

# UML

## Unified Modelling Language

### Birleşik Modelleme Dili

Ders notu

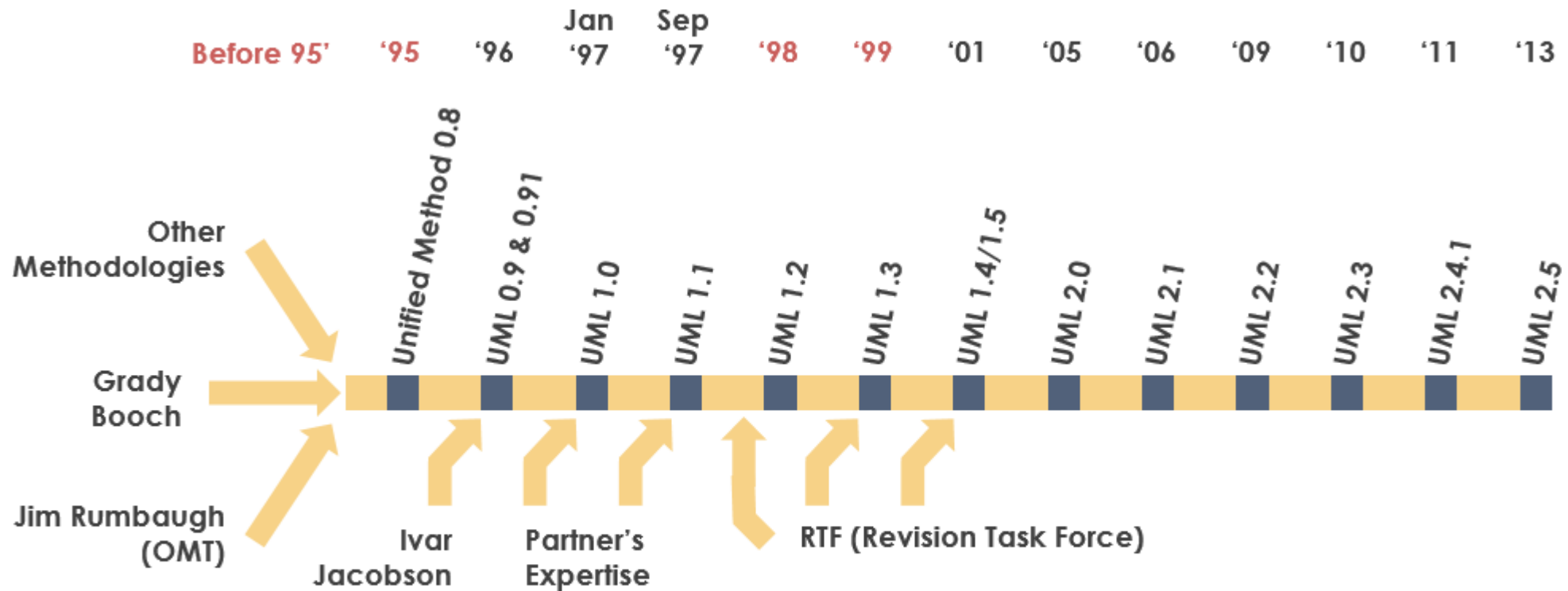
<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/> ve  
<https://online.visual-paradigm.com/diagrams/tutorials/>

linklerinden oluşturulmuştur.

# UML (Birleşik Modelleme Dili)

- Sistem ve yazılım geliştiricilerinin kullandığı standartlaştırılmış bir modelleme dilidir.
- Yazılım sistemlerinin bileşenlerinin belirtimi (gereksinimlerin belirlenmesi), görselleştirilmesi ve belgelemesini sağlar.
- Büyük ve karmaşık sistemlerin modellenmesinde başarılıdır.
- En iyi mühendislik uygulamalarının bir koleksiyonunu içerir.
- Nesne yönelimli yazılım geliştirme ve yazılım geliştirme sürecinde önemlidir.
- Yazılım projelerinin tasarımı çoğunlukla grafiksel gösterimlerle gerçekleşir.
- UML kullanımı, proje ekiplerinin iletişim kurmasına, potansiyel tasarımları keşfetmesine ve yazılımın mimari tasarımını sağlar.

# UML Tarihçesi



Before 95' - Fragmentation ► 95' - Unification ► 98' - Standardization ► 99' - Industrialization

# UML Amacı

- ❑ Tüm nesne yönelimli yöntemler tarafından kullanılabilen standart bir gösterim sağlanarak önemli gösterimlerin en iyi unsurlarını entegre eder.
- ❑ Geniş bir uygulama yelpazesi için vardır.
- ❑ Farklı sistemlere çözüm yapıları sunar.
  - ❖ dağıtılmış sistemler, analiz, sistem tasarımı ve dağıtımında görsel çözümlenmeler sağlanır.

# UML Sürümleri

- ❑ 1996 yılında, Object Management Group (OMG) tarafından ilk çağrı yapıldı.
- ❑ Bu çağrı ortak bir Request for Proposal – Çağrılara Yanıt (RFP) yürütmek üzere güçlerinin birleştirilmesini öngörmekteydi.
- ❑ Rational firması, UML Ortakları (UML Partners) konsorsiyumunu kurdu ve UML 1.0 tanımı oluşturuldu.
- ❑ Ocak 1997’de revize edilmiş UML 1.1 üretildi.
  - ❖ Pek çok teknoloji şirketi birleşerek OMG'ye ayrı RFP yanıtları sundu.
- ❑ UML 1.1 sürümünün odak noktası, UML 1.0 semantiğinin netliğini artırmak ve yeni ortaklardan gelen katkıları dahil etmektir.
- ❑ UML 1.2 den UML 1.5 ‘e kadar geliştirilen sürüm günümüzde UML 2.1 den başlayarak günümüzde UML 2.5 sürümüne ulaşmıştır.

# UML 1 ve UML 2 farkları

## Geliştirme Süreçleri bakımından

□ UML 1 yazılım geliştirme süreçlerine daha az odaklanmışken, UML 2 yazılım geliştirme süreçlerini desteklemek için daha fazla araç ve yöntem sunar.

❖ Özellikle Agile ve diğer modern geliştirme yaklaşımlarına uyum sağlamak için geliştirilmiştir.

## Notasyonlar ve Standartlar

UML 1 de notasyonlar oldukça sınırlıdır; bazı durumlarda belirsizlikler içerebilir. UML 2 'de ise notasyonlar net ve tutarlıdır.

❖ UML 2 'de her bir diyagram için belirli kurallar ve standartlar geliştirilmiştir.

UML 1 deki standartların uygulanabilirliği ve esnekliği çok sınırlıydı; UML 2 ise endüstri standartlarına iyi uyum sağlar ve uygulanabilirliği daha fazla ve esnektir.

## Niçin UML?

- Kullanıcıların, anlamlı modeller geliştirebilmeleri ve paylaşabilmeleri için kullanıma hazır bir görsel modelleme dilidir.
- Temel kavramları genişletmede uzmanlaşma mekanizmaları sunar.
- Belirli programlama dillerinden ve geliştirme süreçlerinden bağımsızdır.
- Modelleme dilini anlamada biçimsel bir temel sağlar.
- Nesne yönelimli araçlar pazarının büyümesini teşvik eder.
- İşbirlikleri (collaborations), çerçeveler (frameworks), kalıplar (patterns) ve bileşenler(components) gibi üst düzey geliştirme kavramlarını destekler
- En iyi uygulamaları entegre eder.

# Yapısal Diyagramlar / Structural Diagrams

- ❑ Sistemin ve parçalarının farklı soyutlama ve uygulama seviyelerindeki statik yapısını ve bunların birbirleriyle ilişkilerini gösterir.
  - ❑ Yapısal diyagramdaki bir öge, sistemin anlamlı kavramlarını temsil eder
  - ❑ Yapısal diyagram aynı zamanda soyut, gerçek dünya ve uygulamaya ait kavramları da içerir.
  - ❑ Yapısal diyagramlar:
    - Class Diagram**
    - Component Diagram**
    - Deployment Diagram**
    - Object Diagram**
    - Package Diagram
    - Composite Structure Diagram
    - Profile Diagram
- olmak üzere 7 kategoride incelenir.

# Davranışsal Diyagramlar / Behaviour Diagrams

- ❑ Sistemdeki nesnelerin dinamik davranışını gösterir.
- ❑ Bu davranışlar, sistemde zaman içinde meydana gelen bir dizi değişikliktir.

Davranışsal Diyagramlar:

**Use Case Diagram**

Activity Diagram

**State Machine Diagram**

**Sequence Diagram**

Communication Diagram

Interaction Overview Diagram

Timing Diagram

olarak gruplanır.

# Sınıf Diyagramları

- ❑ Sınıf diyagramı, nesne yönelimli yöntemlerin kullandığı ortak modelleme tekniğidir.
- ❑ Sistemdeki nesne türlerini ve aralarındaki çeşitli statik ilişkileri tanımlar

## Sınıflar Arası İlişkiler

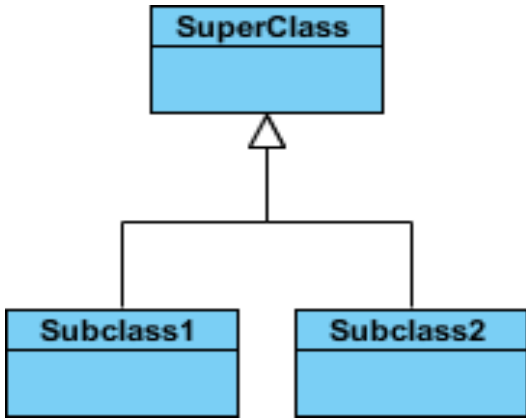
- ❑ İlişkilendirme (association) : Sınıfların örnekleri arasındaki ilişkileri betimler. Örneğin; bir kişi bir şirkette çalışır, bir şirketin birden fazla ofisi vardır).
- ❑ Kalıtım (inheritance) : ER diyagramlarının nesneye yönelik tasarımda kullanılmasıdır.
- ❑ Birleştirme (aggregation) Nesne yönelimli tasarımda bir nesne bileşimi biçimidir.

# Sınıf nedir?

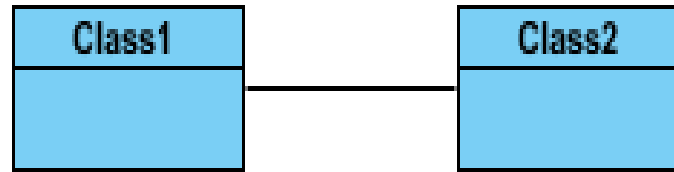
MyClass
+attribute1 : int
-attribute2 : float
#attribute3 : Circle
+op1(in p1 : bool, in p2) : String
-op2(input p3 : int) : float
#op3(out p6) : Class6*

- ❑ Sistemde benzer rollere, işlevlere sahip nesne grubudur.
- ❑ Her sınıf, yapısal özellikler (öznitelikler-attributes) ve davranışsal özellikler (behaviors) içerir.
- ❑ Yapısal özellikler, sınıfın nesnelerinin ne bildiğini tanımlar; sınıfın bir nesnesinin durumunu temsil eder. Sınıfın yapısal, yani statik özelliklerinin açıklamasıdır.
- ❑ Davranışsal özellikler (işlemler/ operasyonlar) sınıfın nesnelerinin neler yapabileceğini tanımlar.
  - ❖ Nesnelerin nasıl etkileşime girebileceğini tanımlar. Operasyonlar, bir sınıfın davranışsal veya dinamik özelliklerinin açıklamalarıdır.

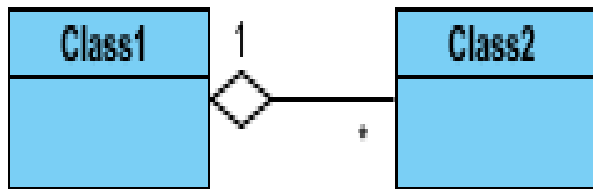
# Sınıf İlişkileri



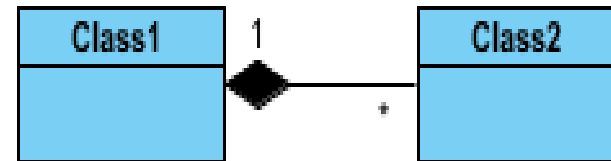
Inheritance / Generalization)



Simple Association



Aggregation



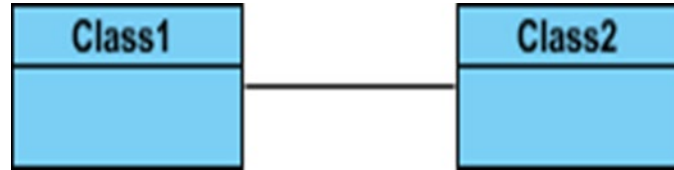
Composition

# Simple Association - Basit İlişki

İki eşdeğer sınıf arasında yapısal bir bağlantı oluşturur.

Class1 ve Class 2 arasında bir ilişki vardır.

İki sınıfı birbirine bağlayan düz bir çizgidir.



# Aggregation İlişkisi

Özel bir ilişki türü olarak bir parçası (a part of) ilişkisi temsil edilir.

Class2, Class1'in bir parçasıdır.

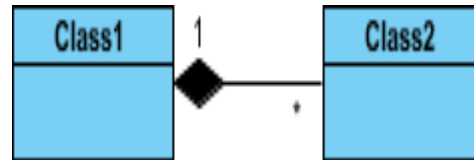
Class2'nin birçok örneği (\* ile gösterilir) Class1 ile ilişkilendirilebilir.

Class1 ve Class2 nesnelerinin ayrı ömürleri vardır.

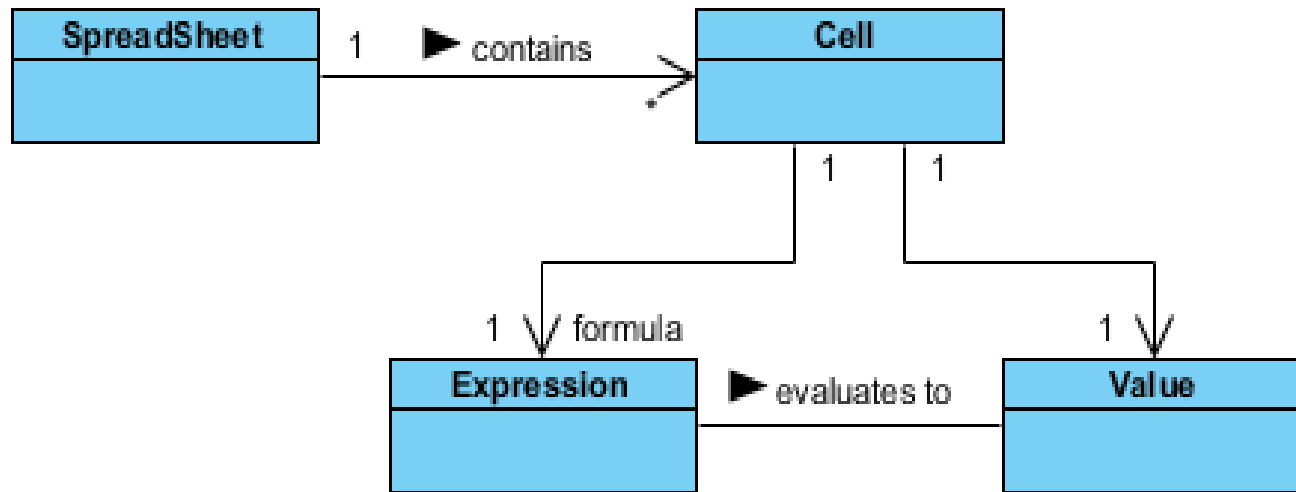
İlişki, ucunda içi boş bir diamond olan çizgi ile betimlenir, bileşik (composite) sınıfa bağlıdır.

# Composition İlişkisi

- ❑ Bütünün yok olmasıyla parçaların da yok olduğu özel birleştirme türüdür.
- ❑ Class2 nesnelere Class1 ile birlikte yaşar ve ölür.
- ❑ Class2 kendi başına var olamaz.
- ❑ Composition sınıfına bağlı olan ilişki noktasında içi dolu bir diamond olan çizgi ile betimlenir.



# İlişkilerin İsimlendirilmesi



# Sınıf Özniteliklerinin ve Operasyonlarının Görünürlüğü

Access Right	public (+)	private (-)	protected (#)	Package (~)
Members of the same class	yes	yes	yes	yes
Members of derived classes	yes	no	yes	yes
Members of any other class	yes	no	no	in same package

# Değişken (attribute) ve Metot Özellikleri

- + attributes ya da operations için **public** olduğu belirtilir.
- attributes ya da operations için **private** olduğu belirtilir.
- # attributes or operations için **protected** olduğu belirtilir.
- ~ attributes ya da operations için package ifade eder.

public (+) imi , sistemdeki tüm sınıflarda kullanılabilecek değişken ve metotları belirtir.

private (-) imi, sadece yer aldığı sınıfta kullanılabilen değişken ve metotları belirtir. private olan metotlar ilgili sınıfta kapsülleme (encapsulation) özelliğini tanımlar.

# Multiplicity – Çokluk Örneği

İlişkilerde ve çoklukta her sınıftan kaç nesne olduğu belirtilir.

Exactly one : 1

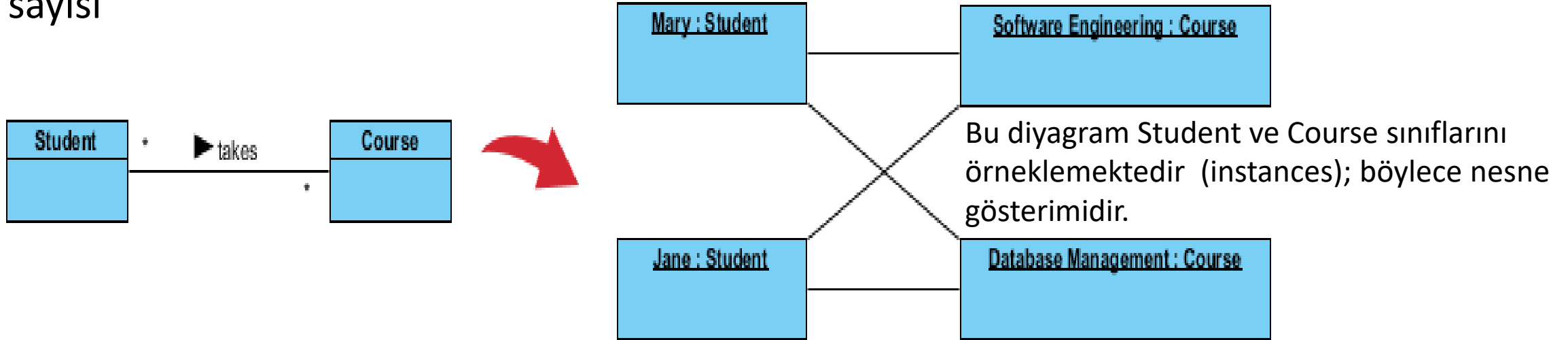
Zero or one : 0..1

Many : 0..\* ya da \*

One or more : 1..\*

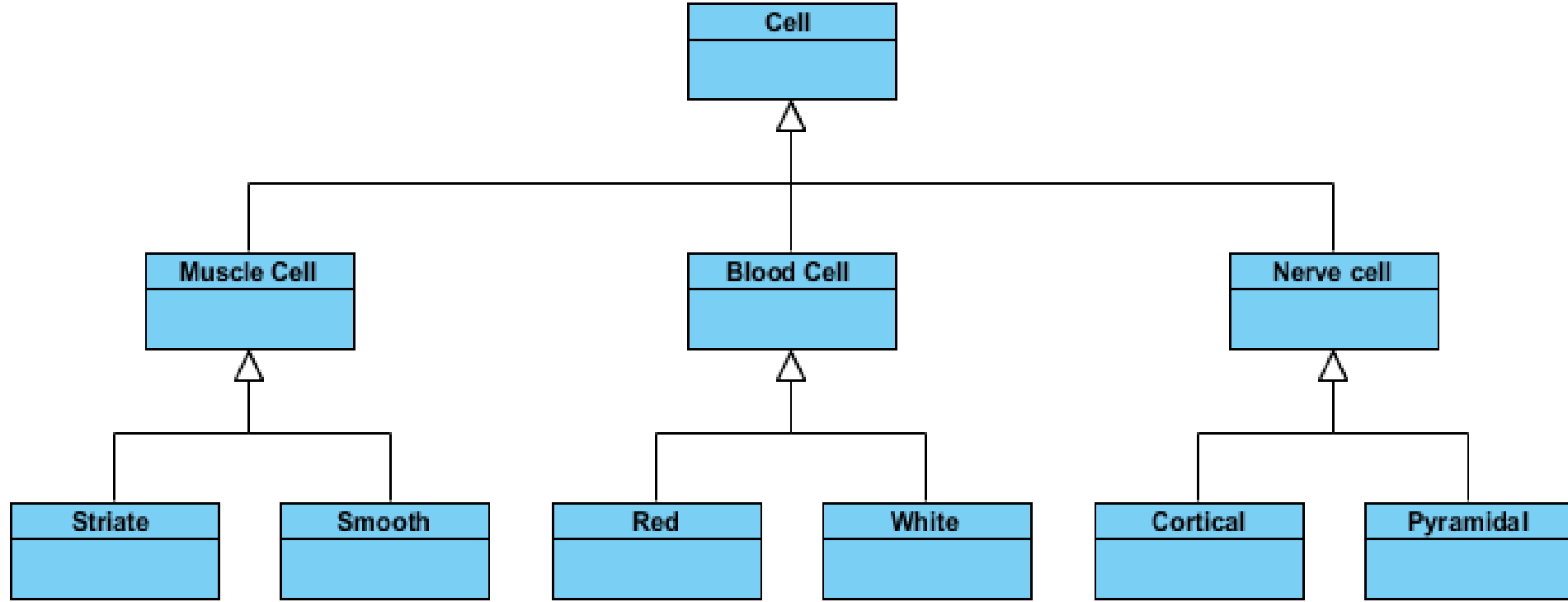
Exact Number : 3..4 ya da 6

a complex relationship : 0..1, 3..4, 6.\* 2 ya da 5 dışında nesnelerin (objects) herhangi bir sayısı



Mary, Jane Student sınıfının nesneleri, Software Engineering ve Database Management Course sınıfının nesnelere (objects)

# Kalıtım –Inheritance

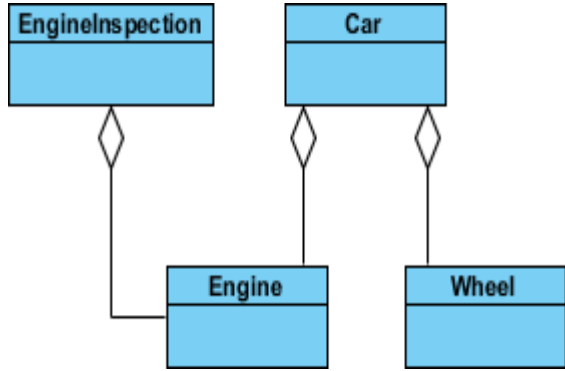


- ❑ Kalıtım, bir tür hiyerarşiyi ifade eden bir ilişkinin özel bir durumudur.
- ❑ Kalıtım, bir taksonomi getirerek analiz modelini basitleştirir.
- ❑ Alt sınıflar, üst sınıfın niteliklerini ve işlemlerini miras alır.

# Birleřtirme –Aggregation

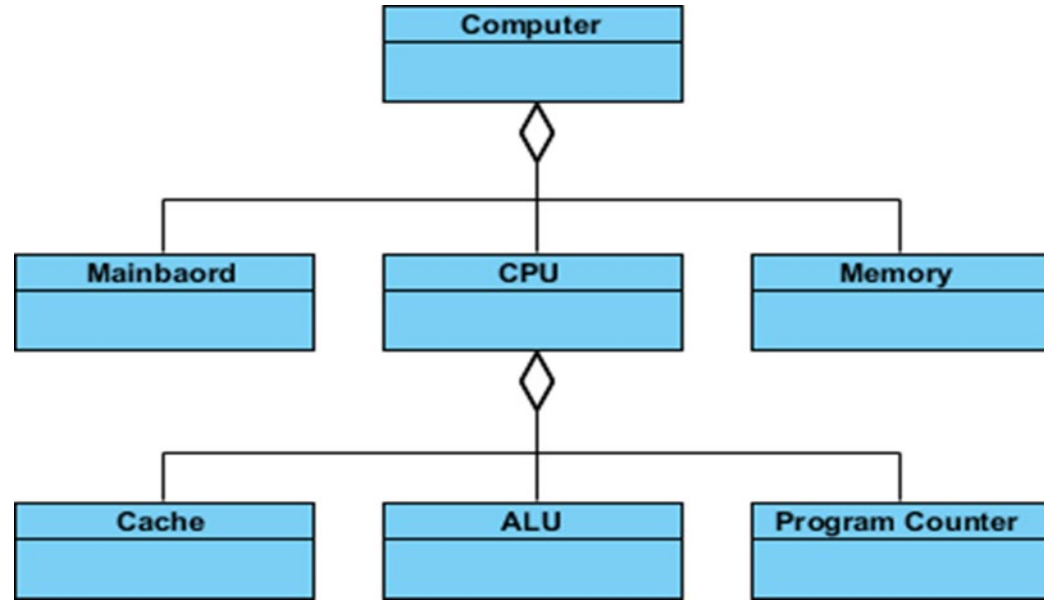
□ Birleřtirme, part of (parçasıdır) hiyerarşisini ifade eden özel bir ilişkilendirmedir.

□ Birleřtirme (aggregation) üst sınıftır; bileşenler (component) ise alt sınıflardır.



Engine sınıfı Car sınıfının bir parçasıdır (part of). Wheel sınıfı Car sınıfının bir parçasıdır (part of).

Bu da Car sınıfı yoksa hem Engine sınıfı hem de Wheel sınıfı varlığını sürdürür. Çünkü Her iki sınıfta ayrı ayrı farklı metotlar vardır. Car sınıfı olmadığında Engine ve Wheel sınıfı kendi metotlarıyla çalışır (run).

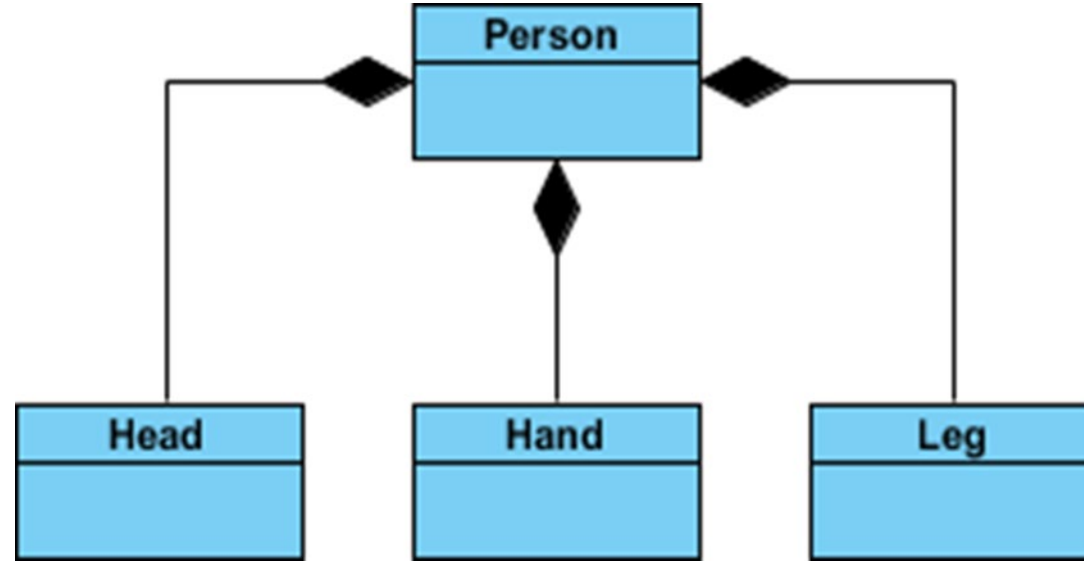


Mainboard, CPU ve Memory sınıfları Computer sınıfının sadece **bir parçasıdır**.

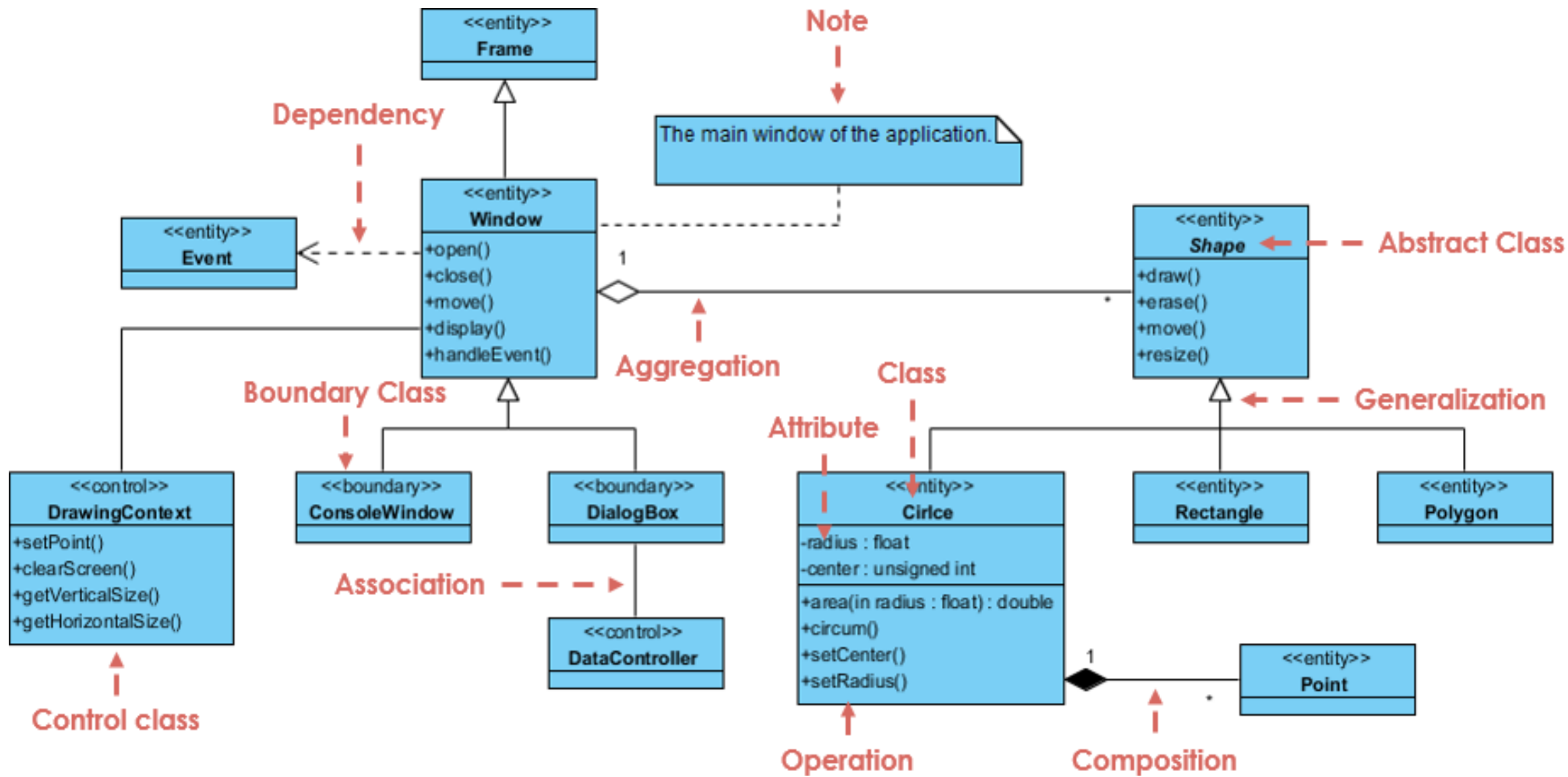
Computer sınıfı sadece alttaki sınıflardan oluşmaz. Yani Computer sınıfı çocuk

olan 3 sınıftan oluşmaz; başka bileşenleri de vardır. O nedenle sadece **part of** ilişkisi tanımlanmıştır.

# whole-part ilişkisi : Composition



- ❑ Bu ilişki aggregation ilişkisinin özel halidir.
- ❑ Person sınıfı ile Head sınıfı arasındaki parça (part of) ilişkisine ek olarak, çok daha güçlü bir bağımlılık vardır.
- ❑ Bu bağımlılık Person Sınıfı silindiğinde Head, Hand ve Leg sınıflarının silinmesidir.
- ❑ Kod olarak tasarlandığında Head, Hand ve Leg sınıflarının tüm metotlarının parametre olarak Person sınıfı tipinde bir değişken içermesidir ya da metotların dönen değerlerinin Person sınıfı tipinde olmasıdır.



- ❑ Shape soyut bir sınıftır; yani örneklenmez ve bir üst sınıftır.
- ❑ Circle, Rectangle ve Polygon, Shape'ten türetilmiştir.
  - ❖ Bu bir genelleştirme/kalıtım ilişkisidir.
- ❑ DialogBox ve DataController arasında basit bir ilişki (association) vardır.
- ❑ Shape, Window sınıfının bir parçasıdır. Bu bir aggregation ilişkisidir. Shape, Window olmadan da var olabilir.
- ❑ Point, Circle'ın bir parçasıdır. Bu bir composition ilişkisidir. Point, Circle olmadan var olamaz.
- ❑ Window, Event sınıfına bağımlıdır (dependency). Ancak Event, Window'a bağımlı değildir.
- ❑ Circle sınıfının öznitelikleri radius ve center değişkenleridir. Circle sınıfı metotları area(), circum(), setCenter() ve setRadius()'tur.
- ❑ Circle sınıfındaki radius parametresi float türünde bir giriş parametresidir.
- ❑ Circle sınıfının area() metodu double türünde bir değer döndürür.
- ❑ Rectangle ve polygon sınıfının öznitelikleri (attributes) ve metot adları gizlidir (hidden).  
Diyagramdaki diğer bazı sınıfların da öznitelikleri ve metot adları gizlidir (hidden)