

Nesne Yönelimli Analiz ve Tasarım

Nesne Tasarımı

Tasarım Desenleri
Design Patterns

Tasarım Desenleri

- * Yazılımlar geliştirilirken karşılaşılan genel tasarım problemlerini tanımlayarak, bu problemin en uygun nasıl çözüleceğini ((high coherence, low coupling) kod tekrar kullanımını artırmak ve değişikliği kolaylaştmak için) ve çözümün ortaya çıkaracağı sonuçları anlatır.
- * Programlama dillerinden bağımsızdır.
- * Yazılım geliştiricilerin tasarımlarla ilgili tartışma yapmasını kolaylaştırır.
- * Tasarım desenleri dört bölümden oluşur
 - * Desenin adı
 - * problemin tanımı: desenin ne zaman/hererde kullanılabileceği
 - * çözüm: problemin nasıl çözüleceği (kullanılması gereken bileşenler, aralarındaki bağıntı vb.)
 - * sonuç: deseni uygulamanın sonuçları?

Tasarım Desenleri

- * İlk olarak Erich Gamma vd. tarafından yazılan aşağıdaki kitapla, yazılım geliştirmede tasarım deseni konusu ortaya çıkmıştır. Bu kitapta 23 tasarım deseni yer almaktadır. Bunlar dışında da çok sayıda desen bulunmaktadır.
- * Creational(Nesne oluşturma): Nesnelerin uygun bir şekilde oluşturulmasını sağlayacak mekanizmalar içerir.
- * Structural(Yapısal): Nesnelerin sistemler içerisinde uygun olarak yerleştirilmesini sağlayacak desenlerdir.
- * Behavioral(Davranışsal): Nesnelerin birbirleriyle uygun olarak etkileşiminini düzenleyen mekanizmalar.

Creational (Nesne oluşturma)	Structural (Yapısal)	Behavioral (Davranışsal)
Singleton Pattern	Adapter Pattern	Template Method Pattern
Factory Pattern	Composite Pattern	Mediator Pattern
Abstract Factory Pattern	Proxy Pattern	Chain of Responsibility Pattern
Builder Pattern	Flyweight Pattern	Observer Pattern
Prototype Pattern	Facade Pattern	Strategy Pattern
	Bridge Pattern	Command Pattern
	Decorator Pattern	State Pattern
		Visitor Pattern
		Interpreter Pattern
		Iterator Pattern
		Memento Pattern

Tasarım Desenleri : Singleton

- * Amaç: Bir sınıfın yalnızca tek nesnesi olmasını ve bu nesneye global olarak erişilmesini sağlamak.
- * Nesne oluşturmaya (creational) ilgili desenlerden biridir.
- * Nesne yoksa oluşturulur döndürülür, varsa olan döndürülür.
- * Nesnenin new komutu ile dışarıdan oluşturulabilmesini engellemek için yapıci yöntem "private" yapılır.
- * Yapıci gibi çalışan "static" bir yöntem tanımlanır. Nesne oluşturma görevi bu yöntemindir. Oluşturulan nesne "static" bir üye saklanır.

Tasarım Desenleri : Singleton

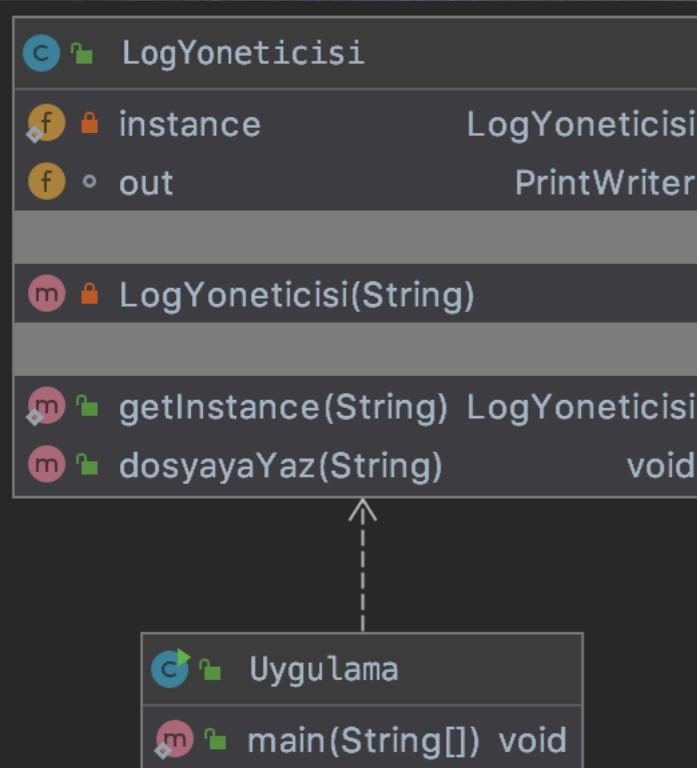
```
public class LogYonetici {
    private static LogYonetici instance;
    PrintWriter out;

    private LogYonetici(String logDosyasi){
        try {
            out = new PrintWriter(new FileWriter(logDosyasi,true), true);
        } catch (IOException e) {e.printStackTrace();}
    }

    public static synchronized LogYonetici getInstance(String logDosyasi){
        if(instance==null)
            instance = new LogYonetici(logDosyasi);
        return instance;
    }

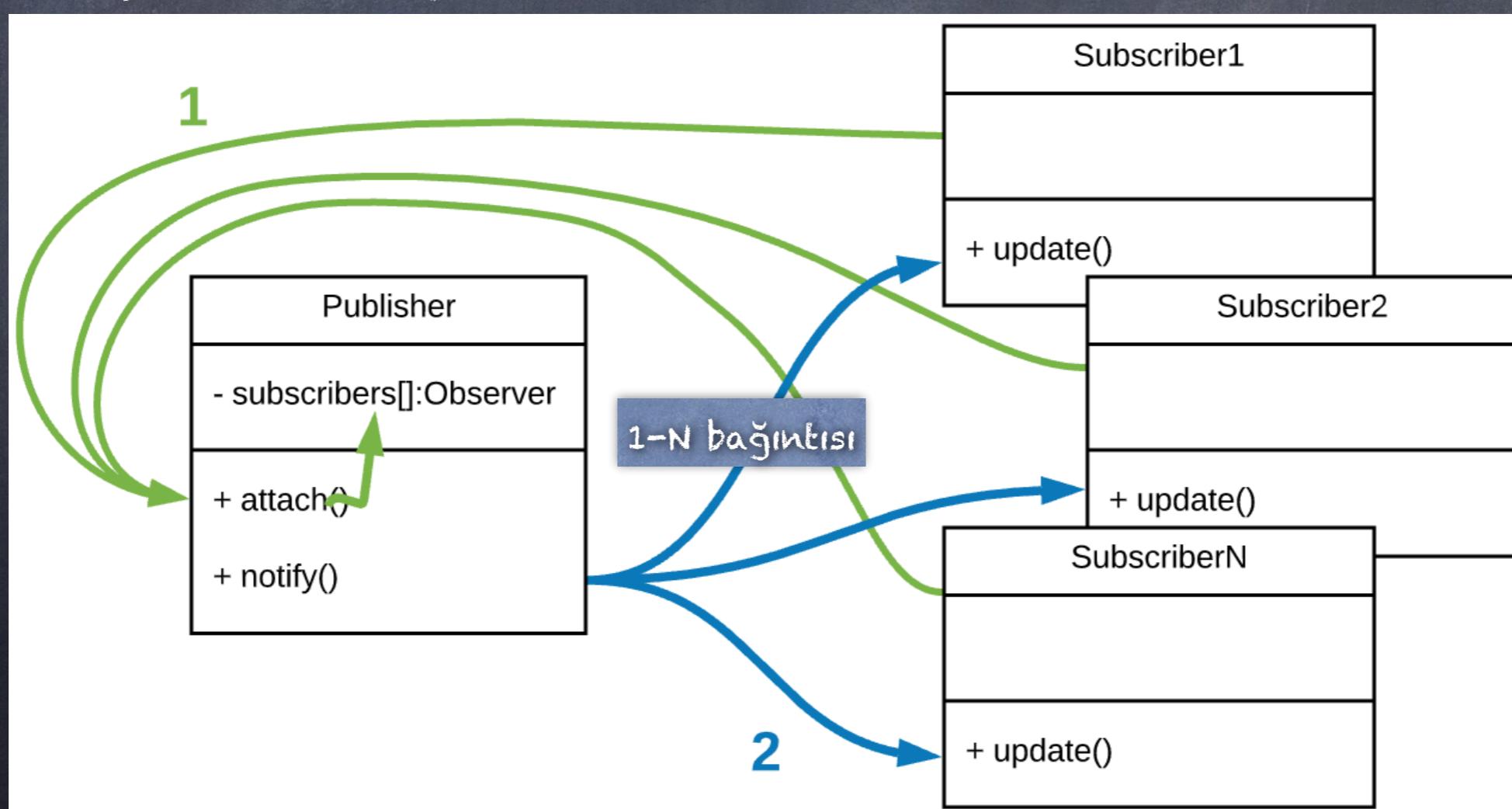
    public void dosyayaYaz(String mesaj) {
        out.println(LocalDateTime.now()+"："+mesaj);
    }
}

public class Uygulama {
    public static void main(String [] args){
        LogYonetici.getInstance("Log.txt").dosyayaYaz("[WARNING] :uyari mesajı 1");
    }
}
```



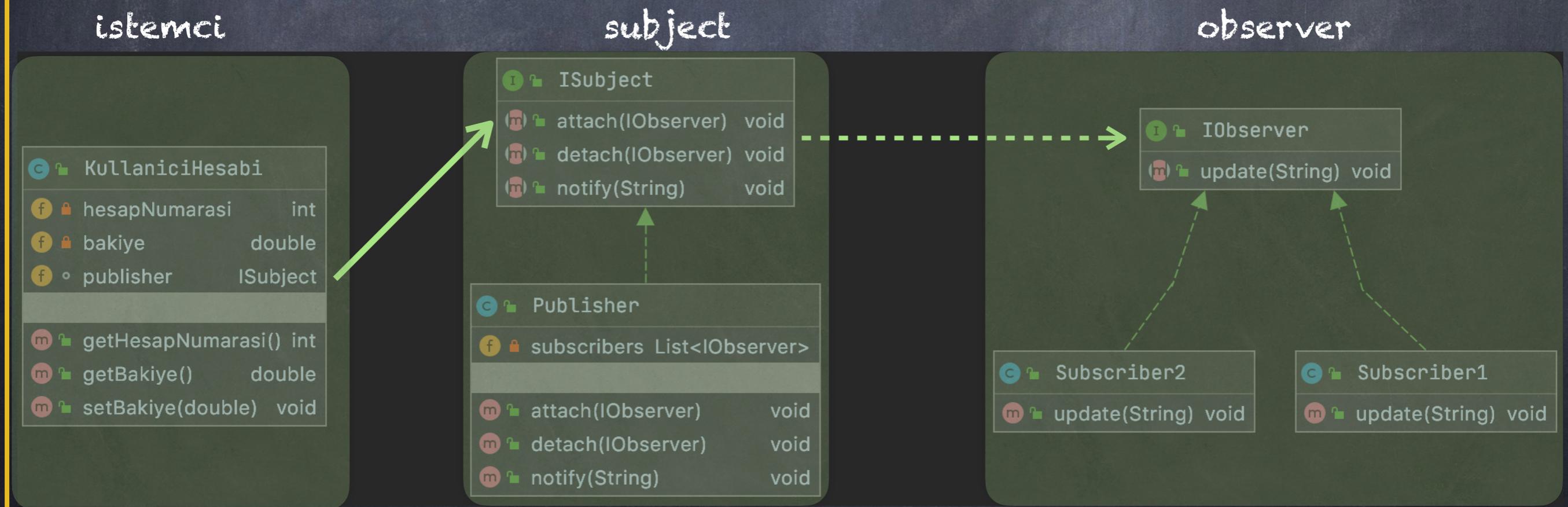
Tasarım Desenleri : Observer (Gözlemevi)

- * Amaç: Çok sayıda nesneye, gözlemledikleri nesnede meydana gelen olayı bildirmek.
- * Davranışsal desenlerden biridir.
- * Kullanım örnekleri:
 - * mağazaya ürün geldiğinde ilgili müşterilere bildirim gönderilmesi, ürün indirime girdiğinde bildirim gönderilmesi
 - * bakiye değiştiğinde müşteri, loglama sistemi ve gerçek zamanlı görüntüleme/ anormal durum tespit sistemine bilgi gönderilmesi vb.
- * Böyle bir etkileşim Publish(Yayın)-Subscribe(abone) olarak da adlandırılır.



Tasarım Desenleri : Observer (Gözlemci)

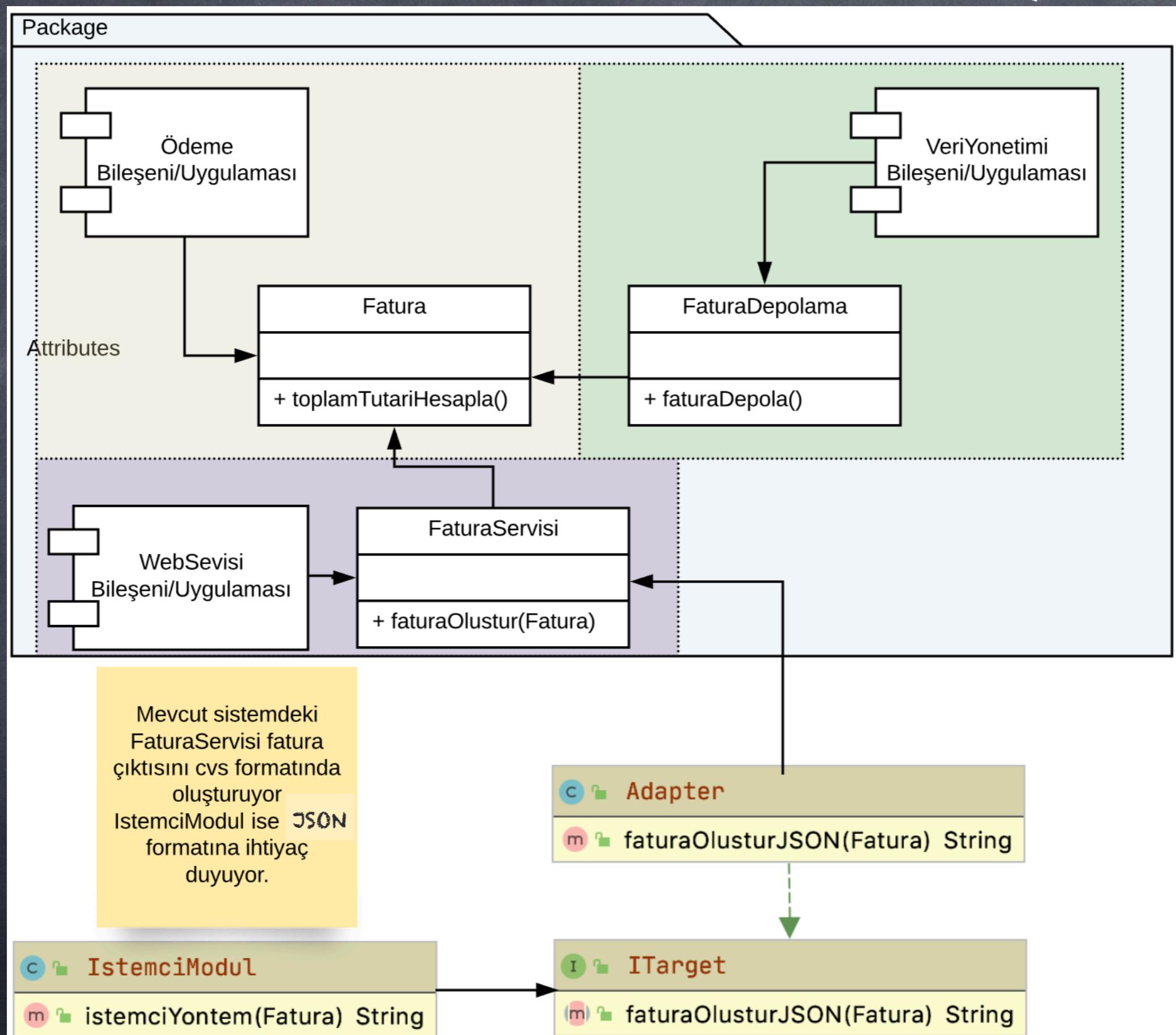
- * Subject (publisher- yayıncı): gözlemcilerin kaydedilmesi/ çıkarılması, istemcideki değişikliklerin gözlemcilere bildirilmesi işlemlerinden sorumludur.
- * Observer (subscriber-abone): istemcideki olayların yansıtıldığı/gönderildiği nesne.
- * İstemci (kullanıcıHesabı): olayın üretildiği nesne.



Tasarım Desenleri : Adapter

- * Yapısal desenlerden biridir.
- * Bir sınıfın arayüzüünü istemci tarafından beklenen başka bir arayüze dönüştürmek için kullanılır.
- * Mevcut sınıfların (kaynak kodunu değiştirmeden) diğer sınıflarla çalışabilmesini sağlar.
- * Bu desen sayesinde, daha önceden geliştirilmiş modüller, yeni modüller ile birlikte (bu modüllere uygun arayzlere sahip olmadığı durumlarda) kullanabiliriz.

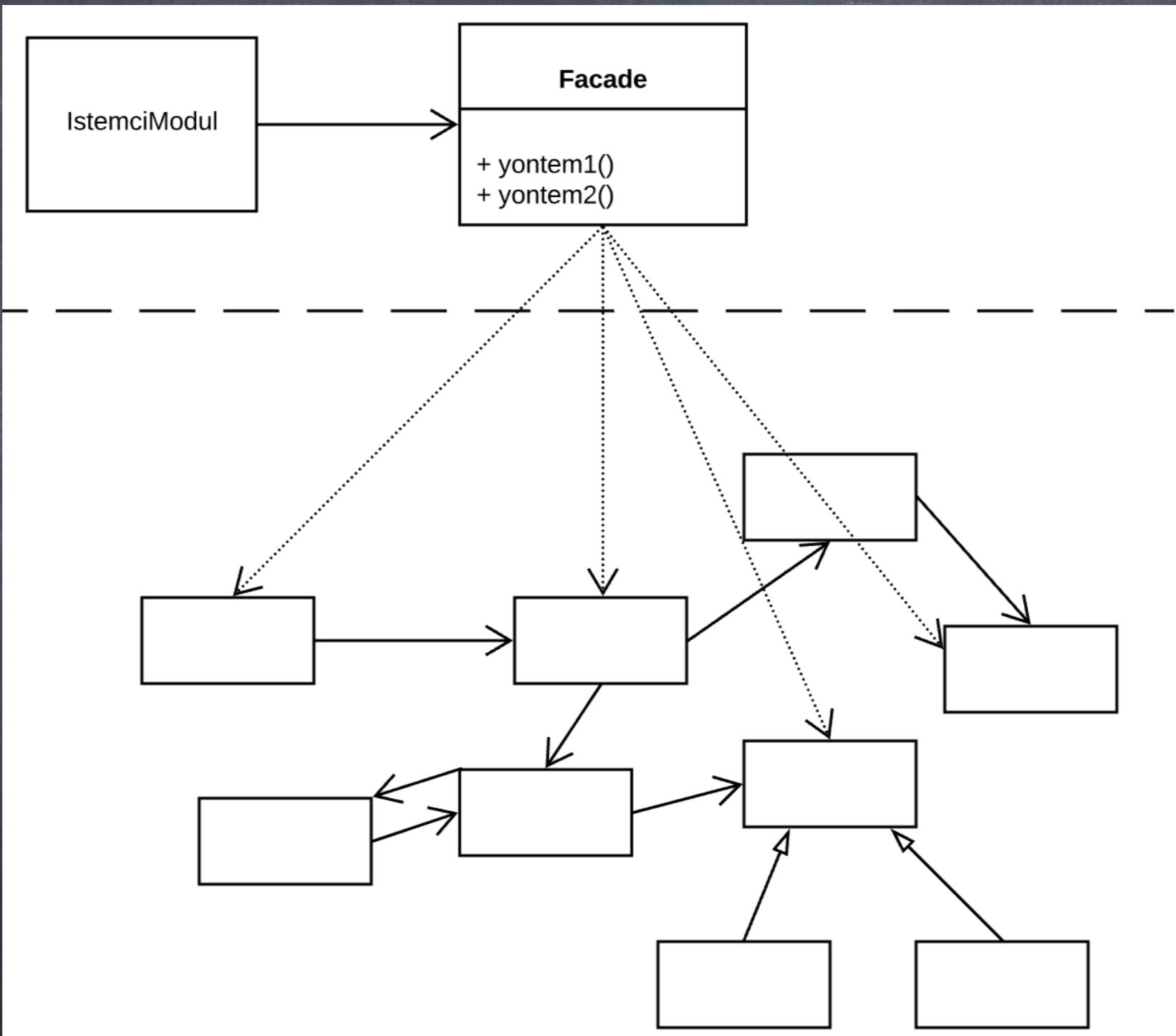
Tasarım Desenleri : Adapter



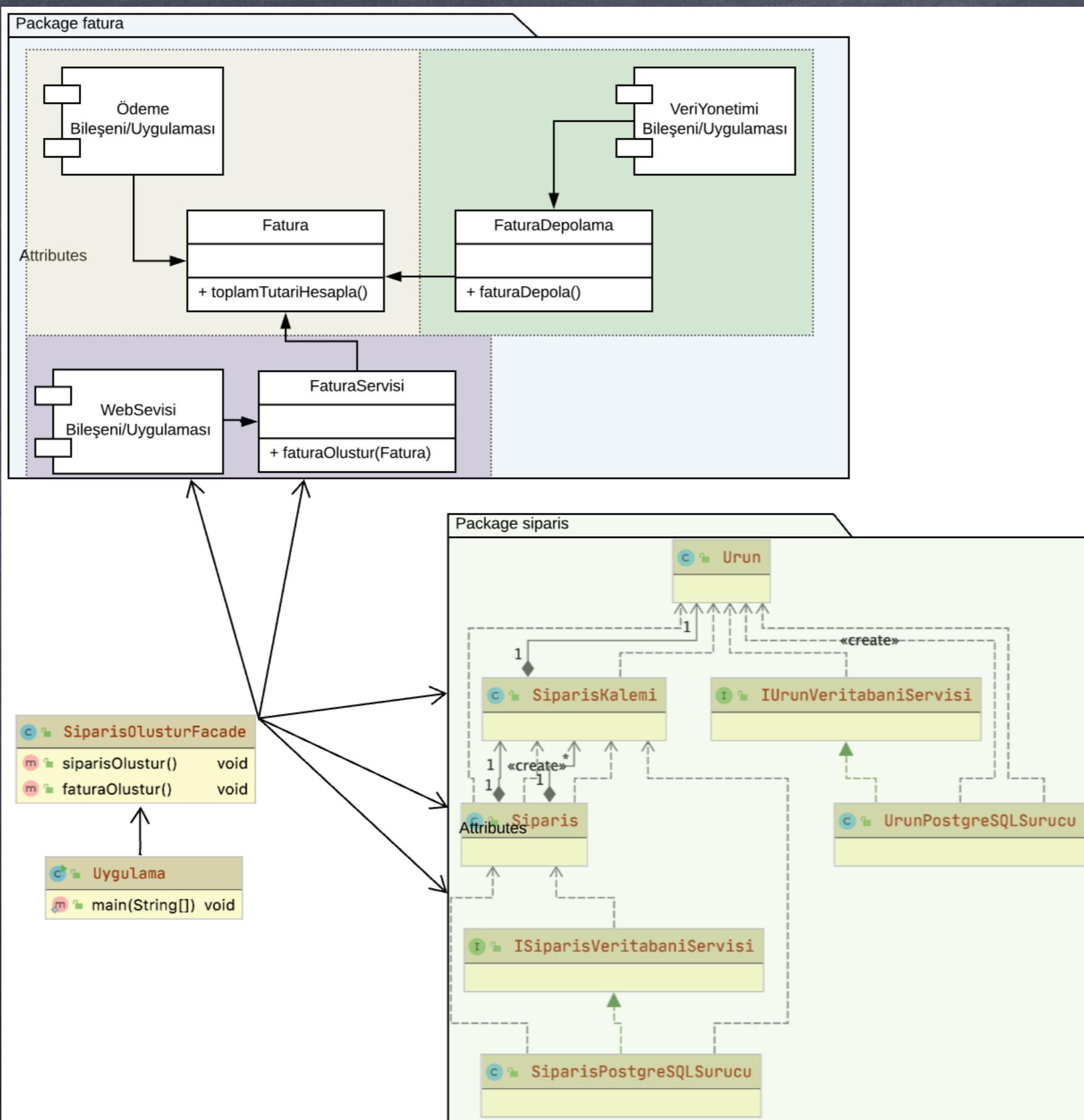
Tasarım Desenleri : Facade

- * Yapısal desenlerden biridir.
- * Karmaşık yapıdaki bir sınıf topluluğu (kütüphane, alt bileşen, eskiden yazılmış kodlar vb.) için basitleştirilmiş arayüz sağlar.
- * İstemci kodun karmaşık alt sistemle etkileşimiğini kolaylaştırır (Loosly coupling).
- * İyileştirilme imkanı olmayan eski modüllerle etkileşim imkanı sağlar.
- * Anlaşılabilirliği artırır.
- * Yazılım içerisinde katmanlar oluşturulmasına imkan verir.

Tasarım Desenleri : Facade



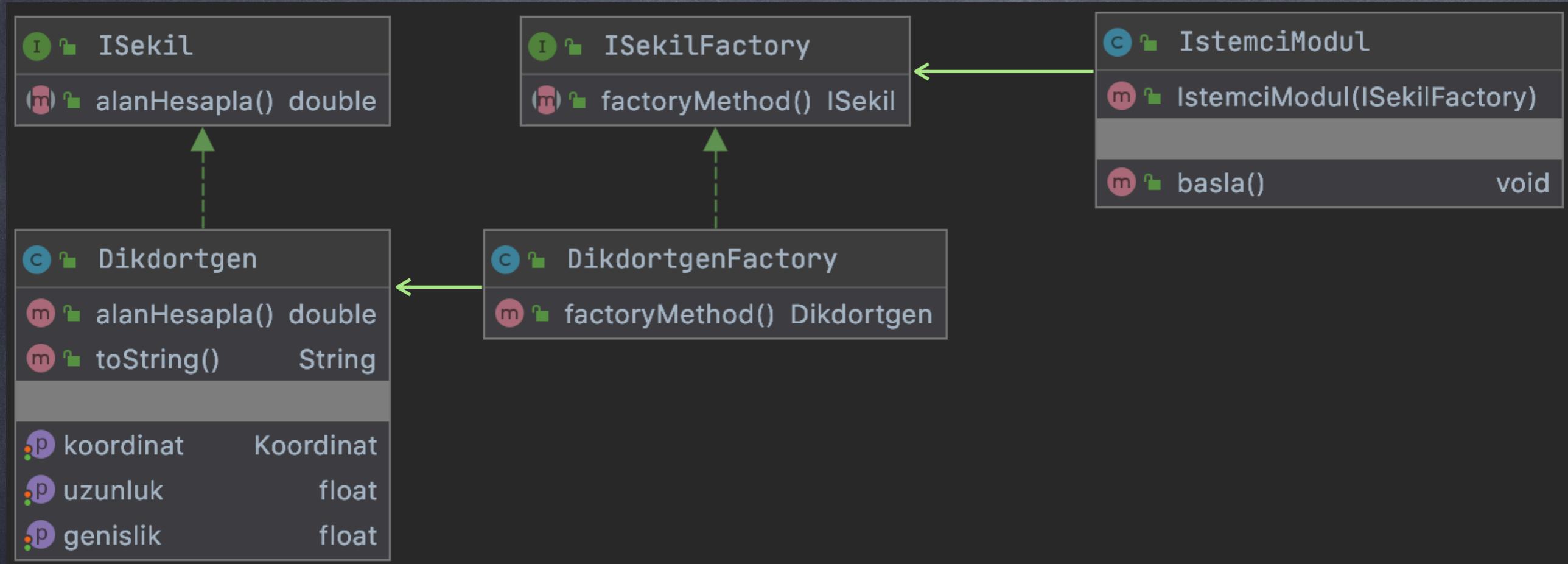
Tasarım Desenleri : Facade



Tasarım Desenleri-Factory Method

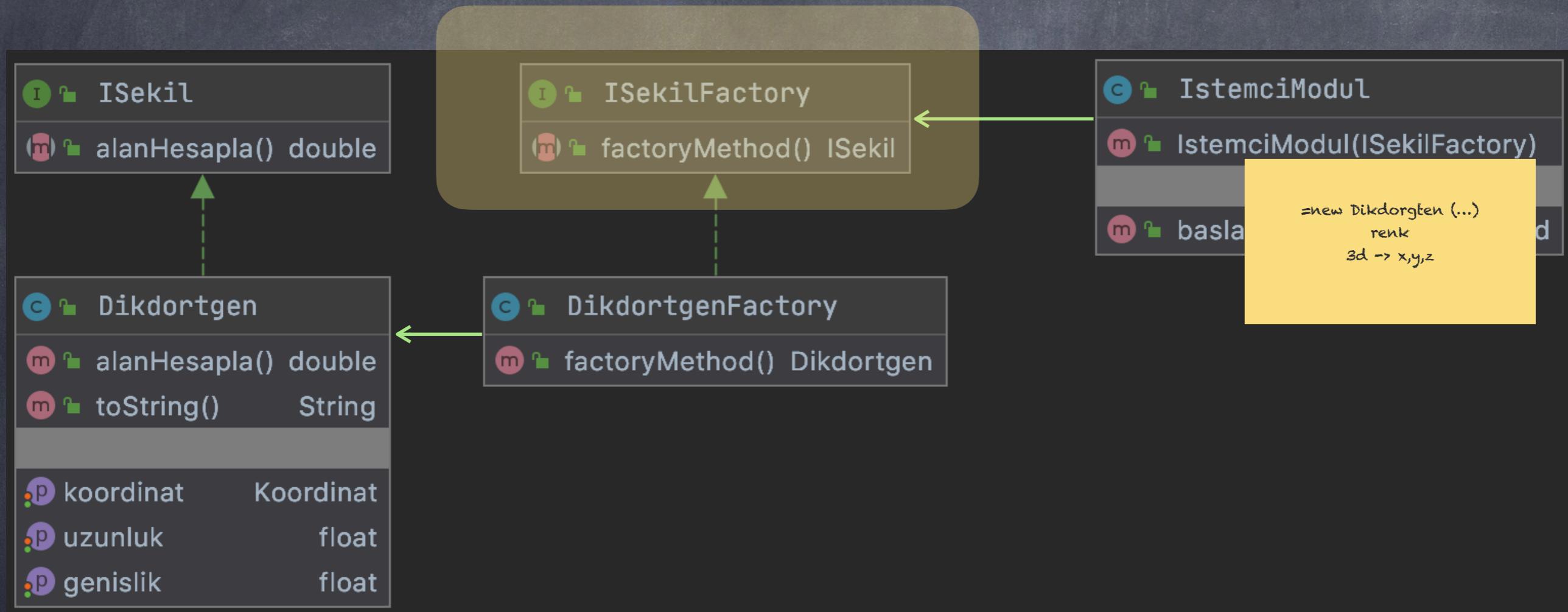
- * Nesne oluşturmayla (creational) ilgili desenlerden biridir.
- * Bir sınıfın nesne oluşturmak gerekiğinde, bu sorumluluğu istemci koddan ayırmak (kapsülleme/SRP) için kullanılır.
- * Özellikle çok sayıda parametre göndermek gerekiğinde (kalitim söz konusu ise) nesne oluşturma işi karmaşıklaşır (kod tekrarı, değişikliklerden istemci kodun etkilenmesi, kodların kötü görünmesi...).
- * Nesne oluşturmak gerekiğinde "factory method" çağırılarak nesne oluşturulur. Sınıfın bulunduğu yol/paket, istisna yönetimi v.s. uğraşmaya gerek yok.
- * Nesne oluşturmayla ilgili herhangi bir değişiklikten (yolların değişimi, parametre değişimleri v.s.), istemci kod etkilenmez.

Tasarım Desenleri-Factory Method (uygulama1)



Tasarım Desenleri-Factory Method (uygulama1)

İstemci modül içerisindeki şekil dikdörtgen olacaksı ve değişme ihtiyacı yoksa DI uygulamaya gerek yok.



Tasarım Desenleri-Factory Method (uygulama2)

