UML NEDİR?

UML (Unified Modeling Language - Birleşik Modelleme Dili), yazılım ve sistem tasarımını anlamak ve iletmek için kullanılan bir standarttır. UML'nin bir parçası olan UML sınıf diyagramları, bir sistemde bulunan sınıfları, aralarındaki ilişkileri ve sınıfların özelliklerini gösteren görsel temsillerdir.

Bir yazılımın hayata geçirilmesinde *Tasarım Mühendisi/Uzmanı, Yazılım Mühendisi/Uzmanı, Analist Uzmanı, Test Mühendisi/Uzmanı, Kalite Mühendisi/Uzmanı/Sorumlusu, Kullanıcılar* vs. gibi farklı görev tanımlamaları bulunmaktadır. Bir yazılım için her kişinin farklı bakış açısı vardır. Müşteri açısından projeye baktığımızda müşteriyi işlerin sıralandırılması, sisteme artıları ve eksiler, işler arasındaki ilişkiler ilgilendirirken bir fonksiyonun detayları ilgilendirmemektedir. Analist açısından baktığımızda nesne özellikleri, fonksiyonlar ve alacakları parametreler yeterli iken tasarımcı açısından parametrelerin veri tipleri, fonksiyonun performansı, yaşam süresi gibi bilgiler de önemli olmaktadır.

Bu nedenle UML bu ekip için gerekli farklı diyagramlar içermektedir. Yazılım geliştirme işinde yer alacak farklı ekiplerin farklı bakış açılarına uygun farklı UML diyagramları bulunmaktadır.

NEDEN UML?

Yazılım dünyasında UML kullanmanın birkaç nedeni vardır. Bunlar:

1. *Görsel Temsil:* UML, sistem tasarımını ve yapılarını görsel olarak temsil etmek için kullanılır. İnsanlar genellikle görsel bilgileri daha hızlı ve etkili bir şekilde anlarlar, bu nedenle UML diyagramları, karmaşık sistemleri anlamak ve iletmek için güçlü bir araçtır.
2. *İletişim ve Anlaşılabilirlik*: Yazılım geliştirme sürecinde, ekipteki farklı üyeler arasında ve müşteri ile geliştirici arasında etkili iletişim önemlidir. UML, karmaşık yazılım tasarımlarını daha anlaşılır ve ortak bir dilde ifade etmek için kullanılır. Böylece, ekip üyeleri ve paydaşlar arasında daha iyi bir anlayış sağlar.
3. *Analiz ve Tasarım*: UML, yazılım geliştirme sürecinin analiz ve tasarım aşamalarında kullanılır. Sistem gereksinimleri ve tasarım kararları, UML diyagramları aracılığıyla belgelenir. Bu, geliştiricilere sistem mimarisini daha iyi anlama ve uygulama aşamasında daha etkili bir şekilde çalışma olanağı tanır.
4. *Modülerlik ve Kapsamlılık*: UML, sistemdeki farklı bileşenleri (sınıflar, modüller, alt sistemler vb.) ve bu bileşenler arasındaki ilişkileri açıkça tanımlayarak modüler bir

yaklaşım sağlar. Bu sayede, sistemdeki değişiklikleri daha rahat bir şekilde yönetmek mümkün olur.

5. *Sürdürülebilirlik*: UML, sistem tasarımının belgelenmesini ve sürdürülebilirliğini artırır. UML diyagramları, yazılımın evrimi ve bakımı sırasında daha iyi bir rehberlik ve anlama sağlar.
6. *Endüstri Standardı*: UML, bir endüstri standardı olarak kabul edilmiştir. Bu standart, farklı yazılım geliştirme ekipleri ve organizasyonları arasında ortak bir dil ve yaklaşım sağlar.

UML CLASS DİYAGRAMLARI

UML sınıf diyagramları, bir sistemdeki sınıfları, sınıflar arasındaki ilişkileri, sınıfların özelliklerini (attributes) ve metotlarını (methods) gösteren bir tür UML diyagramıdır. Bu diyagramlar, bir yazılım sisteminin yapısal özelliklerini görsel olarak temsil etmek için kullanılır.

UML sınıf diyagramlarını anlatan temel kavramlar şunlardır:

Sınıflar (Classes): Sınıf diyagramları, bir sistemde bulunan sınıfları temsil eder. Sınıflar genellikle nesnelerin temsili olarak kullanılır ve bu nesnelerin özelliklerini ve davranışlarını içerir.

Özellikler (Attributes): Sınıfların özellikleri, sınıfa ait verileri temsil eder. Örneğin, bir "Araba" sınıfının özellikleri "model", "renk", "hız" gibi olabilir.

Metotlar (Methods): Sınıfların metotları, sınıfın davranışlarını temsil eder. Örneğin, bir "Araba" sınıfının metotları "hızlan", "yavaşla" gibi olabilir.

İlişkiler (Relationships): Sınıflar arasındaki ilişkiler, sistemdeki farklı sınıfların birbirleriyle nasıl bağlantılı olduğunu gösterir. İlişkiler genellikle oklarla temsil edilir ve bu oklar, bir sınıfın diğerine olan bağımlılığını gösterir.

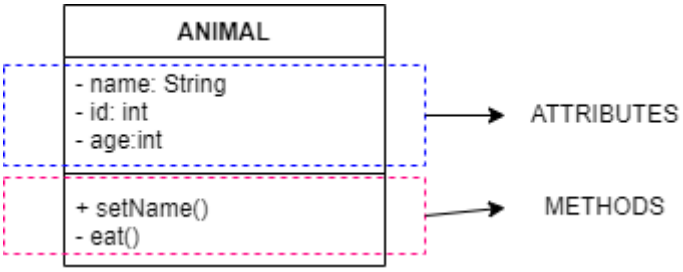
İlişki Türleri:

- ***Bağlantı (Association):*** İki sınıf arasındaki temel ilişkiyi temsil eder. Örneğin, bir "Öğrenci" sınıfı ile "Sınıf" sınıfı arasında bir bağlantı olabilir.
- ***Kompozisyon (Composition):*** Bir sınıfın diğer bir sınıfa tamamen bağımlı olduğunu ifade eder. Yani bir sınıfın ömrü diğer sınıfın ömrüne bağlıdır. Bu ilişki genellikle "bölüm-bütün" ilişkisi olarak düşünülür.

- ***Aggregasyon (Aggregation):*** Bir sınıfın diğerine bağımlı olduğunu, ancak ömrünün tamamen bağlı olmadığını ifade eder. Bu ilişki genellikle "parça-bütün" ilişkisi olarak düşünülür.

Multiplicity (Çokluğ): Sınıflar arasındaki ilişkilerde kaç tane nesnenin birbirine bağlı olduğunu ifade eder. Örneğin, bir sınıfın bir ilişkide "0..1" multiplicity'si, o sınıfa ait bir nesnenin ilişkide hiç veya bir kez bulunabileceğini gösterir.

Arayüzler (Interfaces): Arayüzler, bir sınıfın bir kontratı uyguladığını ve belirli metotları içermek zorunda olduğunu gösteren stereotiplerle temsil edilebilir.



Şekil 1. Class Diyagramı

Örnek bir class diyagramı *Şekil-1*'de gösterilmektedir. Şekil-1'de yer alan class diyagramının en üst kısmında class ismi bulunurken, orta kısımda class'ın literatürde attributes olarak adlandırılan *davranışları, nitelikleri* ya da *değişkenleri* yer almaktadır. En alt kısımda da sınıfa ait *metotları* yer almaktadır. Değişken ve metotların sol tarafında yer alan işaretler ilgili değişken ya da metodun erişim belirleyicisini ifade etmektedir. Literatürde "*Access Modifiers*" olarak geçmektedir. Bu erişim belirleyicisinin gösterim şekilleri *Tablo-1*'de ifade edilen şekilde gösterilir.

Tablo 1. Erişim Belirleyicisi Gösterimleri ve Anlamları

Simge	**İsim**
-	Private
+	Public
#	Protected
~	Default

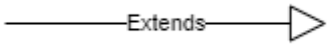
Ancak, sınıfın tek başına gösterimi bir şey ifade etmemektedir. Birden fazla sınıf varsa ve bu sınıflar arasında ilişki varsa sınıflar arasındaki bağlantıların gösterimi önem arz etmektedir. UML'de ilişki gösterim şekilleri şu şekildedir.

1. Generalization/Inheritance
2. Realization/Implementation
3. Association
4. Dependency (Aggregation & Composition)

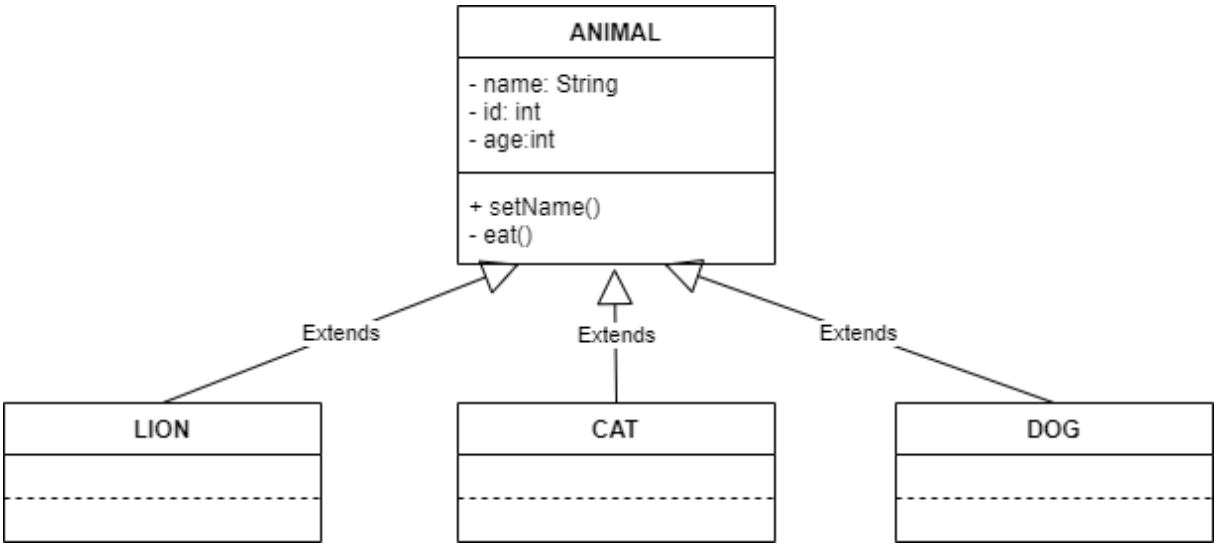
1. GENERALIZATION/INHERITANCE

Bu gösterim sınıflar arasındaki kalıtımın nasıl oluşturulduğunu ifade etmektedir. *Şekil-2*'de yer alan ok şekli, sınıflar arasında inheritance olduğunu belirten bir gösterimdir.

Şekil-3'de yer alan gösterim, sınıflar arasında kalıtım yoluyla oluşturulmasını ve bu gösterimin sınıf diyagramlarında nasıl gösterildiğine dair bir örnektir.



Şekil 2. Sınıflar Arasındaki Kalıtım İlişkisinin Gösterimi

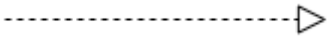


Şekil 3. Sınıflar Arasındaki Kalıtım İlişkisinin Gösterimi

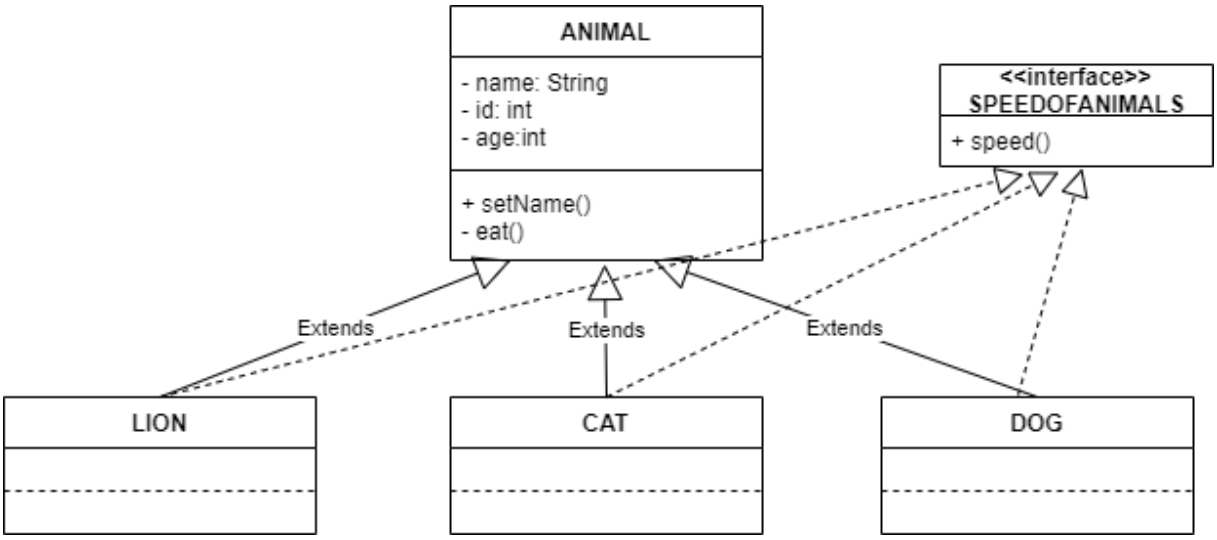
Şekil-3'te yer alan gösterim, abstract sınıftan kalıtım alırken de aynıdır. Bu tür kalıtım ilişkisinde nesneler arasında "**IS-A**" ilişkisi bulunmaktadır. Örneğin; "LION IS A ANIMAL". "Lion bir hayvandır."

2. REALIZATON/IMPLEMENTATION

Bu ilişki arayüzler ile sınıflar arasındaki ilişkiyi modellemek için kullanılır. Dashed (kesikli) çizgi ile ifade edilir. Kalıtımdaki çizginin kesik kesik olan halidir ve Şekil-4'te gösterimi yer almaktadır.



Şekil 4. Interface ve Sınıflar Arasındaki İlişkinin Gösterimi



Şekil 5. Interface ve Sınıflar Arasındaki İlişkinin Gösterimi

Şekil-5'te Lion, Cat ve Dog sınıfları Animal isimli sınıftan kalıtım yoluyla oluşturulurken, Speed of Animals (SPEEDOFANIMALS) isimli sınıfı da arayüz olarak implemente etmektedir. Bu ilişki türünde nesneler arasında "**HAS-A**" ilişkisi bulunmaktadır. Örneğin; "DOG HAS A SPEED", "Köpek, bir hıza sahiptir" veya "Köpeğin bir hızı vardır."

3. ASSOCIATION

Class diyagramlarında en sık kullanılan ilişki türlerinden biri olan 'Association', tasarım desenleriyle ilgili yazılarımda sıklıkla rastladığımız bir kavramdır. Multiplicity kavramı bu noktada devreye girmektedir. Çünkü Association'larda sınıflar arasındaki bağlılık ilişkisinde bu kavram önemli bir yer tutmaktadır ve *Tablo-2*'de gösterimi verilmiştir.

Tablo 2. Sınıflar Arasında Multiplicity Gösterimi

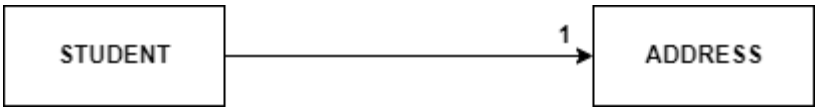
0	Herhangi bir örneği yok
0..1	Herhangi bir örneği yok ya da bir örneğe sahip
1	Sadece bir örneği var.
1..1	Kesinlikle sadece bir örneği var.
0..*	Sıfır ya da daha fazla örneğe sahip.
*	Sıfır ya da daha fazla örneğe sahip.
1..*	En az bir örneğe sahip ya da daha fazla örneğe sahip.

Association 4 (dört) çeşide ayrılmaktadır. Bunlar;

- Uni-Directional (Tek Yönlü)
- Bi-Directional (Çift Yönlü)
- Reflexive
- Aggregation & Composition

UNI-DIRECTIONAL ASSOCIATION

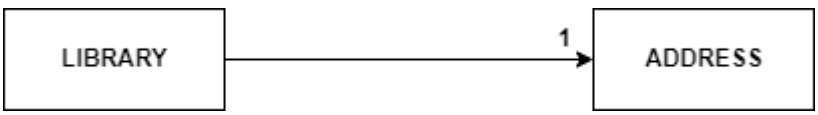
Bu gösterim, sınıflar arasında **tek yönlü bir ilişki** olduğunda kullanılmaktadır. Örnek gösterimi Şekil-6 ve Şekil-7'de verilmiştir.



Şekil 6. Uni-Directional Association

Şekil-5'te yer alan ilişkinin okunuşu "STUDENT HAS A ADDRESS", "Öğrenci, bir adrese sahiptir." şeklindedir. Burada 1 yerine 0...* şeklinde de bir ifade kullanılabilir. Şekil 5'te ilişkide 1 yazdığı için, Student sınıfı içerisinde Address sınıfı tipinde bir attribute bulunacağını belirtiyor. Ancak Address sınıfında Person ile ilgili bir bilgi yer almıyor. Çünkü ilişki türü uni-directional şeklindedir.

Farklı bir örnek incelenecek olursa;

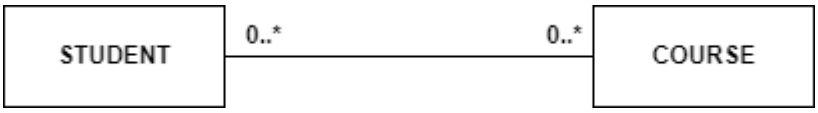


Şekil 7. Uni-Directional Association

Şekil-6’da yer alan diyagramda, "Library" sınıfı "Book" sınıfına doğru bir uni-directional association'a sahiptir. Her bir kütüphane bir veya birden fazla kitaba sahip olabilir, ancak her bir kitap bir kütüphaneye ait değildir. Yani, "Library" sınıfı "Book" sınıfına doğru bir ilişkiye sahiptir, ancak ters yönde bir bağlantı yoktur. Library sınıfı içerisinde Book tipinde bir attribute olacağını belirtiyor. Bu durumu ifade etmek için ok sadece "Library" sınıfından "Book" sınıfına doğru çizilmiştir. Bu, bir kütüphanenin bir veya birden fazla kitaba sahip olabileceğini, ancak bir kitabın bir kütüphaneye ait olmadığını ifade etmektedir.

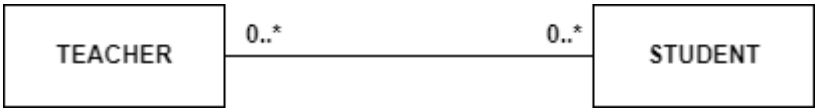
BI-DIRECTIONAL ASSOCIATION

Bu gösterim, sınıflar arasında **çift yönlü bir ilişki** olduğunda kullanılmaktadır. Örnek gösterimi Şekil-8 ve Şekil-9’da verilmiştir.



Şekil 8. Bi-Directional Association

Şekil-8’te yer alan diyagramda Öğrenci sınıfı hiçbir derse kayıt olamayacağı gibi çok fazla derse de kayıt olabilir. Aynı şekilde Ders’e hiçbir öğrenci kayıt olamayacağı gibi çok fazla öğrenci de kayıt olabilir. 0..* yerine 1..* şeklinde de yazabiliriz. Bu durumda Öğrenci, en az bir derse kayıt olmak zorunda ve Ders ise en az bir öğrenciye sahip olmak durumundadır. Student sınıfında Course sınıfından yaratılmış nesnelerin listesi, Course sınıfında da Student sınıfından yaratılmış nesnelerin listesi tutulmaktadır.

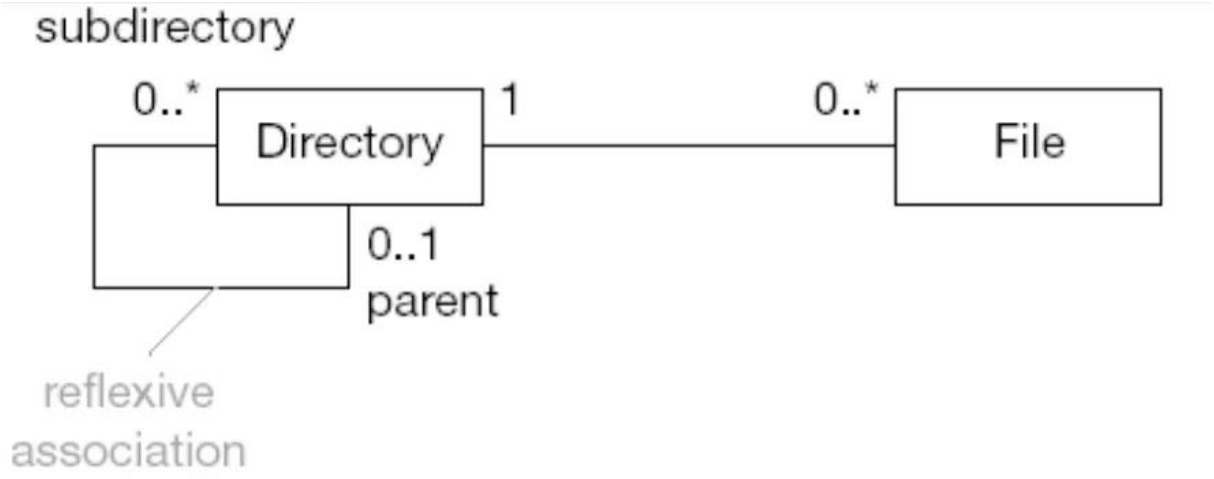


Şekil 9. Bi-Directional Association

Şekil-9’da yer alan diyagramda Öğretmen, bir öğrenciye sahip olamayacağı gibi çok fazla öğrenciye de sahip olabilir. Aynı şekilde, Öğrenci, herhangi bir Öğretmen’e sahip olamayacağı gibi birden fazla Öğretmen’e de sahip olabilir.

REFLEXIVE ASSOCIATION

Reflexive (dönüşlü) yani sınıfın kendisi ile yaptığı ilişkidir. Örnek gösterimi Şekil-10'da verilmiştir.



Şekil 10. Reflexive Association

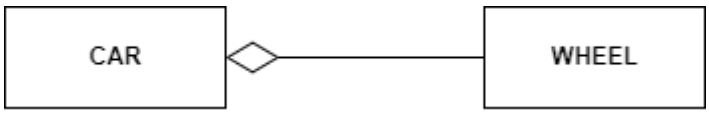
Şekil-10'da Reflexive ilişki için güzel bir örnek yer almaktadır. Directory (Dosya dizini) hiç ya da sonsuz sayıda alt dosya dizinine sahip olabilir. Aynı şekilde hiç ya da 1 tane üst dizine sahip olabilir. Directory sınıfı kendi sınıfından türettiği 2 tane directory nesnesi ile üstteki modeli gerçekleştirebilir.

AGGREGATION AND COMPOSITION (DEPENDENCY)

Aggregation ve Composition, UML sınıf diyagramlarında kullanılan iki özel association (ilişki) türüdür. Bu iki kavram, nesneler arasındaki ilişkiyi daha ayrıntılı bir şekilde belirtmek için kullanılır.

AGGREGATION

Aggregation (Birleştirme), bir bütünün parçalarından oluştuğu ve bu parçaların bağımsız olarak var olabileceği bir ilişki türünü temsil eder. Bu ilişki genellikle "parça-bütün" ilişkisi olarak düşünülür. Aggregation ilişkisi, bir nesnenin başka bir nesneyi içerebileceğini, ancak bu içerilen nesnenin bağımsız olarak var olabileceğini ifade eder. Bu tür ilişkiyi "HAS-A" ya da "IS-PART-OF" şeklinde okunabilir. Şekil-11'de örnek gösterimi yer almaktadır.

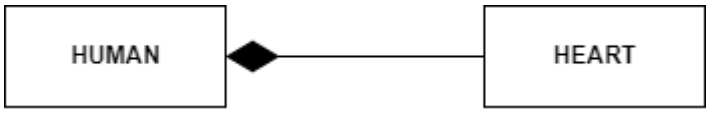


Şekil 11. Aggregation Association

Şekil 11’de yer alan diyagramda, sınıflar arasında Aggregation ilişkisini görebilirsiniz. Tekerlek araba sınıfının bir parçasıdır. Ancak araba sınıfı yok olduğunda tekerlek yok olmak zorunda değildir. Aralarında zayıf bir parça ilişkisi vardır.

COMPOSITION

Composition, Aggregation'a benzer şekilde bir bütünün parçalarından oluştuğu bir ilişki türünü temsil eder. Ancak, Composition ilişkisinde parçalar, bütünden bağımsız olarak var olamazlar. Eğer bütün silinirse, parçalar da silinir. Composition ilişkisi genellikle daha güçlü bir "bütün-parça" bağlantısı sağlar.



Şekil 12. Composition Association

Şekil 12’de yer alan diyagramda, sınıflar arasında Composition ilişkisini görebilirsiniz. Kalp, insan sınıfının bir parçasıdır. İnsan sınıfı yok olduğunda kalpte yok olacaktır. İki sınıf arasında güçlü bir parça ilişkisi vardır. Composition ile agreegation arasındaki fark budur.