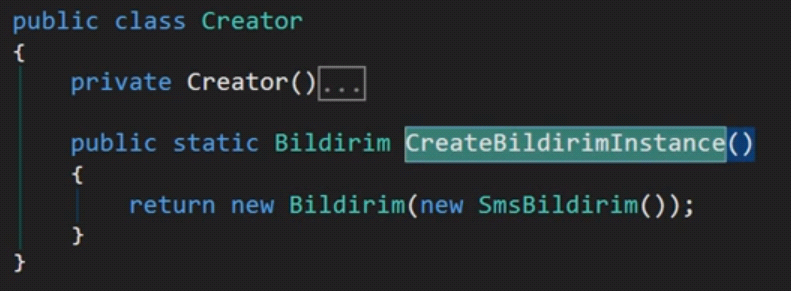
Bu şekilde SOLID prensiplerinden Dependency Inversion ı sağlamış olduk. Ancak, burada bir kullanım zorluğu bulunmakta.

Kullanım Zorluğu:

Proje büyüdüğünde her yerde Bildirim üst sınıfımızdan instance üretirken, constructorına hangi alt sınıfı vereceğimizi bilmek zorunda kalacağız.

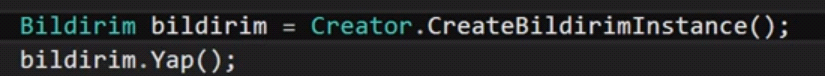
Bunun için bir kademe daha yukarı çıkıp, Bildirim üst sınıfımızı üreten bir Creator sınıfı oluşturacağız.

Creator sınıfımızı oluşturalım:



Bu yapıya IoC Container diyoruz.

Kullanımı:



**Ninject IoC Container Tool**

Manage Nuget ten ninject i indirdikten sonra Ninject Helper classımızı ekliyoruz.

public static class NinjectHelper

{

public static IKernel Kernel { get; set; }

//ninject ayarlarını yapacak

public static void Configure()

{

Kernel = new StandardKernel();

//IBildirim tipinden bir şey istediğimde bana EpostaBildirimini ver

//global asax da bunu proje ayağa kalktında çağırmamız gerekir.

Kernel.Bind<IBildirim>().To<EpostaBildirim>();

}

}

Global.asax a da bu configurasyonu ekliyoruz

protected void Application\_Start()

{

GlobalConfiguration.Configure(WebApiConfig.Register);

NinjectHelper.Configure();

}

**Ninject IoC Container ın Kullanımı:**

* İster aşağıdaki gibi direk alt sınıfımızı kullanarak;

IBildirim bildirim =NinjectHelper.Kernel.Get<IBildirim>();

return bildirim.Bildir();

* İster BildirimManager üst sınıfımızı kullanarak;

Burada BildirimManager ın constractor da Ibildirim e ihtiyac duydugunu anlayıp.

NinjectHelper class ında belirttiğimiz gibi IBildirim istendiğinde hangi sınıf çağrılıyorsa onu getirdiğini görüyoruz.

BildirimManager bm = NinjectHelper.Kernel.Get<BildirimManager>();

return bm.SendBildirim();

Ayrıca SmsBildirim altsınıfını her istek için tekradan olusturmasını istemiyorsak, yani singleton pattern şeklinde çalışmasını istiyorsak NinjectHelper clasındaki configurasyonumuzu şu şekilde güncellememiz gerekiyor.

Kernel.Bind<IBildirim>().To<EpostaBildirim>().InSingletonScope()

Controller ın constructor ında bu configurasyonu sağlamak için nugetten bir şe daha indirmemiz gerekiyor.

* Ninject.Web.WebApi
* Ninject.Web.WebApi.WebHost

Yüklemelerden sonra AppStart klasörümüzün altında NinjectWebCommon adında bir class geliyor. Bu class ın en altında RegisterServices methodu bizim NinjectHelper classımızın işini görüyor. O yüzden NinjectHelper classımızdaki bind işlemini buraya taşıyoruz. Ayrıca Global.asax dan da NinjectHelper ın configurasyonunu kaldırabiliriz

Controller dan kernel a erişme ihtiyacımız olursa, NinjectWebCommon sınıfı içerisinde Bootstrapper değişkeni var bunu public olarak değiştirmemiz lazım. Controllerda da kerneldan daha önce yaptıgımız binding i kendinimiz çağırmak istiyorsak(controller ın constructordan değil) controller a şu şekilde kodumuzu yazabiliriz:

Artık controller ın constructor ında IBildirim i cağırabiliriz.