# **SOLID Nedir ? Solid Yazılım Prensipleri Nelerdir ?**

Her yazılımcının bilmesi gereken ***S.O.L.I.D.*** yazılım geliştirmenin temel prensipleri olarak kabul edilmektedir.



SOLID Yazılım Prensipleri

**SOLID** yazılım prensipleri; geliştirilen yazılımın esnek, yeniden kullanılabilir, sürdürülebilir ve anlaşılır olmasını sağlayan, kod tekrarını önleyen ve Robert C. Martin tarafından öne sürülen prensipler bütünüdür. Kısaltması Michael Feathers tarafından tanımlanan bu prensiplerin amacı;

* Geliştirdiğimiz yazılımın gelecekte gereksinimlere kolayca adapte olması,
* Yeni özellikleri kodda bir değişikliğe gerek kalmadan kolayca ekleyebileceğimiz
* Yeni gereksinimlere karşın kodun üzerinde en az değişimi sağlaması,
* Kod üzerinde sürekli düzeltme hatta yeniden yazma gibi sorunların yol açtığı zaman kaybını da minimuma indirmektir.

Bu prensipler uygulanarak uygulamalarımızın büyürken, karmaşıklığın da büyümesinin önüne geçmiş oluruz. “**İyi kod**” yazmak için bu prensiplere uygun yazılım geliştirmelisiniz.

Biraz açalım;

## [**S — Single-responsibility principle**](https://gokhana.medium.com/single-responsibility-prensibi-nedir-kod-örneğiyle-soli̇d-c8b1602be602)

**ÖZET:** Bir sınıf (nesne) yalnızca bir amaç uğruna değiştirilebilir, o da o sınıfa yüklenen sorumluluktur, yani bir sınıfın(fonksiyona da indirgenebilir) yapması gereken yalnızca bir işi olması gerekir.

## [**O — Open-closed principle**](https://gokhana.medium.com/open-closed-prensibi-nedir-kod-örneğiyle-soli̇d-679619d5376a)

**ÖZET:**Bir sınıf ya da fonksiyon halihazırda var olan özellikleri korumalı ve değişikliğe izin vermemelidir. Yani davranışını değiştirmiyor olmalı ve yeni özellikler kazanabiliyor olmalıdır.

## [**L — Liskov substitution principle**](https://gokhana.medium.com/liskov-substitution-prensibi-nedir-kod-örneğiyle-soli̇d-3cfc1cd63c1a)

**ÖZET**: Kodlarımızda herhangi bir değişiklik yapmaya gerek duymadan alt sınıfları, türedikleri(üst) sınıfların yerine kullanabilmeliyiz.

## [**I — Interface segregation principle**](https://gokhana.medium.com/interface-segregation-prensibi-nedir-kod-örneğiyle-soli̇d-ac0fd6812ecf)

**ÖZET**: Sorumlulukların hepsini tek bir arayüze toplamak yerine daha özelleştirilmiş birden fazla arayüz oluşturmalıyız.

## [**D — Dependency Inversion Principle**](https://gokhana.medium.com/dependency-inversion-prensibi-nedir-kod-örneğiyle-soli̇d-b61296523565)

**ÖZET**: Sınıflar arası bağımlılıklar olabildiğince az olmalıdır özellikle üst seviye sınıflar alt seviye sınıflara bağımlı olmamalıdır.

Her bir prensibi ihlal eden ve doğru olan örnekleriyle birlikte detaylı olarak açıklayacağım bir yazı dizisi olacaktır. Burada hem uygulama yanlışlarını hem de dogru uygulanışını görerek daha iyi anlama ve teknik anlamda kolayca uygulanabilmesi amaçlanmıştır. S.O.L.I.D. temel olarak her yazılımcının öğrenmesi ve yazılım hayatı boyunca uygulanması gereken önemli prensipler bütünüdür.