Nesneye Yönelik Tasarım

İçerik

- Nesneye yönelik tasarım
  - Nesne ve sınıf kavramları
  - Tasarım için kullanılan başlıca UML elemanları
  - Üst düzey tasarım adımları
  - Detay tasarım adımları

Nesne ve Sınıf Kavramları

“Nesne” Nedir?

- İyi tanımlı bir kapsamı ve kimliği olan, belirli bir durum ve davranışı içeren, soyut veya somut varlıktır.
- Nesne gerçek dünyadaki somut bir varlığı temsil edebilir.

  Televizyon, motor, vb.
- Nesne tamamen kavramsal bir varlığı temsil edebilir.

  Banka hesabı, vb.

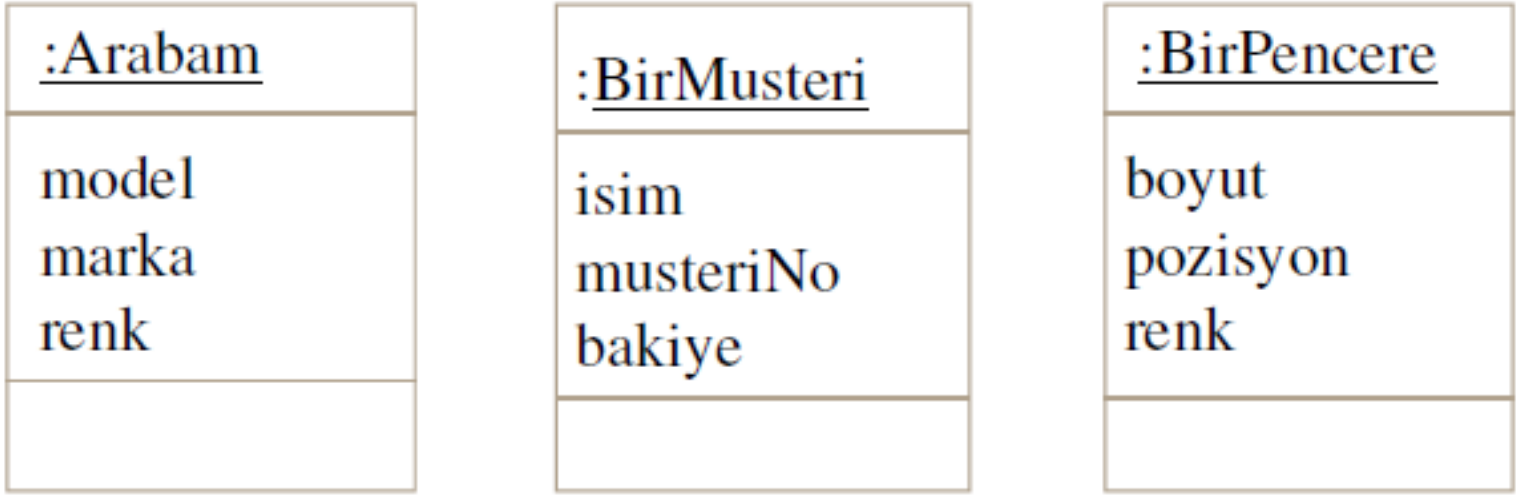
“Nesne”: Örnek

Müşteri Hesabı
miktar
paraÇek ParaYatır miktarSorgula

- Her nesne aşağıdakilere sahiptir:
  - Kişilik (“identity”)
  - Özellik (“attribute”)
  - Durum (“state”)
  - Davranış (“behavior”)
  - Sorumluluk (“responsibility”)

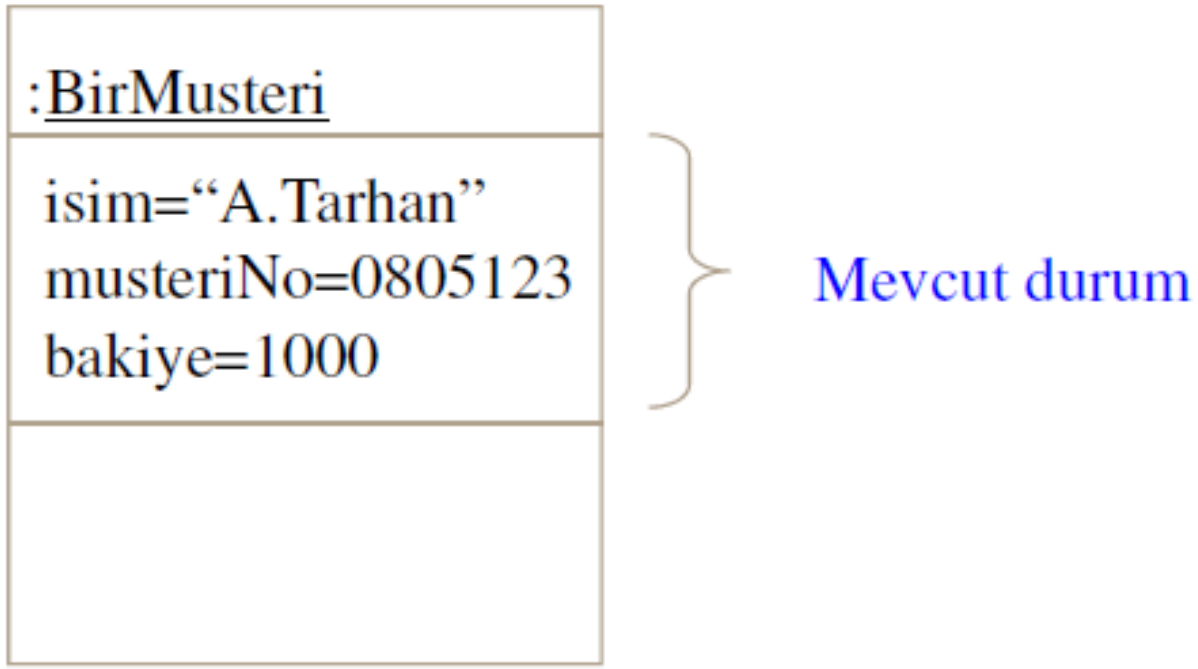
Özellik

- Her nesnenin kendine ait bir dizi özelliği vardır.
  - Özellikler nesneye ait verileri taşır.



Durum (1)

- Nesnenin tüm özellikleri ve bu özelliklerin o anki değerleri, nesnenin durumunu oluşturur.

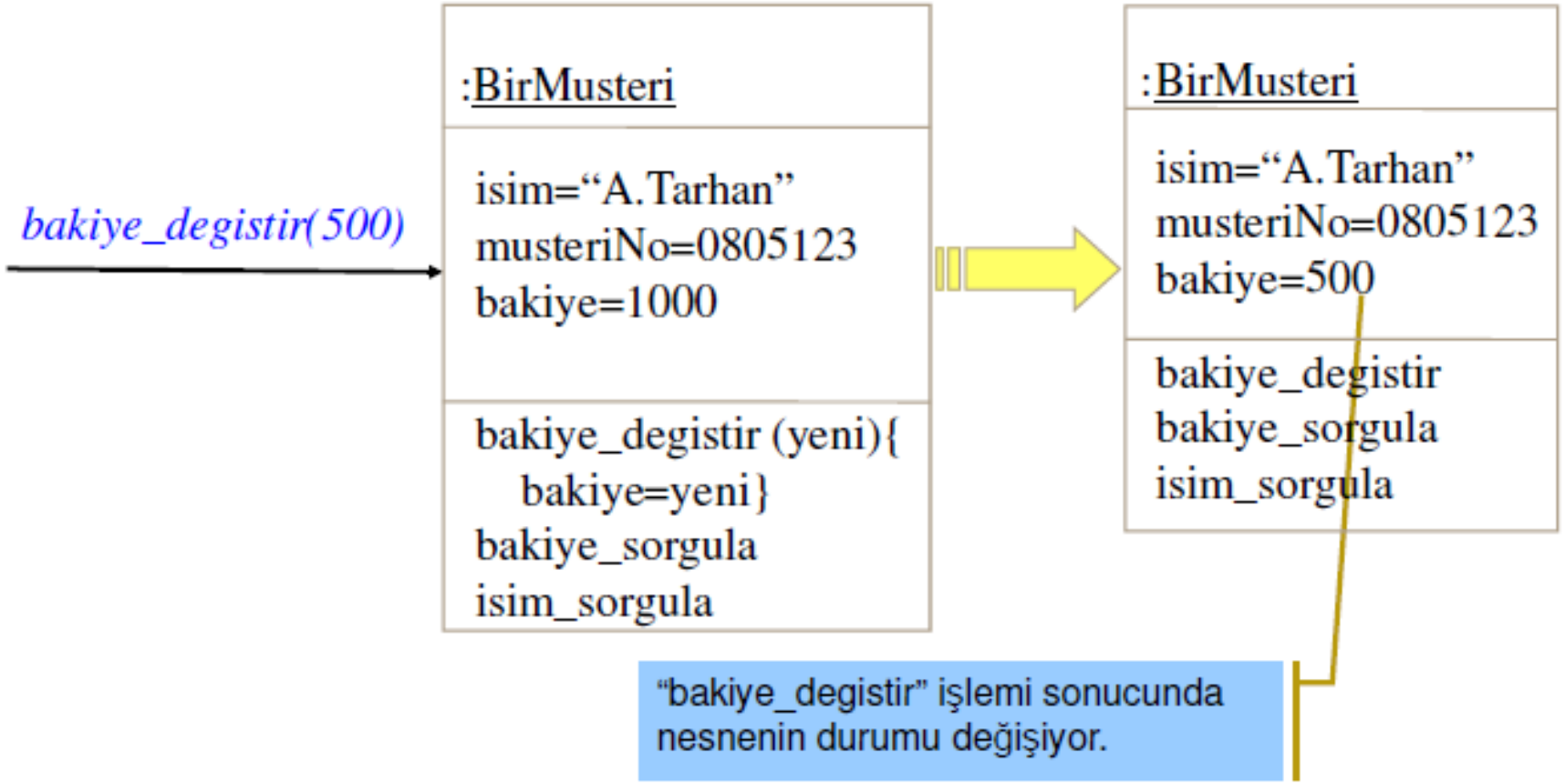


Davranış

- Her nesnenin iş yapabilmeyi sağlayan tanımlı bir davranış şekli vardır.
  - Nasıl davranır, nasıl tepki verir ? (“act”/“react”)
    - Nesnenin operasyonları
    - Görülebilir aktivite
    - Diğer nesneler tarafından kullanılan arayüz (“public interface-operations”)
  - Davranışın gerçekleştirilmesi bilgisayar kodu ile yapılır.
    - Kodlanan nesne yordamları, davranışı gerçekleştirir.
    - Gizli gerçekleştirme (“encapsulated implementation”)

Durum (2)

- İşlemler sonucunda nesnenin durumu değişir.



Sorumluluk

- Nesnenin sorumluluğu tüm sistemin işlevselliğine nasıl katkıda bulunacağını tanımlar.
  - Nesnenin sistemde oynayacağı rol
  - Neden böyle bir nesneye gereksinim var ?
  - Özellikler ve davranışlar, birlikte nesnenin sorumluluklarını yerine getirmesini sağlar.

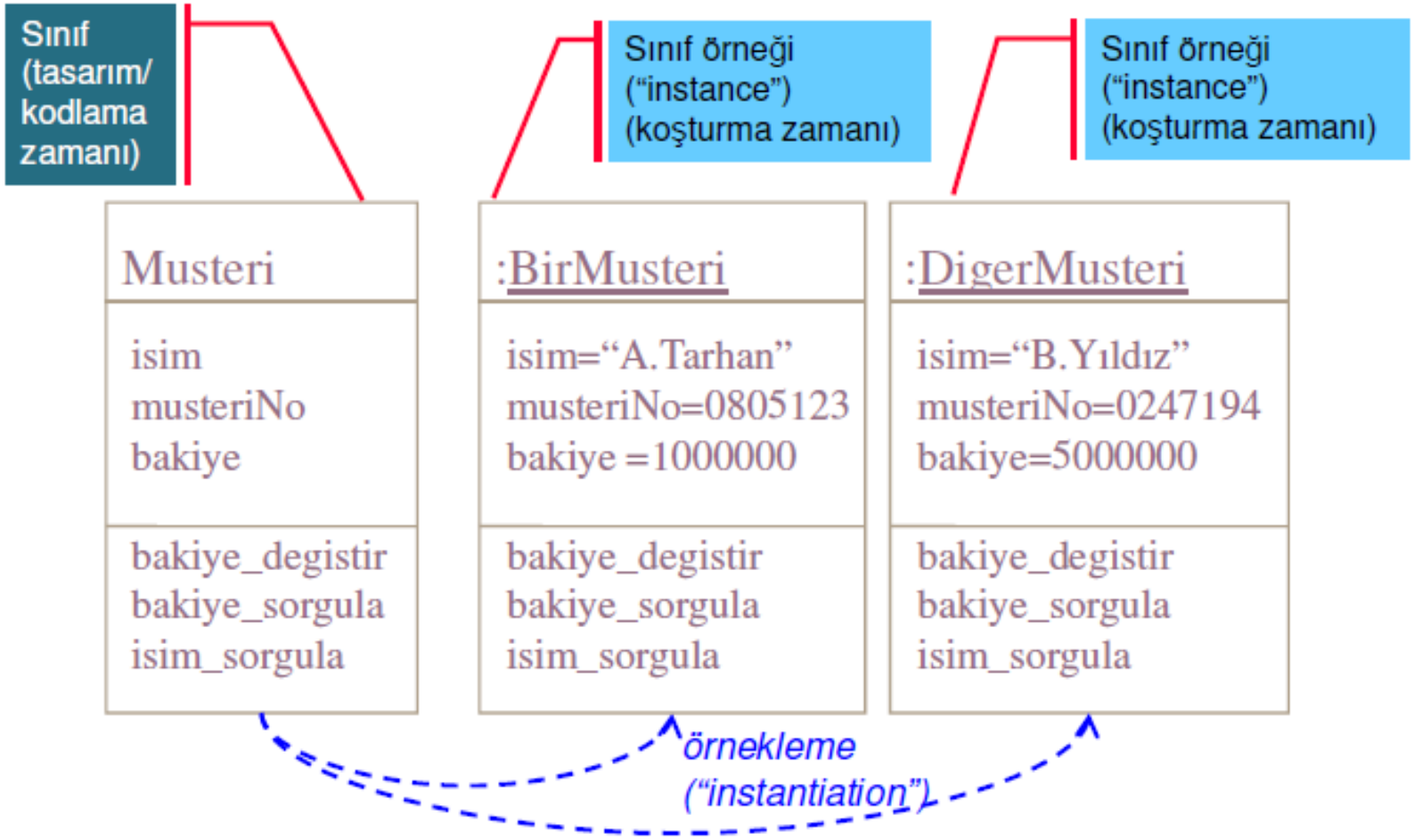
İlişki(Relationship)

- Nesneler arasındaki fiziksel veya kavramsal bağlantı
- En yaygın ilişki: Bir nesne diğer nesnenin sunduğu servislerden yararlanır.
  - Mesaj iletimi
  - Müşteri/tedarikçi ilişkisi
- Sistem işlevselliğini sağlamak amacıyla, nesneler birbirleriyle ilişkiler aracılığı ile işbirliği yaparlar.
  - Nesneye yönelik programlama modeli

Sınıf Nedir?

- Yapısal ve/veya davranışsal olarak aynı özelliklere sahip nesneler SINIF altında gruplanır.
  - Her nesne bir sınıfın örneğidir (“instance”).
  - Sınıfları tanımlar ve nesneleri sınıf tanımından örnekleriz (“instantiation”).
  - Her nesne ait olduğu sınıfı bilir.
- Sınıf, nesneler için şablon tanımdır.
  - Özellikler ve yordamlar sınıf için yalnızca bir kez tanımlanır.
- Sınıfların birbirleri arasındaki ilişkiler sistemin sınıf yapısını (“class structure”) oluşturur.
  - İki sınıf arasında ilişki varsa karşılık gelen nesneler arasında da vardır.

Örnek bir Sınıf



Sınıf ve Nesne

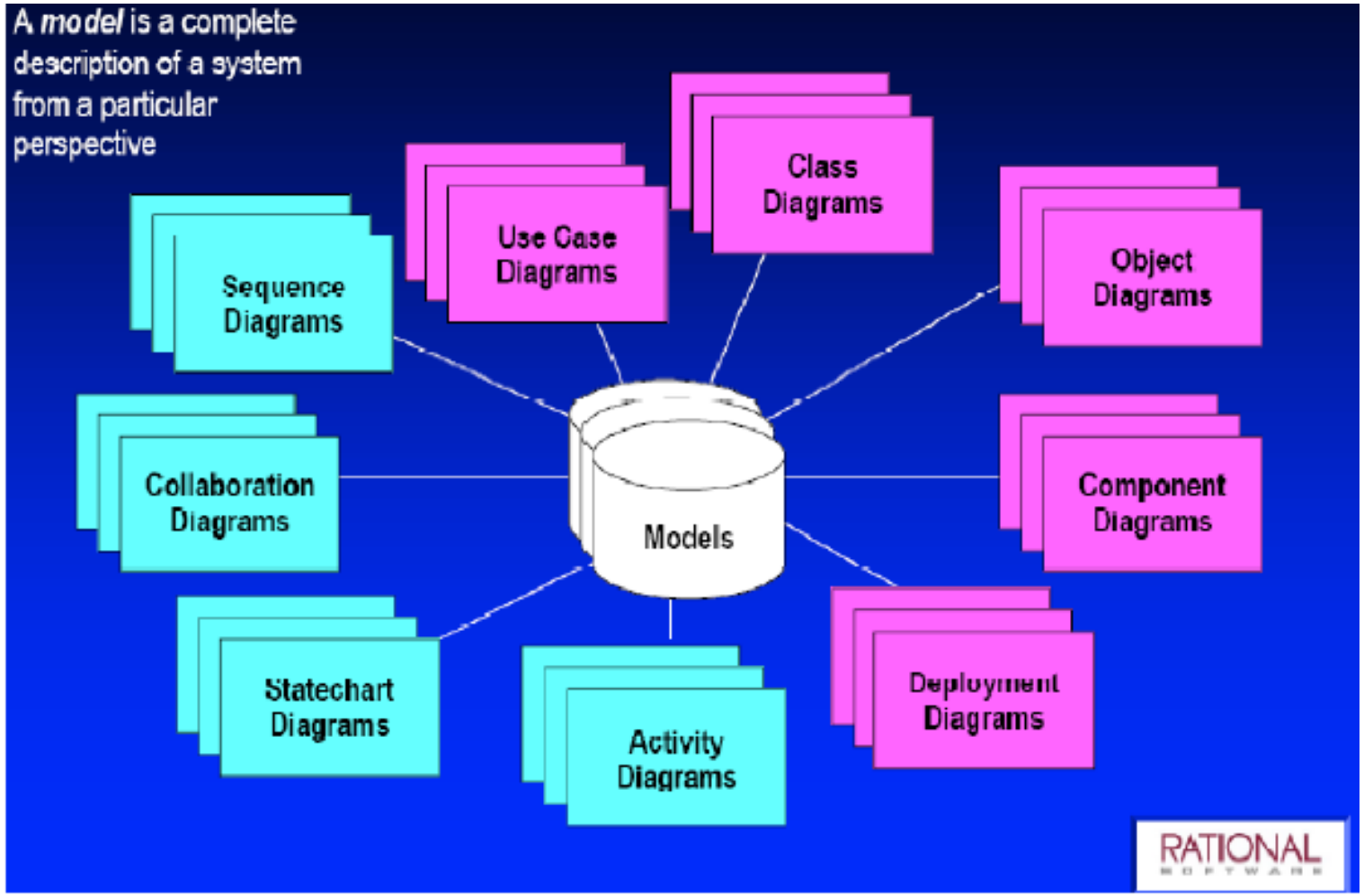
- Her sınıfın sıfır veya daha fazla örneği vardır.

- Sınıf statik, nesne dinamiktir.
  - Sınıfın varlığı, semantiği ve ilişkileri program koşturulmadan önce sabit olarak belirlenmiştir.
  - Nesneler program koşturulduğunda sınıf tanımından dinamik olarak yaratılırlar (“construction”).
  - Nesneler sorumluluklarını tamamladıklarında ortadan kaldırılırlar (“destruction”).

- Nesnenin sınıfı sabittir ve nesne bir kez yaratıldıktan sonra değiştirilemez.

Nesneye Yönelik Tasarım İçin Kullanılan Başlıca UML Elemanları

UML Diyagramları



Sınıf (Class) Diyagramı

- Sınıflar ve sınıfların arasındaki ilişkilerin modellendiği diyagramdır.
- Sınıf diyagramı statik bir diyagramdır.
- Sınıf diyagramının amacı şu şekilde özetlenebilir:
  - ✓Bir uygulamanın statik görünümünün analizi ve tasarımı.
  - ✓Bir sistemin sorumluluklarını açıklayın.
  - ✓Bileşen ve dağıtım diyagramları için temel.
  - ✓İleri ve geri mühendislik.

Sınıf (Class) Diyagramı

Araba
+TekerlekSayisi : byte = 4 -Model : string
+ArizaKontrol() : bool +ArizaKontrol(içeri Arac : Araba) : bool +Init()

- UML de sınıf tanımlama dikdörtgen şekli ile gösterilir. Dikdörtgenin en üstünde sınıfın adı, altında özellikleri ve onunda altında fonksiyonlar gösterilir.

- Araba sınıfımızın adıdır. TekerlekSayisi ve Model sınıfın özellikleridir.
- Özelliğin tipi :byte ve :string şeklinde belirtilmiştir.
- TekerlekSayisi özelliğinin default değeri 4 olarak belirtilmiştir.
- Özellik ve fonksiyonların başındaki + ve – işaretleri de erişim seviyesini belirtir.

Sınıf (Class) Diyagramı

Araba
+TekerlekSayisi : byte = 4 -Model : string
+ArizaKontrol() : bool +ArizaKontrol(içeri Arac : Araba) : bool +Init()

- + işareti public (diğer sınıflardan da erişilebilir) seviyesinde olduğunu gösterir.
- # işareti protected (sınıf içinden ve kendisinden türeyen sınıflardan erişilebilir) seviyesinde olduğunu gösterir.
- - işareti private (sadece sınıf içinden erişilebilir) seviyede olduğunu belirtir.

- Araba sınıfında 3 fonksiyon tanımlanmıştır.
- 1. ArizaKontrol fonksiyonu geriye bool değer döndürür ve parametre almaz.
- 2. ArizaKontrol fonksiyonu ise Araba sınıfı parametre alır ve geriye bool değer döndürür.
- Init fonksiyonu ise parametre almaz ve geriye değer döndürmez.

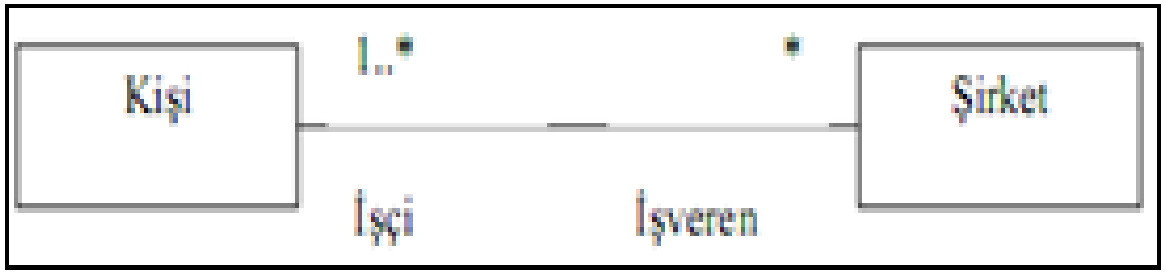
Sınıf (Class) Diyagramı

- Sınıf diyagramları kendi başlarına bir şey ifade etmez ancak aralarındaki ilişkilerle birlikte incelendiklerinde anlamlıdır.
- UML içerisinde sınıflar arasında 4 farklı ilişki tanımlanabilir:
  - ✓Bağlantı İlişkisi (Association)
  - ✓Genelleme/Kalıtım İlişkisi (Generalization/Inheritance)
  - ✓Bağımlılık İlişkisi (Dependency) (Aggregation, Composition)
  - ✓Gerçekleştirim İlişkisi (Realization)

Sınıf (Class) Diyagramı-Bağlantı İlişkisi (Association)

- Bağlantı ilişkisi, sınıf diyagramlarında en çok kullanılan ve en basit ilişki türüdür.
- Bağıntı ilişkisi iki nesne arasına çizilen düz çizgi ile belirtilir.
- Bağıntı ilişkileri için tanımlanmış bilgiler aşağıdaki gibidir;
  - ✓Bağıntının Adı
  - ✓Sınıfın bağıntıdaki rolü
  - ✓Bağıntının çokluğu
- Bağıntı adı, iki sınıf arasındaki ilişkinin küçük bir açıklamasıdır. Bu açıklama ile birlikte yön bilgisi de belirtilebilir.

Sınıf (Class) Diyagramı-Bağlantı İlişkisi (Association)



- işçi-işveren ilişkisinde bir şirketin en az bir işçisi olduğu (1..*), bir işçinin ise 0 ya da herhangi bir sayıda(*) şirkette işçi olarak çalışmış olabileceği ifade edilmektedir.

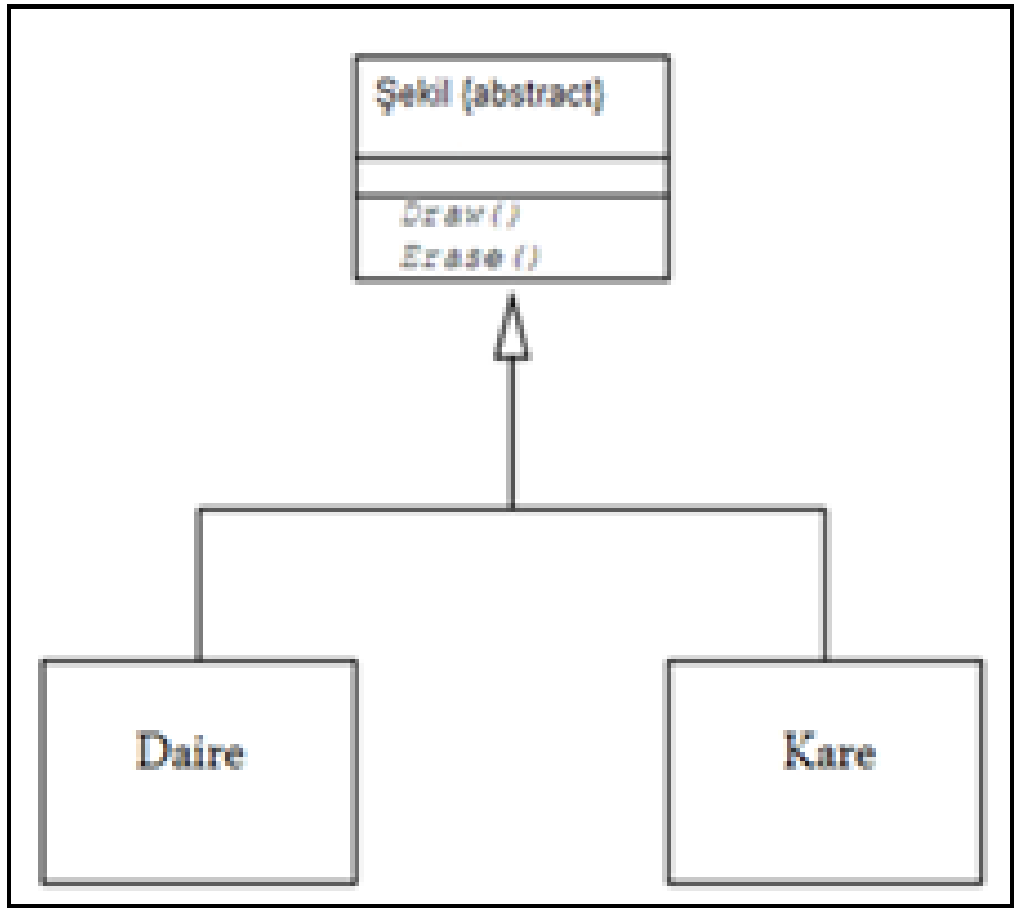
- En temel bağıntı ilişki tipleri aşağıdaki gibi listelenebilir;
- Bire-bir
- Bire-çok
- Bire-bir veya daha fazla
- Bire-sıfır veya bir
- Bire-sınırlı aralık (mesela:bire-[0,20] aralığı)
- Bire-n (UML de birden çok ifadesini kullanmak için '*' simgesi kullanılır.)
- Bire-Beş yada Bire-sekiz

Türetme (Inheritance) ve Genelleme (Generalization) İlişkisi

- Nesne tabanlı programlama tekniğinin en önemli parçası türetme (inheritance) dir.
- Türetme yoluyla bir sınıf başka bir sınıfın var olan özelliklerini alarak, o sınıf türünden başka bir nesneymiş gibi kullanılabilir.
- Bir sınıfın işlevleri türetme yoluyla genişletilecekse, türetmenin yapılacağı sınıfa taban sınıf (base class), türetilmiş olan sınıfa da türemiş sınıf (derived class) denir. Şekilsel olarak türemiş sınıftan taban sınıfa bir ok olarak belirtilir.

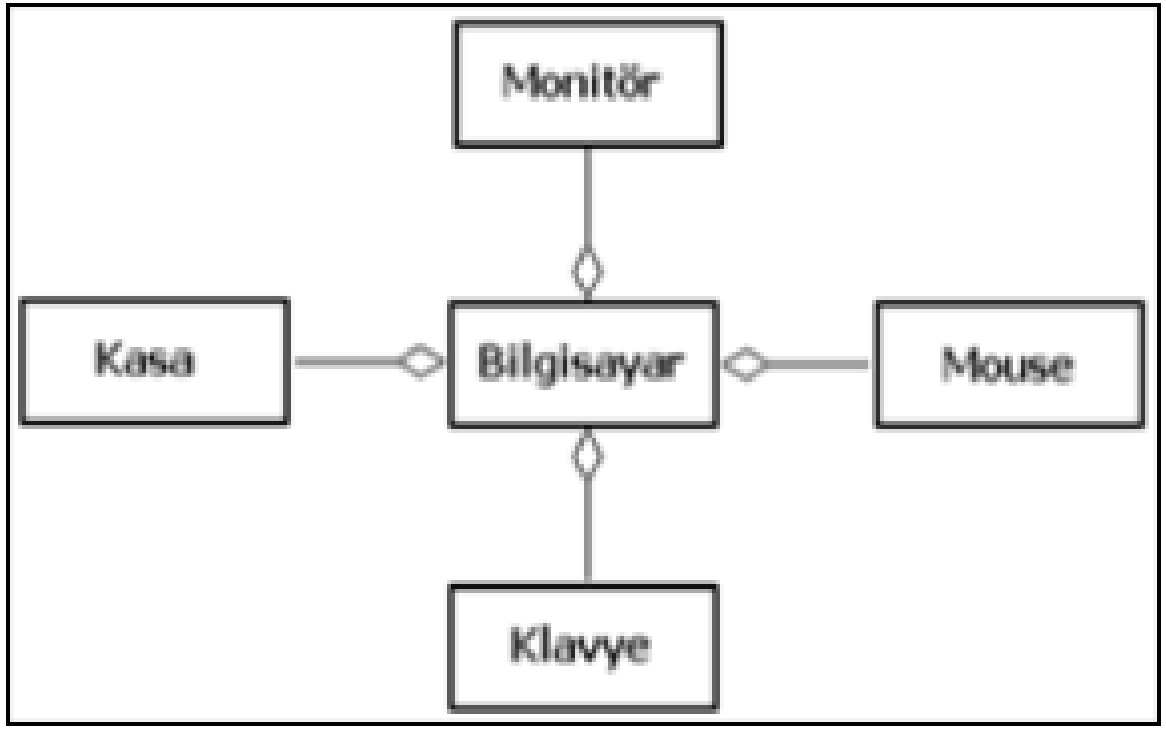
Türetme (Inheritance) ve Genelleme (Generalization) İlişkisi

- Türetme ağacında daire ve kare birer şekildir.
- Sınıflar arasındaki türetme işlemi ucu açık üçgen ve düz bir çizgiyle gösterilir. Bu durumda "Şekil" sınıfı ana sınıf (parent class), daire ve kare ise yavru (sub class) sınıflardır. Türetmeye sınıflar arası ilişki açısından baktığımızda türetmenin "is kind of" (bir çeşit) ilişkisinin olduğu görülür.



Bağımlılık İlişkisi (Dependency) (Aggregation, Composition)- İçerme (Aggregations) Bağıntı İlişkisi

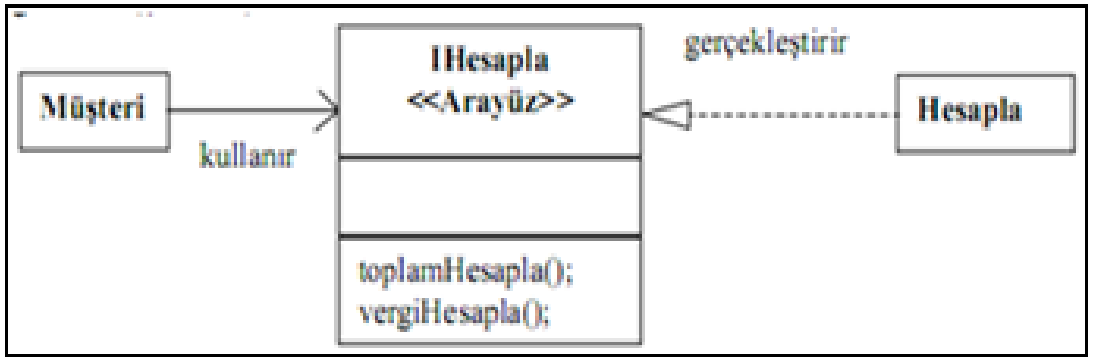
- Birden fazla parçadan oluşan sınıflar arasındaki ilişkiye "Aggregation" denir.
- Aggregation ilişkisini 'bütün parça' yukarıda olacak şekilde ve 'bütün parça'nın ucuna içi boş elmas yerleştirecek şekilde gösterilir.
- İçi boş elmas ile gösterilen ilişkilerde herbir parça ayrı bir sınıftır ve tek başlarına anlam ifade ederler.



Gerçekleştirim (Realization) İlişkisi

- Gerçekleştirim ilişkisi en çok kullanıcı arayüzlerinin (user interface) modellenmesinde kullanılır.
- Arayüz yanlızca method adlarını ve bunların parametrelerini içermektedir.
- Program yazarken, yalnızca arayüzlerin kullanılması ve arayüzü gerçekleştiren sınıfın diğer sınıflardan ayrı tutulması, yazılımın geliştirilmesi ve bakımında önemli kolaylık sağlar.
- Gerçekleştirim ilişkisi kesikli bir çizginin ucuna yerleştirilen içi boş bir üçgen ile gösterilir.

Gerçekleştirim (Realization) İlişkisi - Örnek



- Hesapla adı verilen ve diğer sınıfların kullanımına açılmış arayüz sınıfı tanımlanmıştır. Bu sınıfta yer alan metotların gerçekleştirimi ise Hesapla sınıfı tarafından üstlenilmiş durumdadır.

Üst Düzey Tasarım

Üst Düzey Tasarım

- Amaç: Sistemi oluşturacak temel varlıkları ve ilişkilerini tanımlamak
  - Sistemin sınıf yapısı
    - Uygulama alanına özgü sınıflar (“domain classes”)
    - Sınıflar arası ilişkiler
  - Davranış modeli
    - Sınıfların işlevsel gereksinimleri karşılamak için birbirleri ile nasıl işbirliği yapacakları
  - Literatürde “OO Analysis” olarak da geçiyor

Üst Düzey Tasarım Adımları

1. Sınıfları belirle
   - Amaç: “Use case” tanımlarından analiz sınıflarını (“domain classes”) ve sorumluluklarını belirlemek
     - Odak noktası sınıfları belirlemektir (detaylı tanımların sonra yapılması önerilir)
2. Sınıf yapısını belirle
   - Amaç: Belirlenen sınıfların “use case”lerde tanımlanan iş adımlarını gerçekleştirmek için nasıl etkileşimde bulunacağını saptamak
     - “Use case”ler gözden geçirilerek belirlenen sınıfların arasındaki ilişkiler tanımlanır
3. Davranışı modelle
   - Amaç: “Use case”lerin içerdiği işlevselliğin, belirlenen sınıfların davranışları aracılığıyla nasıl gerçekleştirileceğini tanımlamak
     - Nesnelerin belirlenen işi yerine getirmek için nasıl mesajlaşacakları belirlenir
     - Nesneler arası kontrol akışı dinamik olarak gösterilir

Adım-1. Sınıfları Belirle (1)

- Use case modelinde yer alan her use case tanımı gözden geçirilir
  - Analiz sınıfı ("domain class") özelliklerini taşıyan varlıklar belirlenir
  - Belirlenen sınıflara sorumluluk atanır
    - Use case tanımlarındaki işler esas alınır
  - Önceki use case'lerde belirlenmiş sınıfları kullanmaya çalışırız
    - Analiz sınıfları büyük olasılıkla birçok use case'de karşımıza çıkacaktır

Adım-1. Sınıfları Belirle (2)

- Use case tanımlarının üzerinden geçerken:
  - Kullanılan “isimler” aday sınıf olarak ele alınır
  - Aday sınıflar belirli kurallar ve gereksinimler doğrultusunda elenir
  - Elemeler sonucunda oluşan “isim” kümesi sistemin sınıfları olarak ele alınır
  - Elemelerden geçemeyen “isim”ler belirlenen sınıfların özellikleri olabilirler
  - İşi tanımlayan “fiil”ler sınıfların sorumluluklarını veya sınıflar arası ilişkileri tanımlar

Adım-1. Sınıfları Belirle (3)

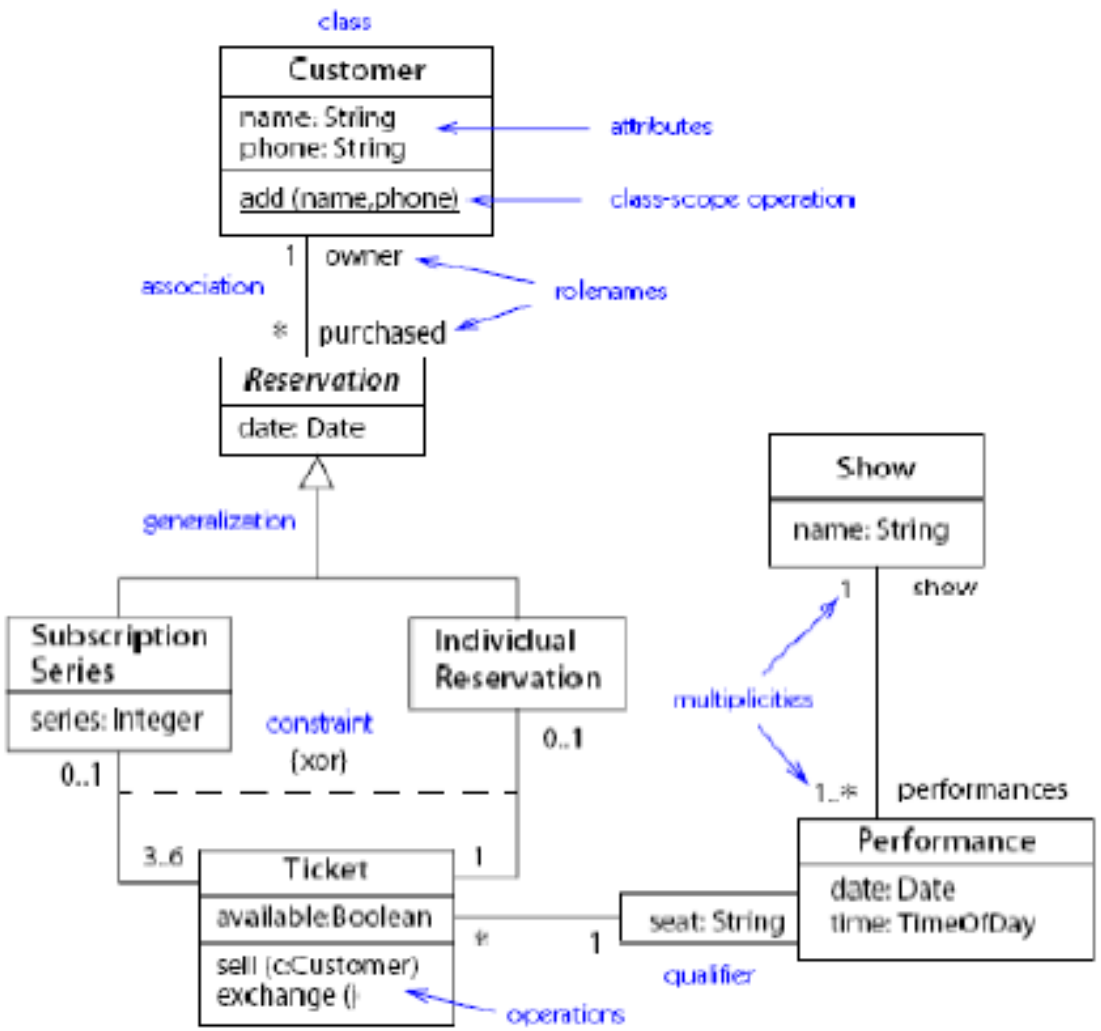
- Belirlenen her sınıf için, sınıfın özellikleri tanımlanır:
  - Özellikler “use case” tanımlarında doğrudan yer alabilir
  - Özellikler sınıfın sorumluluklarından çıkarılabilir
  - Bu aşamada özellikler ile ilgili detaylı bir tanımlamaya gerek yoktur
    - Veri yapılarına girilmeden sadece isimlendirilir

- Belirlenen her sınıf için sınıfın davranışı, sorumluluklar esas alınarak operasyon halinde tanımlanır
  - “Use case”de tanımlanan işlevsellik, sınıflar arasında paylaştırılır
  - Sorumluluk, sınıfın farklı “use case”lerde aldığı rollerin birleşimidir
    - Her rol bir operasyon olabilir
  - Bu aşamada “class methods”, “signatures” detaylarına girilmez

Adım-2. Sınıf Yapısını Belirle

- Belirlenen ilişkiler davranış modellemesine esas oluşturur
  - “Fiil”ler ilişkilerin göstergesidir
- Sınıflar arası ilişkiler kavramsal olarak tanımlanır
  - İlişkinin türlerine ve detay özelliklerine girilmez
- Sınıflar arası ilişkiler en az sayıda tutulmaya çalışılır
- Tasarım süreci devam ettikçe basit ilişkiler özel formlara dönüşebilir
  - “association”, “aggregation”, “composition”, “generalization”, vb.

Class Diagram-Örnek



Adım-3. Davranışı Modelle

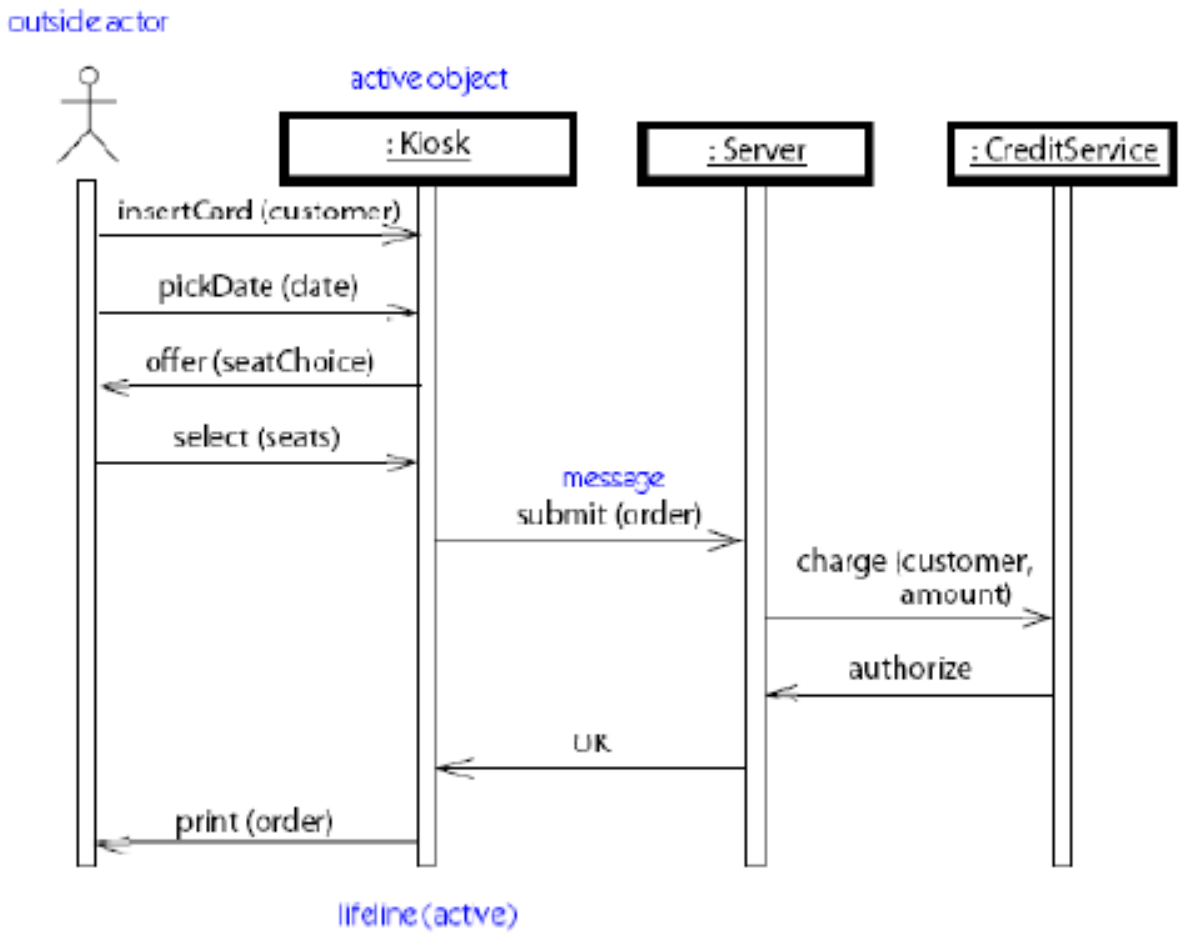
- Davranışları iki şekilde tanımlanabilir:
  - Ardıl-işlem (“sequence”) diyagramları ile
    - Nesneler arası iletişimin zamana göre sıralaması
  - İşbirliği (“collaboration”) diyagramları ile
    - Nesnelerin etkileşimde aldığı roller ve ilişkiler
    - Nesneler arası iletişimin etkileşime göre sıralaması

- Önerilen yöntem:
  - Bir “use case” veya senaryo seçilir
  - Hangi nesnelerin rol alacağı belirlenir
  - Bir nesneden diğerine gönderilecek mesajlar zamana veya etkileşime göre sıralanarak gösterilir

Ardıl-İşlem (“Sequence”) Diyagramı

- Nesneler arasındaki ilişkileri zaman sırasına göre düzenlenmiş olarak göstermeye yarar
- Bir etkileşimde yer alan nesneleri ve çağrılan mesajların sırasını gösterir
- İşbirliği diyagramı ile benzer bilgileri farklı biçimde gösterir
- İşbirliği diyagramında bulunan nesne ilişkileri ardıl-işlem diyagramında açıkça görülmez

“Sequence Diagram”: Örnek

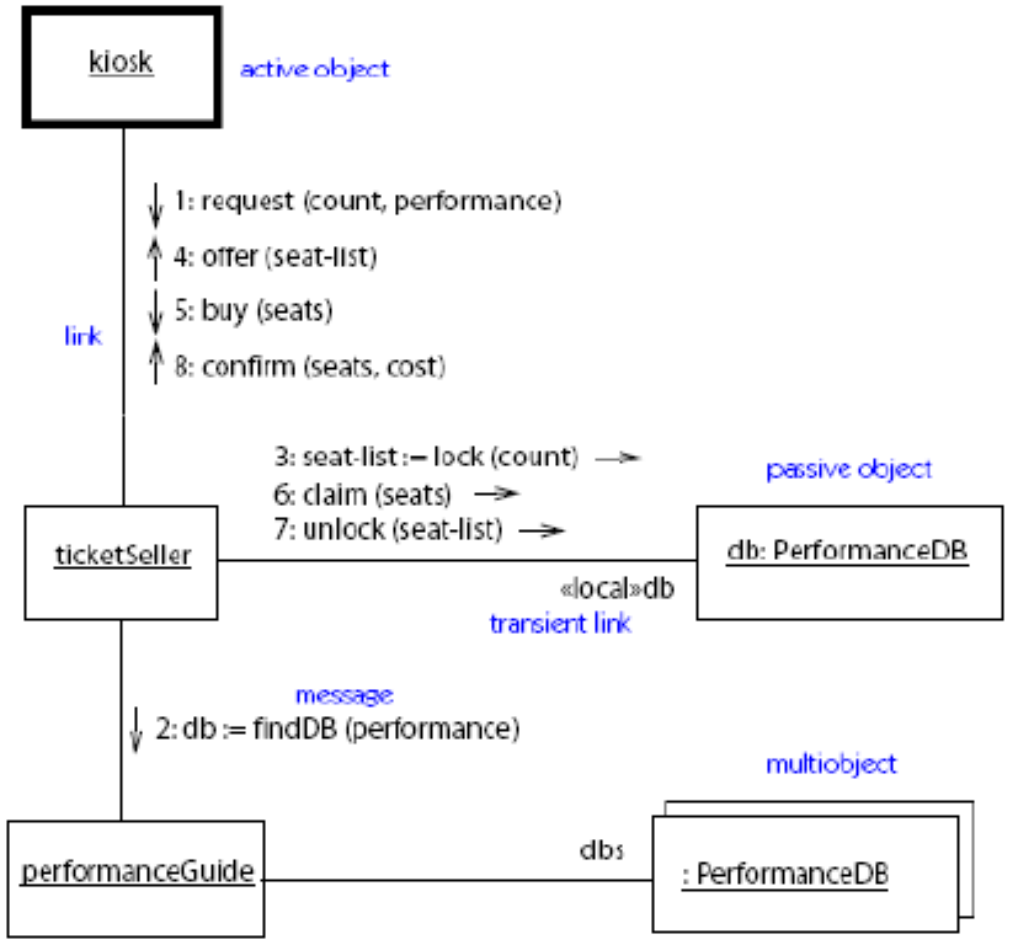


İşbirliği (“Collaboration”) Diyagramı

- İşbirliği ve ardıl-işlem diyagramları nesneleri ve aralarındaki mesajlaşmayı dinamik olarak modellemeye yarar

- İşbirliği diyagramı, ardıl-işlem diyagramından farklı olarak, sınıflar arasındaki ilişkileri doğrulamada yardımcı olur

“Collaboration Diagram”: Örnek



Detaylı Tasarım

Detaylı Tasarım

- Amaç: Tasarım modellerini, gerçekleştirmeye olanak sağlayacak şekilde detaylandırmak
  - Odak gerçekleştirme detaylarına kayar
  - Üst düzeyde oluşturulan UML diyagramları somut kararlar alındıkça zenginleştirilir
  - Sistem mimarisini ve detay davranışı modellemek için farklı UML diyagramları oluşturulur

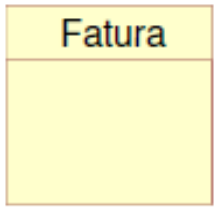
Ele Alınacak Hususlar

- Gereksinimlerde tanımlanan işi gerçekleştirmek için gereken diğer sınıflar
- Gerçekleştirmede kullanılacak algoritmalar ve veri yapıları
- Kullanılacak tasarım örüntüleri
- Kullanılacak hazır parçalar (“component”)
- Veri saklama yöntemleri (ilişkisel veri tabanı, vb.)
- Uygulamanın üzerinde çalışacağı platform
  - Ortam dağıtık olacak mı ?
  - Uygulama nasıl parçalanacak ?
- Performans kısıtları
- ...

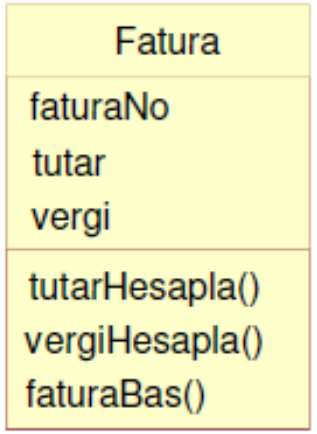
Detaylı Tasarım Adımları

1. Sınıf yapısını detaylandır
   - Amaç: Sınıf diyagramlarını, gerçekleştirmeye yönelik zenginleştirmek
     - Sınıf tanımlarını ve ilişkileri detaylandırmak ve/veya kısıtları belirlemek
       - Tasarım sınıflarını ve aralarındaki ilişkileri belirlemek
       - Tasarım sınıflarının diğer sınıflarla ilişkilerini belirlemek
2. Davranış modelini detaylandır
   - Amaç: Sınıfların ve sistemin davranışını daha detaylı olarak modellemek
     - Nesneler arası işbirliği kapsamında
       - Ardıl-işlem ("sequence") diyagramı; İşbirliği ("collaboration") diyagramı
     - Bir nesne kapsamında
       - Durum ("statechart") diyagramı; Etkinlik ("activity") diyagramı
3. Mimariyi modelle
   - Amaç: Yazılım sistemini parçaları ("components") cinsinden tanımlamak
     - Diyagramları ve model varlıklarını, paket ("package") yapısı altında gruplandırmak

(Adım-1. Sınıf Yapısını Detaylandır)
Tasarım Boyunca Sınıf Tanımının Gelişimi: Örnek



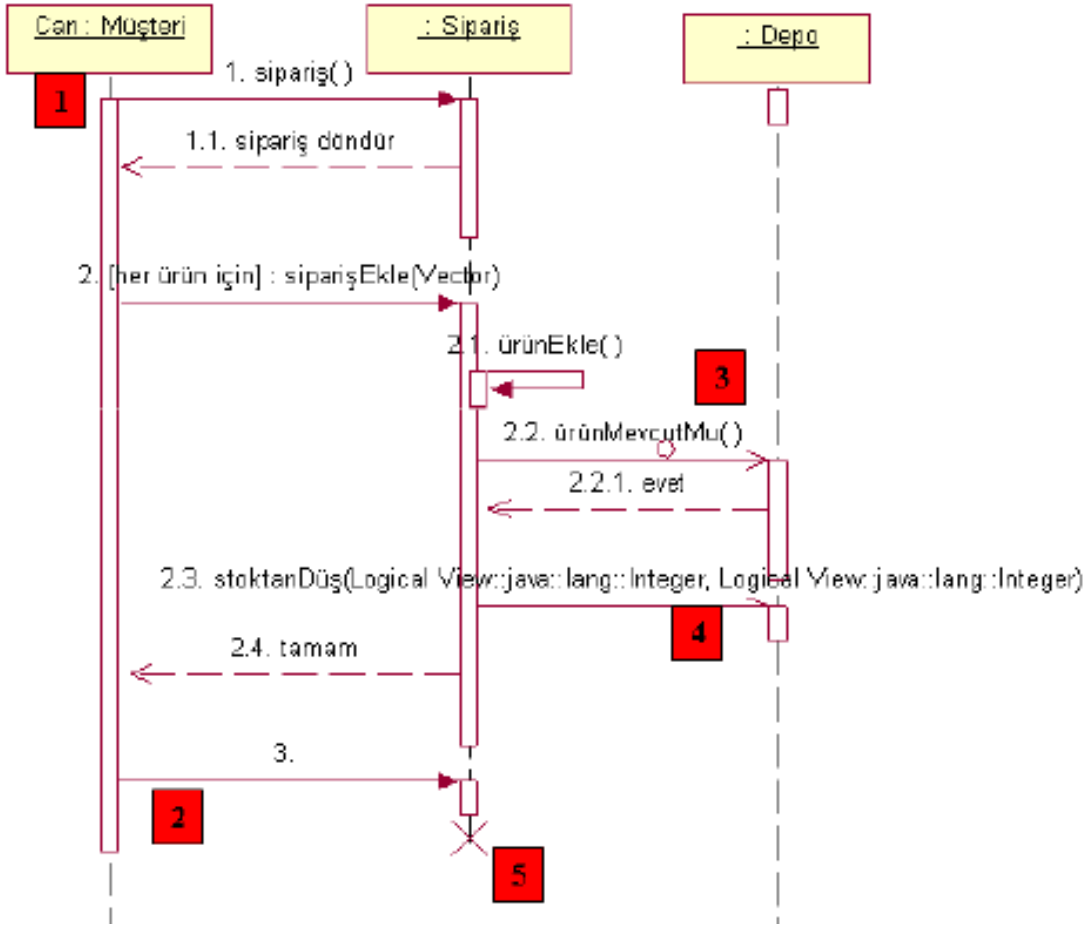
Üst Düzey Tasarım
(ön aşamalar)



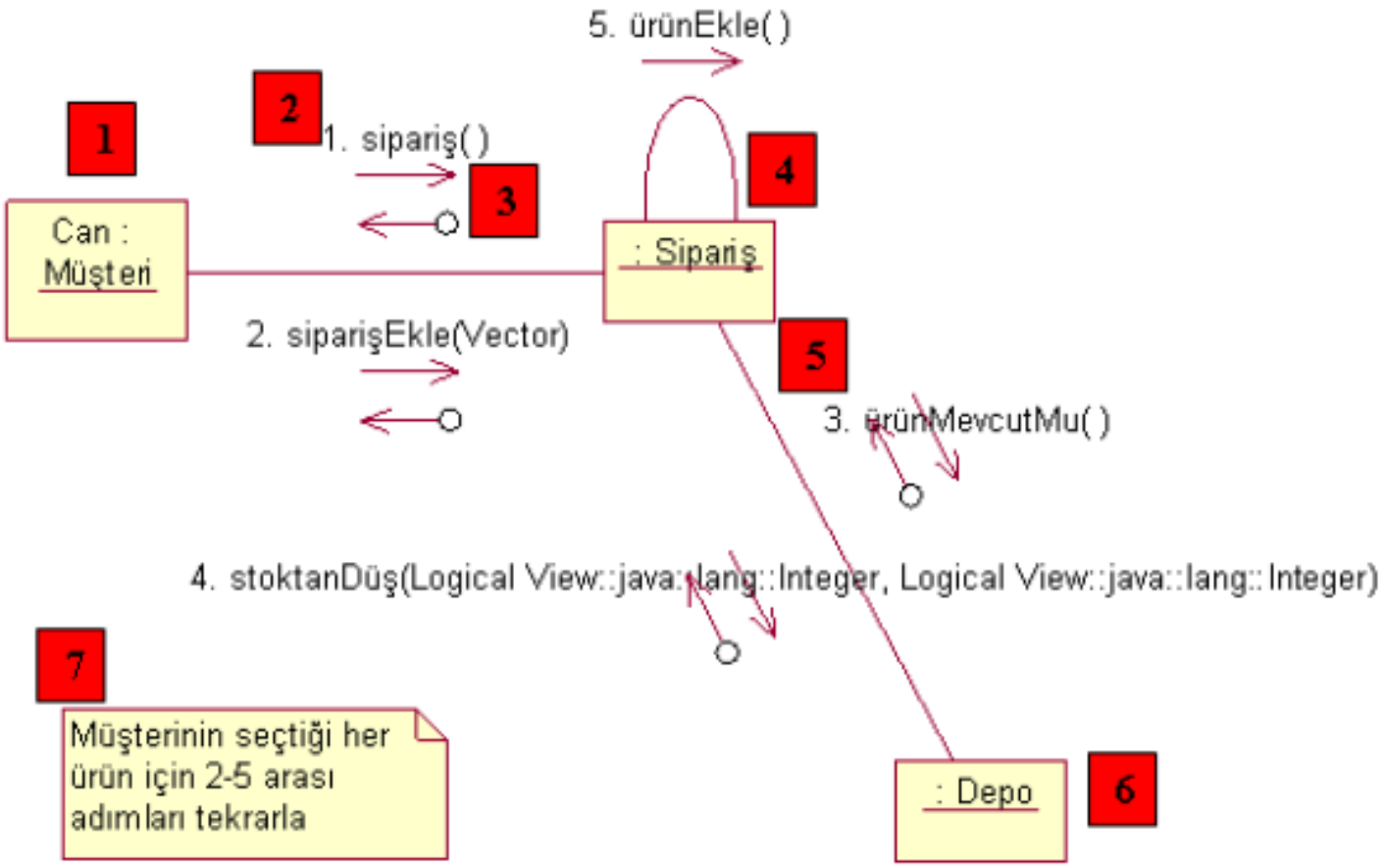
Üst Düzey Tasarım
(ileri aşamalar)

Detay Tasarım

(Adım-2. Davranış modelini detaylandır) Ardıl-İşlem Diyagramı: Örnek



(Adım-2. Davranış modelini detaylandır) İşbirligi Diyagramı: Örnek

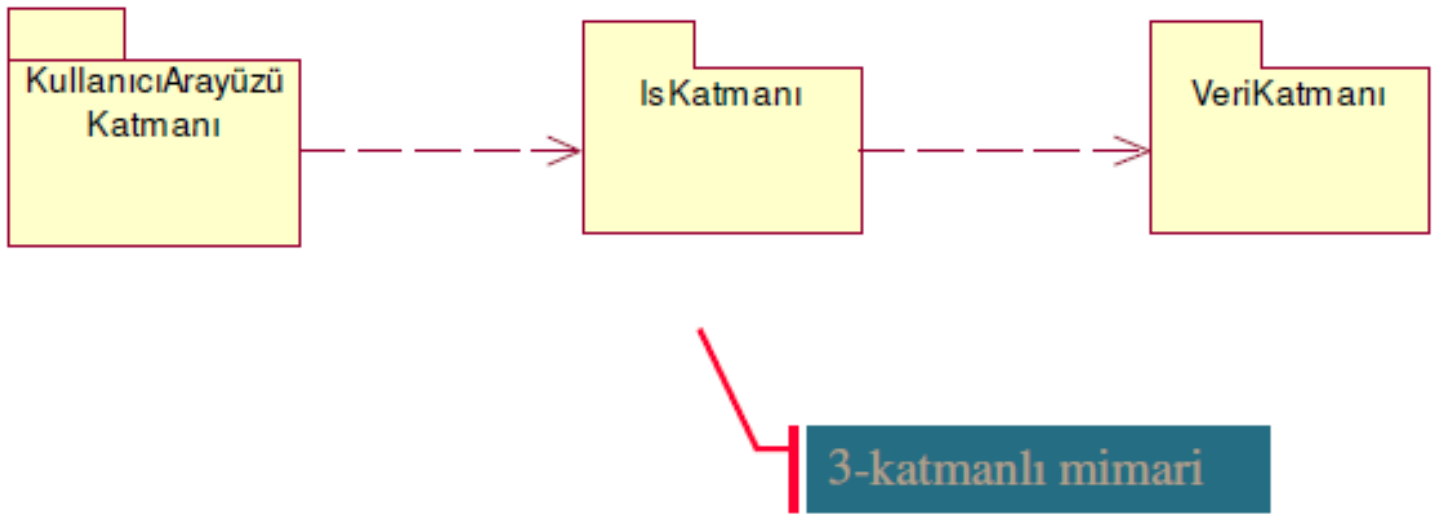


(Adım-2. Davranış modelini detaylandır) Durum (“Statechart”) Diyagramı

- Nesnelerin dinamik davranışlarını tanımlar
- Davranışı karmaşık olan nesneleri modellemek için kullanılır
- Bir nesnenin farklı durumları (“state”) arasındaki geçişleri modeller
- Özellikle reaktif nesnelerin davranışlarının modellenmesinde etkilidir
  - Asenkron olaylara göre davranış gösteren nesneler
- Sistemin genel davranışının modellemesi için de kullanılabilir

(Adım-3. Mimariyi Modelle)
Kavramsal Mimari

- Uygulamanın fiziksel platformdan bağımsız, farklı katmanlar halinde tasarlanması
  - Tek bir platformda çalışacak bile olsa farklı katmanlardan oluşmasında fayda var



(Adım-3. Mimariyi Modelle)
Fiziksel Mimari

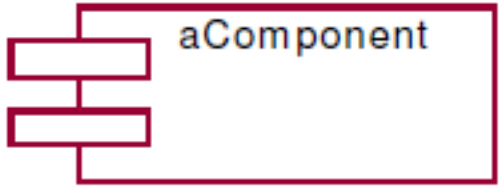
- Uygulamanın fiziksel olarak nasıl parçalanacağını tanımlar
  - Kavramsal mimari ile aynı olabilir
  - Kavramsal mimariden farklı olabilir

- Örnekler:
  - 3-katmanlı mimari
    - Kullanıcı arayüzü: Client
    - İş katmanı: Uygulama sunucusu
    - Veri katmanı: Veri sunucusu
  - 2-katmanlı mimari
    - Kullanıcı arayüzü: Client
    - İş ve Veri katmanları: Uygulama sunucusu

Bileşen (“Component”) Diyagramı

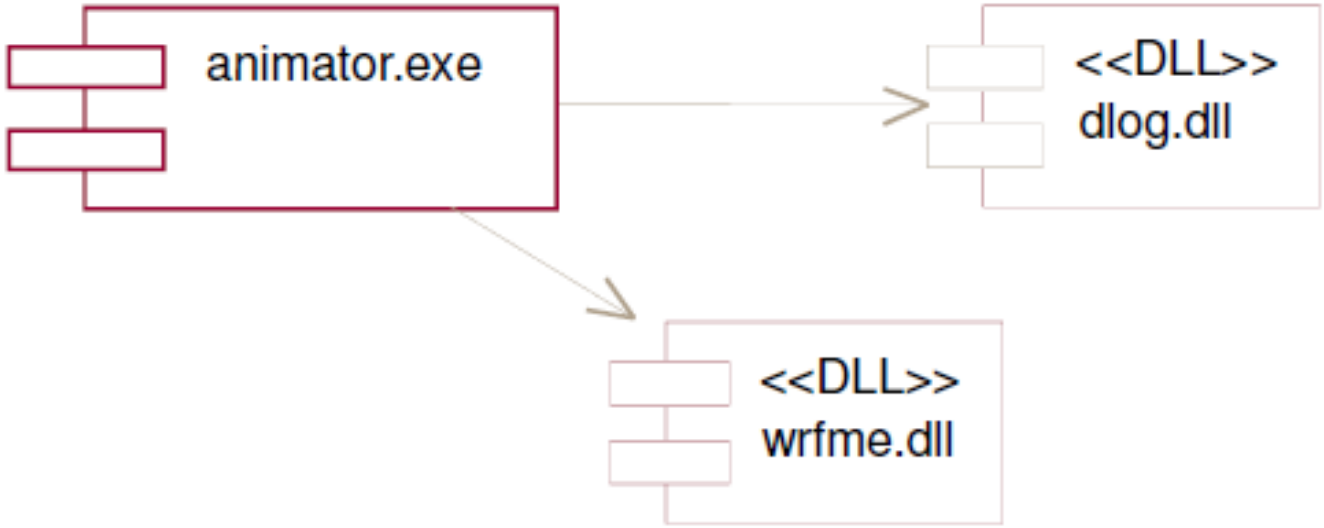
- Temel kullanımı:
  - Fiziksel yazılım bileşenlerini ve aralarındaki ilişkileri modellemek
  - Kaynak kodlar ve dosyalar arasındaki ilişkileri modellemek
  - Yazılım sürümlerinin yapısını modellemek
  - Çalıştırılabilir kod oluşturan kaynak dosyalar arasındaki ilişkileri modellemek

- UML’de bileşen (“component”):
  - Bileşen, işleyen sistemi oluşturan parçalardan biridir
    - İşleyen kod parçası, kütüphane, tablo, dosya, vb.
  - Sistemin fiziksel ve değiştirilebilen bir parçasıdır
    - Aynı arayüze sahip farklı parçalar ile değiştirilebilir
  - Bir veya birden fazla sınıf veya paket içerebilir



Bileşen Diyagramı: Örnek

- Gösterim:
  - Bileşenler ve aralarındaki ilişkileri gösteren oklar
  - Okun yönü bağımlı bileşenden, bağımlı olduğu bileşene doğru
  - İlişki okları “stereotype” kullanarak isimlendirilebilir
    - Örnek: “include”, “derive”, “friend”, “import”, vb.



Yayılma (“Deployment”) Diyagramı

- Bir yazılım sisteminde yer alan donanım öğelerini ve aralarındaki ilişkileri modeller

- Yayılma diyagramında iletişim ilişkisi ile birleştirilmiş “node”lar yer alır
  - “Node”lar sistemdeki işlemci kaynakları temsil eder ve küp şeklinde gösterilir
    - Bilgisayarlar, alıcılar, çevresel veya gömülü sistemler, vb.
  - İletişim ilişkileri “stereotype” kullanılarak iletişimin tipini gösterebilir

Yayılma Diyagramı: Örnek