Open Closed Principle – OCP – (Açık-Kapalı Prensibi)

Merhaba sevgili yazılımcı dostlarım…

Bu makalemizde nesne yönelimli tasarım prensiplerinden (OOP Principles) Open Closed Principle (Açık/Kapalı prensibini) kısaca OCP’yi ele alacağız. Şimdi durup, günümüzde kullandığımız teknolojik ürünlere bir göz gezdirelim… Tüm ürünler, şu anda kullanmakta olduğumuz son versiyona ulaşıncaya dek ne kadar çok değişikliğe uğradı değil mi? Tamam tamam. Sadece teknolojik ürünler değil, var olan bütün sistemler için geçerli bu söylediğimiz.

Yani, biraz klişe olacak ama “değişmeyen tek şey değişimdir”. Peki; madem üretmekte olduğumuz ürünler zamanla değişecek, o zaman biz de bu değişimi “en düşük maliyetli” hale getirmeliyiz değil mi? Bunu sağlamak için bir ürünü, birçok parçanın birleşmesiyle elde ederiz.

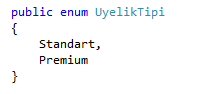
Örneğin bir otomobil aslında binlerce parçadan oluşuyor. Tekerlekler, krank mili, motor, akü, depo vesaire… Bu sayede, örneğin yeni bir araba lastiği teknolojisi geliştirildiğinde, hemen var olan bir otomobile uygulanabiliyor. Tek yapmanız gereken arabanızın lastiklerini değiştirmek o kadar. Düşünsenize bir lastiği değiştirmek için arabayı komple değiştirmeniz gerektiğini!

Hadi gelin bu örneği, bir yazılım projesine uyarlayalım. Projenizde kullandığınız nesneler de tıpkı bir makinenin parçaları gibidir. Dolayısı ile bu nesnelerin işlevleri ve sayıları değişebilir. Demek ki öyle bir yapı tasarlamalısınız ki, gelişmeye AÇIK fakat kaynak kodun değiştirilmesine KAPALI olmalı. İşte açık-kapalı prensibinin anahtar cümlesi tam olarak bu. Ama ben yine de bu kavramın mimarı Bertrand Meyer’in cümlesini de buraya yazmayı üstada bir borcum olarak görüyorum: “**Yazılım varlıkları (classlar, modüller, fonksiyonlar vs.) gelişime AÇIK, kod değişimine KAPALI olmalıdır.**”

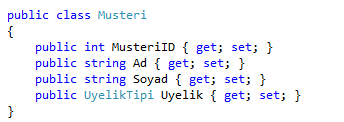
Eğer bu prensibin teorisine biraz daha eğilmek isterseniz hemen Google’ı açıp “Robert Martin open closed principle” yazın ve efsanenin konu hakkındaki makalesini okuyun.

Artık örnek yapma vakti geldi sanırım. Önce senaryomuzu hazırlayalım. Bir e-ticaret uygulamasında, müşterinin üyelik tipine göre indirim yapmak istiyorsunuz. Buna göre sistemde, premium ve standart olmak üzere iki tür üyelik tipi var. Hadi bakalım; ilk olarak bu sorunumuzu, OCP’ye **uymayacak şekilde**çözelim.

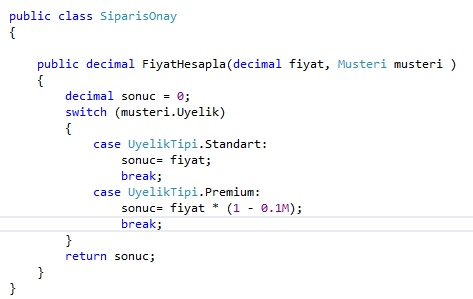
Hemen, üyelik tiplerini tutacak enum tipimizi oluşturalım:



Şimdi de, müşteriden sorumlu sınıfımızı inşa edelim:



Tamamdır. Son olarak da, bize indirimli fiyatı hesaplayacak sınıfımızı halledelim.

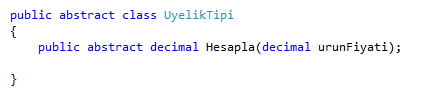


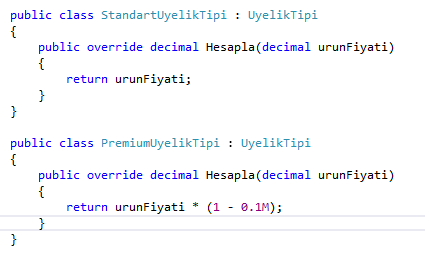
Her şey yolunda gözüküyor değil mi? Projenin çalışmama ihtimali yok. Peki, gelecekte yeni bir üyelik tipi eklenirse ne yapacaksınız! Hah! Gördünüz mü sıkıntıyı şimdi! Projeniz nesnel olarak gelişime açık değil. Çünkü bahsedildiği gibi bir geliştirme yapmak isterseniz, kaynak kodunuzu ciddi bir biçimde değiştirmeniz gerekiyor. Yani UyelikTipi içerisine “Gold” değerini ekledikten sonra bir de SiparisOnay sınıfı içerisindeki FiyatHesapla metoduna yeni bir “case” eklemeniz gerekiyor.

Oysa ne diyordu anahtar cümle: “**Yazılım varlıkları (classlar, modüller, fonksiyonlar vs.) gelişime AÇIK, kod değişimine KAPALI olmalıdır.**”

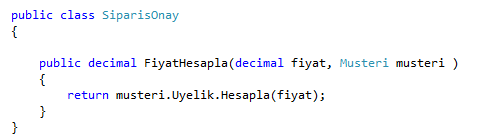
Eğer bu senaryonun OCP’ye uymasını istiyorsam SiparisOnay sınıfını öyle tasarlamalıyım ki, bir daha asla değiştirmeye gerek kalmasın. **Ayrıca yapının geliştirilebilir olması için enum tipinden de kurtulmam gerek!**

İşte bu noktada, imdadımıza abstract class yetişiyor. Üyelik tiplerini tek bir çatı altında toplayarak, çok biçimliliğin nimetlerinden faydalanabiliriz. O halde işte çözüm…





Bu değişiklikten sonra SiparisOnay sınıfında yer alan FiyatHesapla metodunu güncellersem, OCP’ye uygun bir mimari elde etmiş olurum.



Şimdi “Gold” üyelik eklemek istediğinizde, tek yapmanız gereken UyelikTipi sınıfından türeyen yeni bir sınıf inşa etmek olacak o kadar!

Sonuç olarak o kilit soruları soralım:

1. Yeni bir üyelik tipi eklendiğinde, SiparisOnay sınıfında yer alan kodu açıp değiştirmeniz gerekecek mi?

Cevap: Hayır! SiparisOnay sınıfı kod değiştirmeye KAPALI

1. İstediğim kadar üyelik tipi ekleyerek projeyi geliştirebilir miyim?

Cevap: Evet! Tasarımımız, yeni üyelik tipleri ekleyerek gelişmeye AÇIK

İşte sevgili dostlarım. SOLID’in O’su olan Open/Closed Principle’ın olayı bu. Bu arada bu prensibi uygulamak için izlediğim yöntem ise sadece bir yaklaşım. Yani tek yöntem elbette bu değil. Ama sanırım en basiti bu.

Bir sonraki makalede görüşmek üzere.

Hoşçakalın.