Nesne Yönelimli Analiz ve Tasarım

Nesne Tasarımı

Tasarım İlkeleri: SOLID

Tasarım İlkeleri
S.O.L.I.D.

* Tasarım sırasında kullanılan modüllerin bağımlılıkların yönetimiyle ilgili ilkelerdir.
* Modüllerin bağımlılığı iyi yönetilirse;
  * tasarım daha kararlı olur, gerçeklemedeki değişikliklerden (çok) etkilenmez
  * "cohesion" artar "coupling" azalır.
  * sonradan değişiklik kolaylaşır
  * kod tekrar kullanımı/modülerlik artar
  * yeni özellik kazandırma kolaylaşır
  * anlaşılabilirlik artar
  * karmaşıklık azalır (gereksiz bağımlılıklar ortadan kalkar)
  * birim test yazımı kolaylaşır

Robert C. Martin (2000). "Design Principles and Design Patterns"

Tasarım İlkeleri
S.O.L.I.D.

* Tasarım ilkeleri ve desenleri, genellikle çok sayıda modül kullanımına neden olur. Bu ise başarımı (efficiency) düşürebilir.
* Dolayısıyla, uygulamanın ihtiyaçlarına bağlı olarak bu ilkeler ihlal edilebilir.
* Örneğin, bağlı olunan modül değişmeyecekse "DIP" uygulamaya gerek kalmaz. Yazılım genişlemeyecekse "OCP" ve strateji desenini kullanmaya gerek kalmaz...

Tasarım İlkeleri
S.O.L.I.D.

SRP	The Single Responsibility Principle	*A class should have one, and only one, reason to change.*
OCP	The Open Closed Principle	*You should be able to extend a classes behavior, without modifying it.*
LSP	The Liskov Substitution Principle	*Derived classes must be substitutable for their base classes.*
ISP	The Interface Segregation Principle	*Make fine grained interfaces that are client specific.*
DIP	The Dependency Inversion Principle	*Depend on abstractions, not on concretions.*

http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod

The Single Responsibility (SRP)

A class should have only one, and only one, reason to change.

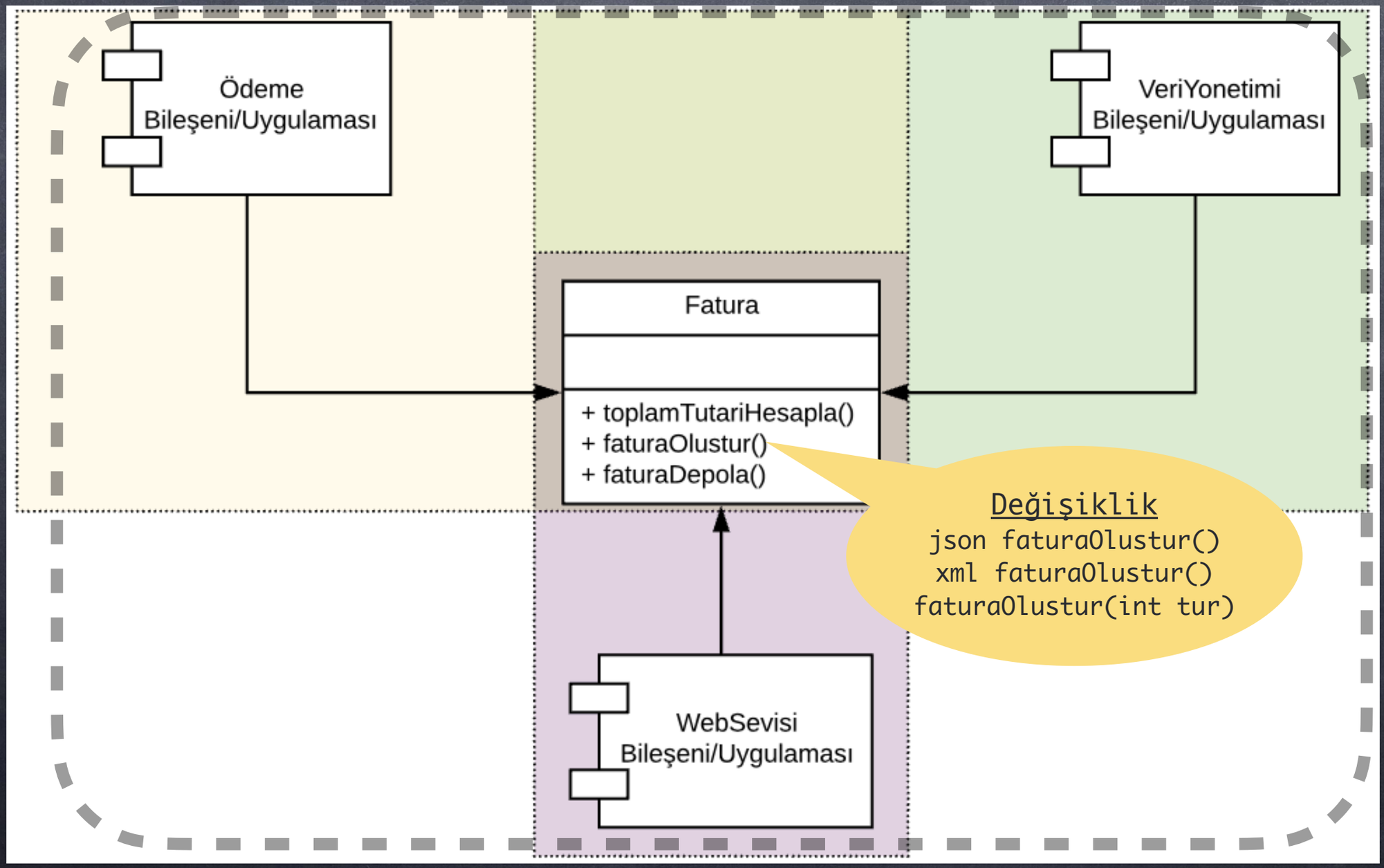
Agile Software Development; Robert C. Martin; Prentice Hall, 2003

Single Responsibility

* Bir sınıfın(modülün) değişmesi için tek bir neden olmalıdır. Her modülün tek bir sorumluluğu olmalıdır
* Bir sınıf için; genel yönetim(business logic), veri yönetimi (saklama, erişme v.s.), sunum (gui, json, xml v.s.), iletim (gönder ,al v.s.), nesne oluşturma (factories), main vb. işlemlerinin her biri farklı sorumluluk olarak ele alınmalıdır.
* Bir modülün tek bir sorumluluğu olmalı ve bu sorumluluğu mükemmel olarak yerine getirmelidir.
* Bir sorumluluğu yerine getirirken çok sayıda nesne ve üye fonksiyon kullanabilir.
* Bir modüle birden fazla sorumluluk yüklenirse;
    * değişiklik yapmak zorlaşır ve bakım aşamasının maliyeti yükselir
        * değişikliğin yapıldığı modüle doğrudan bağlı modüllerde de değişiklik gerekebilir.
        * doğrudan bağlı olmayan modüllerin tekrar derlenmesi, paketlenmesi, gönderilmesi v.s. gerekebilir
        * sonunda her değişiklik çok karmaşık hale gelir ve yan etkiler ortaya çıkar (side effects, regresyon hataları)
    * modülün tekrar kullanılabilme oranı azalır ve giderek projeye özel olmaya başlar (diğer projelerde, projenin diğer bölümlerinde kullanılabilme olasılığı azalır).
    * anlaşılması ve yönetilmesi zorlaşır.
    * birim testi yazmak zorlaşır

The Single Responsibility (SRP)

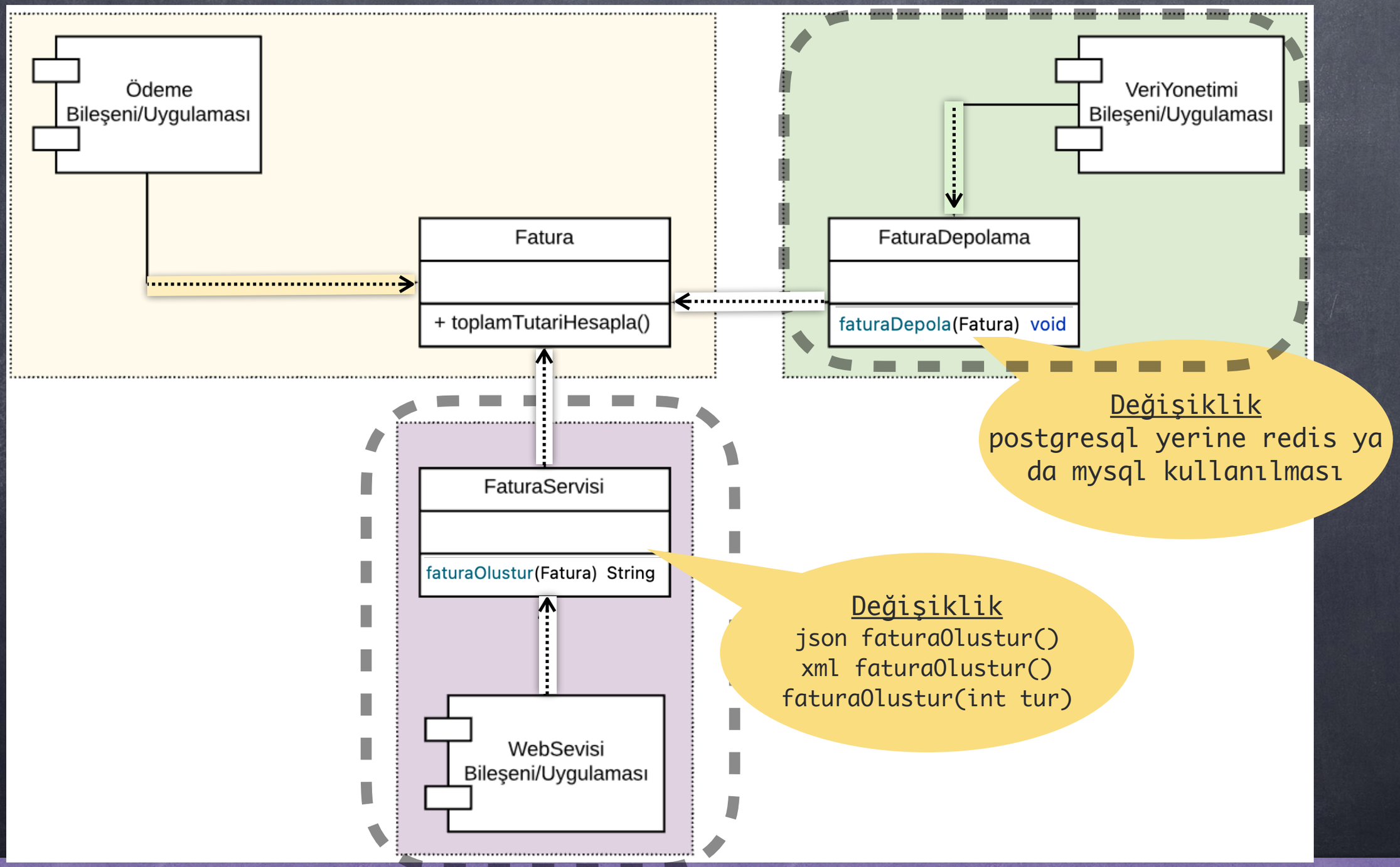
* Fatura modülü; fatura genel yönetimi (toplamTutarHesapla ve set/get yöntemleri), fatura servisi (faturaOlustur) ve fatura depolama sorumluluklarını yerine getiriyor.
* Örneğin; fatura servisindeki (faturaOlustur) bir değişiklik WebServisi modülünü doğrudan etkileyebilir. Ödeme ve VeriYonetimi modüllerini dolaylı olarak etkileyebilir.
* Fatura içerisinde herhangi bir sorumluluk nedeniyle yapılacak değişiklik tüm bileşenlerin yeniden derlenmesini / konuşlandırılmasını gerektirecektir.



https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders10/

The Single Responsibility (SRP)

* Aşağıdaki örnekte sorumlulukların her biri ayrı modüller tarafından yerine getirilmektedir.
* Örneğin; fatura servisindeki bir değişiklik sadece WebServisi modülünü etkileyecektir.
* Diğer bileşenlerin yeniden derlenmesini / konuşlandırılmasını gerektirmeyecektir.



https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders10/

The Open Closed (OCP)

A module should be open for extension but closed for modification.

Robert C. Martin (2000). "Design Principles and Design Patterns"

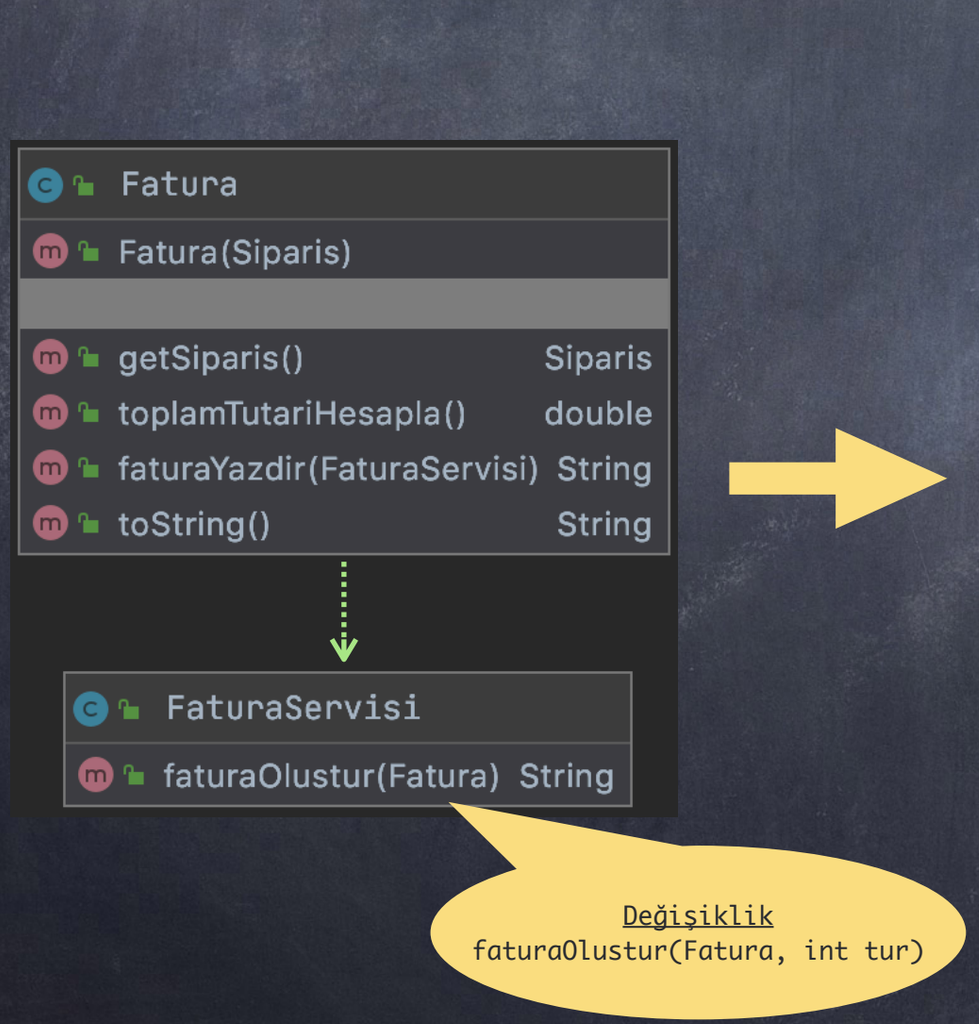
The Open Closed (OCP)

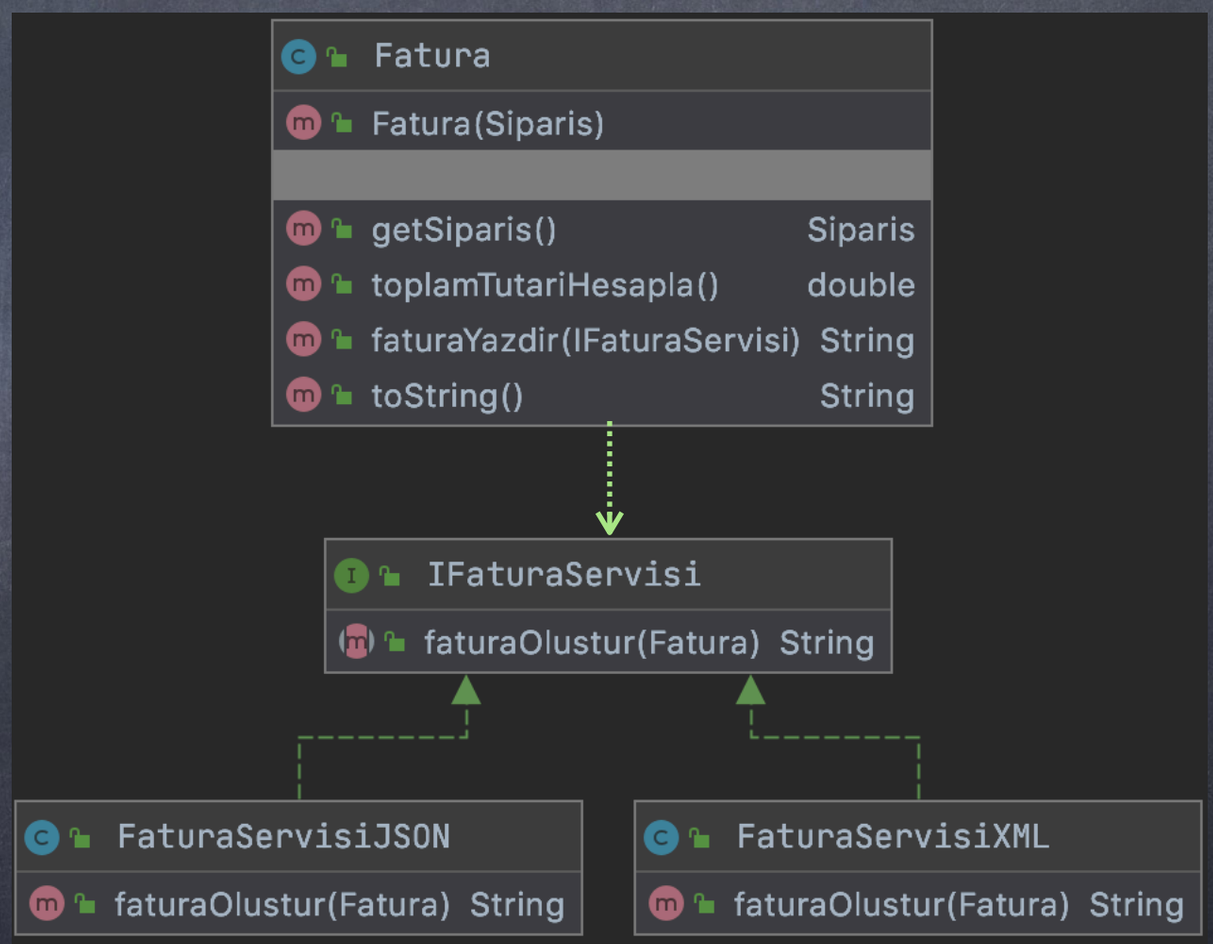
* Bu prensip, değişiklik yapmadan genişleyebilen modüller geliştirmeyi amaçlar.
* Mevcut kodu değiştirmeden yeni kodlar ekleyerek, yeni özellikler eklemeyi sağlar.
* "Open for extension": geliştirilen modüller yeni gereksinimler ortaya çıktığında bunları desteklemek üzere genişleyebilmeli.
* "closed for modification": geliştirilen modüller hata giderme dışında değiştirilmemeli.
* Mevcut kodda değişiklik yapmadan yeni özellik ekleyebilmenin yolu, bağlı olunan modüllerin gerçeklemeleri yerine soyutlamalarının kullanılmasıdır.

Robert C. Martin (2000). "Design Principles and Design Patterns"

The Open Closed (OCP)

* Soldaki örnekte, Fatura modülü FaturaServisi modülünün gerçeklemesine bağımlı. Gerçekleme içerisinde yapılacak değişiklik istemci modülü de etkileyebilir.
* Sağdaki örnekte, OCP ilkesi uygulanmıştır. Fatura modülü FaturaServisi modülünün soyutuna (IFaturaServisi) bağımlı. Gerçeklemelerdeki değişiklikler (yeni özellik gereksinimleri gibi) istemci modülü etkilemeyecektir.





https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders10/

The Liskov Substitution Principle (LSP)

Subclasses should be substitutable for their base classes.

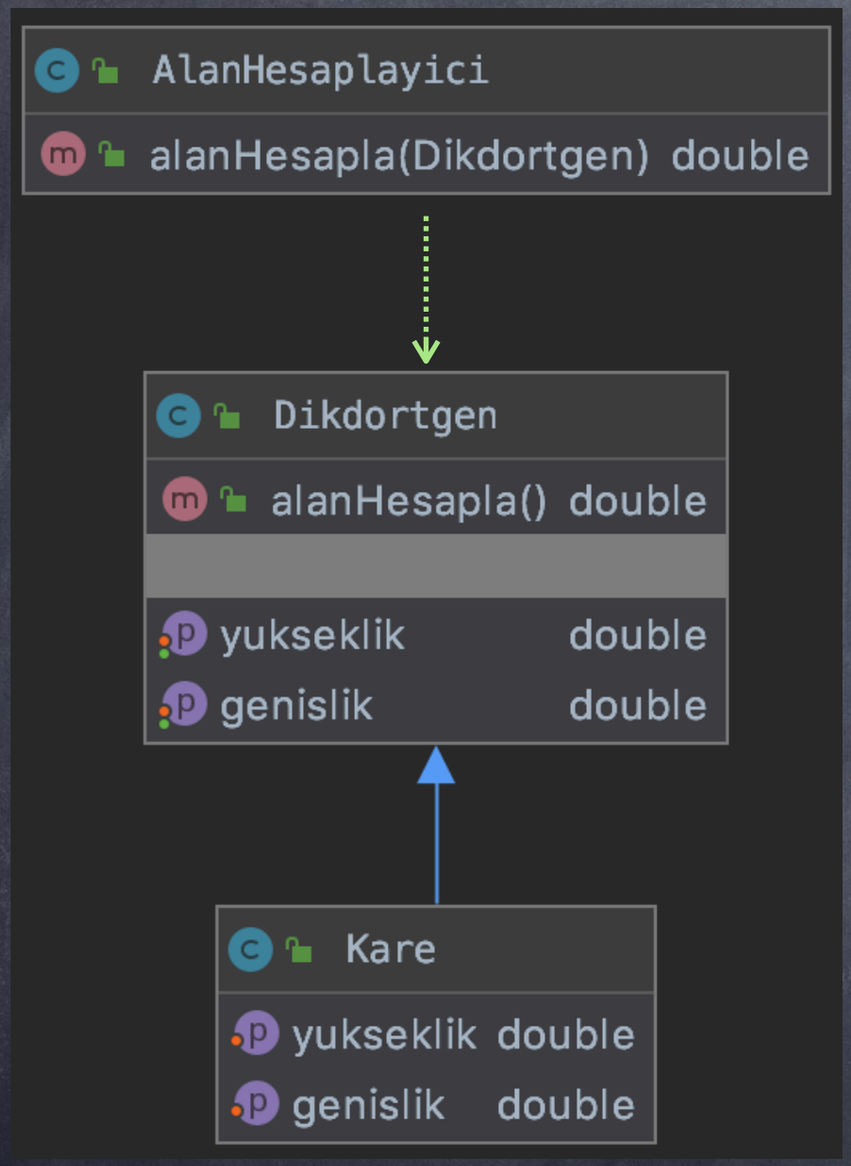
Robert C. Martin (2000). "Design Principles and Design Patterns"

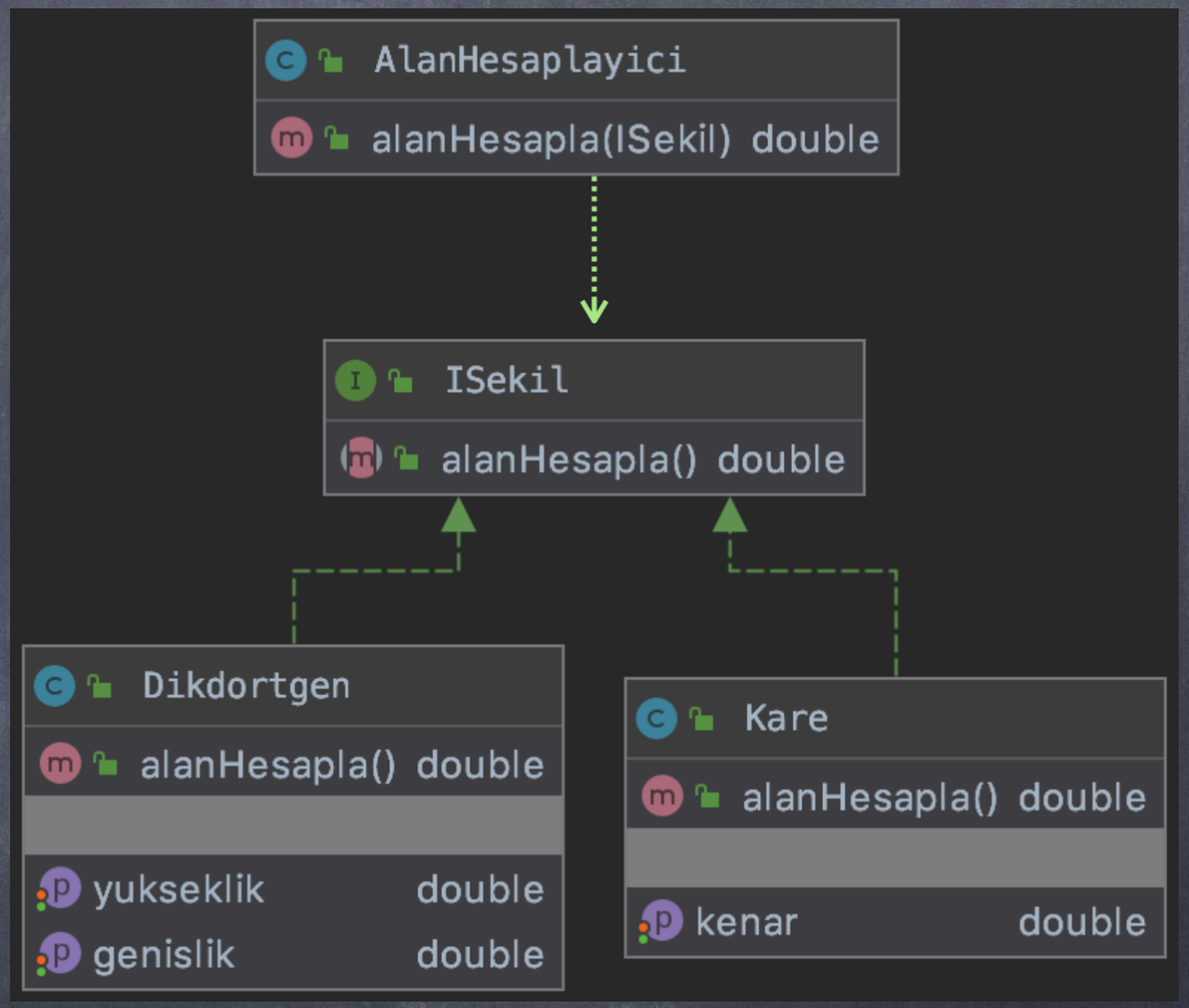
The Liskov Substitution Principle (LSP)

* Barbara Liskov tarafından 1988 yılında ortaya konan bir ilkedir.
* Temel sınıf nesneleriyle çocukların nesneleri, istemci modül içerisinde sorun çıkarmadan yer değiştirebilmelidir.
* Kalıtım hiyerarşisinin tutarlı olarak belirlenmesini sağlar.
* Open/Closed ilkesine uymayan hiyerarşilerin önüne geçmemizi sağlar

Robert C. Martin (2000). "Design Principles and Design Patterns"
Liskov, B. (May 1988). "Keynote address - data abstraction and hierarchy". ACM SIGPLAN Notices. 23 (5): 17–34. doi:10.1145/62139.62141

The Liskov Substitution Principle (LSP)





https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders10/

The Interface Segregation Principle (ISP)

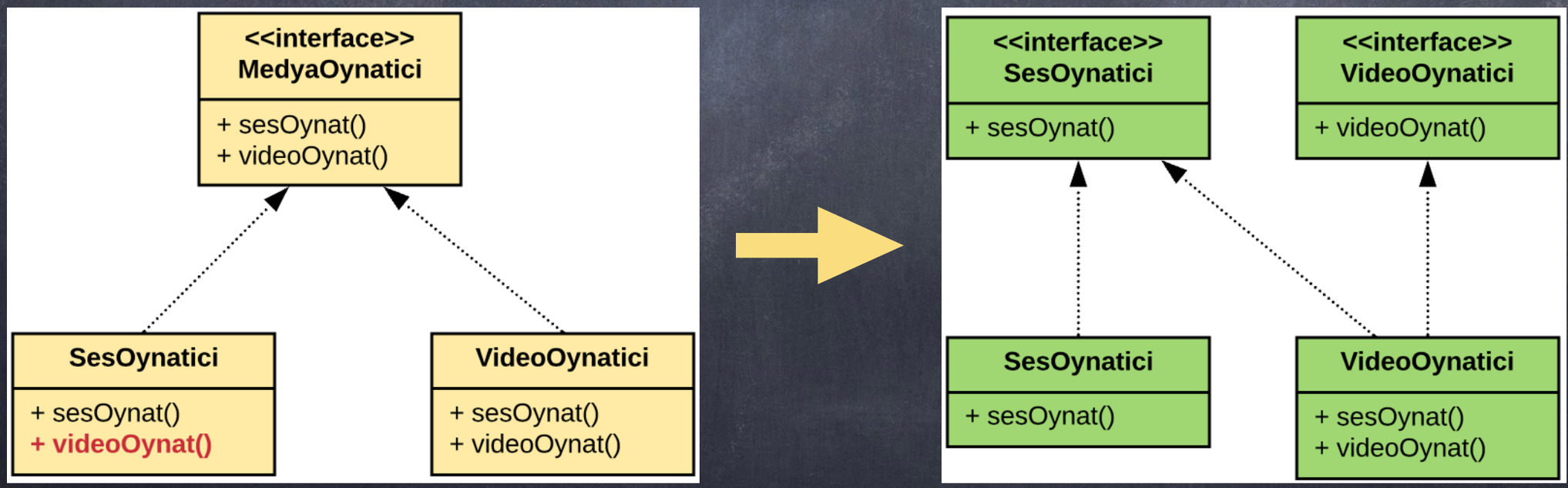
Many client specific interfaces are better than one general purpose interface.

Clients should not be forced to depend upon interfaces that they do not use.

Robert C. Martin (2000). "Design Principles and Design Patterns"

The Interface Segregation Principle (ISP)

* Bu prensip, modüllerin, kullanmayacakları arayüzlere bağlı kalmaması gerektiğini ifade eder.
* Kullanılmayacak arayüzlere bağlı kalmak, bunlarda yapılacak (bağlı diğer modüller tarafından) değişikliklerden etkilenme (side effects) anlamına gelmektedir. SRP ilkesinin ihlal edilmesidir.
* Böyle bir durum ortaya çıktığında arayüzlerin ayrılması gereklidir.
* Soldaki örnekte SesOynatici modülü, öyle bir fonksiyonu olmadığı halde, videoOynat() yöntemini içermek zorunda bırakılıyor-ISP nin ihlali.
* Çözüm sağdaki örnekte verilmiştir.



The Dependency Inversion Principle (DIP)

Depend upon Abstractions. Do not depend upon concretions.

High-level modules should not depend on low-level modules. Both should depend on abstractions.
Abstractions should not depend on details. Details should depend on abstractions.

Robert C. Martin (2000). "Design Principles and Design Patterns"

The Dependency Inversion Principle (DIP)

* Nesneler arasındaki bir bağlantıda, yüksek seviyeli modül ile düşük seviyeli modül (her ikisi birden) soyutlamaya bağlı olmalı. Her ikisi birbirine doğrudan değil arayüz üzerinden bağlanmalı.

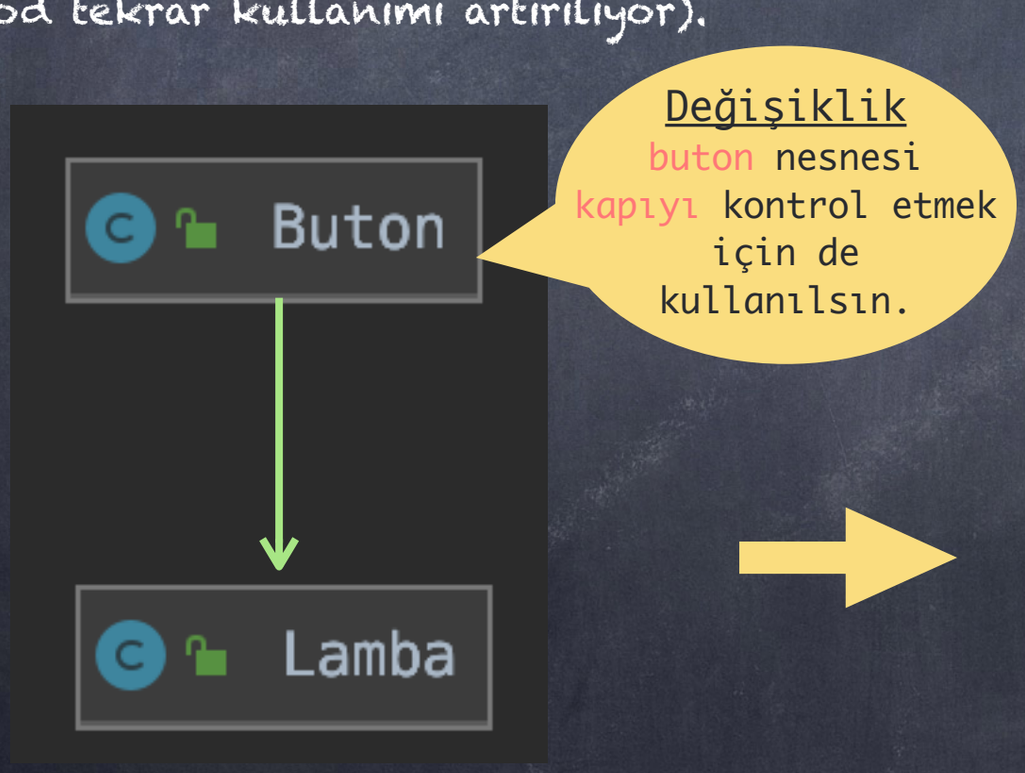
* Böylece modülde yapılacak değişiklik diğer modülleri etkilememiş olur. Değişiklik yapmak kolaylaşır. Yeni özellik eklemek kolaylaşır (OCP).

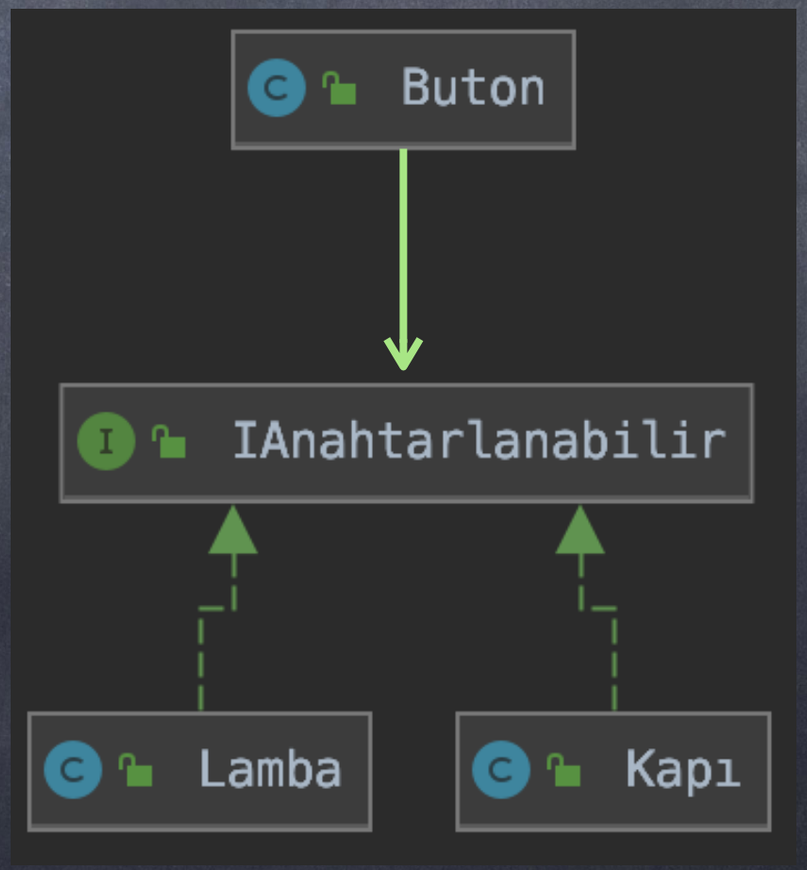
* Modüllerin tekrar kullanım oranı artar.

* OCP ve LSP uygulandığında, aynı zamanda DIP uygulanmış olur.

The Dependency Inversion Principle (DIP)

* Soldaki örnekte Buton nesnesi Lamba nesnesini (gerçekleme) kullanıyor (bağlı).
* Lambada yapılacak değişiklikten etkilenme ihtimali var.
* Buton nesnesinin tekrar kullanımı olanaksız. Sadece Lamba nesnesiyle kullanılabilir.
* Sağdaki değişiklik ile Buton nesnesinin ve Lambanın soyutlamaya (IAnahtarlanabilir) bağlı olması sağlanıyor.
    * Dolayısıyla aralarındaki bağlantı zayıflıyor. Birindeki değişikliğin diğerini etkilemesinin önüne geçilmiş oluyor.
    * Buton nesnesinin başka nesnelerle (örneğin; kapı, su ısıtıcısı v.b.) de kullanılabilmesi sağlanıyor (kod tekrar kullanımı artırılıyor).





https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders10/