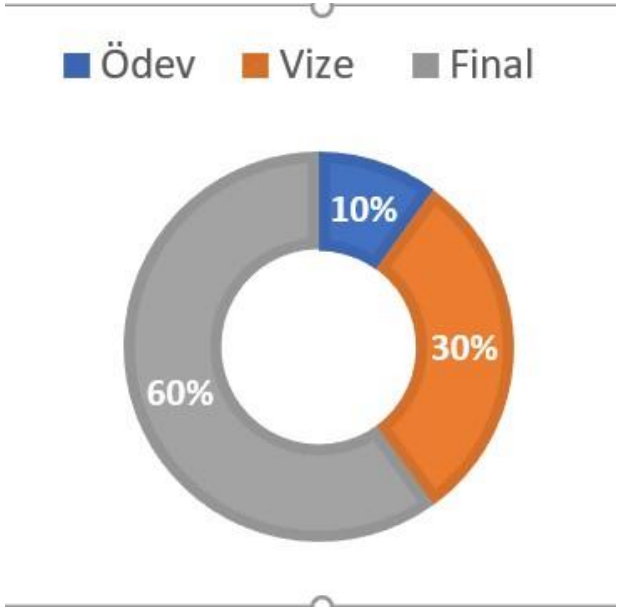
Yazılım Mühendisliği
1906003082015

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

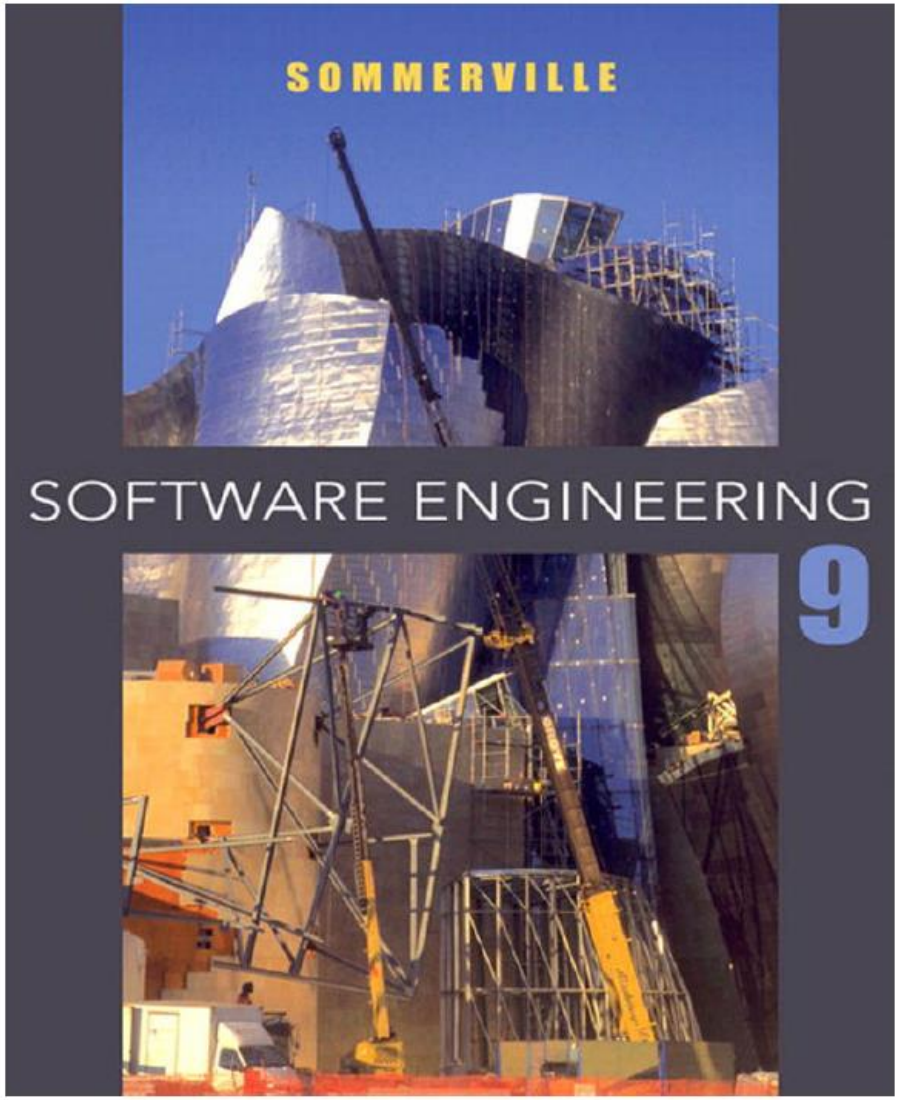
Giriş

Ders Günü ve Saati:
Salı: 09:15-13:00
Devam zorunluluğu %70



HAFTA	KONULAR
Hafta 1	Yazılım Mühendisliğine Giriş
Hafta 2	Yazılım Geliştirme Süreç Modelleri
Hafta 3	Yazılım Gereksinim Mühendisliği
Hafta 4	Yazılım Mimarisi
Hafta 5	Nesneye Yönelik Analiz ve Tasarım
Hafta 6	Laboratuar Çalışması: UML Modelleme Araçları
Hafta 7	Yazılım Test Teknikleri
Hafta 8	Ara Sınav
Hafta 9	Yazılım Kalite Yönetimi
Hafta 10	Yazılım Bakımı - Yeniden Kullanımı ve Konfigürasyon Yönetimi
Hafta 11	Yazılım Proje Yönetimi (Yazılım Ölçümü ve Yazılım Proje Maliyet Tahmin Yöntemleri)
Hafta 12	Yazılım Proje Yönetimi (Yazılım Risk Yönetimi)
Hafta 13	Çevik Yazılım Geliştirme Süreç Modelleri
Hafta 14	Yazılım Süreci İyileştirme, Yeterlilik Modeli (CMM)

Kaynaklar



İÇERİK

- Tekil Modelleme Dili (**UNIFIED MODELING LANGUAGE-UML**)
- Modelleme Gereksinimi
  - UML Tarihçesi
  - UML Diyagramları
    - Sınıf Diyagramları
    - Erişim
    - Sınıflar Arası İlişki
    - Kalıtım (Inheritence)
    - İçerme (Aggregations)

- Modelleme Gereksinimi
  - UML Diyagramları
    - Arayüzler
  - Sınıfların Sahip Olduğu Roller

UML

UML, Yazılım Mühendisliği'nde nesne tabanlı sistemleri modellemede kullanılan açık standart olmuş bir görsel programlama dilidir. Analizden tasarım aşamasına dek oldukça etkili bir notasyon sağlar.

- Yazılım ve donanımların bir arada düşünülmesi gereken,
- Zor ve karmaşık programların,
- Özellikle birden fazla yazılımcı tarafından kodlanacağı durumlarda,
- Standart sağlamak amacıyla endüstriyel olarak geliştirilmiş grafiksel bir dildir.
- ~~Programlama dili~~-Diyagram çizme ve ilişkisel modelleme dili

UML

- UML yazılım sisteminin önemli bileşenlerini tanımlamayı, tasarlamayı ve dokümantasyonunu sağlar
- Yazılım geliştirme sürecindeki tüm katılımcıların (kullanıcı, iş çözümleyici, sistem çözümleyici, tasarımcı, programcı,...) gözüyle modellenmesine olanak sağlar,
- UML
- Yazılımın geniş bir analizi ve tasarımı yapılmış olacağından kodlama işlemi daha kolay ve doğru olur
- Hataların en aza inmesine yardımcı olur
- Geliştirme ekibi arasındaki iletişimi kolaylaştırır
- Tekrar kullanılabilir kod sayısını artırır
- Tüm tasarım kararları kod yazmadan verilir
- Yazılım geliştirme sürecinin tamamını kapsar
- “resmin tamamını” görmeyi sağlar
- gösterimi nesneye dayalı yazılım mühendisliğine dayanır.

2. MODELLEME GEREKSİNİMİ

Karmaşık sistemleri daha iyi anlayabilmek için modeller yaparız. Oluşturulan model sayesinde karmaşık bir gerçeği daha basit bir dille ifade etme şansımız olur. Modeller, karmaşık sistem ya da yapıların gereksiz ayrıntılarını filtreleyerek, problemlerin anlaşılabilirliğini arttırırlar, buna kısaca soyutlama (abstraction) denmektedir.

- program geliştirici ya da tasarımcı, bir model oluşturarak proje elemanlarının detaylarda boğulmadan, birbirleriyle nasıl etkileştiği konusunda bir büyük resim sağlar.
- Modellemenin gerekliliğini.

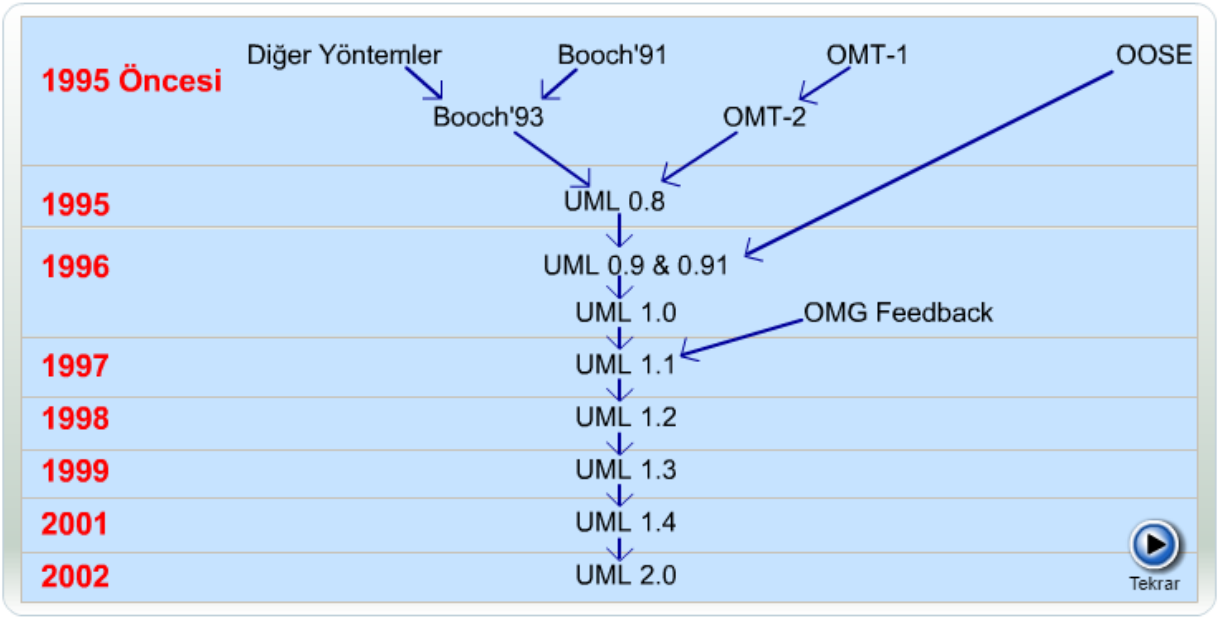
1. Problem çözümü için bir yapı sağlamak
2. Birçok farklı çözüm yolu araştırmak için denemeler yapmak
3. Kompleksliği yönetmek için soyut bir fikir sağlamak
4. Problemlerin çözümü için zamanı azaltmak
5. Gerçekleştirim maliyetini düşürmek
6. Hata yapma riskini düşürmek

UML TARİHÇE

90'lı yıllarda farklı notasyonlara sahip metodolojiler aynı şeyleri farklı yöntemlerle ifade ediyorlardı. Her biri bazı sistemler için mükemmelken, bazıları için işe yaramaz durumdaydı ve hemen hemen hepsi Yazılım Yaşam Döngüsünün (Software Life Cycle) bazı adımlarını tanımlamakta yetersiz kalıyordu.

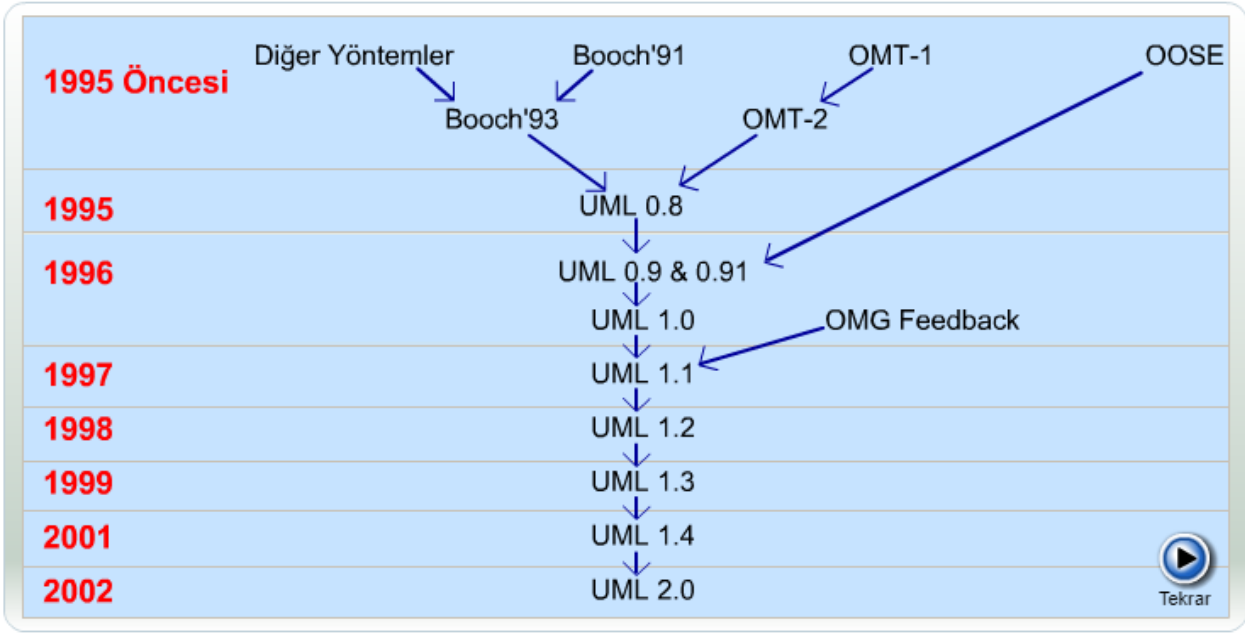
Bunlardan en yaygın olan üç tanesi: OMT (Object Modelling Technology), Booch ve OOSE (Object Oriented Software Engineering) metotlarıydı.

Metot savaşlarının sonu, notasyon açısından UML'nin ortaya konmasıyla geldi. Booch, OMT ve OOSE metotlarının yaratıcıları bir grup oluşturarak kendi yöntemlerinin olumlu taraflarını birleştirerek, komple bir yazılım projesinde sistemi modellemede kullanılabilecek eksiksiz bir modelleme dili geliştirmeye başladılar



UML TARİHÇE

- Microsoft, Oracle, HP gibi büyük firmaların da katıldığı bir UML konsorsiyumu kuruldu ve bu şirketler UML'yi modellemelerinde kullanmaya başladılar. İlk resmi sürüm (ver 0.8), Ekim 1995'te tanıtıldı.
- Kullanıcıdan ve OOSE'nin yaratacısı Ivor Jacobson'dan gelen geri beslemeler ile Temmuz 1996'da sürüm 0.9 v Ekim 1996'da sürüm 0.91 çıkarıldı.
- Kar gütmeyen, bilgisayar endüstrisi standartlarını oluşturan bir organizasyon olan OMG (Object Management Group) tarafından açık standart olarak geliştirilmeye başlandı ve Temmuz 1997'de Sürüm 1.0 çıkarıldı.
- Herhangi bir şirkete veya kişiye ait bir modelleme dili olmayan UML, şu an Rational, MS Visio, Together Soft, vb. birçok modelleme aracı tarafından kullanılmaktadır.



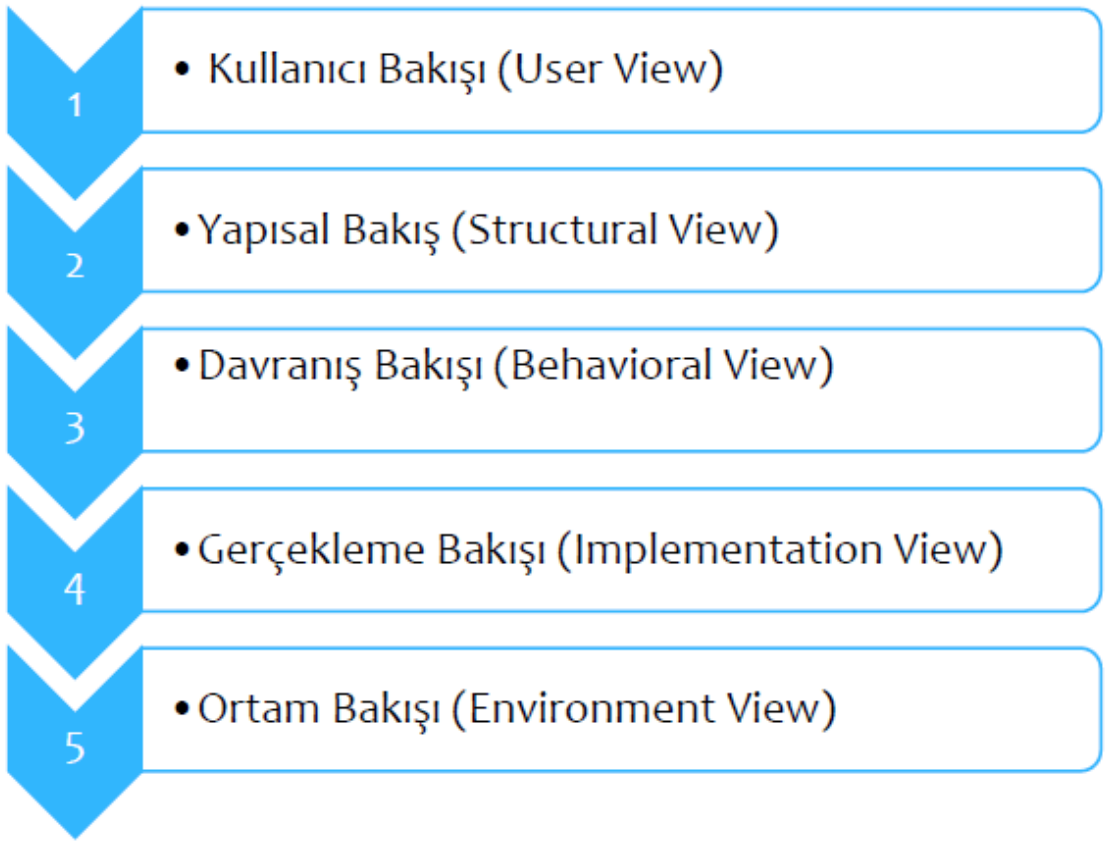
UML ile Yazılım Modellenmesi

- Yazılımın yaşam döngüsü içinde farklı görev gruplarının projeye ve sisteme farklı bakışları vardır. Müşteriyi hangi işin hangi sırayla yapılacağı, sisteme neler verip sistemden neler alacağı veya işler arası ilişkileri ilgilendirirken bir fonksiyonun detayları ilgilendirmemektedir.
- Çözümleyici açısından bir nesnenin özellikleri, fonksiyonları ve alacağı parametreler yeterli iken, tasarımcı açısından parametrelerin veri tipleri veya fonksiyonun ne kadar bir sürede cevap üretmesi gerektiği, bir nesnenin ne zaman etkin olacağı, yaşam süresi gibi bilgiler de önemli olmaktadır. Teknik yazar ise sistemin nasıl davranacağı ve ürünün işleyişi ile ilgilenmektedir. Bu sebeplerle UML çeşitli bakış açılarını ifade eden diyagramlar içermektedir.

1. Kullanıcı Bakışı (User View)
2. Yapısal Bakış (Structural View)
3. Davranış Bakışı (Behavioral View)
4. Gerçekleme Bakışı (Implementation View)
5. Ortam Bakışı (Environment View)

UML ile Yazılım Modellenmesi

- Çözümleyiciler, tasarımcılar, programcılar, testçiler, kalite sorumluları, müşteriler/kullanıcılar, teknik yazarlar yazılım geliştirme işinde rol alan şahıslardır. Bunlardan herbiri sistemin değişik yönleriyle farklı bakış açılarıyla ve farklı detayda ilgilenirken farklı UML diyagramlarından faydalanırlar. Özetleyecek olursak Projeye dahil olan herkesin faydalanacağı bir veya daha fazla diyagram vardır



UML Diyagramları

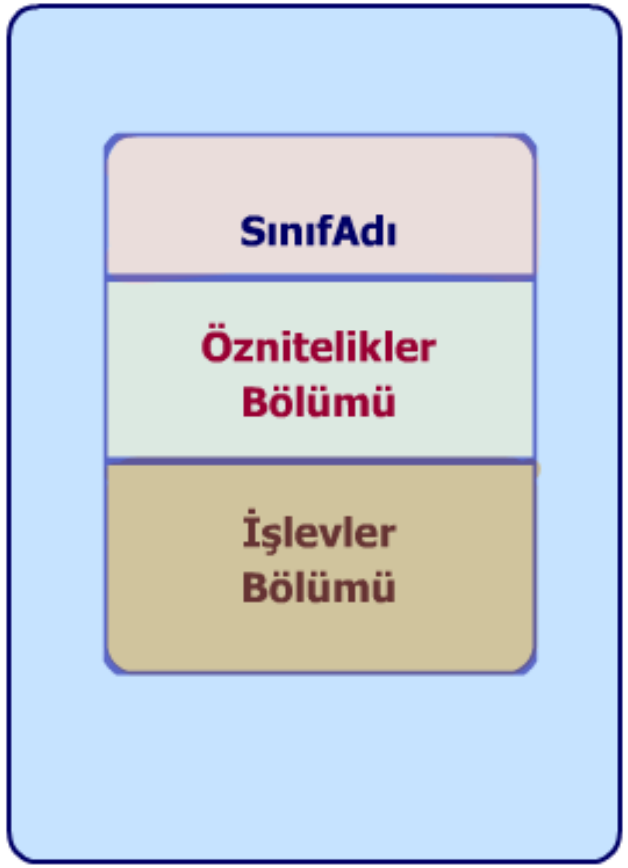
- Nesneler arasında ilişki kurmak için UML bir takım grafiksel elemanlara sahiptir. Bu elemanlar kullanılarak diyagramlar oluşturulur.

Sınıf Diyagramları

- Kavramsal (conceptual) bir model olan Sınıf Diyagramları, hem gereksinimlerin müşteriye özetlenmesinde hem de tasarımda kullanılmaktadır. Sınıf Diyagramları, bazı detayları saklayarak müşteri-çözümleyici iletişimini desteklerken, detaylı haliyle de tasarımcıya ve programcıya yardımcı olmaktadır. Sınıf Diyagramlarının tasarımı ilgilendiren kısmı oldukça kapsamlıdır.
- Nesne tabanlı sistemlerin doğru tasarlanması tamamen tasarımcının tecrübe ve bilgisi ile ilgilidir, bu sebeple tasarlanacak Sınıf Diyagramlarının, **türetme** (**inheritance**), **soyut sınıflar**, **sınıf hiyerarşileri**, **tasarım örnekleri** (**design pattern**) gibi detaylarının düşünülmesi gerekmektedir. Bu noktada UML'nin "doğru tasarım nasıl yapılır?" sorusuna cevap vermediğini sadece bir modelleme dili olduğunu belirtmek gerekir. Sistemin performansı veya gelecekte yüzleşeceğimiz güncelleme ve bakım sorunları, yazılan kodun başka projelerde veya ek modüllerde yeniden kullanılabilir olması (code reuse) gibi detaylar tasarımcı açısından çok önemli olmalıdır.

Sınıf Diyagramları

- Sınıf Diyagramları UML 'in en sık kullanılan diyagram türüdür.
- Sınıflar nesne tabanlı programlama mantığından yola çıkarak tasarlanmıştır.
- Sınıf diyagramları bir sistem içerisindeki nesne tiplerini ve birbirleri ile olan ilişkileri tanımlamak için kullanılırlar.
- UML'de sistem yapısını anlatmak için Sınıf Diyagramlarını kullanırız. Sınıf Diyagramları, aralarında bizim belirlediğimiz ilişkileri içeren sınıflar topluluğudur.
- UML' de bir sınıf yanda görüldüğü gibi 3 bölmeden oluşan dikdörtgen ile ifade edilir. Bunlara ek olarak notlar (notes) ve Kısıtlar (Constraints) tanımlanabilir.
- Dikdörtgenin en üstünde sınıfın adı vardır.
- Sınıfın sahip olduğu **Öznitelikler** SınıfAdı'nın hemen altına ikinci bölmeye yazılırken, son bölmeye de sınıfın sahip olduğu **İşlevler** yazılır. Genel bir kullanım şekli olarak sınıflara isim verilirken her kelimenin ilk harfi büyük yazılır.



Sınıf Diyagramları

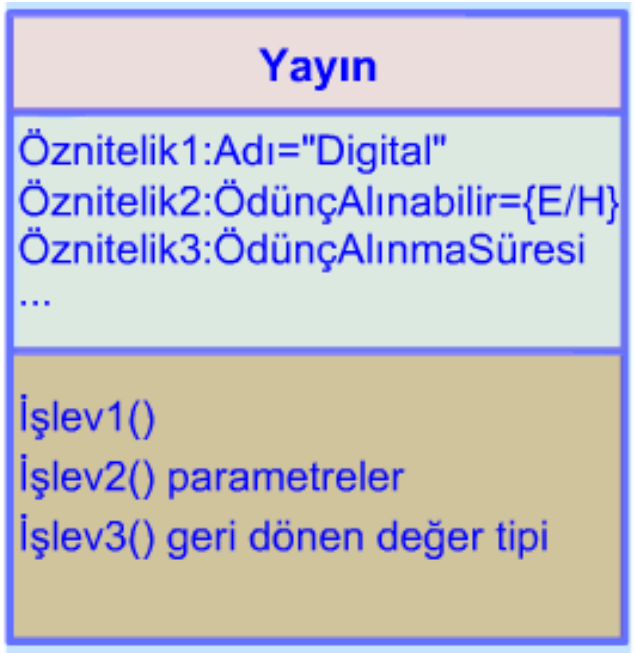
Öznitelik

- Sınıfların kendilerini karakterize eden özellikleri vardır. Mesela yayın nesnesinin Ad, ÖdünçAlınabilir ve ÖdünçAlınmaSüresi bilgileri **öznitellik (attribute)** bildirir.
- Bir özniteliğin değerini sonradan değiştirebileceğimiz gibi, yandaki örnekte *Adı*özniteliğinde olduğu gibi varsayılan bir değer de atanabilir. Atama işlemi öznitelik adından sonra "=" işareti yazılıp daha sonra atanacak değer sayısal bir değer ise doğrudan, sayısal bir değer değil ise tırnak içinde yazılarak ilk değer ataması yapılır.
- Atanan bu değerler number (sayı), string (katar), boolean (lojik 0-1), float (ondalık sayı), double (gerçel sayı) veya kullanıcı tanımlı bir veri tipi olabilir.
- Yayın ve Kitap sınıflarının özniteliklerinin UML notasyonunda nasıl ifade edildiği görülmektedir.

Sınıf Diyagramları

İşlev

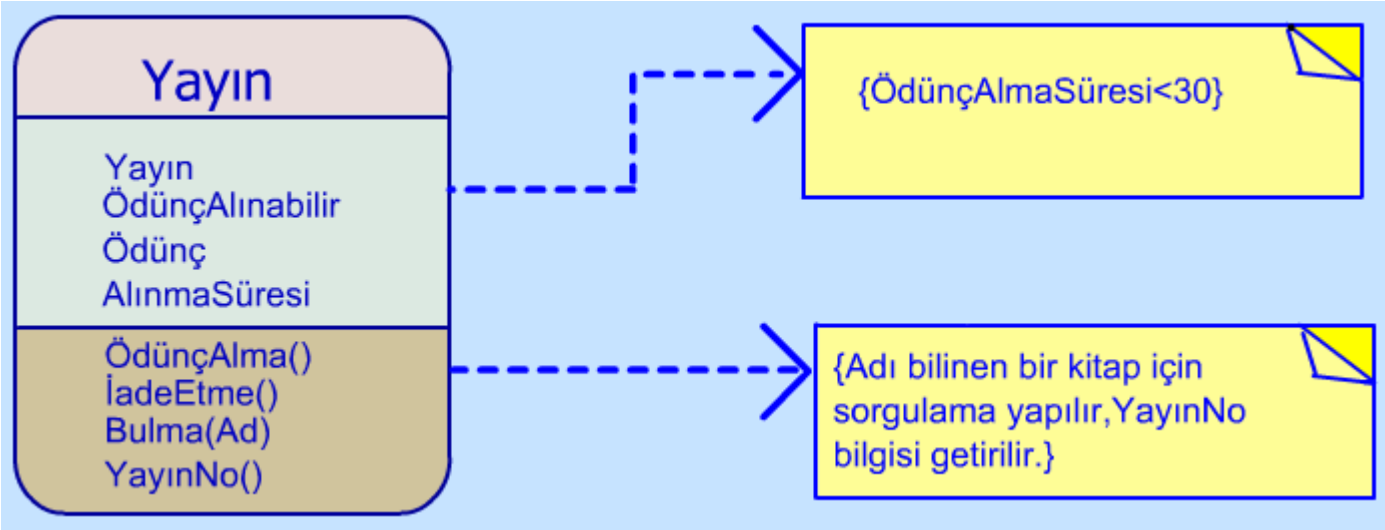
- Sınıfların bir diğer önemli elemanı da İşlevler (Operations)'dir. İşlevler bir sınıfta iş yapabilen elemanlardır. Bu iş başka bir sınıfa yönelik olabileceği gibi, kendi içindeki bir iş de olabilir. Sınıf Diyagramları'nda işlevler şekildeki gibi, özelliklerin hemen altında gösterilirler.
- İşlevler öznitelliklerden farklı olarak, doğru olarak çalışabilmeleri için dışarıdan birtakım bilgilere ihtiyaç duyabilir, veya birtakım bilgileri dışarıya verebilir, ya da bunların hiçbirini yapmazlar.
- Yandaki Sınıf Diyagramı'nda İşlev1'in çalışması için dışarıdan bir bilgiye ihtiyaç yoktur; ve bu işlev dışarıya da herhangi bir bilgi vermez. İşlev2, işini yapabilmek için birtakım bilgilere ihtiyaç duyar. Örneğin, bir toplama işlemi yapıyorsa toplanacak elemanları ister. İşlev3 ise çalıştıktan sonra işinin sonucunu dış dünyaya verir.



Sınıf Diyagramları

Kısıtlar ve Notlar

- Bir Sınıf Diyagramında kullanılabilecek temel yapılar Öznitelik ve İşlev olmasına rağmen, kısıtlar (constraints) ve notlar (notes) dediğimiz elemanları da bunlara ekleyebiliriz. **Notlar** genellikle işlevler ve özellikler hakkında bilgi veren opsiyonel kutucuklardır. **Kısıtlar** ise sınıfa ilişkin birtakım koşulların belirtildiği ve parentez içinde yazılan bilgilerdir.

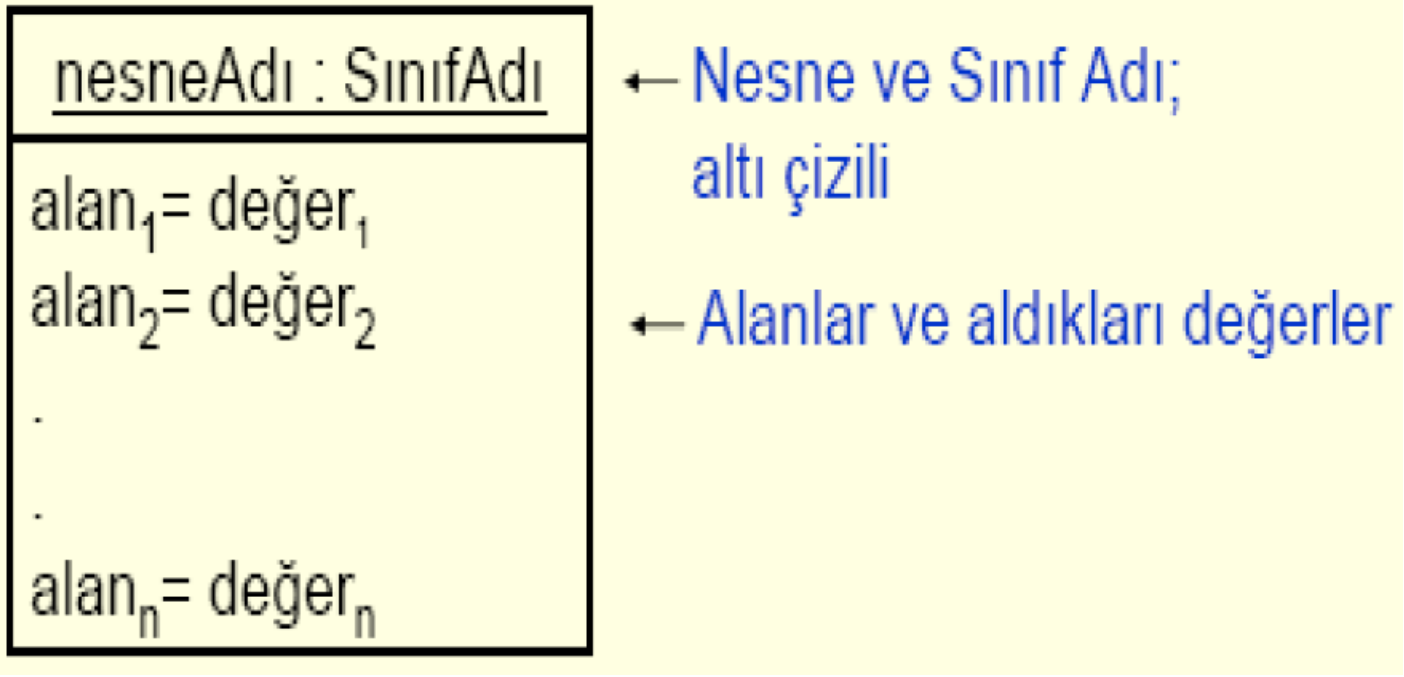


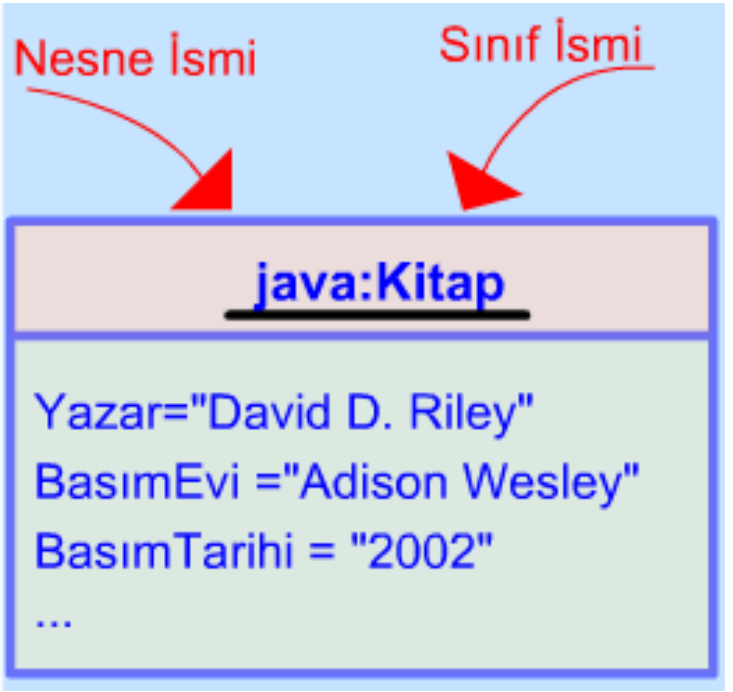
Sınıf Diyagramları

ERİŞİM

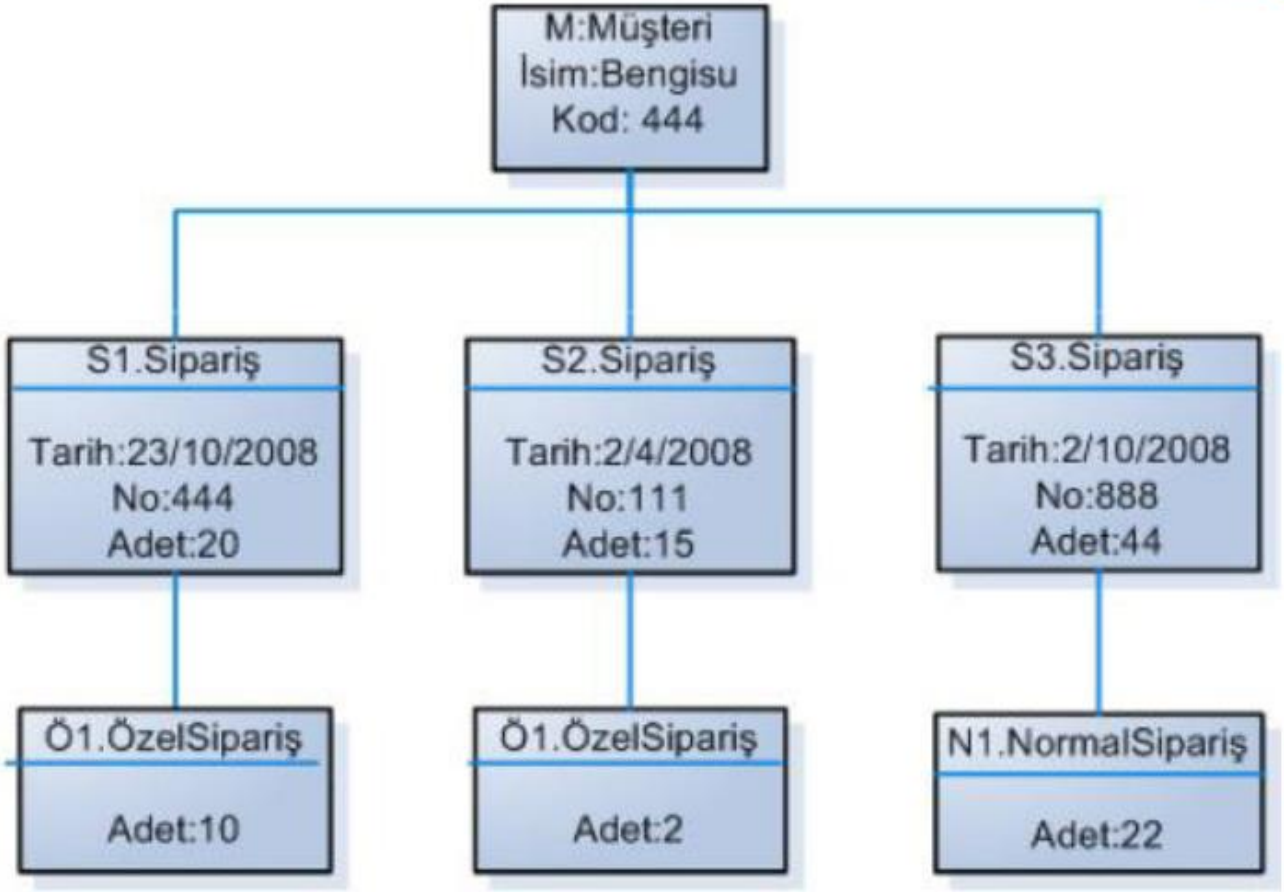
- **Public:**diğer sınıflar erişebilir. UML'de **+** sembolü ile gösterilir.
- **Protected:**aynı paketteki (*package*) diğer sınıflar ve bütün alt sınıflar (*subclasses*) tarafında erişilebilir. UML'de**#**sembolü ile gösterilir.
- **Package:**aynı paketteki (*package*) diğer sınıflar tarafında erişilebilir. UML'de **~**sembolü ile gösterilir.
- **Private:**yalnızca içinde bulunduğu sınıf tarafından erişilebilir (diğer sınıflar erişemezler). UML'de -sembolü ile gösterilir.

UML'de Nesne Gösterimi





UML'de Nesne Gösterimi



Sınıf Diyagramları

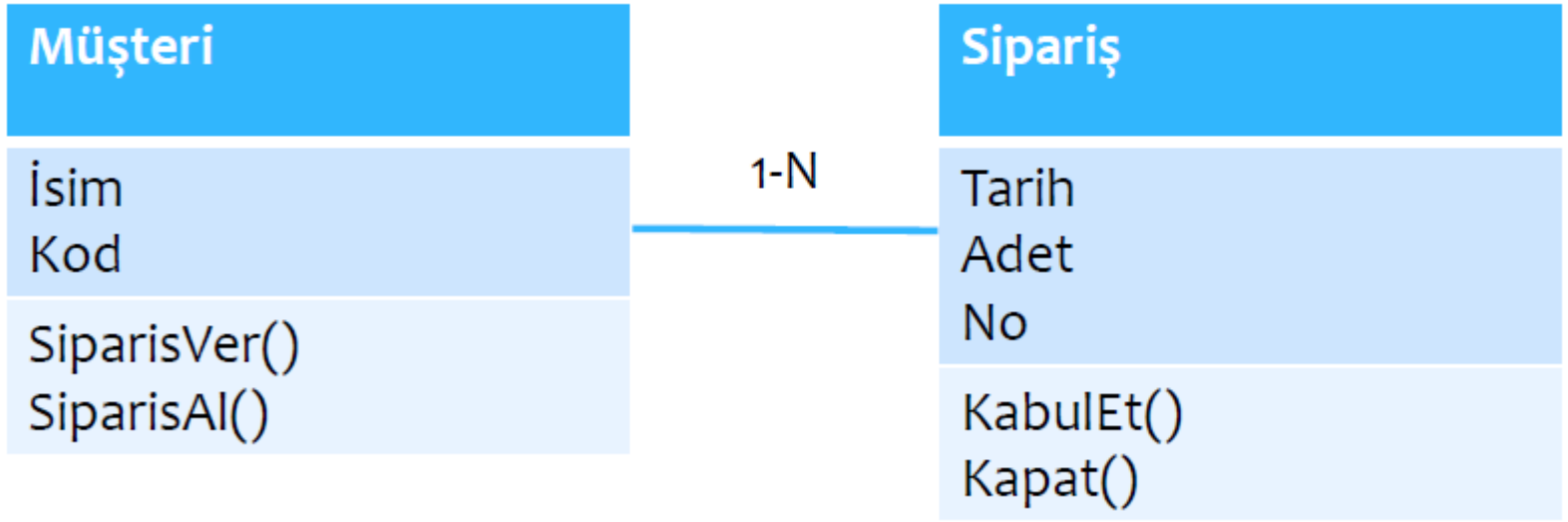
Sınıflar Arası İlişki (Assocation)

- UML'de sınıflar arasındaki ilişki, iki sınıf arasına düz bir çizgi çekilerek ve bu çizginin üzerine ilişkinin türü yazılarak gösterilir. Örneğin, Üye ve Yayın sınıfları olsun. Üye ile Kitap sınıfı arasında "Ödünç alma" ilişkisi vardır. Bu, Sınıf Diyagramı'nda aşağıdaki gibi gösterilir, ve sağ tarafında yazdığı şekilde ifade edilir.

Sınıf Diyagramları

Sınıflar Arası İlişki (Assocation)

- UML'de sınıflar arasındaki ilişki, iki sınıf arasına düz bir çizgi çekilerek ve bu çizginin üzerine ilişkinin türü yazılarak gösterilir.

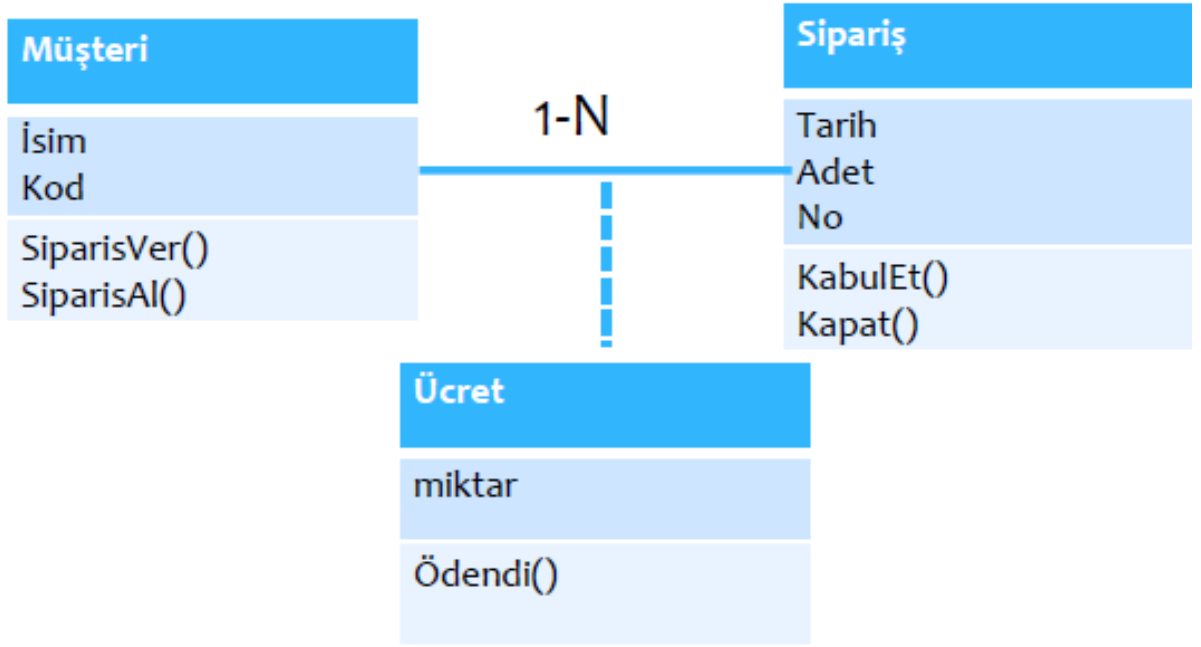


Sınıf Diyagramları

İlişki Sınıfları

- Bazı durumlarda sınıflar arasındaki ilişki, bir çizgiyle belirtebilecekşekilde basit olmayabilir.
- Bu durumda ilişki sınıfları kullanılır.
- İlişki sınıfları bildigimiz sınıflarla aynıdır.
- Sınıflar arasındaki ilişki eğer bir sınıf türüyle belirleniyorsa UML ile gösterilmesi gerekir.

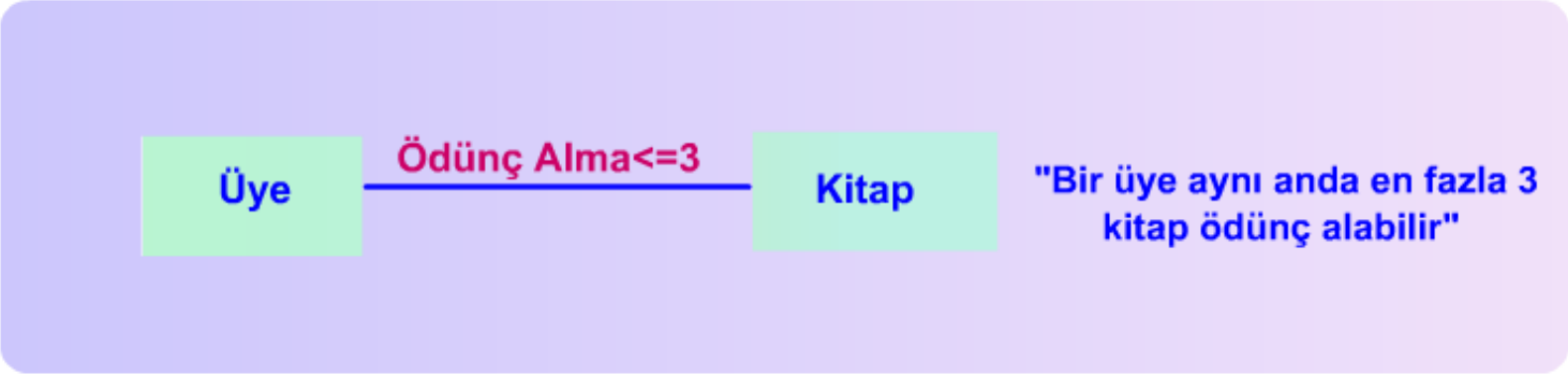
- Müşteri ile Sipariş sınıfı arasında ilişki vardır. Fakat müşteri satın alırken Ücret ödemek zorundadır
- Bu ilişkiyi göstermek için Ücret sınıfı ilişki ile kesikli çizgi ile birleştirilir.



Sınıf Diyagramları

Kısıtlar

Bazı durumlarda belirtilen ilişkinin bir kurala uyması gerekebilir. Bu durumda ilişki çizgisinin yanına kısıtlar (constraints) yazılır. Örneğimizde bir üye aynı anda en fazla 3 kitap ödünç alabilir olsun.



Sınıf Diyagramları

İlişki Tipleri

İlişkiler her zaman bire-bir olmak zorunda değildirler. Eğer bir sınıf, n tane başka bir sınıf ile ilişkiliyse biz buna bire-çok ilişiki deriz. Örneğin bir dersi 30 öğrenci alıyorsa Öğretmen ile Öğrenci sınıfları arasında 1-30 bir ilişki vardır. Çizelgede bunu gösterirken Öğretmen sınıfına 1, Öğrenci sınıfına ise 30 yazarız. Gösterimi aşağıdaki gibidir:

Sınıf Diyagramları

İlişki Tipleri

Temel ilişki tipleri aşağıdaki gibi listelenebilir:

1	Bire-bir
2	Bire-çok
3	Bire-bir veya daha fazla
4	Bire-sıfır veya bir
5	Bire-sınırlı aralık (Örneğin: bire-[0,20] aralığı)
6	Bire-n (UML'de birden çok ifadesini kullanmak için * simgesi kullanılır)
7	Bire-beş ya da Bire-sekiz

KALITIM (INHERITANCE)

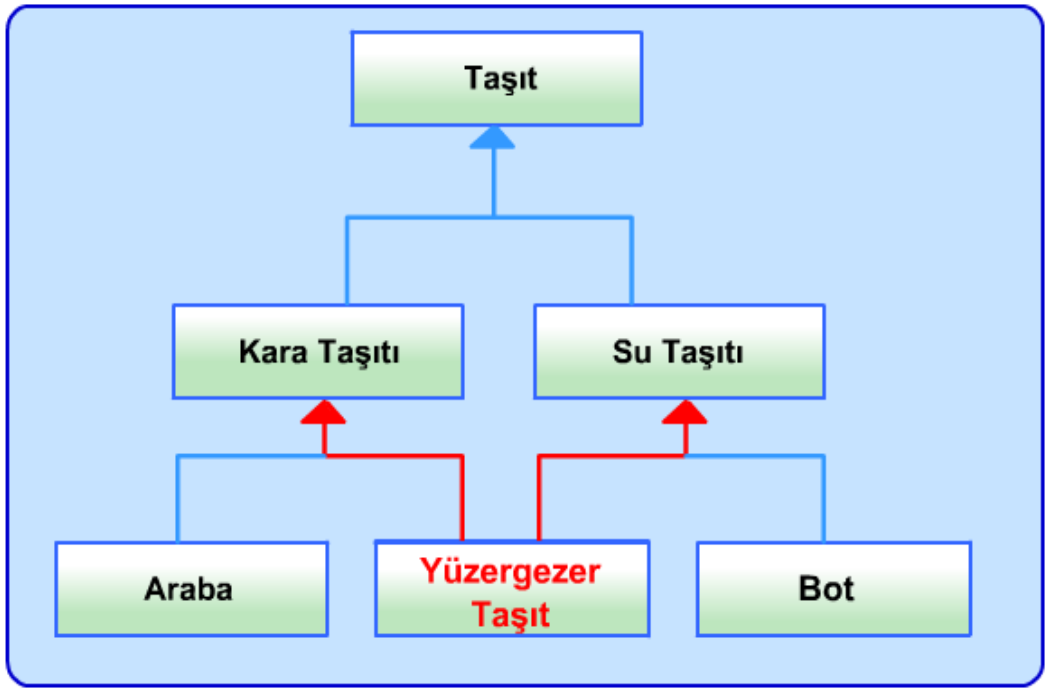
Nesneler arasında ortak özellikler varsa, bunu her sınıfta belirtmek yerine ortak özellikleri bir sınıfta toplayıp, diğer sınıfları da ortak sınıftan türeterek ve yeni özellikler ekleyerek organizasyonu daha etkin hale getirmeye, nesne yönelimli programlamada ***kalıtım (inheritance)*** denir.

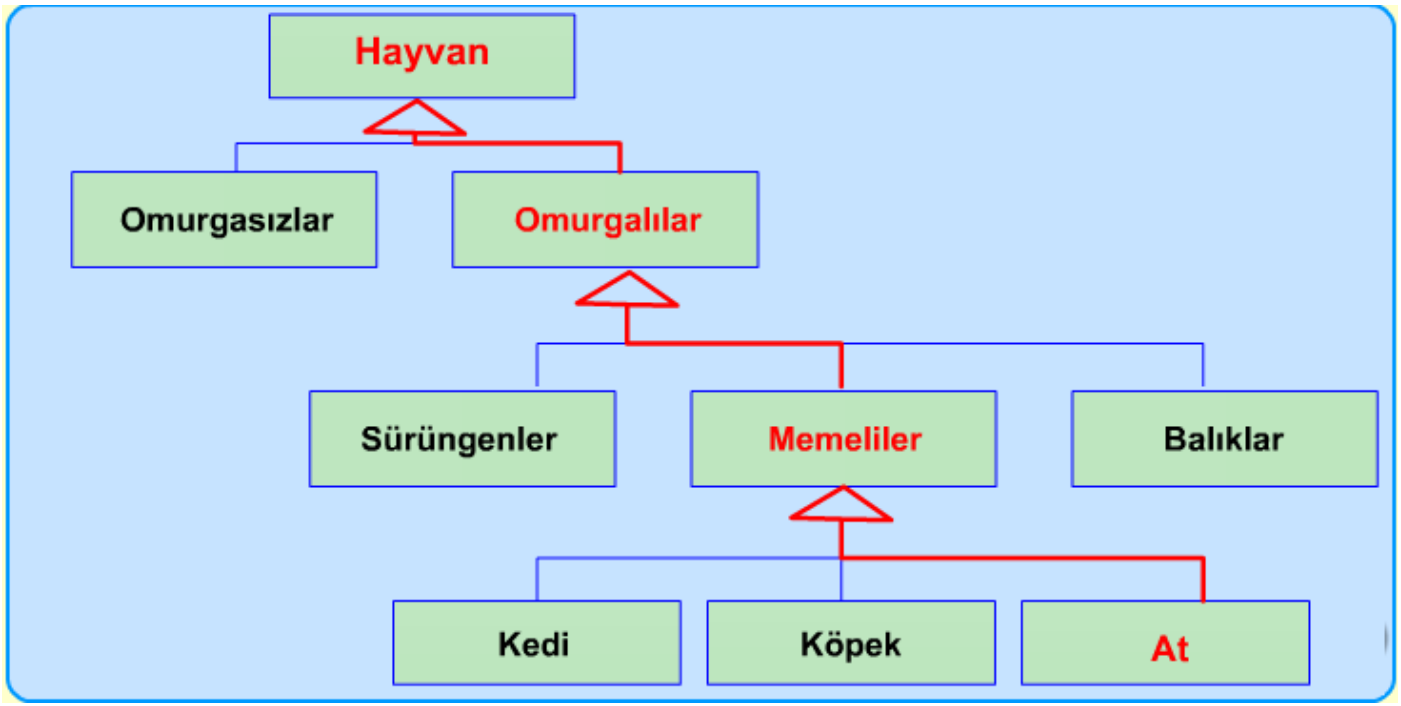
Kalıtıma;

- Nesnenin Görevini arttırarak, yeni üye fonksiyonları ilave etmek
- Bir sınıfın üye fonksiyonlarından birinin görevlerinin farklı biçimde yerine getirecek şekilde değiştirmek

Türetme yapabilmek için öncelikle, tanımlanmış en az bir sınıfın mevcut olması gerekir. Türetme işleminde kullanılan bu mevcut sınıfa taban sınıf (base class), türeme sonucunda ortaya çıkacak yeni sınıfa ise türemiş sınıf (derivated class) denir.

KALITIM (INHERITANCE)

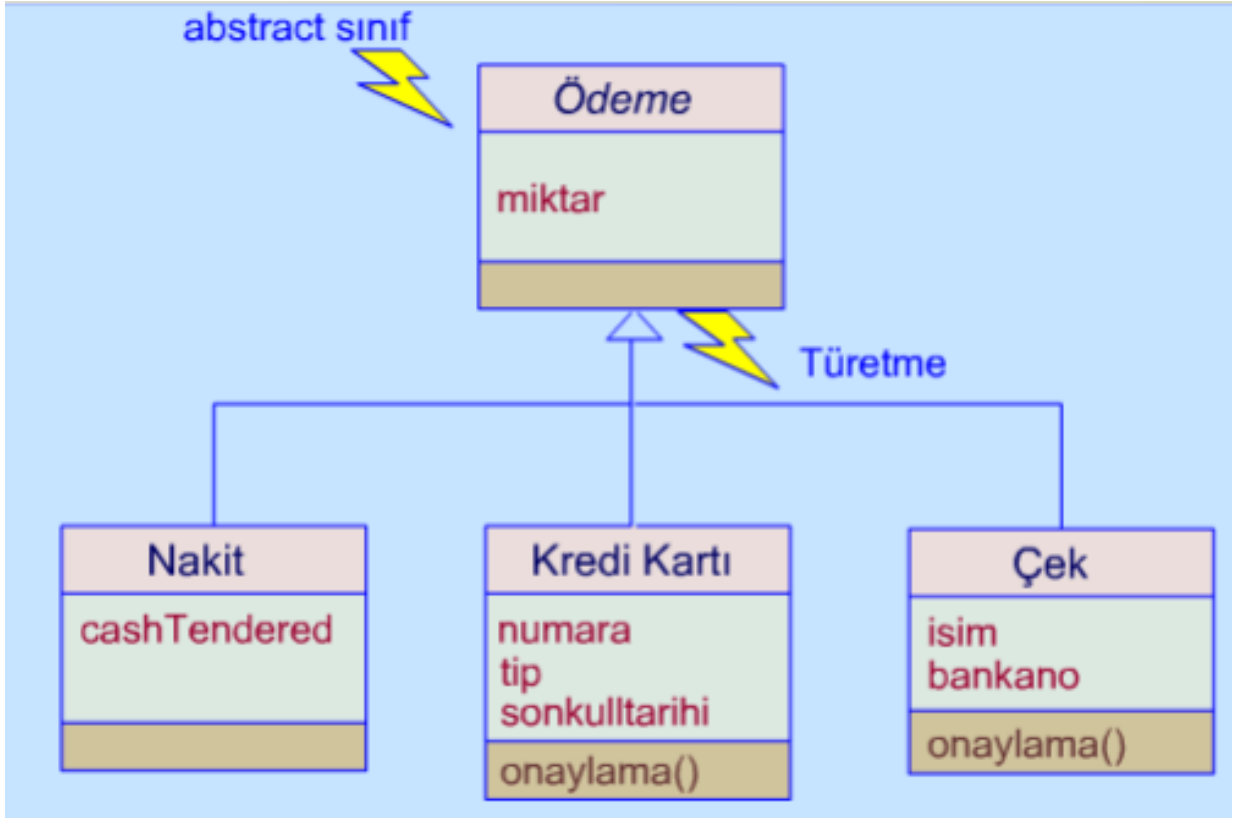




KALITIM (INHERITANCE)

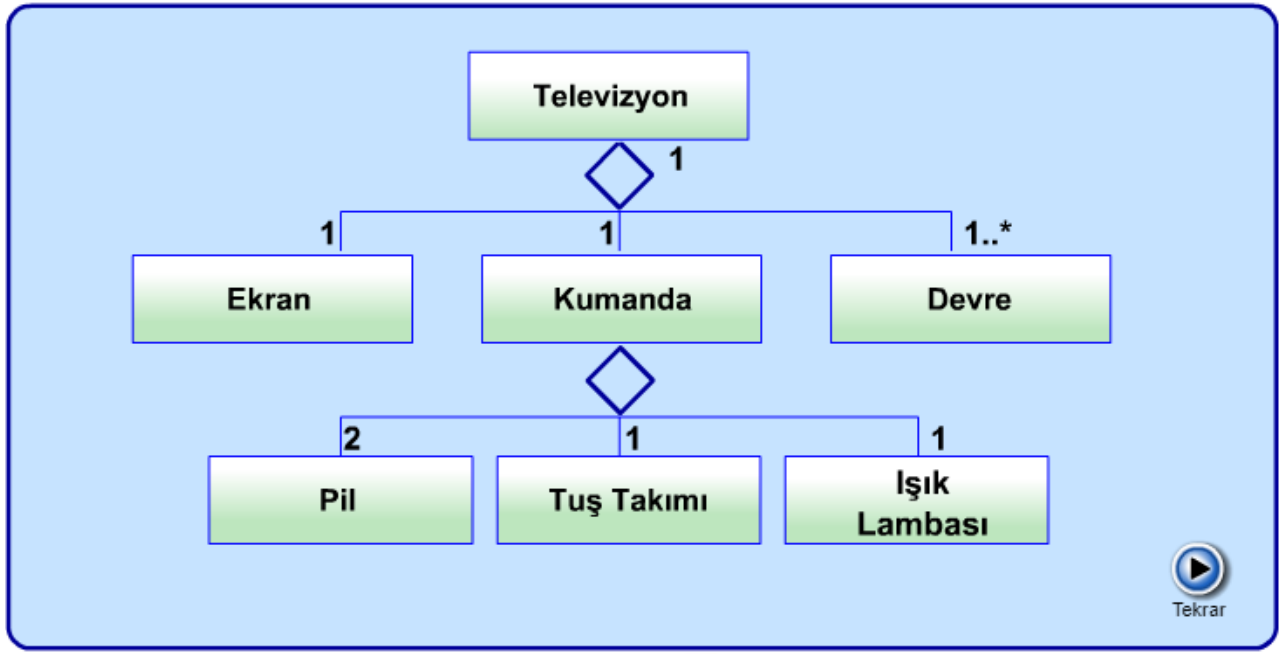
Soyut (Abstrct) Sınıflar

- Eğer ortak özelliklerin toplandığı sınıftan gerçek nesneler türetilmesini engellemek istiyorsak **Soyut Sınıf (Abstract Class)** oluştururuz. UML'de bir sınıfın Soyut (Abstract) olması için sınıf ismini italik yazarız.
- Aşağıdaki örnekte Ödeme sınıfını Soyut (Abstract) Sınıfa örnek verebiliriz. Ödeme nakit, çekle ya da kredi kartıyla yapılabilir. Üçü için de birer sınıf yaratılır, özellik ve işlevleri anlatılır. Ama üçünün de ortak özelliği olan ne kadar ödeme yapılacağı bilgisi için Ödeme soyut sınıfı yaratılır. Diğer üç sınıf bu sınıftan türetilir.



İÇERME (AGGREGATIONS)

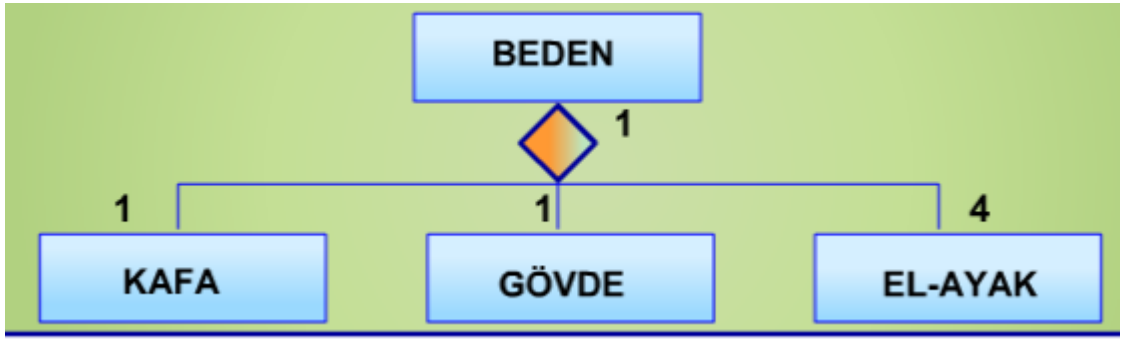
- Bazı sınıflar birden fazla parçadan oluşur.Bu tür özel ilişkiye "Aggregation" denir.Mesela ,bir TV 'yi ele alalım.Bir televizyon çeşitli parçalardan oluşmuştur.Ekran,Uzaktan Kumanda,Devreler vs.. Bütün bu parçaları birer sınıf ile temsil edersek TV bir bütün olarak oluşturulduğunda parçalarını istediğimiz gibi ekleyebiliriz. Aggregation ilişkisini 'bütün parça' yukarıda olacak şekilde ve 'bütün parça'nın ucuna içi boş elmas yerleştirecek şekilde gösteririz.



İÇERME (AGGREGATIONS)

Birleşik İlişkilendirme

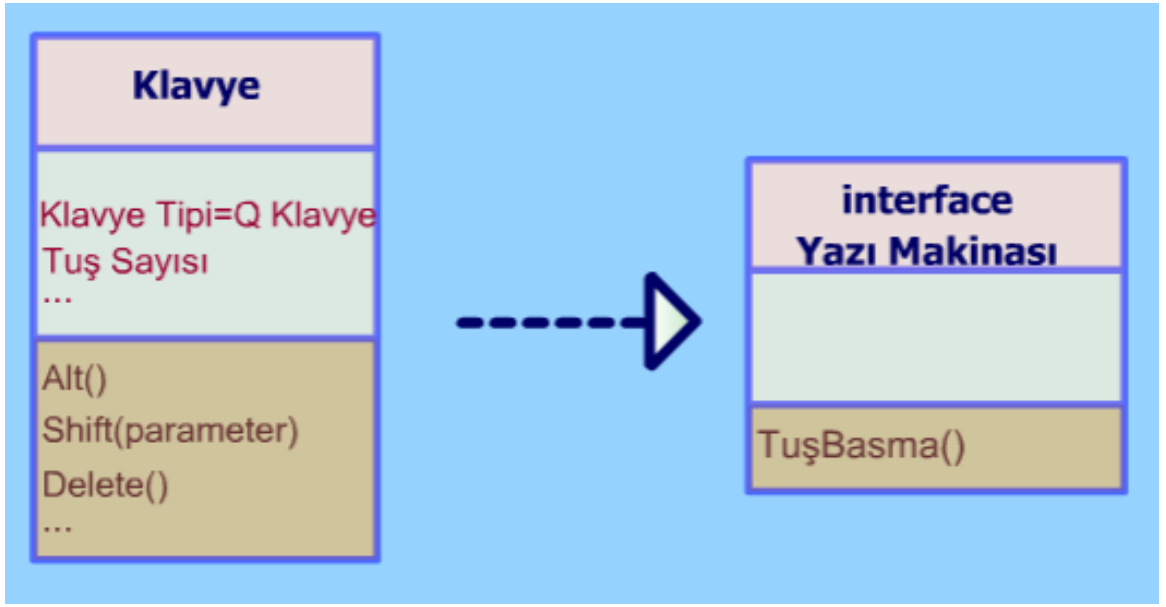
- Sadece "Beden" nesnesini oluşturup, sonradan bedene el, ayak, baş takmak çok mantıklı olmazdı. Bu tür ilişkilerin gösterilmesine ise **Birleşik İlişkilendirme** (Composite Assocation) denir. Bu ilişki türü diğerine göre daha sıkıdır. Bu tür ilişkilerde bütün nesne yaratıldığında parçalar da anında yaratılır. Bazı durumlarda, takılacak parçalar duruma göre değişebilir. Belirli koşullarda Kumanda, bazı durumlarda da Ekran olmayacaksa bu tür durumlar koşul ifadeleri ile birlikte noktalı çizgilerle belirtilir. Bu durumda takılacak parçalar **kısıt** (constraint) ile belirtilir.
- Gerçek bir Beden nesnesi oluştuğunda mutlaka ve mutlaka 1 Kafa, 1 Gövde ve 4 El_Ayak nesnesi yaratılacaktır. Görüldüğü gibi sıkı bir parça-bütün ilişkisi mevcuttur.



ARAYÜZ(INTERFACE)

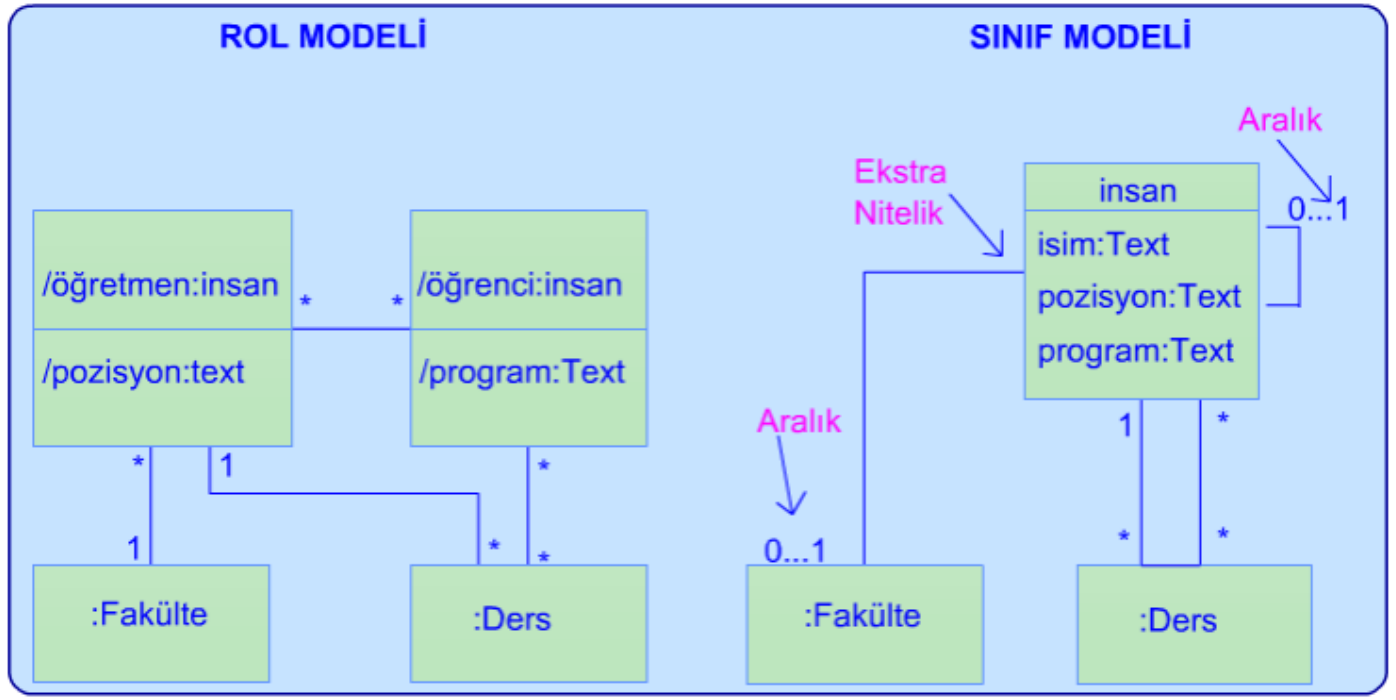
Bazı durumlarda bir sınıf sadece belirli işlemleri yapmak için kullanılır. Herhangi bir sınıfla ilişkisi olmayan ve standart bazı işlemleri yerine getiren sınıfa benzer yapılara arayüz(interface) denir.

- Arayüzlerin özellikleri yoktur. Yalnızca bir takım işleri yerine getirmek için başka sınıflar tarafından kullanılırlar.
- Mesela, bir "TuşaBasma" arayüzü yaparak ister onu "KUMANDA" sınıfında istersek de aşağıdaki şekilde görüldüğü gibi "KLAVYE" sınıfında kullanabiliriz.



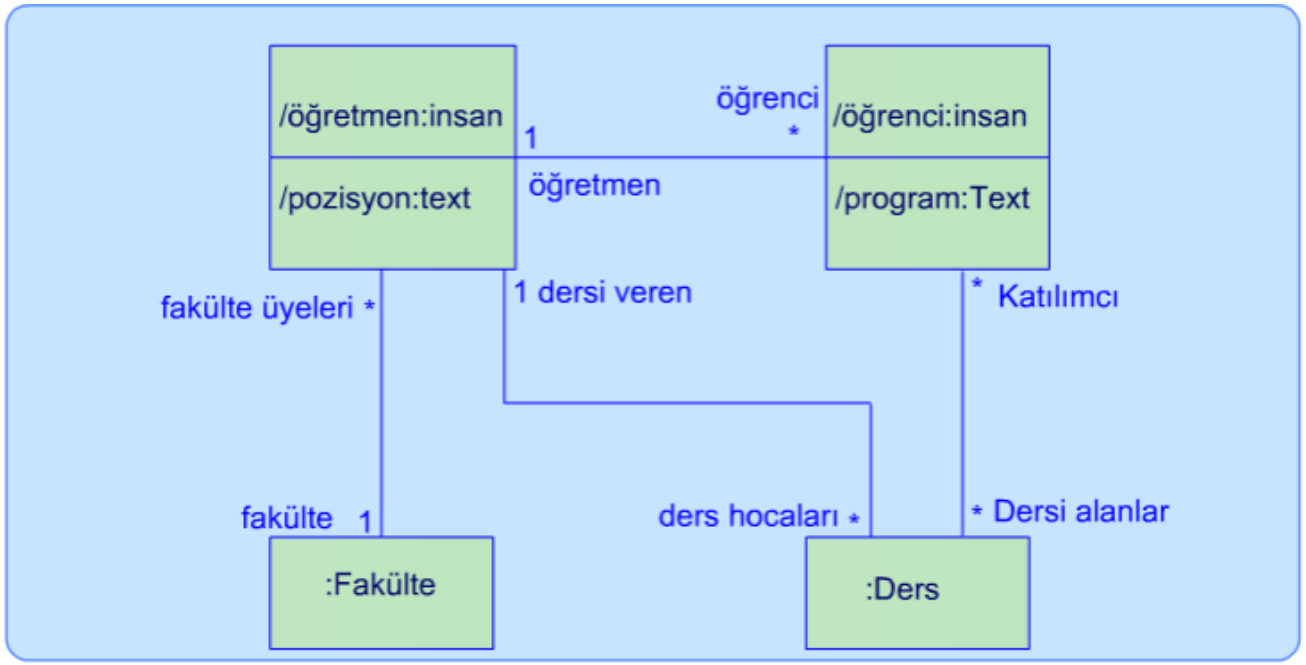
Sınıfların Sahip Olduğu Roller (Sınıflandırıcı-Olgu-Rol Modeli)

- Olgu tek bir kimliği olan ve hareketi, durumu belli olan bir varlıktır. Sınıflandırıcı ise olgunun tanımıdır. Sınıflandırıcı Rolü ise olgunun kullanımını tanımlamaktadır.
- Sınıflar bütün tanımlamayı verirken, Roller sadece bir kullanışı verir. Aşağıdaki şekil Rol modeli ile Sınıf modeline örnek vermektedir.



Sınıfların Sahip Olduğu Roller (Sınıflandırıcı-Olgu-Rol Modeli)

- Sınıf modelinde "İnsan" sınıfı isim, pozisyon ve program özelliklerine sahiptir, aynı zamanda Fakülte ve Ders sınıflarıyla da ilişkilidir. Rol modelinde ise "İnsan" sınıfı öğretmen ve öğrenci rollerine sahiptir. UML'de Rol adlarını belirtirken önce "/" karakteri sonra rol adı yazılır, ":" karakterinden sonra da sınıf adı yazılır.
- Yukarıdaki okul örneğini rol modelini göz önünde bulundurarak daha detaylı çizelim.

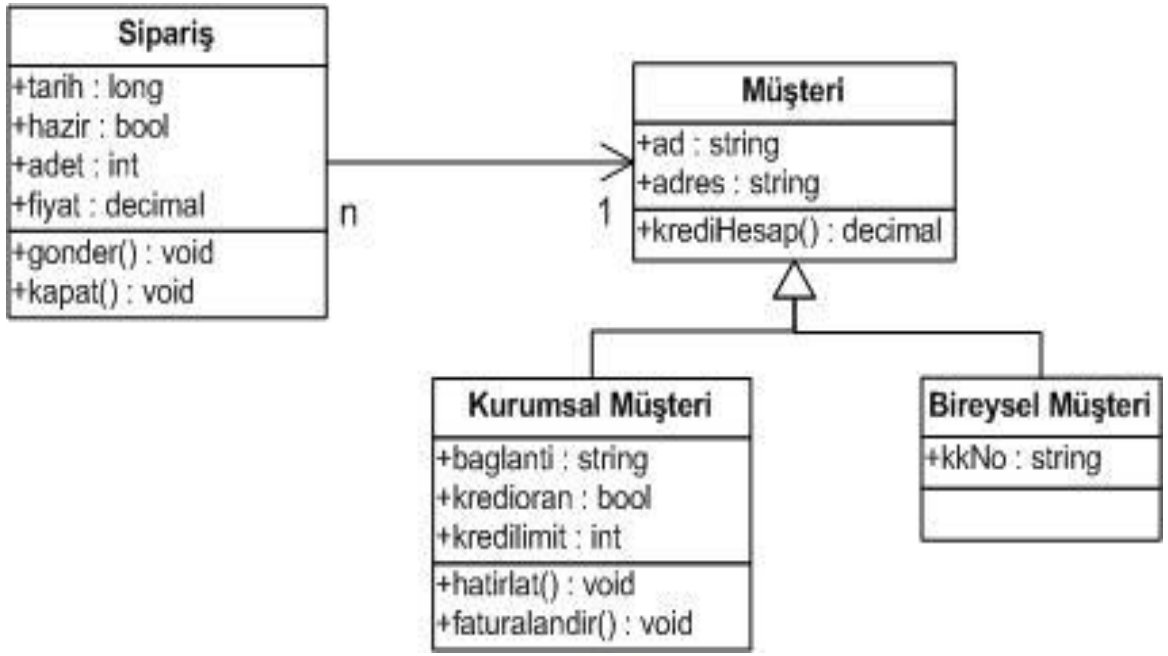


Sınıf UML Örnek 1

- UML diyagramları ile alanları (Özellikleri) "Adi ve Adresi", metotları " krediHesapla()" olan "**Müşteri**" isimli bir sınıfı, bu sınıftan **kalıtımsal** olarak "**Kurumsal Müşteri**" ve "**Bireysel Müşteri**" sınıflarını oluşturun. "Kurumsal Müşteri" sınıfının ayrıca "KrediLimit" özelliği ve "Fatura()" metodu, "Bireysel Müşteri" sınıfının ise "kartNo" özelliği bulunmaktadır. Müşteri Sınıfına 1-N bağlantılı, özellikleri "adet, tarih" ve metotları olan "kabulEt()" olan "**Sipariş**" isimli sınıfı da bu diyagram üzerinde gösterin.

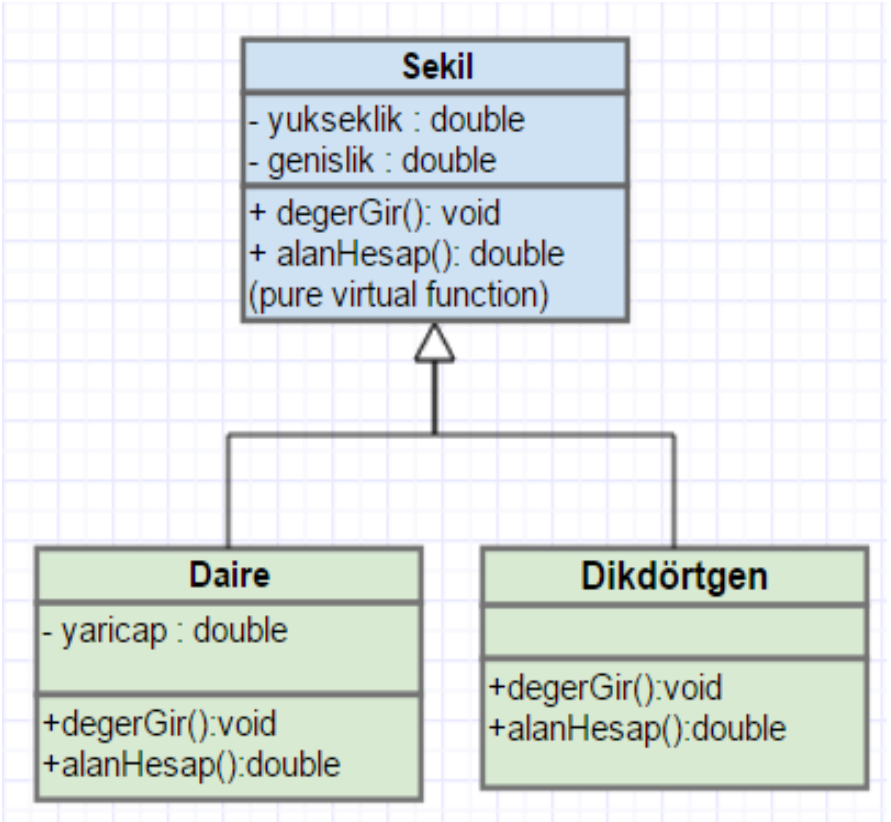
Sınıf UML Örnek 1

- UML diyagramları ile alanları (Özellikleri) "Adi ve Adresi", metotları " krediHesapla()" olan "**Müşteri**" isimli bir sınıfı, bu sınıftan **kalıtımsal** olarak "**Kurumsal Müşteri**" ve "**Bireysel Müşteri**" sınıflarını oluşturun. "Kurumsal Müşteri" sınıfının ayrıca "KrediLimit" özelliği ve "Fatura()" metodu, "Bireysel Müşteri" sınıfının ise "kartNo" özelliği bulunmaktadır. Müşteri Sınıfına 1-N bağlantılı, özellikleri "adet, tarih" ve metotları olan "kabulEt()" olan "**Sipariş**" isimli sınıfı da bu diyagram üzerinde gösterin.



Sınıf UML Örnek 2

- Yandaki UML diyagramında bir “Sekil” Sınıfından kalıtımsal olarak “Daire” ve “Dikdörtgen” Sınıfları tanımlanmıştır. “Sekil” sınıfı “abstract sınıftır” ve bu sınıftaki “alanHesap” metodu, saf sanal fonksiyondur. Bu metot ile, şeklin alan değeri, “Daire” sınıfında $alan = \pi r^2$ (r: yarıçap) olarak ve “Dikdörtgen” sınıfında ise $alan = a * b$ (a: yükseklik, b: genişlik) olarak hesaplanacaktır. degerGir() metodu bir üst sınıftan kalıtımsal olarak bütün sınıflarda tanımlanacaktır (her sınıfın özellikleri dahil edilerek). Bu UML diyagramını verecek olan C++ kodunu oluşturun.



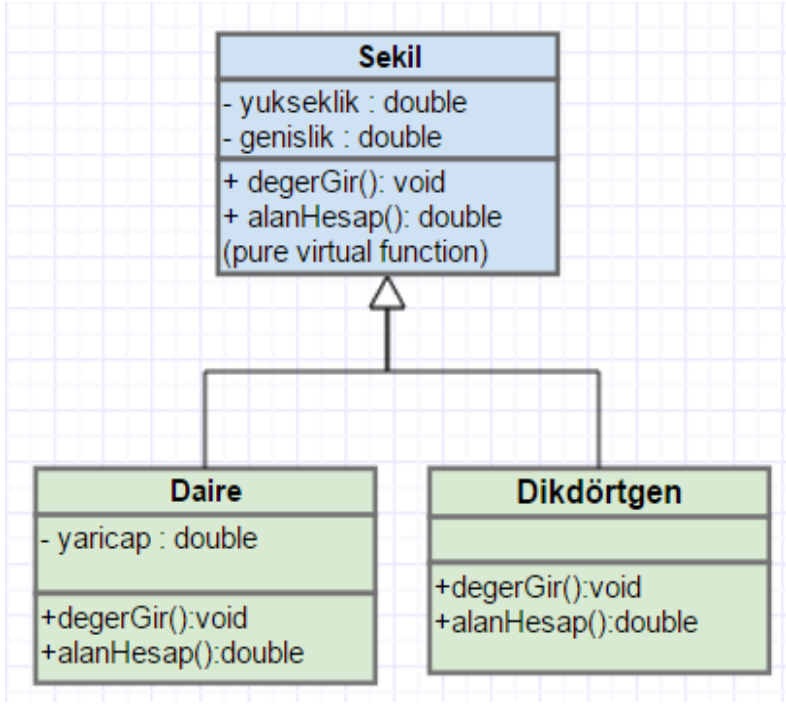
Sınıf UML Örnek 2

```
#include <iostream>
using namespace std;
 class Shape {
      public:
           float width;
           float height;
           virtual float getarea() {} ; };
class Circle: public Shape {
      public:
           Circle(float radius);
           float getarea() ; };
class Rectangle: public Shape {
      public:
           Rectangle(float width, float height);
           float getarea() ; };
```

```
class Square: public Rectangle {
      public:
           Square(float size); };

Square::Square (float size) : Rectangle(size, size) {
}
Rectangle::Rectangle(float width, float height) {
      this->width = width; this->height = height; }
Circle::Circle (float radius) {
      this->width = this->height = radius * 2.0; }

float Circle::getarea() { return width * width *
3.14159265 / 4.0; }
float Rectangle::getarea() { return width * height
; }
```

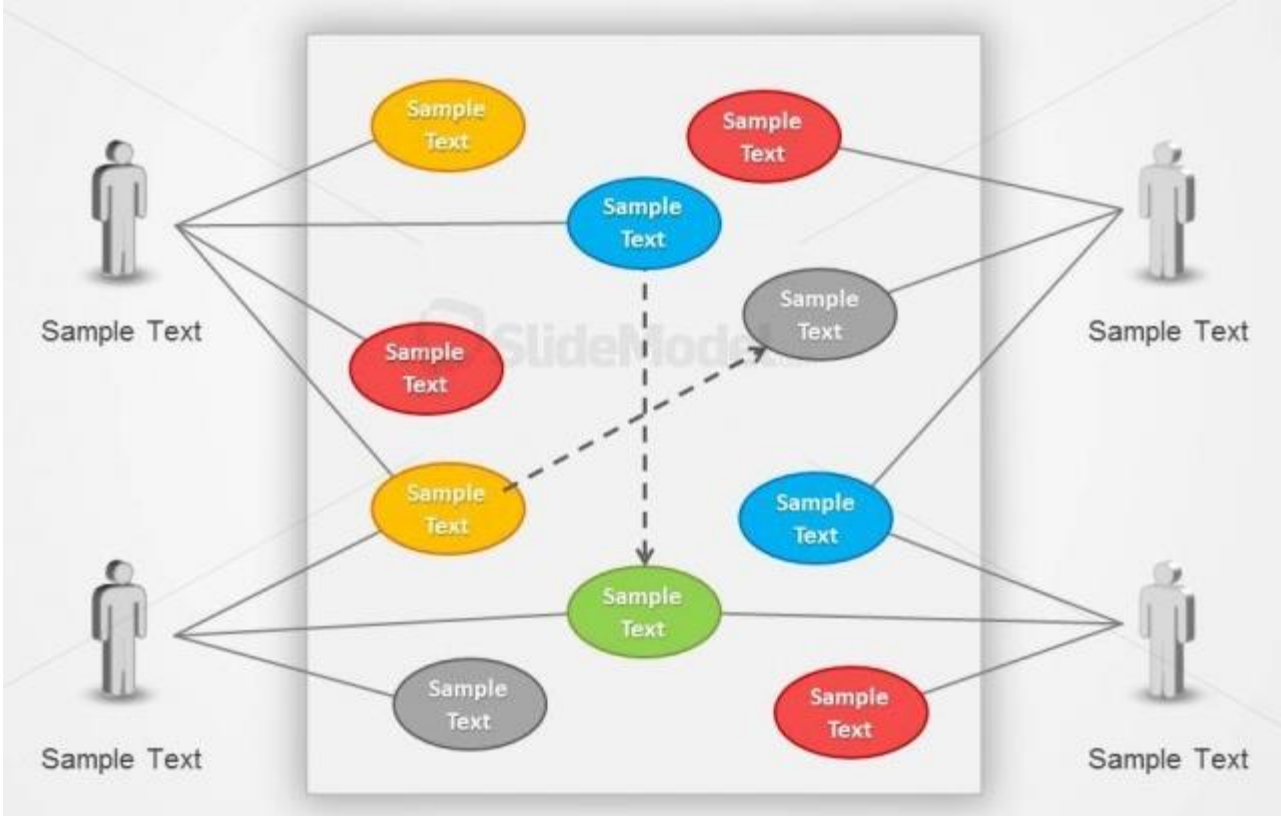


UML DİYAGRAMLARI

KULLANIM DURUMLARI (USE CASE) DİYAGRAMLARI

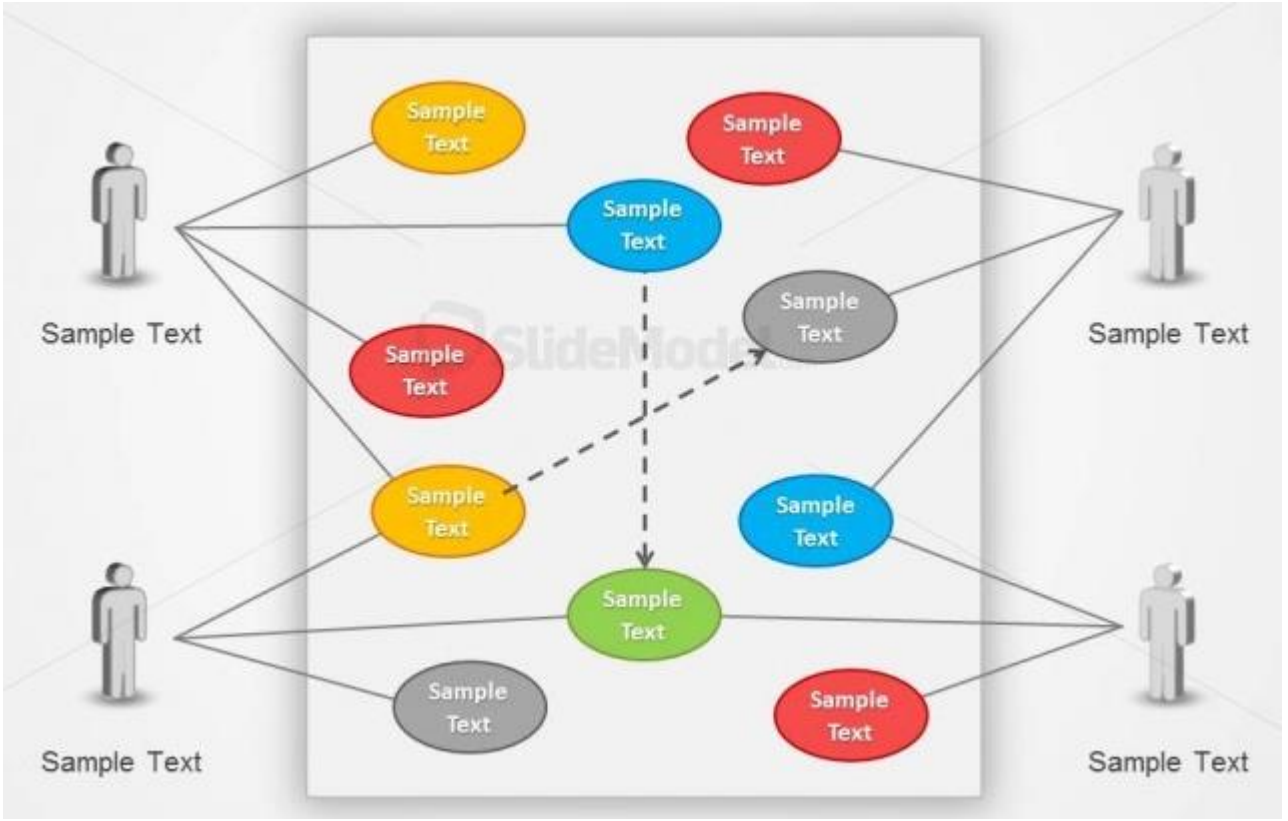
KULLANIM DURUMU MODELİ

- Analiz aşaması projeler için hayati önem taşır. İyi bir analizden geçmemiş projelerin başarı şansı azdır. Analiz ile birlikte kendimize “Ne?” sorusunu yöneltirken, sistemin fonksiyonel gereksinimlerini yakalamaya çalışırız. Use Case UML içerisinde yer alan ve analiz aşamalarında sıkça kullanılan bir tekniktir.



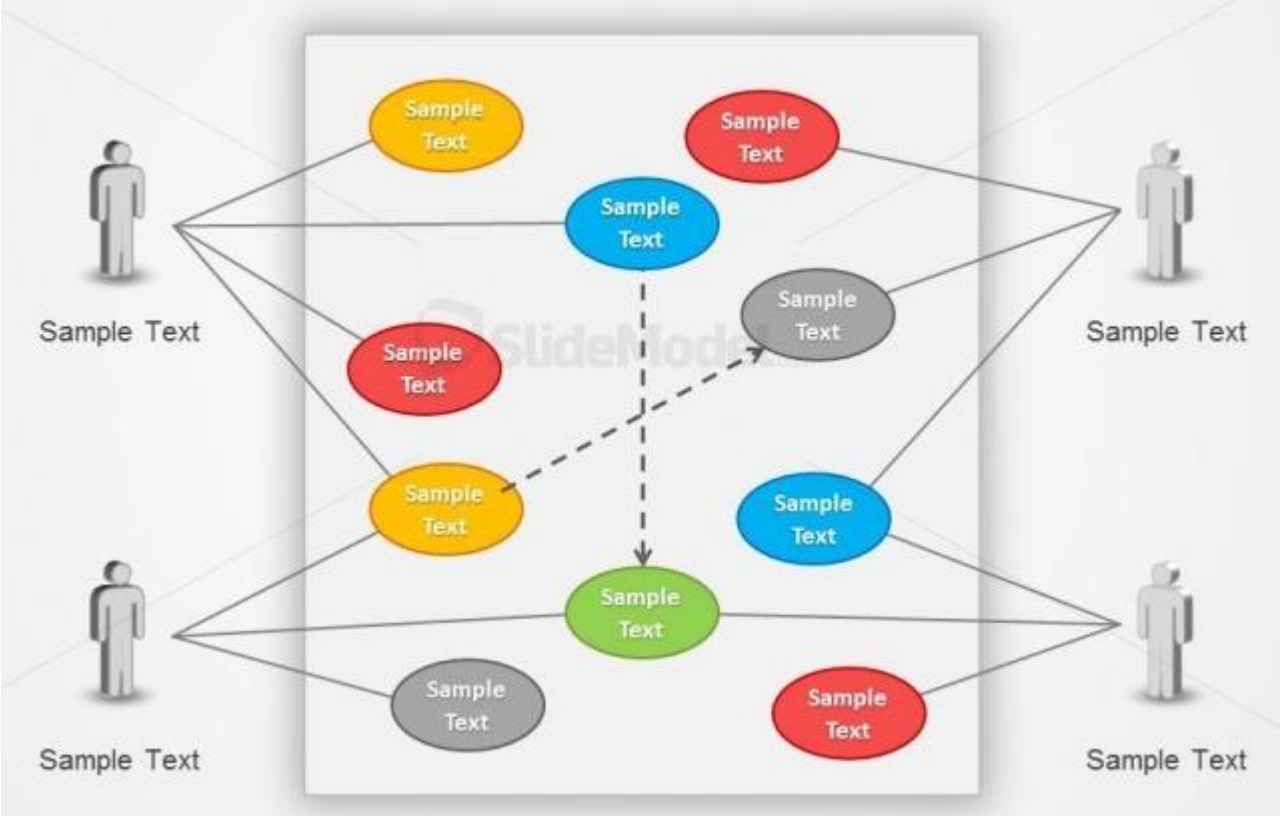
KULLANIM DURUMU MODELİ

- Bir sistemi modellemek, gereksinimleri ortaya koymak ve bunun sonucunda istenilen ürünü tam olarak gerçekleştirmek için en önemli nokta sistemde gerçekleştirilecek olan dinamik davranışları yakalamaktır. Sistemin tasarlanmasında ihtiyaçları ayrıntılı olarak açıklığa kavuşturmak için bahsedilen bu dinamik davranış, sistemin çalıştırıldığındaki davranışını ifade eder.
- Sistem gereksinimlerinin anlatılmasında kullanılan Kullanım Durumu Diyagramları, sistem kullanıcısının bakış açısıyla sistemin davranışlarının diyagramlarla modellenmesidir.



KULLANIM DURUMU MODELİ

- Sınıf Diyagramları'yla bir sistemin ancak durağan bir analizi yapılabilirken, Kullanım Durumu Diyagramları'yla sistemin ve sınıfların zamanla değişimini de içerecek biçimde model oluşturulur. Bu diyagramlar sistemin kabaca ne yaptığını gösterirler; nasıl ve neden yapıldığını göstermezler. Aynı zamanda profesyonel yazılımcılar kadar, meslekten olmayan kişiler tarafından da kolayca anlaşılırlar.

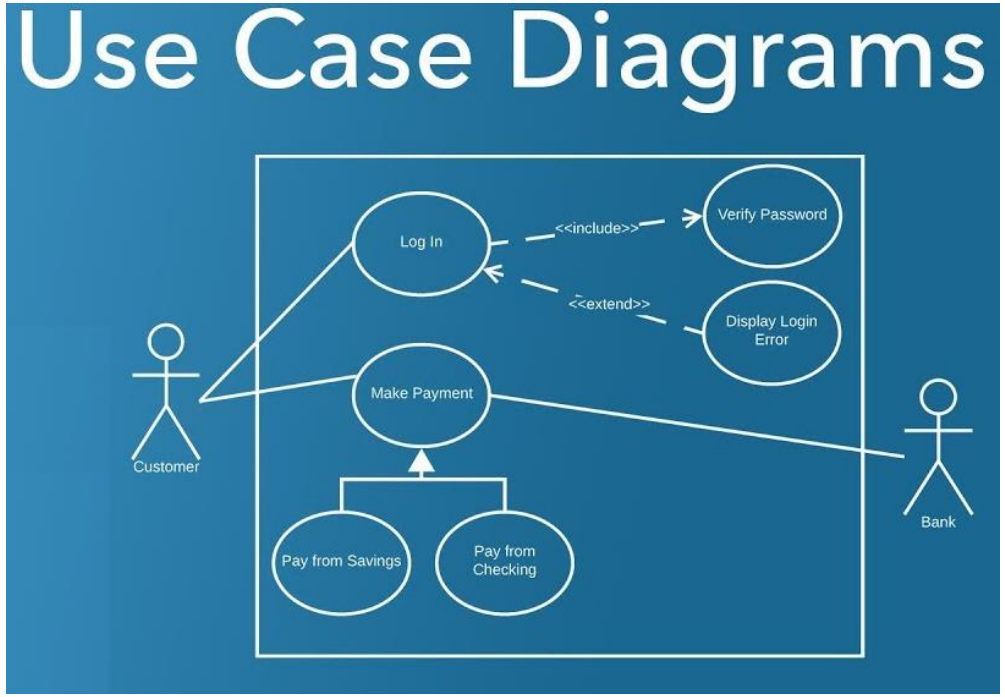


KULLANIM DURUMU MODELİ

- Use case diyagramları aktörler, kullanım örnekleri ve bunların ilişkilerinden oluşur. Diyagram, bir uygulamanın sistemini / alt sistemini modellemek için kullanılır. Tek bir kullanım durumu diyagramı, bir sistemin belirli bir işlevselliğini yakalar.
- Kullanım durumu diyagramları (use-case diyagramları), iç ve dış etkileri de içeren bir sistemin gereksinimlerini toplamak için kullanılır. Bu gereksinimler çoğunlukla tasarım gerekliliklerdir.
- Dolayısıyla bir sistem, fonksiyonlarını toplamak için analiz edildiğinde, kullanım örnekleri hazırlanır ve aktörler tanımlanır.
- Kullanım senaryosu diyagramları sistemin işlevsel gereksinimlerinin ortaya çıkarılması için kullanılır.
- Sistemin kullanıcısının bakış açısıyla modellenmesi amacıyla kullanılır.
- Kullanım senaryosu diyagramları sistemin kabaca ne yaptığı ile ilgilenir, kesinlikle nasıl ve neden yapıldığını incelemez.

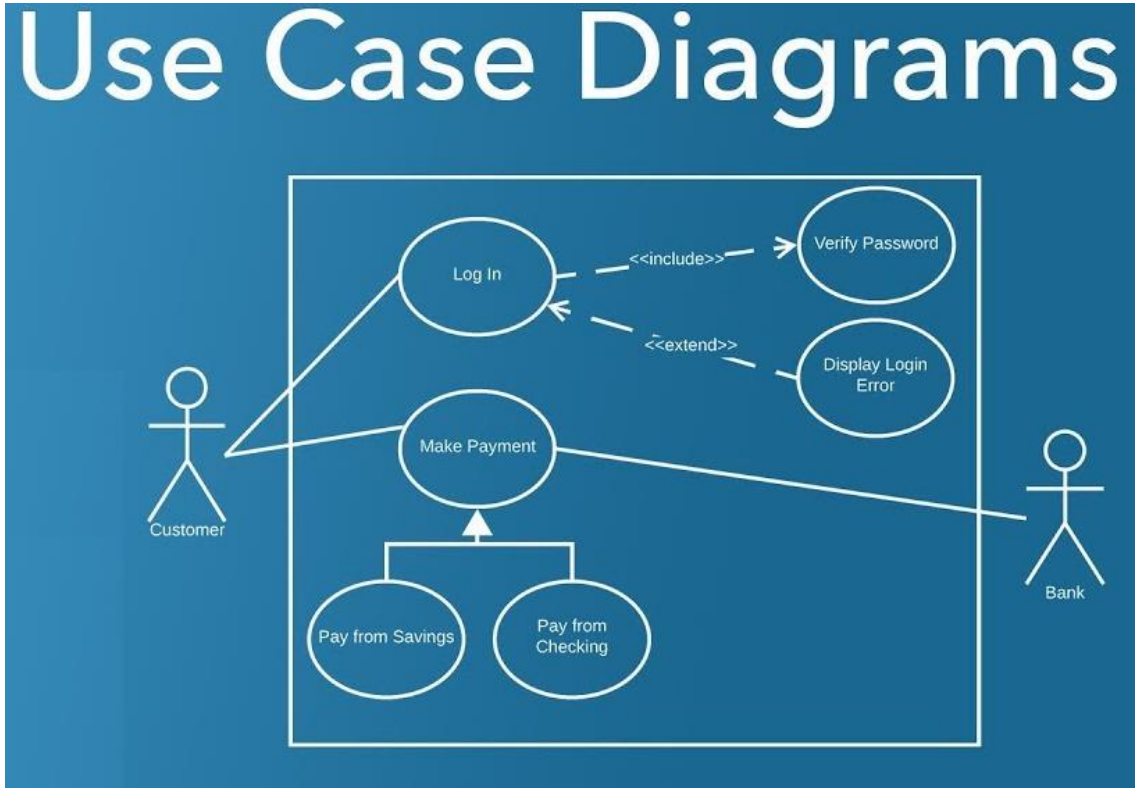
Kullanım Durumu Diyagramlarının Amacı (Use-Case diyagramları)

- Bir sistemin gereksinimlerini toplamak için kullanılır. Toplanan gereksinimler üzerinde anlaşmayı kolaylaştırır.
- Bir sistemin dış görünümünü elde etmek için kullanılır.
- Sistemi etkileyen dış ve iç faktörleri tanımlamak için kullanılır.
- Gereksinimler arasındaki etkileşimi aktörler bazında göstermek için kullanılır.
- Use case diyagramları yazılım ekip üyeleri arasındaki iletişimi geliştirir.
- İş süreçlerinin anlaşılmasını kolaylaştırır.



Kullanım diyagramlarının kullanıldığı yerler

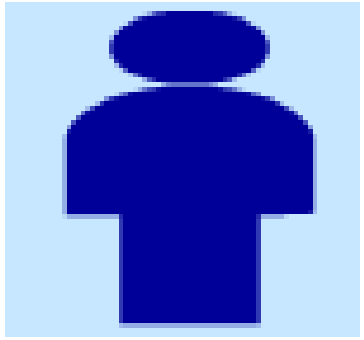
- Gereksinim analizi ve üst düzey tasarım.
- Bir sistem bağlamını modelleyin.
- Tersine mühendislik.
- İleri mühendislik.



Kullanım Durumu Modelini Oluşturan Kısımlar

- **Bir Kullanım Durumu Modeli,** Kullanım Durumu Diyagramları ve Kullanım Durumu Açıklamaları dediğimiz senaryolardan oluşmaktadır.
- **Kullanım Durumu Diyagramları** Aktörler, Kullanım Durumu ve Kullanım Durumları arasındaki ilişkilerden oluşmaktadır. Tablo 1, Kullanım Durumu Diyagramları'nda kullanılan notasyonu göstermektedir.

Kullanım Durumu Modelini Oluşturan Kısımlar



- **Sistem:** Use Case'lerin yer aldığı sınırları ve kapsamı belli, aktörlerin üzerinde çalıştığı yer sistemdir. Sistem projelerinizin kapsamını belirtmesi açısından önemlidir.
- **Aktör :** Aktör belirlenen sistem üzerinde bir rol oynayan ve etkileşime gecen her türlü ögedir. Burada aktör sistemi kullanan bir site ziyaretçisi olabileceği gibi sistemden etkilenen dış sistemler yani kargo şirketi de bir aktör olarak yerini alır. Aktör sistemi "uyarır" ,işlevleri haricen "tetikler"(aktif) yada sistemden "uyarıcı alır"(pasif). Aktör sistemin parçası değildir, "harici" dir.

Kullanım Durumu Modelini Oluşturan Kısımlar

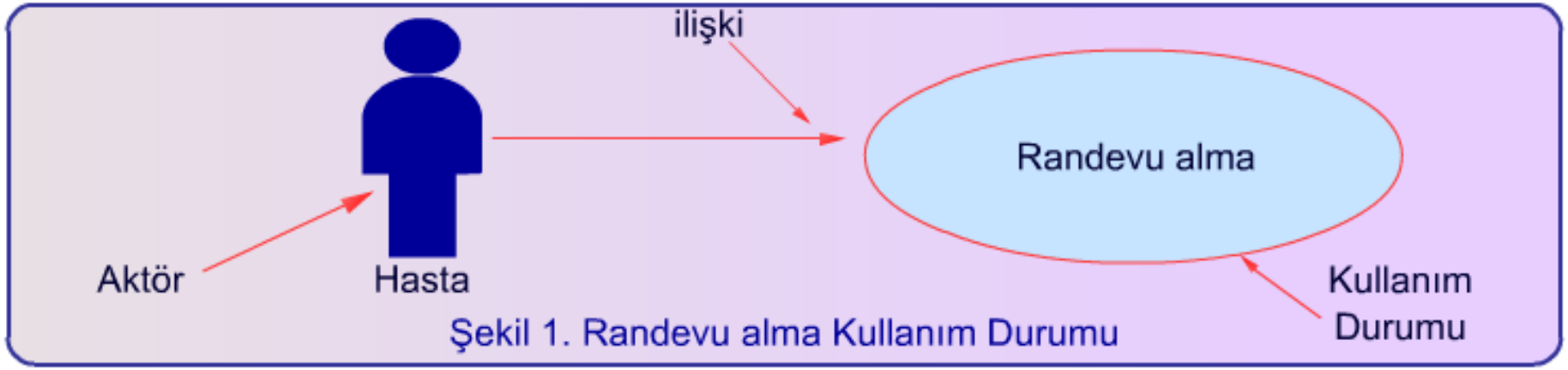
Kullanım Durumları (Use Case)

- Ortak bir kullanıcı hedefi etrafında oluşturulmuş bir takım senaryoların bütünleşmesinden meydana gelen yapıdır. Use caseler temelde senaryolardan oluşur, fakat her senaryo yukarıda verilen örnek gibi başarılı ve tek düze yürümez. Senaryoların bazı istisnai durumları ve alt senaryoları içermesi gerekmektedir. Bu durumda Use Case verilen senaryoların ötesinde bazı artı özellikler de taşır. Örnek olarak müşterinin ödeme işlemini havale ya da kredi kartı olarak yapabilmesi, kredi kartı onayında hata alınması ve teslimat seçenekleri gibi istisnai durum ve alternatif yaklaşımların başarılı senaryo üzerinde toparlanması işlemi UC lerin sorumluluğu altındadır..

Kullanım Durumu Modelini Oluşturan Kısımlar

Kullanım Durumları (Use Case)

- Kullanım Durumları Şekil1'de de görüldüğü gibi, UML'de elips şekliyle ifade edilirler. Aktör, yani kullanıcı, Kullanım Durumu Modeli'nde bir Kullanım Durumu'nu başlatır ve sonuç olarak bir değeri başka bir kullanıcıya verir. Kullanıcıların altında kullanıcıların adı bulunur. Kullanıcı ve Kullanım Durumu arasındaki ilişkiyi belirtmek için ise düz bir çizgi çizilir..



Şekil 1. Randevu alma Kullanım Durumu

Kullanım Durumu Modelini Oluşturan Kısımlar

Senaryolar

- Sistem ve aktör (kullanıcı) arasında geçen etkileşimi anlatan bir dizi adım'a (operasyon) senaryo adı verilir. Basit bir senaryo örneği verecek olursak, online bir mağazadan urun satın alma işlemini düşünebiliriz.
- Kullanım Durumu Modeli'ni oluşturan diğer önemli bir yapı ise Senaryolar'dır. Her Senaryo bir olay dizisini tanımlar. Senaryolar kullanıcı tarafından başlatılan çeşitli olaylar dizisidir. Her ne kadar bazı sistemlerde bir olay ya da senaryo, kullanıcı tarafından değil de donanım tarafından veya belirli bir zaman geçmesi sonucu başlatılsa da, sistemi tasarlamamız için Kullanım Durumları'ndan faydalanırız.

Kullanım Durumu Modelini Oluşturan Kısımlar

Senaryolar

Kullanım Durumu'na ait her senaryo aşağıdaki ifadelerden bir sonuç bekler:

- **Kullanım Durumu'nu başlatan kullanıcıdan,**
- **Kullanım Durumu'nun başlaması için gereken ön bilgilerden,**
- **Senaryo'daki adımlardan,**
- **Senaryo bittikten sonraki durumlardan,**
- **Kullanım Durumu'ndan faydalanacak aktör durumundaki nesnelerden.**

Kullanım Durumu Modelini Oluşturan Kısımlar

Kullanım Durumları Arasındaki Bağıntılar

- UCler arasında kurulan ilişkiler, bu ilişkiler ileriki bölümlerde daha detaylı anlatılacaktır. İlişkilerin yardımı ile UCler üzerinde genelleme, kapsama (inclusion) ve genişleme (extension) gibi gösterimler yapılabilir.

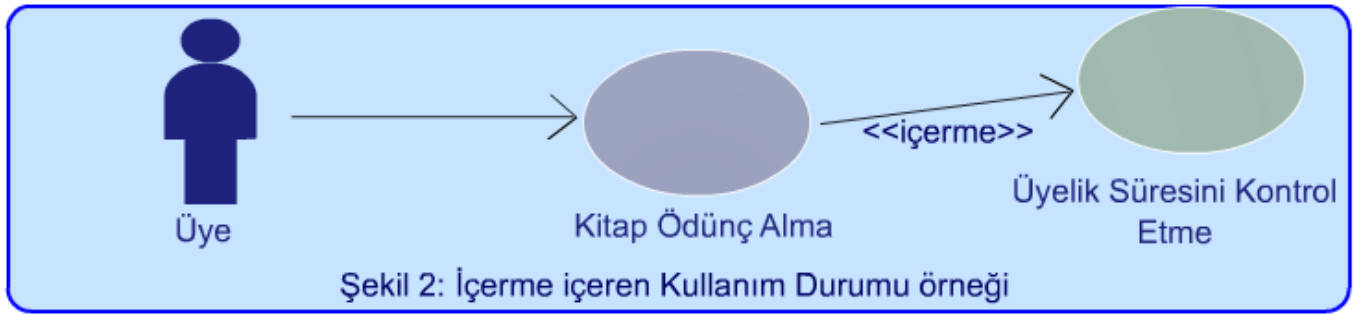
Kullanım durumları tekrar kullanılabilen birimlerdir ve aralarında dört türlü bağıntı vardır:

- İçerme (Inclusion)
- Genişleme (Extension)
- Genelleştirme (Generalization)
- Gruplama (Grouping)

Kullanım Durumları Arasındaki Bağıntılar

İçerme (Inclusion)

- Bu metodla bir Kullanım Durumu içindeki adımlardan biri, başka bir Kullanım Durumu içinde kullanılabilir. İçerme yöntemini kullanmak için <<içerme (inclusion)>> ifadesi kullanılır. Bu ifade, sınıf çizelgelerindeki sınıfları birbirine bağlayan noktalı çizgilerin üzerine yazılır.
- Aşağıdaki örnekte Kitap Ödünç Alma Kullanım Durumu, Üyelik Süresi Kontrol Kullanım Durumu'nu içerir; çünkü kitap ödünç almak isteyen bir kişinin önce üyelik süresi kontrol edilir. Eğer üyelik süresi dolmamışsa ödünç verilir. Bu UML'de aşağıdaki gibi gösterilir:



Şekil 2: İçerme içeren Kullanım Durumu örneği

Kullanım Durumları Arasındaki Bağıntılar

Genişleme (Extension)

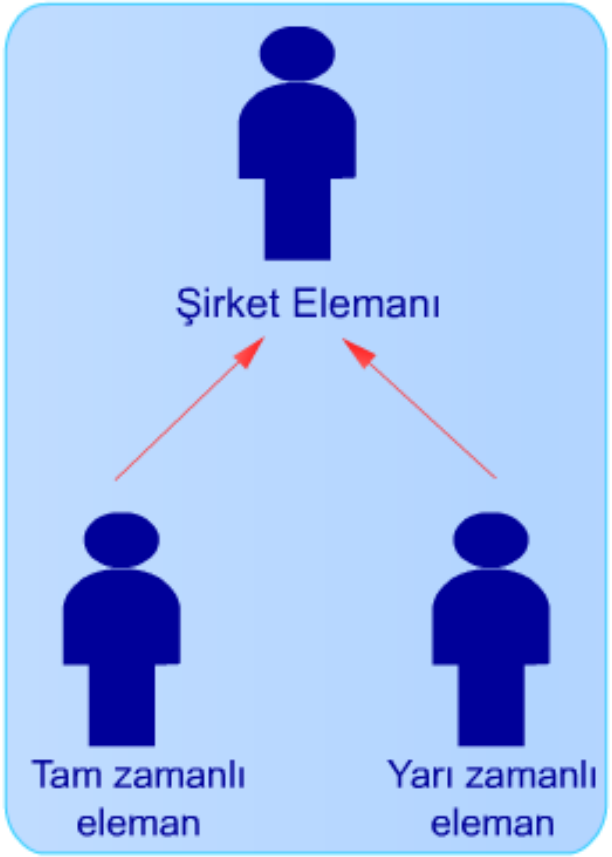
- Bir Kullanım Durumu, başka bir Kullanım Durumu'nun özelliklerini belli koşullarda barındırıyorsa, buna **Genişletilmiş Kullanım Durumu** denir.
- Bu metodla varolan bir Kullanım Durumu'na yeni adımlar ekleyerek yeni bir Kullanım Durumu yaratılır. İçermede olduğu gibi genişlemeleri göstermek için yine Kullanım Durumları arasına noktalı çizgiler konur ve üzerine **<<genişleme (extension)>>** ibaresi yazılır. Aşağıdaki örnekte, eğer kütüphane üyesi kendisine tahsis edilen ödünç kitap alma sayısını aşarsa, ödünç alma istemini reddetme Kullanım Durumu devreye girer.

Şekil 3: Genişleme içeren Kullanım Durumu örneği

Kullanım Durumları Arasındaki Bağıntılar

Genelleme (Generalization)

- Bilindiği gibi sınıflarda türetme (inheritance) kavramı vardır. Türetme ile oluşturulan sınıf, ana sınıfın tüm özelliklerini alır ve istenirse yeni sınıfa yeni özellikler eklenebilir.
- Kullanım Durumları arasındaki bu ilişki de sınıflarda türetme kavramına benzemektedir. Gösterim biçimi, sınıf diyagramlarındaki türetmenin gösteriliş şekliyle aynıdır. Kullanım Durumları arasında düz bir çizgi çizilir ve çizginin taban (base) sınıfı gösteren ucuna içi boş bir üçgen çizilir.
- Yandaki şekilde bir şirkette çalışan şirket elemanlarının genellemesinin nasıl yapıldığı görülebilir.
- Şirket çalışanları saat bazında çalışan yarı-zamanlı elemanlardan ve tam zamanlı çalışan elemanlardan oluşmaktadır. Her iki tür eleman da Şirket Elemanı olarak genelleştirilebilir.



Kullanım Durumları Arasındaki Bağıntılar

Gruplama (Grouping)

- Gruplama metodu tamamen fiziksel olarak bazı Kullanım Durumları'nı bir arada toplamak için kullanılır. Gruplama tekniği genellikle birkaç alt sistemden oluşan büyük sistemlerde kullanılır. İlgili Kullanım Durumları dikdörtgen içine alınarak gruplanır.

Kullanım Durumları ÖRNEKLERİ

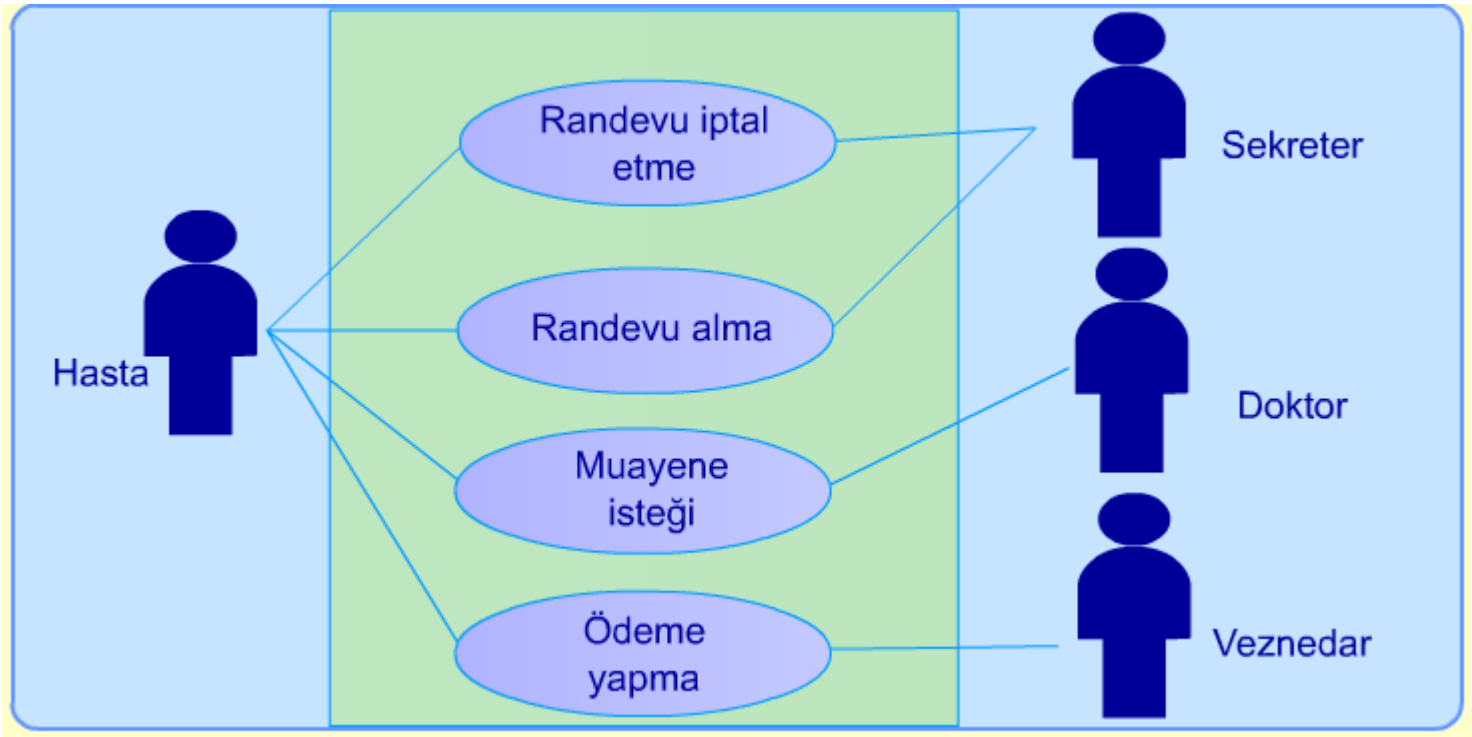
Aşağıdaki örnekte Hastanın muayene olma işlemi için gerekli Kullanım Durumuları bir araya getirilerek gruplanmıştır. Bunlar:

- Randevu alma Kullanım Durumu
- Randevu iptal etme Kullanım Durumu
- Muayene isteği Kullanım Durumu
- Ödeme yapma Kullanım Durumu

Senaryodaki aktörler ise:

- Hasta
- Sekreter
- Doktor
- Veznedar
- olarak belirlenmiştir.

Hasta sekreterden randevu alır, doktor tarafından muayene edilir ve veznedara muayene ücretini öder. Bütün bu işlemleri modelleyen Kullanım Durumu Diyagramı aşağıdaki gibidir.



Kullanım Durumları ÖRNEKLERİ

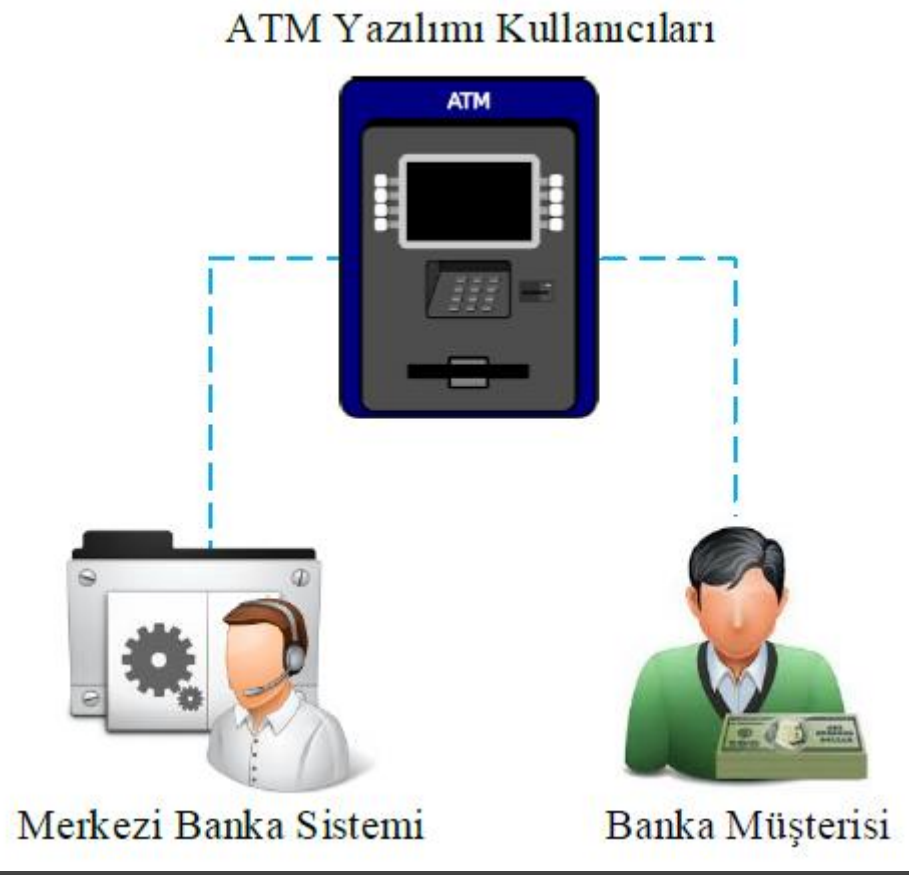
- Bir bankanın ATM cihazı için yazılım geliştirilecektir. ATM, banka kartı olan müşterilerin hesaplarından para çekmelerine, hesaplarına para yatırmalarına ve hesapları arasında para transferi yapmalarına olanak sağlayacaktır. ATM, banka müşterisi ve hesapları ile ilgili bilgileri, gerektiğinde merkezi banka sisteminden alacaktır.
- ATM uygulama yazılımının kullanıcıları:

Aktörler

- Banka müşterisi
- Merkezi Banka Sistemi

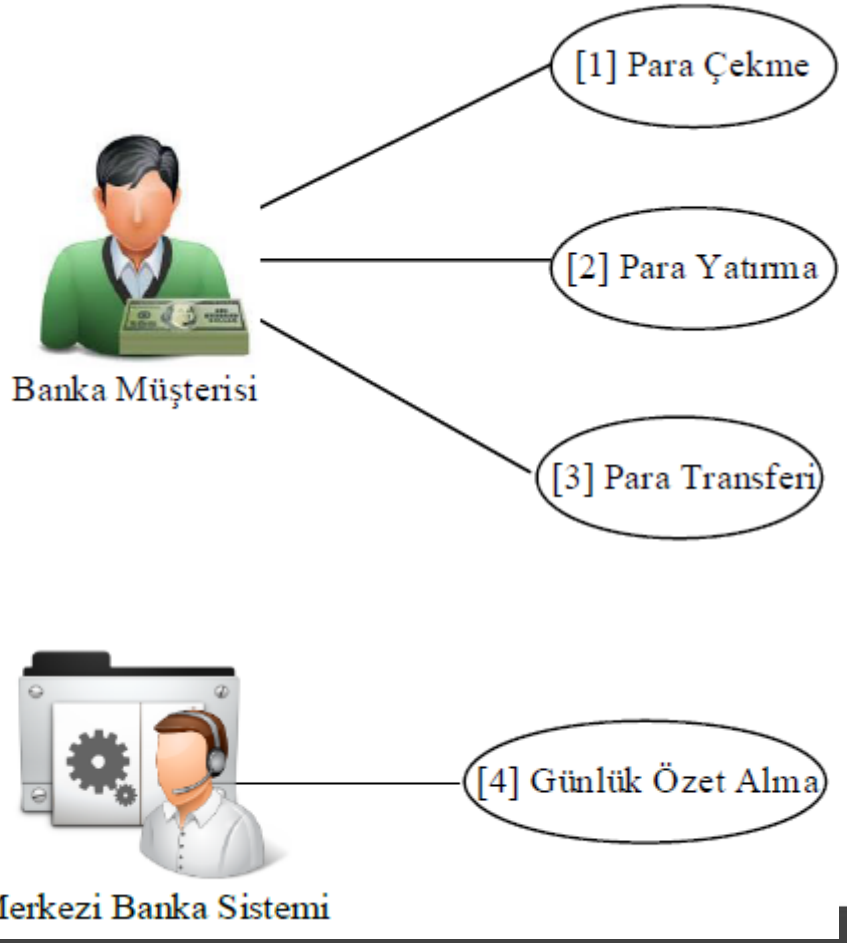
Kullanım Durumları ÖRNEKLERİ

- **Belirlenen aktörler ATM'den ne istiyorlar?**
- **Aktör: Banka müşterisi**
  - Para çekme
  - Para yatırma
  - Para transferi
- **• Aktör: Merkezi Banka Sistemi**
  - Günlük özet alma

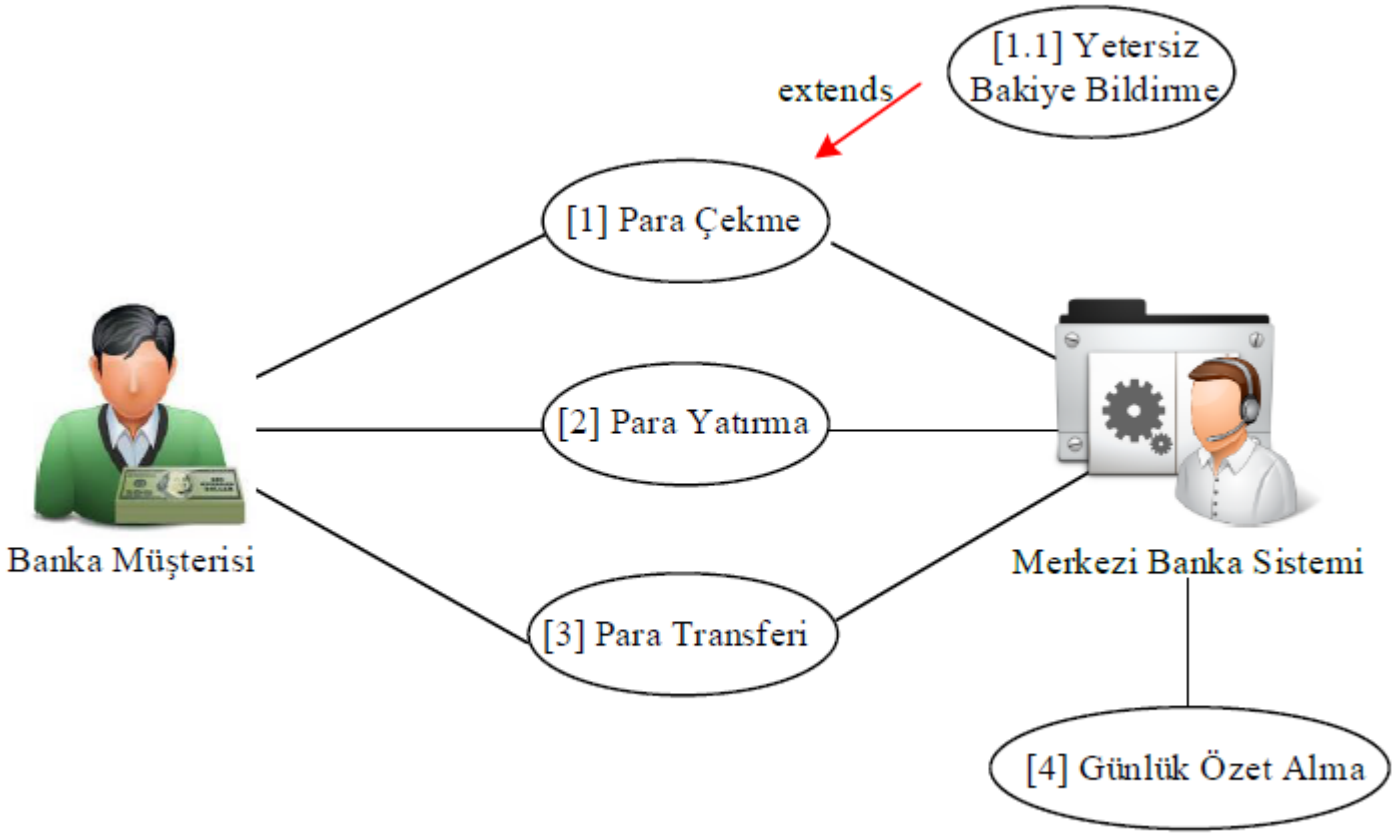


Kullanım Durumları ÖRNEKLERİ

- **Belirlenen aktörler ATM'den ne istiyorlar?**
- **Aktör: Banka müşterisi**
  - Para çekme
  - Para yatırma
  - Para transferi
- **• Aktör: Merkezi Banka Sistemi**
  - Günlük özet alma



Kullanım Durumları ÖRNEKLERİ



Use-case Diagram

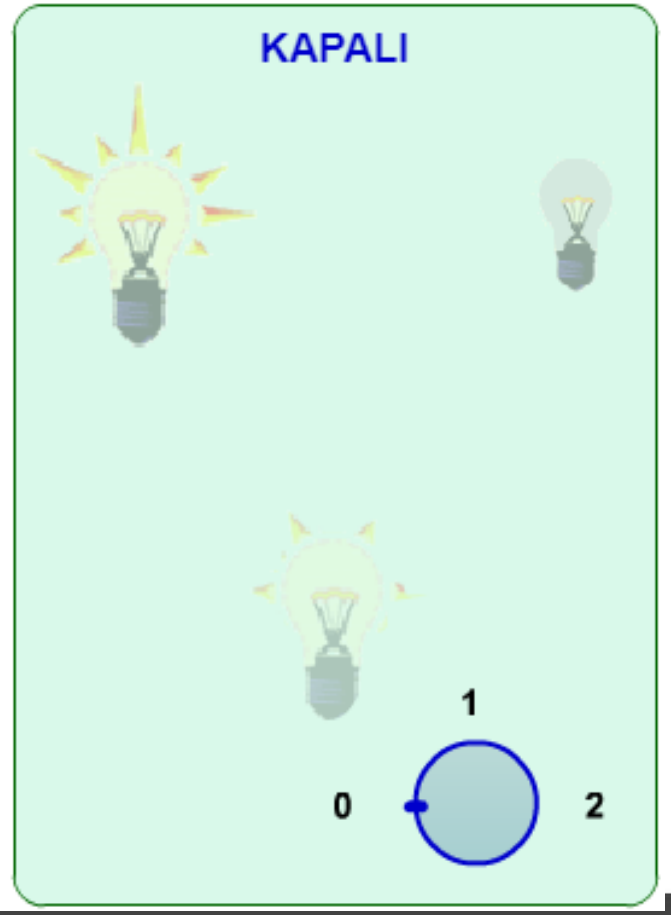
UML DİYAGRAMLARI

DURUM (STATE) DİYAGRAMLARI

UML DURUM DİYAGRAMLARI

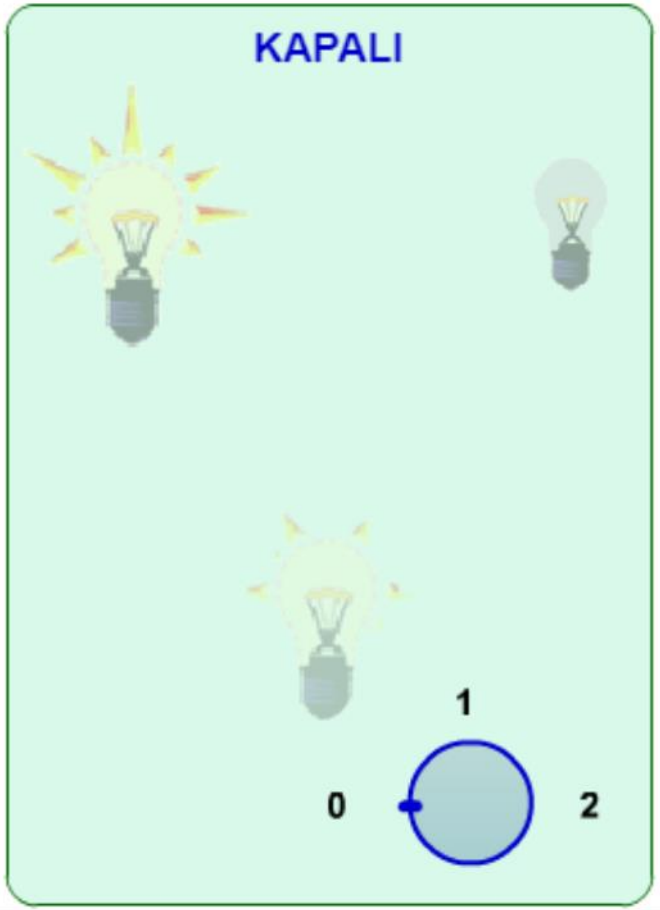
- State diyagramları genel olarak sistemlerin davranışlarını gösterir. Her diyagram tek bir sınıfın nesnelerini ve sistem içerisinde nesnelerin durumlarını (state), geçişlerini (transition), olaylarını (Events) içermektedir.
- Mesela bir lambayı düşünelim, lambanın yaşam süresince 3 farklı durumda olabileceğini söyleyebiliriz:

1. durum: Lambanın kapalı olması,
2. durum: Lambanın açık olması,
3. durum: Lambanın düşük enerji harcayarak yanması



UML DURUM DİYAGRAMLARI

- Bu 3 durum arasında geçişler mümkündür. Bu geçişleri sağlayan ise çeşitli olaylardır. Örneğin lambayı kapalı durumdan açık duruma getirmek için lambanın anahtarını çevirmek gerekir. Aynı şekilde lambayı düşük enerjili halde yanan durumundan açık durumuna getirmek için lamba anahtarını yarım çevirmek gerekir.
- Lamba nesnesinin durum değiştirirken yaptığı işlere Aksiyon (action), lambanın durum değiştirmesini sağlayan mekanizmaya ise olay (event) denilmektedir.
- Sınıf Diyagramları ile nesnenin statik modellemesi yapılırken, Durum Diyagramları ile bir nesnenin dinamik modellemesi yapılmaktadır. Yani Dinamik modellemede nesnenin ömrü boyunca gireceği durumlar belirlenmektedir.



Durum Diyagramlarında Kullanılan Temel Kavramlar

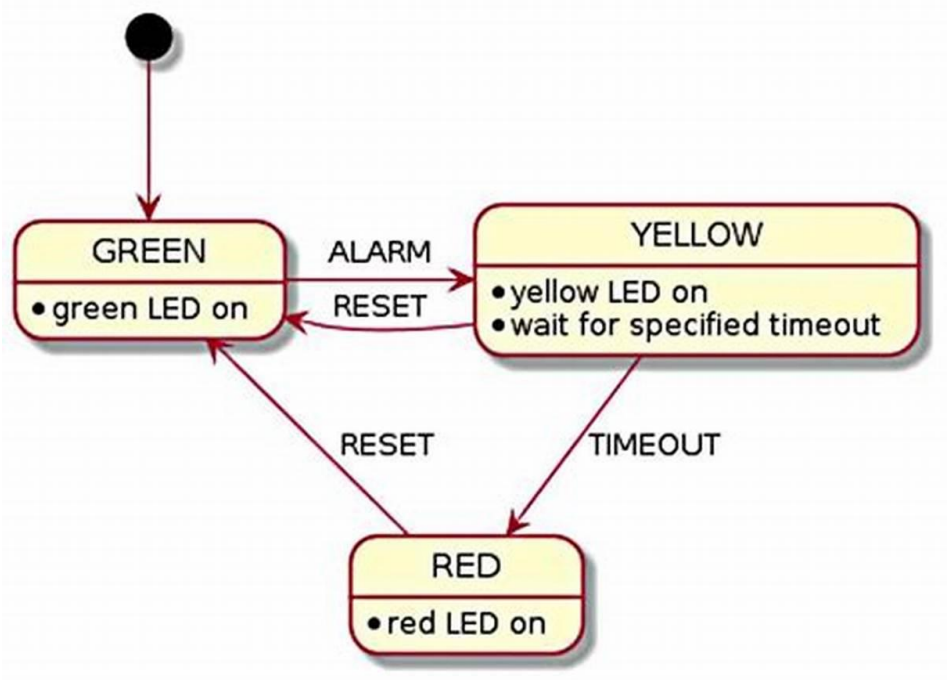
- Durum diyagramları bir sistemin davranışlarını modeller ve bir olay gerçekleştiğinde olası tüm durumları tanımlar. Her bir diyagram bir sınıfın tek bir nesnesini ele alır ve sistem içerisindeki farklı durumlarını irdeler. Bir durum diyagramı aşağıdaki elemanlardan oluşur:

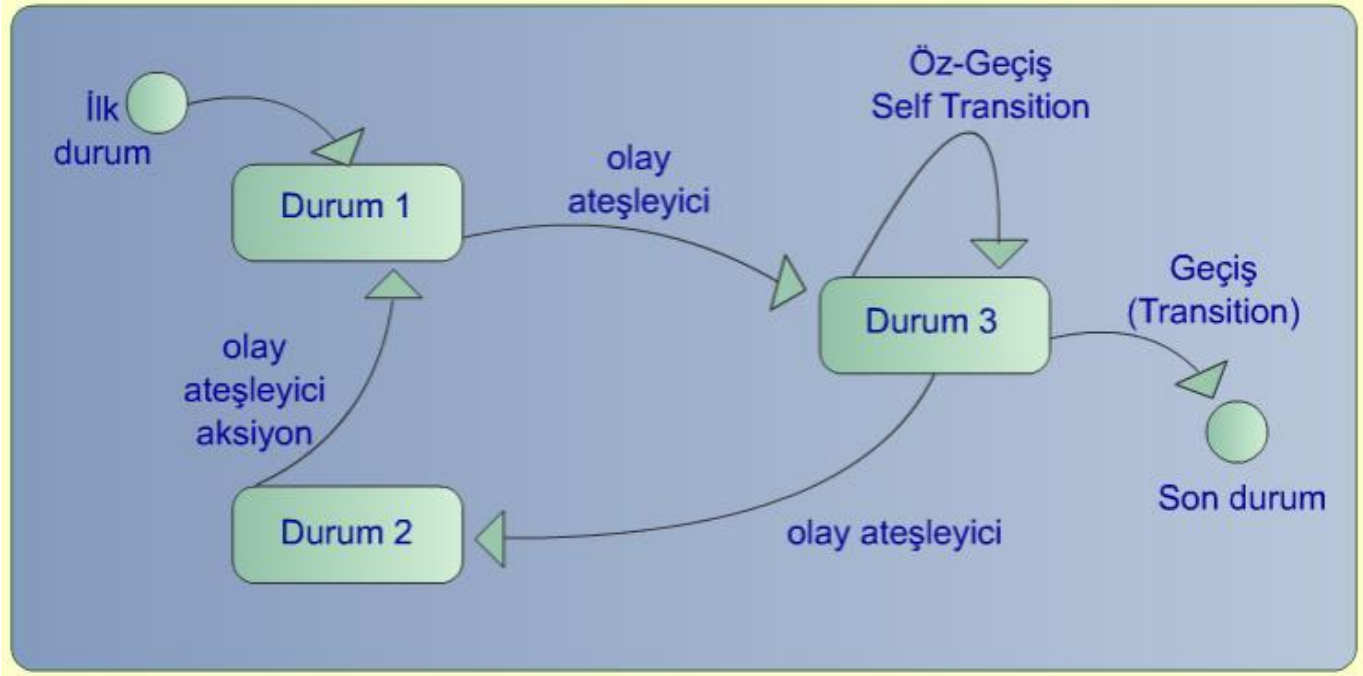
Durum Makinası (State Machine)
Durum (State)
Olay (Event)
Aksiyon (Action)
Geçiş (Transition)
Öz-Geçiş (Self Transition)
İlk Durum (Initial State)
Son Durum (Final State)

Durum Diyagramlarında Kullanılan Temel Kavramlar

State Machine (Durum Makinesi)

- "State Manchine", bir nesnenin bütün durumlarını bir şema halinde gösteren yapıdır

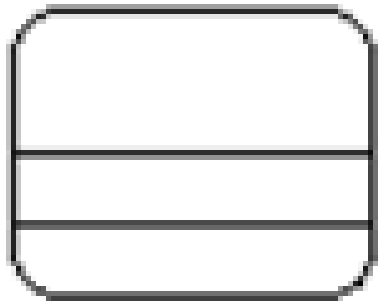




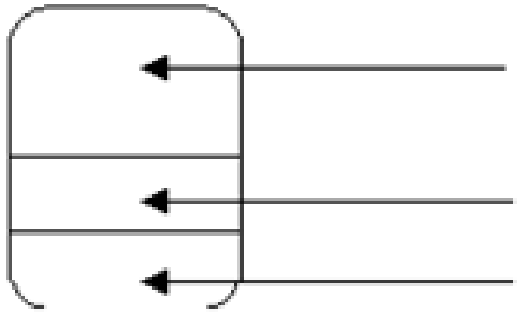
Durum Diyagramlarında Kullanılan Temel Kavramlar

State (Durum)

- Nesnenin ya da sistemin x anındaki durumunu ifade etmek için kullanılır. Köşeleri yuvarlatılmış dikdörtgenler ile gösterilir.



State gösterimi



State elemanının yapısal gösterimi

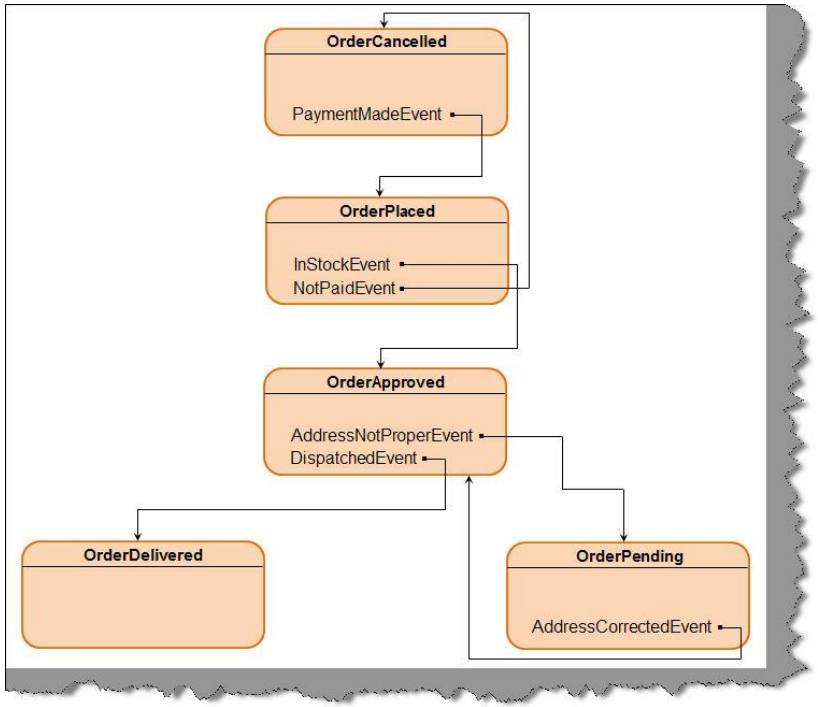
Durum Diyagramlarında Kullanılan Temel Kavramlar

Event (Olay)

- Nesnenin "state" ler arasındaki geçişini sağlayacak yordama event(olay) denir.

Action (Aksiyon)

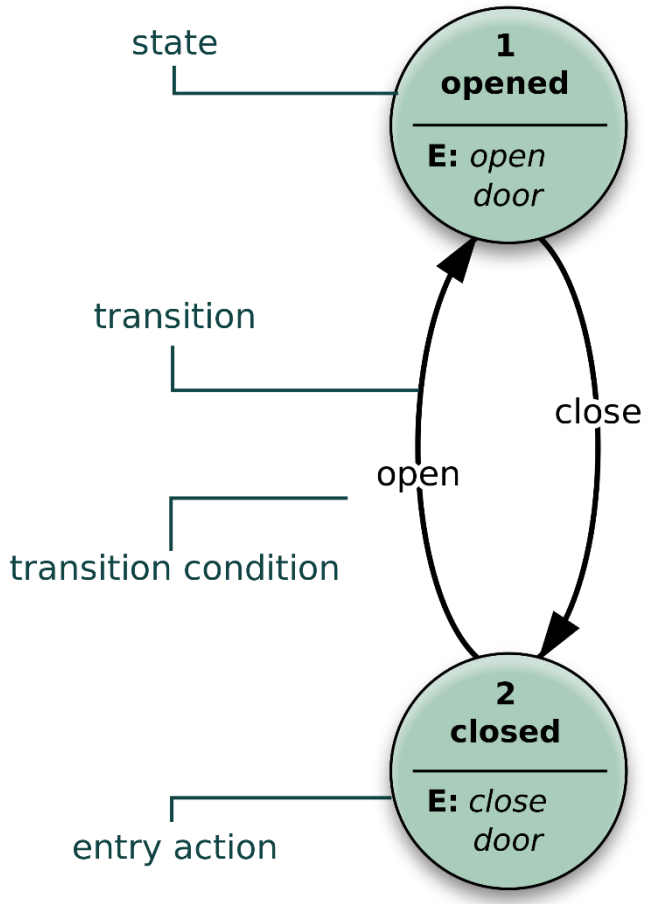
- Nesnenin bir durumdan diğer bir duruma geçtiğinde yaptığı işlere "action" denilmektedir. Action, çalıştırılabilir herhangi bir durum olabilir.



Durum Diyagramlarında Kullanılan Temel Kavramlar

Transition (Geçiş)

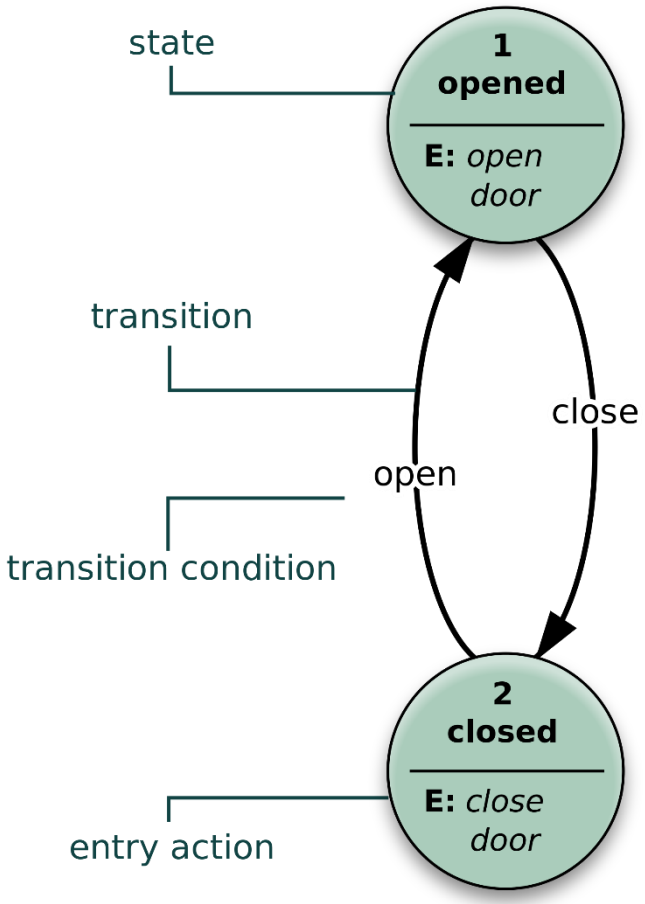
- Nesnenin bir durumdan diğer bir duruma geçişini ifade eder. Ok sembolu ile gösterilir.
- Bir nesnenin her durumu arasında bir ilişki olmayabilir. Örneğin lamba açıkken lamba kapatılamıyorsa bu iki durum arasında bir geçiş yoktur denir.
- Bir transiton(geçiş)'da 4 yapı vardır. Bu yapılardan ikisi doğal olarak "hedef(target)" ve "kaynak(source)" durumudur.
- Geçişler kaynak durumdan hedef duruma doğru yapılır. Üçüncü yapı Kaynak durumdan hedef duruma geçişi sağlayan "event trigger(olay ateşleyicisi)" dır. Son yapı ise nesnenin durum geçişi sonrasında ne şekilde davranacağını belirleyen "action" dır.



Durum Diyagramlarında Kullanılan Temel Kavramlar

Transition (Geçiş)

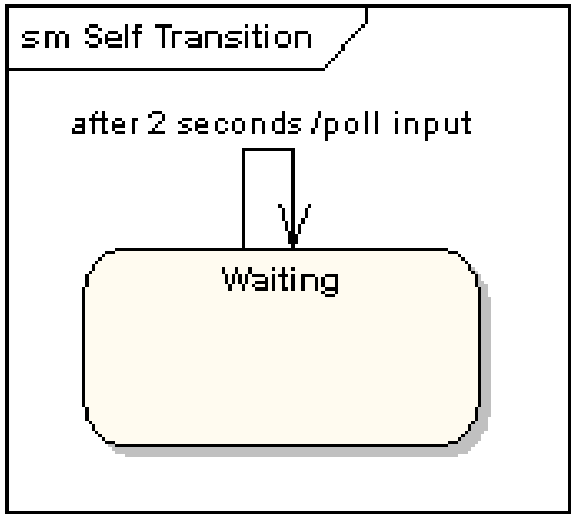
- Bir eylemin (Action) bitimiyle beraber o transition a "triggerless" transition denir.
- Eğer bir olay, bazı olay veya eylem tamamlanmasından sonra meydana gelirse, eylem veya olay “guard condition (Nöbet Durumu)” olarak adlandırılır.
- Çoğu zaman iki durum arasında geçiş sağlayabilmek için, bir event (olay) gerçekleştirilmiş olmasının ötesinde, tanımlı bir koşulun sağlanmış olması da gerekir.
- Guard condition (Nöbet Durumu) tamamlandıktan sonra geçiş(Transition) oluşur. Bu nöbet durum (Guard Condition) / olay(Event) / eylem (Action) köşeli parantez ile gösterilir.
- Diyagramlarda koşullar baklava sembolü ile gösterilirler.



Durum Diyagramlarında Kullanılan Temel Kavramlar

Self Transition (Öz Geçiş)

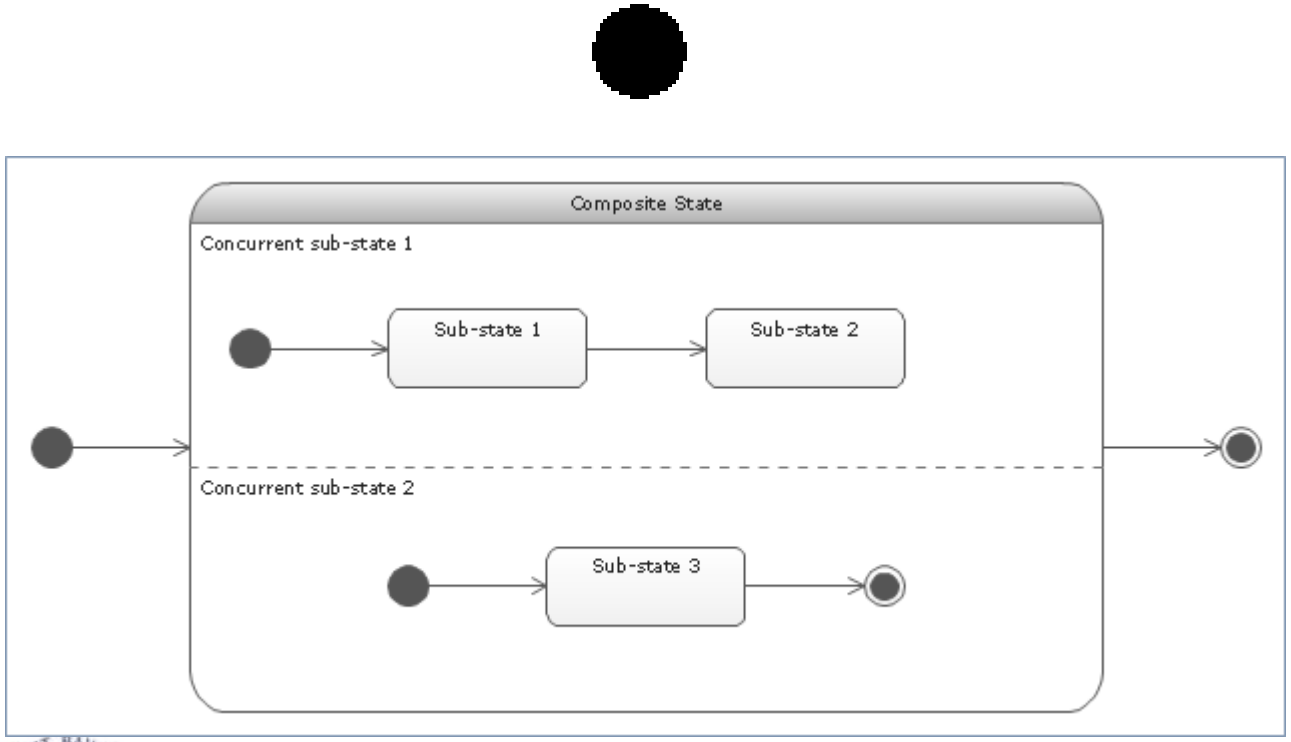
- Bazen bir nesnenin durum geçişleri farklı iki durum arasında olmayabilir.
- Mesela lamba açık durumda olduğu halde lambayı tekrar açmaya çalışmak "self transition" kavramına bir örnektir.
- Kısaca hedef durum ve kaynak durumun aynı olduğu durumlar "Self Transition" olarak adlandırılır.
- Daha net söylersek; geçişin kaynak ve hedef durumu aynı olur. Bu durum özgeçiş olarak adlandırılır.



Durum Diyagramlarında Kullanılan Temel Kavramlar

Initial State (İlk Durum))

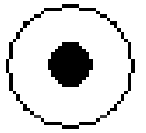
- Yaşam döngüsünün ilk eylemi ya da başlama noktasını ifade eden elemandır.
- İçi dolu yuvarlak ile gösterilir.
- Sözde durum (pseudo state) olarak da adlandırılır. Sözde durum denilmesinin sebebi değişkeni veya herhangi bir eyleminin olmayışıdır.

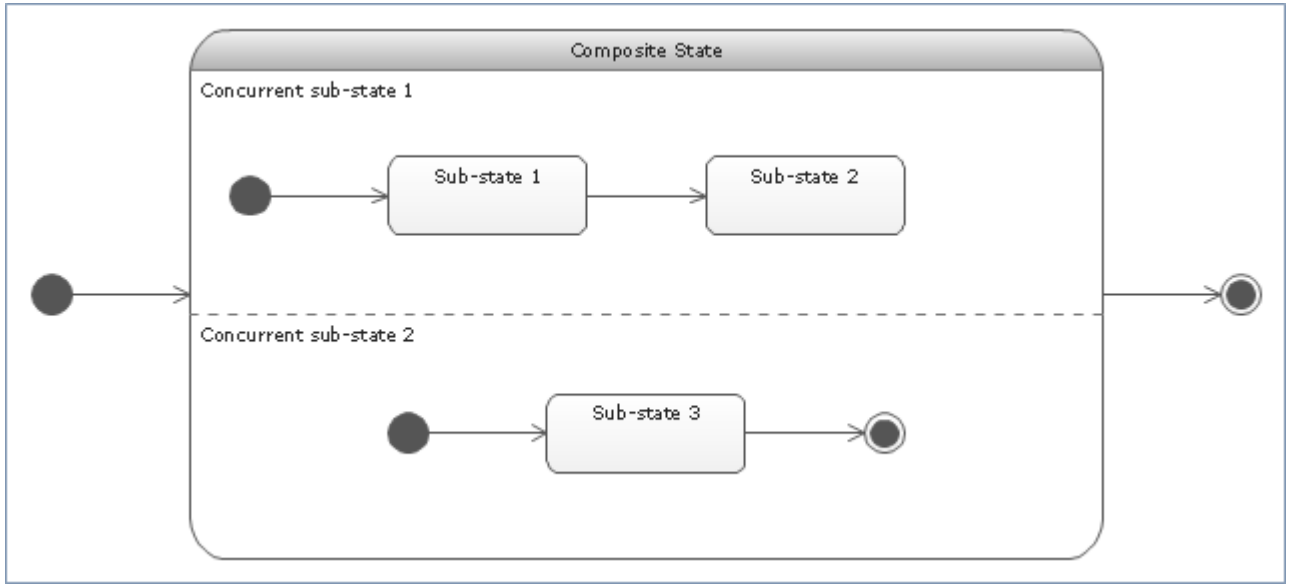


Durum Diyagramlarında Kullanılan Temel Kavramlar

Final State (Son Durum)

- Sistemin yaşam döngüsü içerisindeki son durumunu ifade eder.
- Bir nesne birden fazla "Final State" durumuna sahip olabilir..

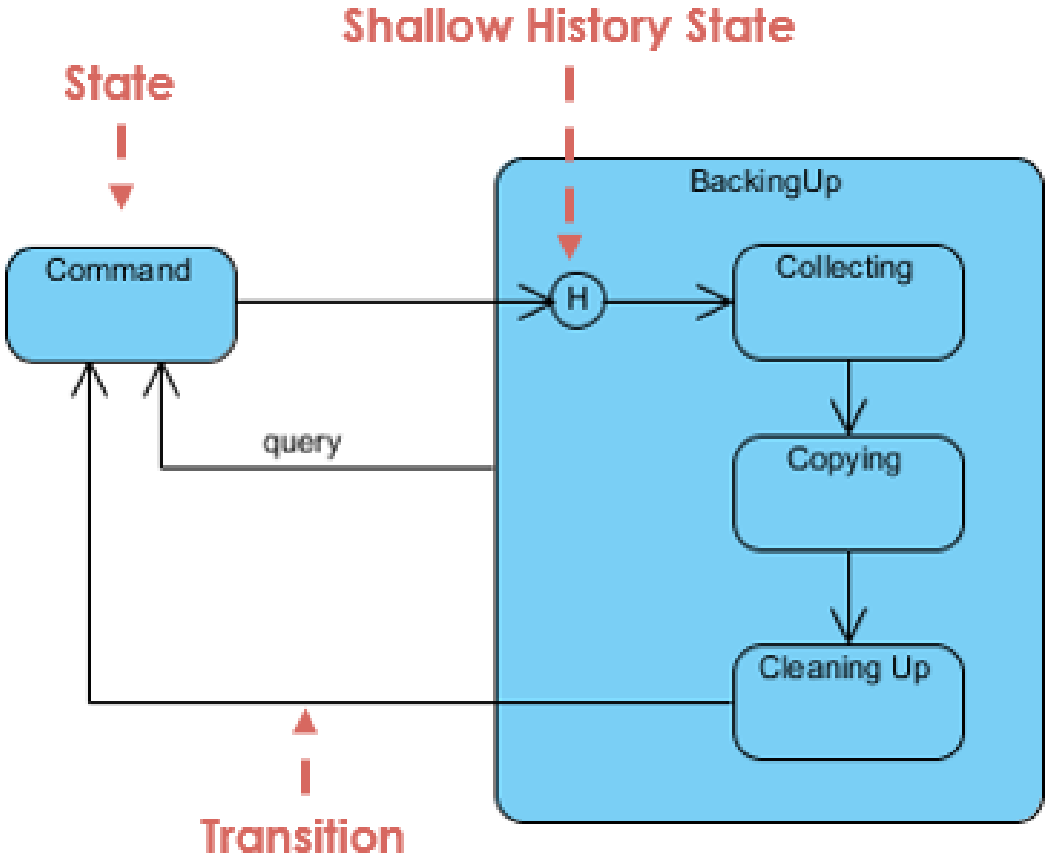




Durum Diyagramlarında Kullanılan Temel Kavramlar

History States (Geçmiş Durumlar)

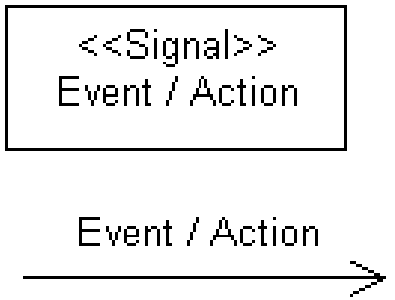
- Bir akışta, bir nesne, transa ya da bekleme durumuna geçebilir. Belirli bir olayın gerçekleşme süresinde bekleme durumuna girdiği zaman son aktif durumuna geri dönmek istenebilir.
- Daire içerisinde H harfiyle gösterilir.



Durum Diyagramlarında Kullanılan Temel Kavramlar

Signal (Sinyal)

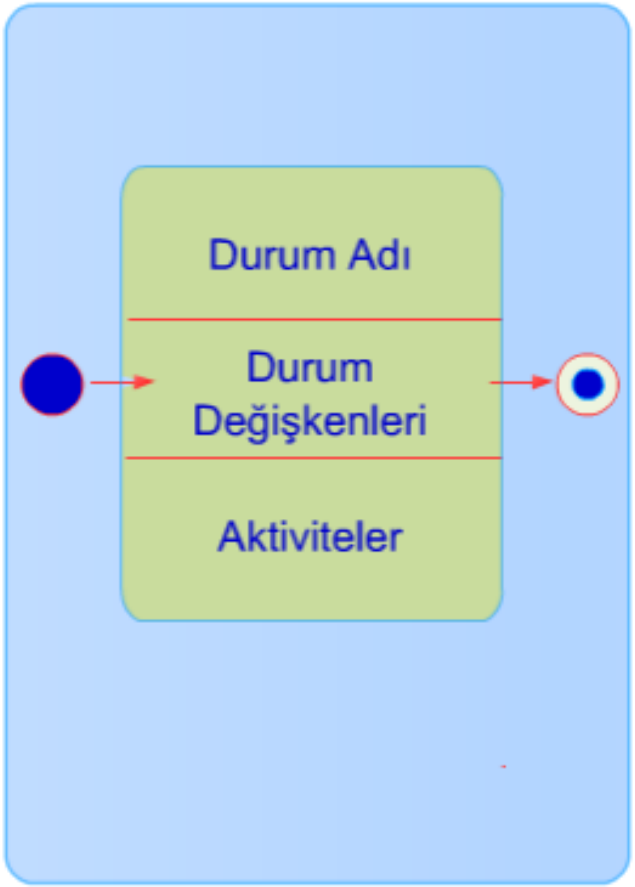
- Duruma bir mesaj yada tetikleyici gönderimi olduğunda geçiş oluşur ve mesaj event (olay) ile gönderildiğinde sinyal olarak adlandırılır. << Sinyal>> şeklinde gösterilir.



Nesne Durumları ve UML

State Diyagramlarının tek bir nesne ile ilişkili olmasıdır.

- Şimdi de nesnelerin durumlarının UML ile nasıl gösterildiğine bakalım.
- Bir Durum yandaki şekilde olduğu gibi köşeleri yuvarlatılmış dörtgen şeklinde gösterilir.
- Bir Durumun 3 ana bileşeni vardır. Birinci bileşen Durum'un adıdır. İkinci bileşen Durum Değişkenleri, üçüncü bileşen ise durum ile ilgili çeşitli aktiviteleri belirtmek için kullanılan yapıdır.
- Bu aktivitilerin bazılarının UML'de standart olarak belirtilmiş olmasına rağmen, istersek kendimiz de aktivite tanımlayabiliriz.

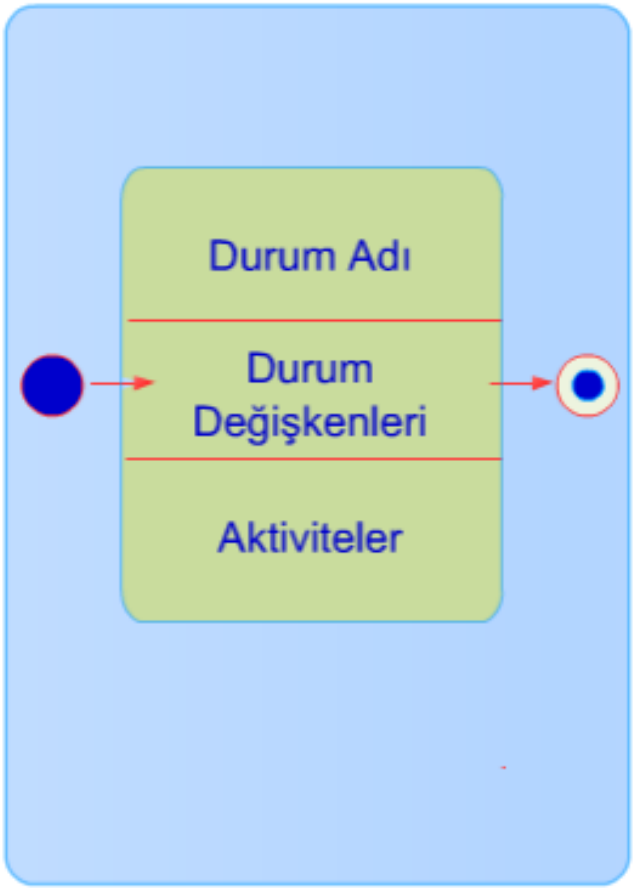


Nesne Durumları ve UML

- Standart olan aktivitelerin sayısı 3'tür. Bunlar:
  - Entry
  - Exit
  - Do

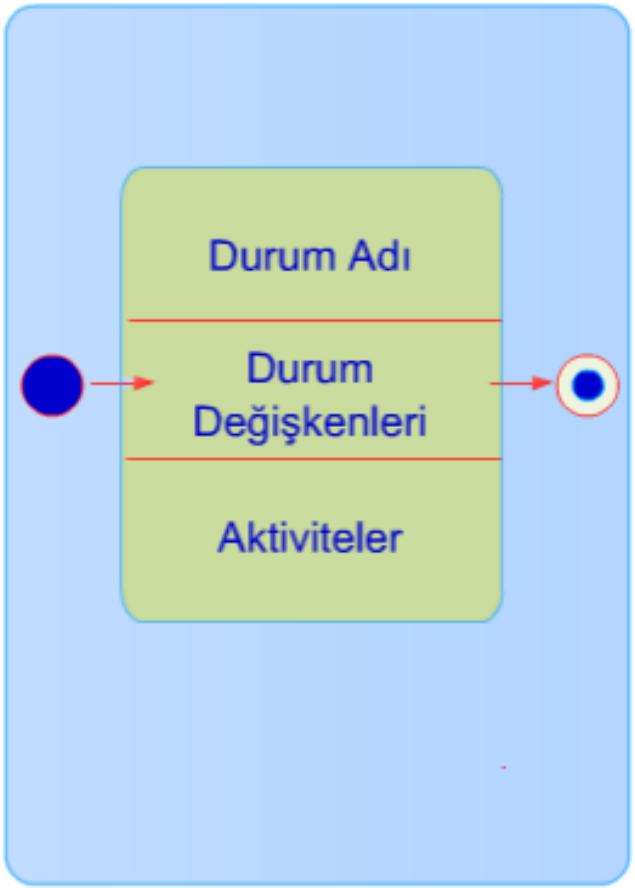
isimli aktivitelerdir.

- *Entry* aktivitesi, nesnenin ilgili duruma geçtiği andaki davranışın adını belirtir. *Exit* aktivitesi, nesne ilgili durumdan diğer bir duruma geçişi sırasındaki davranışını belirler.
- Örneğin, bir lamba açık durumdan kapalı duruma geçtiğinde lambadaki elektrik devresinden akımın kesilmesi, ya da lambanın kapalı durumdan açık duruma geçişi sırasında devreye akımın verilmesi bu tür aktivitelere örnek verilebilir.



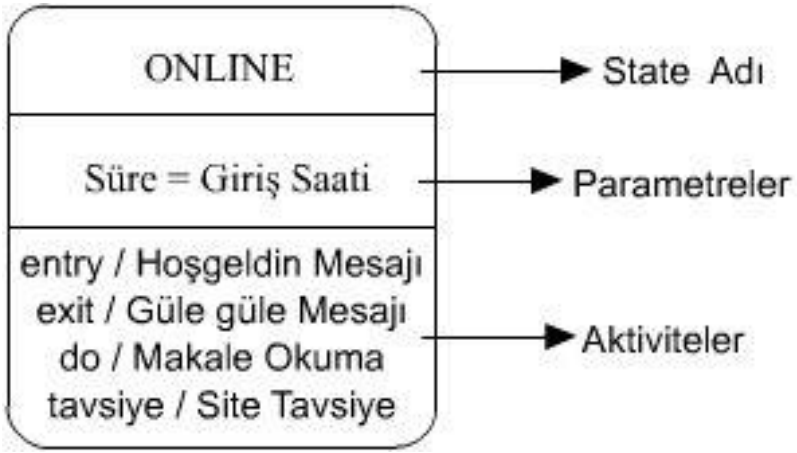
Nesne Durumları ve UML

- Diğer bir standart aktivite olan *Do* ise nesnenin belirtilen durumda olduğu sürece nasıl bir davranış sergileyeceğini belirtir. Lambanın yanması sırasında devreden geçen akımın sabitliğinin korunması *Do* aktivitesine örnek olarak verilebilir.
- Bir durumun bütün standart aktivitelerini tanımlamak zorunda değiliz. Söz gelimi bir durumun yalnızca *Do* aktivitesi olabilir.



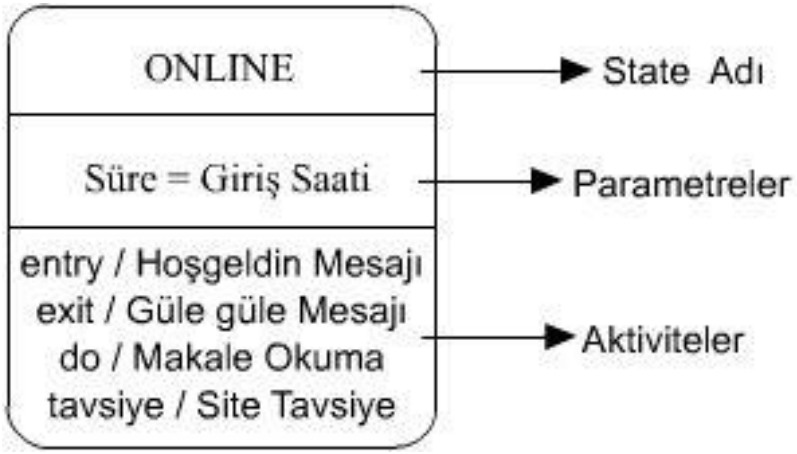
Nesne Durumları ve UML

- Yukarıdaki "State" tanımlamalarını göz önünde bulundurursak UML'de en temel durum modeli aşağıdaki gibi yapılabilir.
- Yukarıdaki şekil teknik makale içeriği sunan bir siteye ait kullanıcının ONLINE olma durumunu modellemektedir. Şekildeki en üstteki kısım "State Name" yani durum adını belirtir. İkinci kısım ilgili duruma ait çeşitli parametreleri gösterir. Bu örnekte ONLINE durumunda olan kullanıcının sisteme ne zaman giriş yaptığı belirtilmektedir.



Nesne Durumları ve UML

- Parametreler bölümüne ihtiyaç olduğunda çeşitli ifadelerde(**ToplamSüre = EskiSüre + YeniSüre** gibi) yazılabilir. Son ve en alttaki bölüm ise "State" ile ilgili aktiviteleri göstermektedir. entry aktivitesinde kullanıcının "Hoş geldin" mesajı ile karşılanması, exit aktivitesinde "Güle güle" mesajı ile uğurlanması, do aktivitesinde ise kullanıcıya makale okuma iznin verileceği belirtilmektedir. Bu standart aktivitelerin dışında olan**tavsiye** isimli diğer bir aktivitede ise kullanıcıya diğer teknik makale içeren sitelerin tavsiye edileceği belirtilmektedir.

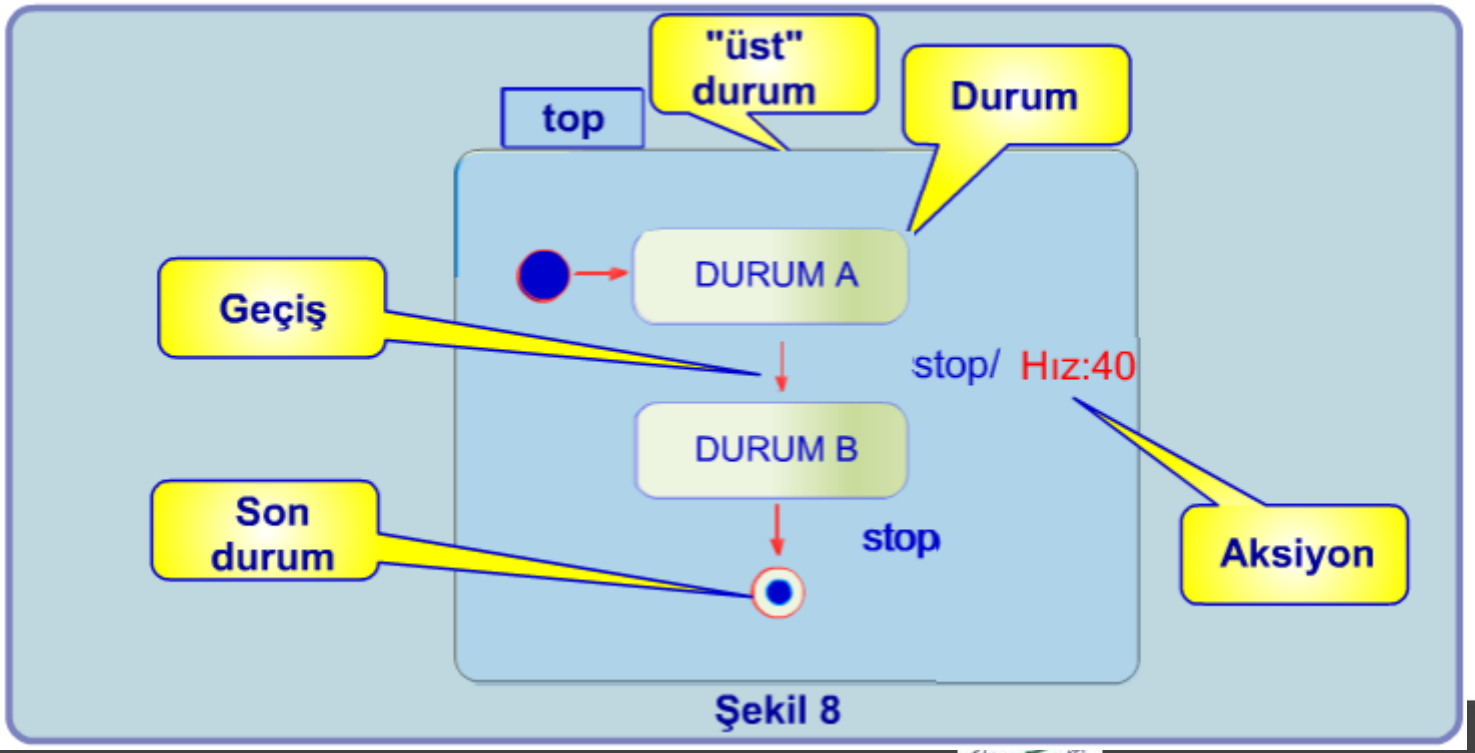


Nesne Durumları ve UML

- Yandaki şekil her kullanıcının sisteme girişte kullandığı Şifre yazımını modellemektedir.
- Şekildeki en üstteki kısım durum adını belirtir. İkinci kısım ilgili duruma ait çeşitli parametreleri gösterir. Burada KarakterSayısı parametresi şifrenin kaç karakter uzunluğunda olduğu bilgisini tutmaktadır. Son ve en alttaki bölüm ise durum ile ilgili aktiviteleri göstermektedir.
- ***Entry*** aktivitesinde kullanıcının girdiği şifrenin başkaları tarafından görülmesini engellemek için görünüm modu olan echo off durumuna getirilerek karakterler "*" karakteri ile gösterilir.
- ***Exit*** aktivitesinde görünüm modu echo on durumuna getirilerek karakterler tekrar görünür hale getirilir.
- ***Help*** aktivitesinde ise kullanıcaya şifreyi girerken uyacağı, şifrenin en fazla kaç karakter uzunluğunda olacağı, şifre içinde büyük harf ve boşluk kullanılmaması gibi kurallar anlatılır.

Durum Detayları ve Geçişlerin Gösterimi

- Durum diyagramlarında bir durumdan diğerine geçişe sebep olan olaylar (*tetikleyici olay - trigger event*), ve durumda değişikliği gerektiren hesaplamalar (*aksiyon-action*) geçiş çizgisinin yanına, tetikleyici olayı aksiyondan "/" karakteri ile ayırarak yazılır.



Şekil 8

- Durum A'dan Durum B'ye geçmek için ya stop düğmesine basılmalı ya da Hız 40 olmalıdır. Burada stop düğmesine basmak tetikleyici olay, Hız'ın 40 olması ise aksiyondur. Bu ifadeler geçiş okunun hemen yanına yazılır.

Geçişlerin Gösterimi

- Bazen bir olay, bir durumdan diğerine herhangi bir aksiyon olmadan geçişe sebep olabilir ya da geçiş bir aksiyon tamamlandıktan sonra herhangi bir olay olmadan gerçeklenebilir. Bu tip geçişlere Tetikleyicisiz Geçiş (Triggerless Transition) denir.
- Mesala ekran koruyucunun n dakikalık durağan halinden sonra aktif hale geçmesi (Şekil 9a), ya da telefonun tuşlarına basıldıktan sonra aranan hat boş ise, aranan telefonun çalması veya aranan hat dolu ise meşgul durumuna düşmesi (Şekil 9b) Koşullu Geçiş'e örnek verilebilir.
- Şekilde de görüldüğü gibi koşullar geçiş okunun üzerinde köşeli parentez içine yazılır.

Geçişlerin Gösterimi

- Ayrıca Koşullu **Geçişler de (Guard Condition)** mümkündür. Koşul sağlandığında geçiş otomatik olarak gerçekleşir.

Şekil 9. Koşullu geçiş örneği

Geçişlerin Gösterimi

Durumlar Arası Koşullu Geçişler (Guard Condition)

- Bir nesnenin bir durumdan diğerine geçişi için sadece olay tetiklenmesi yetmeyebilir. Mesela gün ışığına göre yanıp sönen aynı zamanda açma kapama düğmeside olan bir lambayı düşünün. Gündüz saatlerinde lamba kapalı durumdayken anahtar çevrilerek lambanın açılması istenmeyebilir.
- Yani lambanın açılması ortamın karanlık olmasınada bağlıdır. Sadece lambanın anahtarını çevirerek lambanın açılamayacağı koşullu geçişlere örnek olabilir. Lambanın açılması için "anahtar açma" olayının meydana gelmesi ve ortamın aydınlık seviyesinin lambanın açılması için yeterli olması gerekmektedir.
- Bu durumu geçişler üzerine açılan ve kapanan köşeli parantezler içine koşulu yazarak belirtebiliriz. Örneğin KAPALI ve AÇIK durumları arasındaki geçişe aşağıdaki gibi bir koşul yazılabilir.

[Aydınlık == Gunduz Seviyesi]
KAPALI ----------> AÇIK
Anahtarı Çevirme

Geçişlerin Gösterimi

Durumlar Arası Koşullu Geçişler (Guard Condition)

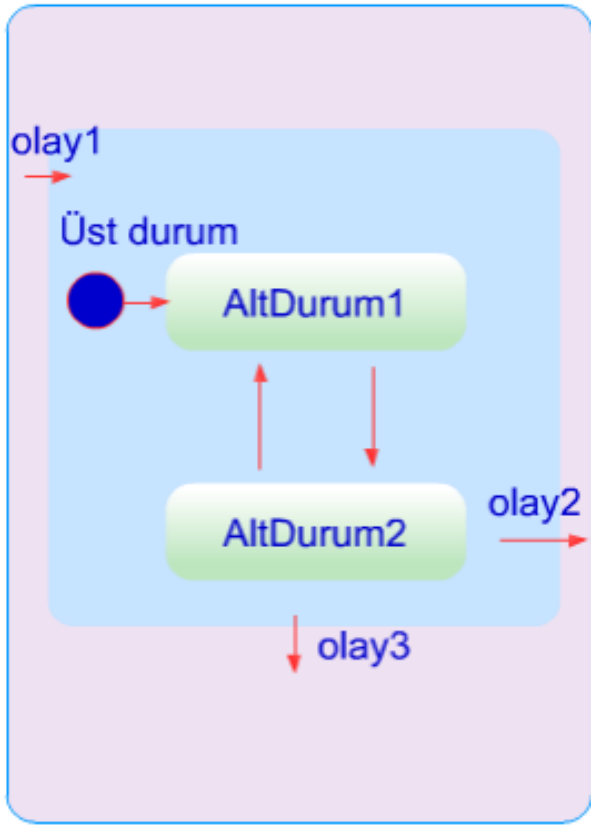
- Eğer geçişler arasında sadece koşul ifadesi yazılırsa, koşul ifadesi gerçekleştiği anda geçişin sağlanması beklenir. Yani olayın ateşlenmesine gerek yoktur. Yukurıdaki gösterimde ise lambanın AÇIK duruma geçebilmesi için hem koşulun sağlanması hemde olayın(anahtarı çevirme) gerçekleşmesi gerekir.

[Aydınlık == Gunduz Seviyesi]
KAPALI ----------> AÇIK
Anahtarı Çevirme

Geçişlerin Gösterimi

Alt Durumlar

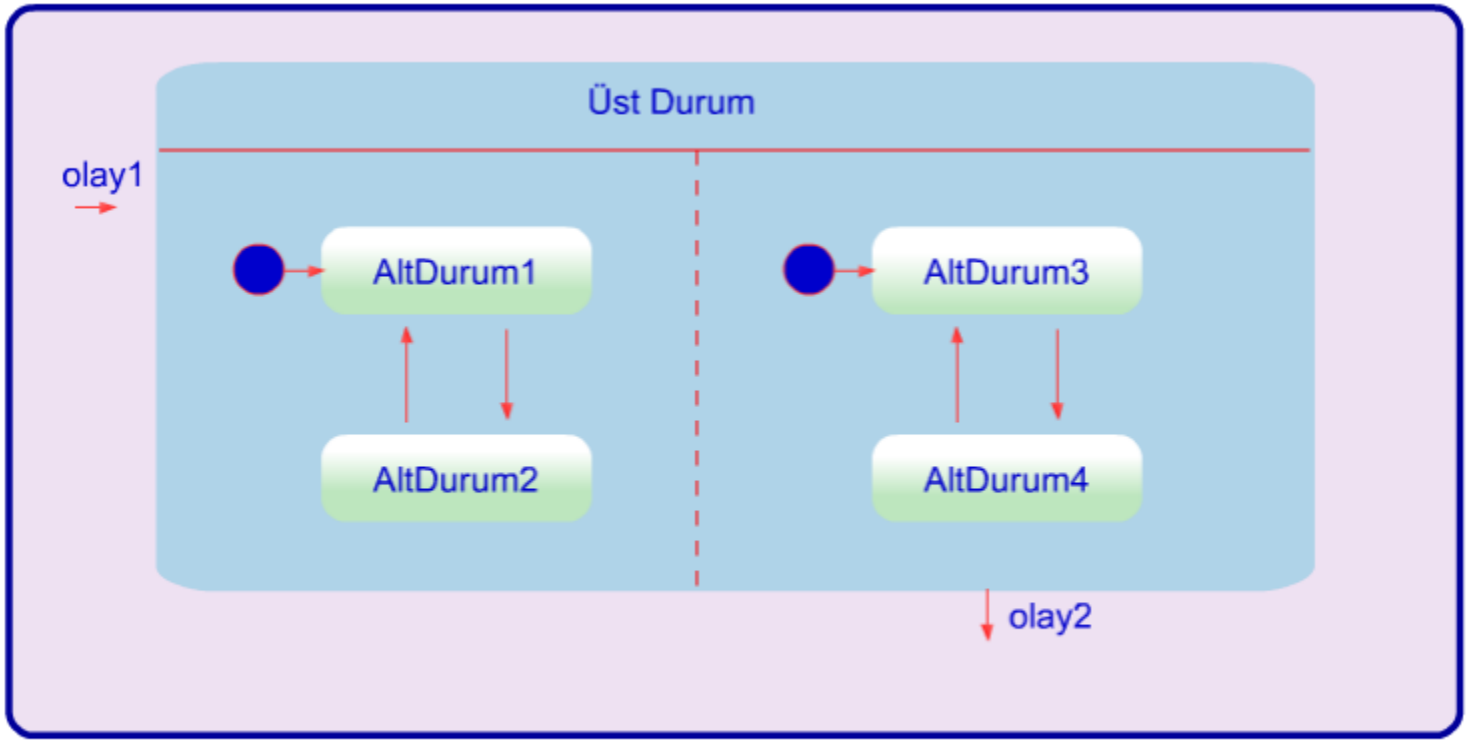
- Gerçek hayattaki sistemler çoğu zaman daha karmaşıktırlar. Hatta birçok sistemin Durum Modeli çıkarıldığında, bazı durumların başka durumları içerdiği görülmektedir. Bu türden durumlara *Altdurum* (*Substates*) denmektedir.
- *Altdurumlar,* ardışıl (sequential) ya da eş zamanlı (concurrent) gerçekleşebilirler.
- Şekil 10'da görüldüğü gibi, *Ardışıl Altdurumlar* (*Sequential substates*) birbirlerinin peşi sıra gelirler. Şekil 11'de görülen *Eş zamanlı Altdurumlar* (*Concurrent substates*) ise bir veya daha fazla ardışıl durumun eş zamanlı gerçekleşmesiyle oluşurlar. Altdurumlar'ın arasında kesikli çizgi kullanılarak Altdurumlar'ın eş zamanlı gerçekleştiği belirtilir.



Şekil 10. Ardışıl Altdurum

Geçişlerin Gösterimi

Alt Durumlar

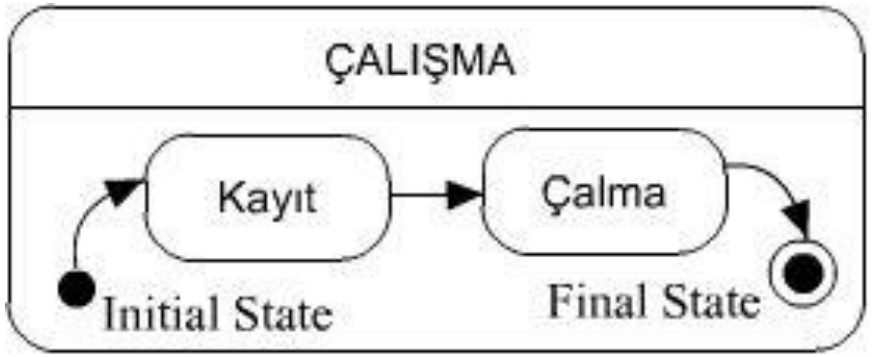


Şekil 11. Eş zamanlı Altdurumlar

Geçişlerin Gösterimi

Alt Durumlar

"Alt durumları" içeren modellemeler UML ile yapılabilmektedir. Alt durumlara bir örnek verelim : Bir CD çalar programını düşünün. CD çalar programı belirli bir anda ya çalışıyor durumdadır yada çalışmıyor durumundadır. Eğer CD çalar programı çalışır durumda ise yine iki durum olabilir. CD çalar programı ya bir kayıt yapıyordur, yada seçilen şarkıyı çalıyordur. Çalışma durumu içindeki bu iki durumu aşağıdaki gibi modelleyebiliriz.



Alt-durumlar ile ilgili önemli durumlar :

1-a) Bir alt-durum, varolan başka bir durum içinde olmalıdır.

1-b) Alt-durumu olan bir duruma "Composite State" denir.

1-c) Bir durum içinde birden fazla seviyede durum olabilir. Örneğin yukarıdaki örnekte Kayıt durumu içinde de alt durumlar olabilir.

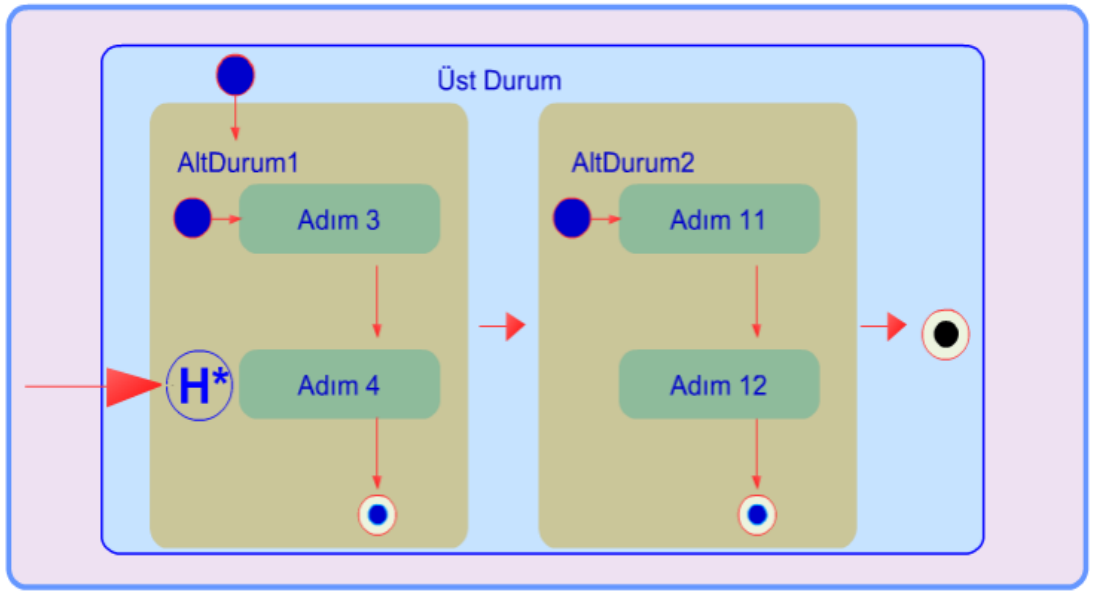
1-d) Herhangi bir alt-durum içermeyen durumlara "Simple State" yani basit durum denir.

Geçişlerin Gösterimi

Basit ve Bileşik Durumlar

Herhangi bir Altdurum'u olmayan durumlara *Basit Durum* (*Simple State*), Altdurum'u olan bir duruma ise *Bileşik Durum* (*Composite State*) denir. Bir Altdurum, varolan başka bir durum içinde olmalıdır ve bir durum içerisinde birden fazla seviyede durum olabilir.

Ayrıca UML bir Birleşik Durum'dan bir başka duruma geçiş sırasında aktif olan Altsistemi hatırlamak/belirtmek için (H) sembolünü kullanır. (H) sembolü kesintisiz düz bir ok işareti ile hatırlanan alt sisteme bağlanır ve ok işareti alt sistemi gösterir.



Şekil 12. Bir önceki ziyaret edilen hiyerarşik duruma dönüş

Geçişlerin Gösterimi

Kompozite(Composite) Durumlar ilgili Önemli Özellikler

2-a) Bir composite duruma geçiş(transition) olabileceği gibi, composite durumdan başka durumlarada geçiş olabilir. Eğer bir geçiş composite duruma doğru ise bu composite durumun "Başlangıç Durumu(Initial State)" içermesi zorunludur. Eğer böyle olmasaydı tanımsız durumlar ortaya çıkardı. Bu durumu aşağıdaki composite durum diyagramından rahatlıkla görebilirsiniz.

2-b) Eğer geçiş(transition) bir alt-duruma doğru ise, ve dıştaki duruma ilişkin her hangi bir entry aktivitesi varsa çalıştırılır ve ardından alt-duruma ilişkin aktiviteler çalıştırılır.

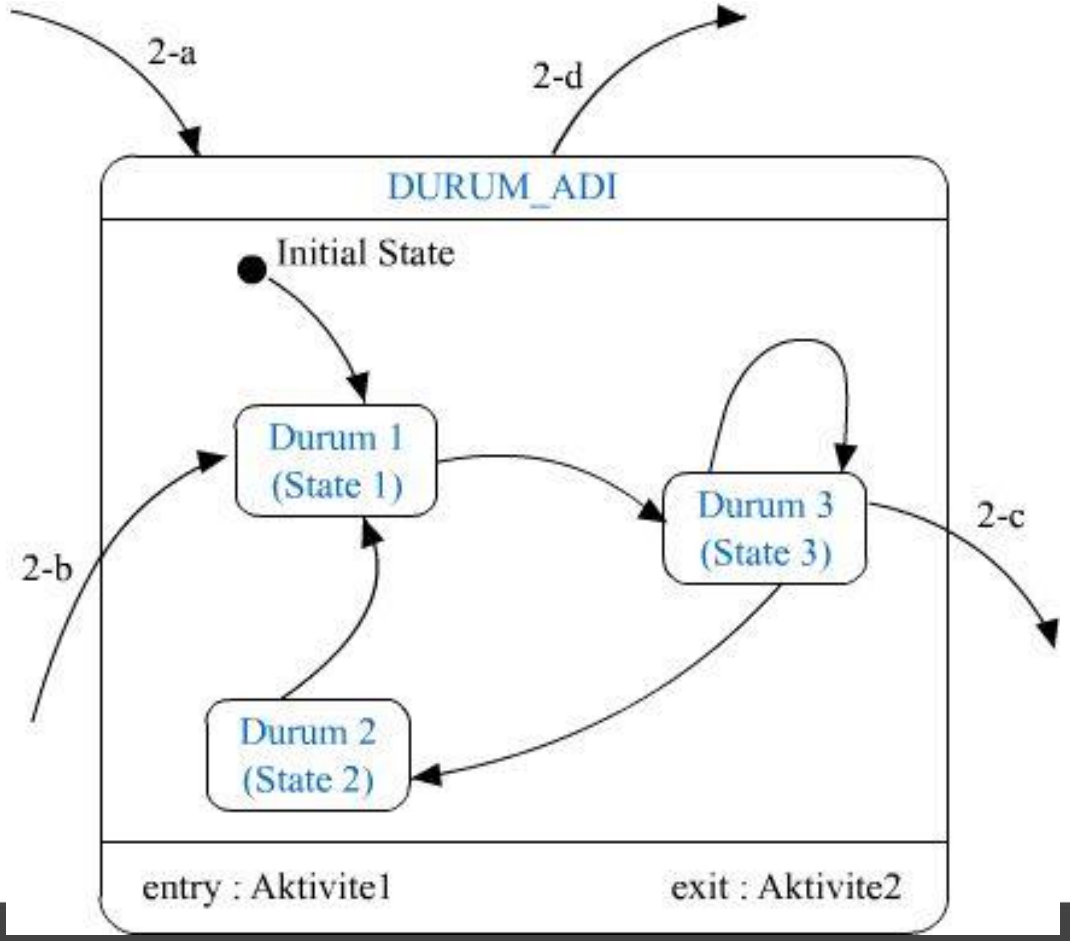
2-c) Eğer geçiş(transition), composite durum içindeki bir alt-durumdan, composite durumun dışındaki bir duruma doğru ise, önce composite durumun exit aktivitesi ardından alt-durumun exit aktivitesi çalıştırılır.

2-d) Eğer geçiş(transition) composite durumdan, dışarıdaki farklı bir duruma doğru ise öncelik sırası **diğer alt-durumlara göre** bu geçiştedir.

Geçişlerin Gösterimi

Kompozite(Composite) Durumlar ilgili Önemli Özellikler

Aşağıda kompleks bir durum modeli verilmiştir. Bu modelde 2-a, 2-b, 2-c ve 2-d maddelerinde verilen geçişler işaretlenmiştir.

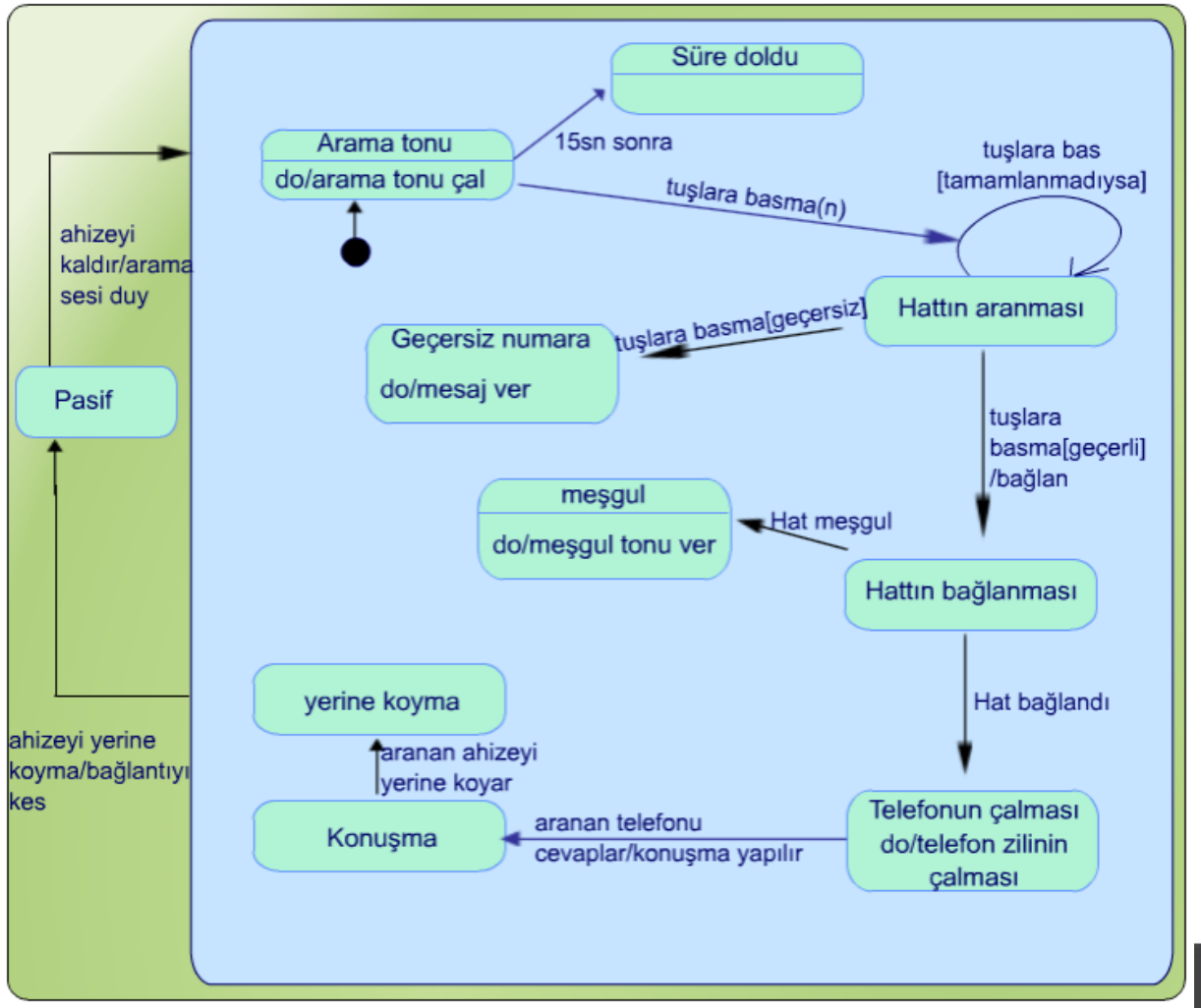


ÖRNEK

Telefonun Modellenmesi

Bu örnekte telefona ait durumlar modellenmiştir.

Telefon temelde iki duruma sahiptir; Aktif ve Pasif. Ahize kaldırılarak pasif durumdan aktif duruma, yerine konularak aktif durumdan pasif duruma geçilir.



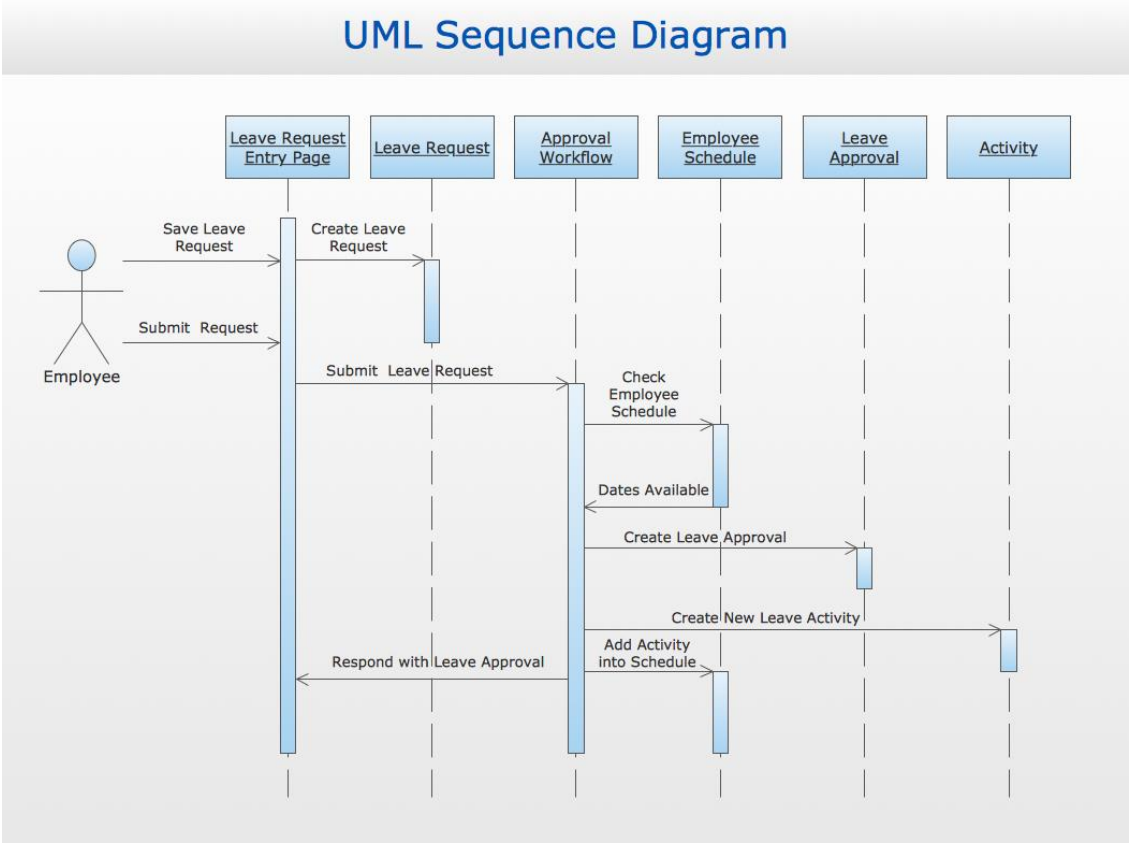
UML DİYAGRAMLARI

ARDIŞIL DİYAGRAMLAR

Ardışıl Diyagramlar (Sequence Diagrams)

- Ardışıl Diyagramlar, zaman içerisinde nesnelerin birbirleriyle nasıl bir iletişim içinde olduklarını gösterir. Böylece genişletilen görüş alanına, önemli bir boyut da dahil edilir: Zaman. Buradaki anahtar fikir, nesneler arasındaki etkileşimin belirli bir ardışıllık içinde gerçekleşmesi ve bu ardışıllığın başlangıçtan sona doğru giderken belirli bir süre tutmasıdır.

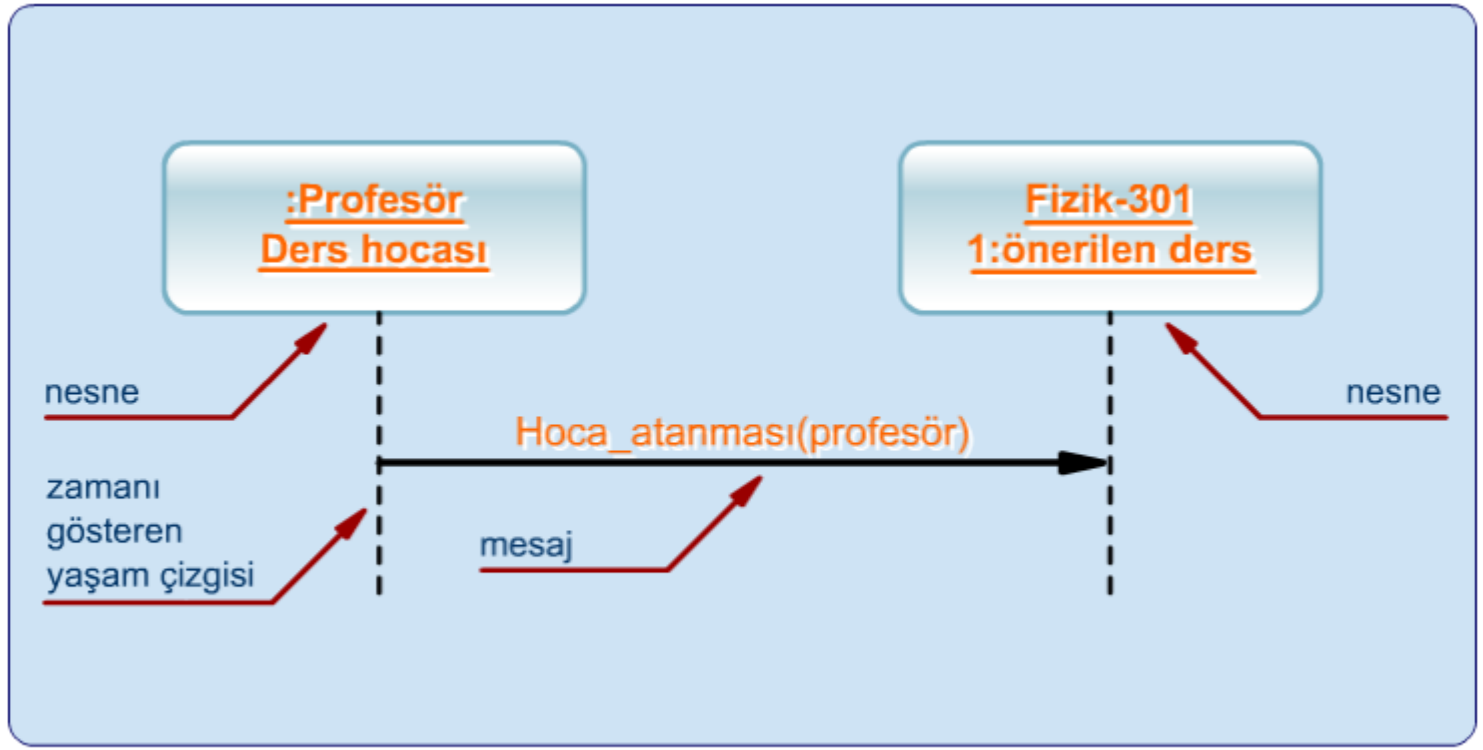
- Ardışıl (Sequence) diyagramlar, sistemdeki nesneler ya da bileşenler arasındaki mesaj akışının olaylarını, hareketlerinin ardaşık şekilde modellenmesinde kullanılan diyagramlar sequence diyagramlardır.



Ardışıl Diyagramlar (Sequence Diagrams)

- Ardışıl Diyagramlar nesneler, mesajlar ve zaman olmak üzere üç parçadan oluşurlar. ***Nesneler***, içinde ismi altı çizili olarak yazılmış dikdörtgenler ile, ***mesajlar*** kesintisiz düz ok işareti ile, ***zaman*** ise aşağı doğru dikey kesikli çizgi ile gösterilir.

- Sequence diyagramlarının akışı soldan sağa doğru olmalıdır. Sequence diyagramları oluşturmadan önce senaryo (use case) oluşturulmalıdır.



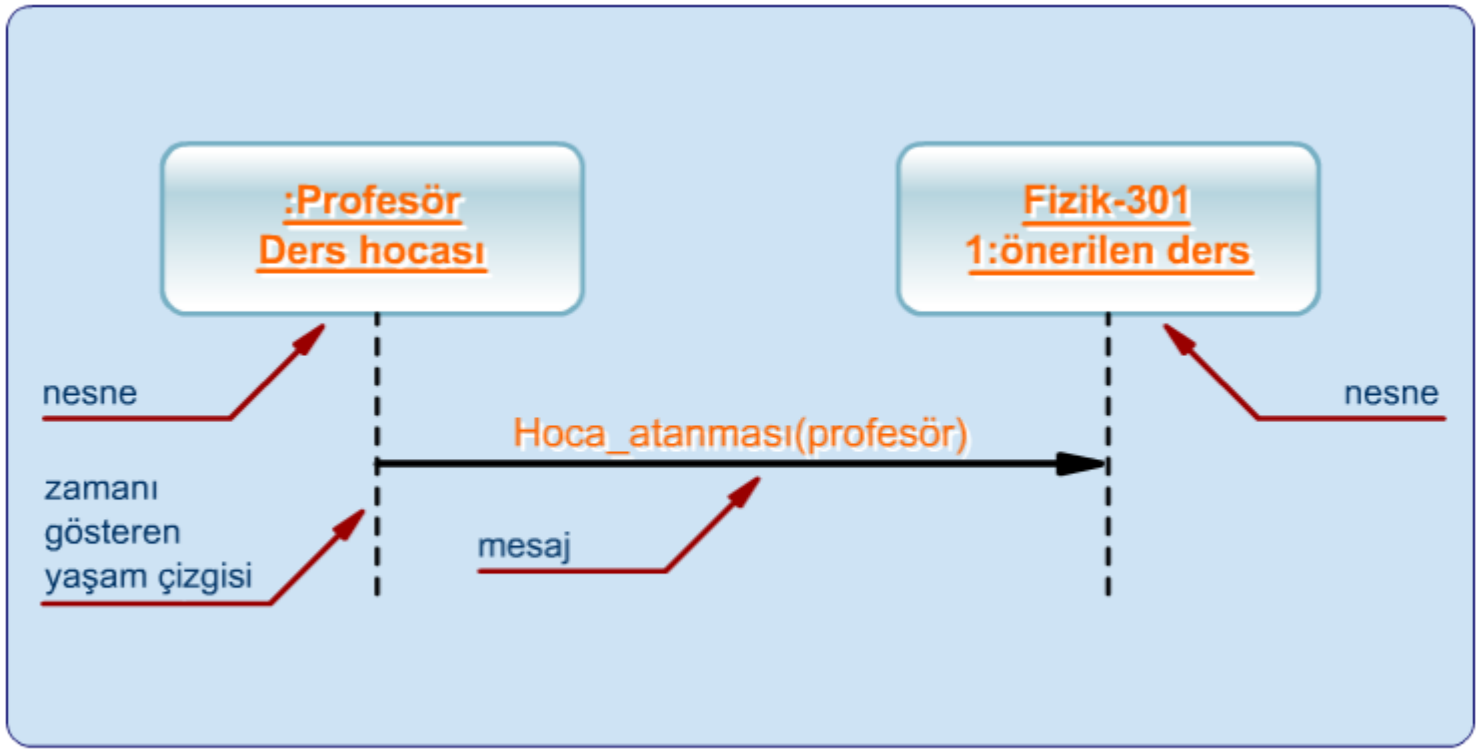
Şekil 1. Basit Ardışıl Diyagram örneği (Derse profesör atanması)

Ardışıl Diyagramlar (Sequence Diagrams)

- Bir sequence diyagramı nesnelerden, mesajlardan ve zaman çizelgesinden oluşmaktadır.

Sequence diyagrami iki boyutludur:

1. **Dikey boyut**: Mesajların/olayların sırasını oluşma zamanı sıralarına göre gösterir.
2. **Yatay boyut:** Mesajın gönderildiği nesne örneklerini (object instances) gösterir.

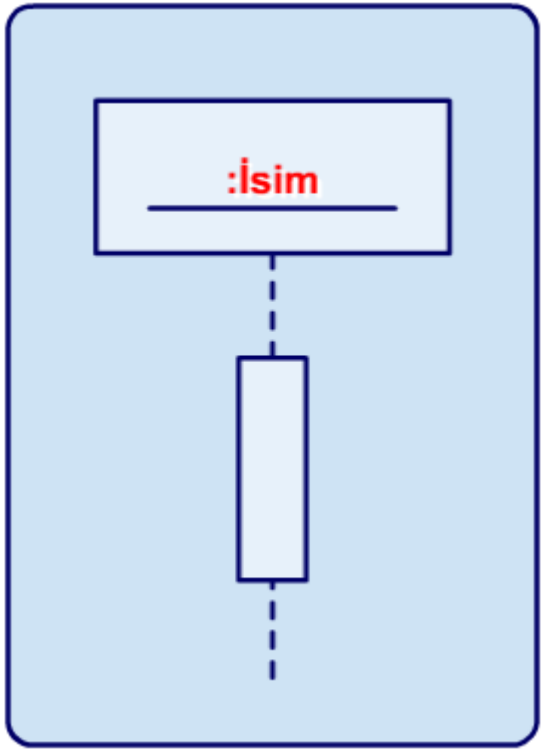


Şekil 1. Basit Ardışıl Diyagram örneği (Derse profesör atanması)

Ardışıl Diyagramlar (Sequence Diagrams)

Nesneler (Objects)

- Nesneler diyagramın en üstünde soldan sağa doğru dizilirler.
- Nesneler, diyagramı basitleştirmek için herhangi bir sırada yerleştirilebilir. Her bir nesne yaşam çizgisi (lifeline) denilen kesikli çizgiler ile aşağı doğru yayılır.
- Yaşam çizgileri boyunca yer alan dar dikdörtgenlere *aktivasyon (activation)* denilir. Aktivasyonlar, nesnenin taşıdığı işin gerçekleşmesini temsil eder. Dikdörtgenin genişliği aktivasyonun devam süresini belirler. Bu yandaki şekildeki gibi gösterilir.



Ardışıl Diyagramlar (Sequence Diagrams)

Mesajlar (Messages)

- Ardışıl diyagramda farklı nesnelerin birbirleri ile etkileşimi mesajlar ile gösterilir.
- Mesaj gösterimi mesajların tipine göre değişir. Sequence diyagramlarda mesajlar, basit(simple) mesajlar, nesne oluşurken ya da bellekten silinirken kullanılacak özel mesajlar ve mesaj yanıtları olarak değişik şekillerde gösterilir.

Ardışıl Diyagramlar (Sequence Diagrams)

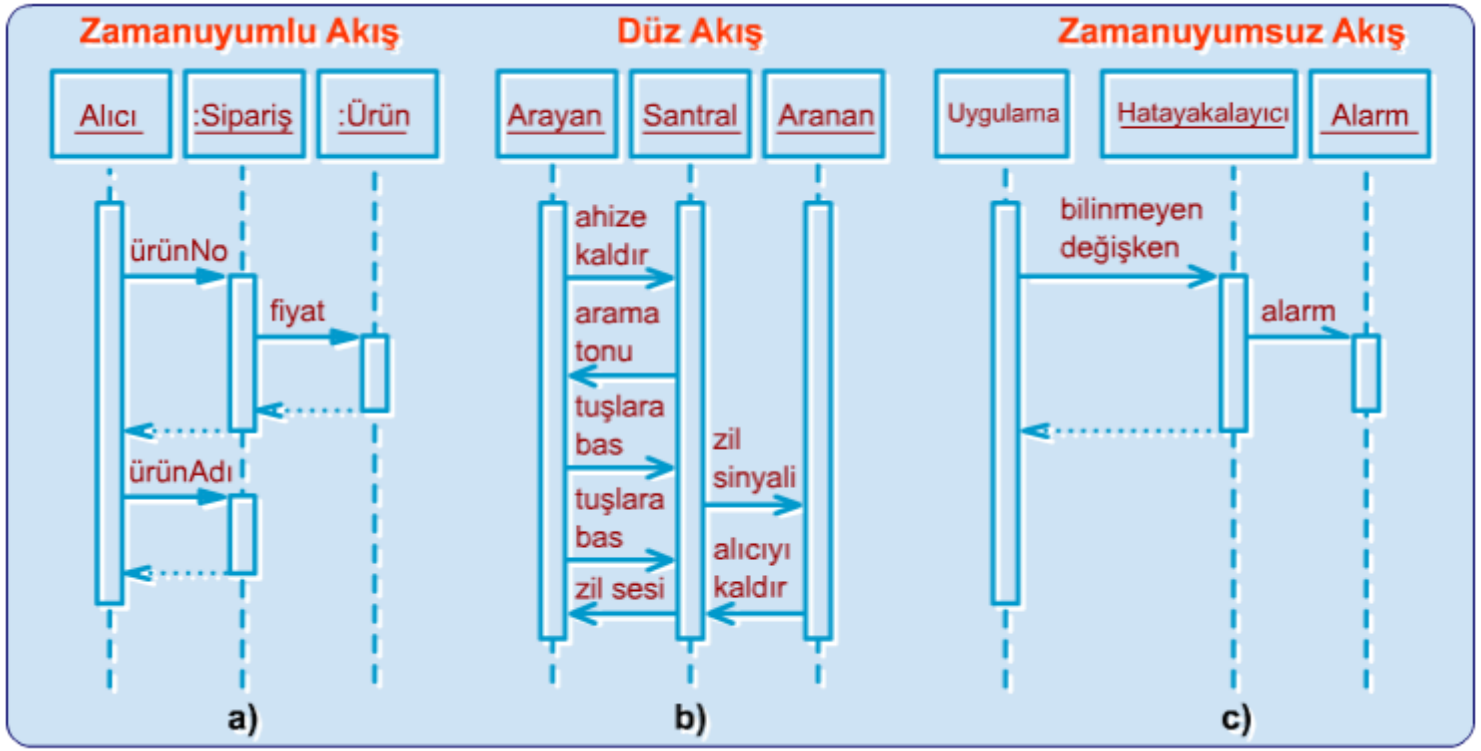
Mesajlar (Messages)

Mesaj Tipleri ve Gösterimleri	
Basit(Simple) Mesaj Tipi: Basit mesajlar nesneler arasındaki akış kontrolünün iletimini göstermek için kullanılır. Nesnelerin methodlarını doğrudan çağıramazlar. Sık kullanılan mesaj tipi değildir.	
Senkron (Syncronous)/ Çağrı yapan(Call) Mesaj Tipi: Zaman uyumlu mesaj. Nesne mesajı alıcı nesneye gönderir ve onun işlemini bitirmesini bekler, bu durumda senkron mesaj tipi kullanılır. Nesne tabanlı programlamada çağırılan birçok method senkron çalıştığından en çok kullanılan mesaj tipidir.	
Asenkron Mesaj Tipi : Senkron mesajların tersine, asenkron mesajlar nesneye mesaj gönderdikten sonra cevap beklemeden işleme devam etmesinin gösteriminde kullanılır. Genellikle komut zincirlerinde kullanılır.	
Dönüş(Return) Mesaj Tipi: Senkron mesajlarda alıcı nesnenin işleminin bitimini, gönderen nesneye bildirmesinde kullanılır.	

Ardışıl Diyagramlar (Sequence Diagrams)

Mesajlar (Messages)

- **(a)** diyagramı zaman uyumlu akışa örnektir: Nesneler arasında gönderilen mesajın geri gönderdiği değer beklenir.
- **b)** diyagramında düz akış örneklenmiştir: Oklar kontrolü bir nesneden diğerine geçirir.
- **(c)** diyagramı ise zaman uyumsuz akışa örnektir: Hata yakalıyıcı nesnesi alarm nesnesine mesaj iletir ama cevabı beklemeden uygulama nesnesine hata değerini gönderir.



Ardışıl Diyagramlar (Sequence Diagrams)

Zaman (Time)

- Ardışıl Diyagramlar zamanı dikey doğrultuda gösterir. Zaman en tepede başlar ve aşağı doğru yayılır. Tepeye daha yakın olan mesaj aşağıya daha yakın olan mesajdan zaman olarak daha önce gerçekleşir.
- Ardışıl Diyagramlar'da, aktör figürü tipik ardışıllığı başlatıyor olarak görünse de, gerçekte aktör figürü Ardışıl Diyagram sembolleri arasında yer almaz.