**Birleşik Modelleme Dili (UML)**

UML, bir programlama dili değil yazılım geliştirme için kullanılan standart bir diyagram çizme ve ilişkisel modelleme dilidir. Yazılım sistemlerinin nasıl modellenebileceğini belirleyen ve açıklayan yöntemlerin bir araya toplanmış halidir.



Programın analiz ve dizayn aşamasında UML'e büyük ölçüde ihtiyaç duyulmaktadır. Düzgün bir modellemeyle birçok problemin çıkmasına engel olunabilir. Daha çok nesneye dayalı programlama dilleri için uygun olan UML, problemleri parçalara ayırıp parçalar arasında belirli ilişkiler sağlayabilmeye dayanır. Eğer problem parçalara ayrılabiliyorsa ve parçalar arasında belirli ilişkiler sağlanabiliyorsa UML kullanılması büyük fayda sağlayacaktır.

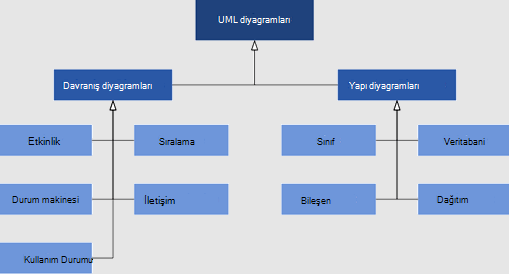
UML'nin Faydaları

* Analizi ve tasarımı geniş bir şekilde yapıldığından kodlama işlemi daha kolay olur. Çünkü programdan beklenilenler ve programlama ile yapılacaklar belirlenmiştir.
* Beklenmedik bir takım mantıksal hatalar (bug) en aza indirgenir.
* Düzgün yapılan tasarım aşamasından sonra, tekrar kullanılabilen kodların sayısı artacak ve program geliştirme maliyeti büyük ölçüde düşecektir.
* UML diyagramları programın tamamını kapsayacağı için bellek kullanımını daha etkili hale getirilebilir.
* Programın kararlılığı artacağı gibi döküman halindeki programı koda çevirmek zamandan da tasarruf etmeyi sağlar.
* Ortak çalışılan projelerde programcıların iletişimi daha kolay hale gelir. Çünkü UML ile program parçalara ayrılmıştır ve parçalar arasında bir ilişki kurulmuştur.

UML Diyagramları

UML, nesneler arasında ilişki kurmak için bir takım grafiksel elemanlara sahiptir. Bu elemanları kullanarak diyagramlar oluşturulur. UML temel olarak aşağıdaki diyagram türlerini kapsar:

* **Sınıf (Class) Diyagramları**: Sınıf, aynı işlevlere, aynı ilişkilere ve aynı anlama sahip nesneler topluluğunun ortak tanımıdır. Sınıflar yazılımın durağan yapısının tanımlanmasında kullanılırlar.
* **Nesne (Object) Diyagramları:** Nesne, sınıfın bir örneğidir. Bu tür diyagramlarda sınıfın yerine her bir sınıftan oluşturulmuş nesneler yer alır.
* **Durum (State) Diyagramları**: Gerçek nesnelerin herhangi bir zaman içindeki durumunu ve durumunun zaman içinde nasıl bir değişim gösterdiğini modelleyen diyagramlardır. Genel olarak durum diyagramları tüm nesneler için değil yalnızca karmaşık olan, davranışı, kendine gönderilen iletilerin yanısıra o an içinde bulunduğu duruma göre de farklılık gösteren nesneler için oluşturulur.
* **Ardıl Etkileşim (Sequence) Diyagramları:** Sınıf ve nesne diyagramları durağan bilgiyi modeller. Ancak gerçek zamanlı sistemlerde zaman içinde değişen durumlar bu diyagramlarla gösterilemez. Bu tür zamanla değişen durumları belirtmek için nesnelerin birbirleriyle zamana bağlı olarak haberleşmelerini ele alan ardıl etkileşim diyagramları kullanılır.
* **Etkinlik (Activity) Diyagramları:**Bir nesnenin durumu zamanla kullanıcı tarafından ya da nesnenin kendi içsel işlevleri tarafından değiştirilebilir.Bu değişim sırası etkinlik diyagramlarıyla gösterilir.
* **Kullanıcı Senaryosu (Use Case) Diyagramları:** Programın davranışının bir kullanıcı gözüyle incelenmesi kullanıcı senaryosu diyagramlarıyla yapılır. Gerçek dünyada insanların kullanacağı bir sistemde bu diyagramlar büyük önem taşırlar.
* **İş Birliği (Collaboration)Diyagramları: Bir** çok parçadan oluşan projelerde bir işin amacına uygun şekilde çalışabilmesi için bütün parçaların işlevlerini eksiksiz yerine getirmesi gerekir. Bu parçalar arasındaki ilişki iş birliği diyagramlarıyla gösterilir.
* **Bileşen (Component) Diyagramları**: Özellikle çok sayıda kişinin çalıştığı büyük çaplı projelerde, projeyi bileşenlerine ayırmak gerekmektedir. Sistemin doğru modellenmesiyle bileşenlerin ayrı ayrı çalışması sağlanmalıdır. Bu tür modellemeler bileşen diyagramlarıyla yapılmaktadır.
* **Dağıtım (Deployment) Diyagramları:** Dağıtım diyagramları yazılımın nasıl dağıtılacağının planlandığı aşamada kullanılırlar. Sistemin fiziksel incelemesini yapmaktadırlar. Bilgisayarlar arasındaki baglantılar, programın kurulacağı makinalar, sistemdeki ağ ve yazıcı bağlantıları gibi her türlü detay dağıtım diyagramlarında gösterilir.
* **Paket (Package) Diyagramları:** Paket diyagramları, büyük yazılımlarda sistemi oluşturan alt yazılımlar veya etkileşimde bulunulan yan sistemler olduğu durumlarda sistemler arası etkileşimi gösteren kısaca sistem mimarisinin paket yönünü özetleyen bir diyagramdır.



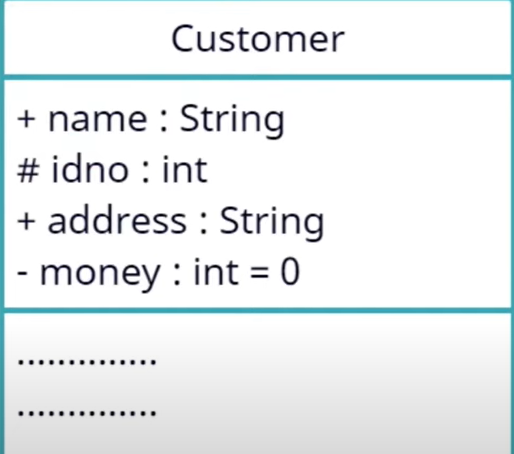
**Sınıflar**

****

Şekilde görüldüğü üzere dikdörtgen 3 parçaya bölünerek ilk parçasına “sınıf Adını” ikinci parçaya “sınıfın nitelikleri” son parçaya “sınıfın davranışları” yazılır.

Programlamada sınıfların niteliklerini “değişkenler” davranışlarını da metodlar tanımlar.

**Değişkenler**



Sınıfa ait niteliklerin söz dizimi (syntax):



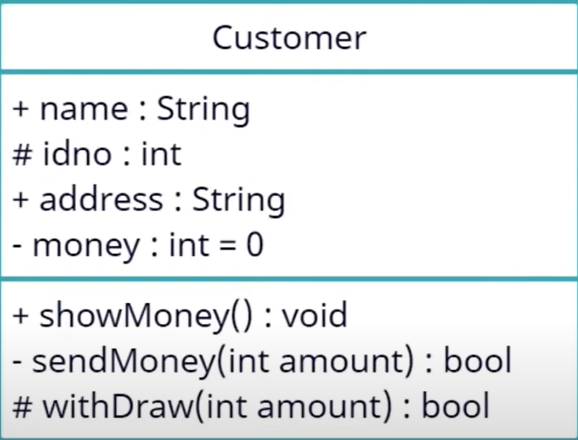
Görünürlük(visibility): Niteliğin dışarıdan erişim ilkesinin ne olduğunu belirtir. Public ise “+” ,private ise “-“ protected ise “#”.

İsim (Name); Niteliğe ait isimlendirme yapılır.

Tür(Type): Bir niteliğin veri tipini belirtir.

Varsayılan Değer (Default Value): Niteliğin ilk değerini belirtir.

**Davranışlar(metotlar)**



Sınıfa ait davranışlara ait sözdizmi(syntax):



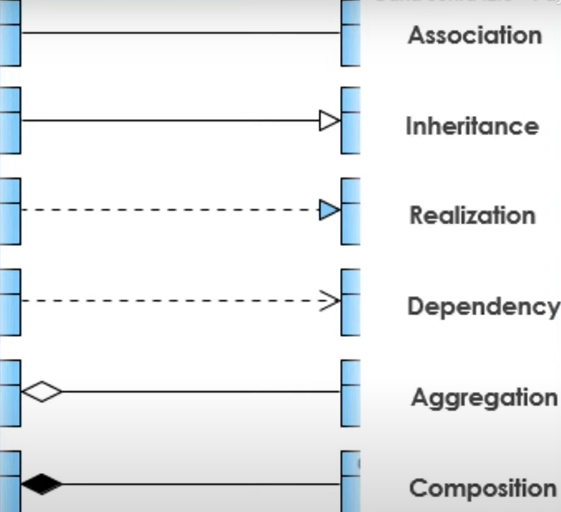
Görünürlük(visibility): Bu davranışın dışarıdan erişim ilkesinin ne olduğunu belirtir. Alabileceği değerler niteliklerde olduğu gibidir.

İsim(Name): Metoda ait isimlendir yapılır.

Parametre Listesi(Parameter List): Programlamada metodun aldığı parametreler tür bilgileri ile birlikte buraya yazılır.

Geri dönüş türü(Return Type): Metodun geri dönüş tipini belirtir.

Sınıflar Arası İlişkiler

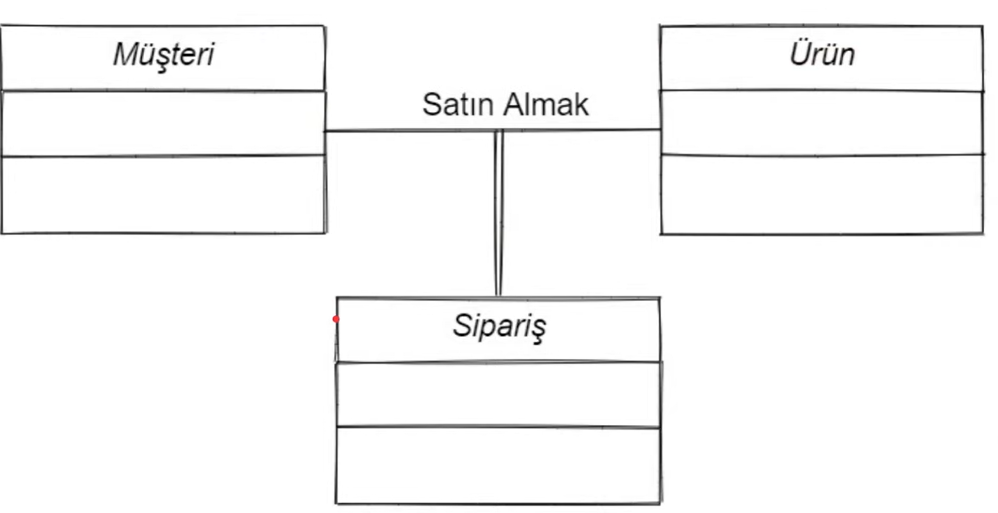


Sınıflar birbirleriyle ilişki içerisinde olan yapılardır. Bu ilişkileri UML diyagramlarında da göstermek gerekir. Elbette ilişkinin türüne gör egösterim de değişiklik gösterecektir.

* Bağlantı İlişkisi(Association)
* Genelleme/Kalıtım İlişkisi(Generalization/Inheritance)
* Bağımlılık İlişkisi(Depedency) (Aggregation,Composition)
* Gerçekleştirim İlişkisi (Realization)

**Bağıntı İlişkisi (Association Class)**

Sınıflar arası ilişkiler çizgi ile gösterilir ve çizginin üstüne ilişki şekli yazılır. Sınıflar arası ilişkiler bire bir,bir çok,bire n gibi olabilir. Örneğin aşağıdaki gösterimde e-ticaret sisteminde müşteri ve ürün ilişkisini görmektesiniz. Örnekteki diyagrama göre müşteri ve ürün arasında satın alma ilişkisi vardır ve bunun için de sipariş oluşturması gerelir.



Sınıf diyagramlarında sınıflar arasında bire n ilişki kurulabilir. Bir sınıf n tane başka bir sınıf ile ilişkili ise buna bire-çok (1-n) ilişki denir.

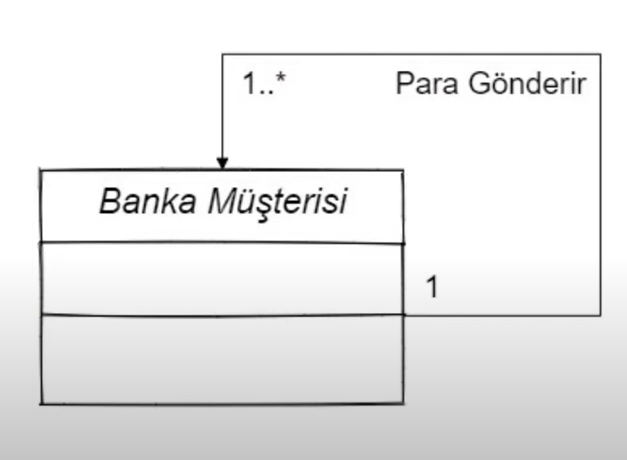


Bu örnekte ise bir sepette n tane ürün olabileceği gösterilmiştir. İki sınıf arasında yalnızca tek bir bağıntı çizilmesi gibi bir kısıt yoktur. En temel bağıntı ilişki tipleri aşağıdaki gibidir;

* Bire-bir(1-1)
* Bire-çok (1-\*)
* Çoka-bir(\*-1)
* Bire-sıfır(1-0)
* Bire-Beş yada Bire-sekiz(1-5 veya 1-8)

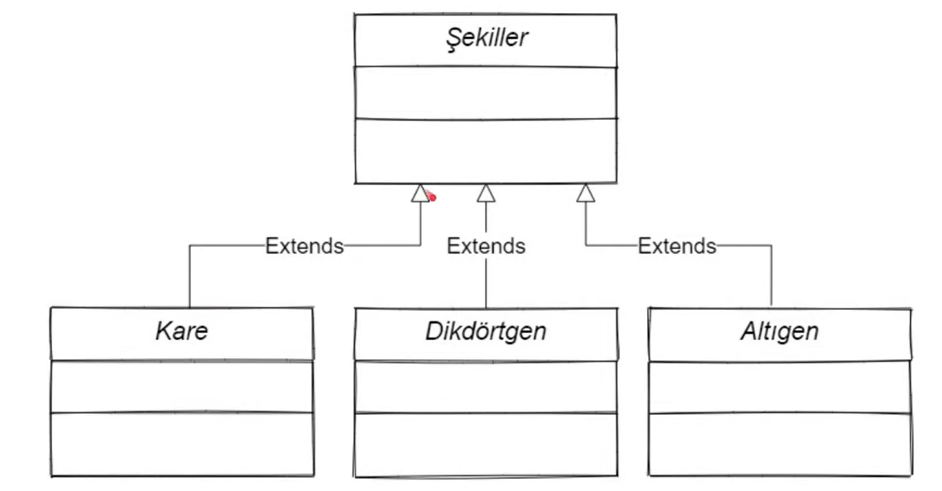
Reflexive Associations

Sınıfın kendisiyle kurduğu ilişkidir. Bu tür ilişkiler genellikle bir sınıfın sistemde birdenf azla rolü varsa ortaya çıkar.



**Kalıtım(Inheritance) ve Genelleme (Generalization) İlişkisi**

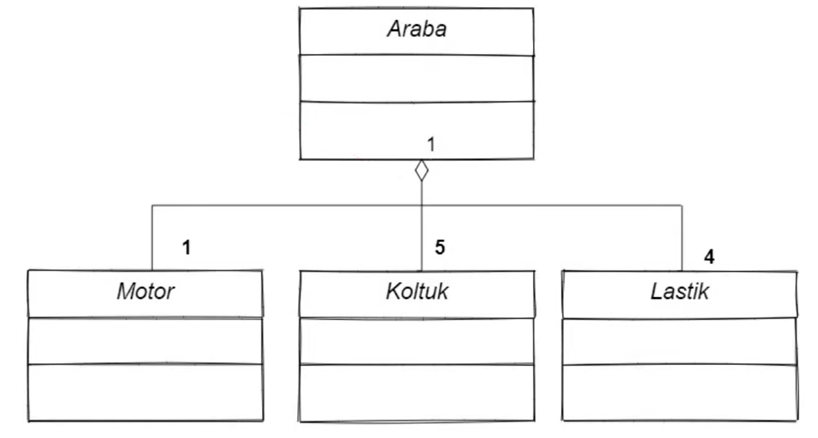
Türetme yoluyla bir sınıf başka bir sınıfın var olan özelliklerini alarak, o sınıf türünden başka bir nesneymiş gibi kullanılabilir. Bir sınıfın işlervleri türetme yoluyla genişletilecekse , türetmenin yapılacağı sınıfa taban sınıf(süper class), türetilmiş sınıfa da türemiş sınıf (sub class) denir. Sekilsel olarak türemiş sınıftan taban sınıfa bir ok olarak belirtir.



İçerme (Aggregation) İlişkisi

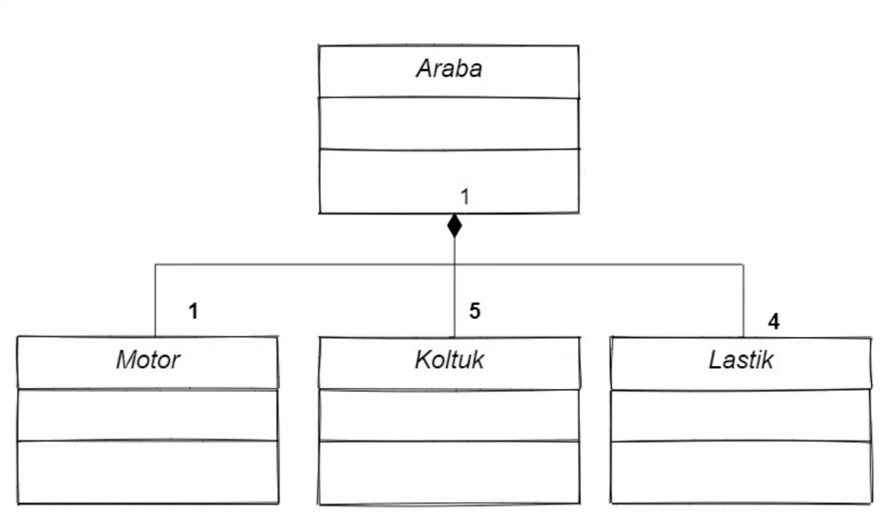
Birden fazla parçadan oluşan sınıflar arasındaki ilişkite “Aggregation” denir. Aggregation ilişkisini ‘bütün parça’ yukarıda olacak şekilde ve bütün parçanın ucuna içi boş elmas yerleştirecek şekilde gösterilir. İçi boş elmas ile gösterilen ilişkilerde her bir parça ayrı bir sınıfıt ve tek başlarına anlam ifade ederler.

Örneğin Araba sınıfını 1 motor, 5 koltuk ve 4 lastik sınıflarının oluşturduğunu düşünürsek aralarındaki ilişki şekildeki gibidir.



Oluşma (Composition) İlişkisi

Asıl sınıf üretildiğinde parçaları da üretilecek ise bu ilişkiye composite denir. Parça-bütün ilişkilerini modellemekte kullanılırlar. Bütün nesneler yaratıldığında parçaları da yaratılmaktadır. Bütün ve bütünü oluşturan parçalar arasında sıkı bir ilişki vardır. Oluşma ilişkisi ‘bütün parça’ yukarıda olacak şekilde ve bütün parça ucuna dolu bir elmas gelecek şekilde gösterilir.



**Gerçekleştirim (Realization) İlişkisi**

Gerçekleştirim ilişkisi en çok kullanıcı arayüzlerinin (user interface) modellemesinde kullanılır. Arayüz yalnızcs 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.

