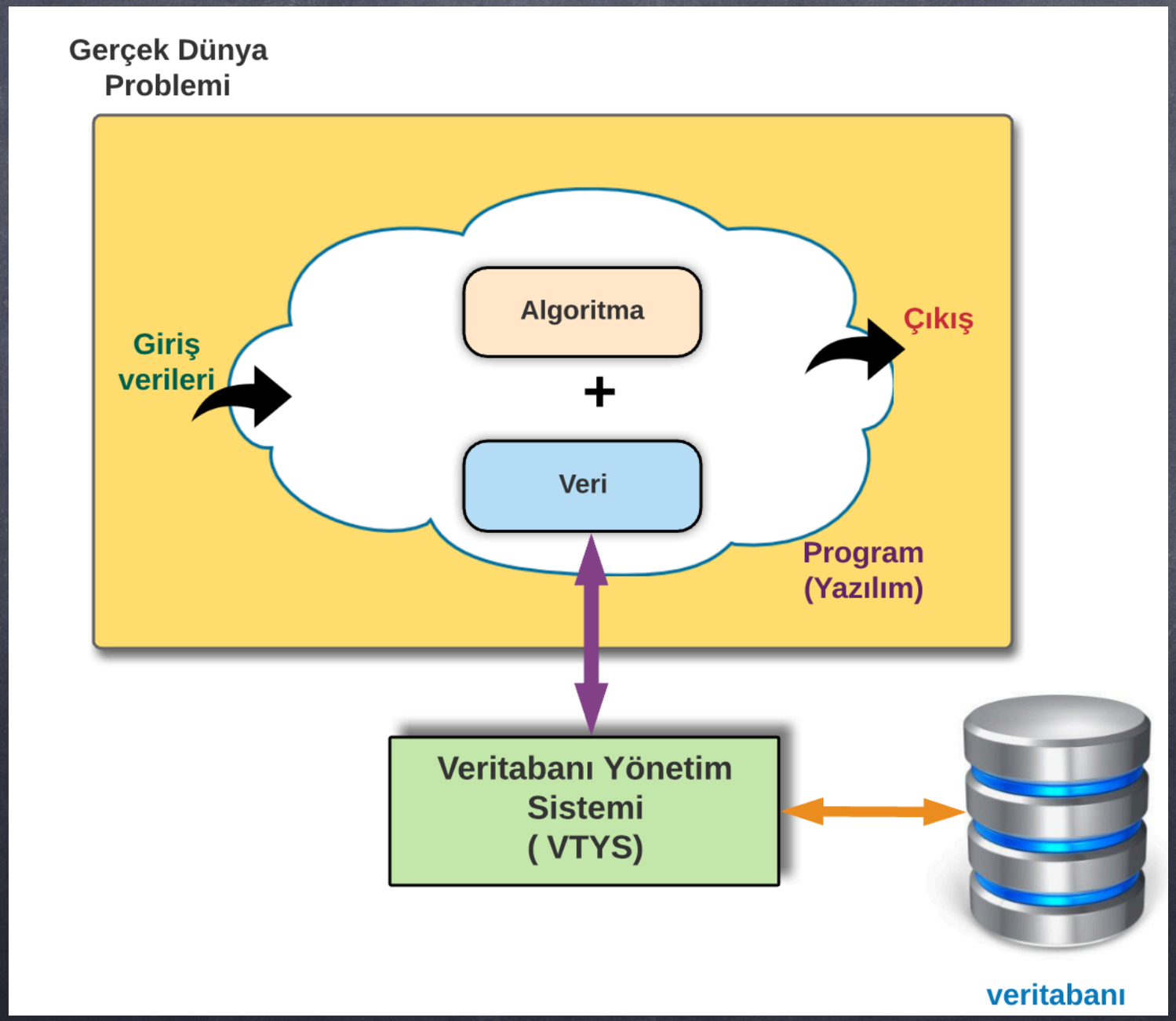
# Nesne Yönelimli Analiz ve Tasarım

## Yazılım Geliştirme Yaşam Döngüsü

# Büyük Resim


Gerçek Dünya
Problemi
Algoritma
+
Veri
Giriş
verileri
Çıkış
Program
(Yazılım)
Veritabanı Yönetim
Sistemi
( VTYS)
veritabanı

# İyi Yazılım?

iyi bir yazılım;

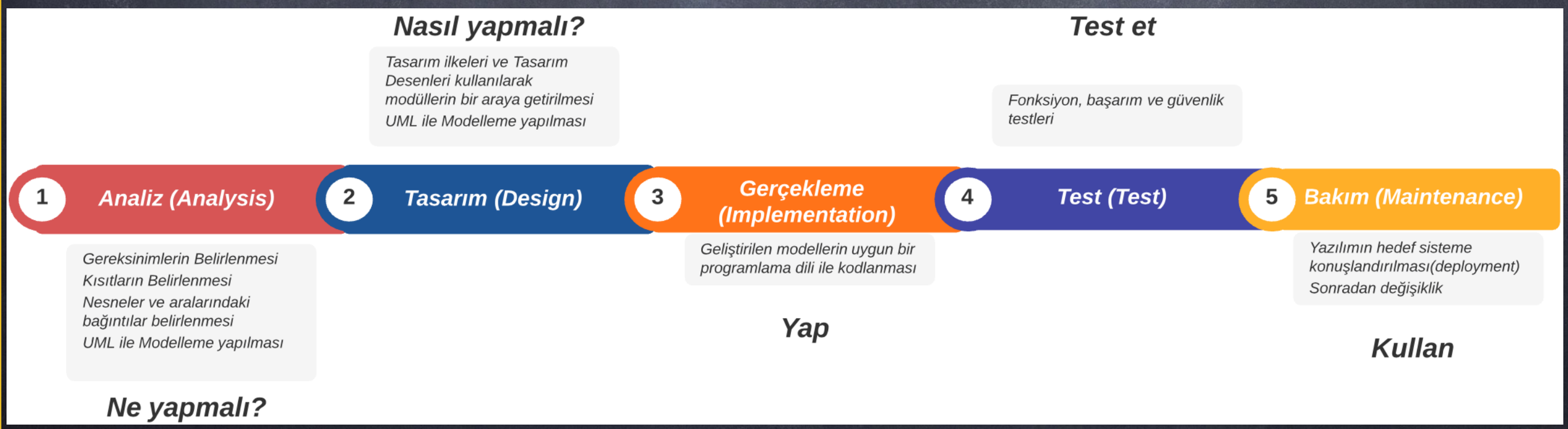
* son kullanıcı açısından bakıldığında,
  * beklendiği gibi çalışan, istenen fonksiyonları yerine getiren, hatalı sonuçlar vermeyen yazılım
* yazılım geliştiriciler açısından,
  * sonradan değişikliği kolaylaştıran
  * kod tekrar kullanımının fazla olduğu
  * yeni özellik kazandırmanın kolay olduğu
  * temiz kodlanmış, anlaşılabilir, karmaşıklığı azaltılmış yazılımlardır
  * bunu sağlamanın yolu ise tasarım ilkelerinin ve desenlerinin uygun kullanımına bağlıdır.

# Yazılım Geliştirme Yaşam Döngüsü

YGYD içerisinde 5 temel evre bulunmaktadır.

Günümüzde farklı yazılım geliştirme yaşam döngüsü modelleri (software processes) bulunmaktadır: waterfall, spiral, iterative/ incremental, agile, unified vb...

bu modellerin tamamı, bir şekilde bu evreleri içerir.



# 1. Analiz

- Geliştirilecek yazılımın gereksinimleri belirlenir.
- Bu adım proje yöneticisi, iş analisti ve uygulama programı yöneticisi nin sorumluluğundadır. Uç kullanıcılar, yöneticiler ve geliştirilecek yazılımla ilgili diğer paydaşlarla görüşülerek gereksinimler belirlenir (requirements elicitation). Aşağıdaki sorulara yanıtlar aranır
  - Sistem ne yapmalı? ne yapması bekleniyor?
  - sistemi kimler, ne yapmak için kullanacak?
  - giriş verileri neler?
  - çıkış verileri neler?
  - ne tür veriler ele alınacak/işlenecek (telefonNo, TC kimlikNo ... Bu verilere bakılarak veritabanı tasarımı için ihtiyaç duyulan iş kuralları da oluşturulur )
  - takip edilmesi gereken standartlar, kurallar, yönetmelikler, güvenlik gereksinimleri v.s. neler?

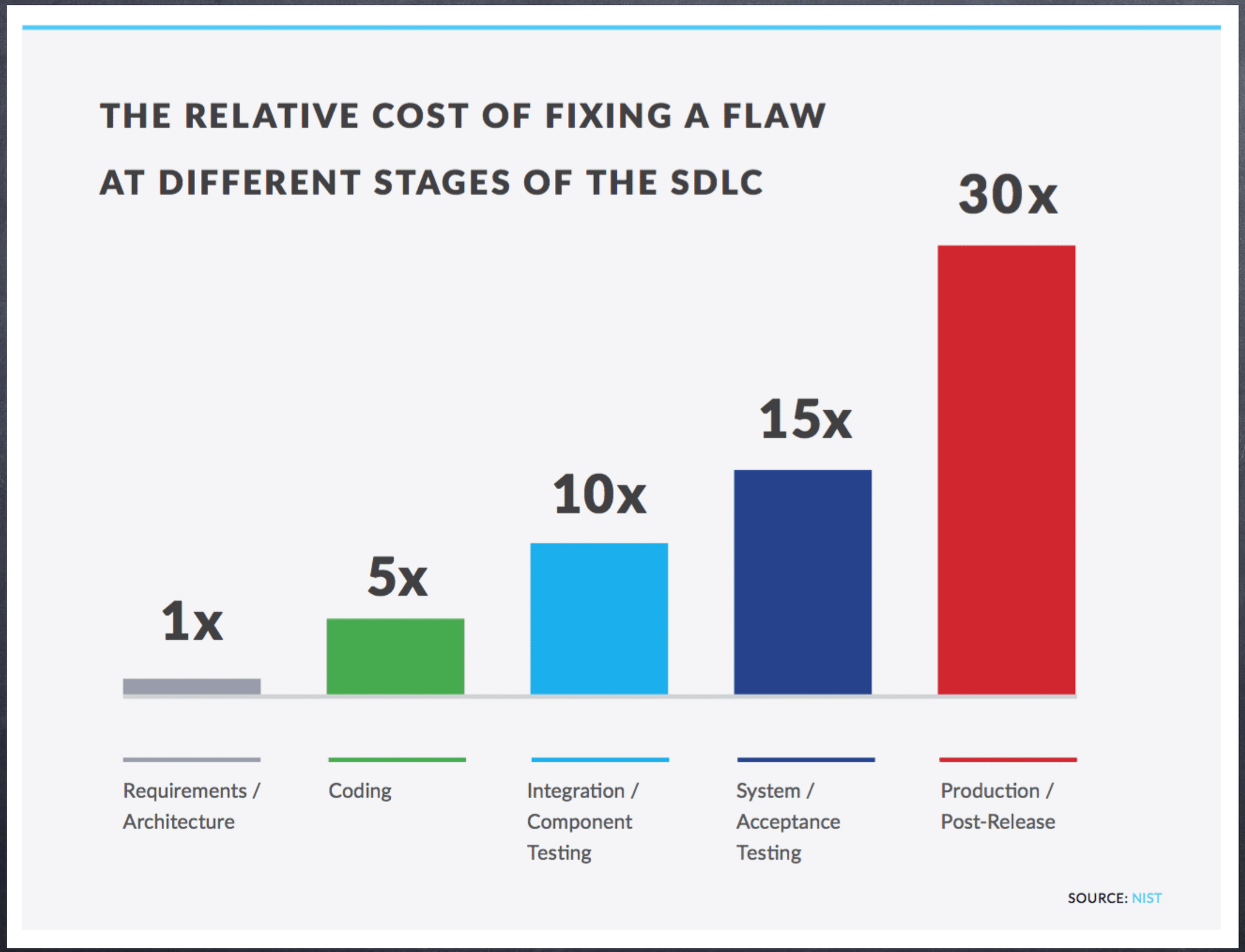
- Geliştirilecek sistemle ilgili kısıtların belirlenmesi
- Uygulama Program yöneticisi nin sorumluluğundadır
  - Uygulama hangi işletim sisteminde çalışacak? işletim sistemi kısıtları neler?
  - bellek kısıtları, işlem gücü kısıtları, ağ kısıtları vb. operasyonel ya da mimariyle ilgili kısıtlar neler?

Kaynak kullanımı, hız, güvenlik gibi ihtiyaçlara uygun dillerin/teknolojilerin/algoritmaların/kütüphanelerin seçilmesi....

# 2. Tasarım

- Kısıtlar ihlal edilmeden yazılım gereksinimlerinin nasıl karşılanacağı belirlenir.
- Tasarım aşamasında oluşturulacak modeller için yaygın olarak UML (Unified Modeling Language) kullanılır.
- Yazılıma bakış açıları farklı aktörler arasındaki haberleşmeyi sağlar
- Kodlamayı kolaylaştırır. Tasarımdaki bileşenler yazılım modüllerine dönüştürülür. Bu modüllerin nasıl etkileşeceği de bu adımda belirlenir.
- Tasarım ekibi tarafından gereksinimler ve mimari kısıtlar göz önüne alınarak aşağıdakiler belirlenmelidir:
  - Kullanılacak diller ve programlama yöntemleri
  - Takip edilecek kodlama standartları (Kodlama sitili, isimlendirme kuralları, kullanılmaması gereken modüller vb.)
  - Kullanılacak modüller, kütüphaneler
  - Veri yapıları, veritabanı yönetim sistemi ...
- Oluşturulan modeller gerçeklenmeden önce dikkatli olarak incelenmelidir. Böylece, olası hataların düzeltilme maliyeti düşürülmüş olur.

YGYD evreleri ilerledikçe, değişiklik yapma maliyeti artar.




https://www.opswat.com/blog/secure-sdlc-at-opswat

# 3. Gerçekleme

- Önceki aşamalarda geliştirilen modeller kodlanarak yazılım uygulamasına dönüştürülür.
- Yazılım geliştirme ekibi sorumluluğundaki bu adımda:
  - Kodlama standartlarına uyulmalı
  - Tasarım dökümanı takip edilmeli
  - kod incelemesi (code review) yapılarak kod kalitesi ve güvenlik artırılmalı.
  - Güvenlik gereksinimleri dikkate alınmalı (giriş filtreleme, çıkış kodlama vb.)

# 4. Test

- Geliştirilen uygulama; işlevsellik, başarım ve güvenlikle ilgili gereksinimler dikkate alınarak test edilir.
- Böylece uygulamanın beklendiği gibi çalışması sağlanır.
- Bu aşamada birim testi (unit testing), tümleştirme testi (integration testing), sistem testi (system testing), kabul testi (acceptance testing) yapılır.
- Güvenlik açısından sızma testleri yapılır.
- Bu aşamadan sorumlu Test ekibi aşağıdakileri kontrol etmelidir:
  - sistem kararsız çalışmamalı
  - testler kodların tamamını kapsamalı (code coverage %100)
  - sistem tüm gereksinimleri karşılamalı

# 5. Bakım&Konuşlandırma (Maintenance&Deployment)

- Sürüm Yöneticisi (Release Managers) yazılımı kullanıma hazırlar
- uygulamanın kurulum işlemi
- gerekli bileşenlerin kurulumu
- Güncellemeler

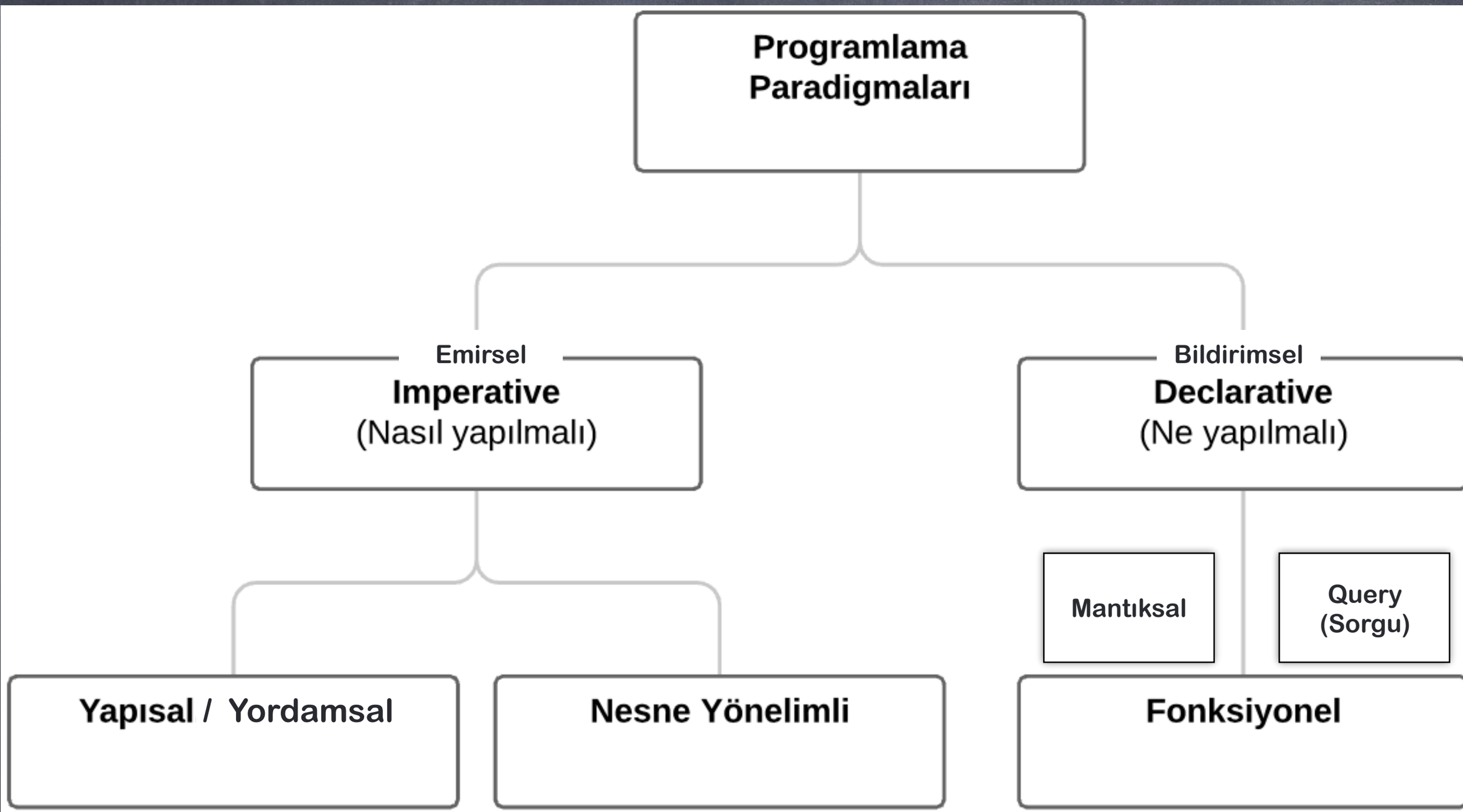
# 5. Bakım&Konuşlandırma (Maintenance&Deployment)

- Uygulama kullanılmaya başladıktan sonra meydana gelen değişiklik gereksinimleri bu aşamada (maintenance) ele alınır.
- Değişiklik nedenleri (Lientz and Swanson):
  - Adaptive - yazılımın çalıştığı ortamın değişmesinden kaynaklanan (DBMS, OS)
  - Perfective - yeni/değişen kullanıcı isteklerinden (sistem fonksiyonlarının iyileştirilmesi) kaynaklanan
  - Corrective - kullanıcılar tarafından tespit edilen hatalardan kaynaklanan
  - Preventive - gelecekte meydana gelebilecek sorunların önlenmesini sağlamak için
- 75% adaptive ve perfective, ~%21 hataların düzeltilmesi (corrective).

# Nesne Yönelimli Analiz ve Tasarım

Programlama Paradigmaları

# Programlama Paradigmaları

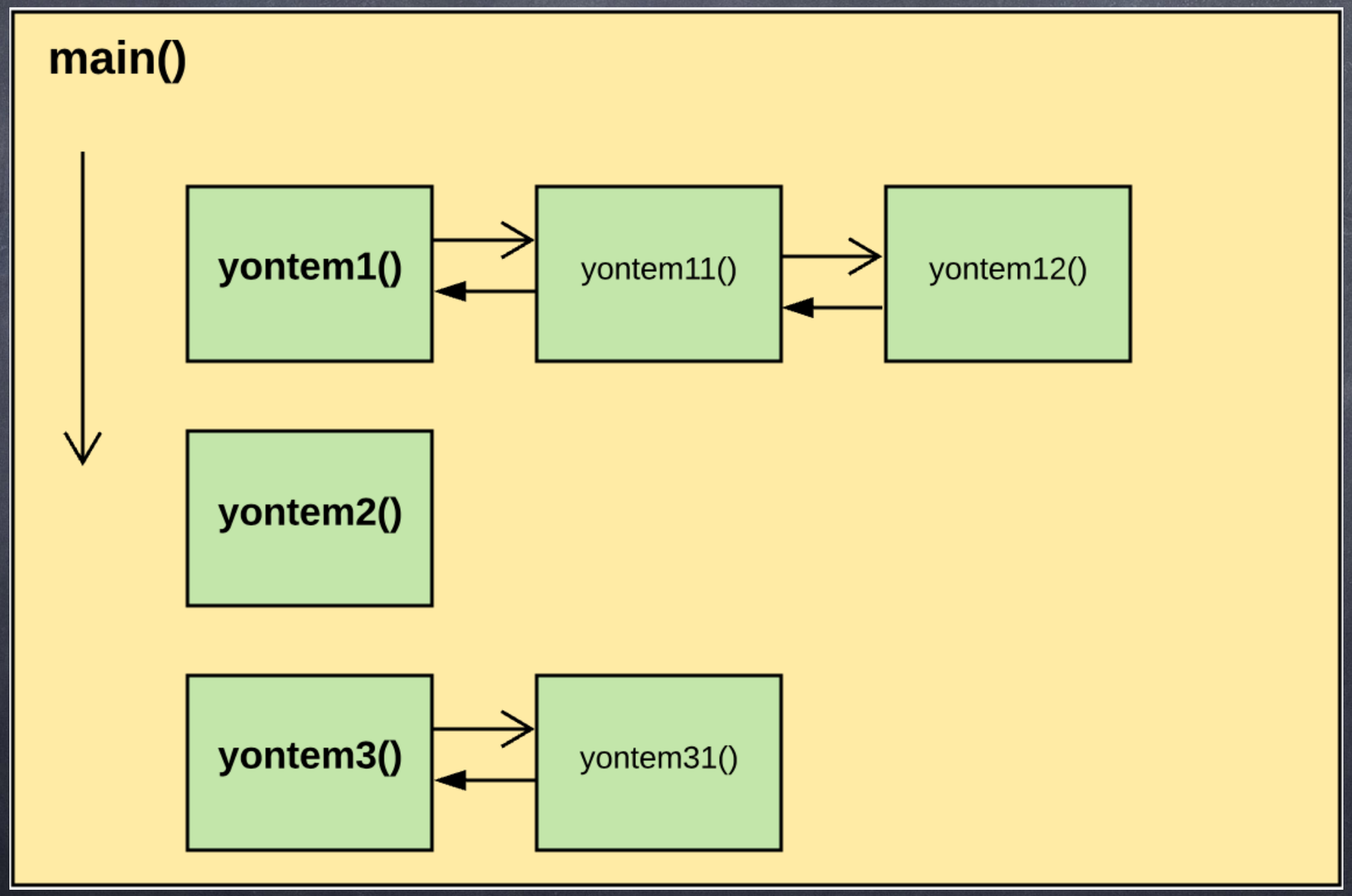


# Yapısal (Yordamsal-Prosedürel) Programlama Paradigması

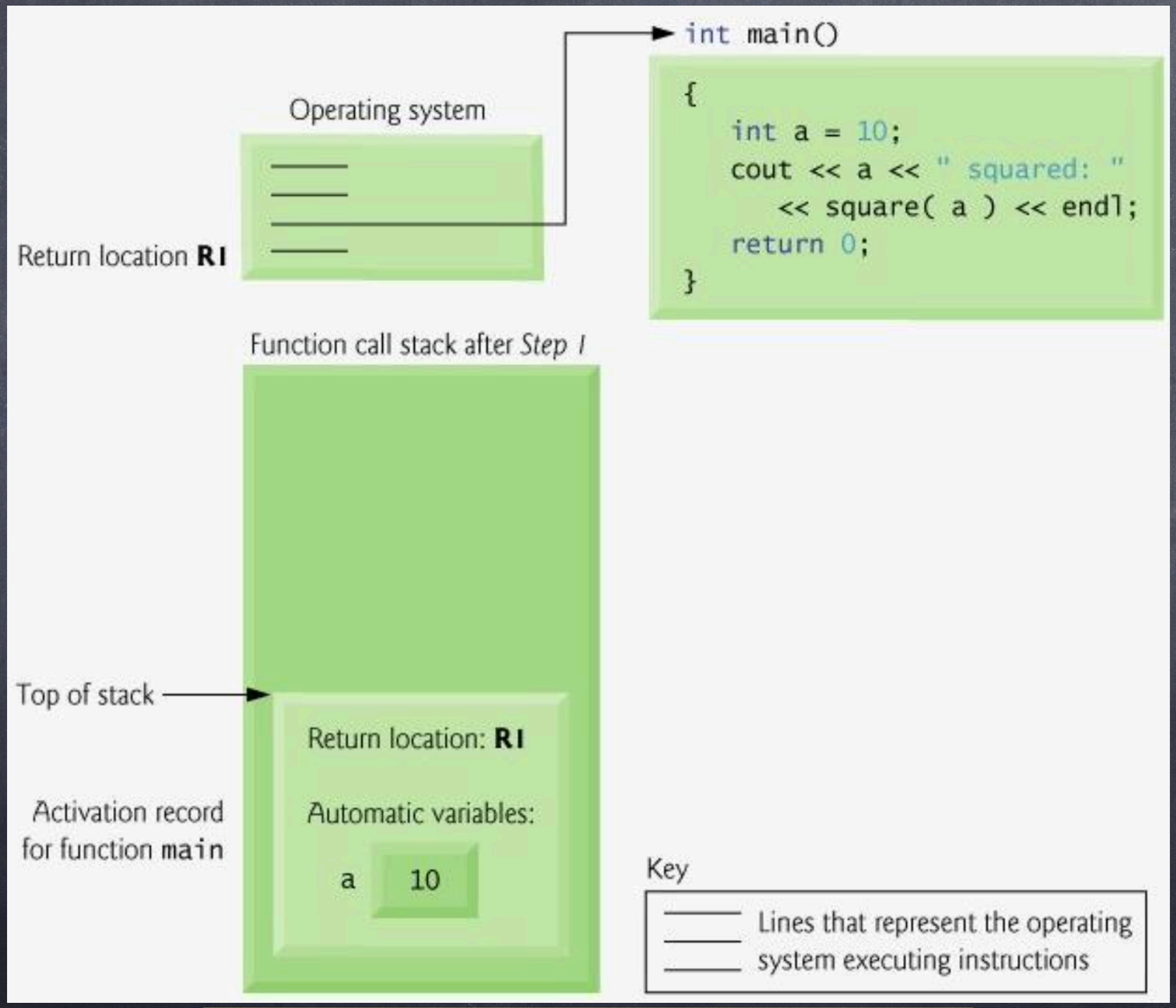
* Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of subroutines, block structures and for and while loops in contrast to using simple tests and jumps such as the goto statement which could lead to "spaghetti code" which is difficult both to follow and to maintain.

https://en.wikipedia.org/wiki/Structured_programming

# Yapısal (Yordamsal-Prosedürel) Programlama Paradigması

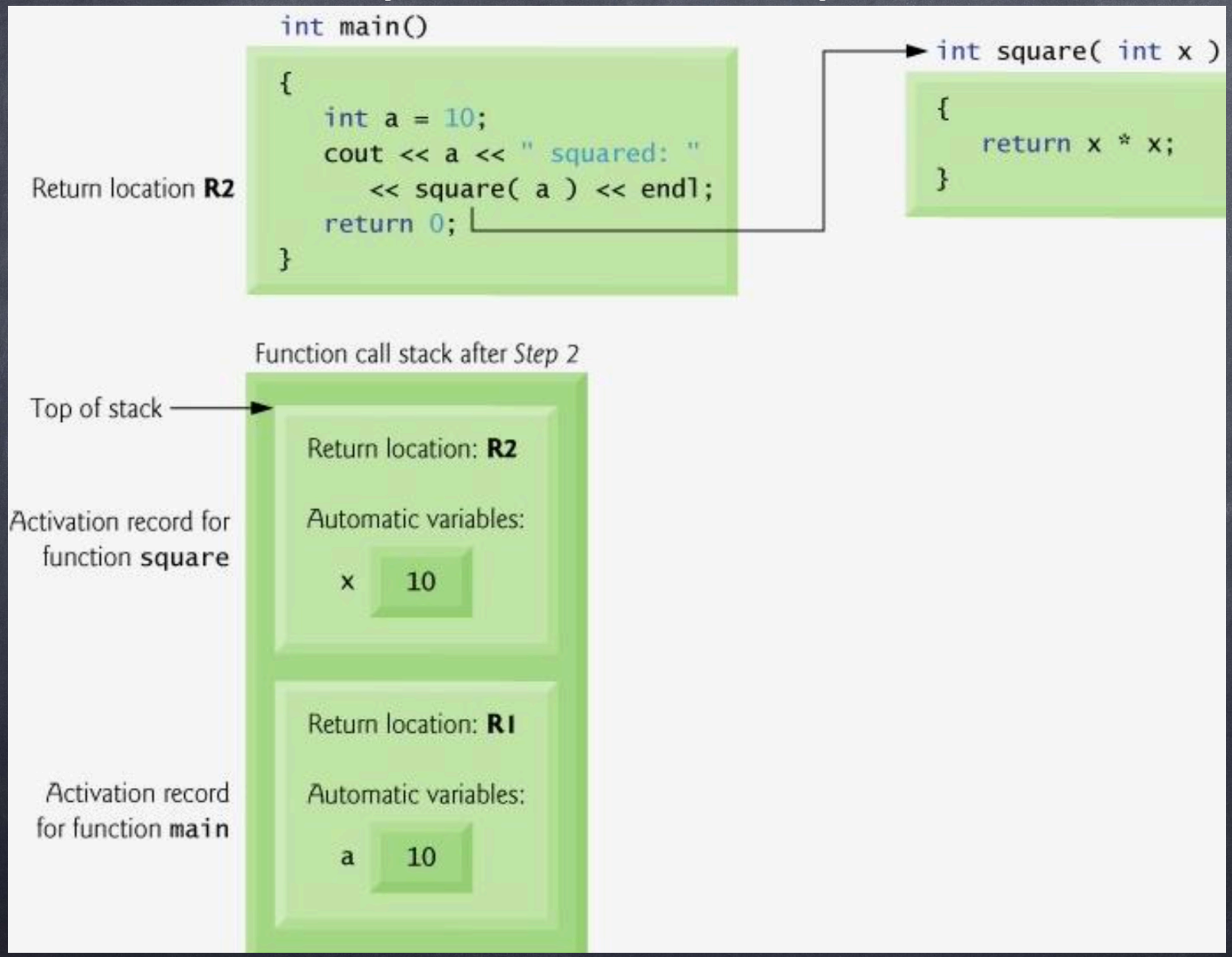


# Yapısal (Yordamsal-Prosedürel) Programlama Paradigması




Deitel, C++ How To Program, Prentice Hall

# Yapısal (Yordamsal-Prosedürel) Programlama Paradigması

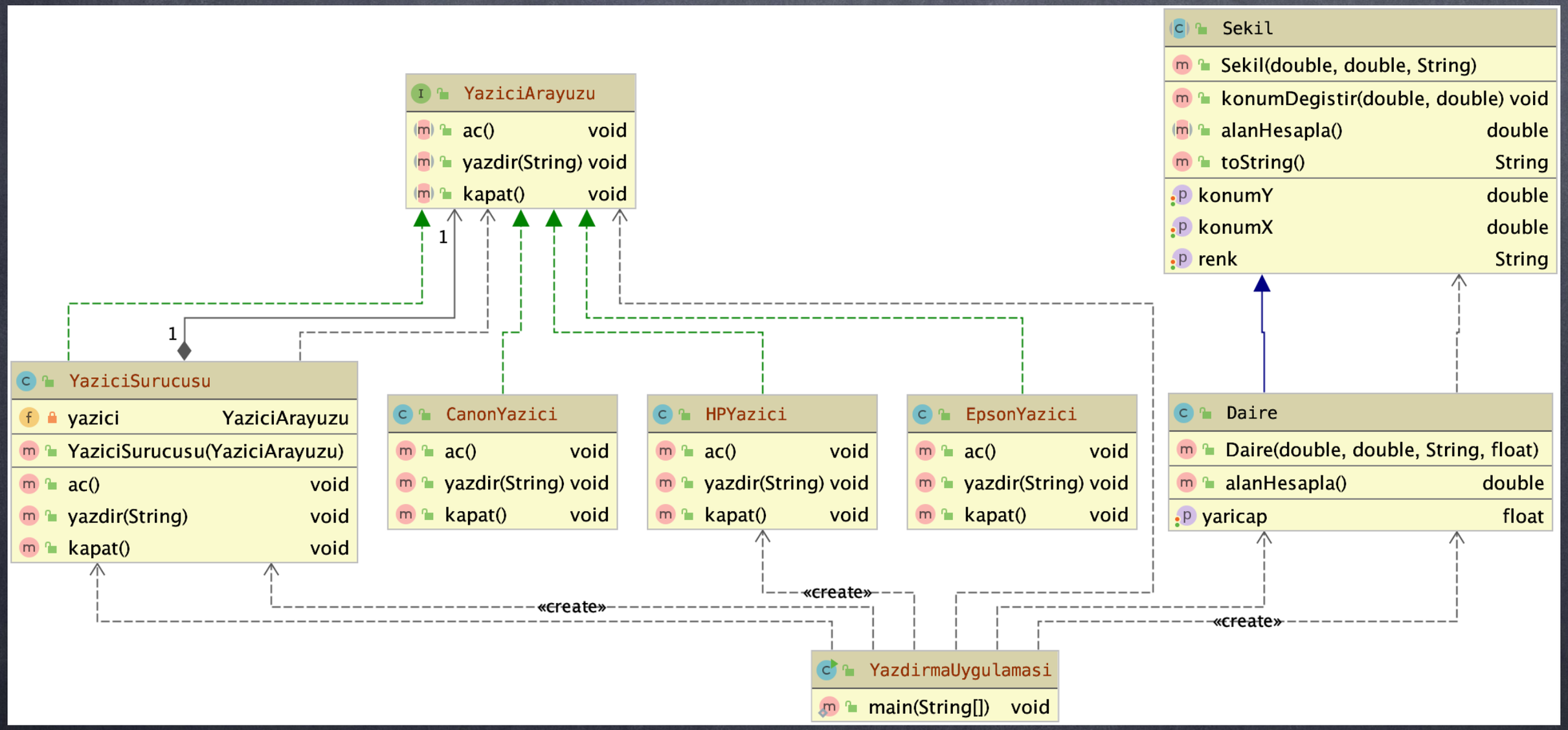



Deitel, C++ How To Program, Prentice Hall

# Nesne Yönelimli Yazılım Geliştirme Paradigması

* Yapısal teknikte doğrudan probleme odaklanılır ve problemin çözümüne ilişkin yöntemler geliştirilir (control-centric).
* Nesne yönelimli programlama tekniğinde ise temel bileşen nesnedir (data-centric) ve programlar nesnelerin birlikte çalışmasından meydana gelir.
* Nesne hem veriyi hem de bu veriyi işleyen yöntemleri içerir. Yazılım geliştiriciler dikkatlerini nesneleri oluşturan sınıfları geliştirmeye yoğunlaştırır. Böylece program zekası veri merkezli olarak tüm modüllere dağıtılır.
* Yapısal teknikte bir fonksiyon herhangi bir görevi yerine getirmek için veriye ihtiyaç duyarsa, gerekli veri parametre olarak gönderilir. NYP de ise yerine getirilmesi gereken görev nesne tarafından icra edilir ve fonksiyonlar verilere parametre gönderimi yapılmaksızın erişebilirler.

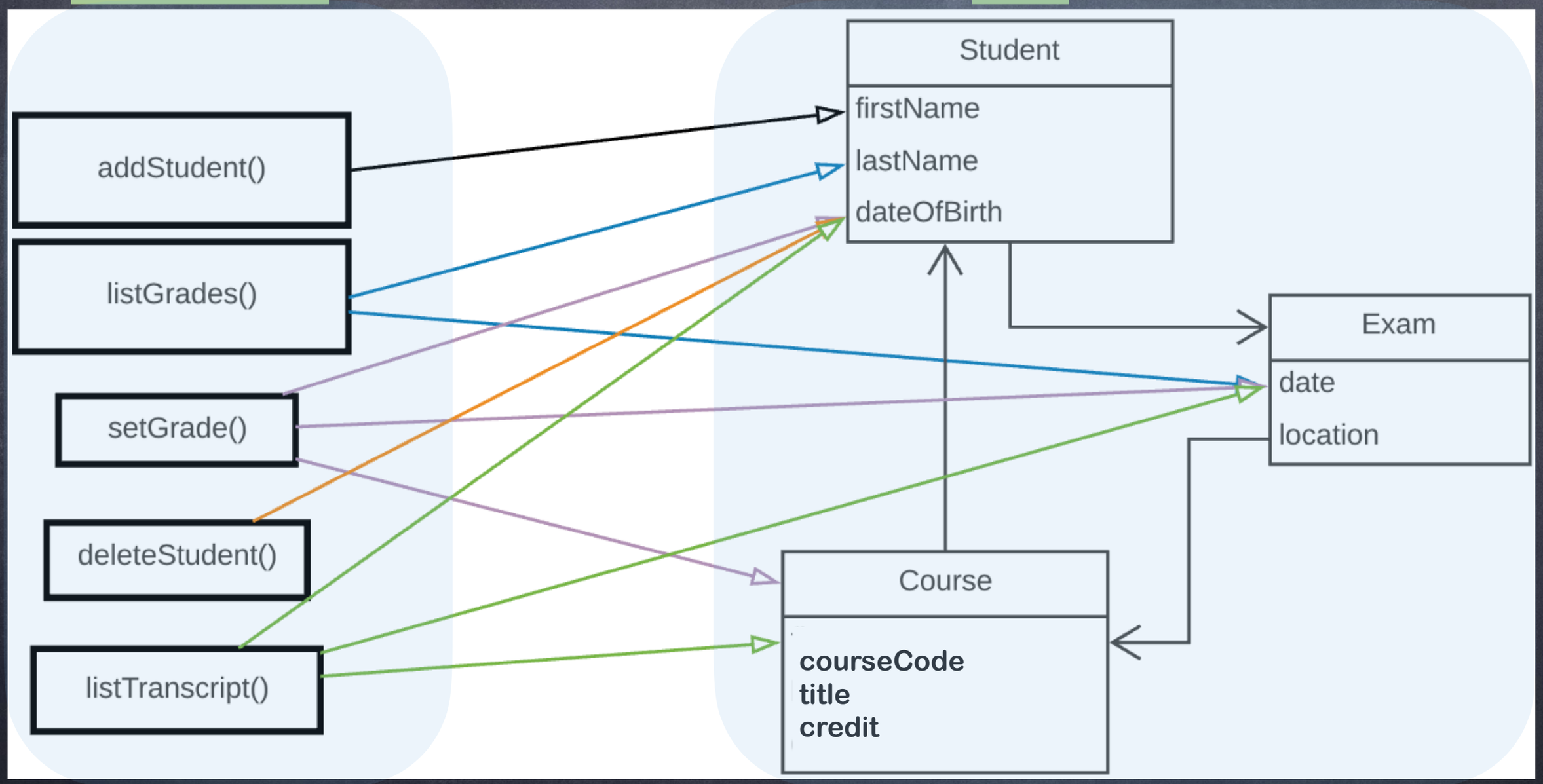
# Nesne Yönelimli Yazılım Geliştirme Paradigması



# Nesne Yönelimli Yazılım Geliştirme Paradigması

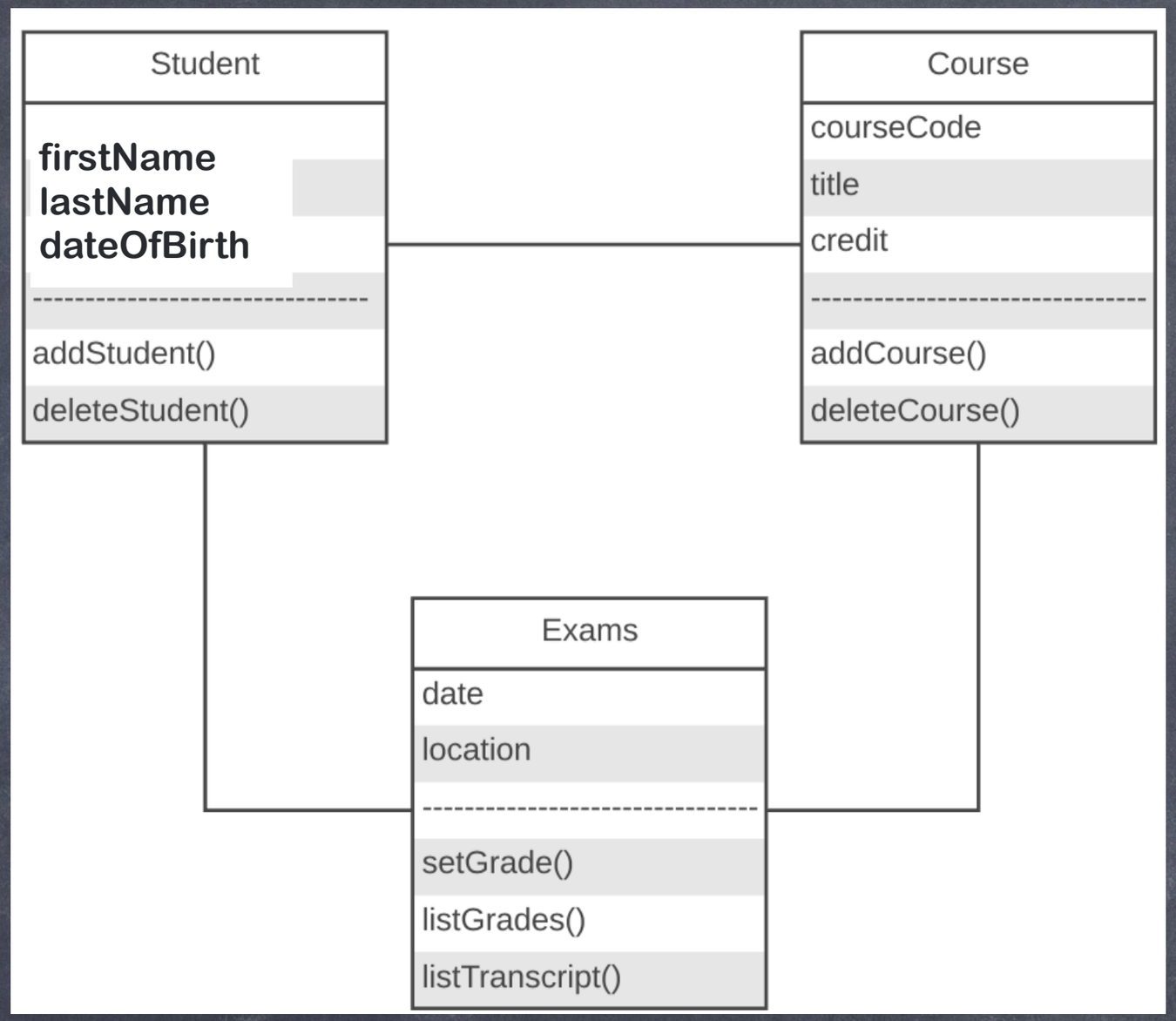
* Sistem büyüdükçe ilişkiler/bağımlılık daha da karmaşıklaşır.
* Sonradan değişiklik zorlaşır.
* Program içerisinde değişiklik (ekleme, çıkarma, düzeltme) yapmak zorlaşır ve beklenmeyen etkilere neden olabilir...
* Örneğin(bir sonraki yansıda yer alan ÖBS uygulaması);
  * Ogrenciler tablosunda ogrencinin dogum tarihi iki haneli
  * Bu alanı 4 haneli yapmak istiyoruz...
  * Bu veriyi kullanan yöntemlerinde güncellenmesi gerekecektir.
  * Hangi yöntemler hangi veriye erişiyor ?

# Nesne Yönelimli Yazılım Geliştirme Paradigması




Hangi yöntemler hangi veriye erişiyor ?

# Nesne Yönelimli Yazılım Geliştirme Paradigması




**High Coherence**

Modüller tek ve özel bir işi, mükemmel bir şekilde yapmalı. Alt sistemler içerisindeki sınıflar benzer işi yapmalı ve ilgili olmalı. "Cohesion" Bunun ölçüsüdür .

**Low Coupling**

Modüller arasındaki bağlantı zayıf olmalıdır. Böylece yapılacak değişikliğin bağlantılı modülleri etkilemesi önlenir.

# Nesne Yönelimli Yazılım Geliştirme Paradigmasının Temel Özellikleri

* Encapsulation (Information Hiding)
* Inheritance
* Polymorphism
* Abstraction
* Modular Programming
* Code Reuse
* Maintenance
* Design Principles (SOLID)
* Design Patterns

Uygulamalar
https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders2

# Fonksiyonel Programlama

* (Pure) saf fonksiyonların birleşiminden oluşur
    * aynı giriş için aynı çıkış üretilir (no side effect)
    * paylaşılan/global değişken yoktur
    * parametre olarak gelen nesnenin durumları değiştirilemez.
* Tekrarlı yapılar (döngü) yoktur.
* Durum değişimine izin verilmediği için paralel programlar yazılabilir (akış işlemede kullanılır).

Uygulamalar
https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders4/VeriToplulukLariUzerindeIslemler

```
List<Kitap> kitaplarV = new Vector<Kitap>();
kitaplarV.add(new Kitap("Veritabanı Yönetim Sistemleri ",100.00));
kitaplarV.add(new Kitap("Nesne Yönelimli Programlamaya Giriş ",125.00));
kitaplarV.add(2,new Kitap("Bilgisayar Ağları",150.00));
kitaplarV.add(3,new Kitap("Veri Yapıları",160.00));

for (int i=0; i<kitaplarV.size();i++)
    System.out.println(kitaplarV.get(i).getAdi());

// *************Lambda İfadeler*************
kitaplarV.forEach(kitap -> System.out.println(kitap.getAdi()));

double sum=0;
for (int i=0; i<kitaplarV.size();i++){
    sum+=kitaplarV.get(i).getBirimFiyati();
}
System.out.println(sum);

// *************Lambda İfadeler*************
sum= kitaplarV.stream().mapToDouble(item->item.getBirimFiyati()).sum();
System.out.println(sum);

List<Kitap> filtrelenmisKitaplar = kitaplarV
        .stream()
        .filter(item -> item.getAdi().startsWith("V"))
        .collect(Collectors.toList());
System.out.println(filtrelenmisKitaplar);
```

# Nesne Yönelimli Analiz ve Tasarım

Analiz Aşaması
Gereksinimlerin Belirlenmesi
UML (Unified Modelling Language) Birleştirilmiş Modelleme Dili
Kullanım Durum (Use Case) Diyagramı
Sıralama Şeması
Etkinlik Şeması
Durum Makinesi Diyagramı

# Nesne Yönelimli Analiz

* Analiz aşamasının ilk adımı gereksinimlerin belirlenmesidir.
* Gereksinimler, sistemin düzgün çalışması için yapması beklenenleri ifade eder
* Gereksinimler ilk olarak kullanıcılarla(paydaşlarla) görüşülerek (gerekli ise standart, yasa, yönetmelik vb. dokümanlar takip edilerek) oluşturulur, ve neler istendiği belirlenir.
* Daha sonra kullanım durumları (use case) belirlenerek gereksinimler ayrıntılandırılır.
* Sistemin tüm istenenlerle birlikte, beklenmeyen ya da istenmeyen durumları da göz önüne alınır.
* Kullanım durumları analiz edilerek nesneler, nesne üyeleri ve nesneler arasındaki bağıntılar belirlenir.

The Unified Modeling Language

UML

Birleştirilmiş Modelleme Dili

# Modelleme

Gerçeğin soyutunu (basit gösterimini) oluşturma işlemidir.

Soyutlama sayesinde gereksiz ayrıntılar gizlenir ve ilgilenilecek detaylara odaklanılır. Böylece karmaşıklıktan kurtulunur.

Ayrıntıların gerekli ya da gereksiz olduğu modelin amacına bağlıdır.

Günümüz yazılımlarının kod karmaşıklığı her geçen gün artmaktadır.

Bu nedenle, yazılımların farklı yönlerini, farklı ayrıntılarla ifade etmek üzere model oluşturulur.

# Modelleme

Model sayesinde;

* fikirlerin tartışılması/paylaşılması, yazılım aktörlerinin (son kullanıcı, yazılım geliştirici, tasarımcı, iş analisti vb.) iletişimi kolaylaşır.
* yazılımın gerçekleneren (kodlanması) ürün haline dönüştürülmesi daha kolay olur.

# UML (Unified Modelling Language)
# Birleştirilmiş Modelleme Dili

Yazılım sistemlerinin temel bileşenlerini ifade etmek ve farklı yönlerini, farklı ayrıntılarla göstermek için kullanılan grafiksel modelleme dilidir.

UML is a common language for business analysts, software architects and developers used to describe, specify, design, and document existing or new business processes, structure and behavior of artifacts of software systems. (https://www.uml-diagrams.org/)

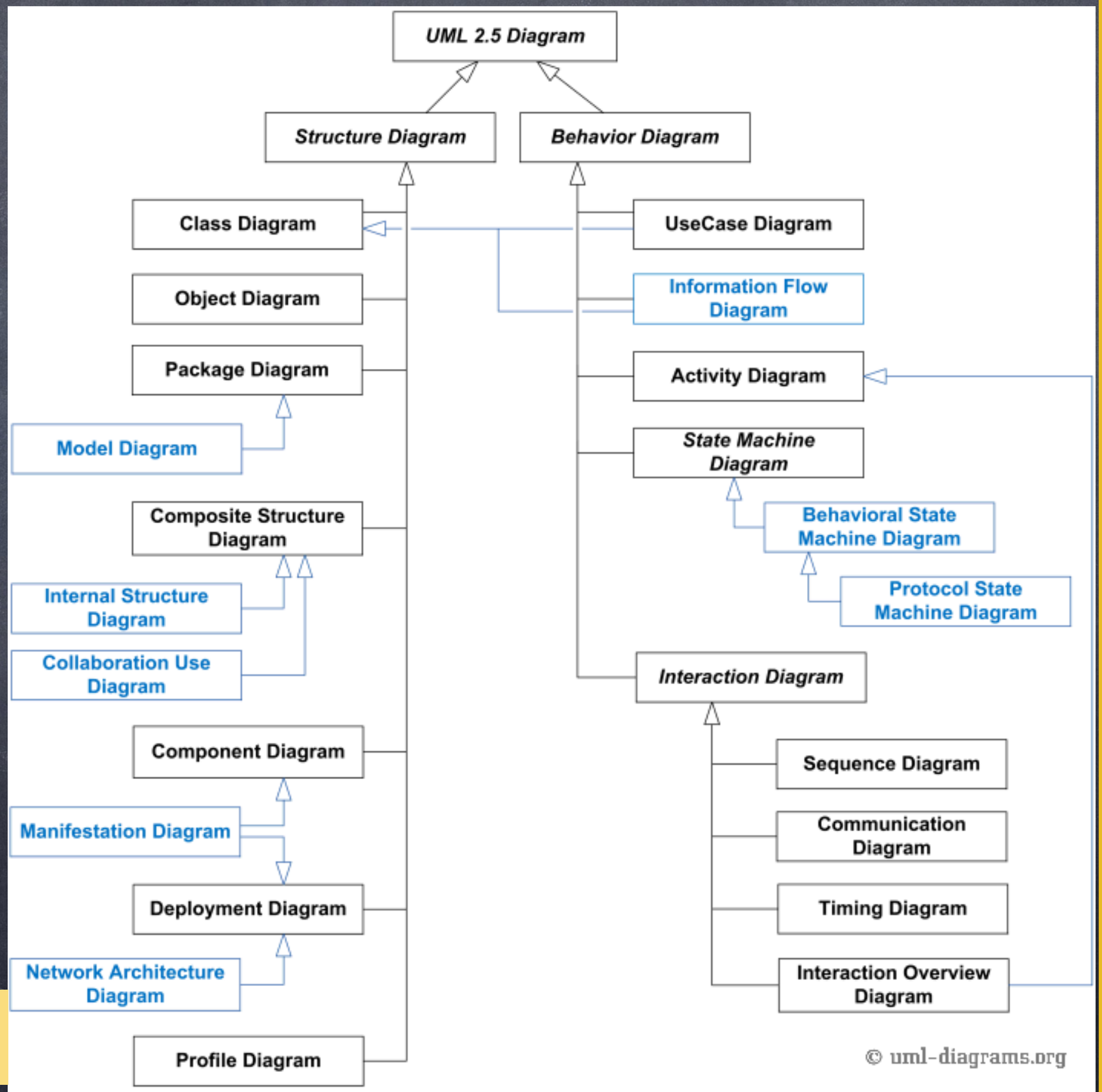
# UML

* Yapı diyagramları (Structure Diagrams): sistemin(sistem bileşenlerinin) statik yapısını gösterir

* Davranış Diyagramları (Behavior Diagrams): Sistemdeki nesnelerin davranışlarını (zamana göre değişimlerini) gösterir.

Not: mavi ile belirtilen gösterim şekilleri UML 2.5 standardı değildir.

https://www.uml-diagrams.org/uml-25-diagrams.html

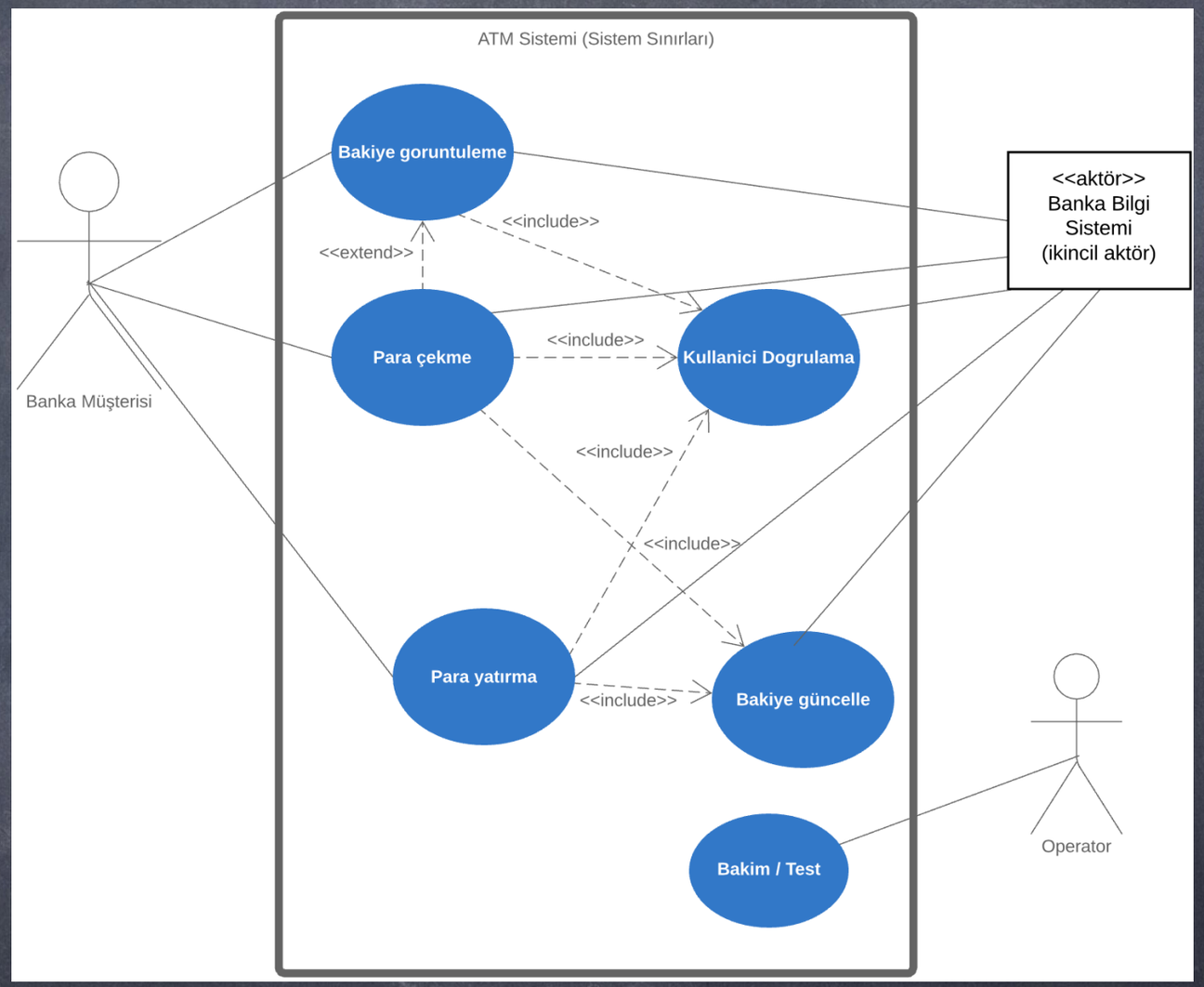


# Örnek Uygulama: ATM İçin Yazılım Geliştirme Gereksinimlerin Belirlenmesi

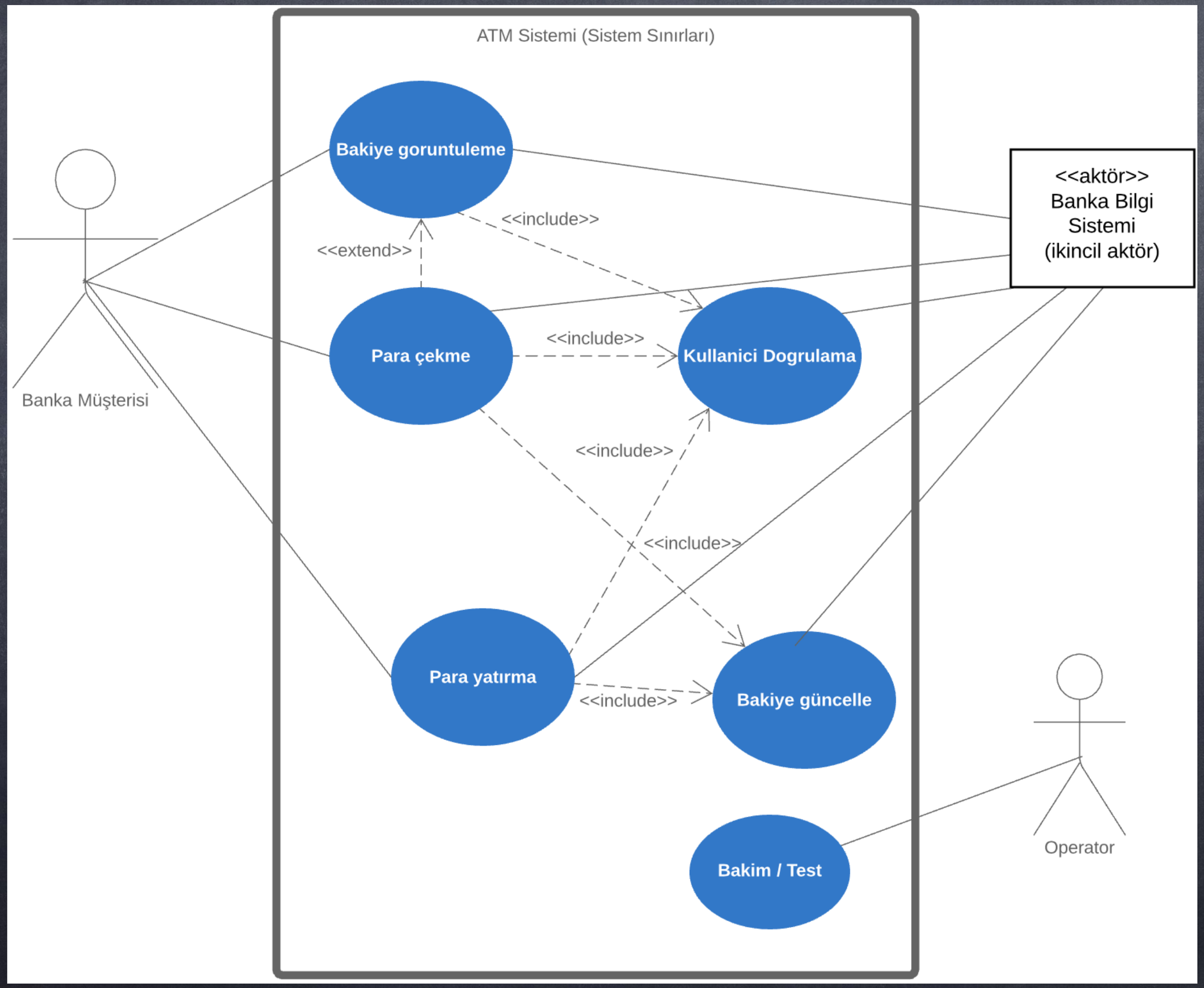
* ATM aşağıdaki donanım bileşenlerinden oluşur:
  * Tuş takımı; veri girişinden sorumludur.
  * Ekran; mesajların görüntülenmesinden sorumludur.
  * Para bölmesi; para alma ve para verme işleminden sorumludur.
  * Kart bölmesi; banka kartını alınıp doğrulanmasından sorumludur.
* Kullanıcılar banka kartı ile para çekme, para yatırma, bakiye görüntüleme işlemlerini yapabilmelidirler.
* Banka görevlileri ATM sisteminin bakımını (para doldurma, sıkışan/yutulan kartları alma v.b. ) yapmalıdır.
* Tüm işlemlerde güvenlik gereksinimleri göz önünde bulundurulmalıdır (yetki kontrolü).
* Sistem, Banka Bilgi Sistemiyle (kredi kartı servis sağlayıcıları ile) çevrimiçi çalışmalıdır.

# UML - Kullanım Durumu (Use Case) Diyagramları

* Davranışsal gösterim şekillerinden biridir.
* Kullanım durumu, sistem ile kullanıcıların (aktör/rol) etkileşimini tanımlar.
* Her kullanım durumunun tek bir hedefi olmalıdır. Bu hedefe giden farklı yollar olabilir.
* Kullanım durumları bir araya gelerek (kullanım durumu senaryosu/modeli), aktör sistem ile etkileşime girdiğinde neler olduğunu anlatır.
* Kullanım durumları "sistem aktör açısından ne yapar" ile ilgilenir. Nasıl yapar ile ilgilenmez.
* Aktör belirli bir görev için olayları başlatan nesnedir. (Banka müşterisi, operator, banka v.s.)

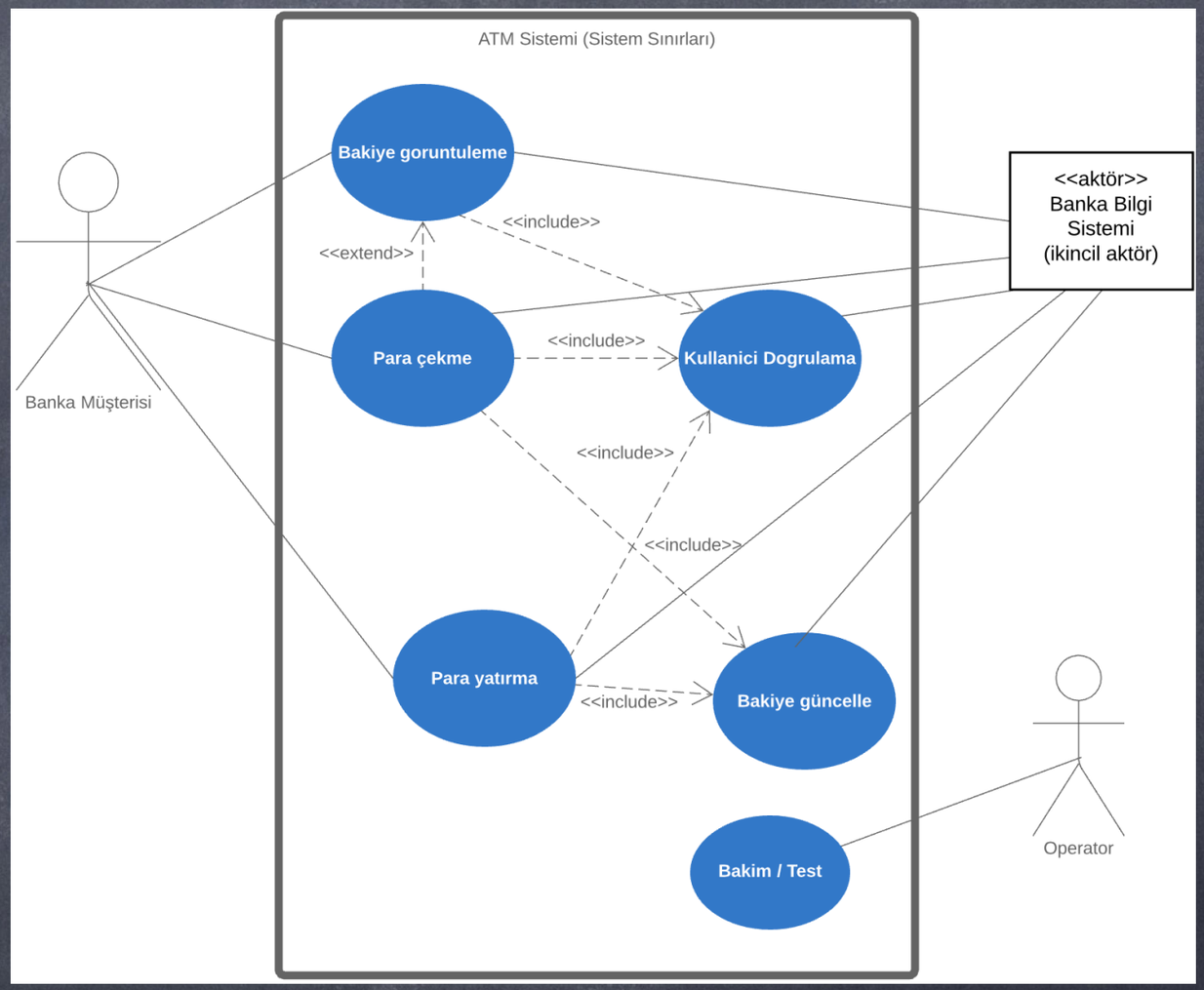


# UML - Kullanım Durumu (Use Case) Diyagramları



# Aktörler

* Aktör belirli bir görev için olayları başlatan nesnedir. (yönetici, öğrenci, hasta, doktor, müşteri, operator, personel, banka, yolcu, diğer sistemler v.s.)
* Sistem ile etkileşime giren harici varlıklardır
  * Kullanıcılar
  * Diğer sistemler / alt sistemler
  * Fiziksel ortam (Hava, su vb.)
* İkincil Aktör, Kullanım durumunun tamamlanması için etkileşime girilen aktördür.
* Kullanım durumları birincil aktörler tarafından başlatılır. İhtiyaç duyulduğunda ikincil aktöre istek gönderilir.

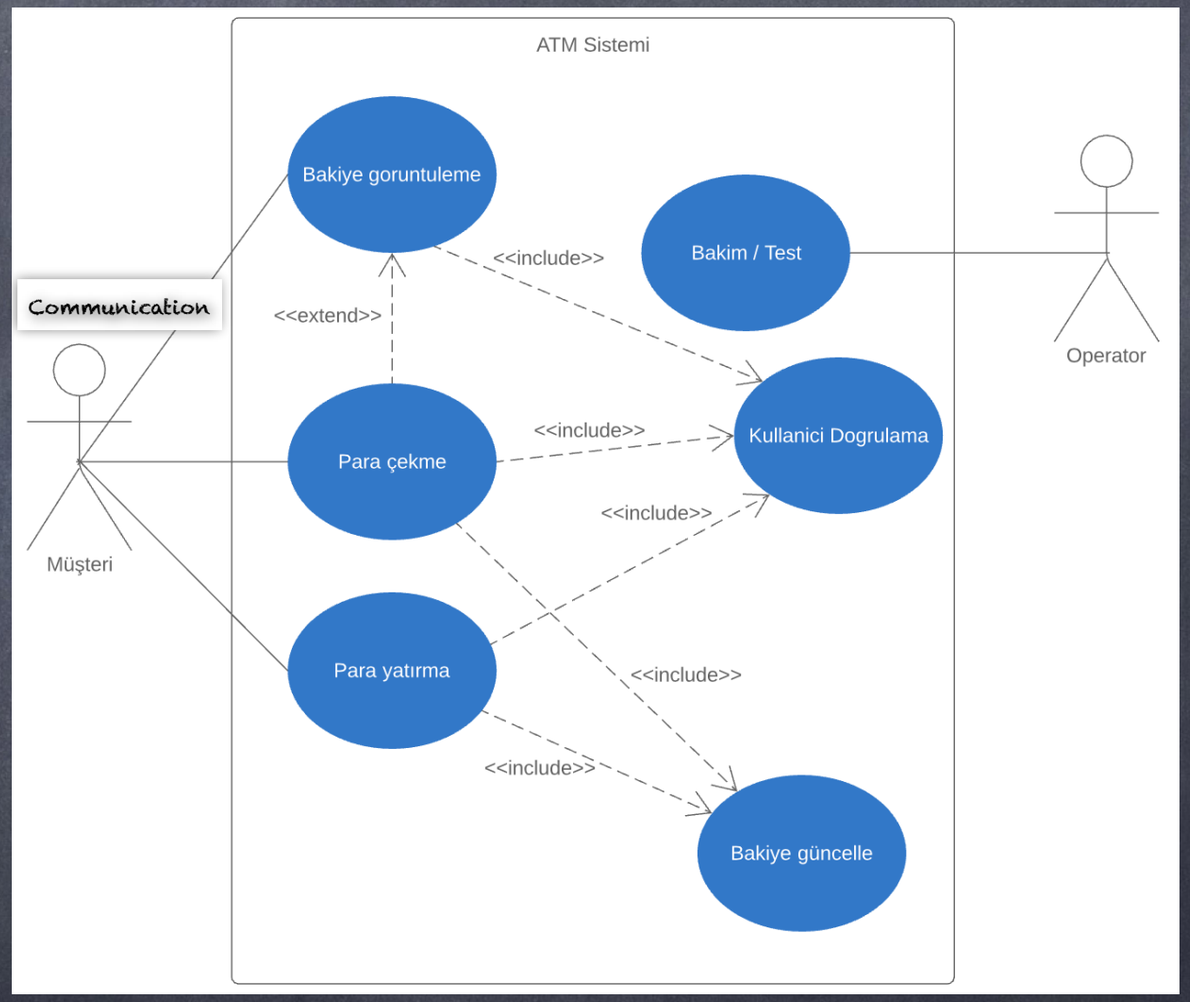


# Kullanım Durumu (Use Case) Ayrıntıları

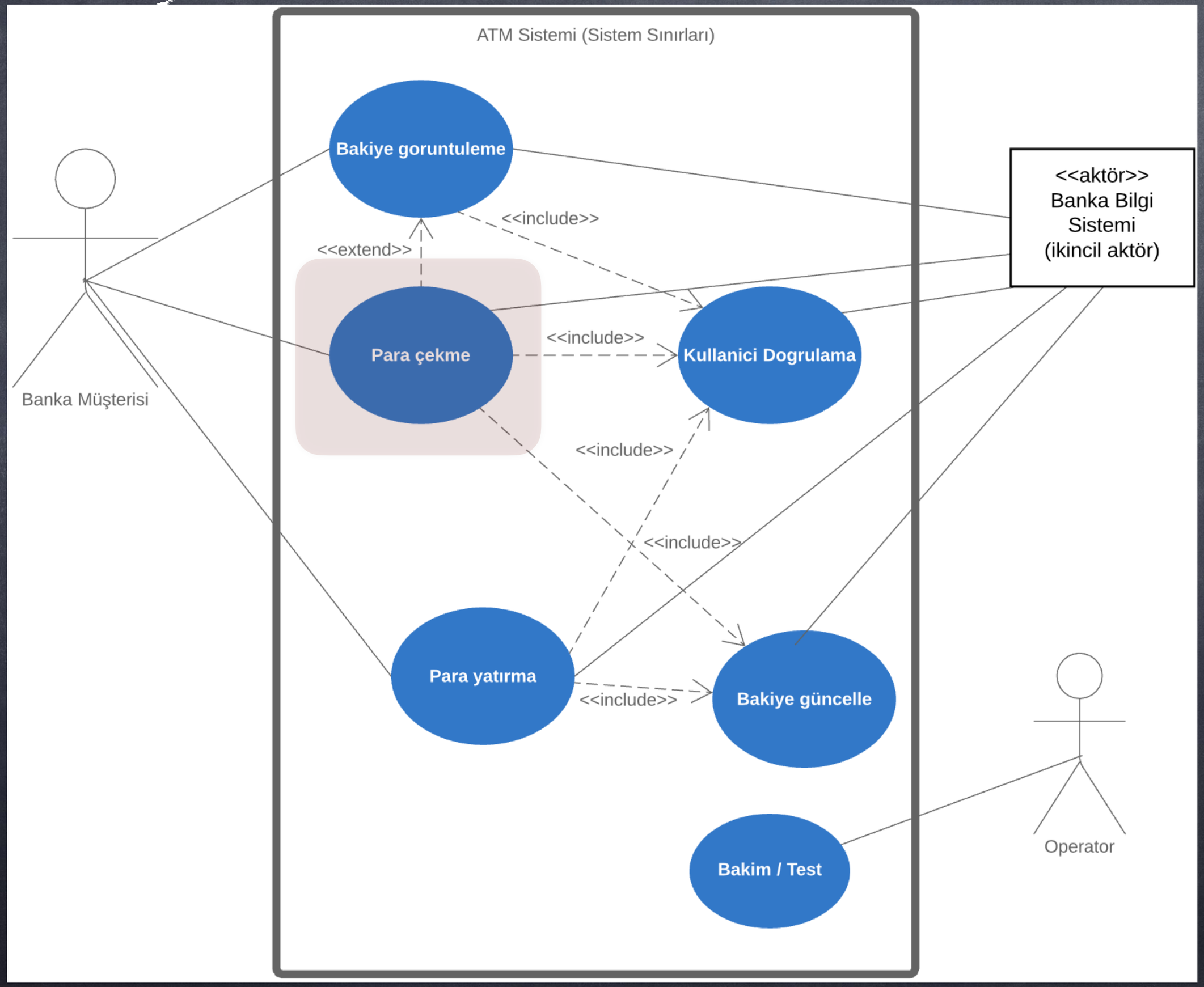
* Her kullanım durumunun tek bir hedefi olmalıdır. Bu hedefe giden farklı yollar olabilir.
* Kullanım durumunun ayrıntıları için, gereksinim listesi kullanılarak, metinsel ifadeler oluşturulur. Aktör ile sistem arasındaki olay akışı, istisnalar/ beklenmeyen durumlar da göz önüne alınarak anlatılır.
* Kullanım durumu ayrıntılarında aşağıdakiler bulunur:
  * Kullanım durumunun tanımı:
    * Eşsiz bir ad, özet, oluşturma ve değiştirme tarihleri, sürüm numarası, sorumlu kişiler
  * İlgili aktörler
  * Giriş koşulu
  * Çıkış koşulu
  * Olay akışı
  * Özel gereksinimler
    * cevap süresi 5sn olmalı, yazılan şifreler görünmemeli vb.

# Bağıntılar

* Communication: Aktör-sistem etkileşimi
* Include: Birden fazla kullanım durumu için gerekli işlevler (code reuse)
* Extends : istisna olarak ya da ara sıra gerçekleşen kullanım durumlarını ifade eder. Soyutlama adına (karmaşıklığı azaltmak) ayrı olarak verilebilir.
* Kalıtım: ortak işlemler için temel kullanım durumu tanımlanır ve kalıtım kullanılır



# Para Çekme Kullanım Durumu (Use Case)



# Para Çekme Kullanım Durumu (Use Case) Ayrıntıları

* Kullanım durumunun tanımı:
  * Para çekme
  * ATM den para çekme işlemini tanımlar
  * 24.07.2009, 27.07.2009  V1.1.1  kullaniciadi1
* ilgili aktörler: Banka müşterisi, Banka BS (ikincil)
* Giriş koşulu: Banka kartına sahip olan Müşteri ATM makinasına gelir.
* Çıkış koşulu: Müşteri işlemini (transaction) tamamlar
* Olay akışı:
* Özel gereksinimler: UI gereksinimleri, İşlem gecikmesi en fazla 1 dk. olmalı,  24 saat çalışmalı vb.

# "Para Çekme" Kullanım Durumu Olay Akışı (Ana senaryo-başarılı)

1. ATM sistemi Ekrana, müşteriden "kartını kart bölmesine takmasını" isteyen mesaj yazdırır.
2. Kart sahibi kartını kart bölmesine takar.
3. ATM sistemi kart doğrulamasını yapar.
4. Ekrana şifre girilmesini isteyen mesaj gönderilir.
5. Tuş takımını kullanarak girilen şifreyi alır.
6. Kullanıcı doğrulama ve yetki kontrolü için banka bilgi sistemine istek gönderilir.
7. Banka sistemi erişim isteğini kabul eder ve gerekli bilgileri gönderir.
8. ATM sistemi ekrana, çekilecek tutarları listeler.
9. Kart sahibi çekilecek tutarı girer.
10. ATM sistemi kartı çıkartır.
11. Kart sahibi kartı alır.
12. ATM sistemi parayı ve makbuzu verir.
13. Kart sahibi parayı ve makbuzu alır.

Not: Kırmızı renkli işlem basamakları Kullanıcı Doğrulama (erişim denetimi) kullanım durumunu ifade eder.

# "Para Çekme" Kullanım Durumu Olay Akışı (Alternatif akışlar)

A1. Kart doğrulanamadı (3)

4. Kart bölmesi kartı çıkartır.

5. İşlem sonlandırılır.

A2. Yanlış şifre (6)

8. 3 den az kez yanlış ise yeniden gir.

8. 3 kez yanlış girilmiş ise kart yutulur.

9. işlem sonlandırılır.

A3. Yeterli para yok (9)

10. ATM de yeterli bakiye yoksa yeni tutar girmesi istenir.

11. Hesapta yeterli bakiye yoksa yeni tutar girmesi istenir.

A4. Banka kartı alınmadı(10)

11. 30 sn. boyunca alınmadı ise kart yutulur

12. Gerekli bilgilendirme yapılır

13. İşlem sonlandırılır

A5. Para alınmadı (12)

13. 30 sn. boyunca alınmadı ise para yutulur

14. Gönderilen ve yutulan para karşılaştırılır.

15. Gerekli ise hesap güncellenmesi yapılır

12. Gerekli bilgilendirme yapılır

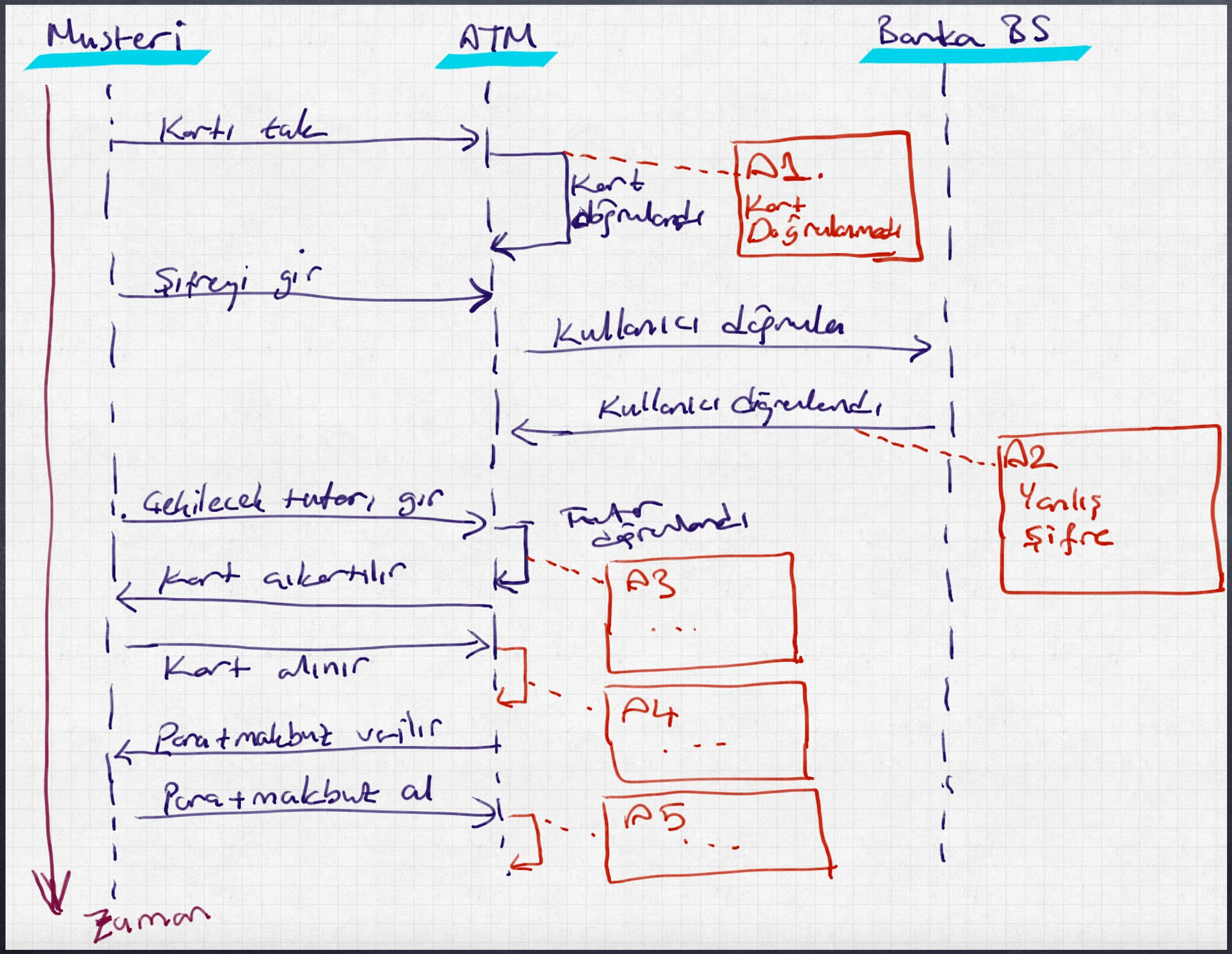
13. İşlem sonlandırılır

...

# UML - Sıralama Şeması (Sequence Diagram)

* Davranışsal (dinamik) gösterim şekillerinden biridir.
* Kullanım durumunun grafiksel gösterimi için kullanılır
* Nesneler arası işbirliğinin zamana göre anlatımı/anlaşılması gerektiğinde oluşturulabilir
* Ayrıntıları ihtiyaca göre değişiklik gösterebilir

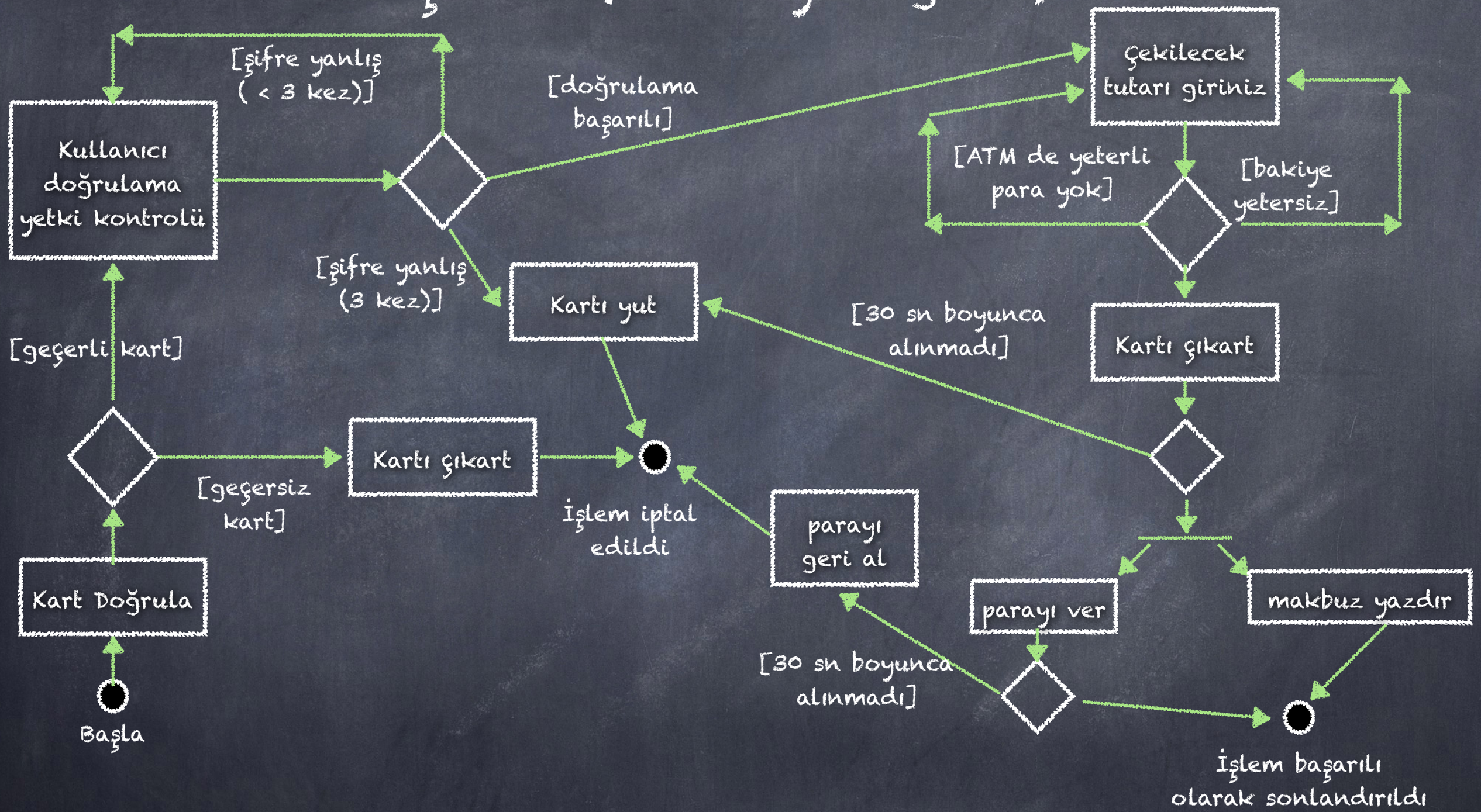
# Para Çekme Kullanım Durumu İçin ATM Sıralama Şeması



# UML - Etkinlik Şeması (Activitiy Diagram)

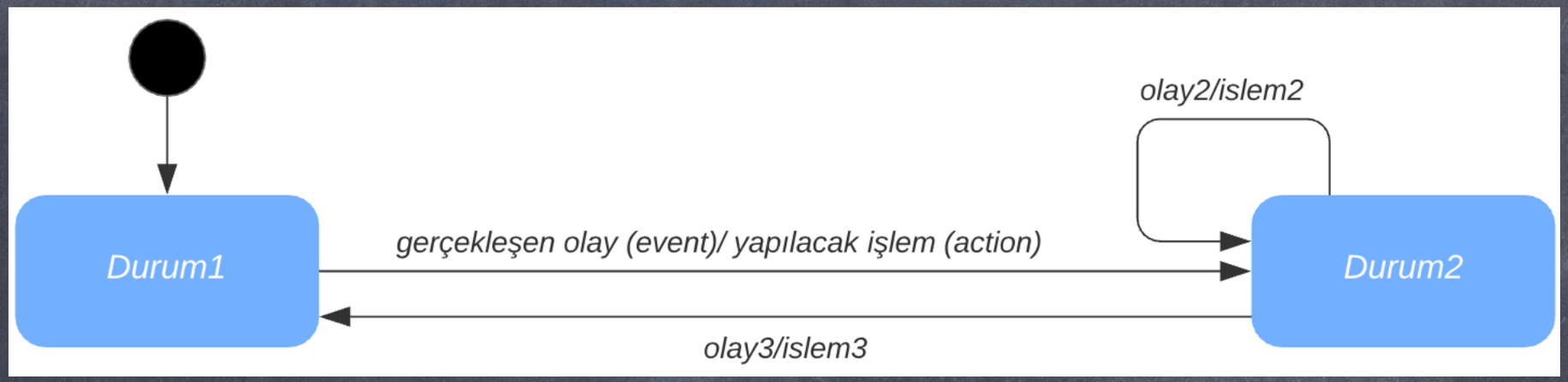
* Davranışsal (dinamik) gösterim şekillerinden biridir.
* Kullanım durumlarındaki akış denetiminin grafiksel gösterimi için kullanılır.
* Ana akışla birlikte alternatif akışları da gösterir.
* Akış şemalarına benzer.
* Ayrıntıları ihtiyaca göre değişiklik gösterebilir.

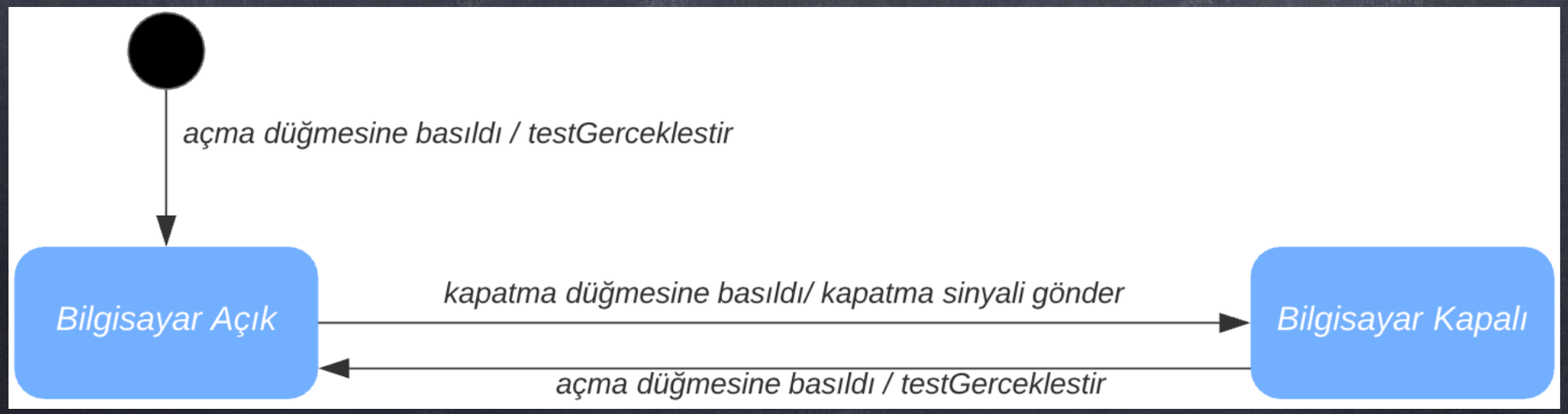
# Para Çekme Kullanım Durumu İçin ATM Etkinlik Şeması (Activitiy Diagram)



# UML - Durum Makinası (State Machine) Diyagramı

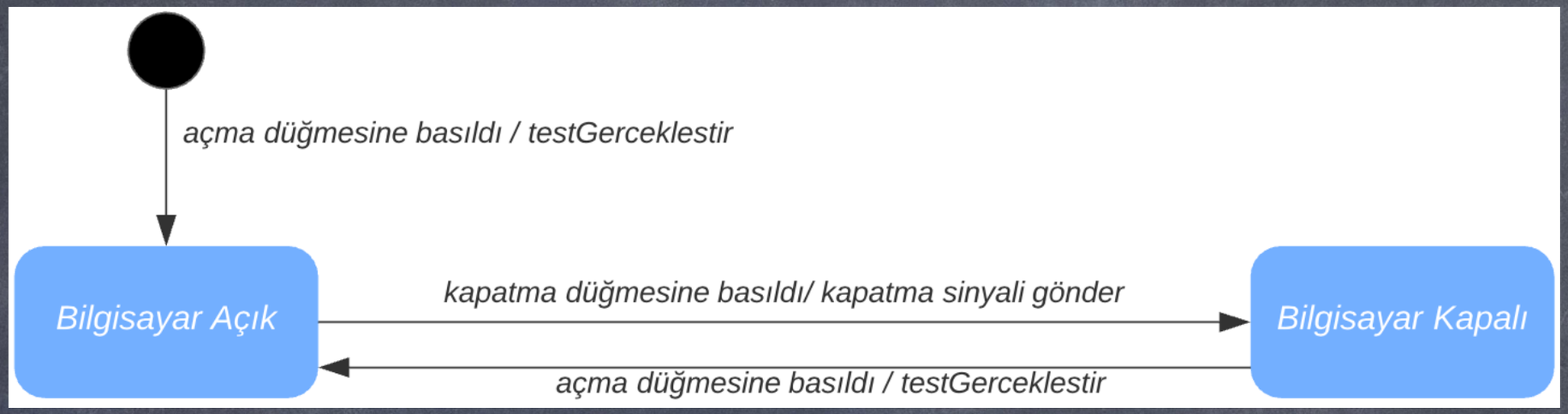
* Davranışsal (dinamik) gösterim şekillerinden biridir.
* Sistemin içinde bulunduğu durumları ve durumlar arası geçişleri modellemek için kullanılır.
* Haberleşme protokolleri, kullanıcı arayüzleri vb. olay güdümlü sistemlerin anlaşılabilmesini sağlar



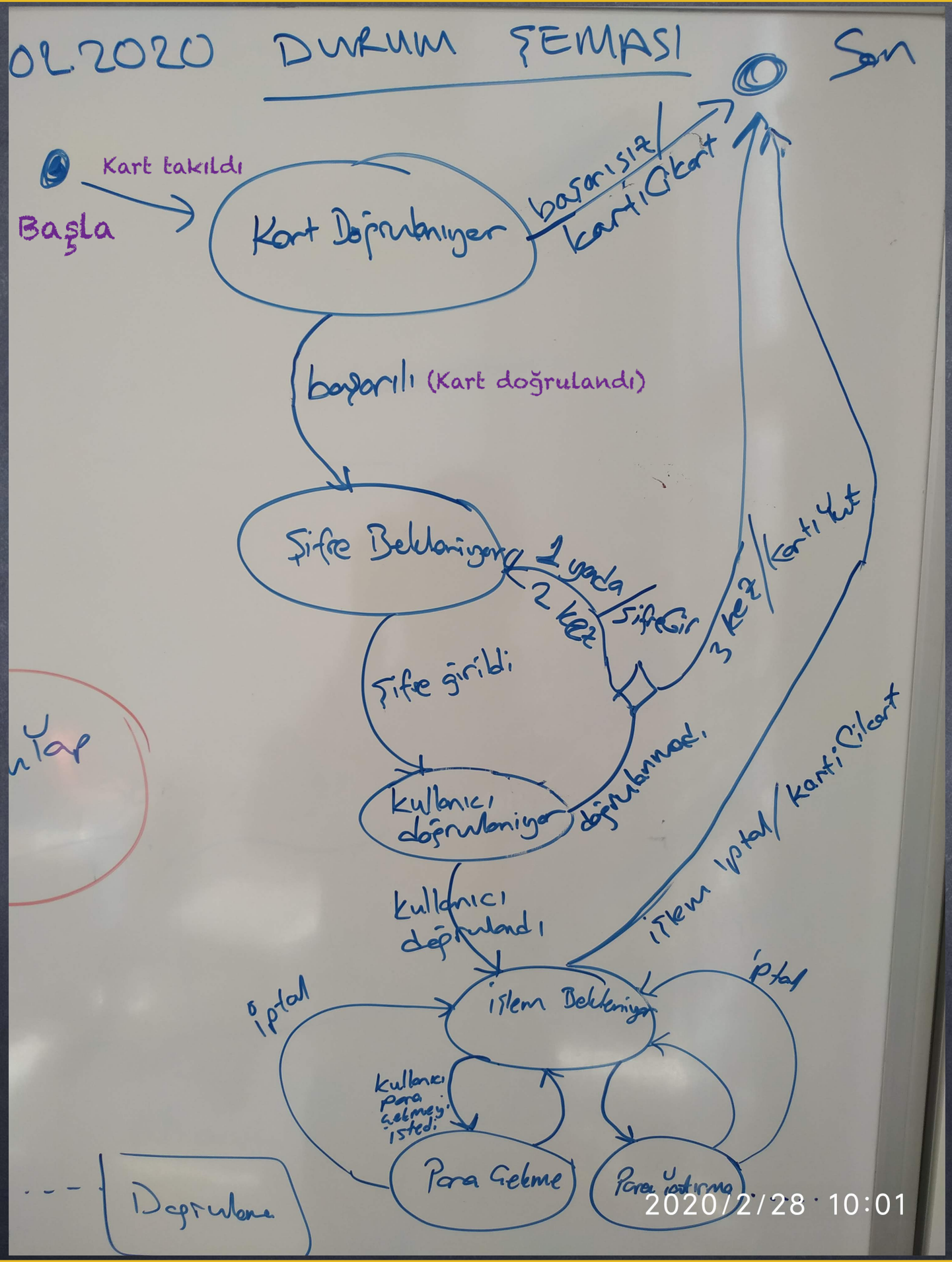
# UML - Durum Makinası (State Machine) Diyagramı

* Durum diyagramları, genişletilmesi daha kolay olacağı için, metinsel olarak ifade edilebilir.



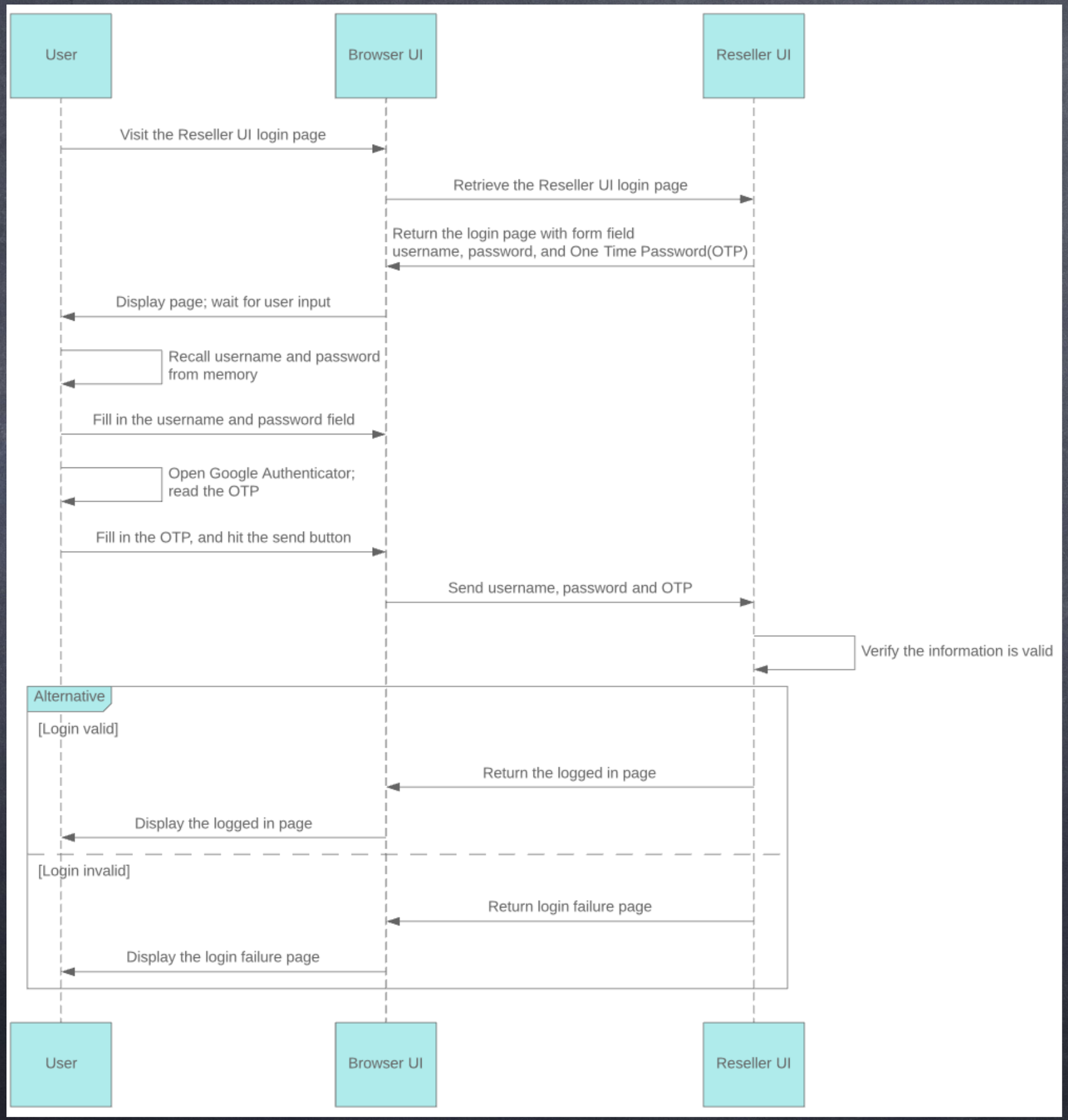

| Mevcut Durum | Gerçekleşen Olay | Yeni Durum | Hareket (action) |
|---|---|---|---|
| Başla | aç | Bilgisayar Açık | testGerçeklestir |
| Bilgisayar Açık | kapat | Bilgisayar Kapalı | kapatmaIslemi |
| Bilgisayar Kapalı | aç | Bilgisayar Açık | testGerçeklestir |

# ATM Durum Makinası Diyagramı




2020/2/28 · 10:01

# Örnek: Kimlik Doğrulama İşlemi Sıralama Şeması




Lucidchart Template

# Örnek: E-Ticaret Uygulaması

Belirtilen gereksinimlere (iş kurallarına) göre kullanım durum diyagramı oluşturunuz.

## Senaryo

Elektornik ticaret yazılımının ihtiyacı olan verilerin yönetimi için bir veritabanı geliştirilmesi isteniyor. İş kuralları verilen bu veritabanının VB diyagramını ve ilişkisel şemasını oluşturunuz.

## İş Kuralları

- Bu veritabanında her müşteriye bir numara verilerek müşterinin TC Kimlik numarası, adı, soyadı, yaşadığı ili saklanması düşünülüyor.
- İllerin plaka numarası ve adı saklanır.
- Her siparişe bakan bir satış temsilcisi mevcuttur. Satış temsilcilerinin TCKimlikNo, ad ve soyad bilgileri mevcuttur.
- Ürünlerin (kişisel bilgisayar, telefon vb.) kodu, adı, fiyatı ve stok miktarlarının saklanması gerekmektedir.
- Ürünlerin kategorileri (bilgisayar, ev elektroniği, kozmetik vb.) mevcuttur.
- Müşterilerin ürün siparişleri saklanarak her bir siparişe bir fatura kesilmesi sağlanmalıdır.
- Her siparişin eşsiz bir sipariş numarası ve sipariş tarihi mevcuttur.
- Sipariş edilen bir ürünün sipariş adedi ve birim fiyatı (kişiye özel indirim v.s. nedeniyle ürün tablosundaki fiyattan farklılık gösterebilir) da kaydedilmelidir.
- Faturaların fatura numarası, tarih ve fatura adresi bilgileri saklanmalıdır. (Toplam fiyat hesaplanabilir ya da saklanabilir).
- Siparişler bir kargo firması tarafından iletilir. Kargo firmasının kodu, adı, adresi bilgileri yer alır. Her kargo firmasında siparişlerden sorumlu bir yetkili yer alır.
- Bir ürünün yalnızca bir kategorisi mevcuttur. Bir kategori çok sayıda ürünün kategorisi olabilir.
- Bir siparişte en az bir ürün bulunur. Ancak çok sayıda ürün de bulunabilir. Bir ürün çok sayıda siparişte yer alabilir.
- Bir müşteri çok sayıda sipariş verebilir. Bir sipariş yalnızca bir müşteri tarafından verilebilir.
- Bir siparişin yalnızca bir faturası olabilir. Bir fatura yalnızca bir siparişin faturası olabilir.
- Bir sipariş ile yalnızca bir satış temsilcisi ilgilenir. Bir satış temsilcisi çok sayıda sipariş ile ilgilenebilir.
- Bir müşteri yalnızca bir ilde yaşayabilir. Bir ilde çok sayıda müşteri yaşayabilir.
- Bir sipariş yalnızca bir kargo firması tarafından iletilir. Bir kargo firması çok sayıda sipariş iletebilir.

# Nesne Yönelimli Analiz ve Tasarım

Nesnelerin Belirlenmesi
Sınıf Şeması
Nesneler Arası Bağıntılar
Modelin Koda Dönüştürülmesi
Örnek Uygulamalar (Veri Topluluklarıyla İşlemler),
Örnekler (Sıralama Şeması, Etkinlik Şeması, Durum Şeması)

# Nesne Yönelimli Analiz

* Analiz aşamasının ilk adımı gereksinimlerin belirlenmesidir.
* Gereksinimler, sistemin düzgün çalışması için yapması beklenenleri ifade eder
* Gereksinimler ilk olarak kullanıcılarla(paydaşlarla) görüşülerek (gerekli ise standart, yasa, yönetmelik vb. dokümanlar takip edilerek) oluşturulur, ve neler istendiği belirlenir.
* Daha sonra kullanım durumları (use case) belirlenerek gereksinimler ayrıntılandırılır.
* Sistemin tüm istenenlerle birlikte, beklenmeyen ya da istenmeyen durumları da göz önüne alınır.
* Kullanım durumları analiz edilerek nesneler, nesne üyeleri ve nesneler arasındaki bağıntılar belirlenir.

# Nesnelerin Belirlenmesi

* NYP geçek dünyayı daha doğru modelleyebilmek için geliştirilmiştir.
* Gerçek hayatta çevremizde gördüğümüz her şey nesnedir.
  * insan, ürün, sipariş, fatura, öğrenci, ekran, televizyon, masa, sıra, bisiklet, araba, köpek v.s.
* Nesnelerin durum ve davranışları bulunur.
  * İnsan için durumlar; adı, yaşı, boyu v.s. iken davranışlar; öğrenmek, anlamak, uyumak, konuşmak, koşmak v.s.
  * Bisiklet için durumlar; rengi, o anki vitesi, hızı, tekerlek sayısı, vites sayısı v.s. iken davranışlar; fren yapması, hızlanması, yavaşlaması, vites değiştirmesi v.s.
* Yazılım nesneleri de durum ve davranışlara sahiptir.
  * Nesnelerin durumları nitelik (özellik) olarak da adlandırılır ve üye değişkenler ile ifade edilir.
  * Davranışlar ise üye fonksiyonlar/methodlar/yöntemler ile temsil edilir.

# Nesnelerin Belirlenmesi

* Gereksinimlerden nesnelerin belirlenmesi işlemi için; nesne modelleme, alan modelleme, kavramsal modelleme gibi isimler kullanılır.
* Nesneler belirlenirken, analiz aşamasında oluşturulan gereksinim listesi (kullanım durumlarının ayrıntıları) üzerinde metinsel analiz yapılır (Abbott's tekniği).
* Gereksinim listesindeki isim ve isim tamlamaları sınıf adaylarıdır. Özellikleri ya da davranışları bulunan isim ya da isim tamlamaları sınıf olabilir.
* Gereksinim listesindeki eylemler nesnelerin üye yöntemlerini ifade edebilir.
* Bu işlem sadece bir başlangıçtır ve yinelemeli bir süreçtir.
* Gereksinim listesi incelendikçe yeni sınıflar, özellikler, yöntemler, sınıflar arası bağlantılar belirlenir. Sınıflar değiştirilebilir, niteliğe dönüşebilir, kullanılmayabilir. Nitelikler sınıfa dönüşebilir vb.
* Sınıflar belirlendikten sonra tasarım ayrıntıları içermeyen Analiz seviyesi Sınıf Şeması (kavramsal model) oluşturulur.

# Nesnelerin Belirlenmesi

Nesne adayları (Craig Larman ):

* Fiziksel ya da elle tutulabilir nesneler (ürün, insan, bilgisayar, klavye, ekran ...)
* Yerler (okul , bina, yerleşke, sınıf, oda ...)
* İşlemler (transactions) (para çekme..., kayıt yaptırma...)
* Roller (yönetici, öğrenci, personel, kayıtlı kullanıcı v.b.)
* Harici sistemler (Veritabanları, sürücüler, banka bilgi sistemi, web servisleri... )
* Kuruluşlar (okul, işyeri, firma...)
* Olaylar (ActionListener, ActionEvent, KeyListener, KeyEvent...)
* loglar ...

Nesnelerin doğru olarak belirlenmesi için en etkili yol paydaşlarla görüşmektir (beyin fırtınası).

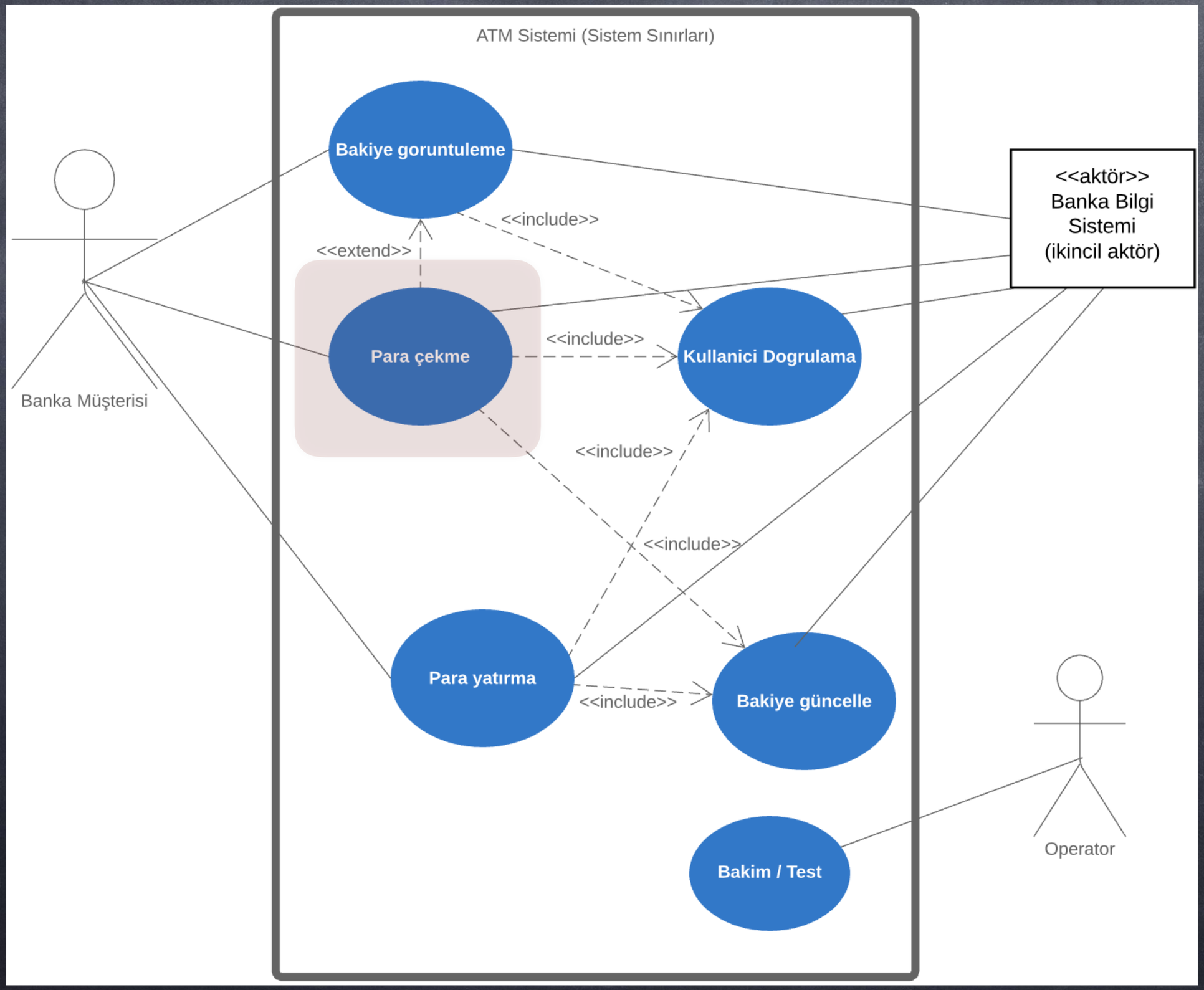
# ATM Sistemindeki Nesnelerin Belirlenmesi

* ATM sistemi aşağıdaki donanım bileşenlerinden oluşur:
  * Tuş takımı; ile veri girişi yapılır.
  * Ekran; kullanıcıya mesaj görüntüler.
  * Para bölmesi; para alma ve para verme işleminden sorumludur.
  * Kart bölmesi; banka kartını alır ve doğrular.
* Müsteriler hesaplarından banka bilgi sistemini kullanarak; para çekme, para yatırma, bakiye görüntüleme vb. işlemlerini yapabilmelidirler.
* Banka görevlileri ATM sisteminin bakımını (para doldurma, sıkışan/yutulan kartları alma v.b. ) yapmalıdır.
* Tüm işlemlerde güvenlik gereksinimleri göz önünde bulundurulmalıdır (yetki kontrolü).
* Sistem, Banka Bilgi Sistemiyle çevrimiçi çalışmalıdır. Kullanıcılar Banka Bilgi Sisteminde yer alan hesaplarına erişerek işlemlerini gerçekleştirirler.

Not: Sarı renkliler nesne (sınıf), yeşil renkliler yöntem adaylarıdır.
Aynı cümlede bulunan nesneler bağıntılıdır.

# ATM Sistemindeki Nesnelerin Belirlenmesi

## Para Çekme Kullanım Durumu (Use Case)



# ATM Sistemindeki Nesnelerin Belirlenmesi

## Use Case- "Para Çekme" Olay Akışı (Ana senaryo-başarılı)

1. ATM sistemi Ekrana, müşteriden "kartını kart bölmesine takmasını" isteyen mesaj yazdırır.
2. Kart sahibi kartını kart bölmesine takar.
3. ATM sistemi kart doğrulamasını yapar.
4. Ekrana şifre girilmesini isteyen mesaj gönderilir.
5. Tuş takımını kullanarak girilen şifreyi alır.
6. Kullanıcı doğrulama ve yetki kontrolü için banka bilgi sistemine istek gönderilir.
7. Banka sistemi erişim isteğini kabul eder ve gerekli bilgileri gönderir.
8. ATM sistemi ekrana, çekilecek tutarları listeler.
9. Kart sahibi çekilecek tutarı girer.
10. ATM sistemi kartı çıkartır.
11. Kart sahibi kartı alır.
12. ATM sistemi parayı ve makbuzu verir.
13. Kart sahibi parayı ve makbuzu alır.

Not: Kırmızı renkli işlem basamakları Kullanıcı Doğrulama (erişim denetimi) kullanım durumunu ifade eder.

# ATM Sistemindeki Nesnelerin Belirlenmesi

## "Para Çekme" Kullanım Durumu Olay Akışı (Alternatif akışlar)

A1. Kart doğrulanamadı (3)

4. Kart bölmesi kartı çıkartır.

5. İşlem sonlandırılır.

A2. Yanlış şifre (6)

8. 3 den az kez yanlış ise yeniden gir.

8. 3 kez yanlış girilmiş ise kart yutulur.

9. işlem sonlandırılır.

A3. Yeterli para yok (9)

10. ATM de yeterli bakiye yoksa yeni tutar girmesi istenir.

11. Hesapta yeterli bakiye yoksa yeni tutar girmesi istenir.

A4. Banka kartı alınmadı(10)

11. 30 sn. boyunca alınmadı ise kart yutulur

12. Gerekli bilgilendirme yapılır

13. İşlem sonlandırılır

A5. Para alınmadı (12)

13. 30 sn. boyunca alınmadı ise para yutulur

14. Gönderilen ve yutulan para karşılaştırılır.

15. Gerekli ise hesap güncellenmesi yapılır

12. Gerekli bilgilendirme yapılır

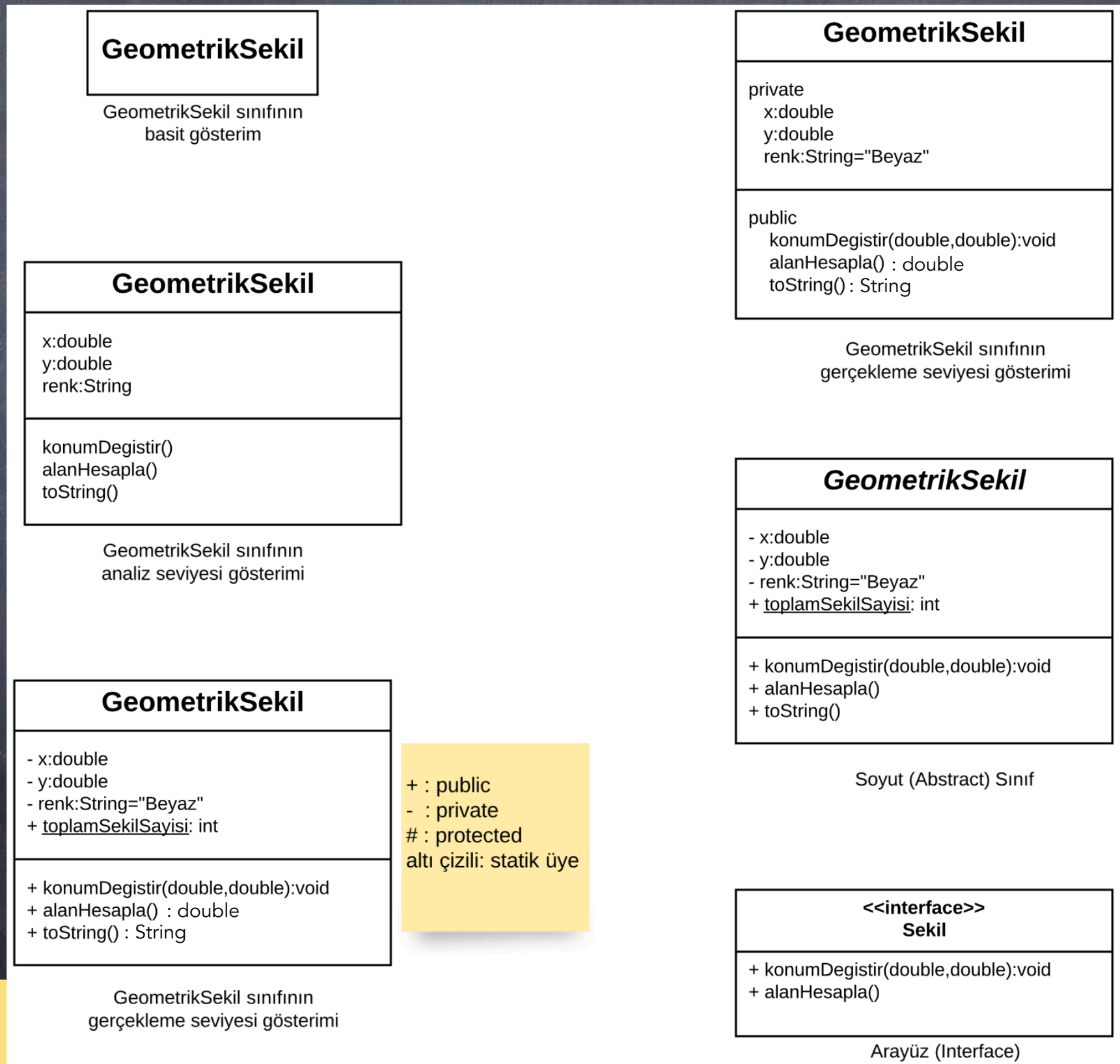
13. İşlem sonlandırılır

...

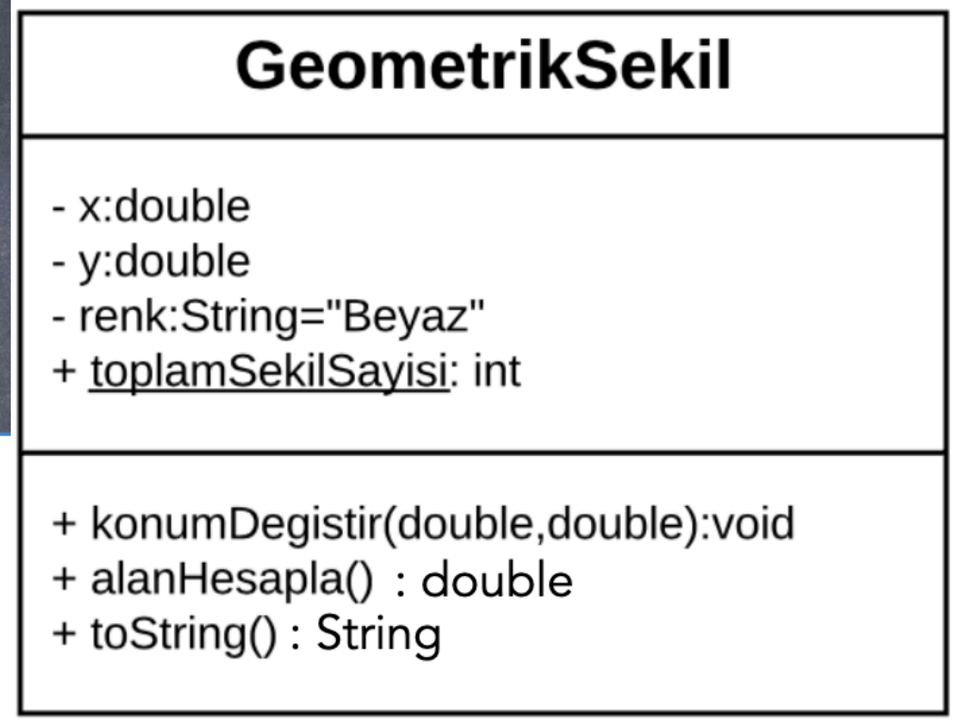
# UML - Sınıf Diyagramı

* Yapısal gösterim şekillerindendir.
* Tasarlanan sistemin sınıflar ve arayüzler seviyesinde statik yapısını gösterir.
* Sınıf ve arayüzlerin; özelliklerini, üyelerini, kısıtlarını ve bağıntılarını içerir.

https://www.uml-diagrams.org/uml-25-diagrams.html



# Modelin Koda Dönüştürülmesi / Sınıf




+ : public
- : private
# : protected
altı çizili: statik üye

```
package cc.ders6;

public class GeometrikSekil {

    private double x;
    private double y;
    private String renk;
    public static int toplamSekilSayisi;

    public void konumDegistir(double x, double y){
        .....
    }


    ....

}
```

GeometrikSekil sınıfının
gerçekleme seviyesi gösterimi

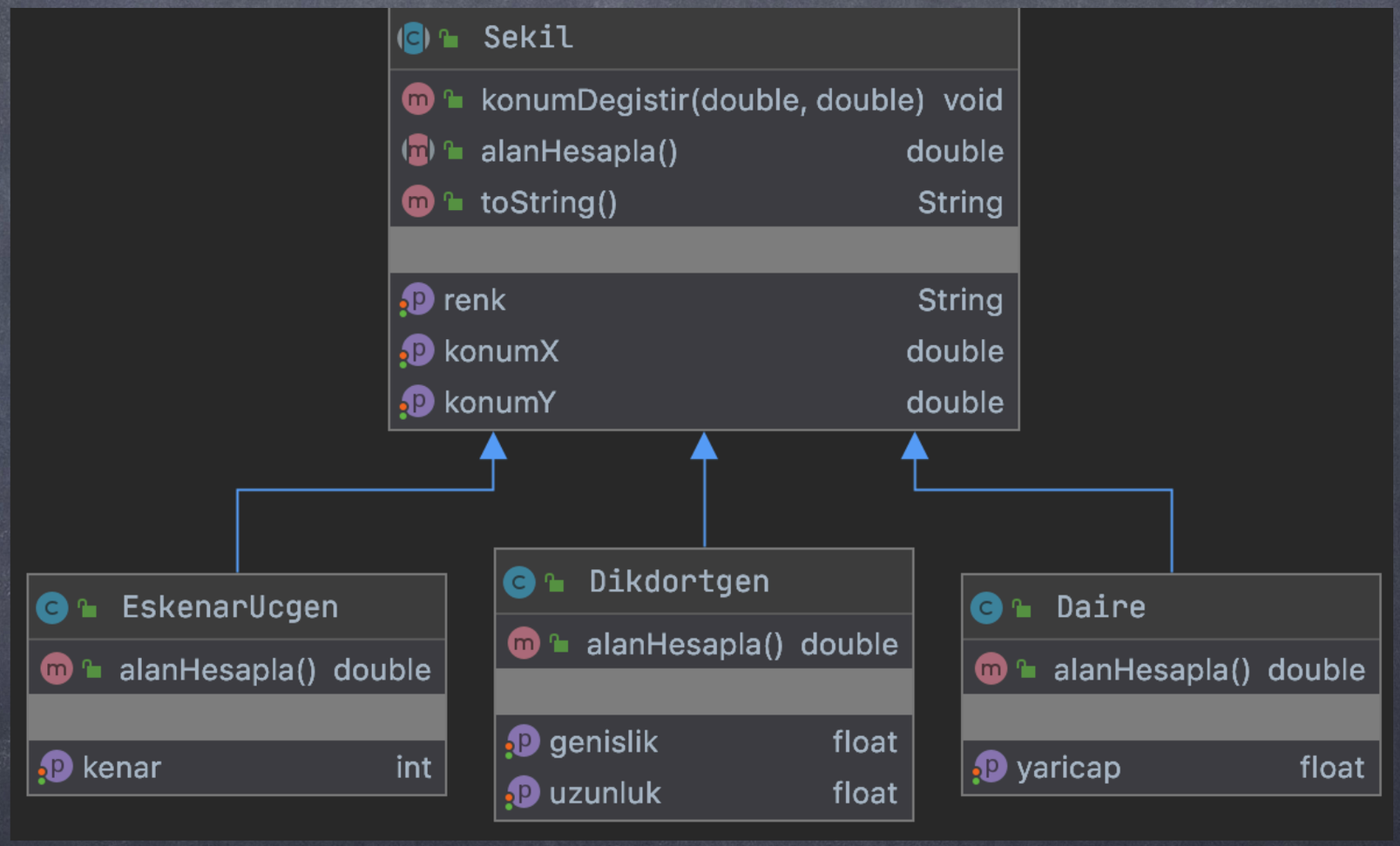
# Veri Topluluklarıyla İşlemler

https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders4/VeriTopluluklariUzerindeIslemler

# Sınıf Diyagramı - Elementler arası bağıntılar

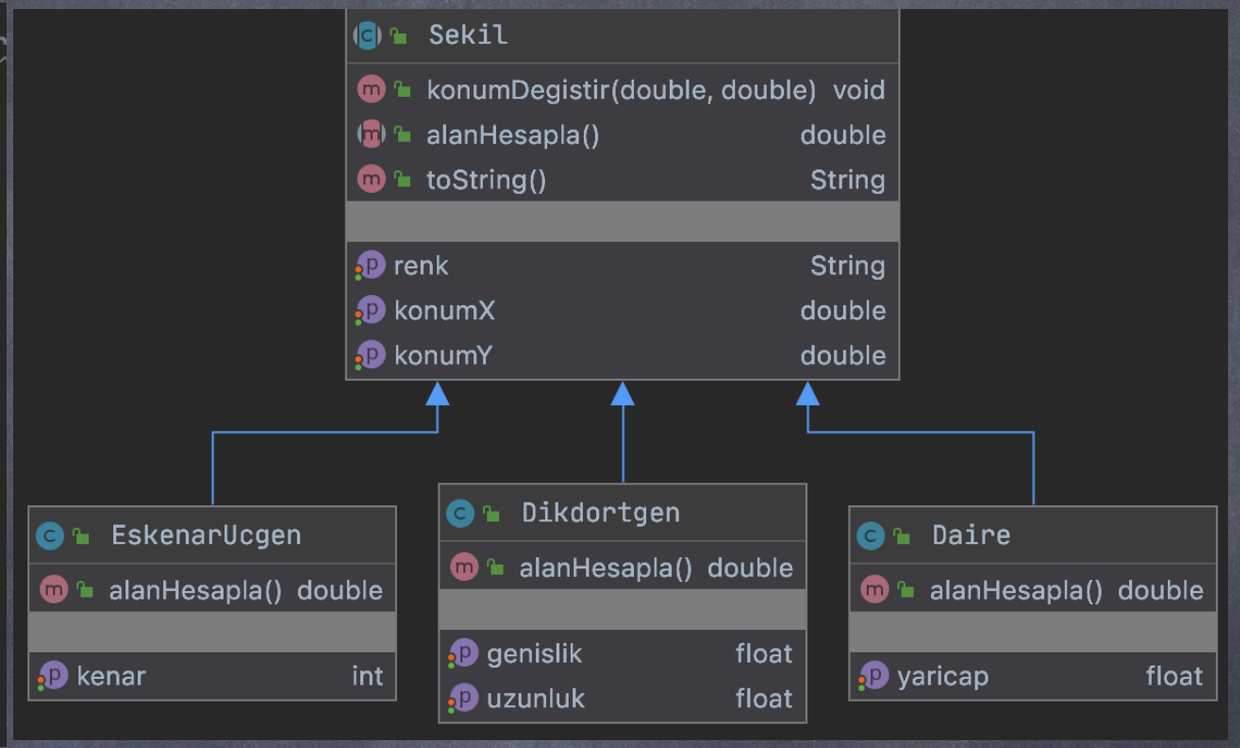
## Kalıtım Bağıntısı

* Benzer varlıklar arasında olmalıdır.
* "is a" , "is kind of"
* Türetilmiş sınıf (Derived class) Temel Sınıfın (base class) üyelerine kalıtım yoluyla sahip olur.
* Kod tekrar kullanımı sağlanır



# Sınıf Diyagramı - Elementler arası bağıntılar

```java
abstract public class Sekil {  //Soyut yöntem sadece soyut sınıflar
    private double konumX;
    private double konumY;
    private String renk;

    public Sekil(double konumX, double konumY, String renk) {
        this.konumX = konumX;
        this.konumY = konumY;
        this.renk = renk;
    }


    public double getKonumX() { return konumX; }

    public void setKonumX(double konumX) { this.konumX = konumX; }
```




```java
public class Daire extends Sekil {

    private float yaricap;

    public Daire(double konumX, double konumY, String renk, float yaricap) {
        super(konumX, konumY, renk);
        this.yaricap = yaricap;
    }

    public float getYaricap() { return yaricap; }

    public void setYaricap(float yaricap) { this.yaricap = yaricap; }
```

# Sınıf Diyagramı - Elementler arası bağıntılar

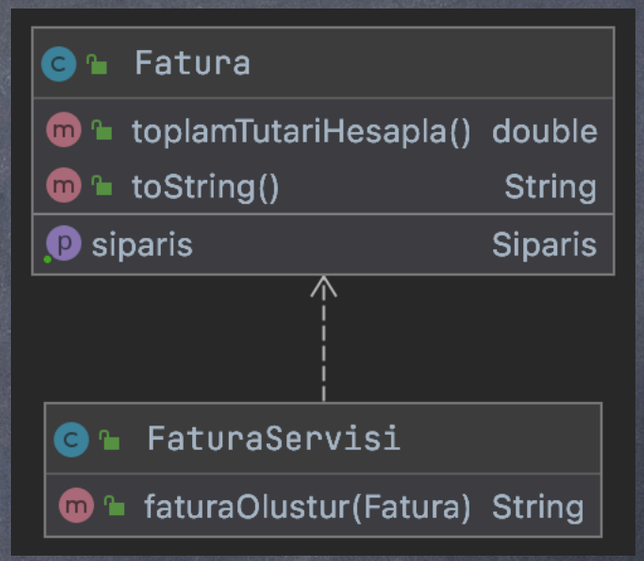
"Dependency" Bağıntısı

* Bir elementin (istemci- ClassA) diğer elementi (tedarikçi-ClassB) kullanmasını ifade eder.

* "uses"

* ClassA içerisindeki bir yöntem ClassB yi kullanır.

* FaturaServisi sınıfında yer alan yöntem, sorumluluğunu yerine getirebilmek için Fatura elementini kullanır.




```
public class FaturaServisi {

    public String faturaOlustur(Fatura fatura){

        String faturaXML="<XML>"+fatura.toString();
        return faturaXML;
    }

    .......
```

# Sınıf Diyagramı - Elementler arası bağıntılar

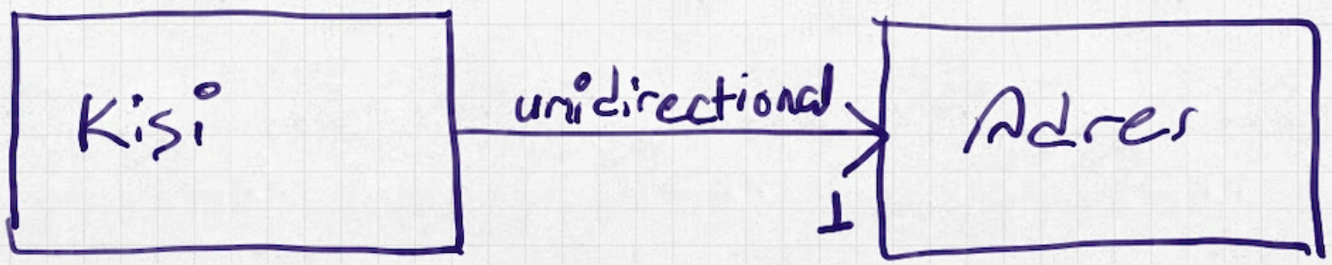
## "Association" Bağıntısı

* Bir nesne diğer nesneyi üye (özellik) olarak içerir.
* "has a"
* "Bidirectional" (iki yönlü) ve unidirectional (tek yönlü) olabilir.
* "Aggregation" ve "Composition" bağıntıları "Association" bağıntısının özel halleridir.

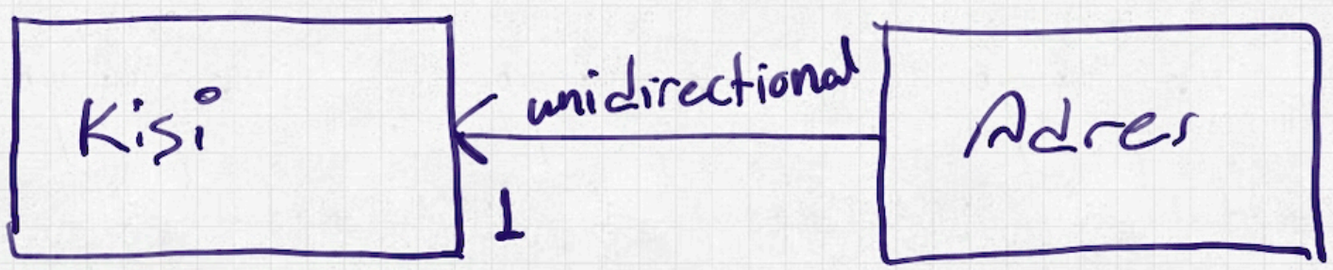
* 1 Musteri nesnesi çok sayıda Siparis nesnesini (referansını) içerir.

# Sınıf Diyagramı - Elementler arası bağıntılar

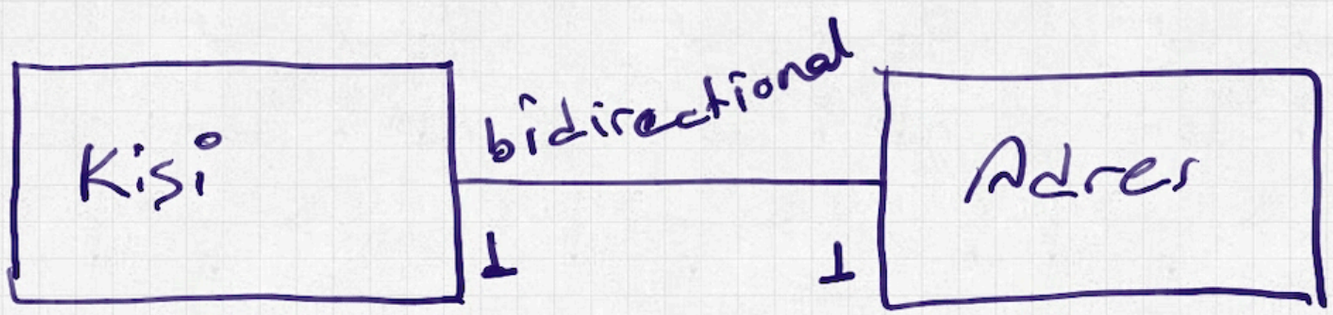
## "Association" Bağıntısı




```
class Kisi
{ private Adres adres;
  ⋮
```



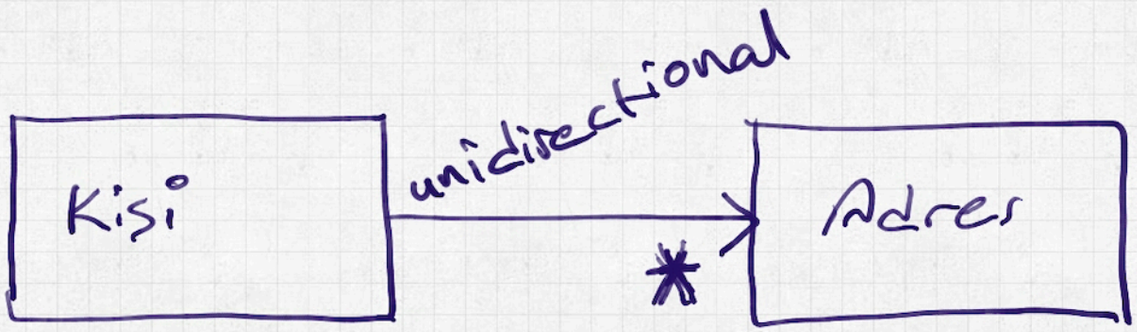

```
class Adres
{ private Kisi kisi;
  ⋮
```



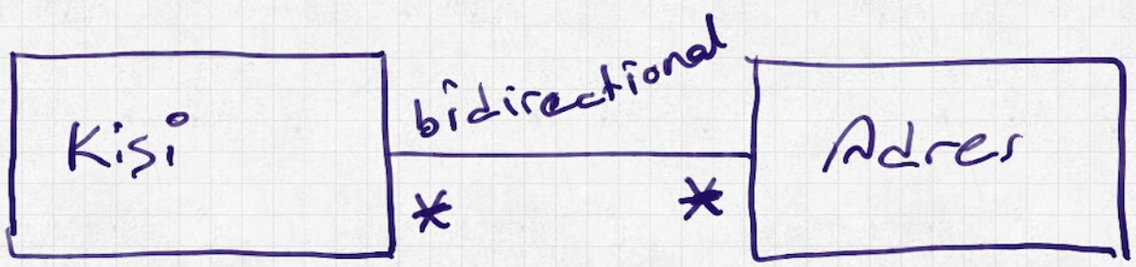

```
class Kisi
{ private Adres adres;
  ⋮
class Adres
{ private Kisi kisi;
```

# Sınıf Diyagramı - Elementler arası bağıntılar

## "Association" Bağıntısı




```
class Kişi
{ private List<Adres> adresler;
  ⋮
```

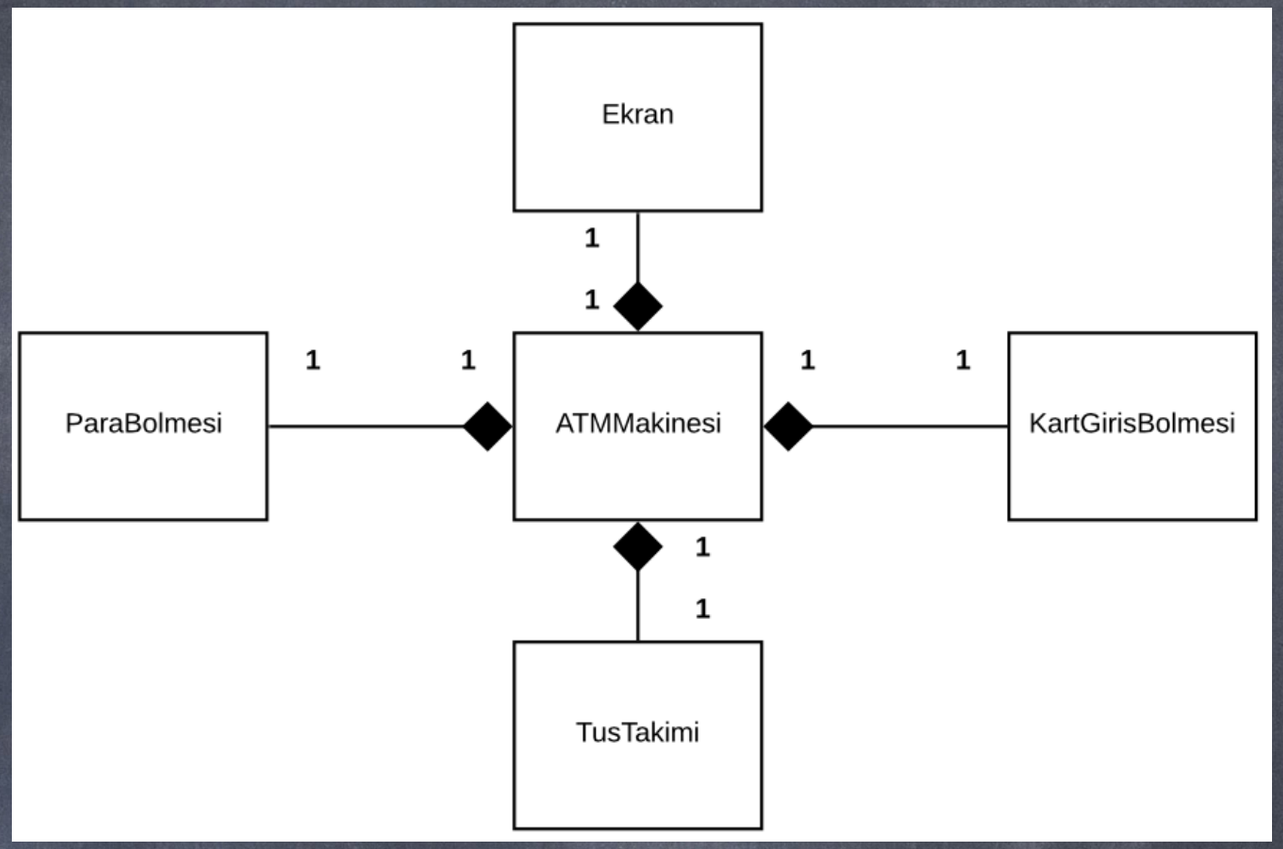



```
class Kisi
{ private List<Adres> adresler;
  ⋮
class Adres
{ private List<Kisi> kisiler;
  ⋮
```

# Sınıf Diyagramı - Elementler arası bağıntılar

"Composition" Bağıntısı

* Bütün parça bağıntısı (whole/part)
* Bütün, işini yapabilmek için parçaya ihtiyaç duyar.
* Parça nesneler bütün içerisinde yer alırlar. parçaları bütün oluşturur ve yok eder.
* "is part of" (whole/part)
* Her parça en çok 1 bütün içerisinde bulunur. Bir bütünde çok sayıda parça yer alabilir.

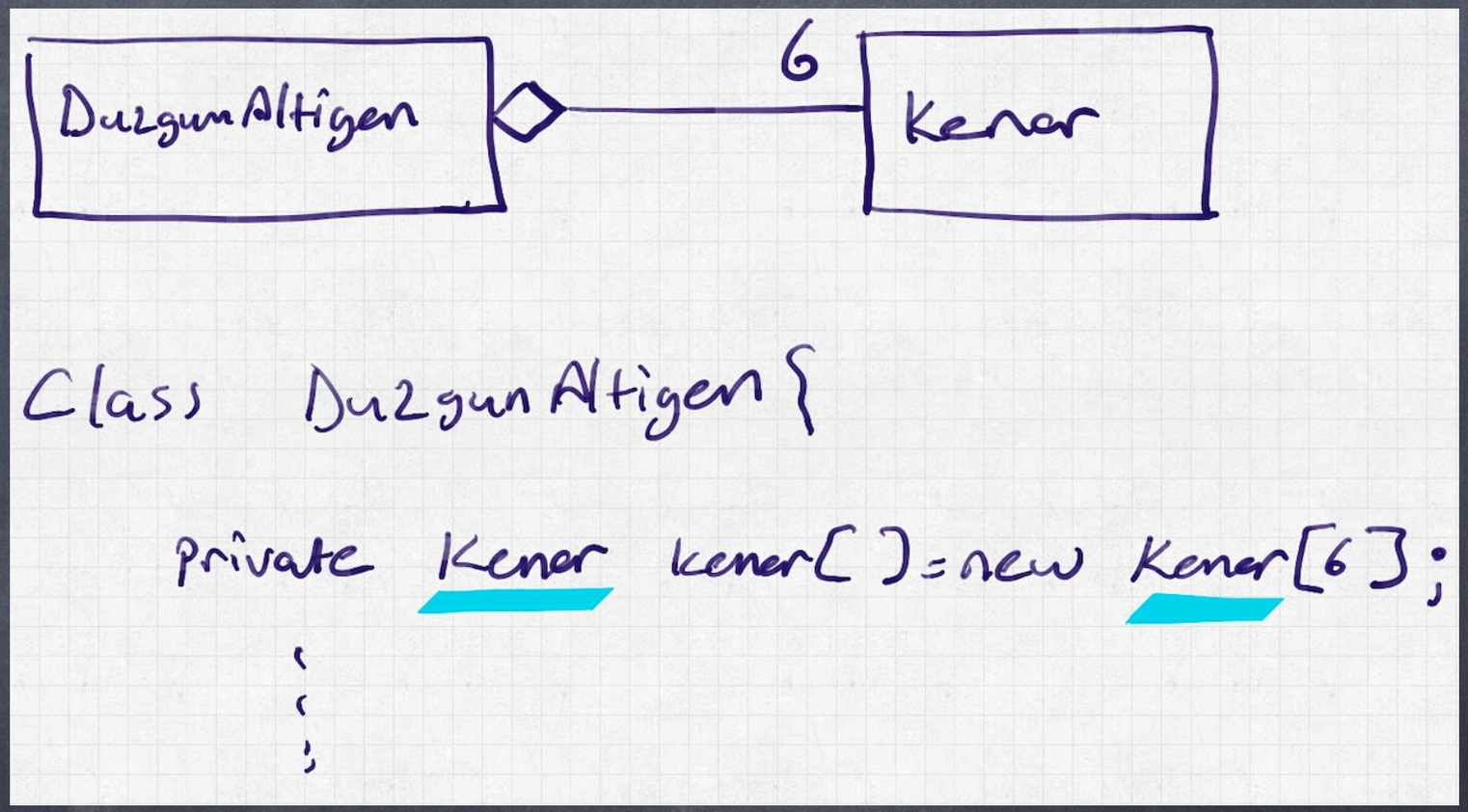



```
public class ATMMakinesi
{
    private Ekran ekran;
    private TusTakimi tusTakimi;
    private ParaBolmesi paraBolmesi;
    private KartGirisBolmesi kartBolmesi;

    public ATM() {
        ekran=new Ekran();
        tusTakimi=new TusTakimi();
        paraBolmesi=new ParaBolmesi();
        kartBolmesi=new KartGirisBolmesi();
    }
```

# Sınıf Diyagramı - Elementler arası bağıntılar

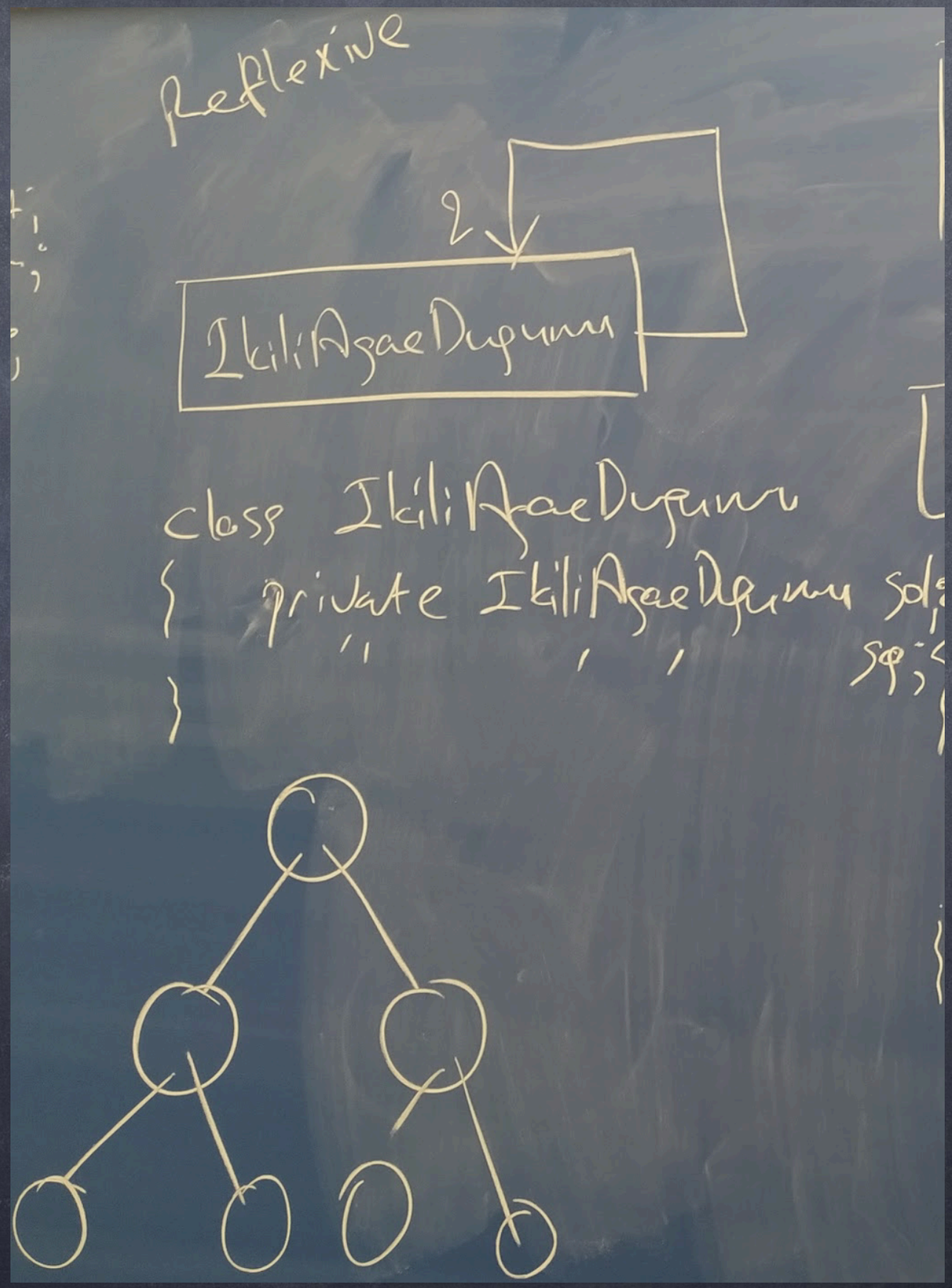
"Aggregation" Bağıntısı

* Bütün parça bağıntısı (whole/part)
* Bütün, işini yapabilmek için parçaya ihtiyaç duyar.
* Parça nesneler bütün içerisinde yer alırlar. parçalar başka bütün içerisinde de kullanılabilir. Bütün yok edildiğinde parçanın yok edilmesi gerekmeyebilir (başka bütün içerisinde kullanılıyor olabilir)
* "consists of"



# Sınıf Diyagramı - Elementler arası bağıntılar

* "Reflexive" Bağıntısı



# Sınıf Diyagramı - Elementler arası bağıntılar

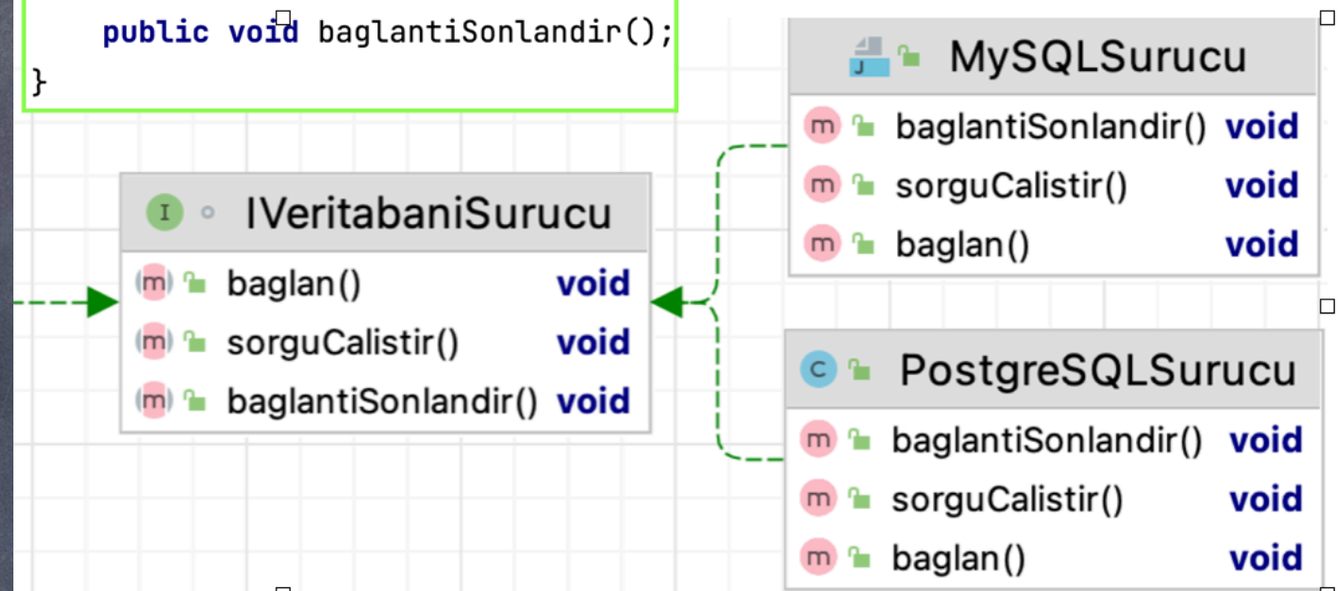
"Interface"

* Bir sınıfın yapmasını istediklerimizi belirtmek için kullanırız ("is capable of"). Ne yapması gerektiği belirtilir, nasıl ile ilgilenilmez.

Arayüzler sayesinde:

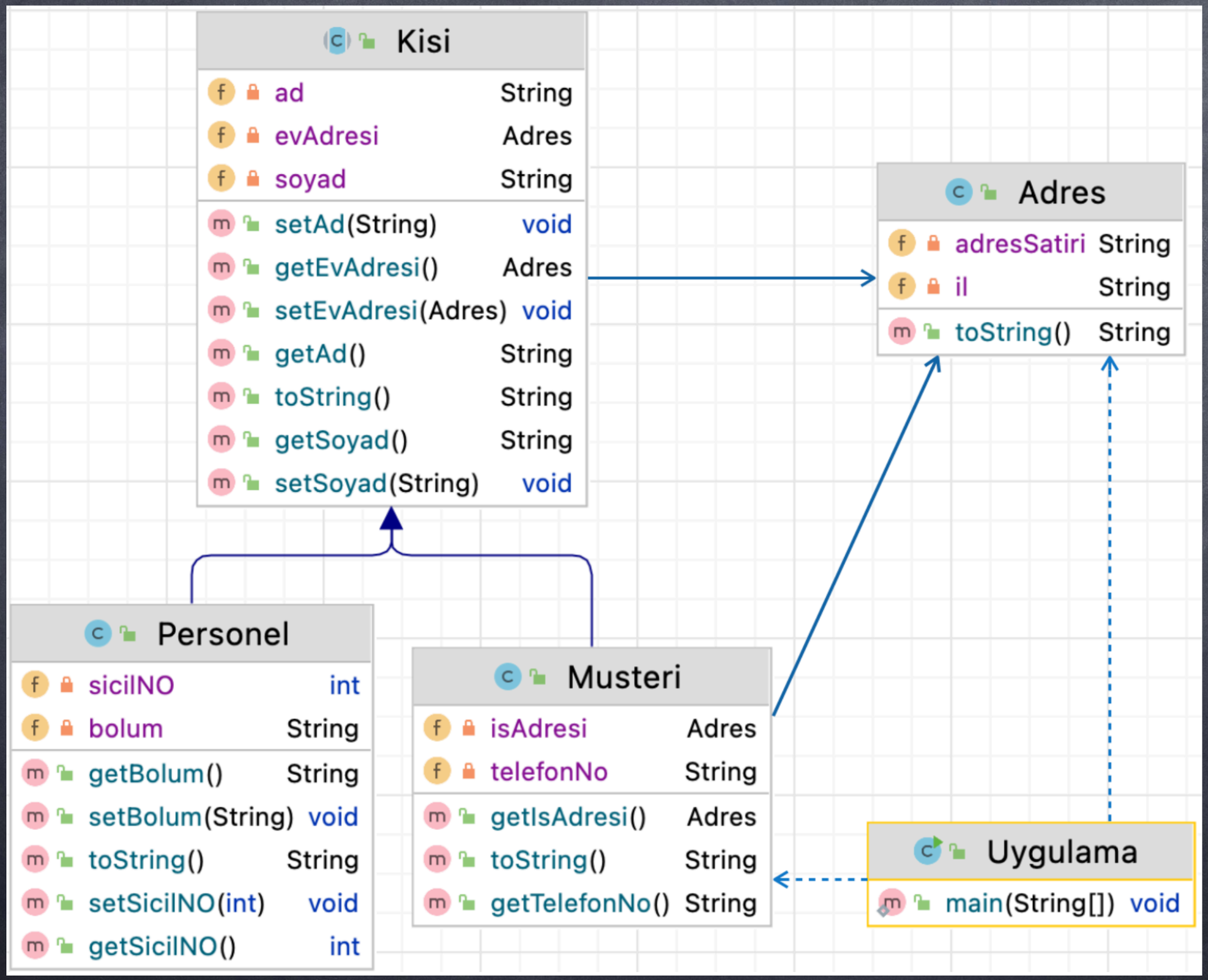
* gerçeklemeler(sınıflar) henüz ortada yok iken istemci kod içerisinde kullanılabilir
* gereçekleme ve istemci modüller aynı anda farklı kodlayıcılar tarafından oluşturulabilir (takım çalışması)
* yeni özellik eklendiğinde istemci kod değiştirilmez (sözleşme yapılır)
* istemci kod, gerçekleme kısmındaki değişikliklerden etkilenmez (bağımlılığın zayıflatılması)
* aynı istemci kod farklı modüller tarafından defalarca kullanılabilir (kod tekrar kullanımı-code reuse)

```java
interface IVeritabaniSurucu {

    public void baglan();
    public void sorguCalistir();
    public void baglantiSonlandir();
}
```




```java
public class PostgreSQLSurucu implements IVeritabaniSurucu {

    public void baglan() {
        System.out.println("PostgreSQL veritabanına bağlanıyor...");
    }

    public void sorguCalistir() {
        System.out.println("PostgreSQL sorgusu çalıştırılıyor...");
    }

    public void baglantiSonlandir() {

        System.out.println("PostgreSQL veritabanı bağlantısı sonlandırıl

    }
}
```

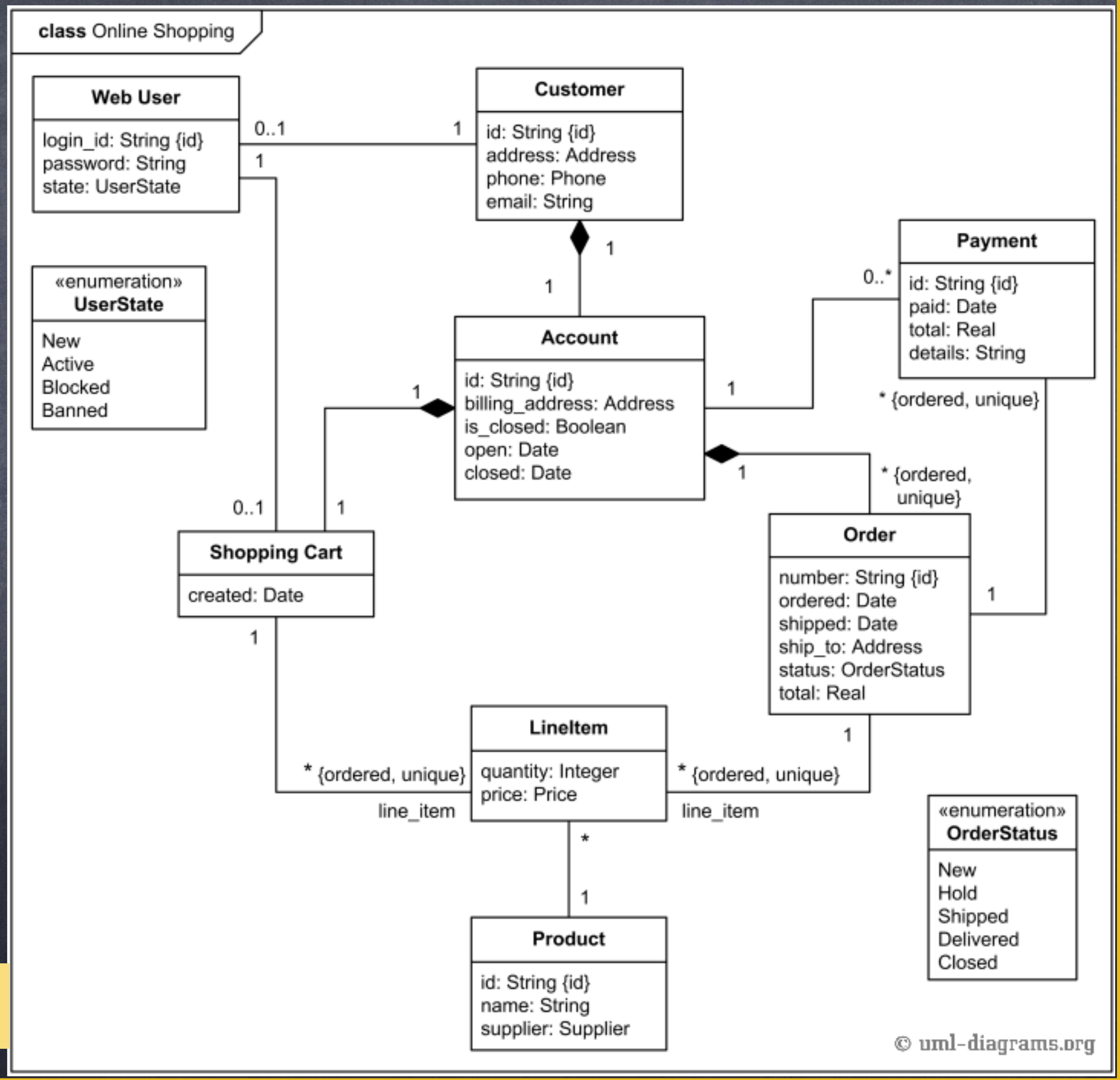
# Örnek Uygulama: Elementler arası bağıntılar




https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders5

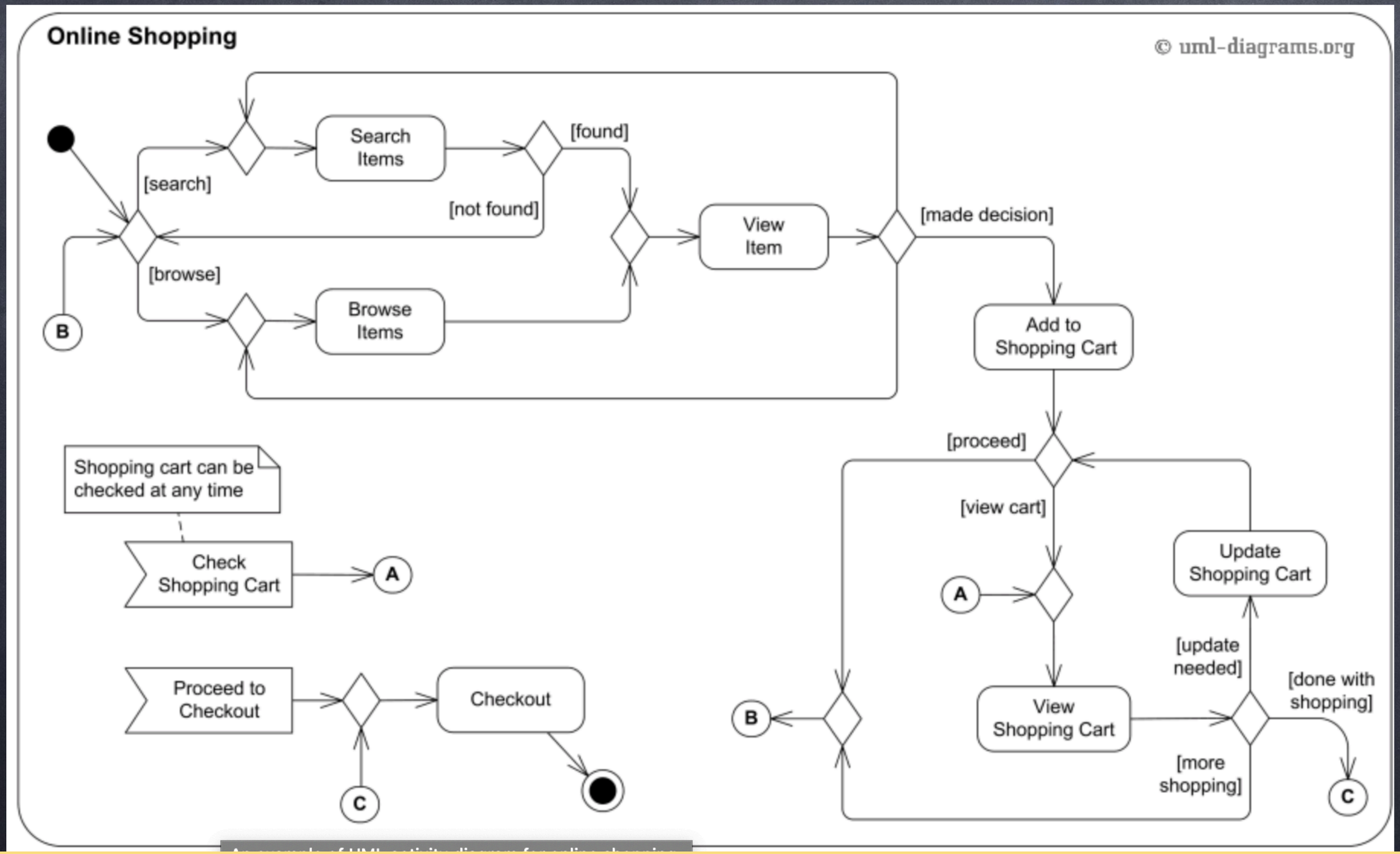
# Örnek: Çevrimiçi Alış Veriş Sistemi Sınıf Şeması

- Her müşterinin eşsiz bir "id" si ve sadece bir hesabı vardır.
- Hesaba ait bir alış veriş kartı (sepeti) (internet üzerinde alışveriş için) ve siparişler (klasik alışveriş, telefon, yüz yüze vb.) (0 ya da çok) vardır.
- Sipariş ve alış veriş kartı alışveriş kalemlerine sahiptir.
- Alış veriş kalemleri mutlaka bir ürünle ilgilidir. Ürünler çok sayıda alış veriş kalemleriyle ilgili olabilir.



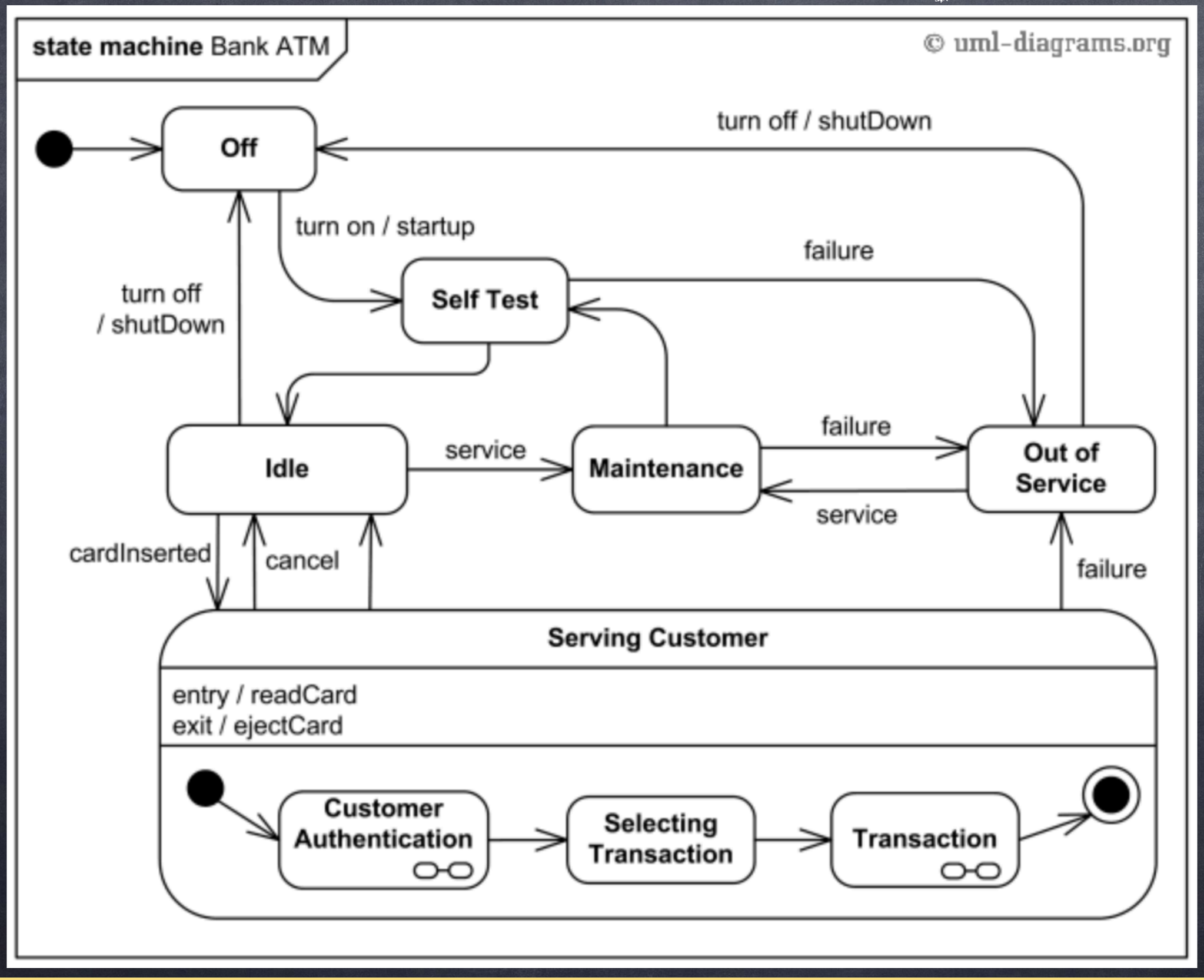

https://www.uml-diagrams.org/examples/online-shopping-domain-uml-diagram-example.html?context=cls-examples

# Örnek: Çevrimiçi Alış Veriş Sistemi Etkinlik Şeması




https://www.uml-diagrams.org/online-shopping-uml-activity-diagram-example.html?context=activity-examples

# Örnek: ATM Durum Makinası Diyagramı




https://www.uml-diagrams.org/bank-atm-uml-state-machine-diagram-example.html?context=stm-examples

# Örnek: E-Ticaret Uygulaması

Belirtilen gereksinimlere (iş kurallarına) göre nesneleri belirleyerek sınıf şemasını oluşturunuz.

## Senaryo

Elektornik ticaret yazılımının ihtiyacı olan verilerin yönetimi için bir veritabanı geliştirilmesi isteniyor. İş kuralları verilen bu veritabanının VB diyagramını ve ilişkisel şemasını oluşturunuz.

## İş Kuralları

- Bu veritabanında her müşteriye bir numara verilerek müşterinin TC Kimlik numarası, adı, soyadı, yaşadığı ili saklanması düşünülüyor.
- İllerin plaka numarası ve adı saklanır.
- Her siparişe bakan bir satış temsilcisi mevcuttur. Satış temsilcilerinin TCKimlikNo, ad ve soyad bilgileri mevcuttur.
- Ürünlerin (kişisel bilgisayar, telefon vb.) kodu, adı, fiyatı ve stok miktarlarının saklanması gerekmektedir.
- Ürünlerin kategorileri (bilgisayar, ev elektroniği, kozmetik vb.) mevcuttur.
- Müşterilerin ürün siparişleri saklanarak her bir siparişe bir fatura kesilmesi sağlanmalıdır.
- Her siparişin eşsiz bir sipariş numarası ve sipariş tarihi mevcuttur.
- Sipariş edilen bir ürünün sipariş adedi ve birim fiyatı (kişiye özel indirim v.s. nedeniyle ürün tablosundaki fiyattan farklılık gösterebilir) da kaydedilmelidir.
- Faturaların fatura numarası, tarih ve fatura adresi bilgileri saklanmalıdır. (Toplam fiyat hesaplanabilir ya da saklanabilir).
- Siparişler bir kargo firması tarafından iletilir. Kargo firmasının kodu, adı, adresi bilgileri yer alır. Her kargo firmasında siparişlerden sorumlu bir yetkili yer alır.
- Bir ürünün yalnızca bir kategorisi mevcuttur. Bir kategori çok sayıda ürünün kategorisi olabilir.
- Bir siparişte en az bir ürün bulunur. Ancak çok sayıda ürün de bulunabilir. Bir ürün çok sayıda siparişte yer alabilir.
- Bir müşteri çok sayıda sipariş verebilir. Bir sipariş yalnızca bir müşteri tarafından verilebilir.
- Bir siparişin yalnızca bir faturası olabilir. Bir fatura yalnızca bir siparişin faturası olabilir.
- Bir sipariş ile yalnızca bir satış temsilcisi ilgilenir. Bir satış temsilcisi çok sayıda sipariş ile ilgilenebilir.
- Bir müşteri yalnızca bir ilde yaşayabilir. Bir ilde çok sayıda müşteri yaşayabilir.
- Bir sipariş yalnızca bir kargo firması tarafından iletilir. Bir kargo firması çok sayıda sipariş iletebilir.

# Nesne Yönelimli Analiz ve Tasarım

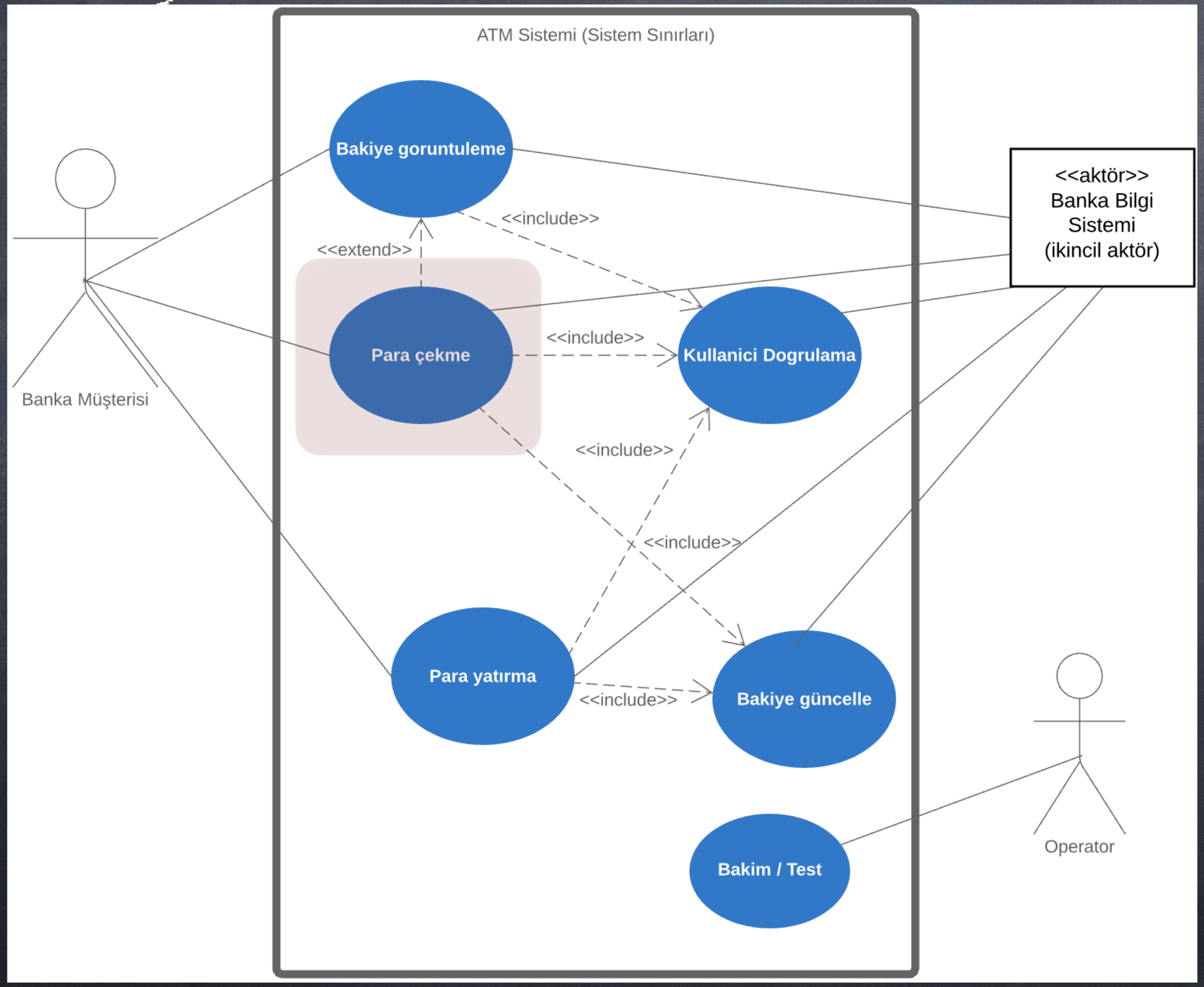
UML Modellerinden Koda Dönüşüm
CRC Kartları
Haberleşme Şeması

# ATM Sistemindeki Nesnelerin Belirlenmesi

* ATM sistemi aşağıdaki donanım bileşenlerinden oluşur:
  * Tuş takımı; ile veri girişi yapılır.
  * Ekran; kullanıcıya mesaj görüntüler.
  * Para bölmesi; para alma ve para verme işleminden sorumludur.
  * Kart bölmesi; banka kartını alır ve doğrular.
* Müsteriler hesaplarından banka bilgi sistemini kullanarak; para çekme, para yatırma, bakiye görüntüleme vb. işlemlerini yapabilmelidirler.
* Banka görevlileri ATM sisteminin bakımını (para doldurma, sıkışan/yutulan kartları alma v.b. ) yapmalıdır.
* Tüm işlemlerde güvenlik gereksinimleri göz önünde bulundurulmalıdır (yetki kontrolü).
* Sistem, Banka Bilgi Sistemiyle çevrimiçi çalışmalıdır. Kullanıcılar Banka Bilgi Sisteminde yer alan hesaplarına erişerek işlemlerini gerçekleştirirler.

Not: Sarı renkliler nesne (sınıf), yeşil renkliler yöntem adaylarıdır.
Aynı cümlede bulunan nesneler bağıntılıdır.

# Para Çekme Kullanım Durumu (Use Case)



# Use Case- "Para Çekme" Olay Akışı (Ana senaryo-başarılı)

1. ATM sistemi Ekrana, müşteriden "kartını kart bölmesine takmasını" isteyen mesaj yazdırır.
2. Kart sahibi kartını kart bölmesine takar.
3. Kart bölmesine kart doğrulamasını yapar.
4. Ekrana şifre girilmesini isteyen mesaj gönderilir.
5. Tuş takımını kullanarak girilen şifreyi alır.
6. Kullanıcı doğrulama ve yetki kontrolü için banka bilgi sistemine istek gönderilir.
7. Banka bilgi sistemi erişim isteğini kabul eder ve gerekli bilgileri (Müşteri hesabi) gönderir.

kullanıcı doğrulama

8. ATM sistemi ana menüyü gösterir ve yapılacak işlemin seçilmesini ister.

işlem seçimi

9. Ekrana, çekilecek tutarlar listelenir (ParaCekme).
10. Tuş takımı ile çekilecek tutar alınır.
11. Kart bölmesi kartı çıkartır.
12. Kart sahibi kartı alır.
13. ATM sistemi parayı ve makbuzu verir.
14. Kart sahibi parayı ve makbuzu alır.

para çekme işlemi

# "Para Çekme" Kullanım Durumu Olay Akışı (Alternatif akışlar)

A1. Kart doğrulanamadı (3)

4. Kart bölmesi kartı çıkartır.

5. İşlem sonlandırılır.

A2. Yanlış şifre (6)

8. 3 den az kez yanlış ise yeniden gir.

8. 3 kez yanlış girilmiş ise kart yutulur.

9. işlem sonlandırılır.

A3. Yeterli para yok (9)

10. ATM de yeterli bakiye yoksa yeni tutar girmesi istenir.

11. Hesapta yeterli bakiye yoksa yeni tutar girmesi istenir.

A4. Banka kartı alınmadı(10)

11. 30 sn. boyunca alınmadı ise kart yutulur

12. Gerekli bilgilendirme yapılır

13. İşlem sonlandırılır

A5. Para alınmadı (12)

13. 30 sn. boyunca alınmadı ise para yutulur

14. Gönderilen ve yutulan para karşılaştırılır.

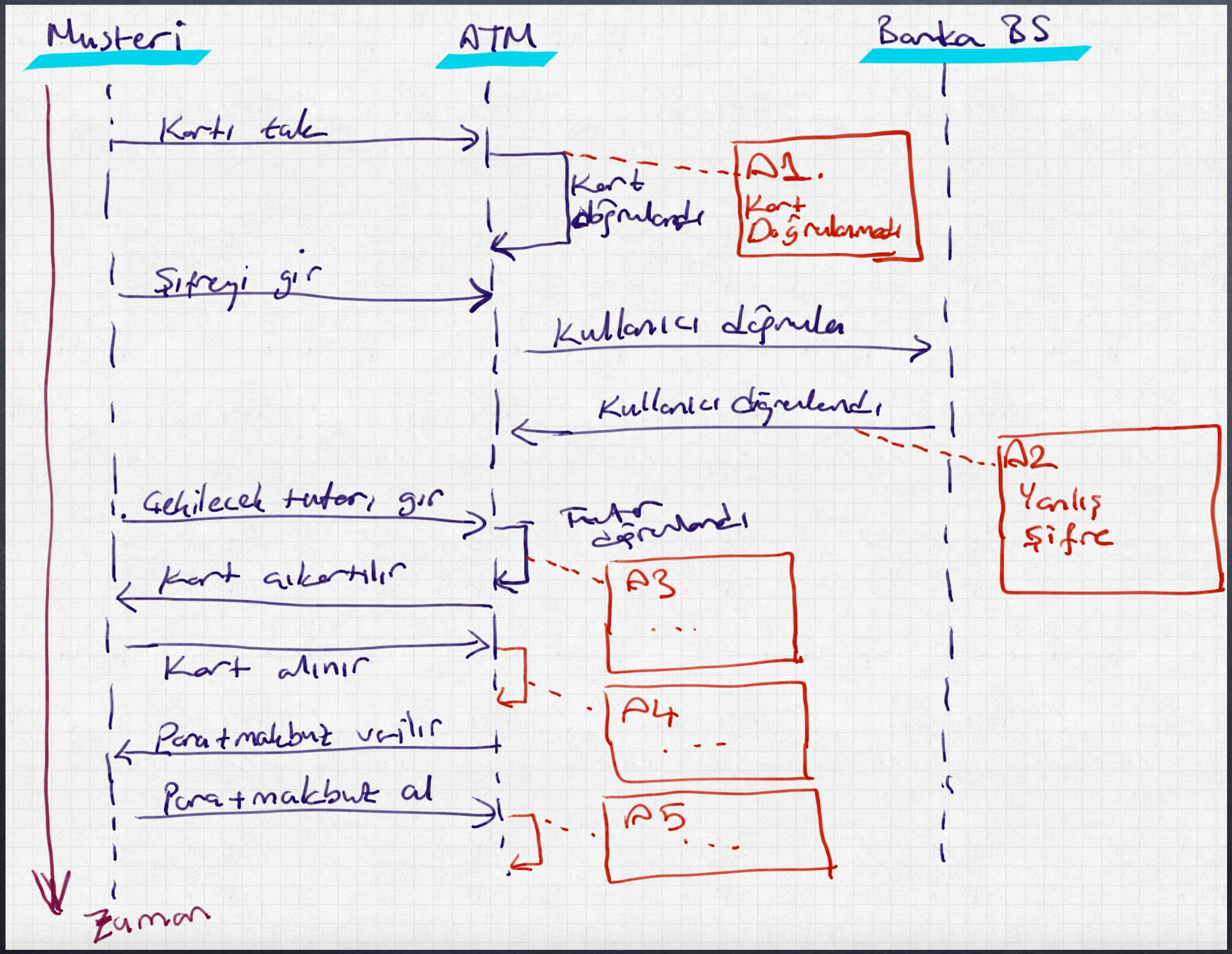
15. Gerekli ise hesap güncellenmesi yapılır

12. Gerekli bilgilendirme yapılır

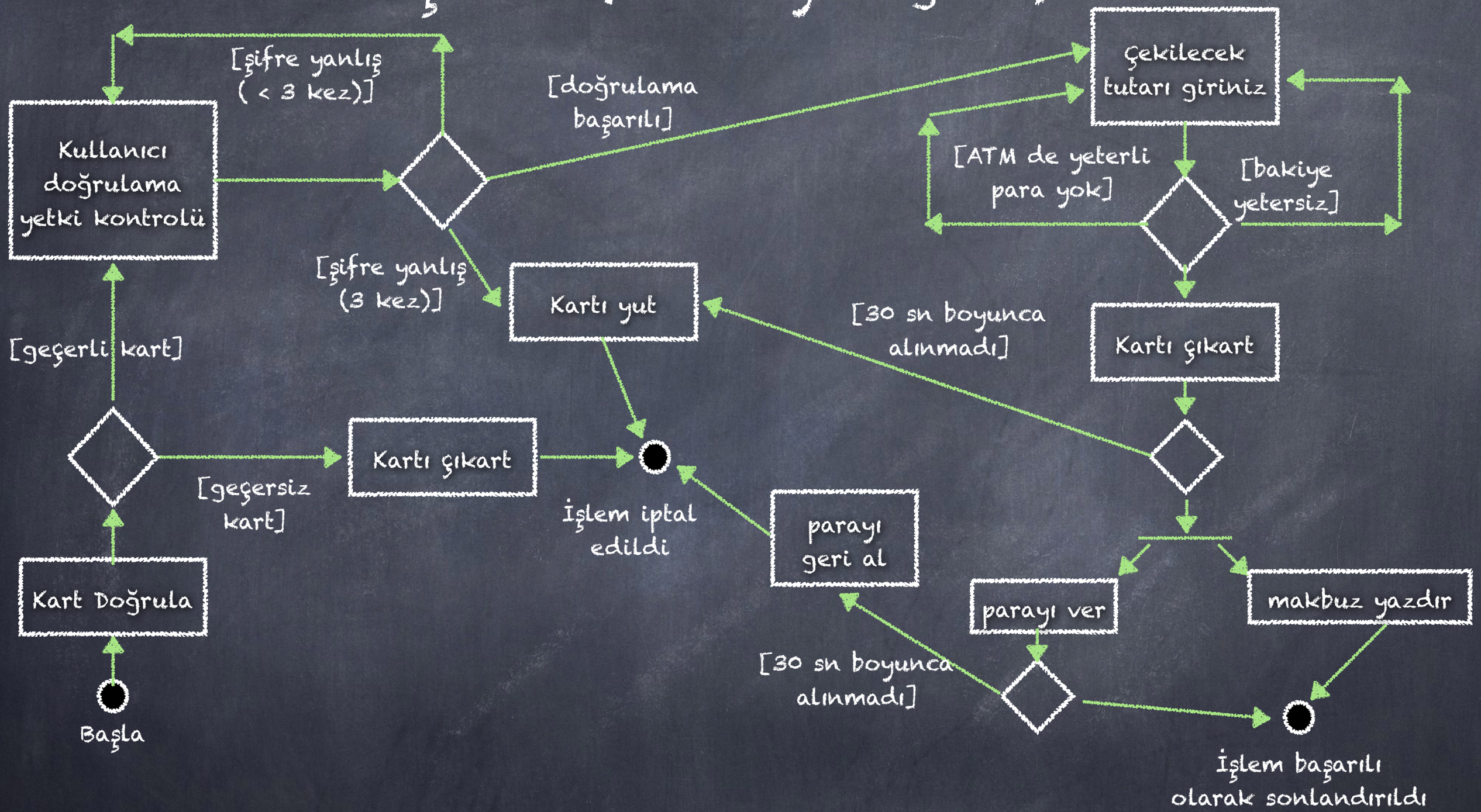
13. İşlem sonlandırılır

...

# Para Çekme Kullanım Durumu İçin ATM Sıralama Şeması



# Para Çekme Kullanım Durumu İçin ATM Etkinlik Şeması (Activitiy Diagram)



# Class Responsibility Collaboration(CRC) Kartları

Nesnelerin sorumluluklarını ve bu sorumlulukları yerine getirmek için hangi nesnelerle işbirliği içerisinde olduklarını gösterir.

Sistem içerisindeki tüm nesneler için CRC kartı oluşturulabilir.

* Nesnelerin sorumluluklarını görmemizi sağlar (SR ilkesine uygun mu?)
* Kodlamaya, sonradan değişikliğe yardımcı olur.
* Nesne geliştirecek olanların, hangi servislere ihtiyaç olduğunu anlamasını kolaylaştırır.
* Sağlanacak servisler için hangi nesnelere ve ilgili servislerine ihtiyaç duyulduğu anlaşılır.

# Class Responsibility Collaboration(CRC) Kartları

Sistem içerisindeki tüm nesneler için CRC kartı oluşturulabilir. (Şekilde ATM sınıfına ait CRC kartı yer almaktadır. Diğer sınıflar için de oluşturabiliriz.)

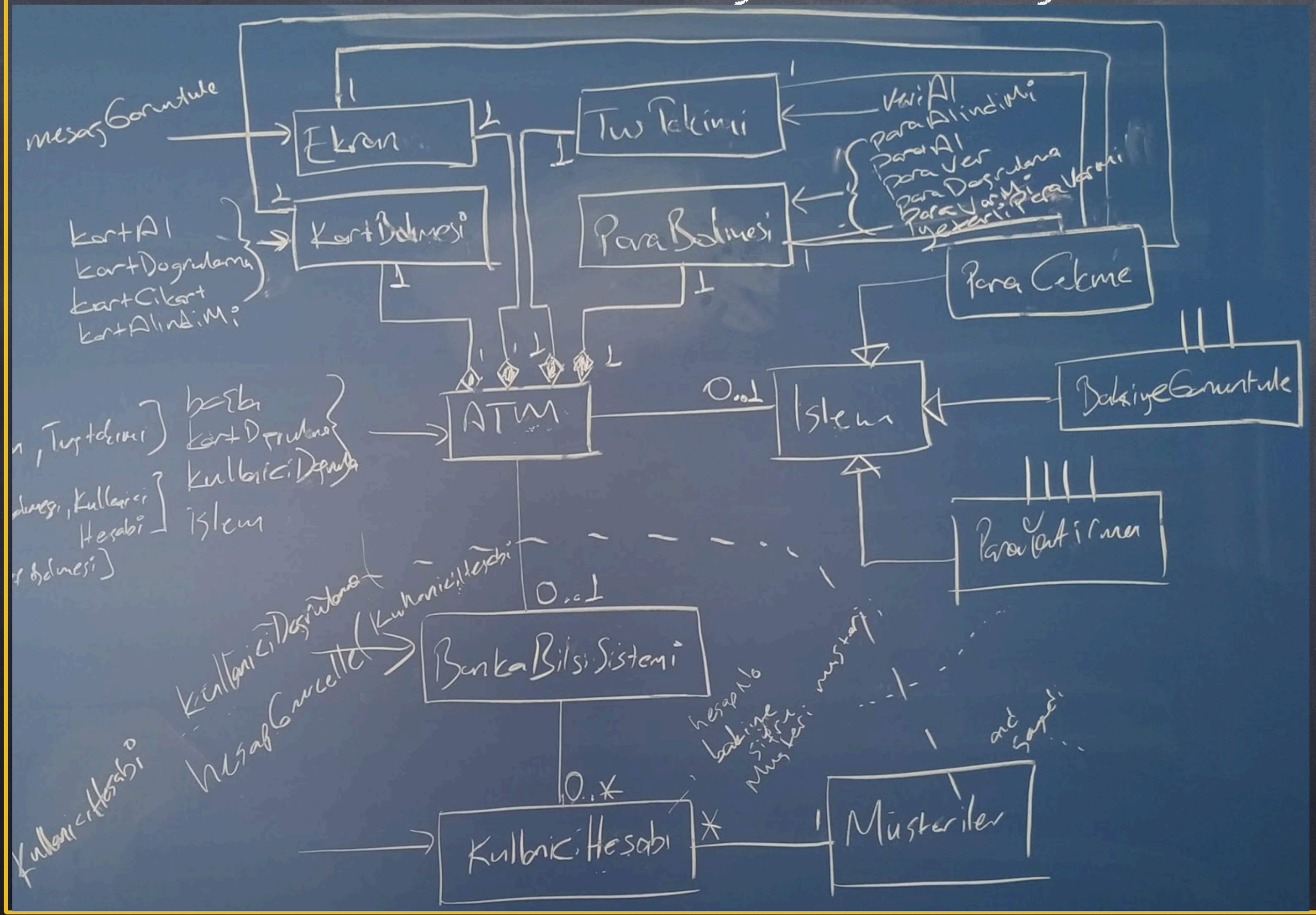
| ATM (Banka müşterilerine finansal servis sağlar) | |
|---|---|
| **Sorumluluk** | **İşbirliği Yapılan Sınıf** |
| uygulamanın başlatılması | kendisi |
| mesaj yazdır (mesajGoruntule()) | Ekran |
| kullanıcı verisi al (veriAl()) | TusTakimi |
| işlem (transaction) yap (islemYap()) | Islem ( ParaCekme, ParaYatirma, BakiyeGoruntuleme ) |
| kullanıcı Doğrula | BankaBilgiSistemi, MusteriHesabı |
| kart işlemleri | KartBolmesi |
| para işlemleri | ParaBolmesi |
| ana menü ve işlem seçimi | kendisi |

# Class Responsibility Collaboration(CRC) Kartları

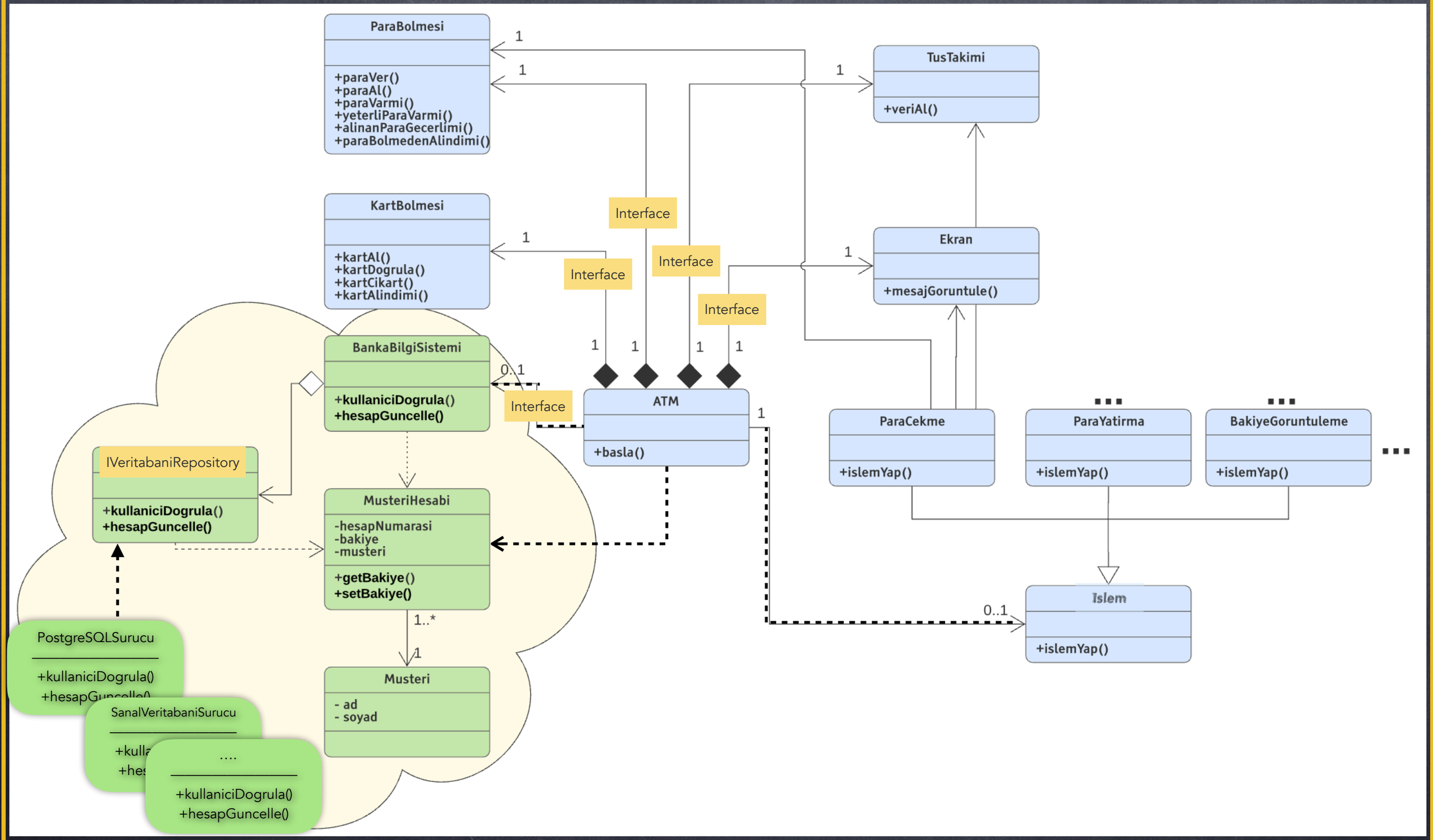
Sistem içerisindeki tüm nesneler için CRC kartı oluşturulabilir. (Şekilde ParaCekme sınıfı CRC kartı yer almaktadır. Diğer sınıflar için de

| ParaCekme (Para çekme işlemini gerçekleştirir.) | |
|---|---|
| **Sorumluluk** | **İşbirliği Yapılan Sınıf** |
| para çekme işlemi | kendisi |
| mesaj yazdır | Ekran |
| kullanıcı verisi al | TusTakimi |
| hesap güncelleme | BankaBilgiSistemi, MusteriHesabi |
| para işlemleri | ParaBolmesi |

# ATM Sisteminin Analiz Aşaması Sınıf Şeması



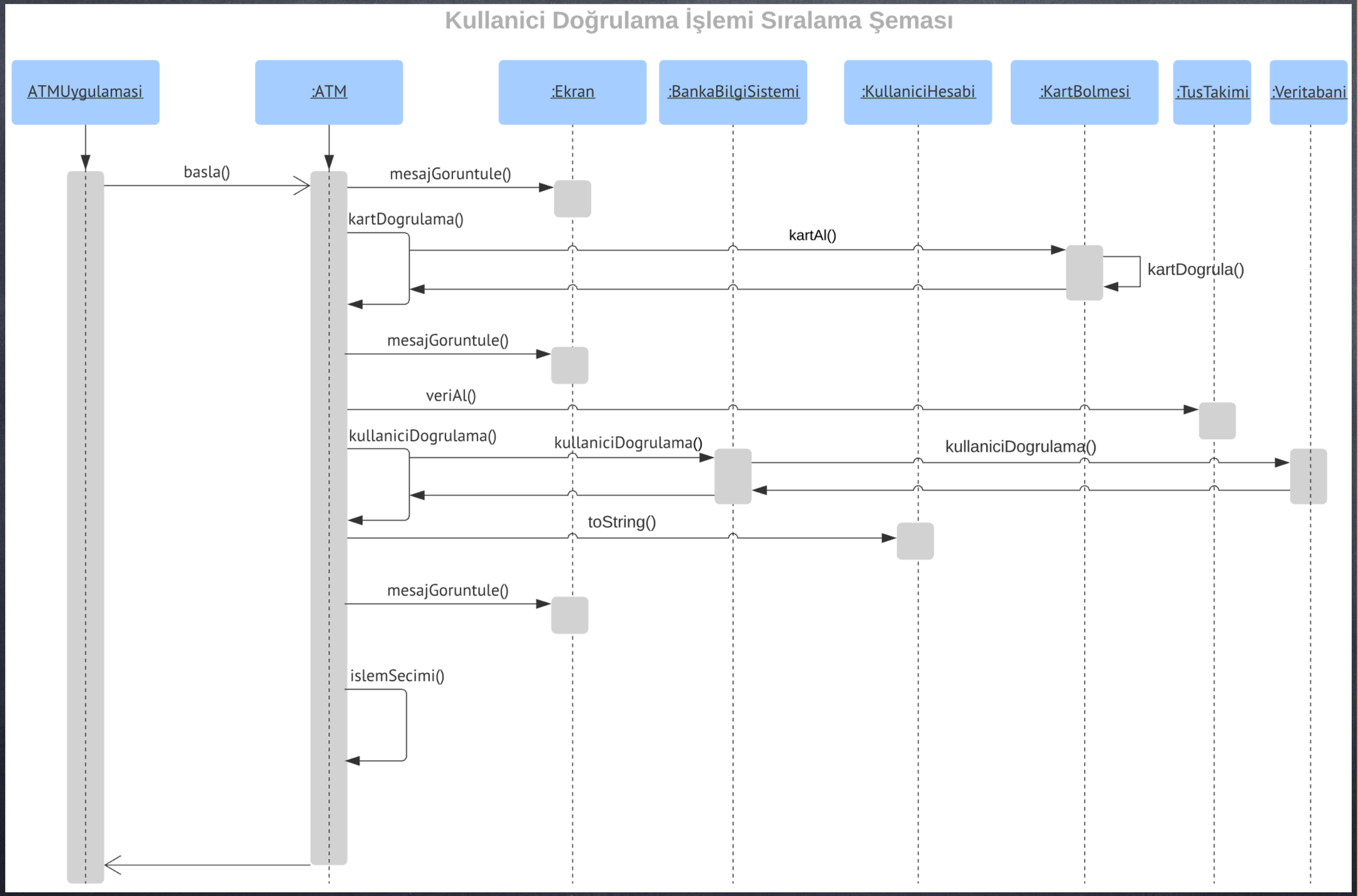
# ATM Sisteminin Analiz Aşaması Sınıf Şeması




ATM Sistemi

https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders6

# ATM Sıralama Şeması (Kullanıcı Doğrulama)




zaman

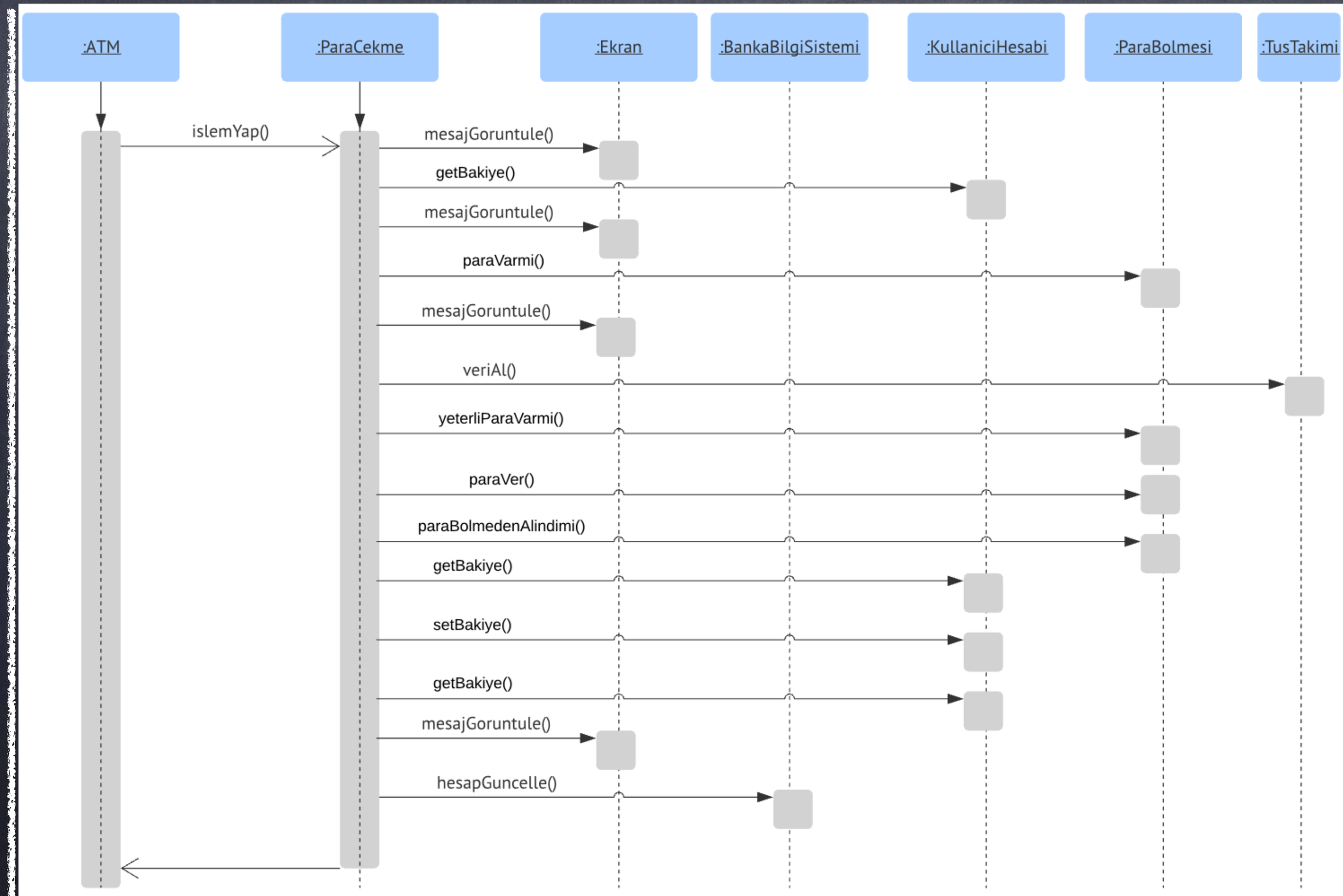
# ATM Sıralama Şeması (Kullanıcı Doğrulama)

```java
public void basla() {
    ekran.mesajGoruntule("Lütfen kartınızı bölmeye takınız...");
    Araclar.bekle();
    int hesapNumarasi=this.kartDogrulama();
    if(hesapNumarasi!=0){
        ekran.mesajGoruntule("şifreniz");
        int sifre= tusTakimi.veriAl();
        //IBankaBilgiSistemi bankaBilgiSistemi=new BankaBilgiSistemi(new PostgreSQLSurucu());
        IBankaBilgiSistemi bankaBilgiSistemi=new BankaBilgiSistemi(new SanalVeritabaniSurucu());
        MusteriHesabi musteriHesabi = this.kullaniciDogrula(hesapNumarasi, sifre, bankaBilgiSistemi);
        //MusteriHesabi musteriHesabi = bankaBilgiSistemi.kullaniciDogrula(hesapNumarasi, sifre);
        if (musteriHesabi != null){
            ekran.mesajGoruntule("Kullanıcı doğrulama işlemi başarılı...:"+ musteriHesabi);
            islemSecimi(bankaBilgiSistemi, musteriHesabi);
        } else{
            ekran.mesajGoruntule("hesabınız doğrulanamadı");
            kartBolmesi.kartCikart();
        }
    } else{
        ekran.mesajGoruntule("Kartınız doğrulanamadı");
        kartBolmesi.kartCikart();
    }
}
```

İstenenler:

1. Şifrenin yanlış girilme durumunu da göz önüne alacak düzenlemeyi yapınız (Etkinlik şemasına bakınız).
2. Kart bölmesi tarafından okunan hesap numarası banka bilgi sistemi tarafından doğrulansın.

# ATM Sıralama Şeması (Para Çekme)




zaman

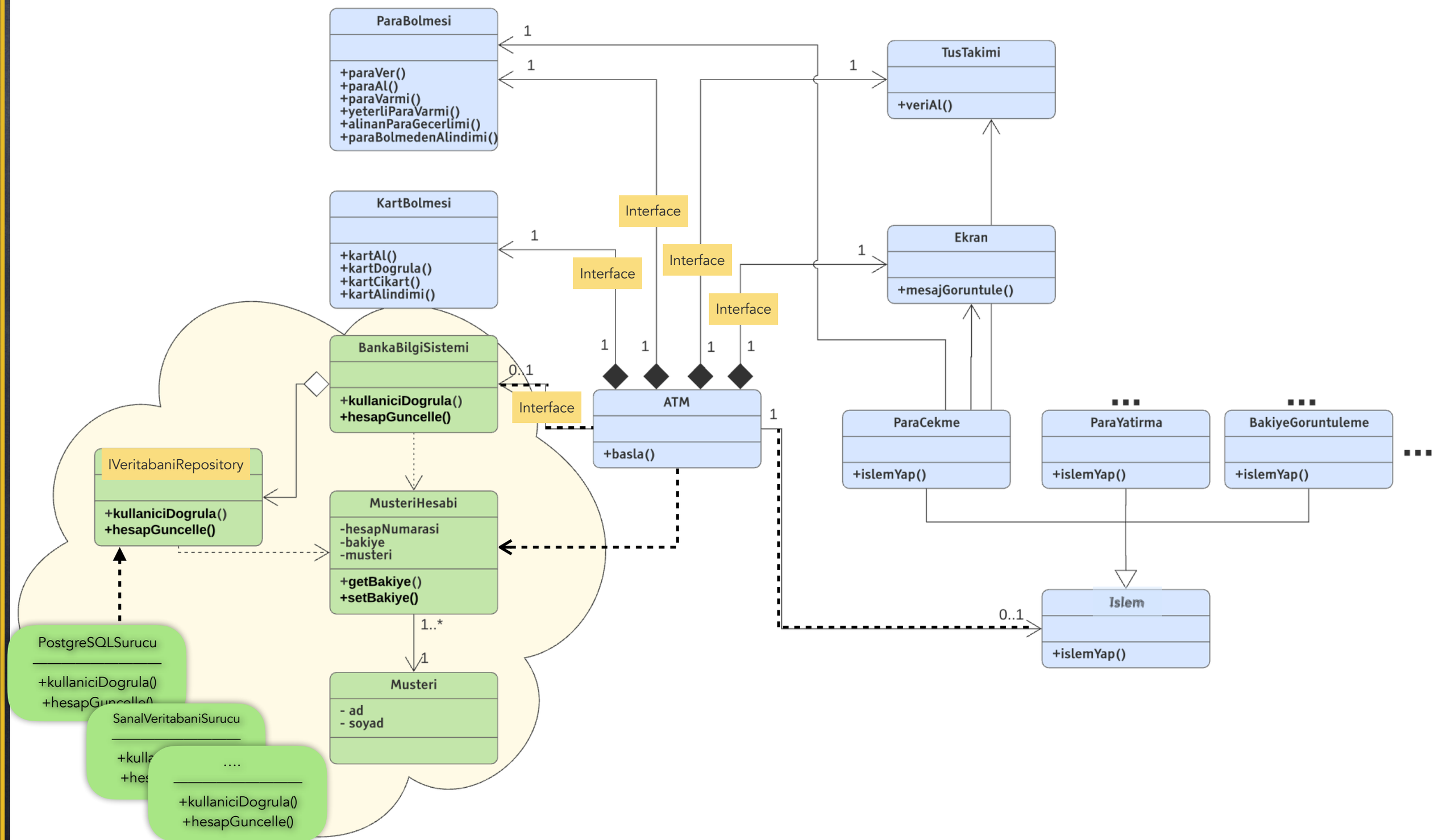
# ATM Sıralama Şemasından (Para Çekme) Koda Dönüşüm

```java
@Override
public void islemYap() {
    ekran.mesajGoruntule("para çekme işlemini seçtiniz");
    ekran.mesajGoruntule("Bakiyeniz:"+ musteriHesabi.getBakiye());
    if(paraBolmesi.paraVarmi()) {
        ekran.mesajGoruntule("Tutarı giriniz");
        int cekilecekTutar = tusTakimi.veriAl();
        if (paraBolmesi.yeterliParaVarmi(cekilecekTutar)) {
            paraBolmesi.paraVer(cekilecekTutar);
            if(paraBolmesi.paraBolmedenAlindimi()) {
                musteriHesabi.setBakiye(musteriHesabi.getBakiye() - cekilecekTutar);
                ekran.mesajGoruntule("son bakiyeniz" + musteriHesabi.getBakiye());
                bankaBilgiSistemi.hesapGuncelle(musteriHesabi);
            }else{
                ekran.mesajGoruntule("Para bölmeden alınmadı");
                paraBolmesi.paraAl(cekilecekTutar);
            }
        } else
            ekran.mesajGoruntule("Bankamatikte yeterli para yok...");
    } else{
        ekran.mesajGoruntule("Bankamatikte para yok...");
        Araclar.bekle(2000);
    }
}
```

İstenenler:

1. ATM de para yok ise ya da müşteri bakiyesi sıfır ise işlem hiç başlatılmasın.
2. Çekilmek istenen tutar bakiyeden ya da para bölmesindeki miktardan fazla ise yeni tutar girilmesi istensin.
3. Çekilecek tutarların listesi gelsin ve kullanıcı seçebilsin.

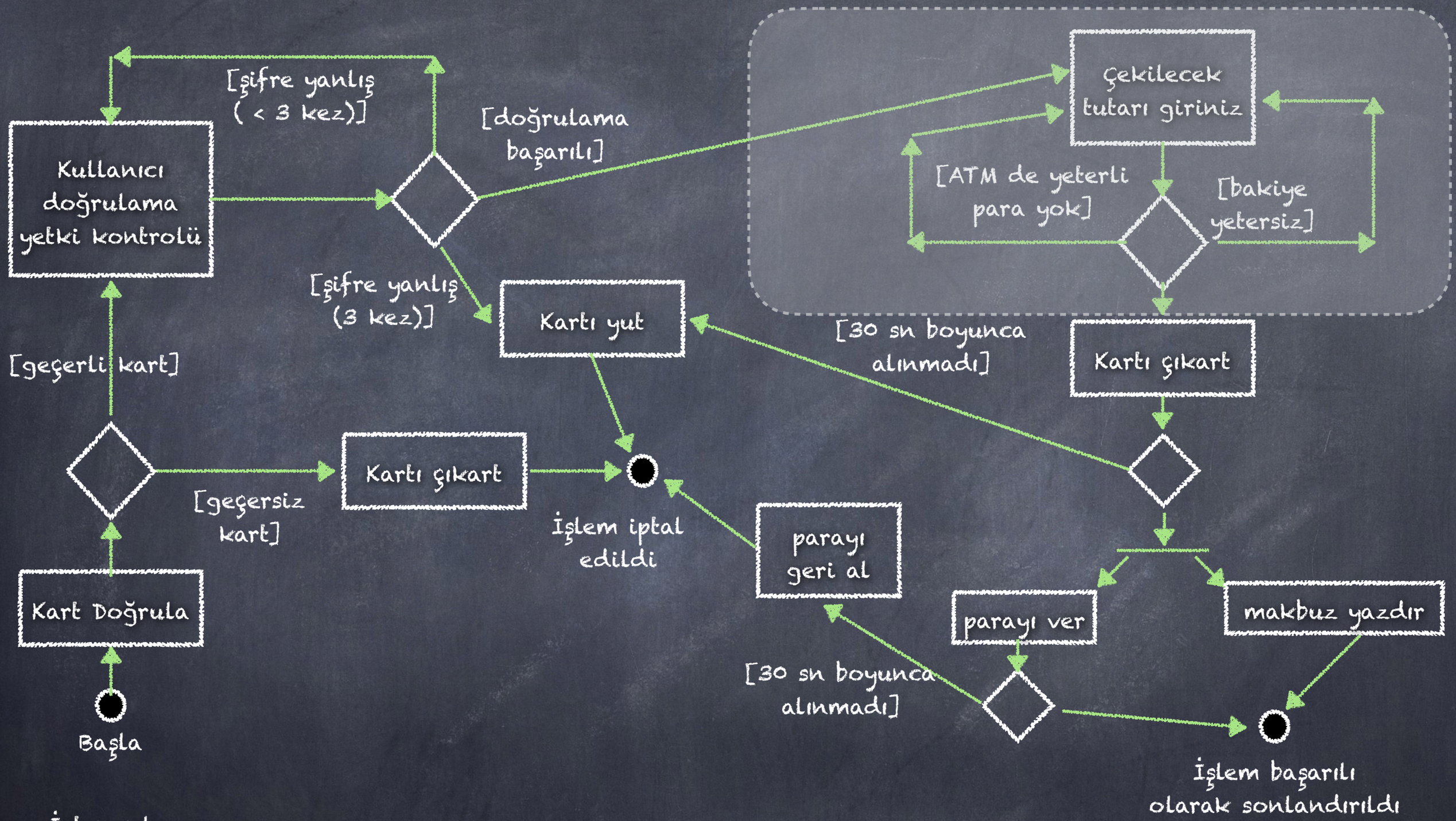
# ATM Sisteminin Analiz Aşaması Sınıf Şeması




İstenenler:

1. Para Yatırma ve Bakiye Görüntüleme kullanım durumları için sıralama ve etkinlik şemalarını oluşturunuz.
2. Para transfer işlemini ekleyiniz.

# Para Çekme Etkinlik Şeması (Değişiklik)




İstenenler:

1. Çekilmek istenen tutar bakiyeden ya da para bölmesindeki miktardan fazla ise yeni tutar girilmesi istensin.
2. Çekilecek tutarların listesi gelsin ve kullanıcı seçebilsin.
3. Makbuz yazdırma işlemi de ele alansın.

# UML - Haberleşme(İletişim) Şeması (Communication(collaboration) Diagram)

* Davranışsal (dinamik) gösterim şekillerinden biridir.
* Nesneler ya da roller (lifeline) arası etkileşimi ve aralarındaki mesajlaşmayı gösterir.
* Sıralama şemasına benzer bilgi sunar. Sıralama şemasında mesajların zamana göre sıralaması gösterilirken, haberleşme şemasında nesnelerin etkileşimi ve masajlaşmasına odaklanılır.

# Örnek: Haberleşme Şeması

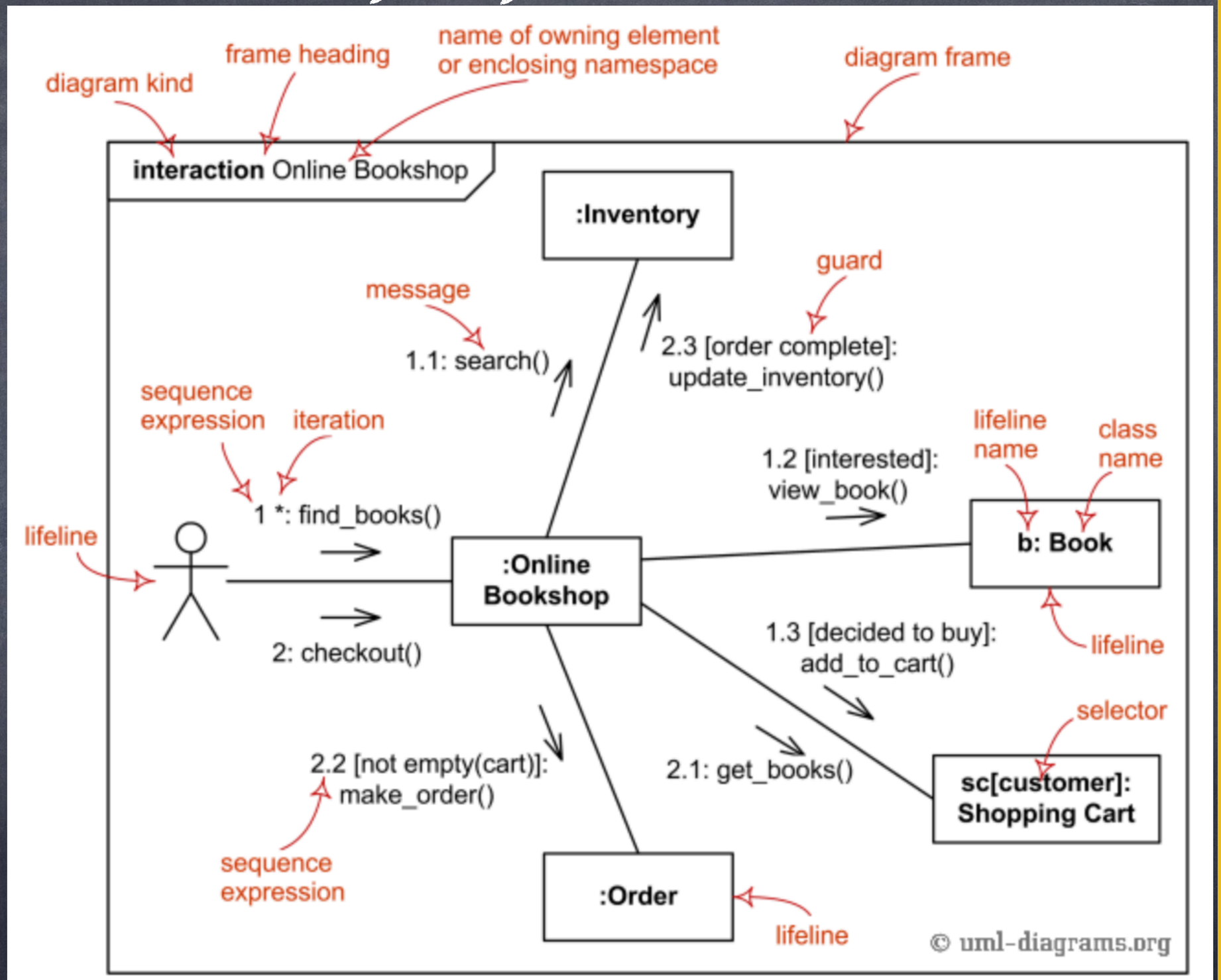
1. Müşteri, alınacak kitabın bulunması için, OnlineBookshop tipli nesnenin find_books() yöntemini çağırır.

1.1. OnlineBookshop nesnesi, kitabın aranması için Inventory nesnesinin search() yöntemini çağırır.

1.2. OnlineBookshop nesnesi, bulunan kitabın ayrıntıları için b nesnesinin view_book() yöntemini çağırır.

1.3. Alınması kararlaştırılan kitabın alışveriş kartına eklenmesi için, OnlineBookshop nesnesi, sc[customer] nesnesinin add_to_cart() yöntemini çağırır.

...




*The major elements of UML communication diagram.*

https://www.uml-diagrams.org/communication-diagrams.html

Alıştırma çalışması: ATM sistemi Para Çekme işleminin haberleşme şemasını oluşturunuz.

# Nesne Yönelimli Analiz ve Tasarım

## Sistem Tasarımı

Tasarım Hedeflerinin Belirlenmesi
Sistemin alt bileşenlerine ayrıştırılması
Yazılım mimarileri

# Nesne Yönelimli Tasarım- Sistem Tasarımı

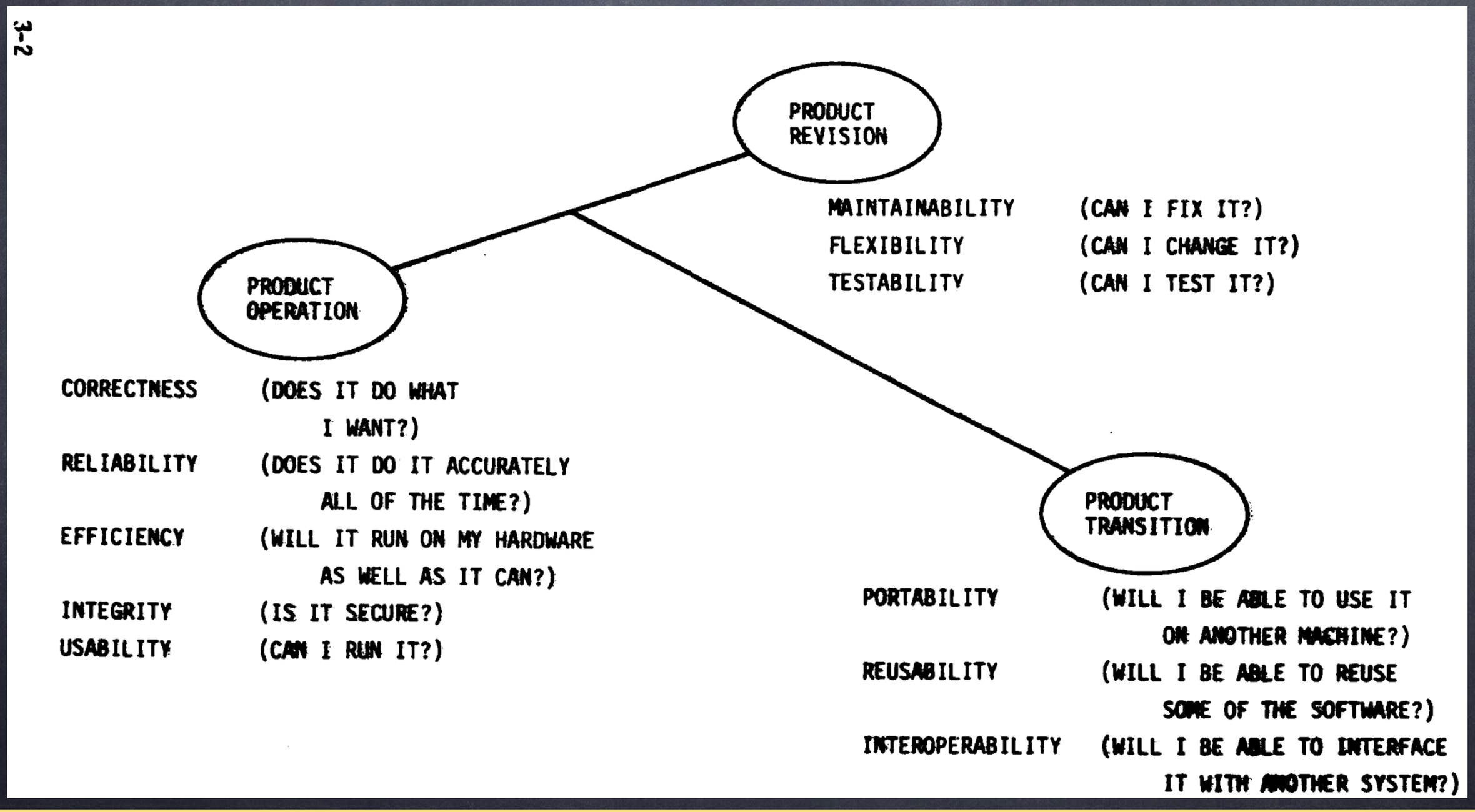
* Analiz aşaması uygulama alanına (application domain) odaklanır.
* Uygulama alanı gerçekleştirilecek yazılımın çalışacağı ortamı ifade eder.
* Tasarım aşaması Çözüm alanına (solution domain) odaklanır.
* Tasarım aşaması, sistem tasarımı ve nesne tasarımı olarak iki bölüme ayrılabilir.
* Sistem tasarımı kapsamında sistem alt bileşenlerine ayrılır. Sistemle ilgili tasarım hedefleri belirlenir ve kısıtlar ihlal edilmeden bu hedeflere ulaşmak için gerekli mimari oluşturulur.

# Nesne Yönelimli Tasarım- Sistem Tasarımı

* Analiz aşaması:
  * gereksinimler, kısıtlar belirlenir
  * kullanım durumları tanılanır
  * analiz sınıf şeması/ nesne modeli, dinamik modeller ( sıralama şeması, etkinlik şeması vb.) oluşturulur
* Sistem tasarım aşaması:
  * tasarım hedefleri (kalite ölçütleri) belirlenir
  * sistemin alt bileşenlerine ayrıştırılır
  * tasarım hedeflerine ulaşmayı sağlayacak yazılım mimarisi (mimari stiller), veri modelleri, erişim denetimi yöntemleri vb. belirlenir.

# 1. Tasarım Hedeflerinin Belirlenmesi / Yazılımların Kalitesi (McCall Kalite Üçgeni)

Bir yazılımın kalitesine etki eden faktörler üç sınıfa ayrılır.




McCall, J.A., Richards, P.K. and Walters, G.F. (1977) Factors in Software Quality. RADC TR-77-369, Rome Air Development Center, Rome.

# Kalite Ölçütlerinin Birbirlerine Olumsuz Etkileri

* Integrity <-> Efficiency
  * Güvenlik denetimi ek kontrol ve fazladan yer anlamına gelir
  * bankacılık uygulaması
* Portability <-> Efficiency
  * tüm platformları desteklemesi için eklenen denetimler fazladan kaynak kullanımı ve gecikme anlamına gelir
* Reusability (Flexibility, Maintainability) <-> Efficiency
  * program daha küçük parçalar halinde yazılırsa hız azalır.
  * gömülü sistemler, gerçek zamanlı sistemler...
* Diğerleri neler olabilir?

# 2. Sistemin alt bileşenlerine ayrıştırılması

* Karmaşıklığı azaltmak için yazılım sistemi alt bileşenlerine (alt sistem) ayrılır.
* Alt sistemler, birbirleriyle yakından ilişkili; sınıflar, ilişkiler ve kısıtlardan oluşur.
* Alt sistemler sayesinde;
    * problem çözümü kolaylaşır
    * çok sayıda kişinin aynı projede aynı anda çalışabilmesi sağlanır
    * sistemin anlaşılması, anlatılması kolaylaşır
    * bakım kolaylaşır
    * modülerlik, kod tekrar kullanımı artar...
* Analiz aşamasındaki alt bileşenler
    * sınıf
    * paket (UML paket şeması)
* Tasarım aşamasındaki alt bileşenler için "UML Component" şeması kullanılabilir.

# 2. Sistemin alt bileşenlerine ayrıştırılması

## İyi bir tasarım için

**High Coherence**

Modüller tek ve özel bir işi, mükemmel bir şekilde yapmalı. Alt sistemler içerisindeki sınıflar benzer işi yapmalı ve ilgili olmalı. "Cohesion" Bunun ölçüsüdür .

**Low Coupling**

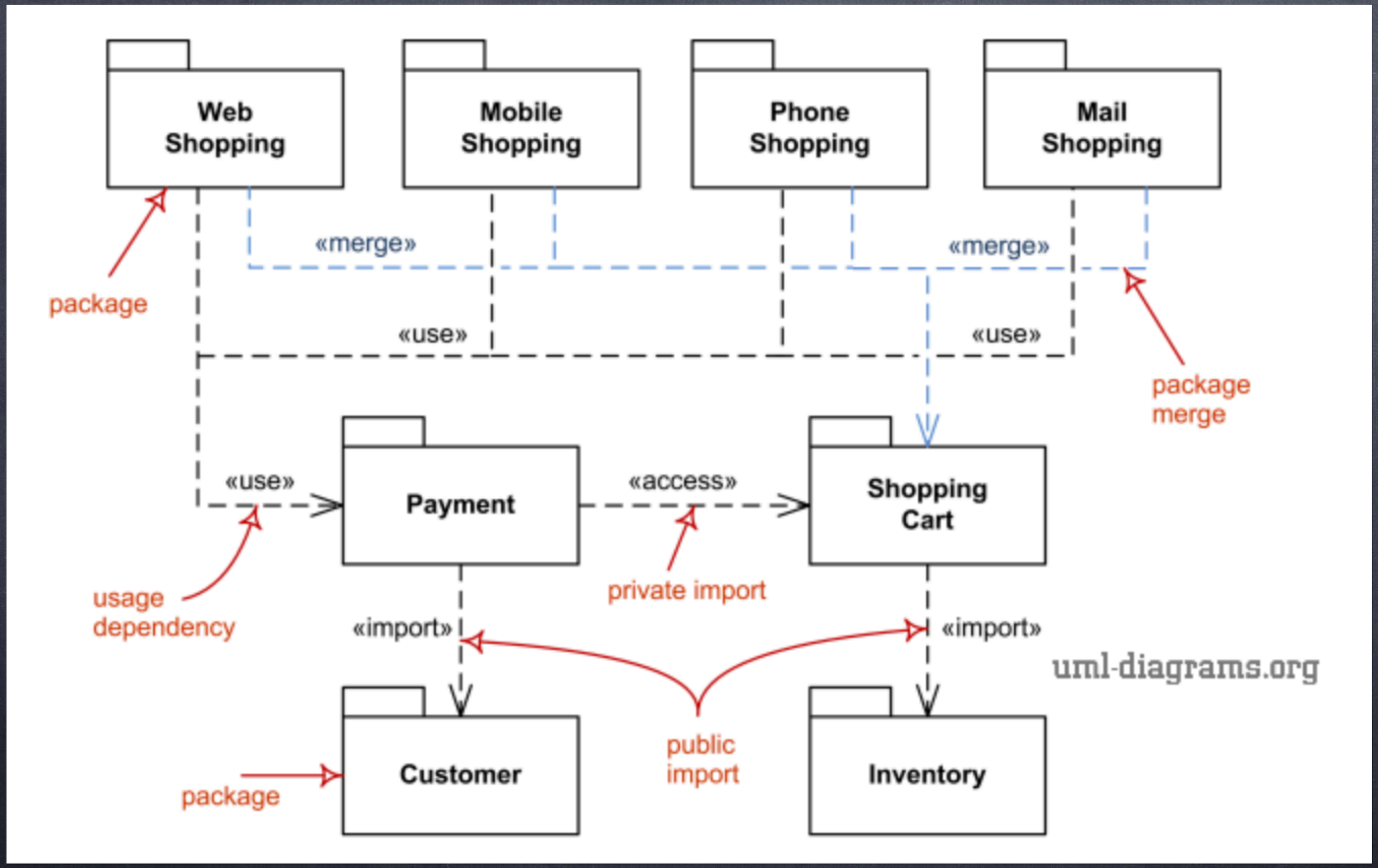
Aradaki bağıntılar ne kadar fazla olursa nesne içerisinde yapılacak köklü değişiklik ona bağlı olan modülleride etkileyecektir.

## 2. Sistemin alt bileşenlerine ayrıştırılması- UML Paket Diagramı

* Yapısal gösterim şekillerindendir.
* Paketleri ve paketler arası ilişkileri göstermek için kullanılır.
* Paketler içerisinde; sınıf, paket, bileşen (component), kısıtlar ve bağımlılıklar olabilir.
* Java platformunda; package ifadesi paket oluşturmak için, import ifadesi ise kullanılacak paketleri içe aktarmak için kullanılır.
  * import java.net.Socket;
  * import java.io.PrintWriter;
  * MusteriliSiparis.java
    * package cc.ders7.lab;
    * import cc.ders5.Musteri;
    * import cc.ders6.siparis.Siparis;

```
package cc.ders5;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Uygulama {

    public static void main(String [] args){
```

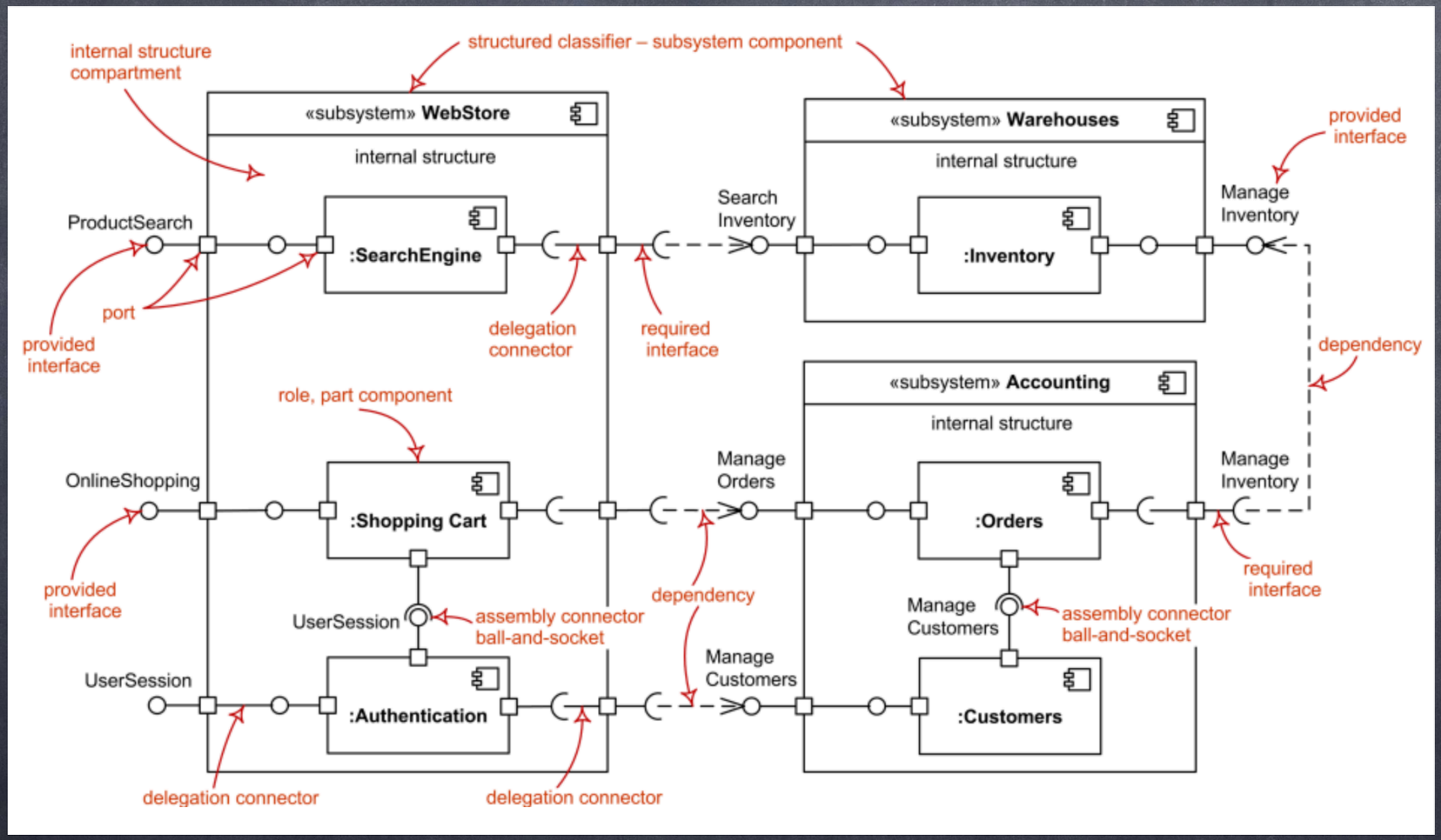
# 2. Sistemin alt bileşenlerine ayrıştırılması- UML Paket Diagramı




https://www.uml-diagrams.org/package-diagrams-overview.html

# 2. Sistemin alt bileşenlerine ayrıştırılması- UML "Component" Diagramı

* Yapısal gösterim şekillerindendir. "Component" (Bileşen), sistemin diğer bölümleriyle haberleşmeyi sağlayan arayüzlere (API-Application Programming Interface) sahip alt sistemler ya da sistemlerdir. Ortak amaca hizmet eden sınıfların bir araya getirilmesiyle oluşur.




* Şekilde bağıntılı üç alt sistem (WebStore(Web mağazası), Warehouse(depo, ambar), Accounting(Muhasebe)) bulunmaktadır. Alt sistemler içerisinde başka alt sistemlerde bulunmaktadır.
* "SearchEngine" sağladığı "ProductSearch" arayüzüyle ürünlerin aranmasını sağlar. Bu işlem için "Inventory" bileşeninin sağladığı "SearchInventory" arayüzünü kullanır.
* "Shopping Cart" bileşeni çevrimiçi alış veriş arayüzü sağlar. Alış veriş için kullanıcının oturum açması gerekir. Siparişlerin yönetimi için "Orders" bileşeninin "Manage Orders" arayüzünü kullanır.
* ...

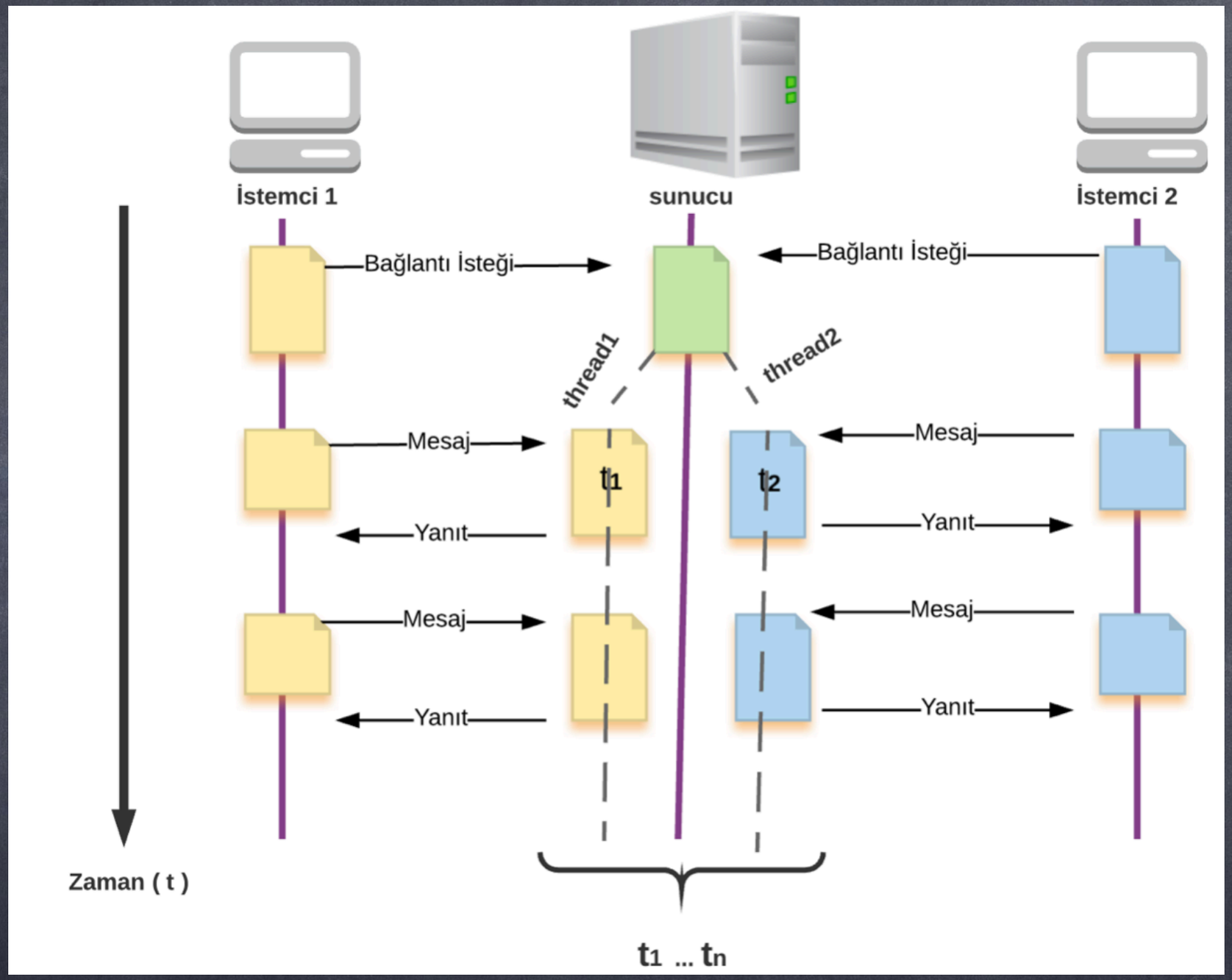
https://www.uml-diagrams.org/component-diagrams.html

# 3. Yazılım mimarileri (mimari stiller)

* İstemci/Sunucu (Client/Server)
* Üç Katmanlı Web Mimarisi (Three-tier Web Architecture)
* Servis Yönelimli Mimari (Service-Oriented Architecture (SOA))
* Model/Görünüm/Denetleyici (Model/View/Controller(MVC))
* Peer-To-Peer

https://www.uml-diagrams.org/package-diagrams-overview.html

# 3. Yazılım mimarileri (İstemci/Sunucu)

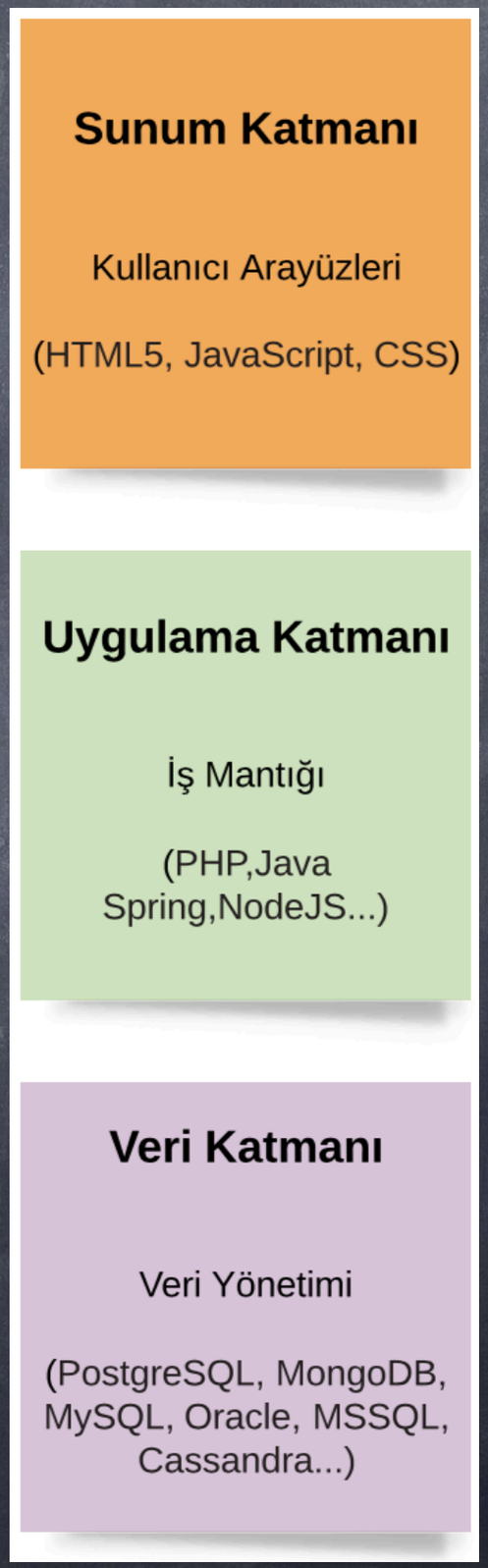



İstemci/Sunucu Örneği

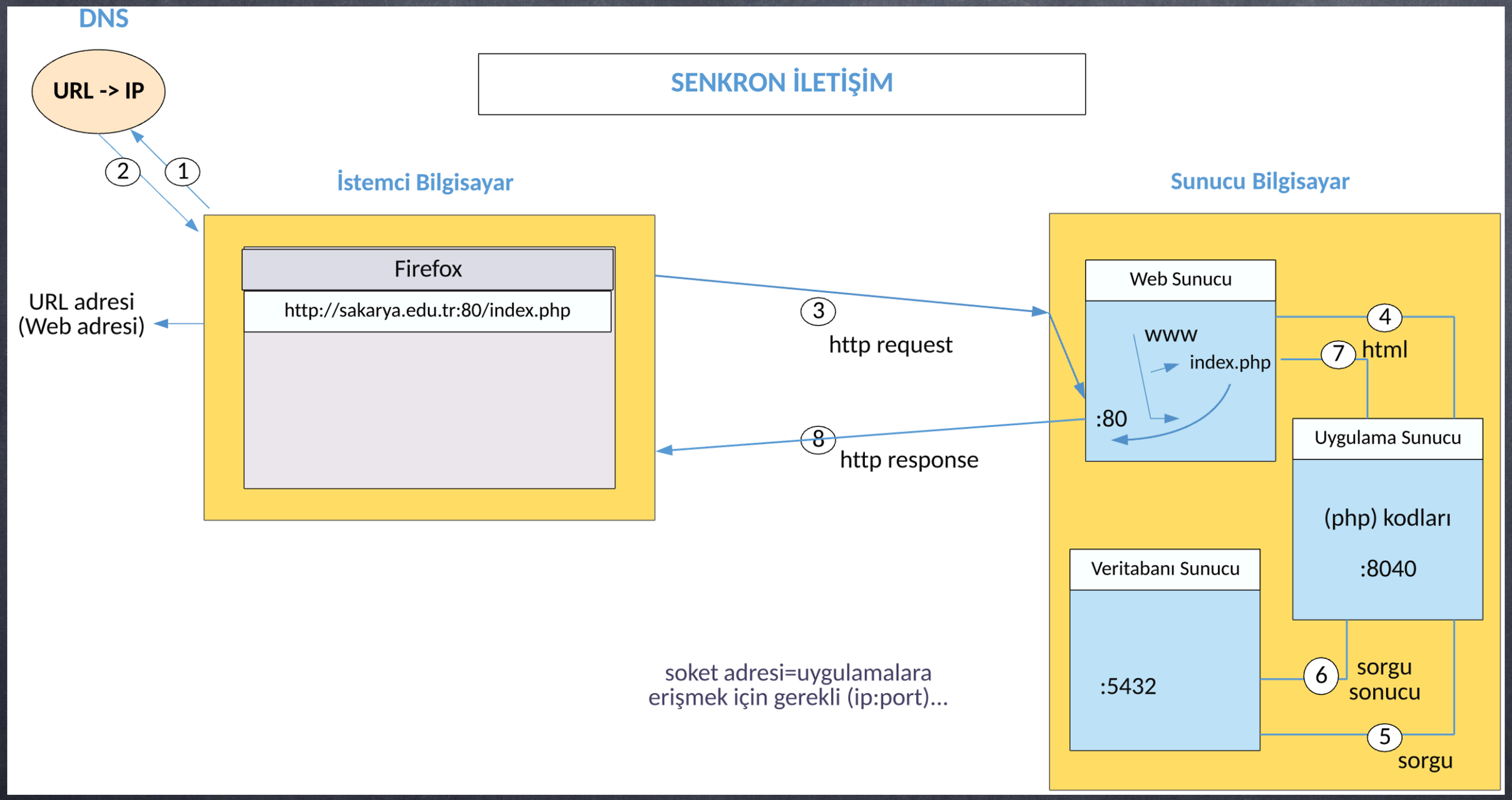
https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders8/IstemciSunucuMimarisi

# 3. Yazılım mimarileri (Üç Katmanlı Web Mimarisi (Three-tier Web Architecture))

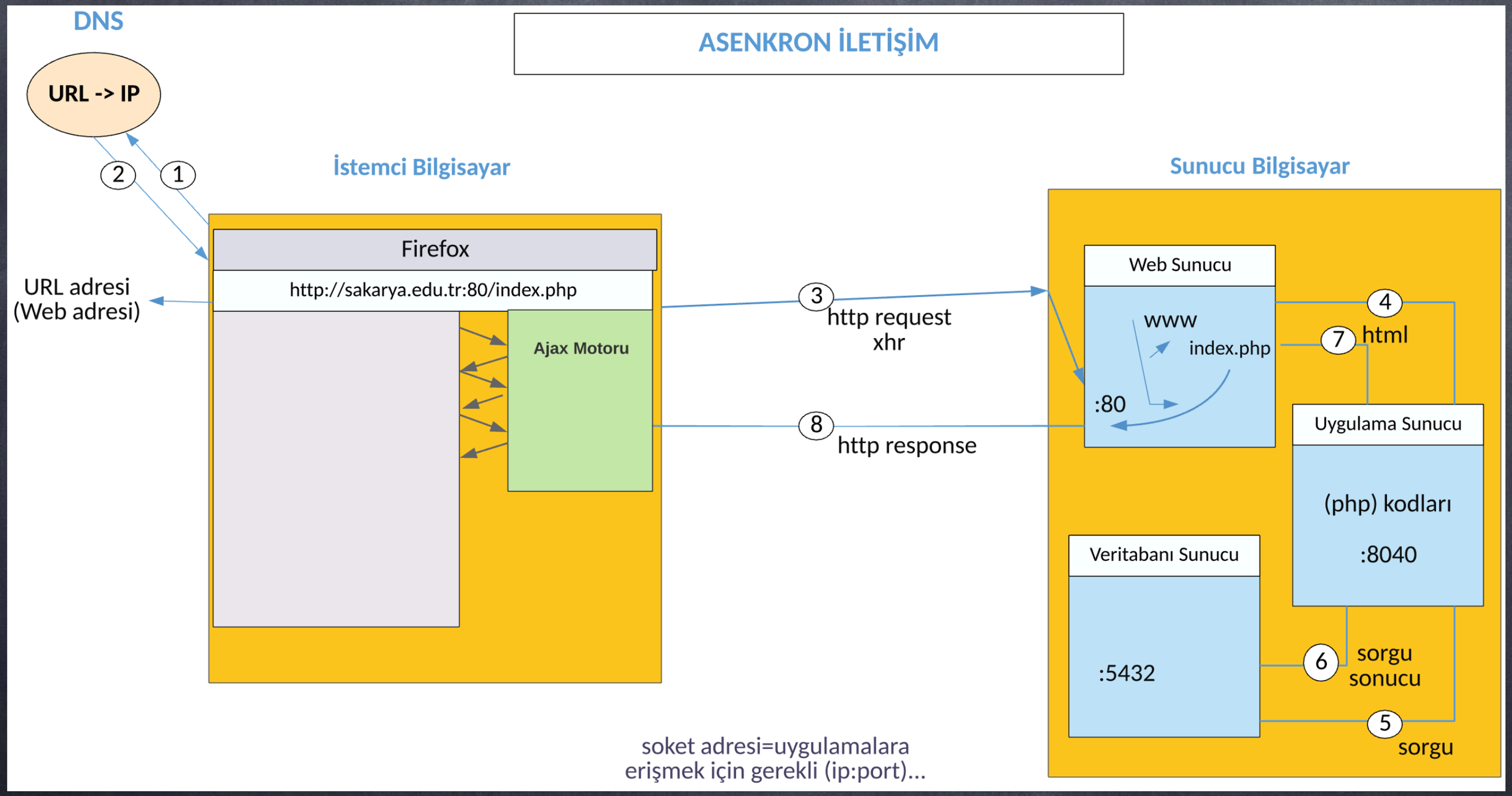
* Sunum, iş mantığı ve veri yönetimi fonksiyonlarının fiziksel olarak ayrıldığı İstemci/Sunucu mimarisidir.
* Modülerlik, başarım, bölümler birbirlerinden bağımsız geliştirilebilir



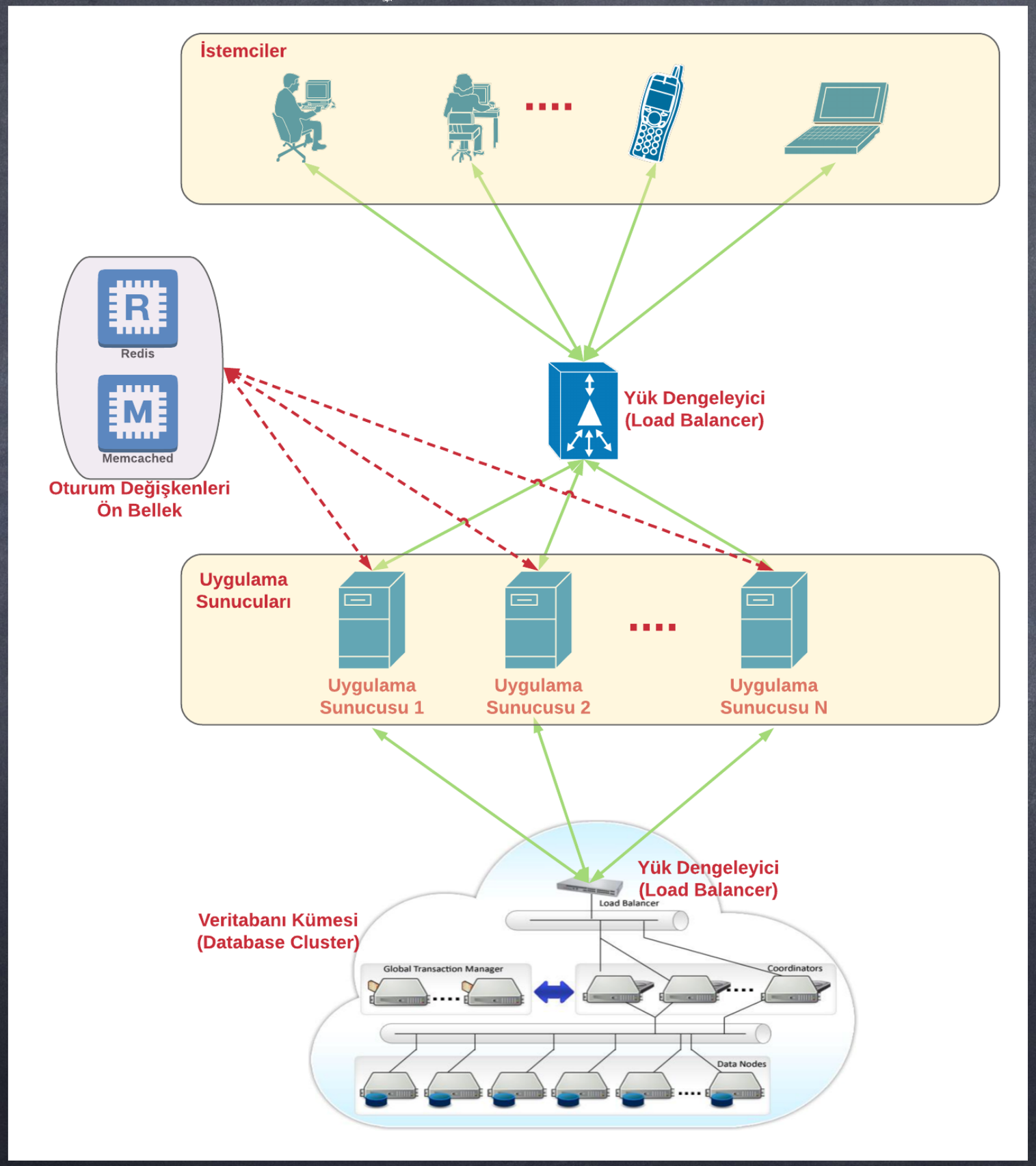
# 3. Yazılım mimarileri (İstemci/Sunucu(Web Uygulama Mimarisi))



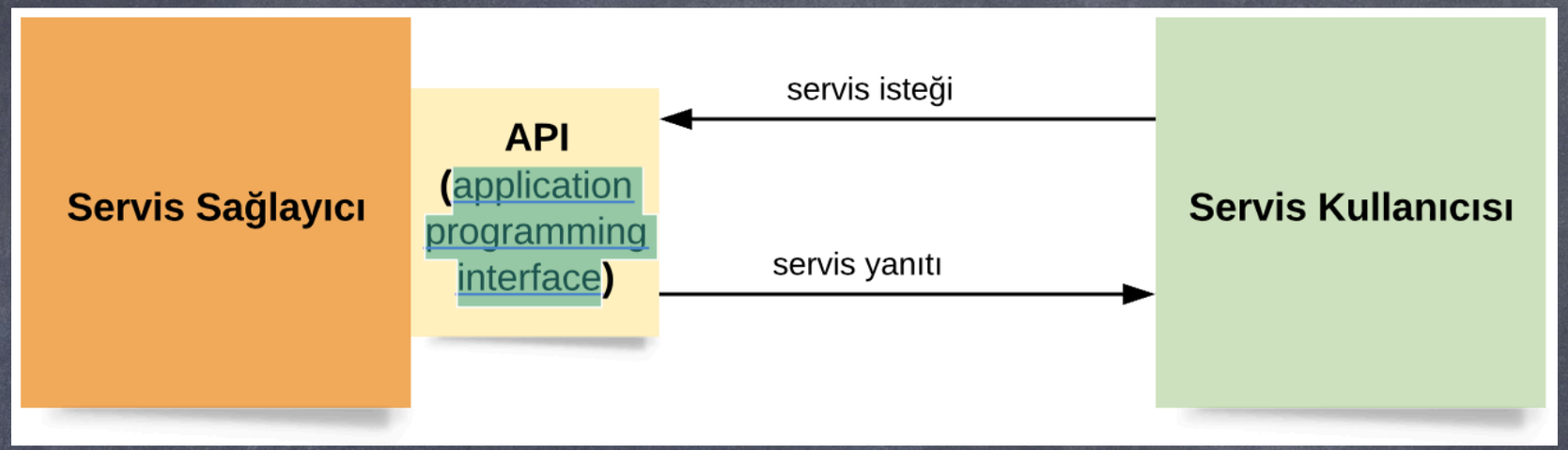
# 3. Yazılım mimarileri (İstemci/Sunucu(Web Uygulama Mimarisi))



# 3. Yazılım mimarileri (İstemci/Sunucu(Ölçeklenebilir Web Uygulama Mimarisi))



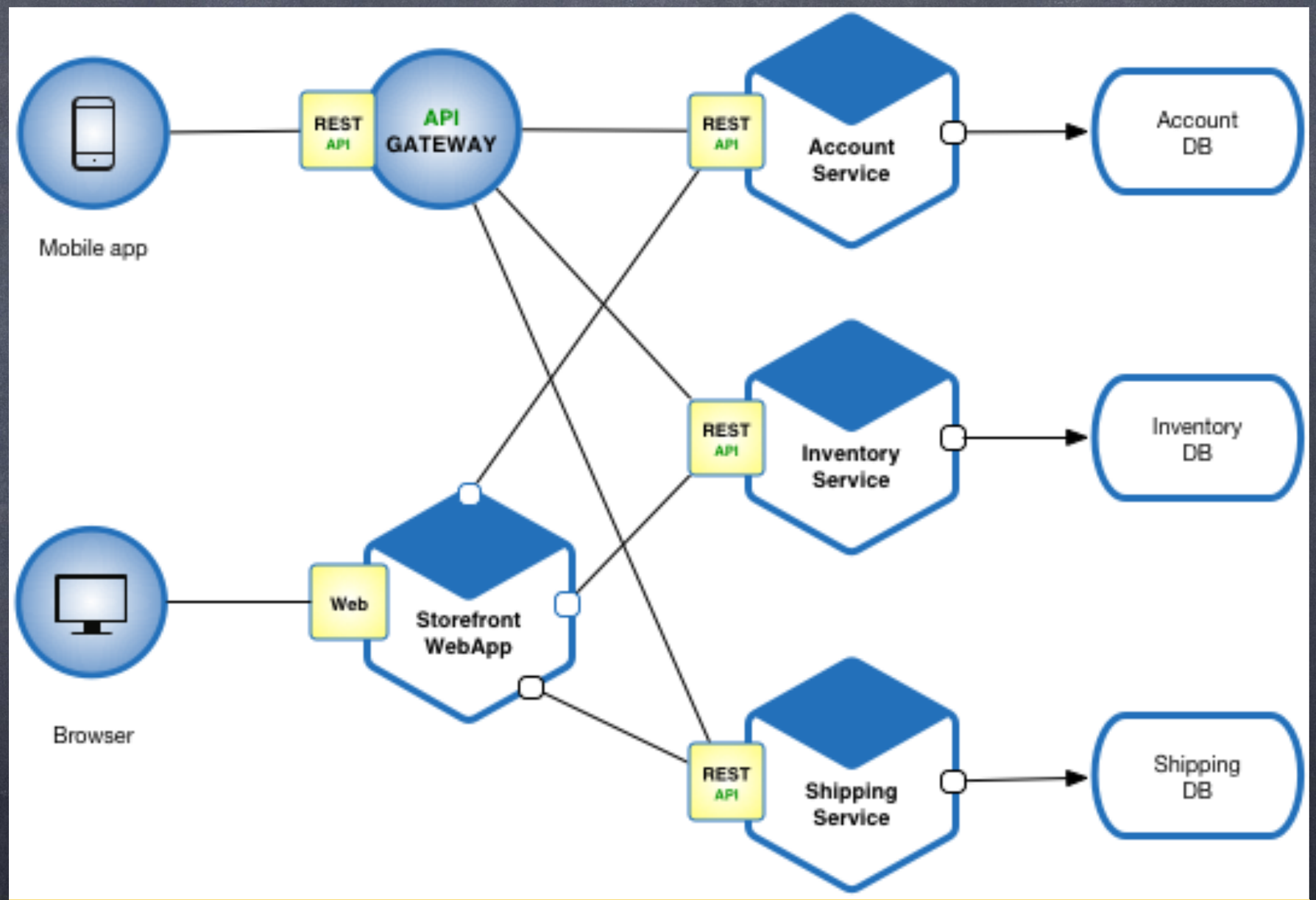
# Servis Yönelimli Yazılım Mimarisi (Service-Oriented Architecture (SOA))




* Sistemlerin birlikte çalışmasını sağlar
* Servisler birbirlerinden bağımsızdır.
* RESTful, gRPC, SOAP, XML-RPC, Jini, COBRA...
* RESTful ile json desteği

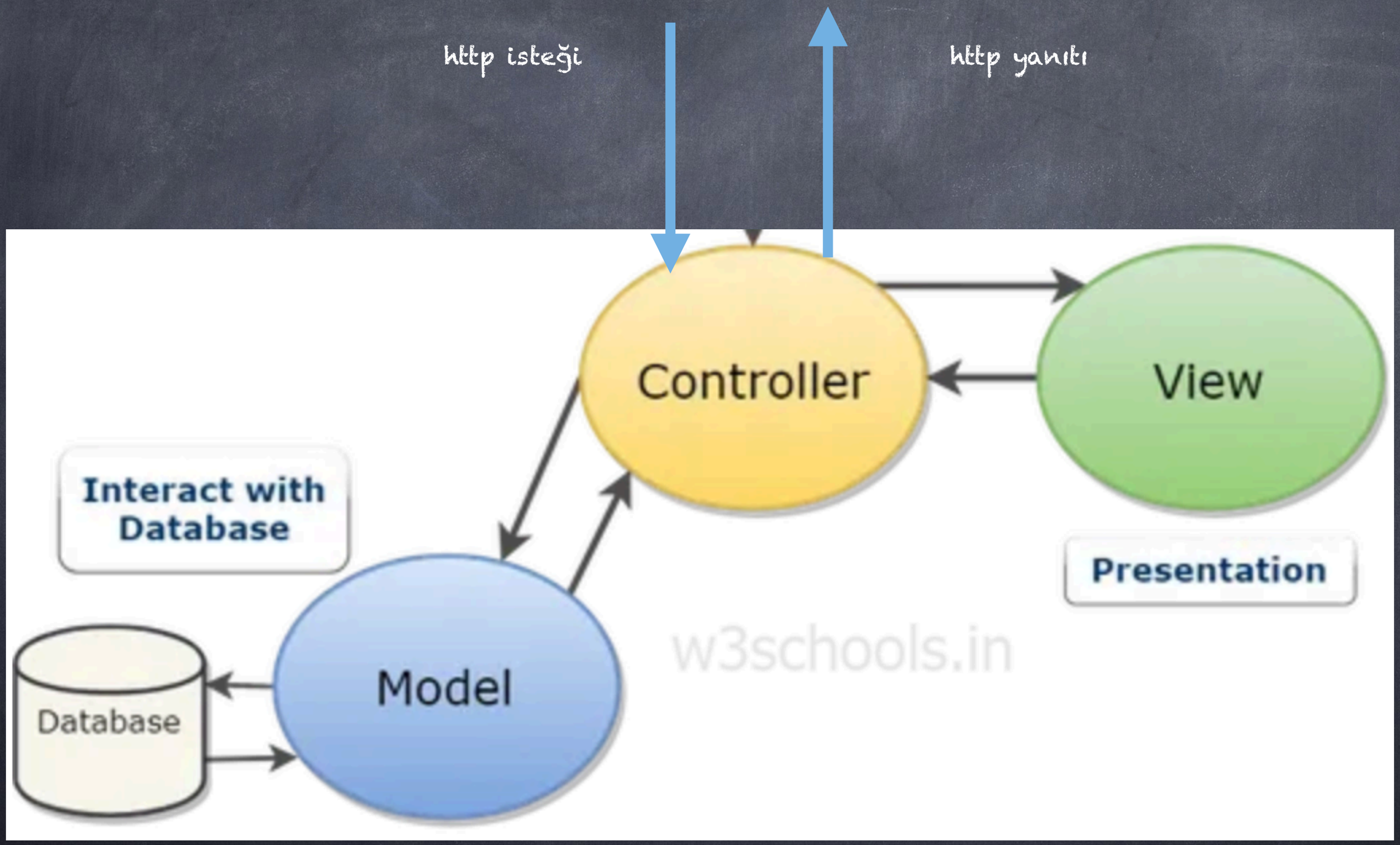
# Mikro Servis Mimarisi

* servis yönelimli mimarinin modern bir türevidir.
* uygulamalar çok sayıda servisin bir araya getirilmesiyle oluşturulur.
* servislerin; bakımı ve testi kolaydır, bağımsız olarak konuşlandırılırlar/ geliştirilirler.




https://microservices.io/

# 3. Yazılım mimarileri (MVC)

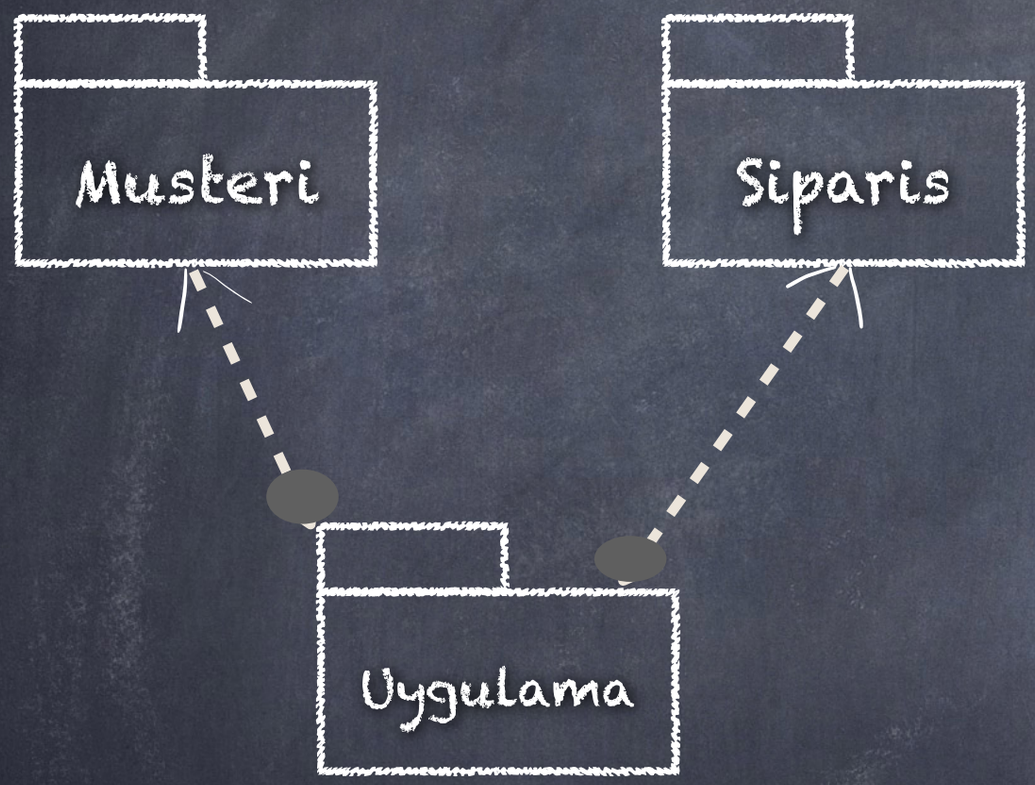


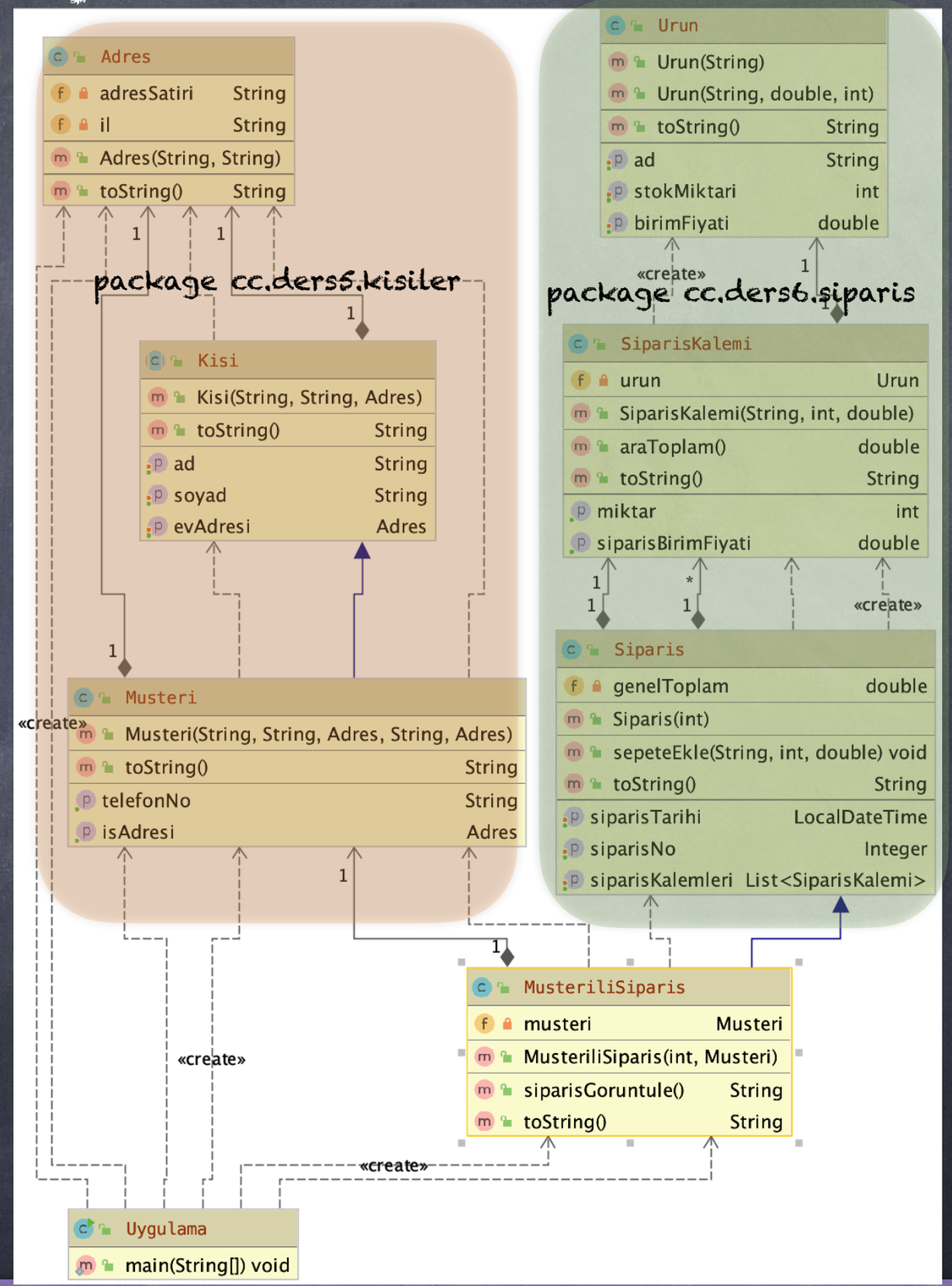

https://www.w3schools.in/mvc-architecture

# Nesne Yönelimli Analiz ve Tasarım

Laboratuvar Uygulaması

# Laboratuvar Uygulaması




Müşteri Sipariişleri

https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders8/LaboratuvarUygulamasi

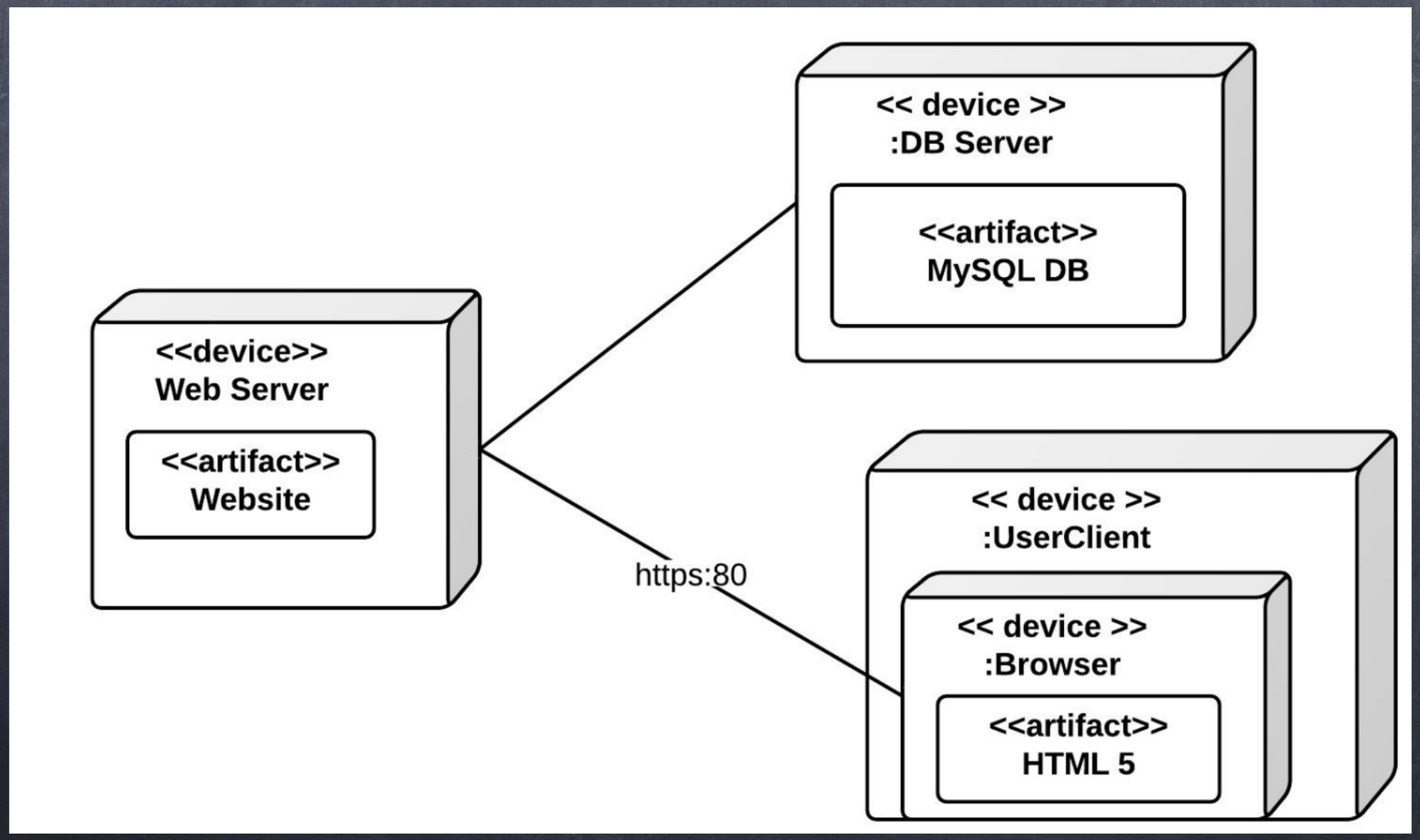
# Nesne Yönelimli Analiz ve Tasarım

## Sistem Tasarımı

Sistemin Konuşlandırılması
Erişim Denetimi (Rol Tabanlı Erişim Denetimi)
Nesnelerin Depolanması

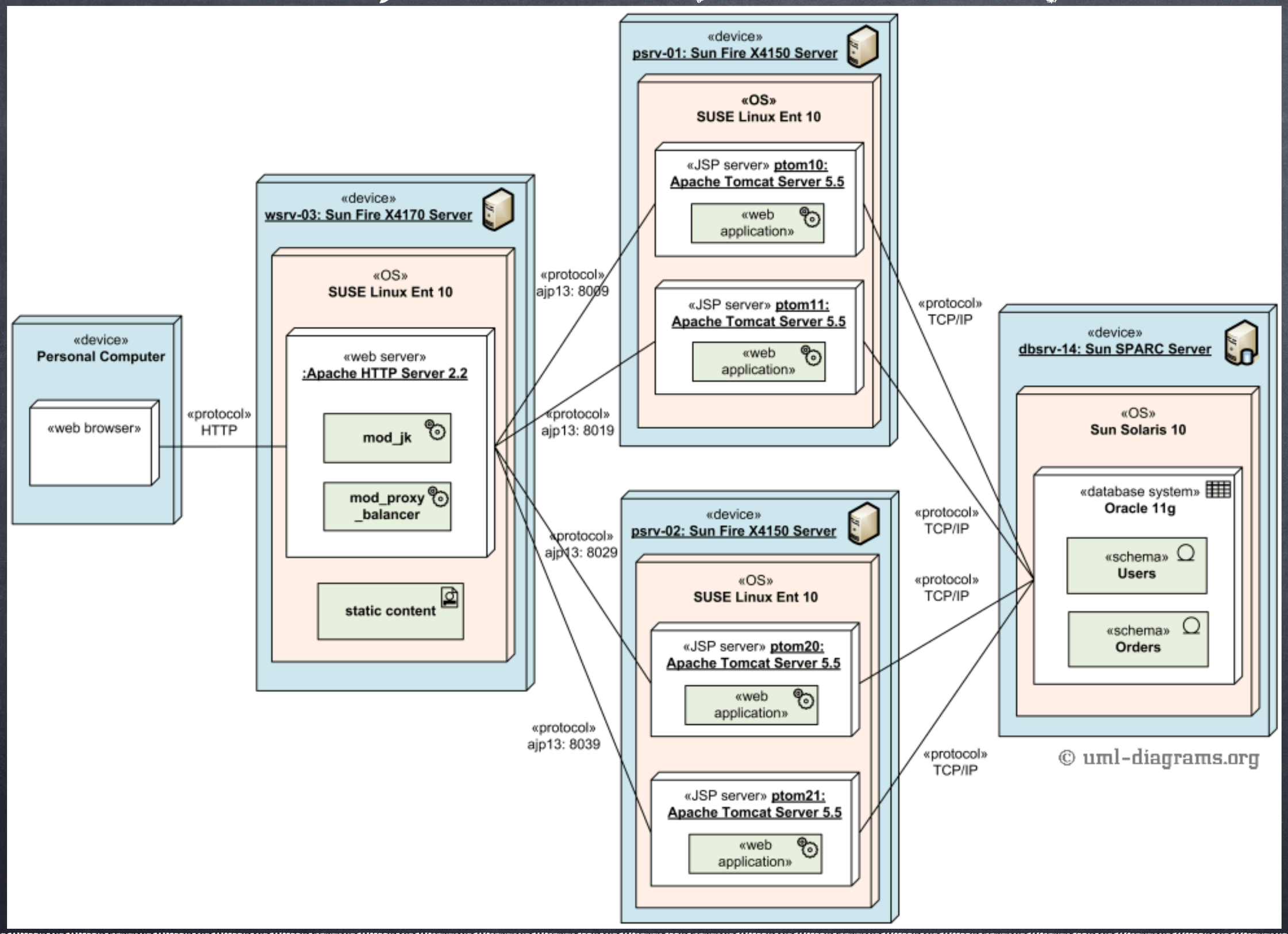
# 4. Sistemin Konuşlandırılması/ UML "Deployment" Diagramı

* Yapısal gösterim şekillerindendir.
* Yazılım ürünlerinin/bileşenlerinin (artifact/component) düğümlere (cihaz, işletim sistemi, sanal makine vb.) konuşlandırılmasıyla ilgili bilgi verir.
* Yazılımların konuşlandırıldığı fiziksel topolojiyi gösterir.




https://www.lucidchart.com/pages/uml-deployment-diagram

# 4. Sistemin Konuşlandırılması- Ölçeklenebilir Web Uygulaması




Soru1: SABİS uygulamasına ait konuşlandırma diyagramını oluşturunuz.

https://www.uml-diagrams.org/web-application-clusters-uml-deployment-diagram-example.html?context=depl-examples

# 5. Erişim Denetimi (Rol Tabanlı Erişim Denetimi)

* Kimlik Doğrulama (Authantication), bir kullanıcının kim olduğunu doğrulama işlemidir.
* Yetkilendirme (Authorization): Bir kullanıcının kaynaklar üzerindeki haklarının belirlenmesi işlemidir.
* Erişim Denetimi: yetki kontrolü.
* Kaynak/Değer/Varlık :
    * İşletim Sistemi: bellek (okuma, yazma), dosya(okuma, yazma, çalıştırma), klasör, port (okuma, yazma) vb.
    * Veritabanı: tablo (insert, update, select, ...), view, fonksiyon vb.
    * Uygulama Yazılımları: sayfa (erişim), fonksiyon (çalıştırma)
* Yetki (permission,privilege, access right): Kaynaklar (Değer/Varlık) üzerindeki sahip olunan haklar.
* Yetkiler rollerle ilişkilendirilir, kullanıcılar rollere üye yapılır. Böylece yetkilerin yönetimi daha kolay yapılır.

# 5. Erişim Denetimi (Rol Tabanlı Erişim Denetimi)

## Case Study: Online Shopping Application

| ACTIVITY MATRIX | | | | |
|---|---|---|---|---|
| | | | | |
| | | External Entities & Roles | | |
| **Assests** | **Action** | **Guest** | **Registered User** | **Admin** |
| Customer Data | Create | Always | Own Data | Always |
| | Read | Never | Own Data | Always |
| | Update | Never | Own Data | Always |
| | Delete | Never | Own Data | Always |
| | | | | |
| Login Page | Access | Always | Always | Always |
| Customer Profile Page | Access | Never | Always | Never |
| | | | | |
| Login Function | Execute | Always | Always | Always |
| Add Shopping Chart Function | Execute | Never | Always | Never |
| Checkout Fuction | Execute | Never | Always | Never |

Soru1: Proje çalışmasında geliştirilecek uygulamanın etkinlik matrisini oluşturunuz.

# 6. Erişim Denetimi (Web Application (Spring Security))

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
            .authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
                .antMatchers("/resources/**", "/").permitAll()
                .antMatchers("/dashboard").hasRole("ADMIN")
                .anyRequest().authenticated()
                .and() HttpSecurity
            .formLogin() FormLoginConfigurer<HttpSecurity>
                .defaultSuccessUrl("/dashboard")
                .and() HttpSecurity
            .logout() LogoutConfigurer<HttpSecurity>
                .permitAll();
}
```

```
@GetMapping("/registeredUser")
@PreAuthorize("hasAuthority('RegisteredUser')")
public String moderatorAccess() {
    return "RegisteredUser Board.";
}

@GetMapping("/administrator")
@PreAuthorize("hasAuthority('Administrator')")
public String adminAccess() { return "Admin Board."; }
```

# 6. Erişim Denetimi (Veritabanı Yönetim Sistemi)

## Yetkilendirme İşlemleri

> PUBLIC: Tüm roller / kullanıcılar.
> kullaniciAdi: Tek bir kullanıcı.
> ALL: Tüm yetkiler.

- rol1 isimli role customers tablosu üzerinde seçim yapma yetkisi ver.

```
GRANT SELECT ON "customers" TO "rol1";
```

- Tüm rollere customers tablosu üzerinde kayıt ekleme yetkisi ver.

```
GRANT INSERT ON "customers" TO PUBLIC;
```

- rol1 isimli kullanıcıya customers tablosu üzerinde tüm yetkileri ver.

```
GRANT ALL ON "customers" TO "rol1";
```

- rol1 isimli rolün customers tablosu üzerindeki güncelleme yetkisini geri al.

```
REVOKE UPDATE ON "customers" FROM "rol1";
```

- rol1 isimli rolün customers tablosu üzerindeki tüm yetkilerini geri al.

```
REVOKE ALL ON "customers" FROM "rol1";
```

- rol1 kullanicisinin Sema1 icerisindeki nesnelere ait tüm yetkileri geri alinir.

```
REVOKE ALL ON SCHEMA "Sema1" FROM "rol1";
```

https://github.com/celalceken/DatabaseManagementSystems/blob/master/VYS14.md

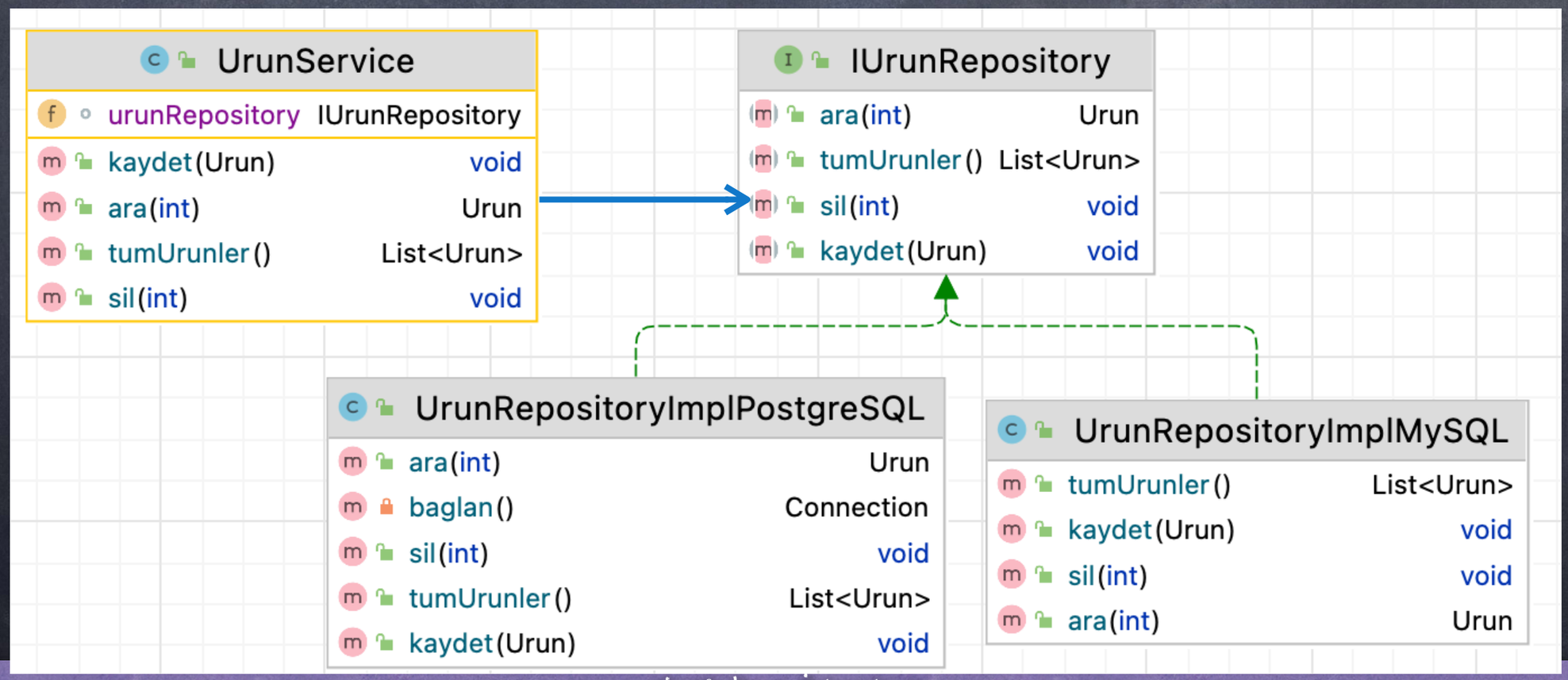
# 7. Nesnelerin Depolanması

* Bazı nesnelerin (entity object) kalıcı olarak saklanması gerekir.
* Kalıcı depolama için klasik dosyalar ya da veritabanı yönetim sistemleri (ilişkisel vys, nosql vb.) kullanılır.
* Veritabanı sürücülerinin yüklenmesi ve uygulamalar içerisinde kullanılması:
    * https://github.com/celalceken/DatabaseManagementSystems/blob/master/VYS06.md
* İlişkisel VYS kullanılması durumunda saklanacak nesneler için tablo oluşturulur.
* Tablodaki her satır bir nesneye, tablo ise sınıfa karşılık gelir.

# 7. Nesnelerin Depolanması

Depolama işleminin iş mantığından soyutlanması-Interface ile soyutlama

* gerçeklemeler(sınıflar-UrunRepositoryImplX) henüz ortada yok iken istemci kod (UrunService) içerisinde kullanılabilir
    * gereçekleme ve istemci modüller aynı anda farklı kodlayıcılar tarafından oluşturulabilir (takım çalışması)
* mevcut kodlar değiştirilmeden yeni özellikler eklenebilr.
* istemci kod, gerçekleme kısmındaki değişikliklerden etkilenmez (bağımlılığın zayıflatılması)
* aynı istemci kod farklı modüller tarafından defalarca kullanılabilir (kod tekrar kullanımı-code reuse)
* SOLID Dependency inversion ve open-closed prensipleriyle ilgilidir.



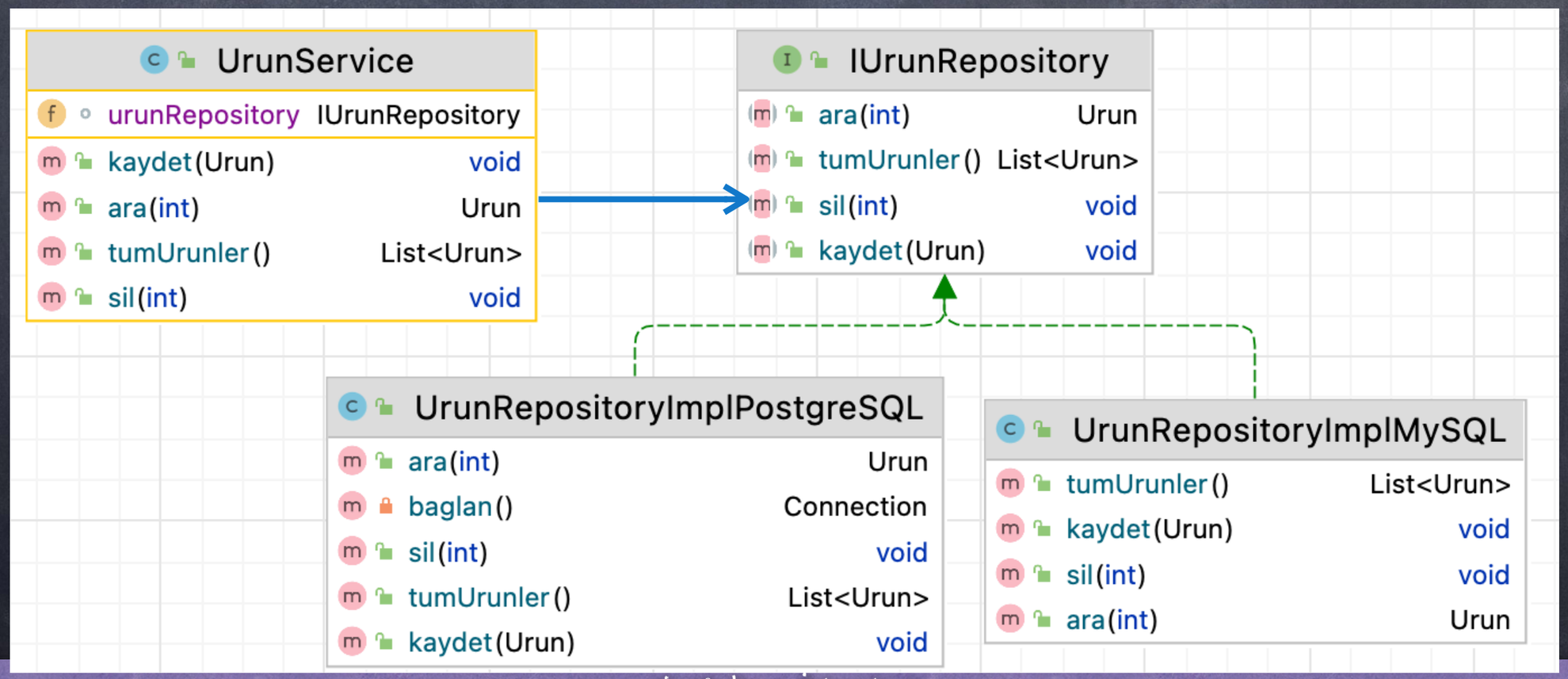



https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders9/NesnelerinDepolanmasi/repository

# 7. Nesnelerin Depolanması

Soru1: Veritabanında bulunan ürünlerin sayısını hesaplayan yöntemi ekleyiniz.
Soru2: Ürün adına göre arama yapılabilmesini sağlayınız.

Soru3: Ürün bilgilerinin güncellenebilmesini sağlayınız.
Soru4: Urun sınıfı içerisinde stok miktarı (stokMiktari) da saklanmaktadır. Stokta bulunan ürünlerin tespitini sağlayacak yöntemi (urunMevcutMu(int)) de ekleyiniz. Bu yöntemi, UrunService içerisindeki tumUrunler() yöntemi içerisinde, stokta bulunan ürünlerin döndürülmesini sağlamak için kullanınız.
Soru5: Uygulamanın MongoDB için de çalışabilmesini sağlayınız.





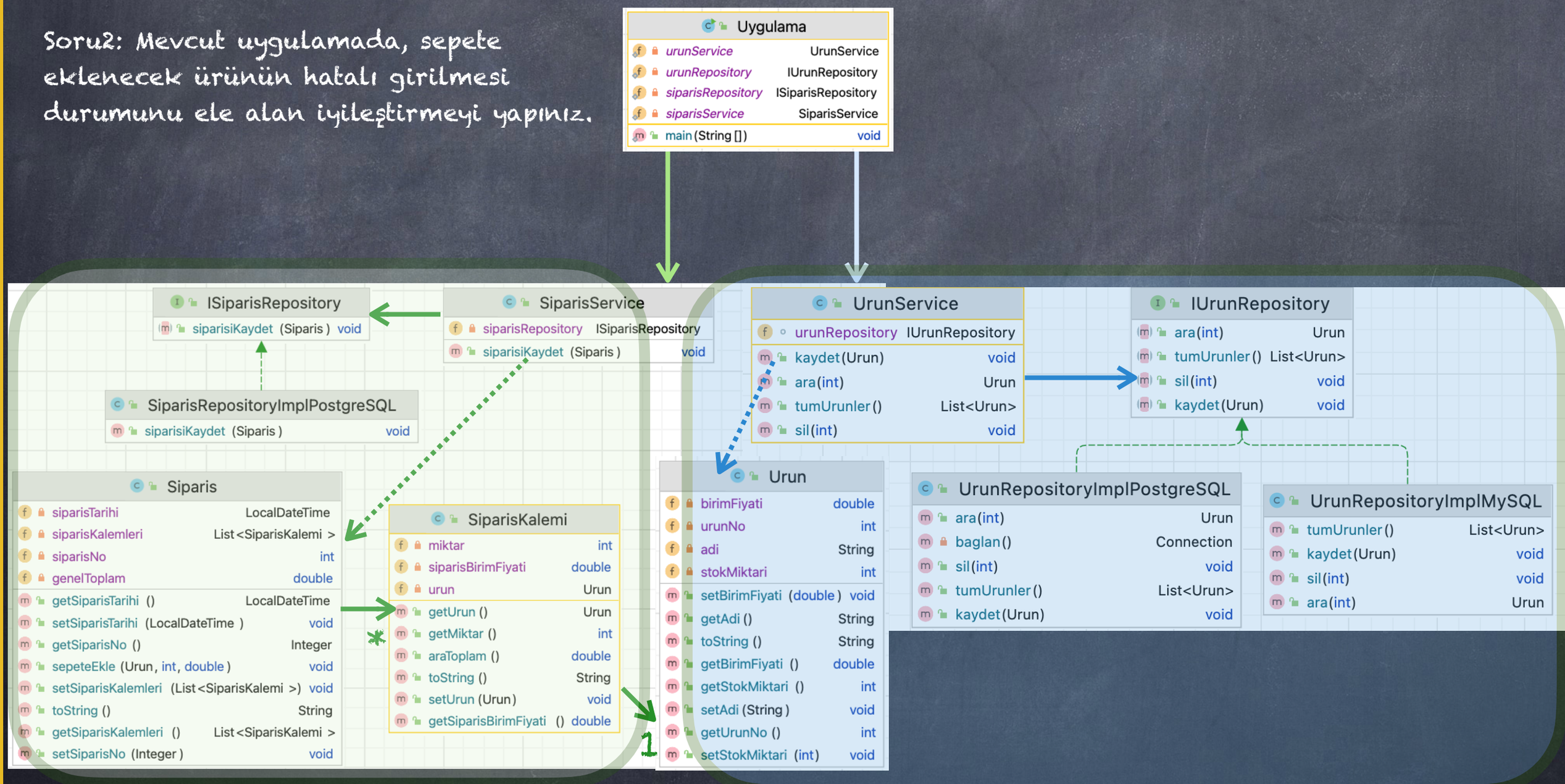
https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders9/NesnelerinDepolanmasi/repository

# 7. Nesnelerin Depolanması

SiparisKalemi, Siparis ile Urun arasındaki çok-çok bağıntısının sonucu olan tabloyu ifade eder.

Soru1: Sipariş veren müşterinin de aşağıdaki uygulamaya/veritabanına eklenmesini sağlayınız.

Soru2: Mevcut uygulamada, sepete eklenecek ürünün hatalı girilmesi durumunu ele alan iyileştirmeyi yapınız.

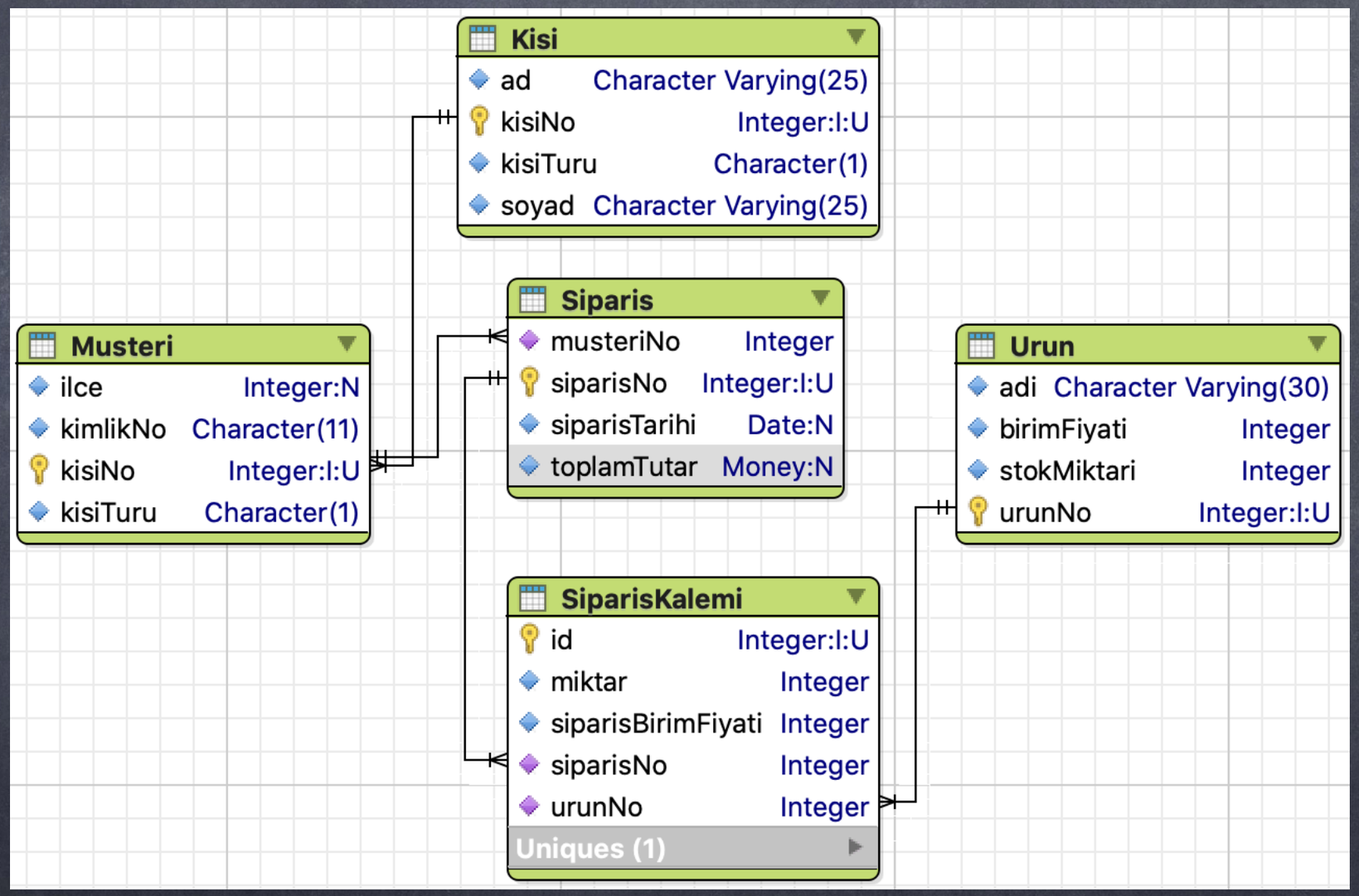



Veritabanı İşlemleri

https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders9/NesnelerinDepolanmasi/siparis

# 7. Nesnelerin Depolanması

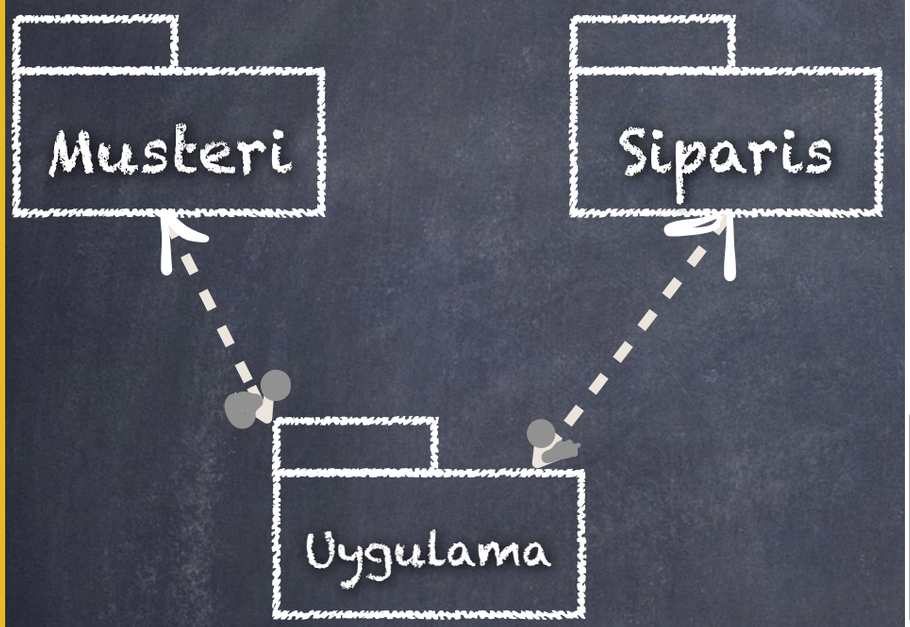
* SiparisKalemi, Siparis ile Urun arasındaki çok-çok bağıntısının sonucu olan tabloyu ifade eder.

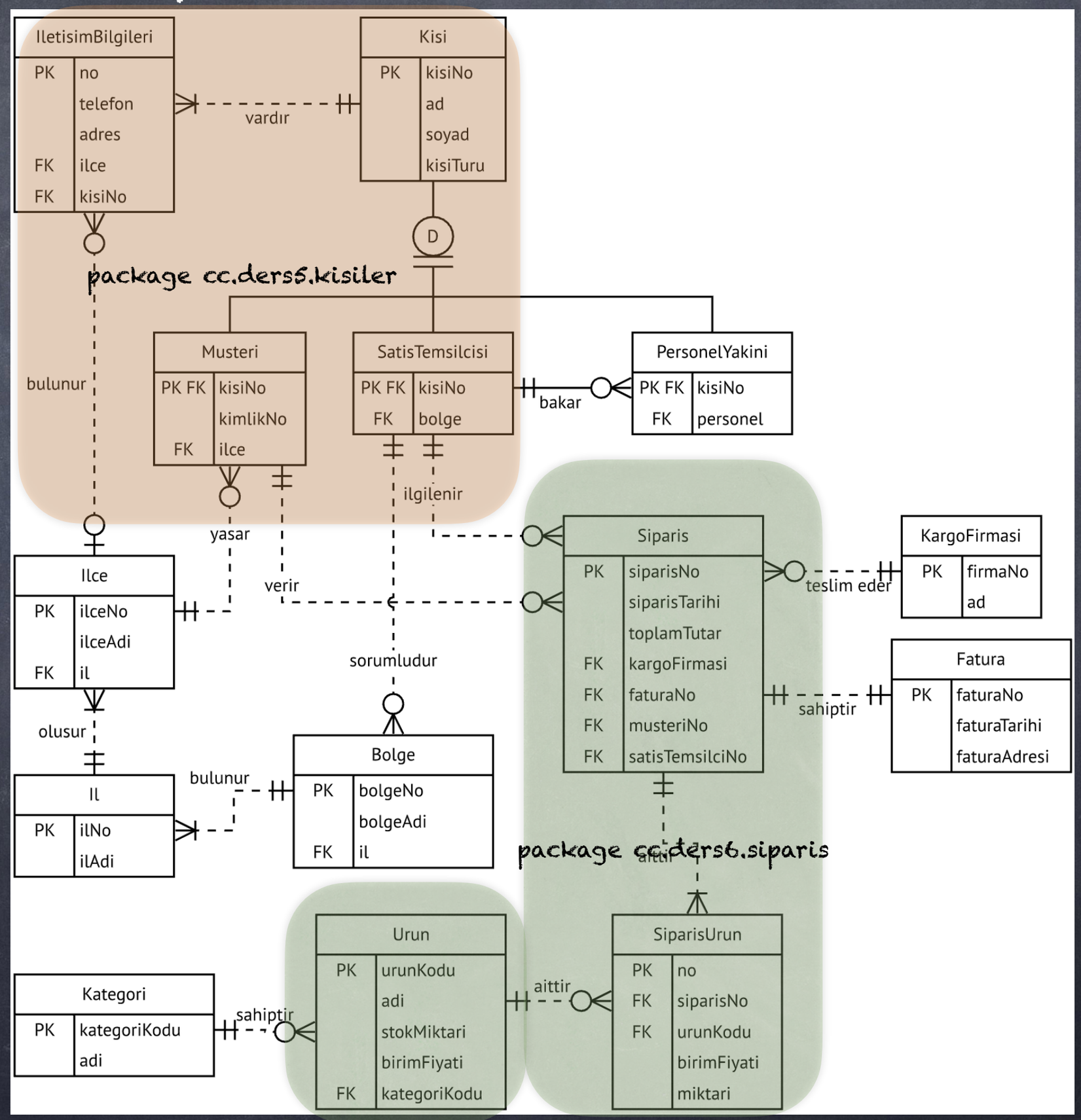




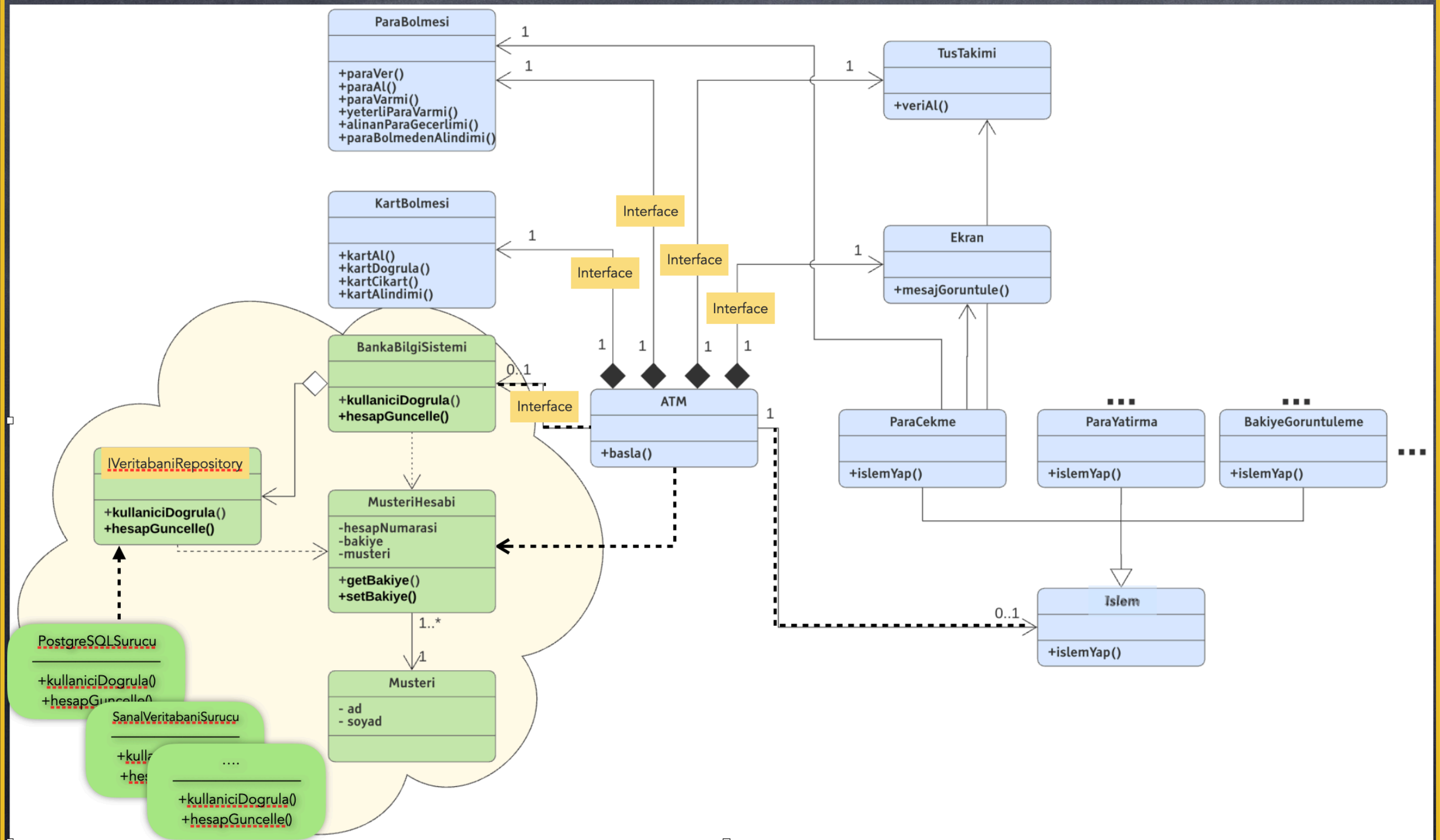
https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders9/NesnelerinDepolanmasi/siparis

# 6. Nesnelerin Depolanması/VB Modeli



# 6. Nesnelerin Depolanması/ATM Sistemi




Soru: ATM sisteminin MongoDB ile de çalışabilmesini sağlayınız.

ATM Uygulaması

https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders6

# 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.

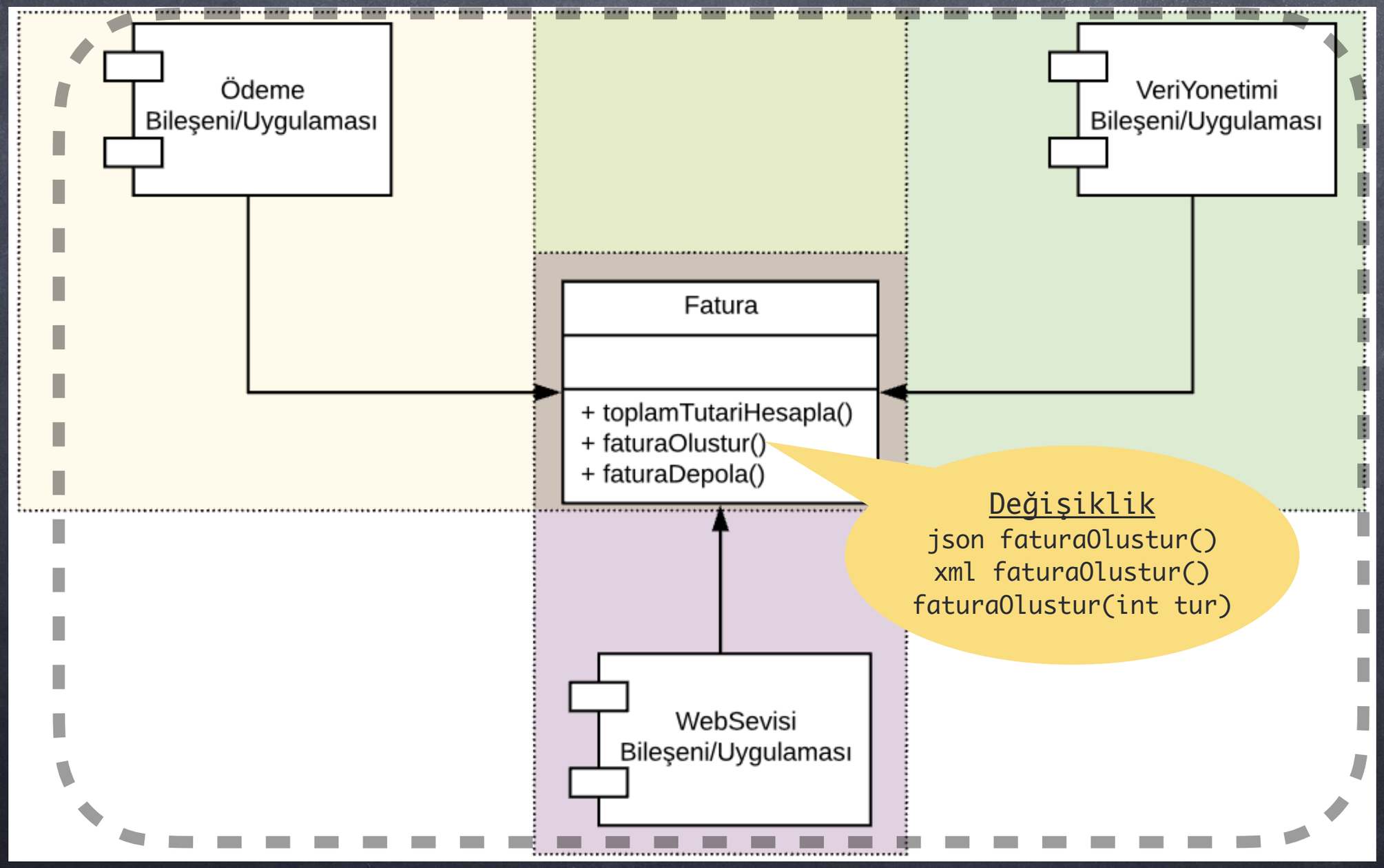
Agile Software Development; Robert C. Martin; Prentice Hall, 2003

# 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

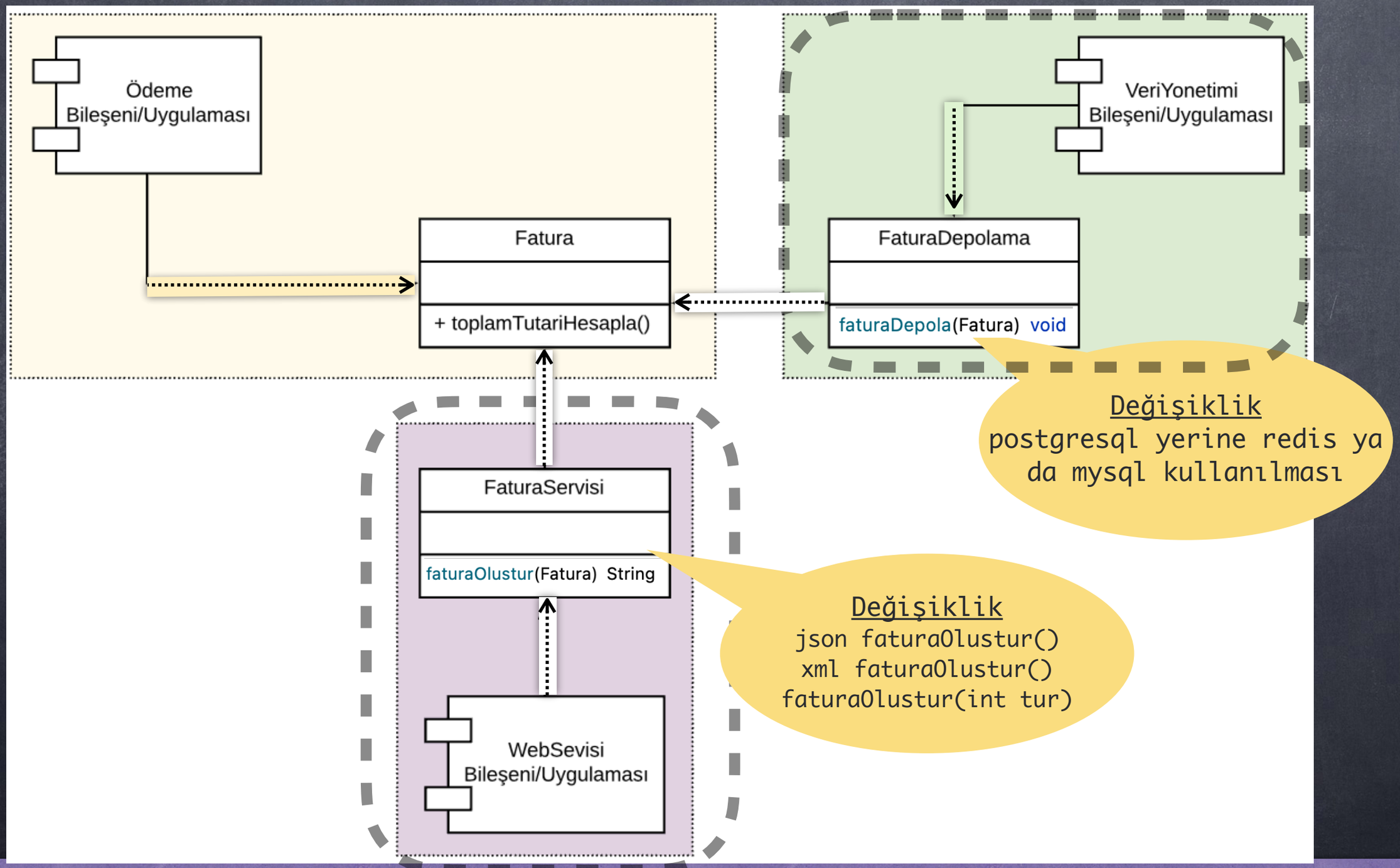
# The Single Responsibility (SRP)

* 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.



# 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"

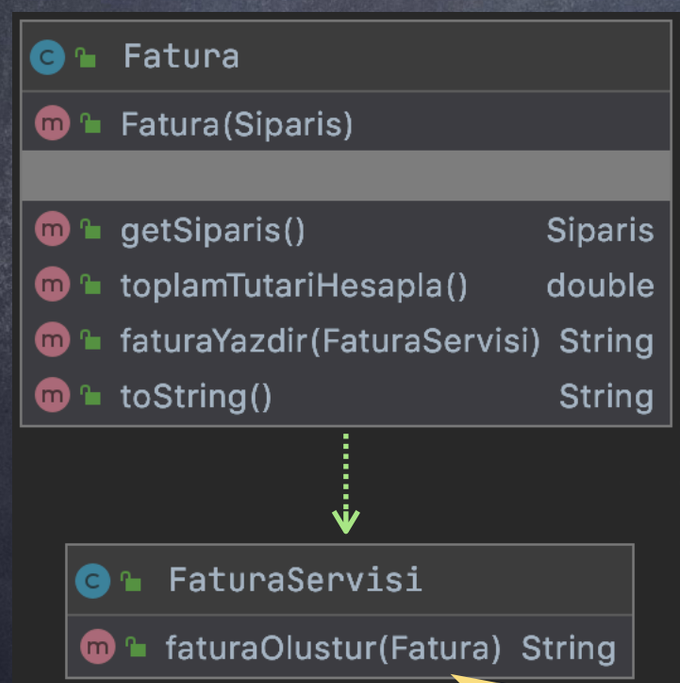
# The Open Closed (OCP)

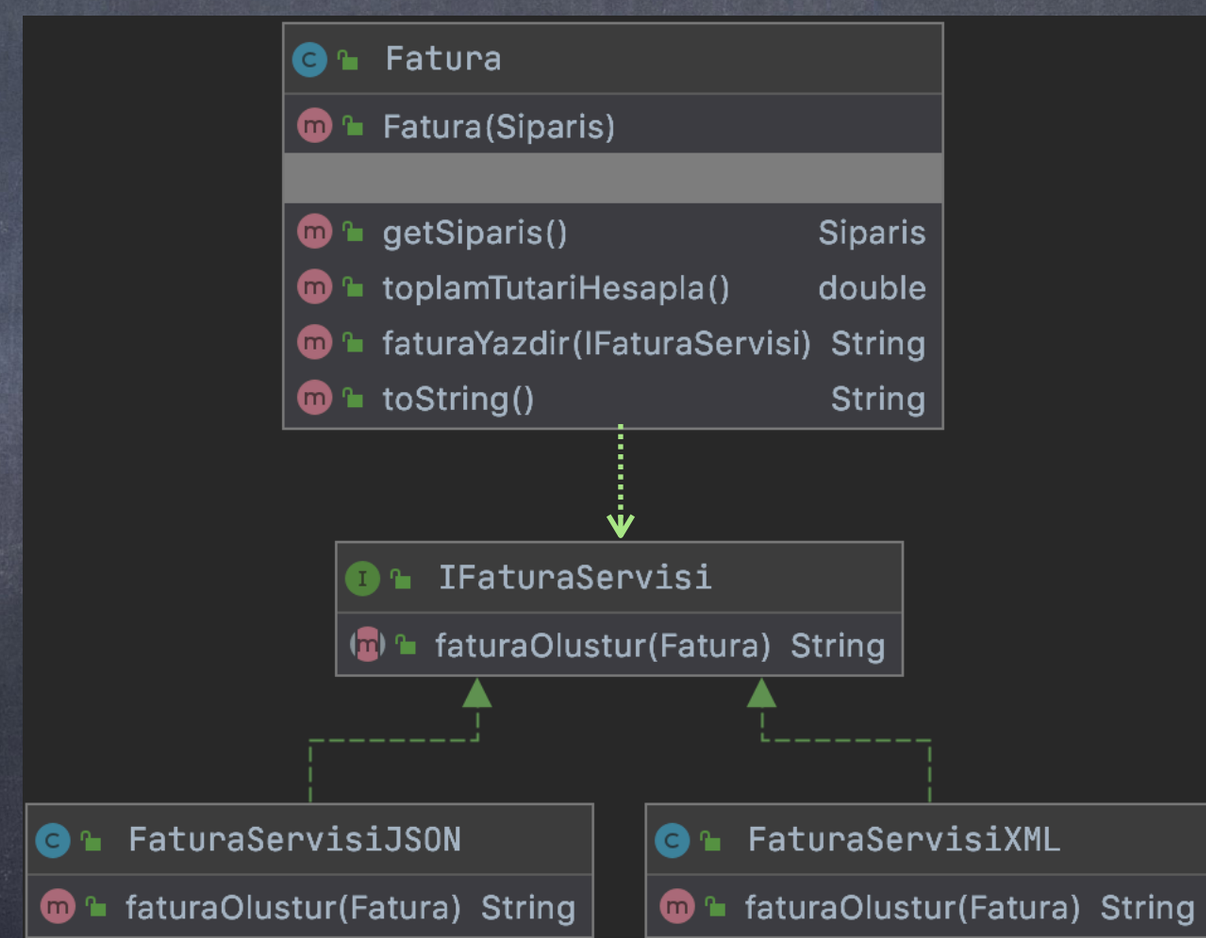
* Bu prensip, değişiklik yapmadan genişleyebilen modüller geliştirmeyi amaçlar.
* Mevcut kodu değiştirmeden yeni kodlar ekleyerek, yeni özellikler eklemeyi sağlar.
* "Open for extension": geliştirilen modüller yeni gereksinimler ortaya çıktığında bunları desteklemek üzere genişleyebilmeli.
* "closed 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.




Değişiklik
faturaOlustur(Fatura, int tur)

Örnek:
https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/blob/master/Ders2/3Interface/README.md

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

# The Liskov Substitution Principle (LSP)

Subclasses should be substitutable for their base classes.

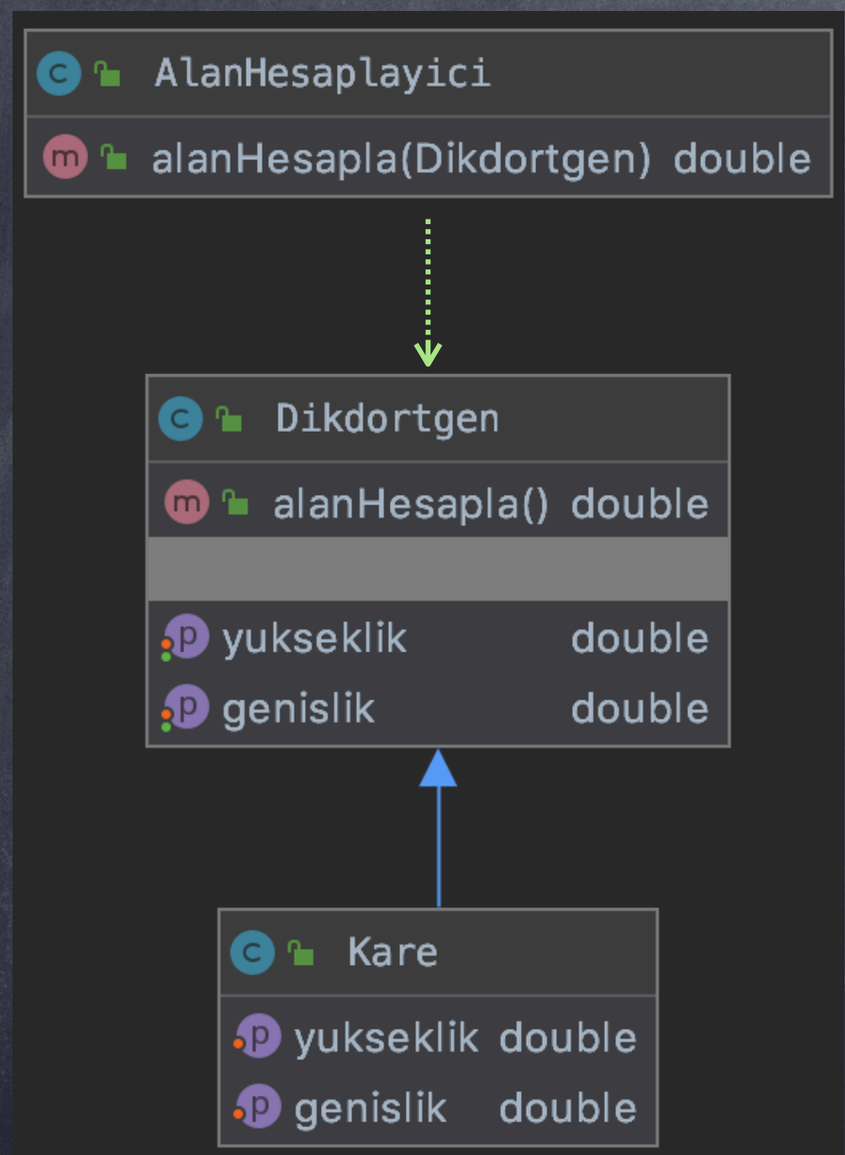
Robert C. Martin (2000). "Design Principles and Design Patterns"

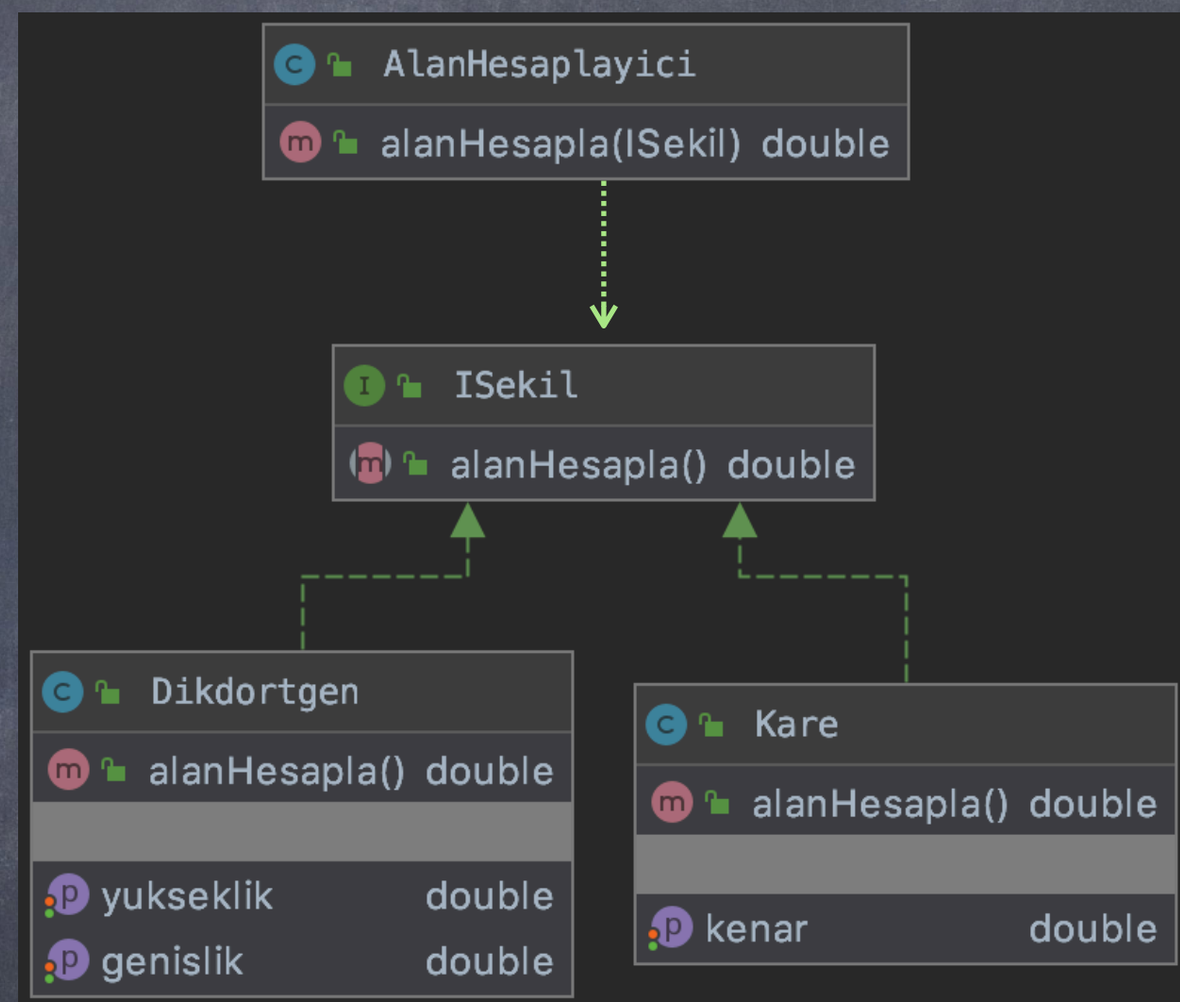
# The Liskov Substitution Principle (LSP)

* 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)

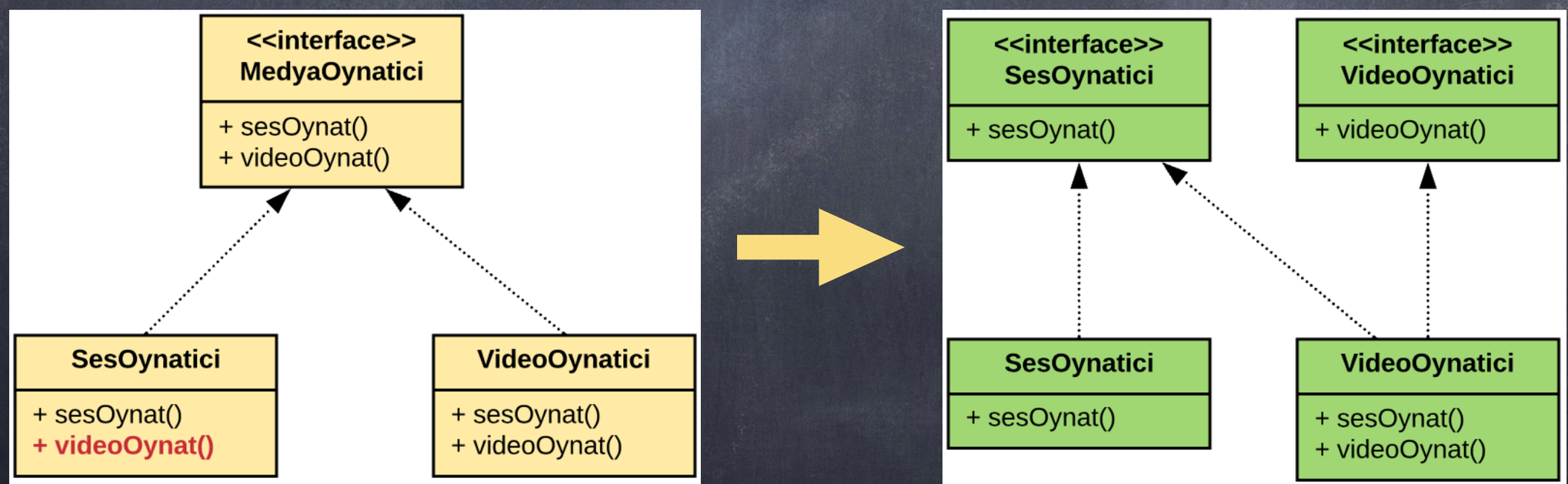
Many client specific interfaces are better than one general purpose interface.

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.

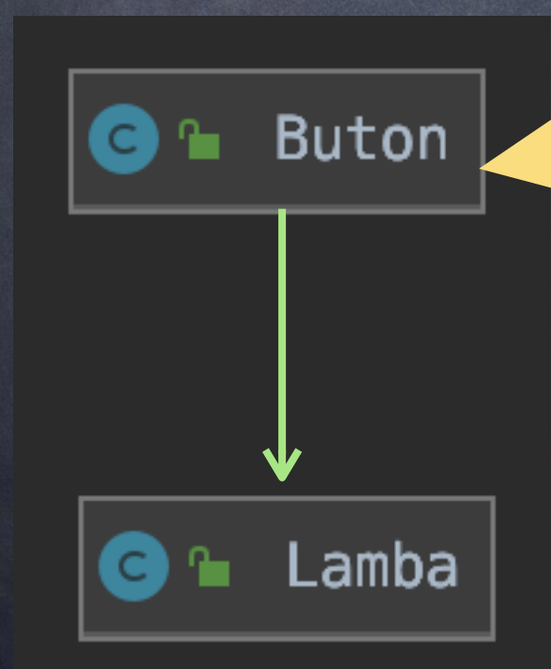
Robert C. Martin (2000). "Design Principles and Design Patterns"

# The Dependency Inversion Principle (DIP)

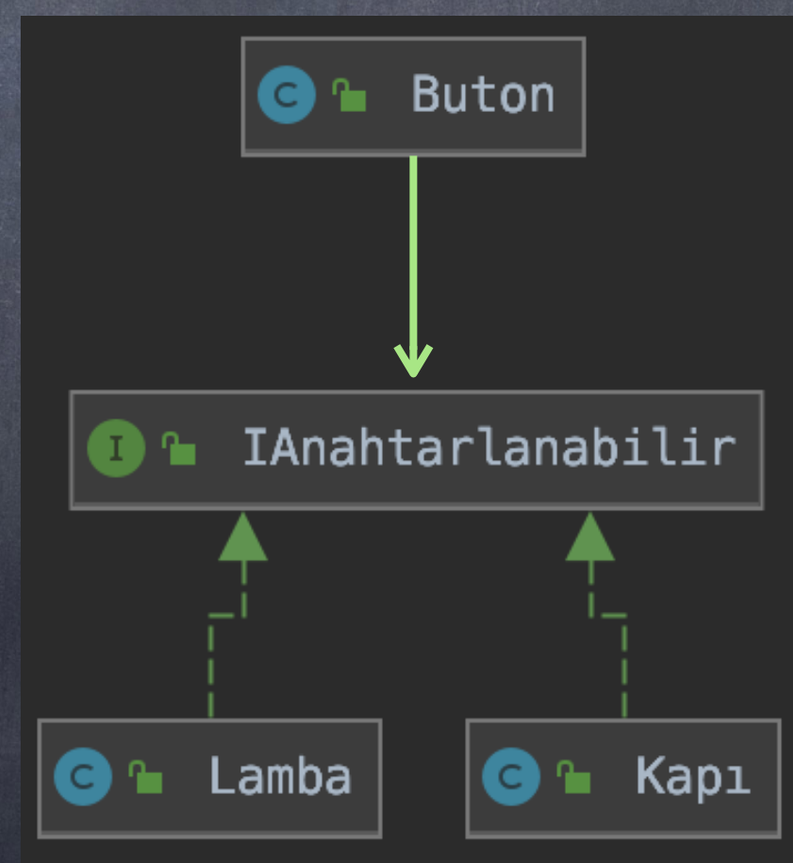
* Nesneler arasındaki bir bağlantıda, yüksek seviyeli modül ile düşük seviyeli modül (her ikisi birden) soyutlamaya bağlı olmalı. Her ikisi birbirine doğrudan değil arayüz üzerinden bağlanmalı.
* Böylece modülde yapılacak değişiklik diğer modülleri etkilememiş olur. Değişiklik yapmak kolaylaşır. Yeni 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).




Değişiklik
buton nesnesi kapıyı kontrol etmek için de kullanılsın.




Örnek:
https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/blob/master/Ders2/3Interface/README.md

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

# Nesne Yönelimli Analiz ve Tasarım

## Nesne Tasarımı

Tasarım Desenleri
Design Patterns

# Tasarım Desenleri (Design Patterns)

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

Gamma, E. vd. (1994). "Design Patterns:Elements of Reusable Object-Oriented Software "

# Tasarım Desenleri (Design Patterns)

* 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 içerisine 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 |

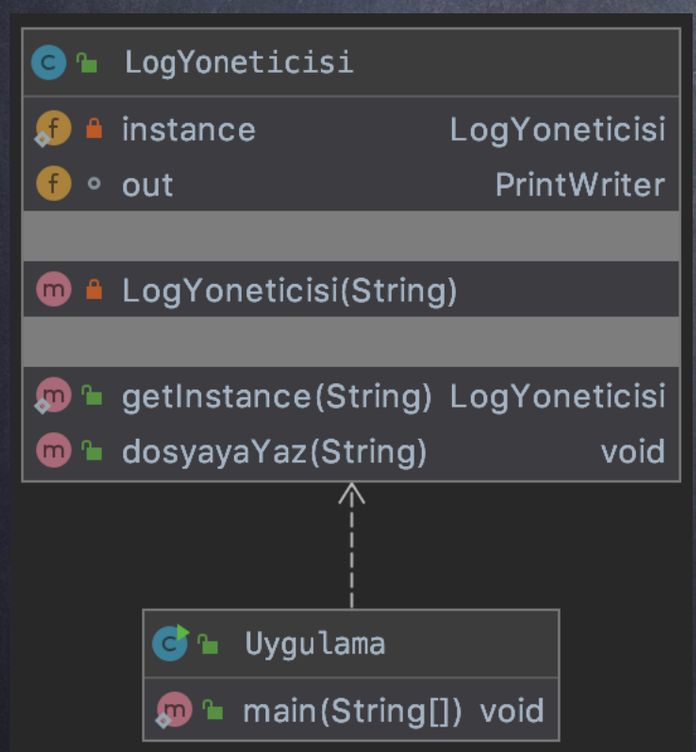
Gamma, E. vd. (1994). "Design Patterns:Elements of Reusable Object-Oriented Software "

# Tasarım Desenleri : Singleton

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

# Tasarım Desenleri : Singleton

Soru1: Bu modülü, ATM uygulaması içerisine ekleyiniz. ATM sisteminde kart doğrulama, kullanıcı doğrulama ve bakiye değişimi bilgilerinin log dosyasına yazılmasını sağlayınız.



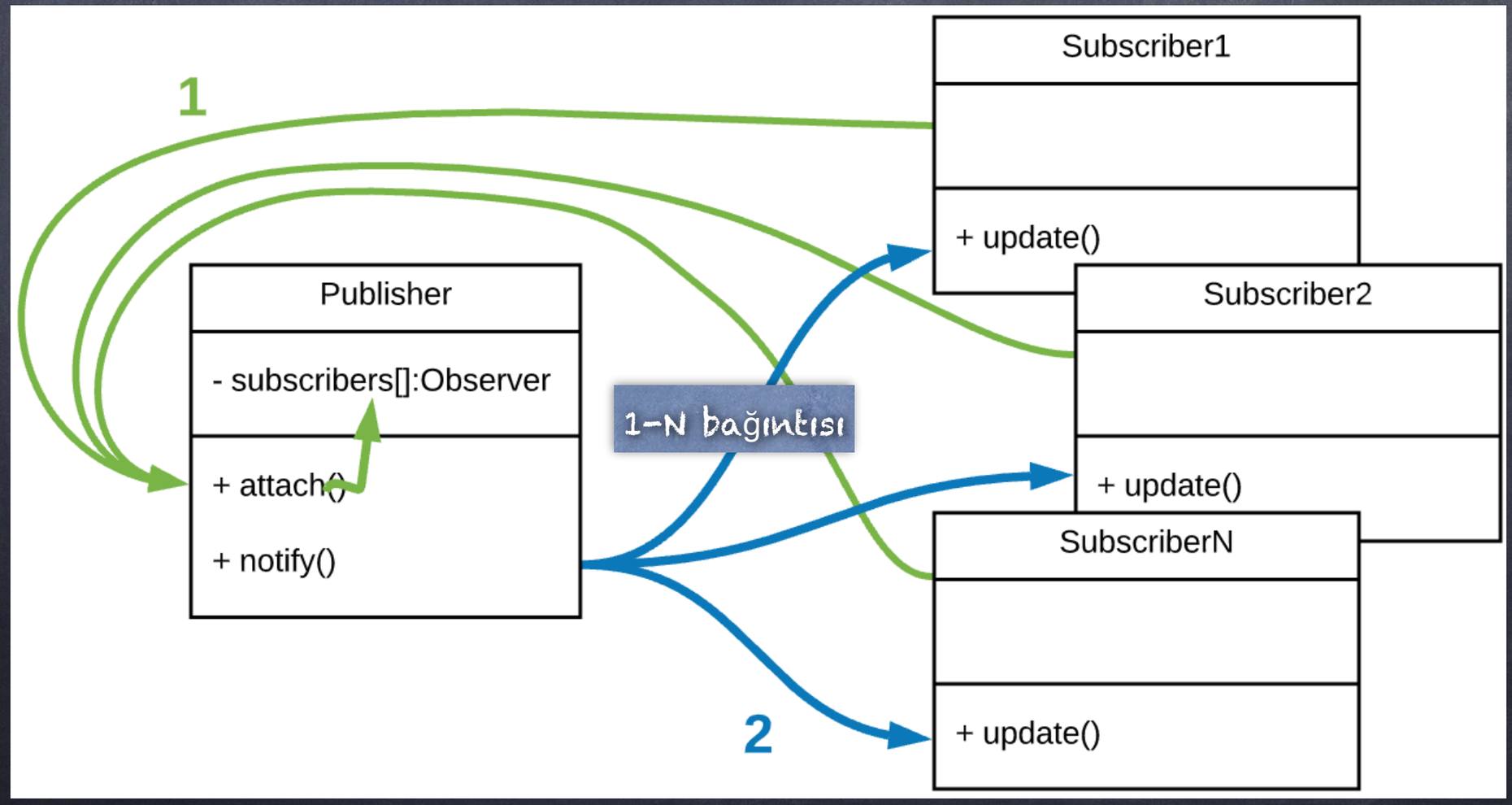

```java
public class LogYoneticisi {

    private static LogYoneticisi instance;
    PrintWriter out;

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

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

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

}
```

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

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

# Tasarım Desenleri : Observer (Gözlemci)

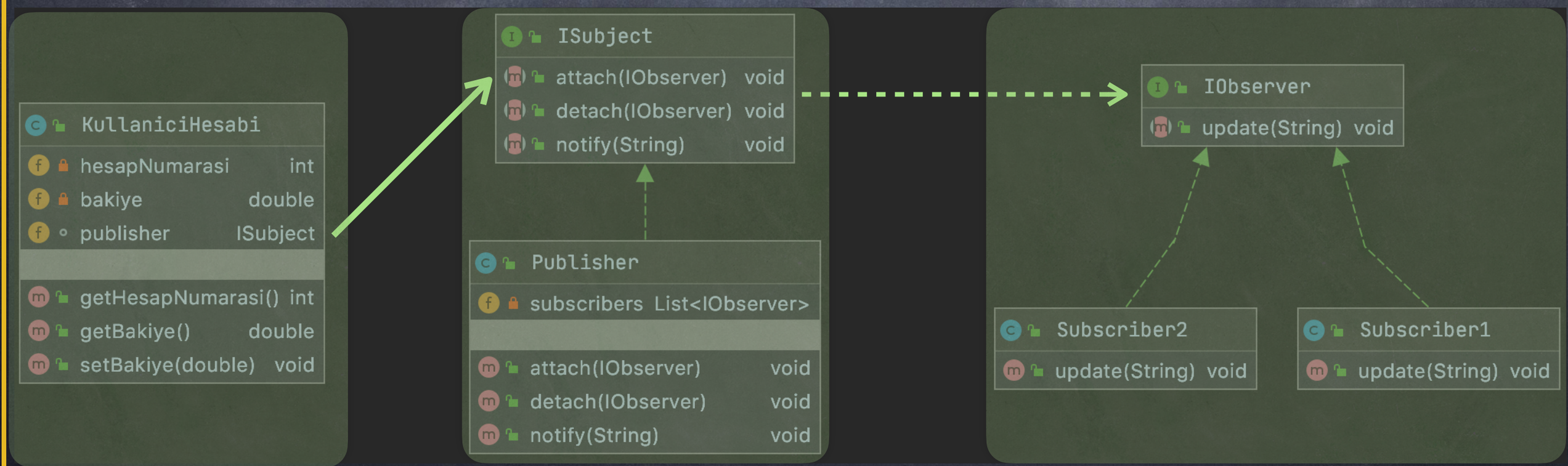
* Çok sayıda nesneye, gözlemledikleri nesnede meydana gelen olayı bildirmek gerektiğinde kullanılabilir.
* Davranışsal desenlerden biridir.
* Kullanım örnekleri:
    * mağazaya ürün geldiğinde ilgili müşterilere bildirim gönderilmesi, ürün indirime girdiğinde bildirim gönderilmesi
    * bakiye değiştiğinde 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/çıkartılması, istemcideki değişikliklerin gözlemcilere bildirilmesi işlemlerinden sorumludur.
* Observer (subscriber-abone): istemcideki olayların yansıtıldığı/gönderildiği nesne.
* İstemci (KullaniciHesabi): olayın üretildiği nesne.

Örnek: Kullanıcının bakiyesi değiştiğinde, bu olayı abonelere bildiren uygulama.



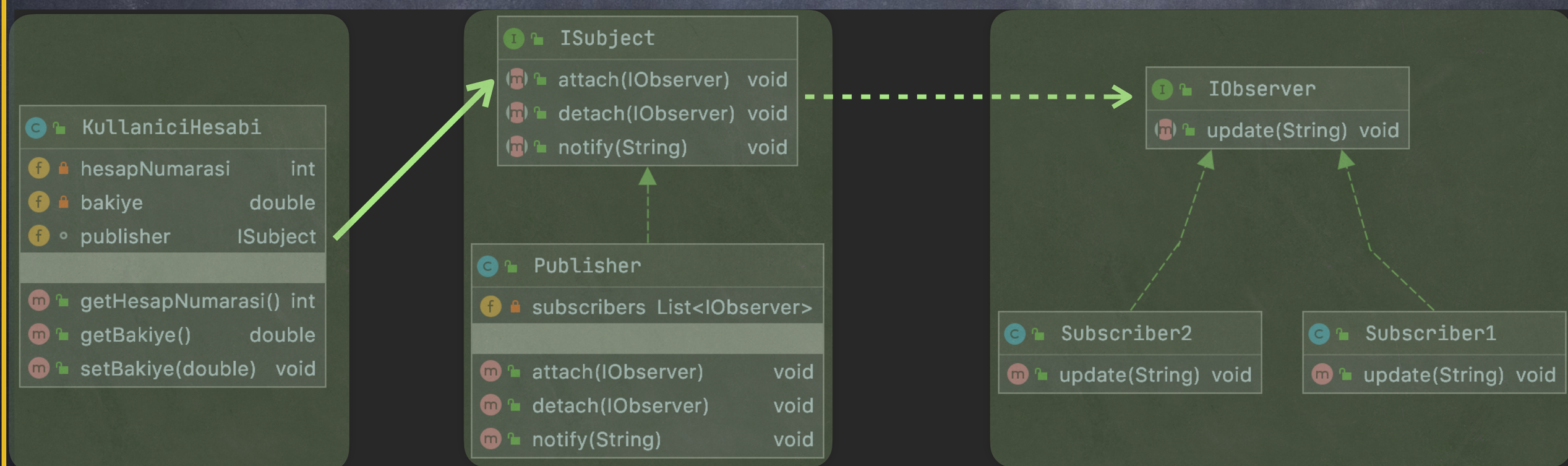

https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders10/

# Tasarım Desenleri : Observer (Gözlemci)

Soru1: Bu modülleri ATM sistemine ekleyiniz. Bakiye değiştiğinde bu değişiklik ilgili abonelere gönderilsin. Abonelerden biri içerisinde LogYoneticisi ile log alma işlemi gerçekleştirilsin.

Örnek: Kullanıcının bakiyesi değiştiğinde, bu olayı abonelere bildiren uygulama.

istemci(event source) | subject (publisher) | observer (subscriber)

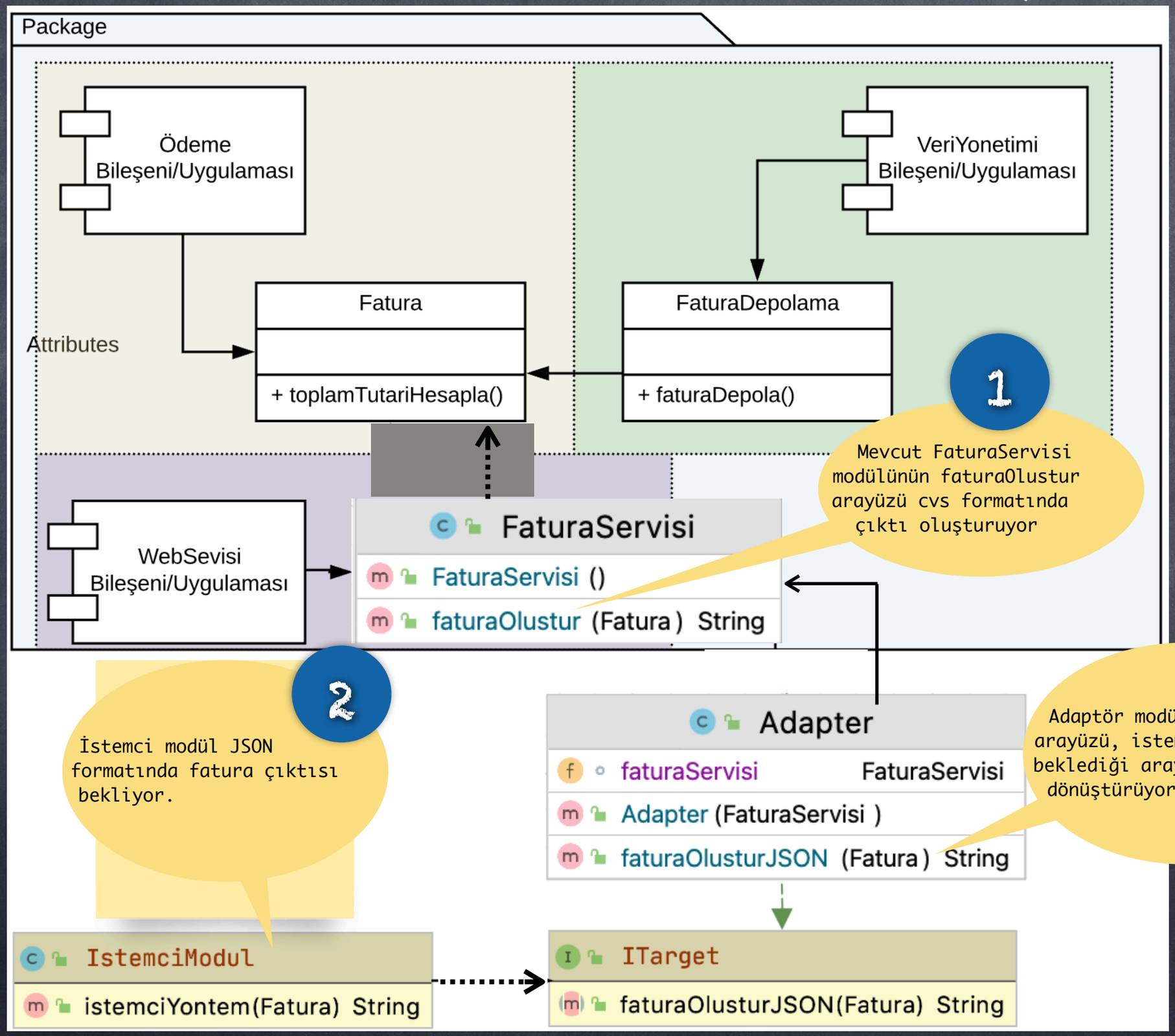



https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders10/

# Tasarım Desenleri : Adapter

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

# Tasarım Desenleri : Adapter

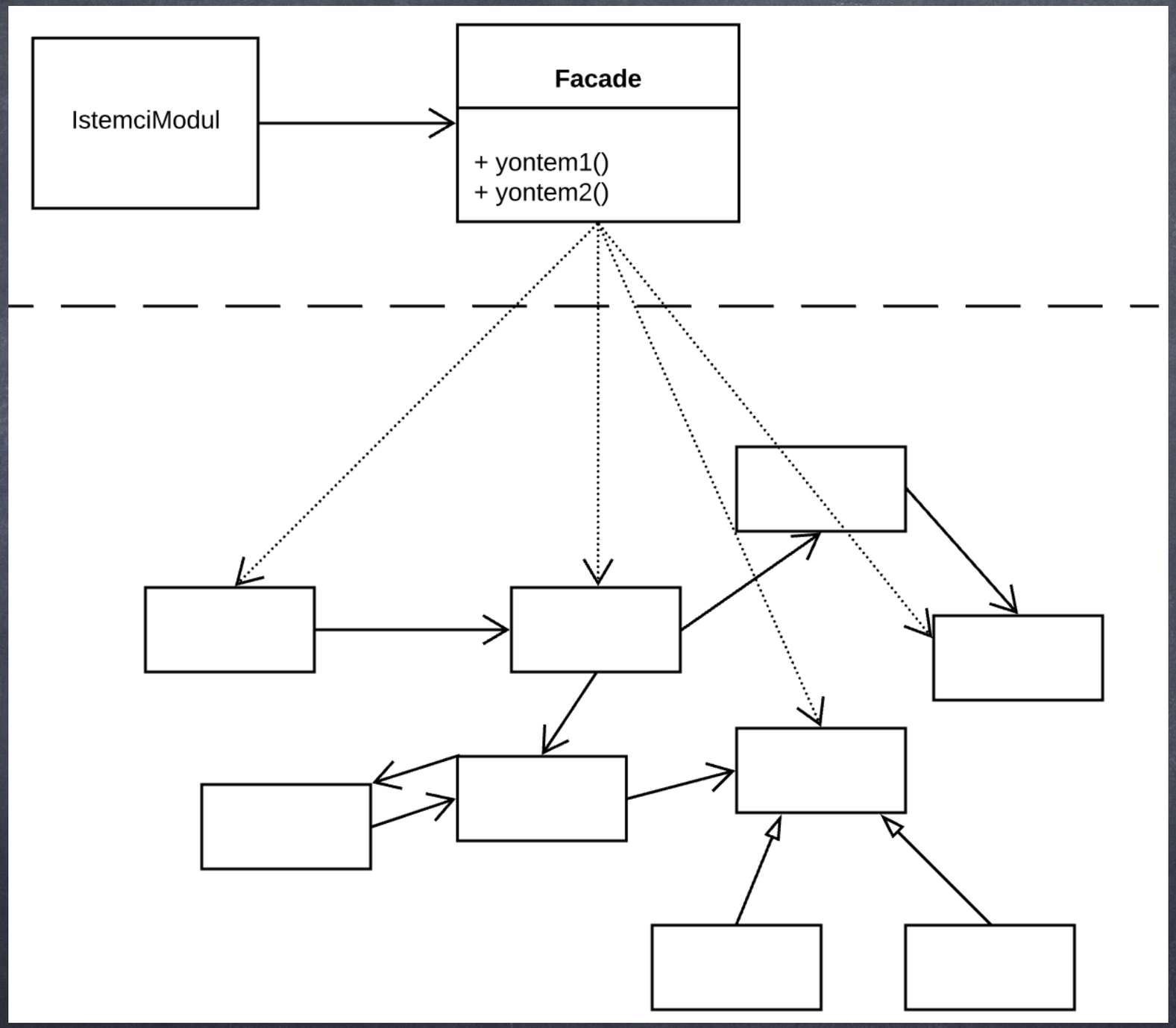



https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders11/

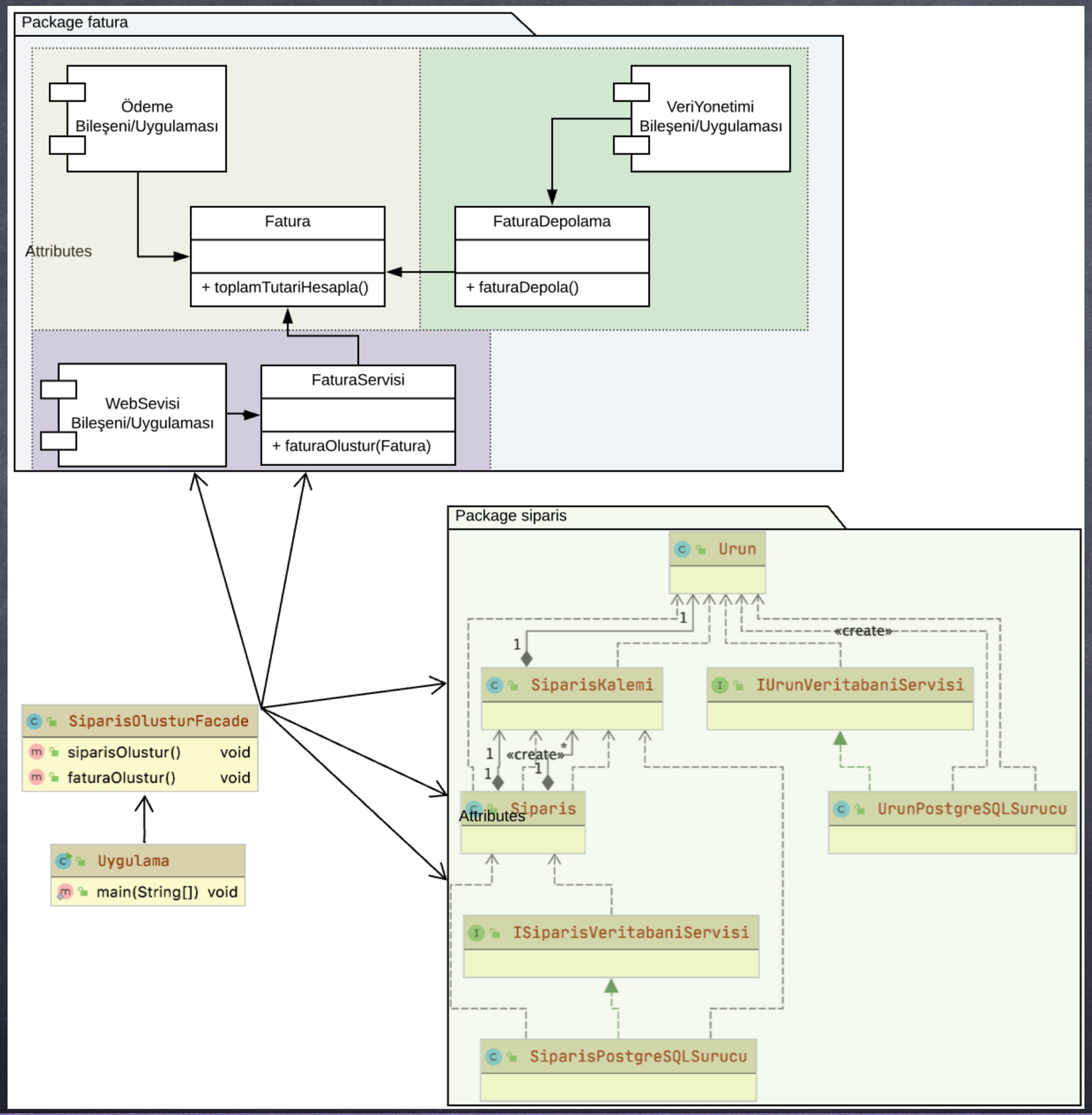
# Tasarım Desenleri : Facade

* Yapısal desenlerden biridir.
* Karmaşık yapıdaki bir sınıf topluluğu (kütüphane, alt bileşen, eskiden yazılmış kodlar vb.) için basitleştirilmiş arayüz sağlar.
* İstemci kodun karmaşık alt sistemle etkileşimini 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

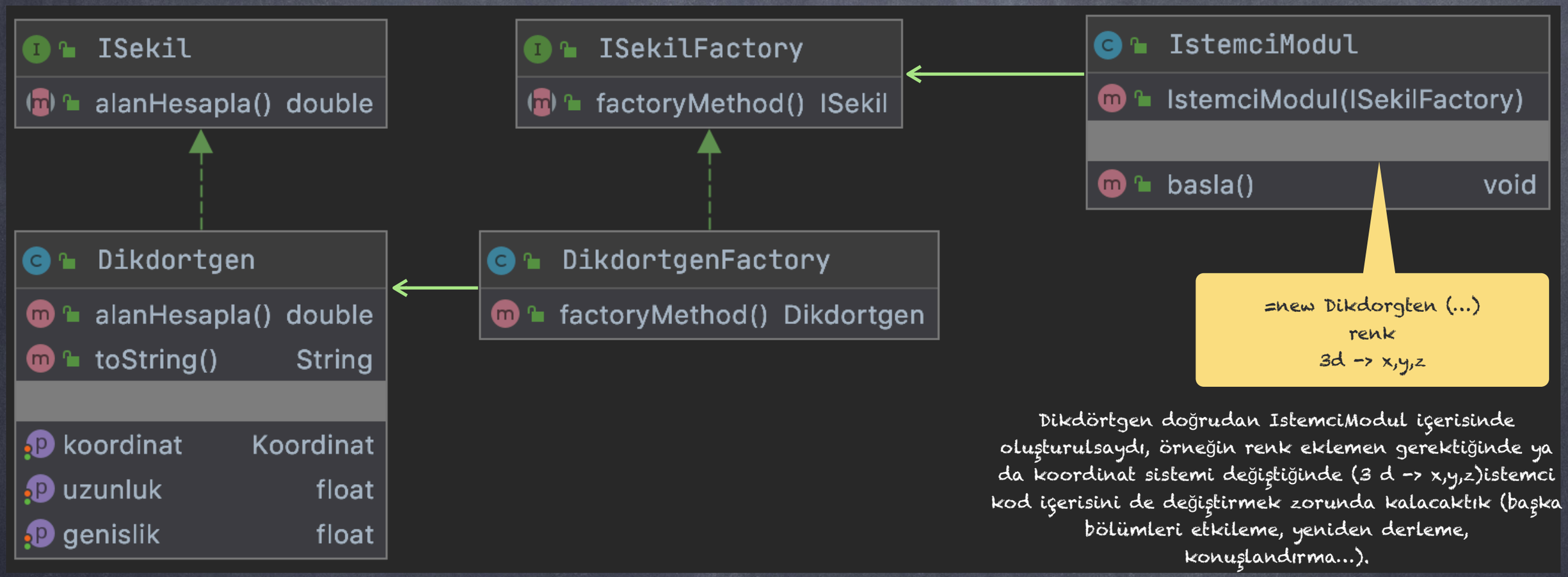



https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders11/

# Tasarım Desenleri-Factory Method

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

# Tasarım Desenleri-Factory Method (uygulama1)

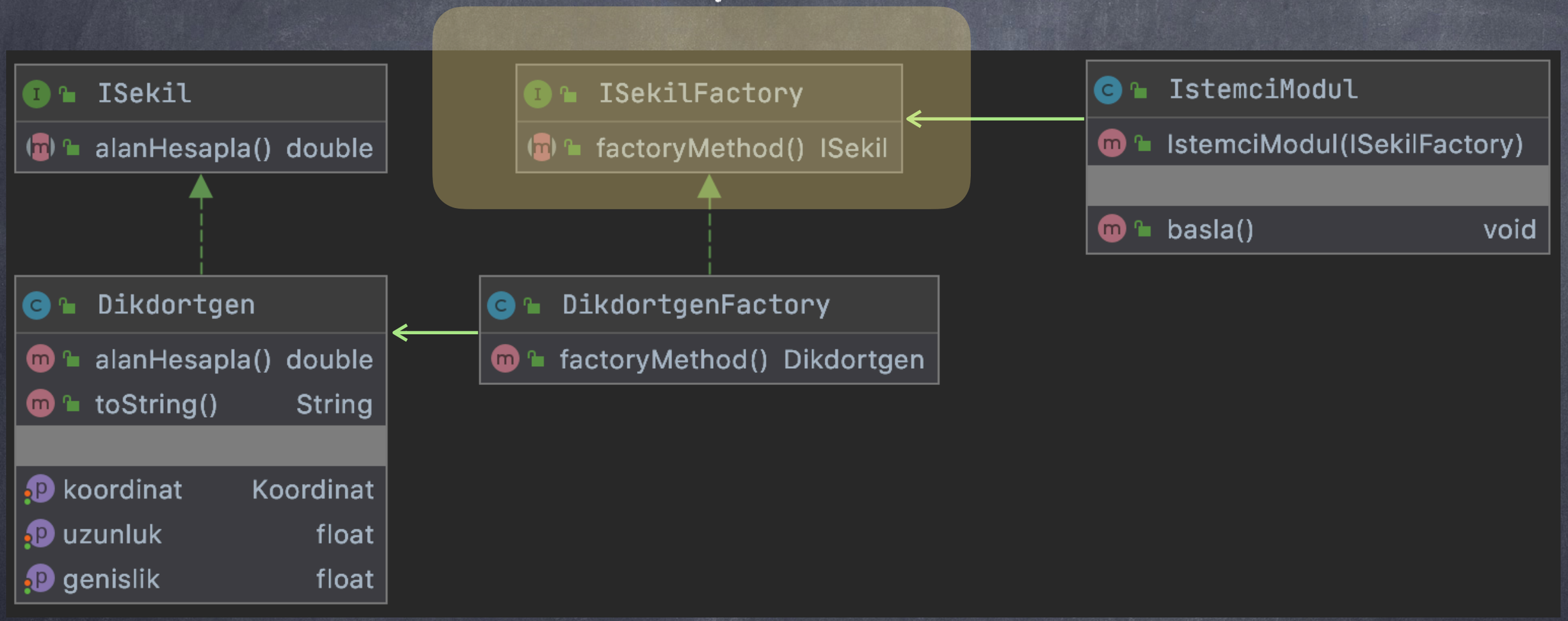



Dikdörtgen doğrudan IstemciModul içerisinde oluşturulsaydı, örneğin renk eklemen gerektiğinde ya da koordinat sistemi değiştiğinde (3 d -> x,y,z)istemci kod içerisini de değiştirmek zorunda kalacaktık (başka bölümleri etkileme, yeniden derleme, konuşlandırma...).

https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders11/

# Tasarım Desenleri-Factory Method (uygulama1)

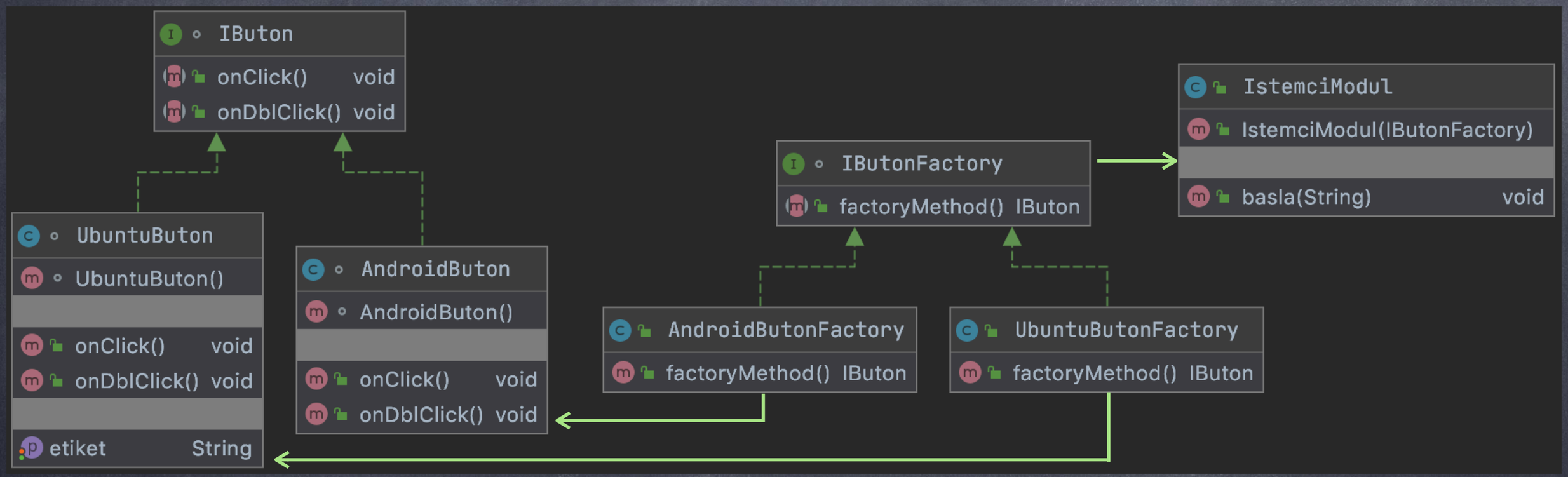
1. İstemci modül içerisindeki şekil dikdörtgen olacaksa ve değişme ihtimali yoksa DI uygulamaya gerek yok.




Soru1: ATM uygulamasında Banka Bilgi Sistemi nesnesi oluşturma işlemini "factory method" deseniyle yapınız.

https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders11/

# Tasarım Desenleri-Factory Method (uygulama2)




https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders11/

# Nesne Yönelimli Analiz ve Tasarım

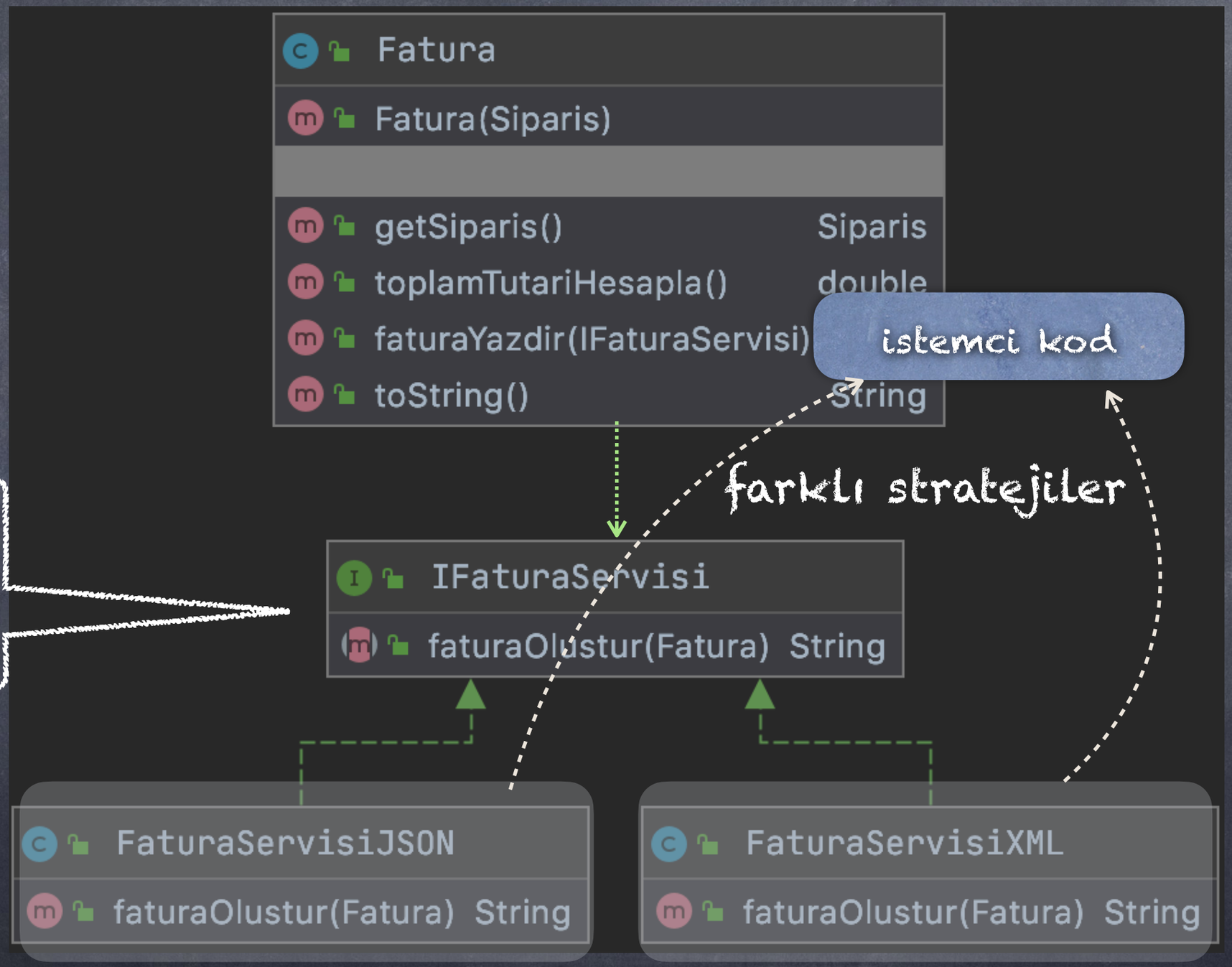
## Nesne Tasarımı

Tasarım Desenleri
Design Patterns

# Tasarım Desenleri : Strategy

* Davranışsal desenlerden biridir.
* Aynı istemci kodun, farklı algoritmaları/stratejileri desteklemesini sağlar.
* Alternatif algoritmalardan uygun olanının, çalışma zamanında seçilmesi gereken durumlarda kullanılabilir.
* Örneğin; geliştirdiğiniz uygulama "buble sort" algoritmasını kullanıyorken, istemci kodu değiştirmeden, bu algoritma yerine "quick sort" algoritmasını çalıştırmasını sağlamak isterseniz, bu deseni kullanabilirsiniz.
* Farklı algoritmalar/stratejiler soyut bir modülden (arayüz) türetilir/gerçeklenir.
* İstemci kod içerisinde, bu algoritmalar yerine soyut modül kullanılır (program to interface...)

# Tasarım Desenleri : Strategy

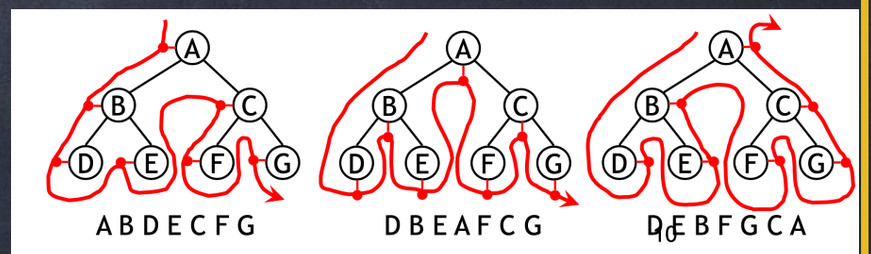



OCP
Open for extension,
Closed for
modification

https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders12

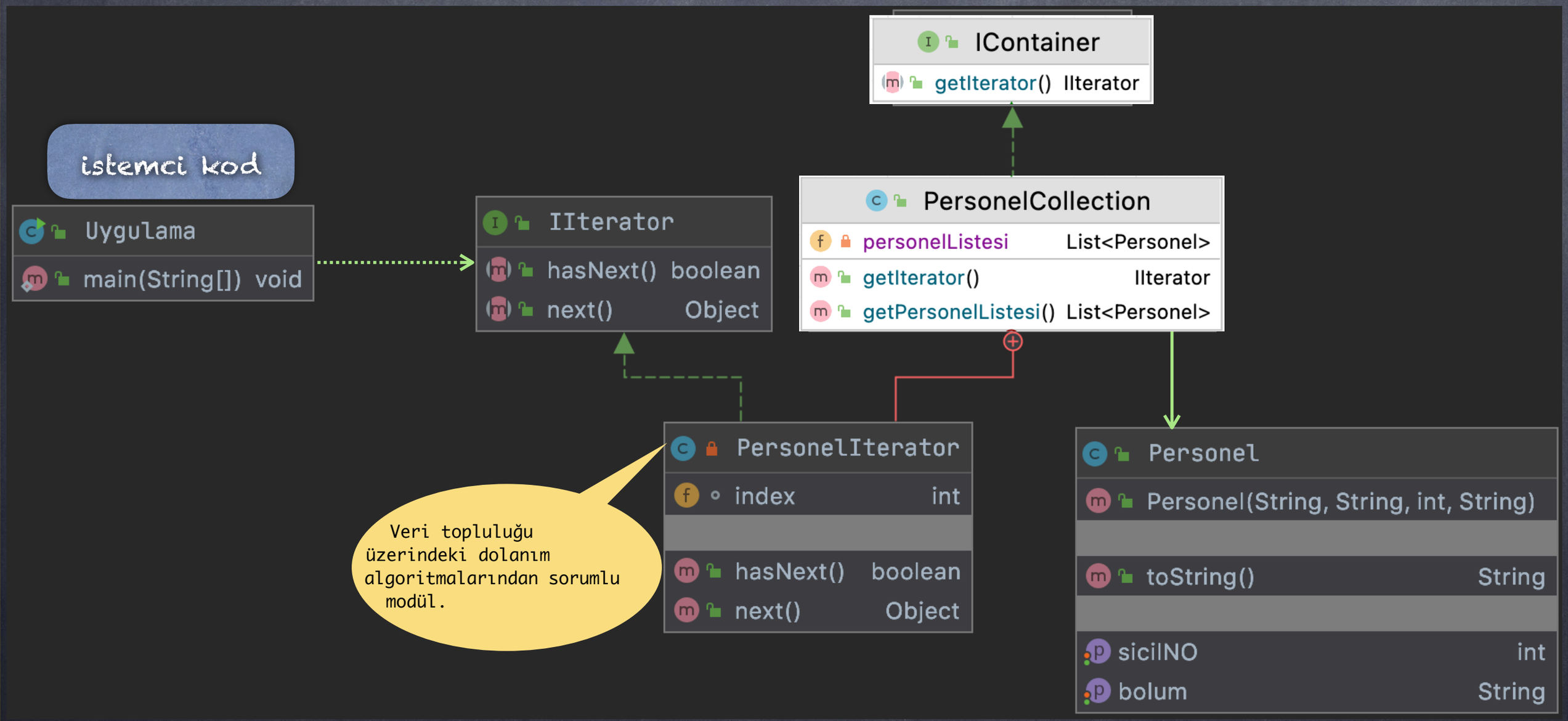
# Tasarım Desenleri : Iterator

* Davranışsal desenlerden biridir.
* Veri toplulukları (collections) içerisinde elementler (nesneler) bulunur ve bu elementler çeşitli veri yapıları (dizi, bağlı liste, ağaç, graf vb.) kullanılarak bir arada tutulur.
* Veri toplulukları içerisindeki her bir elemente erişilmesi gerekir. Bu işleme dolaşım (traversal) denir.
* Veri toplulukları için kullanılan veri yapıları basit olduğunda (dizi, liste vb.) dolaşım için kullanılacak algoritma basit bir şekilde gerçekleştirilebilir.
* Kullanılan veri yapıları karmaşıklaştıkça dolaşım algoritmaları zorlaşabilir ve zaman zaman farklı dolaşım algoritmalarına (ağaç veri yapısı için preorder, postorder, inorder gibi) ihtiyaç duyulabilir.
* Iterator deseni, istemci modülün, veri topluluğu içerisinde kullanılacak dolaşım algoritmalarından etkilenmesini önlemek için, bu işlemi (sorumluluk) başka bir nesneye (iterator) yaptırır (SRP gereği).




www.cis.upenn.edu/~matuszek/cit594-2008 adresinden alınmıştır.

# Tasarım Desenleri : Iterator




https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders12

# Tasarım Desenleri : Proxy

* Yapısal desenlerden biridir.
* Sınıfın fonksiyonlarını (arayüzünü) değiştirmeden davranışını değiştirmek istediğimizde kullanabiliriz.
* Sınıfı değiştirmek, onu kullanan diğer sınıfların etkilenmesine neden olabilir.

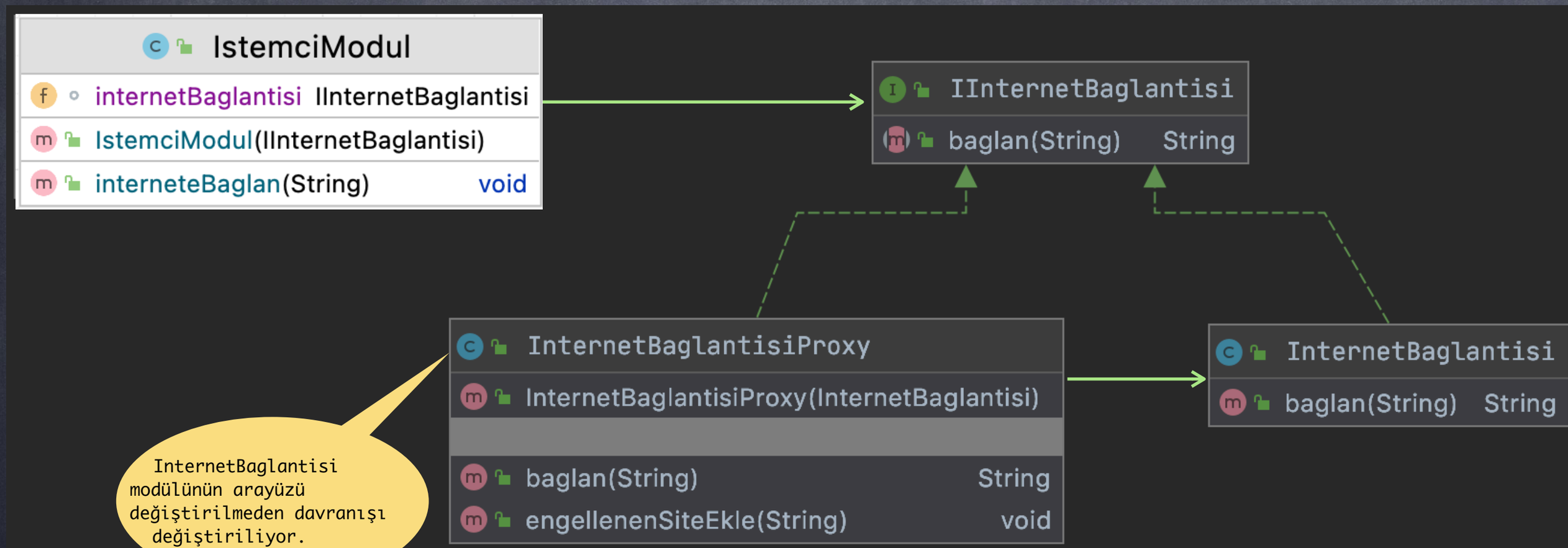
Örnek kullanım durumları:

Bir nesneye erişimi kontrol etmek, güvenlik veya kimlik doğrulama sağlamak.

Performansı artırmak için sık kullanılan veya maliyetli işlemleri önbelleğe alma.

Tembel (lazy) başlatma işlemini uygulayarak gerçek nesneyi sadece gerektiğinde oluşturma.

# Tasarım Desenleri : Proxy




https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders12

# Tasarım Desenleri : Builder

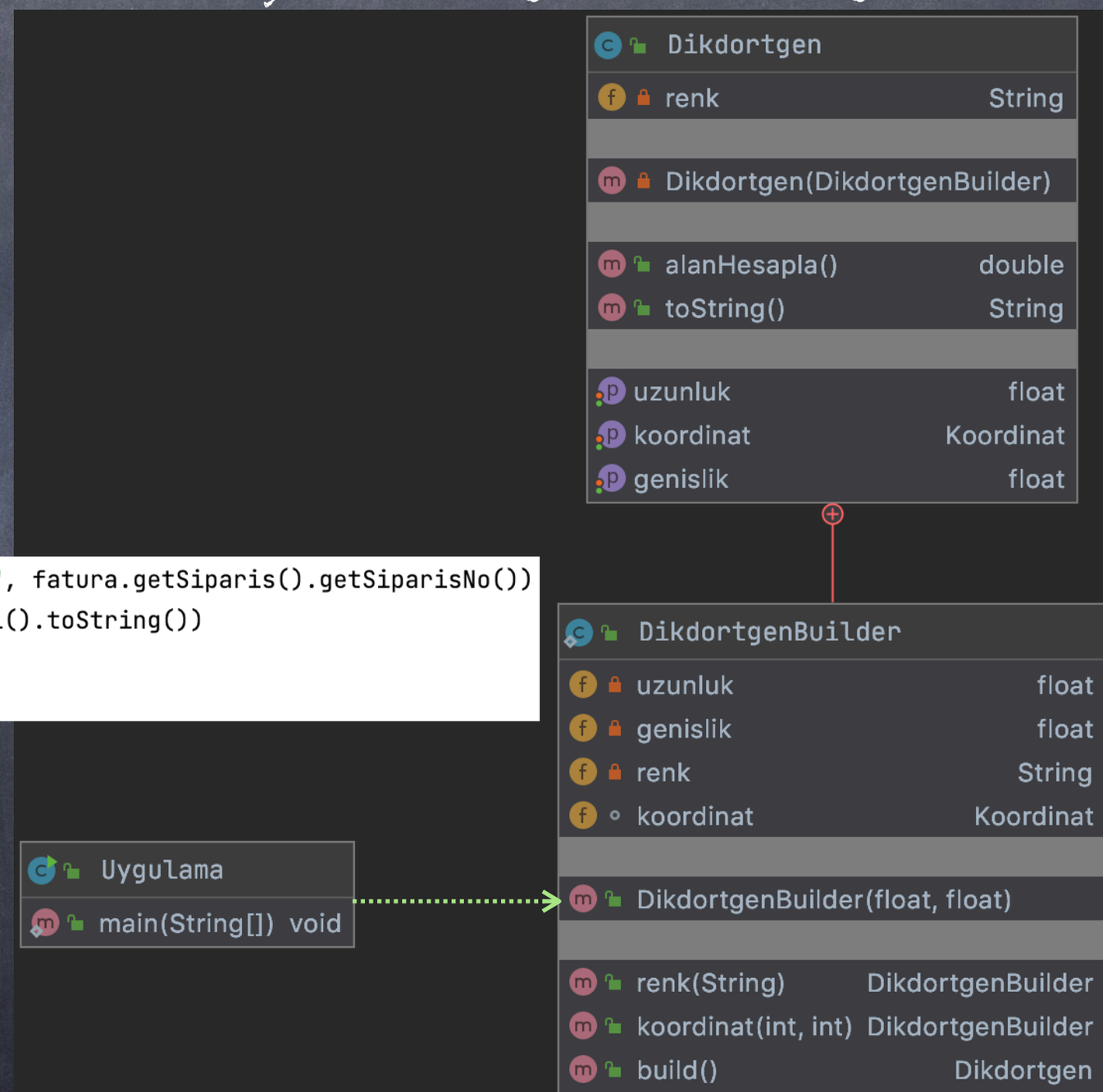
* Nesne oluşturmayla (creational) ilgili desenlerden biridir.
* Karmaşık nesnelerin (içerisinde çok sayıda üye değişken ve üye nesne olan) oluşturulması için kullanılır.
* Karmaşık bir nesnenin yapımını, temsilinden (sunumundan) ayırır. Böylece, aynı yapım süreci farklı temsiller oluşturabilir.
* Nesnelerin farklı temsillerinin (sunumunlarının) her biri için ayrı ayrı yapıcı tanımlamak yerine, nesne oluşturma işini adım adım gerçekleştiren "builder" deseni kullanılabilir.
  * Böylece nesne oluşturma işi nesnenin kendisinden (yapıcı) ayrılmış olur (SRP).
  * Nesne oluşturma işlemi istemci koddan ayrılmış olur (SRP, loosly coupling)

# Tasarım Desenleri : Builder

* Builder sınıfı nesnenin tüm üye değişkenlerini/nesnelerini içermelidir
* Nesne oluşturmak gerektiğinde, builder sınıfının (static olmalı) nesnenin ilgili özelliklerine ilk değer ataması yapan yöntemleri sırasıyla çağrılır.
* Çağrılan son yöntem (build) Dikdörtgen nesnesini oluşturur. Nesne oluşturmak için, nesnenin varsayılan yapıcısına, nesnenin ilgili üyelerinin değerlerini içeren DikdortgenBuilder sınıfı gönderilir.

```
Calendar cal = new Calendar.Builder()
        .setCalendarType("iso8601")
        .setDate(2013, 1, 22).build();
```

```
JsonObject faturaJSON = Json.createObjectBuilder().add("siparisNo", fatura.getSiparis().getSiparisNo())
        .add("siparisTarihi", fatura.getSiparis().getSiparisTarihi().toString())
        .add("toplam", fatura.getSiparis().getGenelToplam())
        .build();
```

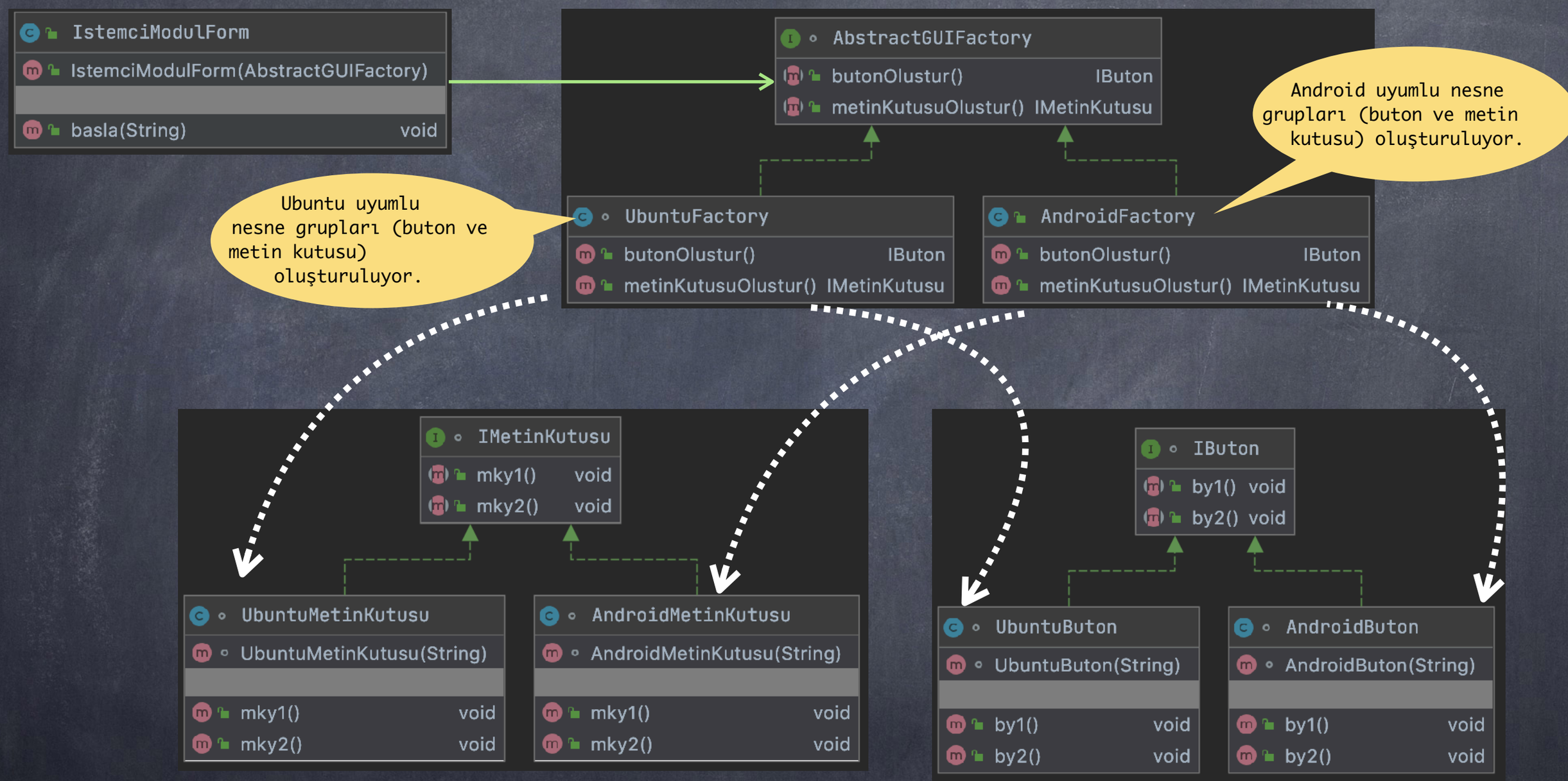



https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders12

# Tasarım Desenleri : Abstract Factory

* Nesne oluşturmayla (creational) ilgili desenlerden biridir.
* İstemci modül içerisinde, somut sınıflarını belirtmeden, birbirleriyle ilgili ya da birbirlerine bağlı nesne aileleri/ grupları oluşturmak için kullanılabilir.
* Örneğin; bir geliştirme ortamının görünümünü değiştirmek istediğimizde ya da bir yazılımın farklı platformlarda sorunsuz çalışabilmesini istediğimizde, bu deseni kullanabiliriz.

# Tasarım Desenleri : Abstract Factory

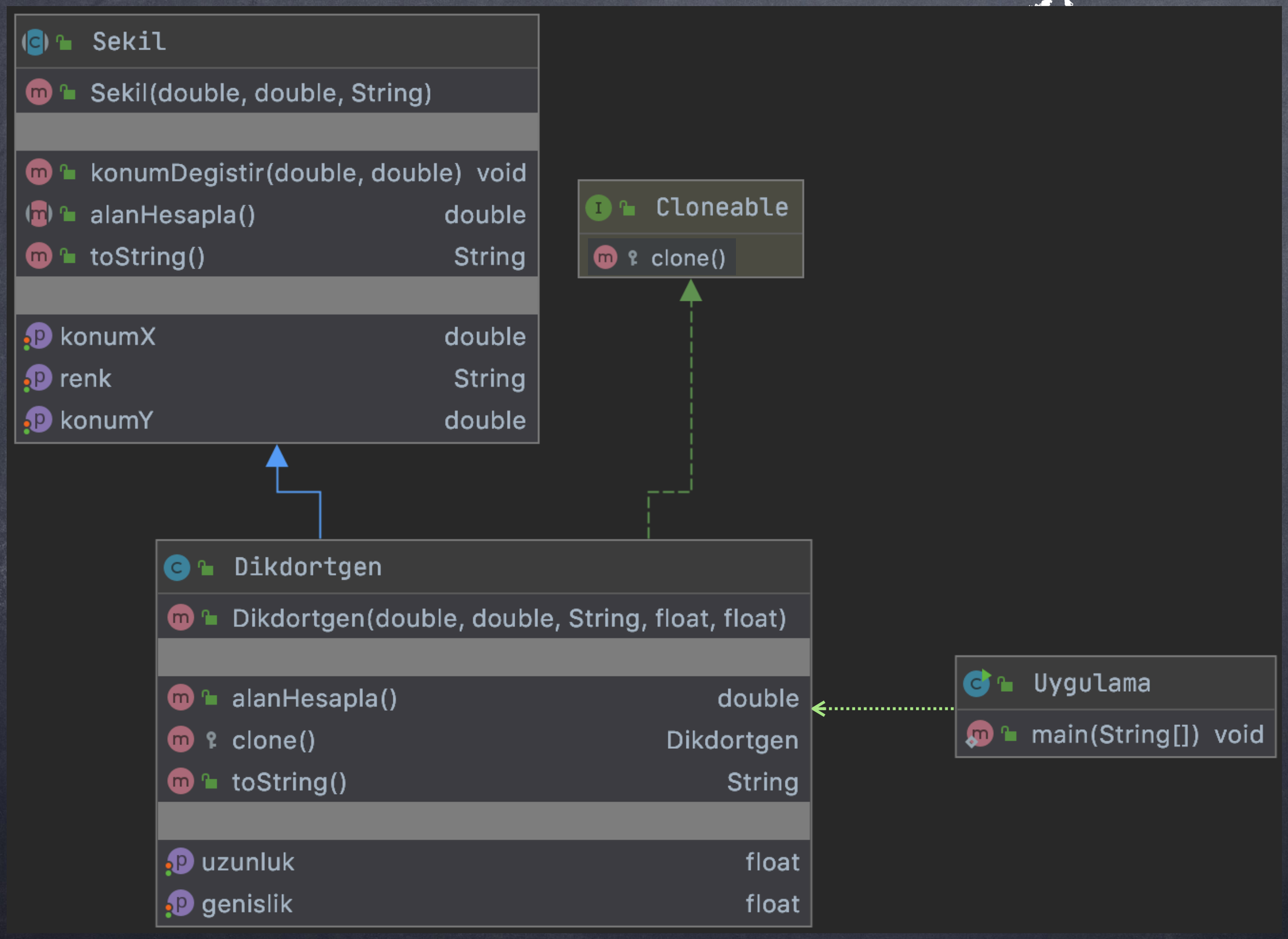



https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders12

# Tasarım Desenleri : Prototype

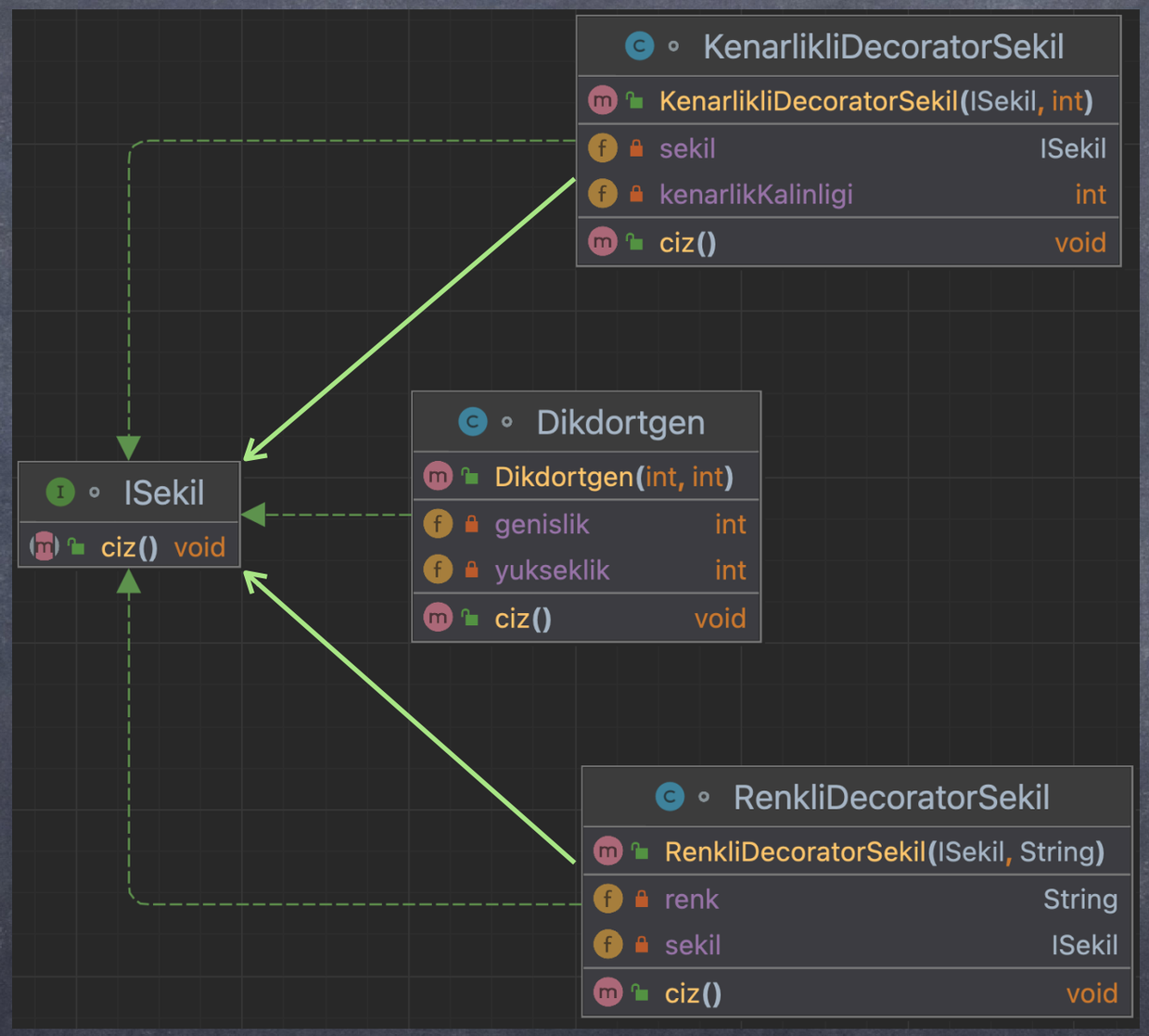
* Nesne oluşturmayla (creational) ilgili desenlerden biridir.
* Yeni nesne oluşturma işleminin maliyetli olduğu durumlarda, bu desen kullanılarak, mevcut nesnenin kopyası oluşturulabilir.
* Karmaşık işlemler sonucu oluşan nesneler (örneğin, veritabanı sorguları ya da diğer sistemlerden gelecek verilerle oluşturulacak nesneler) "cache" içerisinde saklanır ve bu nesnelere ihtiyaç duyulduğunda, önce "cache" içerisinde aranır. Bulunursa kopyası oluşturulur ve veritabanı sorgusuna gerek kalmaz.
* Nesneler, prototype (cloneable) arayüzü içerisinde tanımlı clone() yöntemini gerçekleyerek, (new komutu ile) kendi kopyalarını oluştururlar.

# Tasarım Desenleri : Prototype




https://github.com/celalceken/NesneYonelimliAnalizVeTasarimDersiUygulamalari/tree/master/Ders12

# Tasarım Desenleri : Decorator

- Yapısal desenlerden biridir.
- Aynı sınıftan oluşturulan mevcut nesneleri etkilemeden, nesnelere yeni davranışlar/fonksiyonlar eklemek gerektiğinde kullanılabilir.
- Sınıfı değiştirmek, onu kullanan diğer sınıfların etkilenmesine neden olabilir. Bu nedenle, nesneler özel sarıcı (wrapper) nesneler içerisine yerleştirilerek onlara yeni davranışlar kazandırılması sağlanabilir.
- Kalıtım ile davranış eklemenini kullanışsız olduğu yerlerde, bu işin "composition" ile gerçekleştirilmesi sağlanmış olur.

# Tasarım Desenleri : Decorator