Liskov Substitution Principle – LSP – ( Liskov Yerdeğiştirme Prensibi)



Merhaba sevgili yazılım dostları

SOLID prensiplerini irdelemeye Liskov Substitution Principle ile devam ediyoruz. Bu yazımda, sayın Barbara Liskov hanımın hayatımıza katmış olduğu prensibi anlatacağım. Hali hazırda MIT (Massachusetts Institute of Technology) programlama metodolojileri grup liderliği yapan bu dahi kadın bakın 1988 yılında Data Abstraction and Hierarchy adlı kitabında ne demiş?

“**Aranan yer değiştirme özelliği şöyle tanımlanabilir: T cinsinden parametre alan tüm programlar (fonksiyonlar) P olacak şekilde, S tipinde o1 nesnesi ve T tipinde o2 nesnesi olsun. Eğer o1 ile o2 nesneleri yer değiştirdiğinde P’nin davranışı değişmiyorsa S tipi T tipinin alt tipidir!**”

Ney? Çok iyi anlayan (hatta anlayan  ) varsa beri gelsin! Kendi adıma söyleyeyim; beynimde şiddetli bir karıncalanma hissetmiştim ilk okuduğumda. Hele bir de İngilizcesini okuyunca inanın bana kafayı yiyorsunuz. İşte Barbara ablamızın orijinal cümlesi:

***“What is wanted here is something like the following substitution property:  If for each object o1 of type S there is an object o2 of type T such that for  all programs P defined in terms of T, the behavior of P is unchanged when  o1 is substituted for o2 then S is a subtype of T.”***

Ama bakın, bunu resmi olarak 1994 yılında nasıl formüle etmişler (baştan uyarıyorum; bu da matematik ile açıklanmış bir ifade):

**q(x)**fonksiyonu T tipindeki x nesnesi için kanıtlanabilirdir. O halde T tipinin alt tipi olan S tipindeki y nesnesi için de **q(y)**fonksiyonu kanıtlanabilir olmalıdır.

Eğer hala bu satırları okuyorsanız (yani “Aha! Şişman Adam sonunda delirdi. Yazık lan” dememişseniz), bu prensibin yazılım tarafında nasıl ifade edileceğini merak ediyorsunuz demektir. Kendi tanımlamalarıma geçmeden önce, bu tarz kavramları ilk kez basite indirgeyerek yazılım dünyasına anlatan Robert Martin’den de bir alıntı yapmamak olmaz.

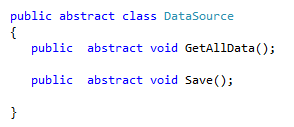
Robert Martin (namı diğer Bob amca), bakın bu prensibi nasıl anlatıyor:

**“Temel (base) sınıfın işaretçisini (pointer) ya da referansını kullanan fonksiyonlar, bu sınıftan türemiş olan (derived) sınıfları da ekstra bilgiye ihtiyaç duymaksızın kullanabilmelidir.”**İşte biraz daha yazılım kavramlarıyla bezenmiş bir tanım.

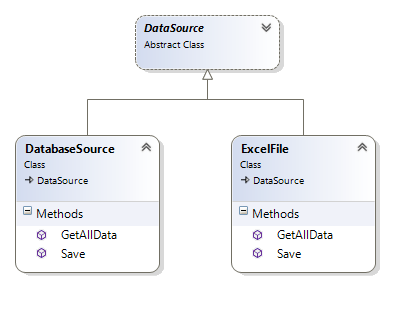
Bakalım ben aynı prensibi nasıl tanımlayacağım?

**“Aynı temel sınıftan türeyen tüm sınıflar, birbirlerinin yerine kullanılabilir olmalıdır. Bu yer değiştirme durumunda, sınıfa özel bir istisna kesinlikle oluşmamalıdır.”**Tabii bu tanım için yorumlarınızı, bu sayfanın altına bekliyorum.

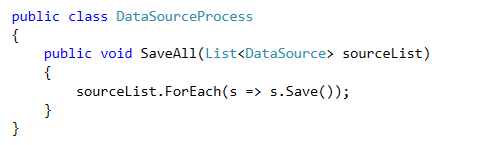
Elbette bu prensibin anlaşılması için yine bir örnekten faydalanacağız. Diyelim ki, farklı veri kaynakları (SQL, Excel dosyası vs.) ile çalışabilecek bir uygulama geliştiriyorsunuz. Tüm bu veri kaynakları ile “yükleme” ve “kayıt” işlemleri yapabileceğinizi analiz ettiniz ve bu metotları bir abstract sınıfta (ya da interface) tutmaya karar verdiniz. Sonuç olarak şöyle bir abstract sınıf geliştirmiş oldunuz:



Sonra da bu sınıftan türeyen sınıflarınızı aşağıdaki gibi oluşturdunuz.



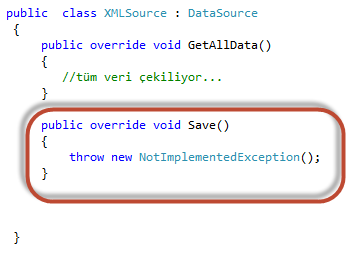
Sıkıntı yok. Her şey yolunda… Şimdi de veri kaynakları ile işlem yapacak bir sınıfa ihtiyacınız var. Kolları sıvadınız ve aşağıdaki sınıfı oluşturdunuz.



Tamam. Her şey tıkırında. Şu ana dek geliştirmiş olduğunuz tasarım da LSP’ye gayet uyuyor. Ortak bir sınıftan türeyen DatabaseSource ve ExcelFile dosyaları birbirlerinin yerine kullanılabilir durumda. Hadi gelin şimdi işleri biraz karıştıralım.

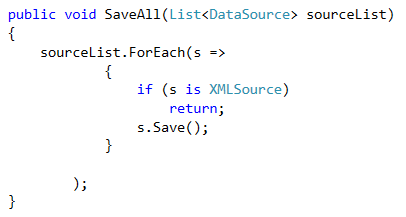
Projenizi geliştirirken; XML kaynağına da ihtiyaç duydunuz ve yine aynı abstract sınıftan türeyen XMLSource sınıfını oluşturdunuz. Ancak, müşterinin size özellikle vurguladığı bir şey var: “XML dosyaları sadece yüklenebilmeli. Kaydedilmelerini istemiyoruz.” Ne yaparsınız?

Akla hemen iki seçenek geliyor. Bunlardan ilki, XmlSource sınıfında yer alan Save metodunun Exception fırlatması olabilir.



Ancak bu çözümü uygularsanız, DataSourceProcess sınıfında yer alan SaveAll() metodunda; try-catch bloğu kullanmanız gerekecek. Yani, “sınıfa özel istisnai bir durum” şüphesi içinde olacaksınız.

Diğer bir çözüm ise, SaveAll() metodu içerisinde tip kontrolü yapmak olacaktır. Yani metodunuzu şöyle geliştireceksiniz:

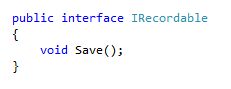


Fakat burada da yine “sınıfa özel istisna” yok mu sizce? Bariz bir şekilde tipe özel bir fonksiyon yazmış oluyorsunuz. Yani şu an itibariyle LSP’ye ***uymayan*** bir tasarım geliştirmiş bulunmaktasınız.

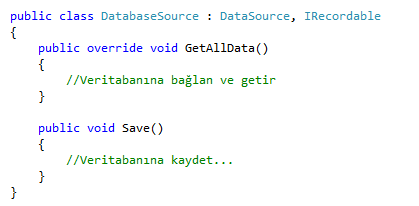
Peki, nasıl çözeceğiz bu işi?

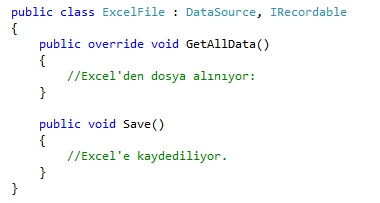
Aslında bu problemin çözümüne hizmet edecek birkaç yöntem var. Benim burada göstereceğim çözüm, bunların içinden en basit olanı.

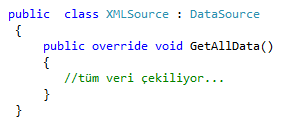
Bu yöntemde, özel bir durum oluşmasına sebep olan Save() metodunu ayrı bir abstract sınıf (veya interface) içerisine alacağız. Hadi burada Interface’i tercih edelim:



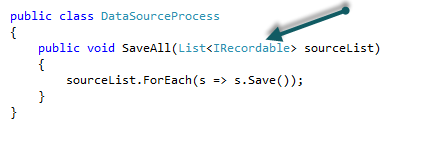
Böylece abstract DataSource sınıfımda yalnızca GetAllData metodunu bıraktım. Bu durumda ExcelFile ve DatabaseSource sınıflarım hem DataSource sınıfını miras alacak hem de IRecordable interfece’ini uygulayacak (implemente edecek). XMLSource sınıfım ise sadece DatabaseSource sınıfından türeyecek. Yani:







Sonuç olarak; DataSourceProcess sınıfımda SaveAll metodunu tekrar düzenleyerek LSP’ ye uygun bir hale getirebilirim.



Gördüğünüz gibi sevgili dostlarım; SaveAll metodunun son halinde, herhangi bir sınıfa özel bir durum söz konusu değil. IRecordable interface’ini uygulayan her tip birbirinin yerine kullanılabilir durumda.

Bu noktada bir yazımızın daha sonuna gelmiş bulunuyoruz. Bir sonraki yazımda görüşmek üzere kendinize iyi bakın.