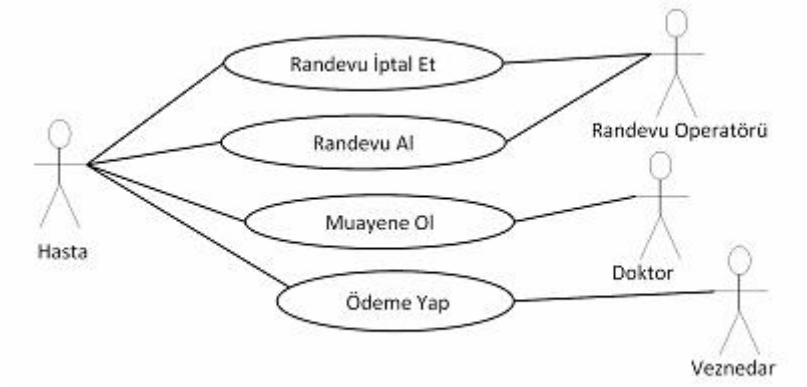
Use-Case Diagramları

Use-case, tek bir hedefi ya da görevi yerine getirecek özet bir senaryodur. Bu senaryoda **aktör** (actor), olayları başlatan veya tetikleyen şeydir. Use-case diyagramları, sistemin ne yaptığını dışarıdan bakan bir gözlemcinin bakış açısıyla anlatan diyagramlardır.



Yukarıdaki diyagram, bir kliniği anlatmaktadır. Bu klinikte hasta **aktörü** (actor), randevu ise use case'i ifade etmektedir. Burada hasta ile randevu arasında bir **ilişki** (association) vardır. Bu ilişki, **iletişim** (communication) veya **iletişim ilişkisi** (communication association) olarak da adlandırılır.

Diyagramlarda sistemin ne (what) yapıldığı üzerinde durulur. İşlemlerin nasıl (how) yapıldığı üzerinde durulmaz.

Genellikle use case diyagramlarında, aşağıda olduğu gibi, birden fazla use-case ve aktör bir arada gösterilir.

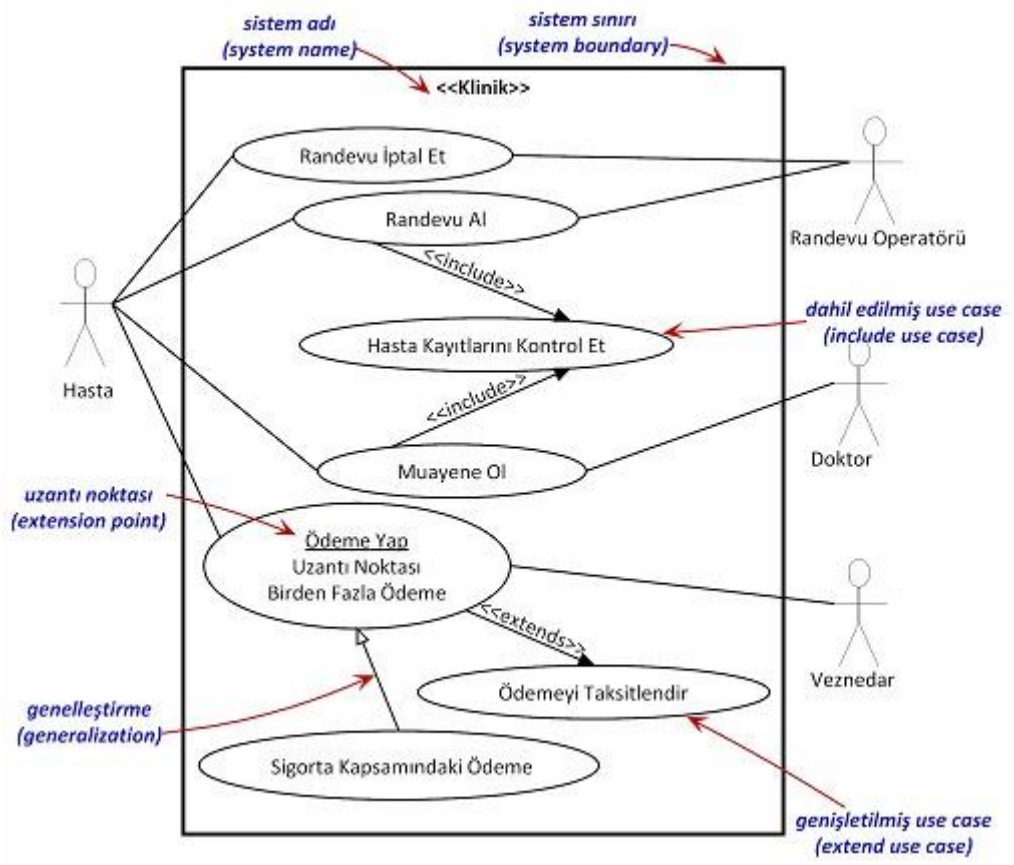
Bu diyagram, üç temel alanda bize yardımcı olur;

1. Sistemi belirleyen niteliklerin (determining features/requrirements) tespit edilmesinde,
2. Yazılımı geliştirenlerin, yazılımın kullanıcı tarafı ile ilgili kısımlarının (communicating with clients) daha iyi tanımlamasını sağlar,
3. Yazılımın test edilmesine yönelik örneklerin oluşturulmasında (generating test cases) yarar sağlar.

Her use-case diyagramı en az bir aktöre, use-case ve iletişime sahiptir. Yukarıdaki diyagramdan da görüleceği üzere use-case'ler arası birkaç ilişki vardır:

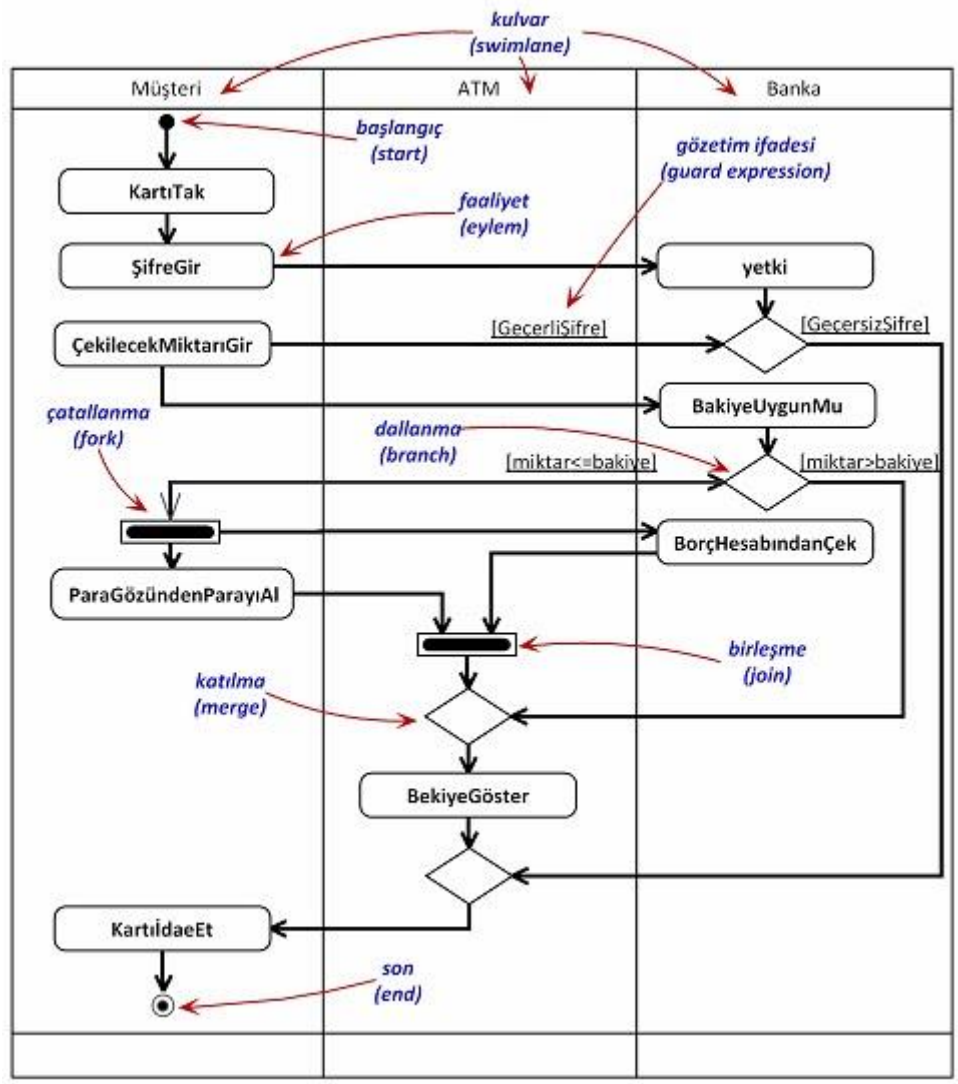
1. **Genelleştirme** (generalization): Bir use-case'in diğerinin özel bir hali olduğunu gösterir.
2. **İçerme** (include): Bir use-case'in bir başka use-case'i içine alması durumudur.
3. **Genişletme** (extend): Bir use-case'in bir başka use-case'in değişik biçimlerine (variation) sahip olduğunu gösterir.

Aşağıda bir kliniğin detaylı use case diyagramı verilmiştir.



Faaliyet Diyagramları

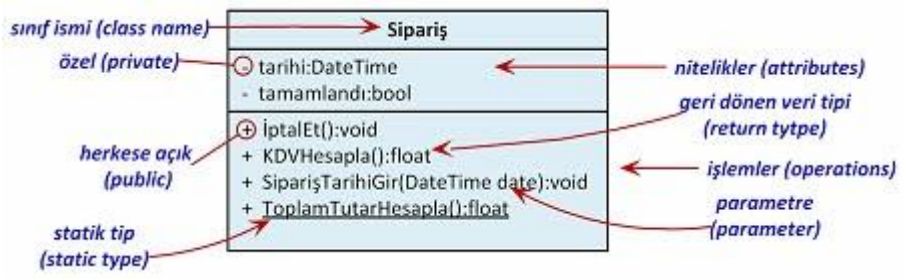
Durum diyagramı ile faaliyet (activity) diyagramları birbiriyle yakından ilişkilidir. Durum diyagramında nesnelerin içinde bulundukları süreçte ne yaptıkları üzerinde durulur. Faaliyet diyagramlarında ise söz konusu süreç içerisinde yapılan faaliyetlerin akışı ya da sırası üzerinde durulur.



Faaliyet diyagramları nesne **kulvarlarından** (swimlane) oluşur. Bu kulvarlarda nesnenin hangi faaliyetten (activity) sorumlu olduğu belirlenir. Bu diyagramda bir faaliyet bittikten sonra bir sonraki faaliyetin başladığını gösteren **geçişler** (transition) vardır. Bu geçiş birden fazla paralel faaliyet olabilir. Buna **çatallanma** (fork) adı verilir.

Sınıf Diyagramları

Sınıf diyagramları, yazılımı geliştirilecek olan sisteme ait sınıfları (class) ve bu sınıflar arasındaki ilişkiyi (relationship) gösterir. Sınıf diyagramları durağandır (static). Yani sadece sınıfların birbiriyle etkileşimini (what interacts) gösterir, bu etkileşimler sonucunda ne olduğunu (what happens) göstermez.



Sınıf diyagramlarında sınıflar, bir dikdörtgen ile temsil edilir. Bu dikdörtgen üç parçaya ayrılır. En üstteki birincisine sınıf ismi (class name) yazılır. Ortadaki ikincisinde nitelikler (attribute), en alttaki ve sonuncusunda ise ve işlemler (operation) yer alır.

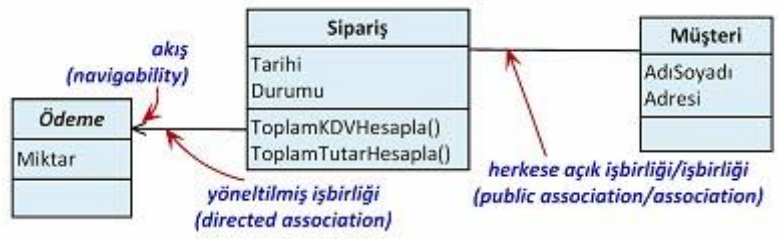
Sınıf isminin italik yazılması, sınıfın soyut (abstract class) sınıf olduğunu gösterir. Benzer şekilde işlemlerin yer aldığı kısımda, yöntemlerin italik yazılması halinde, ilgili yöntemin soyut yöntem (abstract method) olduğunu gösterir.

Sınıflarda görünürlük (visibility), adlarının önüne gelen özel karakterlerle birbirinden ayrılır. Burada "-" karakteri özel (private), "+" karakteri herkese açık (public), "#" karakteri ise korumalı (protected) olduğunu anlatmaktadır.

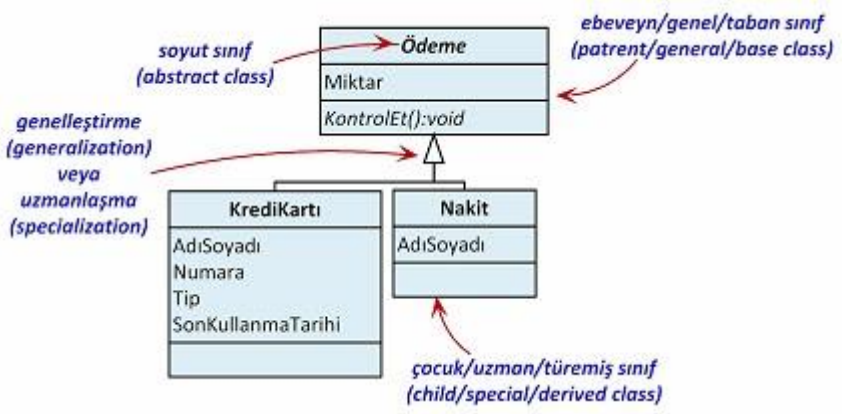
Statik üyeler (static member), altı çizili olarak gösterilir.

Sınıflar arasındaki ilişkiler ise bu dikdörtgenleri birbirine bağlayan çizgiler ile temsil edilir. *"Nesneler Arası İlişkiler"* başlığı altında bu ilişkilere ilişkin kodlama örneği ve ne oldukları anlatılmıştır.

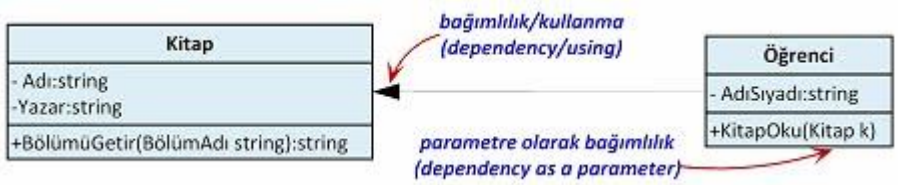
1. **İşbirliği** (public association/association): Düz bir çizgi ile gösterilir. Ucunda bir ok bulunan işbirliği ise yöneltilmiş işbirliğidir (directed association).



2. **Genelleştirme** (generalization): Ucunda üçgen olan bir çizgi ile gösterilir. Üçgenin sivri ucunun gösterdiği sınıf ebeveyn sınıftır.



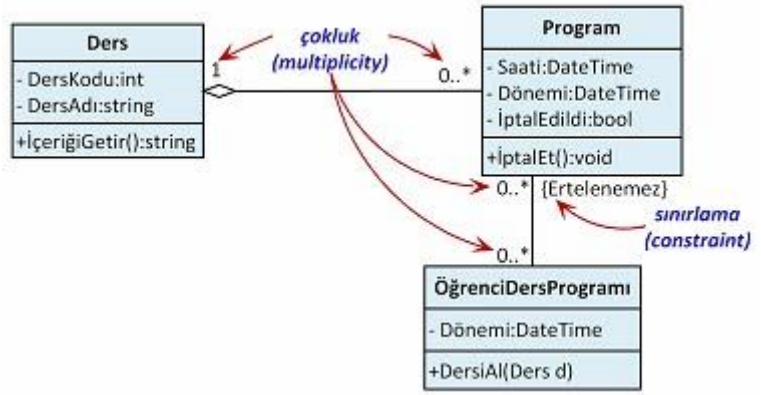
3. **Bağımlılık** (dependency) ya da **kullanma** (using):Ucunda ok bulunan kesikli bir çizgi ile gösterilir. Okun gösterdiği sınıfın, diğer sınıf tarafından kullanıldığını gösterir.



4. **Bileşim** (composition/private association): İçi dolu baklava dilimi ile başlayan düz bir çizgi ile gösterilir. Kitap sayfalardan oluşmak zorundadır. Yani bütün olarak kitap sayfalardan (parça) oluşur.

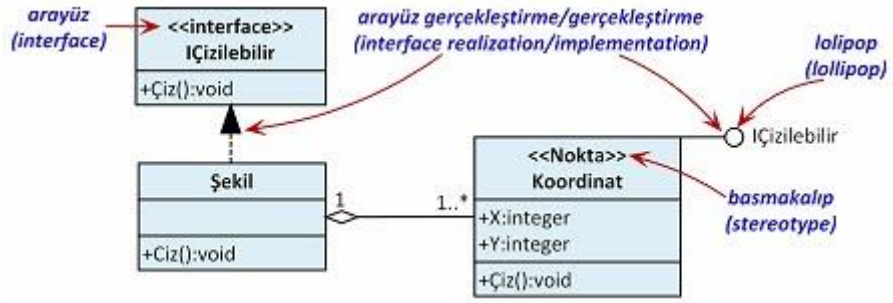
5. **Bütünleşme** (aggregation): İçi boş bir baklava dilimiyle başlayan düz bir çizgidir. Bileşim gibidir ancak bütünün n elzem olmadığı gösterilmiştir.

Sınırlamalar (constraint), tasarımın sağlaması gereken her bir durumu ifade eder. Sınırlamalar tırnaklı parantez içinde yazılırlar.



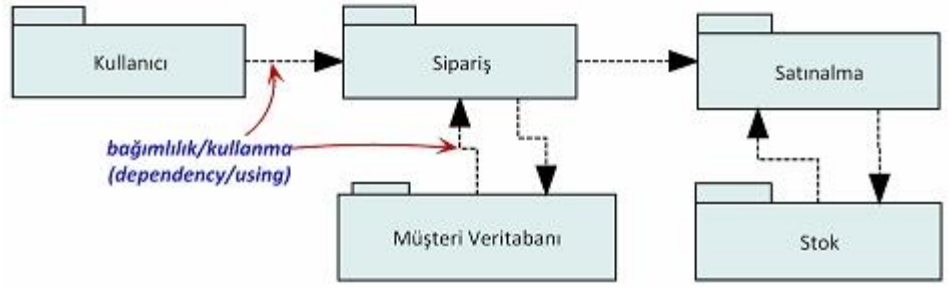
Çokluk (multiplicity), ilişkide bir sınıfın örnek (instance) sayısını gösterir. İlişkiyi gösteren çizginin sonunda bir numara olarak gösterilir. Bu numara, mümkün olabilecek örnek sayısıdır. Çokluk, işbirliği (association), bileşim (composition) ve bütünleşmeye (aggregation) uygulanır. Çokluk örnekleri:

- **0..1**: Sıfır ya da 1 örnek.
- **0..** yada ***: Örnek sayısı sınırsızdır.
- **1:** Yalnızca 1 örnek
- **1..**: En az bir örnek



Paket ve Nesne Diyagramları

Paket (package), mantıksal olarak birbirine bağlı birkaç UML elemanının bir araya gelmiş halidir. Paketler, üzerinde kare çıkıntıları olan dikdörtgenlerle temsil edilirler.


Kullanıcı
Sipariş
Satınalma
bağımlılık/kullanma
(dependency/using)
Müşteri Veritabanı
Stok