SOLİD PRENSİPLERİ NELERDİR ?

SOLID prensipleri, bir yazılım geliştiricinin [**Nesne Yönelimli Programlama (OOP)**](https://yazilimcigenclik.com.tr/nesne-yonelimli-programlama-oop-nedir/) ile yazılım geliştirirken, geliştirdiği yazılımın esnek ve geliştirilmeye uygun olması için uyması gereken kurallar bütünüdür. Bu prensipleri dikkate alarak geliştirmiş olduğumuz projelerin büyük ne olursa olsun karmaşıklığı önler diyebiliriz. SOLID prensipleri 5 ana prensibin baş harflerinden oluşan bir mimaridir diyebiliriz. 5 ana başlığı aşağıdaki gibi sıralayabiliriz:

1. S – Single Responsibility Principle
2. O – Open/Closed Principle
3. L – Liskov Subtitution Principle
4. I – Interface Segregation Principle
5. D – Dependency Inversion Principle

1 – Single Responsibility Principle

Türkçe karşılığı “Tek Sorumluluk” olan bu prensipte amacımız; Her bir sınıfın tek bir amacı olup sadece tek bir amaca hizmet etmesidir. Daha anlaşılır olması için şöyle bir örnek ile pekiştirebiliriz.

Bir tane restoranda çalışan bir aşçı düşünelim, bu aşçının normalde tek görevi nedir? Yemek yapmak değil mi? ama biz gidip aşçıya bulaşıkları da sen yıka, servise de sen çık, masaları da sen temizle, hesabı da sen al, müşteri siparişlerini de sen al gibi restoranın bütün yükünü aşçıya yüklersem bu aşçı tek başına bu işin altından kalkamaz değil mi? Ama daha küçük ölçekte iş görevleri dağılımı yaparsak herkes kendi işini rahatlıkla yapabilir bunu proje üzerinden değerlendirecek olursak her class kendi işini yapacak şekilde oluşturulmalıdır. Her iş için ayrı bir class oluşturularak iş bölümü yapılmalıdır.



Yandaki resimdeki gibi bir canlandırma yapabiliriz. Farklı işlere yarayan objeler kendisi için ayrılmış bölmelerde(class) ayrı ayrı tutuluyor ve tümünün birleşimi ise bir projenin bütününü temsil ediyor.

2 – Open / Closed Principle

Türkçe çevirisi “Açık/Kapalı” olan prensip, projede geliştirilen nesnelerin geliştirilmeye açık ama değişime kapalı olmaları gerektiğini ifade eder.

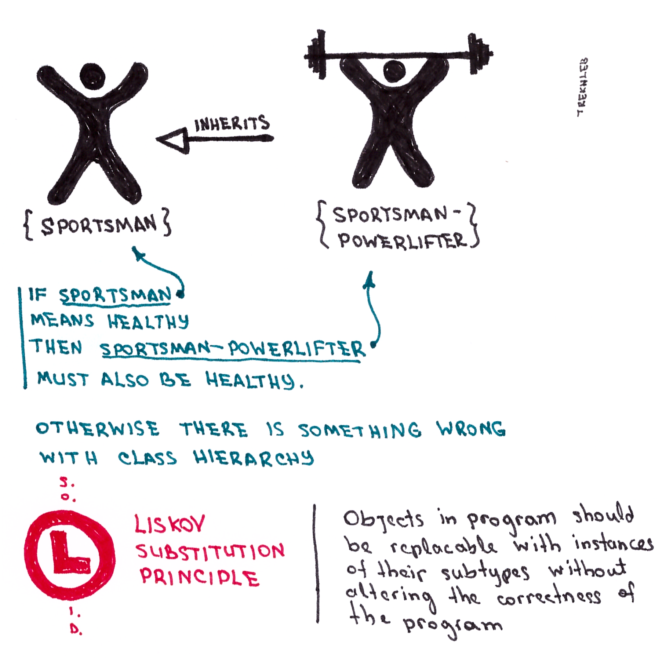
Bu prensip; sürdürülebilir ve tekrar kullanılabilir yapıda kod yazmanın temelini oluşturur. (Robert C. Martin)

metin, beyaz tahta içeren bir resim

Açıklama otomatik olarak oluşturulduYandaki resimdeki mixer üzerinden benzetme yapacak olursak mixerin çırma şeklini değiştirmek için mixerin motorunu şeklini vb. değiştirmemiz gerekmez sadece mixere takılıp çıkarılabilen başlıkları değiştirmemiz yetecektir. Open / Closed prensibi de yazacağımız kodları bu yapıya uygun yazmamız gerektiğini söyler. Prejemize ekleyeceğimiz yeni bir özellik projeyi baştan yazmamıza sebep olmamalı ilerde özellik ekleme, çıkarma dinamiğine sahip olacak şekilde projeler üretmeliyiz.

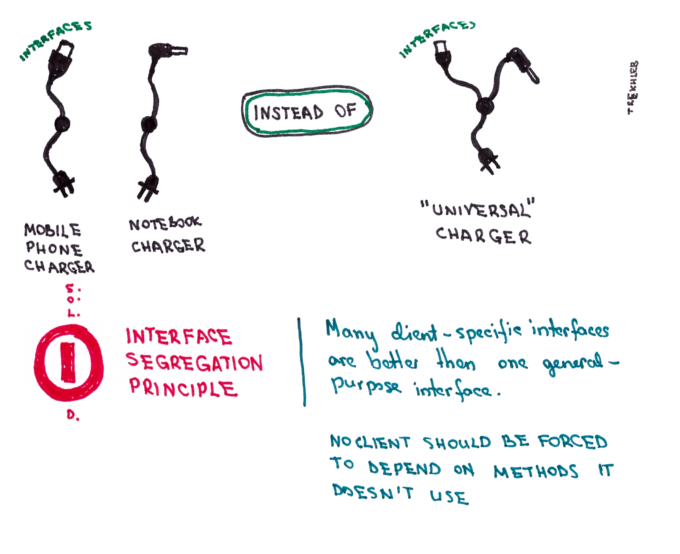
3 – Liskov Substitution Principle

Türkçeye “Yerine Geçme Prensibi” olarak çevrilen Liskov Substitution prensibine göre; miras alarak türemiş olan class’ların miras aldıkları nesnelerin tüm özelliklerini kullanması ve ekstra olarak kendi özelliklerini barındırabilmelidir.



Yandaki görselde görüldüğü gibi SPORTMAN-POWERLIFTER, SPORTMAN’ dan türemiştir. Bu durumda SPORTMAN sağlıklıdır dersek SPORTMAN-POWERLIFTER’ da sağlıklı olmak zorundadır. Ekstra olarak SPORTMAN-POWERLIFTER ‘ın kendine özel özellikleri de bulunur SPORTMAN’ da kaslı değildir ama SPORTMAN-POWERLIFTER kaslı olma özelliği olabilir.

4 – Interface Segregation Principle

 Türkçeye çevirisi “Arayüz Ayırımı” olan interface segregation prensibinde; bir interface’ e gerekenden fazla sorumluluk eklemek yerine, daha özelleştirilmiş birden fazla interface oluşturulmalıdır. Yani her farklı sorumluluğun kendine özgü bir arayüzü olması gerekmektedir. Böylece interface’ i kullanan kişide sadece ihtiyacı olanlarla ilgilenmiş olur. Örneğin bir oyun yazıyoruz diyelim silah diye bir nesnemiz var genel bir nesne (base class) ve çeşitli özellikte silahlar yapacağız. İlk silahımız taramı silah olsun ikinci silahımızda pompalı silah çeşidi olsun eğer tek bir interface olarak ateş etme özelliği yaparsak iki silahta aynı olur farkı olmaz bundan dolayı tüm işleri birbirinden ayırmalıyız taramalı ateş etme interface’si ayrı pompalı sıkma interface’ si ayrı ayrı yazılıp hangi özellik hangi silahta kullanılacaksa sadece o silaha tanımlanmalıdır.

5 – Dependency Inversion Principle

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu Türkçeye çevirisi “Bağımlılığın Ters Çevrilmesi” olan dependency inversion prensibinde; alt sınıflarda yapılan değişiklikler üst sınıfları etkilememelidir yani sınıflar arası bağımlılıklar olabildiğince az olmalıdır ve özellikle üst seviye sınıflar, alt seviye sınıflara bağımlı olmamalıdır.

Aklımızda daha rahat canlandırmak için bilgisayarımızı düşünebiliriz bilgisayarın kendisi base class olsun içindeki donanımlarda alt sınıflar olsun ilerde bilgisayarın fanı bozulduğunu, değişeceğini düşünecek olursak fan değişimi yapmamız bilgisayarı değiştirmeye zorlamaz bizi sadece ilgili alt parça gerekli değişimi alır üst parçaya dokunulmadan işlem tamamlanır.