**SOLİD PRENSİPLERİ**

İçindekiler

[***Tek Sorumluluk Prensibi (SRP)*** 1](#_Toc69726698)

[***Zayıf Bağlaşım Prensibi (LCP)*** 2](#_Toc69726699)

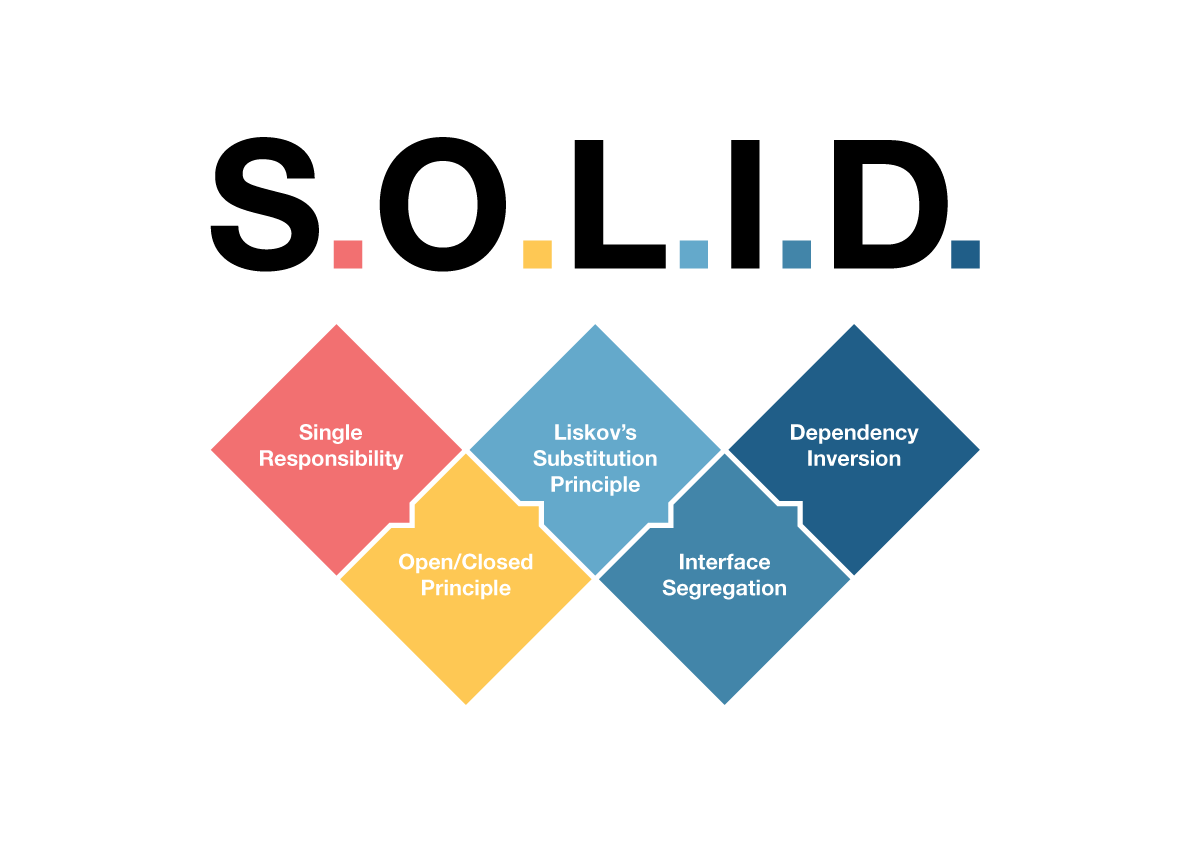
[***Açık/Kapalı Prensibi (OCP)*** 3](#_Toc69726700)

[***Liskov Yerine Geçme Prensibi (LSP)*** 4](#_Toc69726701)

[***Bağımlılığı Ters Çevirme Prensibi (DIP)*** 4](#_Toc69726702)

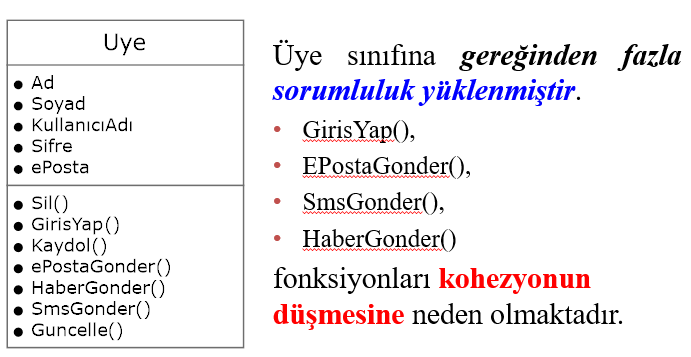
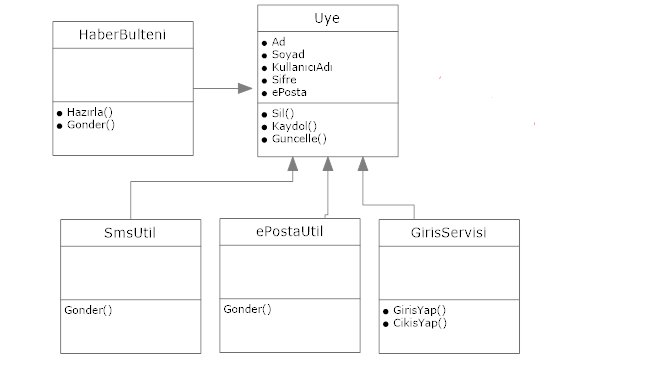
[***Arayüz Ayrım Prensibi (ISP)*** 5](#_Toc69726703)

[***Kaynakça*** 5](#_Toc69726704)



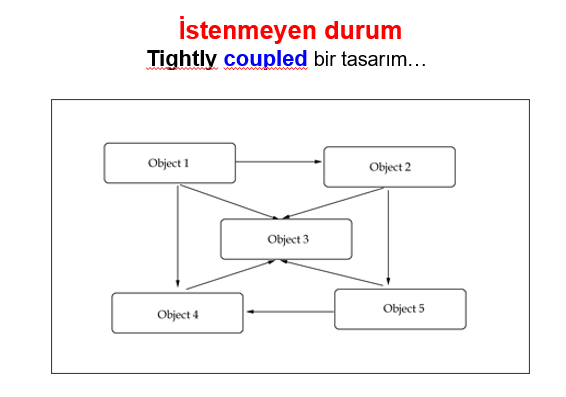
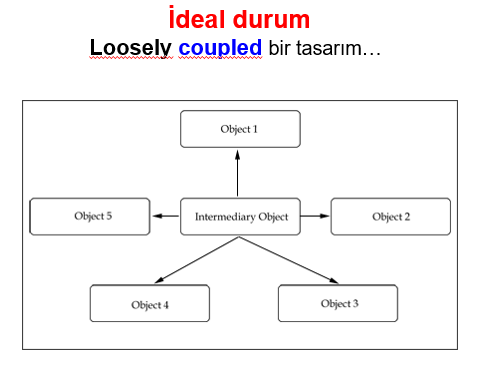
# Tek Sorumluluk Prensibi (SRP)

* Tek Sorumluluk Prensibi (Single Responsibility Principle) Kohezyon olgusuyla ***yakından ilişkili*** olup, bir modülün (ya da sını­fın) sadece tek bir sorumluluğu yerine getirmek üzere tasarlanmasını öngörmektedir.
* Zira gereksiz sorumluluklardan arındırılarak sadece belirli bir görevi yerine getirmek üzere tasar­lanmış bir sınıf elde etmenin yolu; sınıfın yazılma amacını çarpıtacak yani kohezyonunu düşürecek fonksiyonlara **yer vermemektir**
* Doğru bir tasarımda ileride olabilecek bir değişiklikte sadece değişen durumun sorumluluğunu üstlenmiş sınıf değiştirilebilir olması öngörülür.



Yüksek kohezyon(yapışıklık) yazılım sisteminin esnekliğini **arttırır**, bakım ve yeniden kullanılabilirliği **kolaylaştırır**.

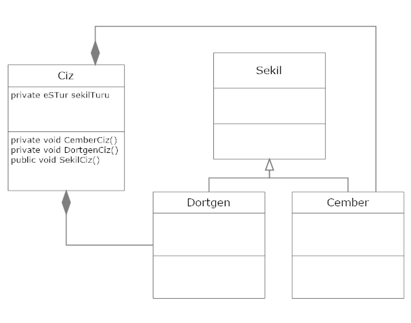
# Zayıf Bağlaşım Prensibi (LCP)

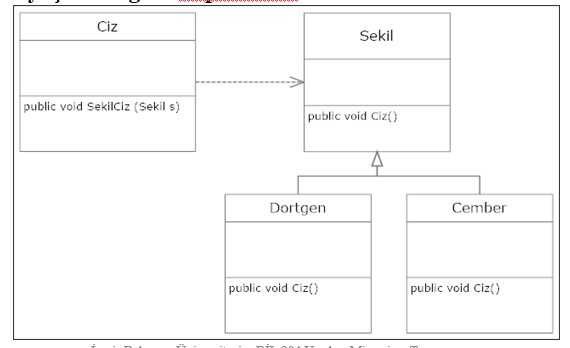


* **Coupling**, *nesneler arasındaki* ***ilişkilerin*** *nesneleri* ***birbirlerine ne kadar bağlı kıldığının*** *ölçütüdür*.
* Kendi içinde **kohezyonu yüksek** sınıfların birbirleriyle ilişkiler kurarken, bu ilişkide **coupling** faktörü olabildiğince **düşük tutulmalıdır**.
* Hight cohesion, low coupling( kısmen ilişki denebilir ).

# Açık/Kapalı Prensibi (OCP)

* Değişim gerektirmeyen yazılımın ömrü bitmiştir.
* Bu noktada yapılması gereken şey; yazılım sisteminin herhangi bir yerindeki değişimin başka yerlerde de zincirleme değişim gerektirmeyecek şekilde tasarlanmasıdır. Yani esnek olmalıdır.
* *Sistemler gelişime açık, ancak değişime kapalı olmalıdır.*
* Bu prensibe göre, sistemlerdeki değişimleri kodları değiştirerek gerçekleştirmek yerine **yeni kod blokları eklenerek** yapılması öngörülmektedir.
* Composition ilişkinin yoğun olduğu tasarımlarda esneklik yok denecek kadar azdır. Tasarım esnek değildir.

**



# Liskov Yerine Geçme Prensibi (LSP)

* Türemiş sınıf nesnelerinin taban sınıf nesnesi yerine geçmesini öngörür.
* Daha açık bir ifadeyle, **taban sınıf** türündeki nesne üzerinde operasyon yapacak şekilde **geliştirilmiş bir fonksiyon**, bu sınıftan türeyen farklı **sınıflara** ait nesneler üzerinde de aynı operasyonu yapabilmelidir.
* Bu prensip türemiş sınıf türündeki nesnelerin taban sınıfa ait nesnelere atanması *halinde gerçekleşen* **otomatik tür dönüşümünden** (**upcast**) **faydalanır**.

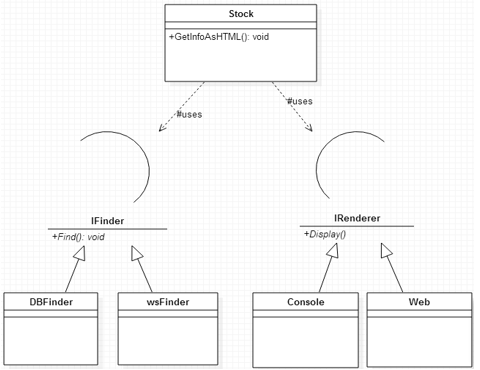
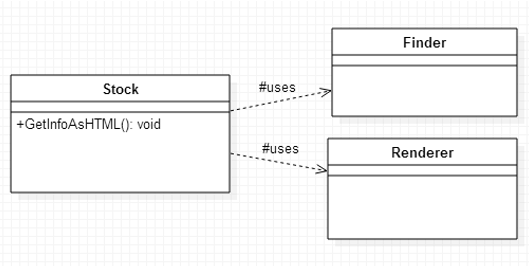
# Bağımlılığı Ters Çevirme Prensibi (DIP)

* Yüksek seviyeli sınıfların, düşük seviyeli sınıflarla doğrudan bir bağımlılığının olmamasını öngörmektedir.

*Bağımlılığın artmaması için*

* Yüksek seviyeli sınıflar ile
* Düşük seviyeli sınıfların arasına

bir ara yüz ya da soyut sınıf sokulması gerekmektedir.

****

# Arayüz Ayrım Prensibi (ISP)

* Sınıflar, ihtiyaç duymadıkları metotların bulunduğu Interface’lere bağlı olmaya zorlanmamalıdır.

# Kaynakça

* Doç. Dr. Deniz Kılınç Yazılım Mimari ve Tasarımı Ders Slaytları
* <https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design>
* <https://en.wikipedia.org/wiki/SOLID>
* <https://stackify.com/solid-design-principles/>
* https://www.baeldung.com/solid-principles