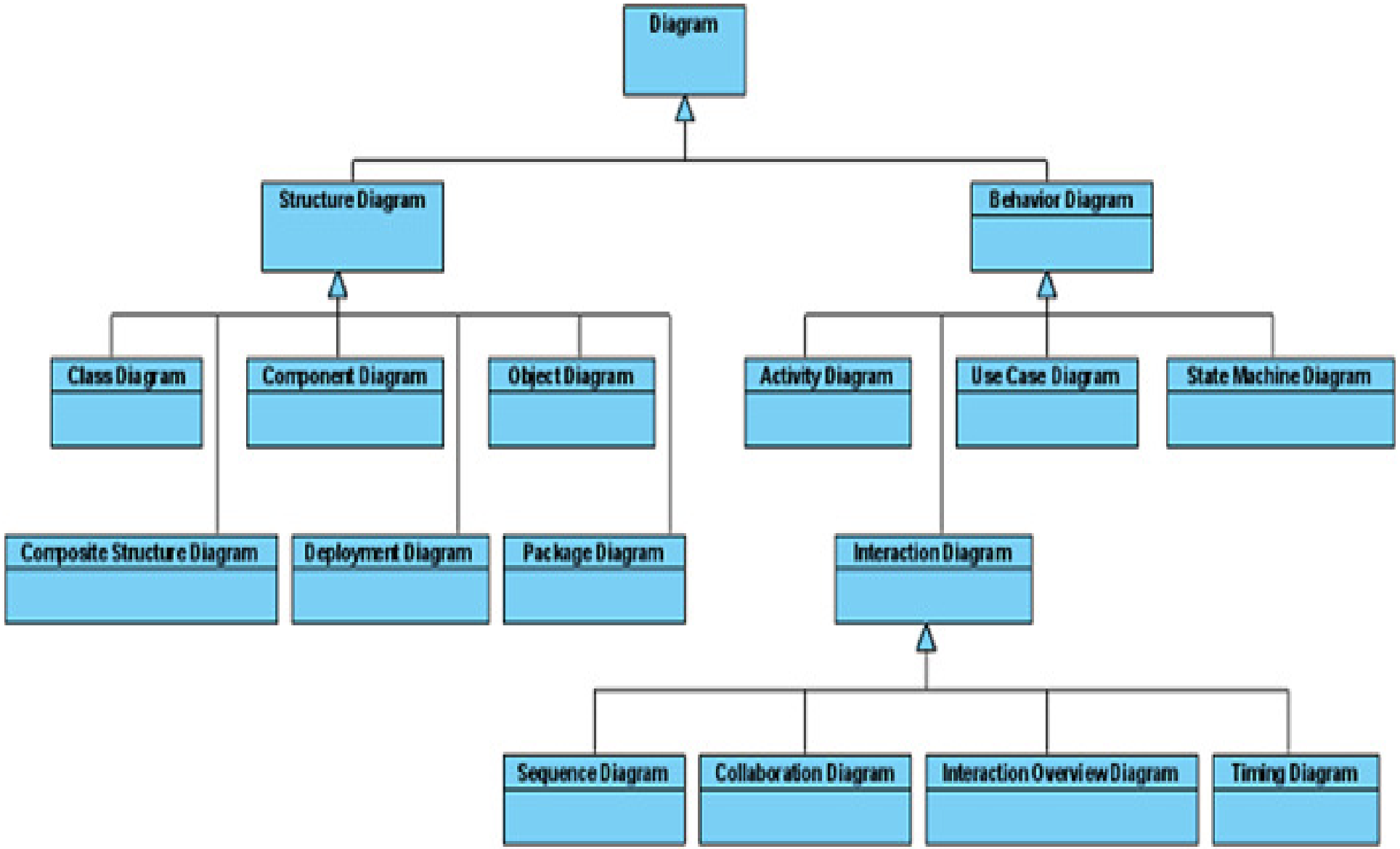
Yazılım Gereksinimlerinin Görsel Çözümlemeleri:

UML

(Unified Modeling Language)

Birleştirilmiş Modelleme Dili

UML Diyagramlarının Sınıflandırması



Davranışsal Diyagramlar II

(Behavioural Diagrams)

«Sequence» Diyagramı

Sequence Diyagramı

- «Sequence» diyagramı bir nesneden (object) diğer nesneye aktarılan mesajları gösterir.
- Bu akışlar fonksiyon çağrıları şeklinde gerçekleştir.
- Böylece «sequence diyagramı», belli bir işlevi (fonksiyonelliği gerçekleştirmek üzere) sistem tarafından gerçekleştirilen bir dizi çağrıdan meydana gelir.

Nesneler, Ömür Çizgisi, Mesajlar
(Objects, Lifelines, Messages)

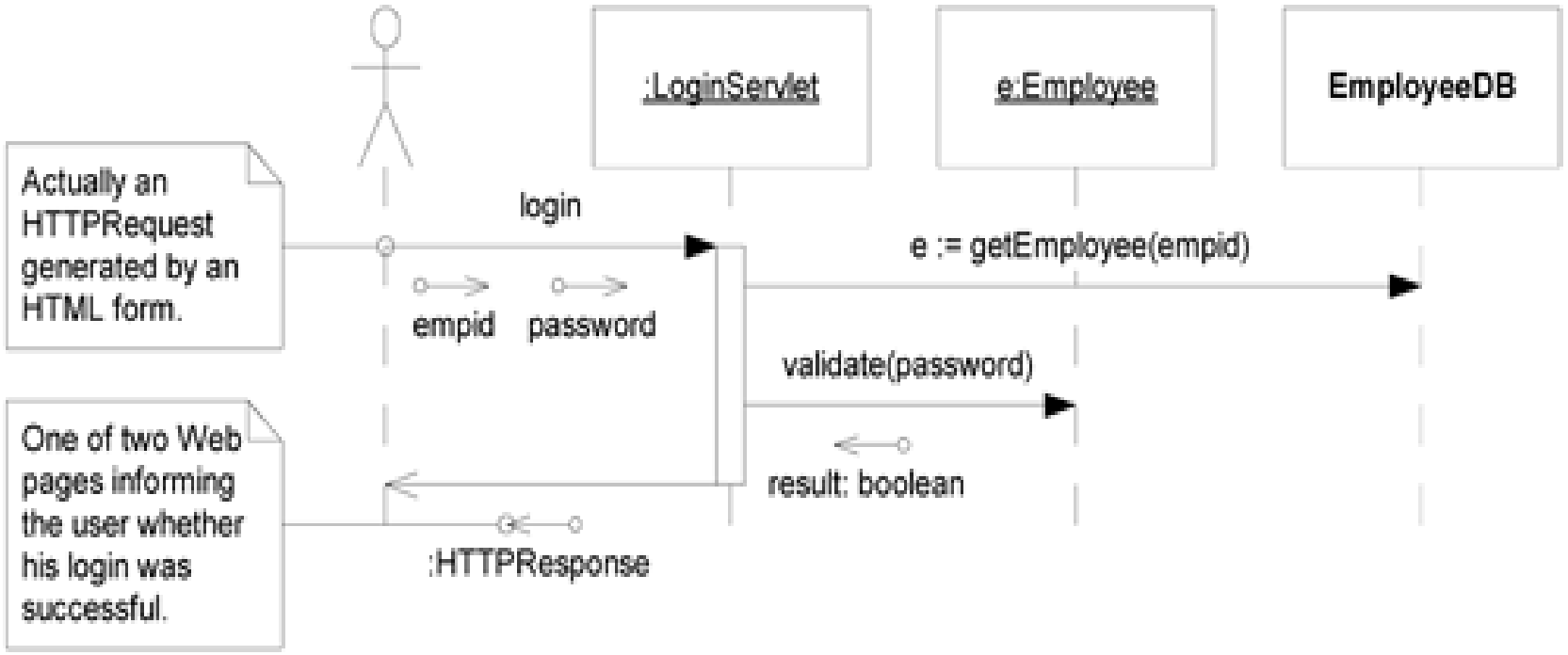
- Nesneler ve sınıflar diyagramın üstünde yatay eksende yer alırlar.
- Nesneleri sınıflardan ayırmanın bir yolu nesnelerin altında çizgi olmasıdır.
- Aktör diyagramın en solunda bulunuyorsa, isimsiz bir nesneyi simgeler.
  - Bu aktör kaynağı (source) temsil eder ve sisteme giren /çıkan tüm mesajları biriktirir.
  - Tüm «sequence» diyagramlarında isimsiz aktör olmak zorunda değildir.

Nesneler, Ömür çizgisi, Mesajlar

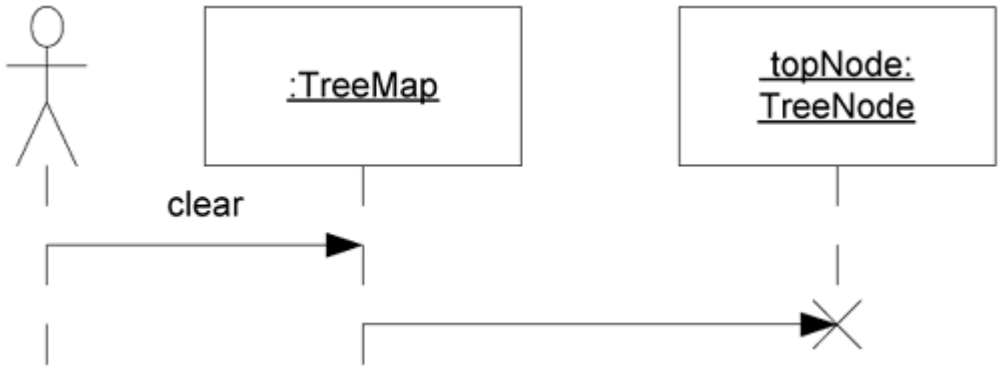
(Objects, Lifelines, Messages)

- Nesnelerden aşağıya kesikli çizgiler (dashed lines) aktörlerin aktif olama süresini, yani ömrünü (lifelines) betimler.
- Bir nesneden diğerine gönderilen mesaj ok ile gösterilir.
  - Yani mesaj, iki ömür çizgisi (lifeline) arasındaki ok olarak resmedilir.
  - Her mesajın bir ismi vardır, yani bir isim ile etiketlenmiştir.
  - Argümanlar, yani parametreler parantez içerisinde görüntülenirler.
- Zaman (Time) dikey boyutta simgelenir.
  - Daha düşük düzeyde olarak bir mesaj gönderildikten sonra görünecektir.

Örnek 1
Sequence Diyagramı

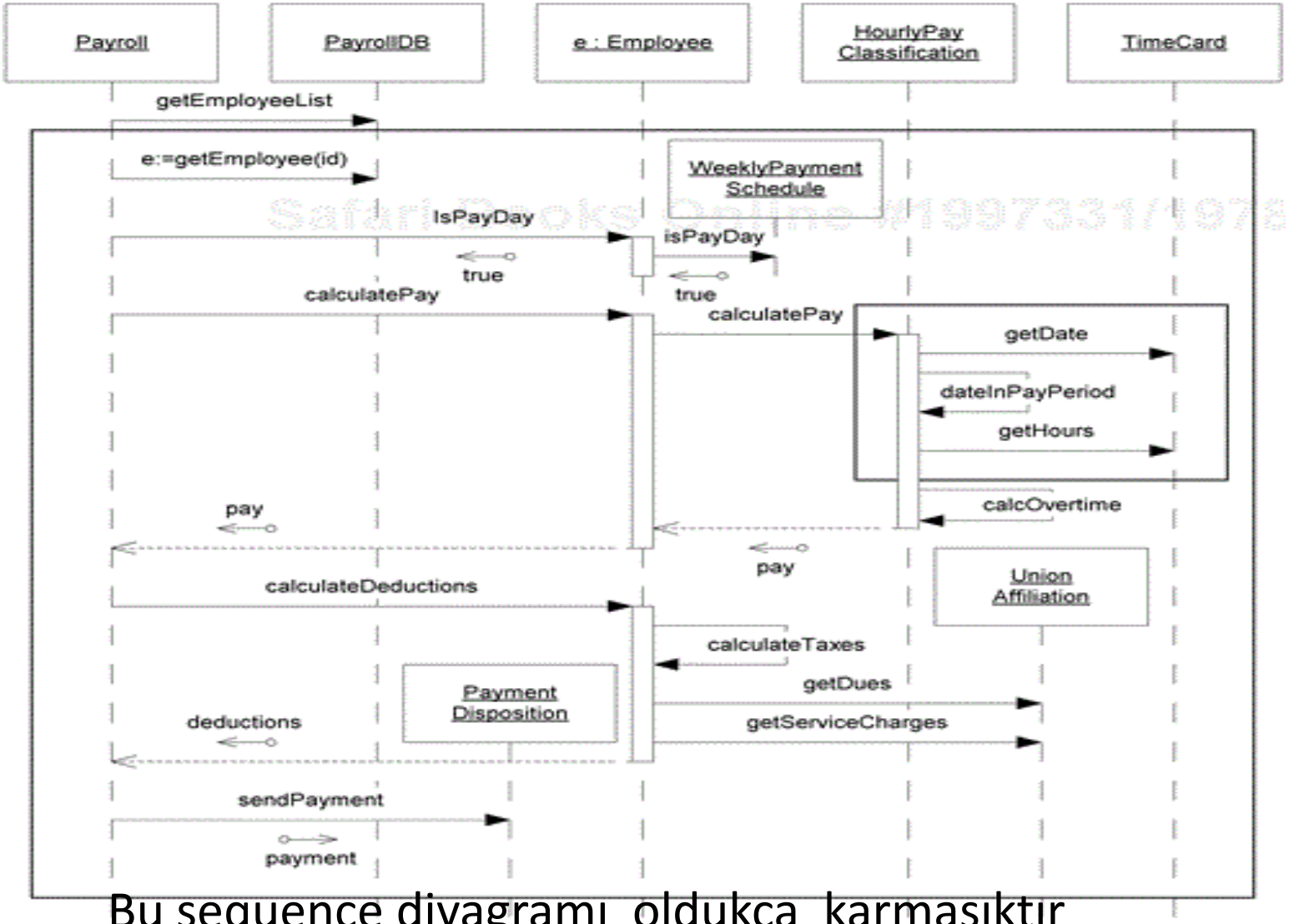


Örnek 2
Sequence Diyagramı



Bu diyagram bir nesneyi çöp kutusuna gönderir.
(Releasing an object to the garbage collector)

Örnek 3
Farklı Durumlar ve Senaryolar

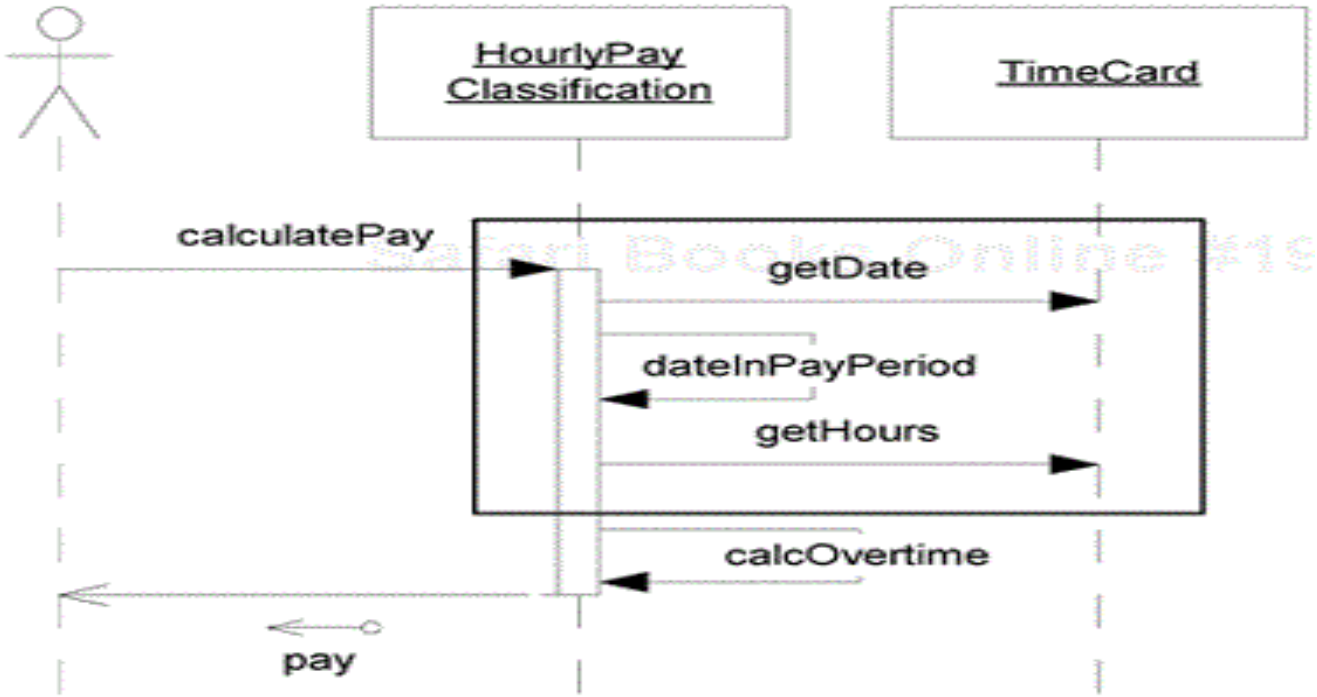


Bu sequence diyagramı oldukça karmaşıktır

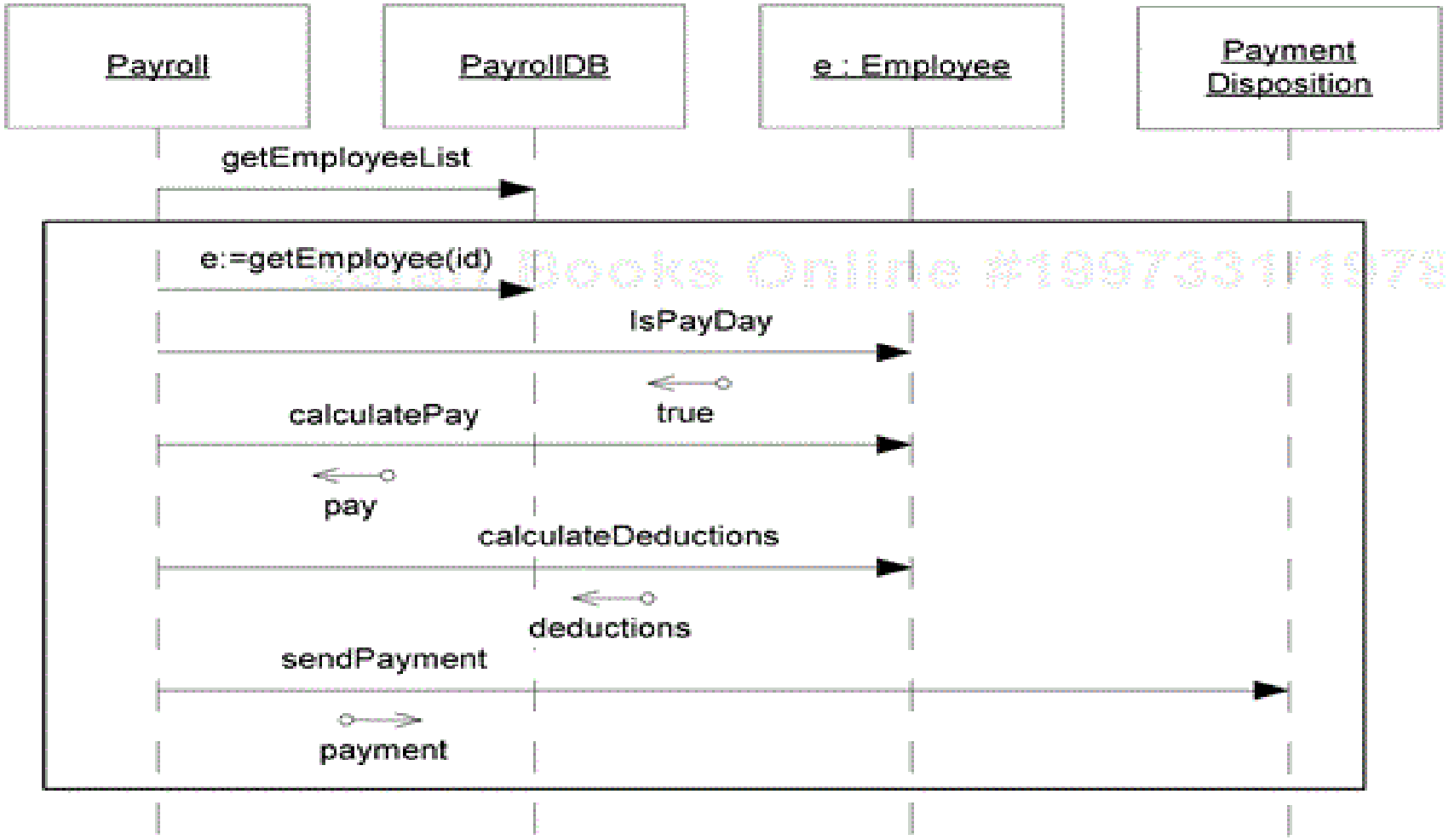
«Sequence» Diyagramı Tasarlama Kuralları

- sequence diyagramları kesinlikle karmaşık tasarlanmamalıdır
  - Karmaşık bir diyagramı kimse okuyamaz .
  - Karmaşık bir diyagramı okuyabilecek kişi olsa bile karmaşıklığından dolayı okumayacaktır.
  - Bu da gereksiz bir zaman kaybıdır.
- O nedenle daha küçük «sequence» diyagramının nasıl tasarlanacağı öğrenilmelidir.
- Her «sequence» diyagramı tek bir sayfada tasarlanmış olmalıdır

Örnek 3
parçalanmış sequence diyagramı



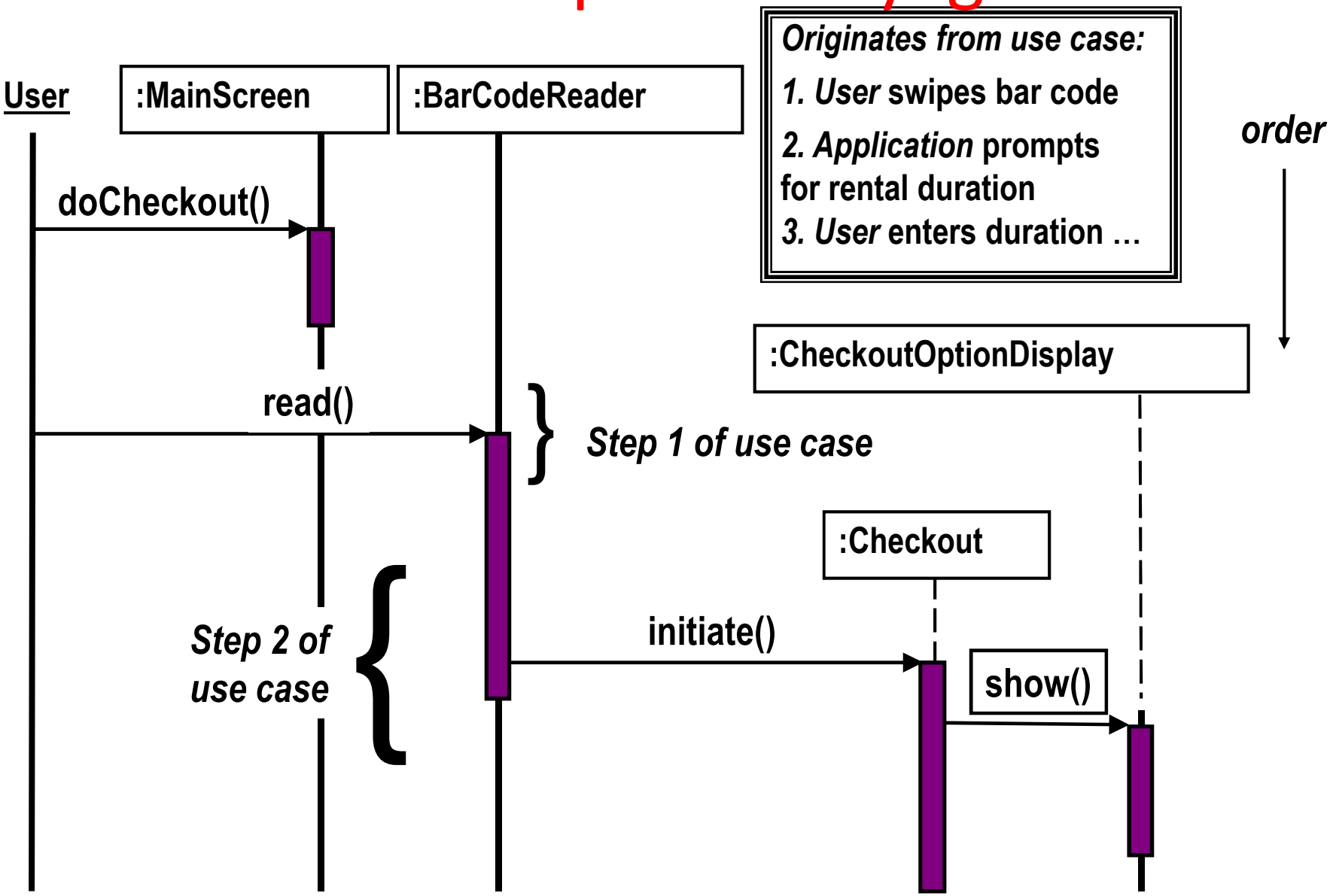
Örnek 3 Sequence Diyagramı Farklı Senaryo



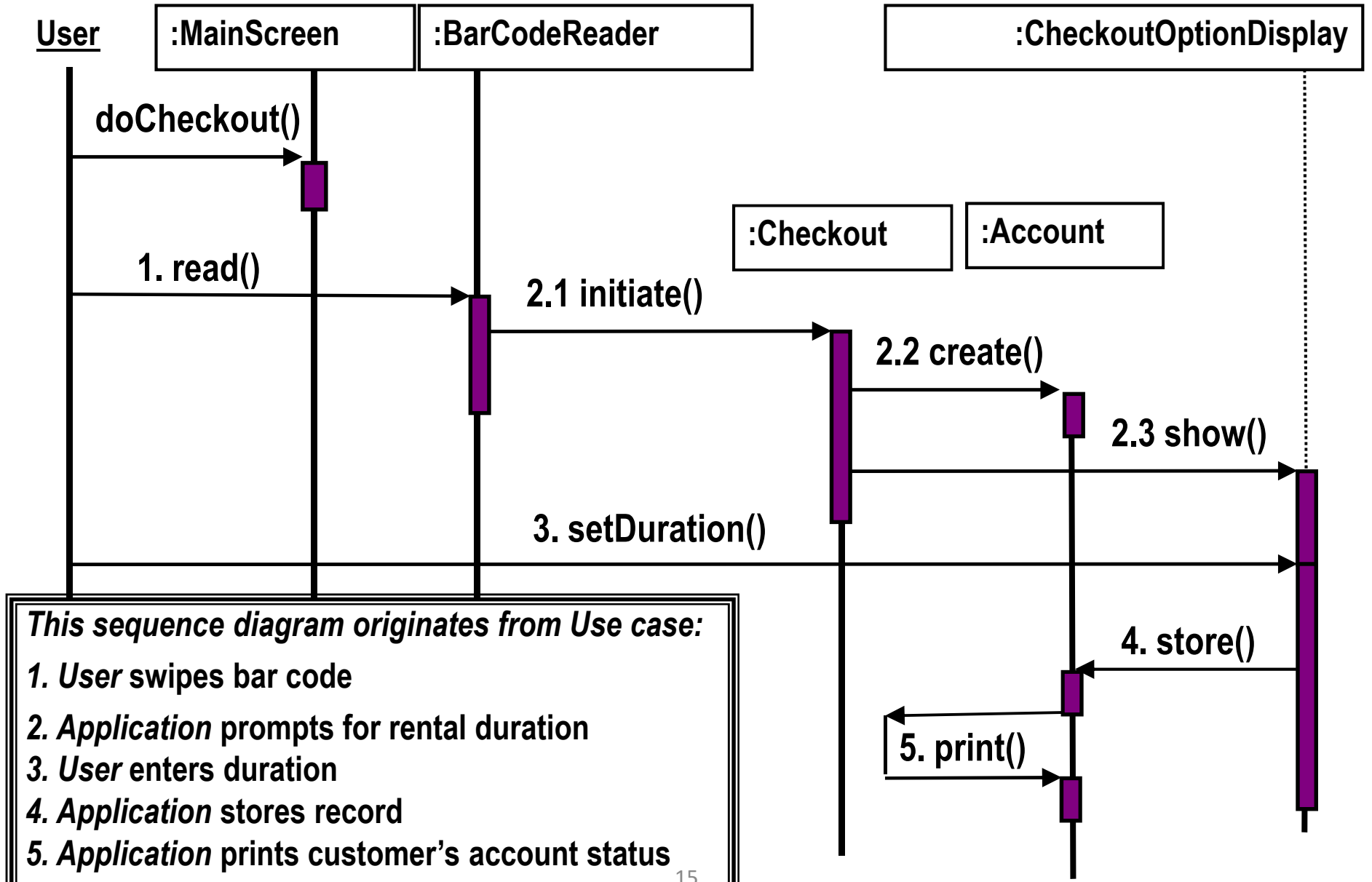
Örnek 4
Sequence Diyagramı

- ***Use case:***
- ***1. User* swipes bar code**
- ***2. Application* prompts for rental duration**
- ***3. User* enters duration**
- ***4. Application* stores record**
- ***5. Application* prints customer's account status**

Örnek 4 Sequence Diyagram



Örnek 4 Sequence Diyagram



UML ile

Statik

Yapısal (Structural) Modelleme

Sınıf (class) diyagramları

Sınıf Diyagramı (Class Diagram)

- Sınıf diyagramları UML diyagramları içerisinde en yaygın kullanılan diyagramlardır ve geliştirilecek sistemin oluşturulmasında çok önemlidir.
- Sınıf diyagramları class, interface, association ve collaboration bildirimlerinden oluşur.
- Sınıf diyagramları bir sistemin nesneye yönelik tasarımının (object oriented view of a system) statik olarak tasarlanmasıdır.
  - Ürünün geliştirilmesi aşamasında tasarımı oluşturur.

Nesne Diyagramı (Object Diagram)

- Nesne diyagramları (Object diagrams) sınıf diyagramının bir örneğidir (instance of class diagram).
  - Bu diyagramlar problemin implementasyonunda gerçek dünya senaryolarını örnekler.
- Nesne diyagramları da aynı sınıf diyagramında olduğu gibi bir dizi nesne ve bu nesneler arasındaki ilişkilerden oluşur.

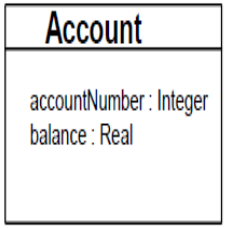
Nesne & Sınıf Tasarım Kavramları

- Nesneler gerçek dünyadaki şeyleri (things) simgeler.
  - Gerçek dünyanın anlaşılabilirliğini sağlar.
  - Problemin bilgisayar çözümünün temelini oluşturur.
- Bir nesne Object (object instance) tek bir «şeydir.»

  birHesap, birCalışan gibi
- Bir nesne sınıfı (Object Class) aynı özelliklere sahip nesnelerin bir koleksiyonudur (collection of objects).
- Öznitelik (Attribute) o nesnenin sahip olduğu veri değeridir. (Data value)

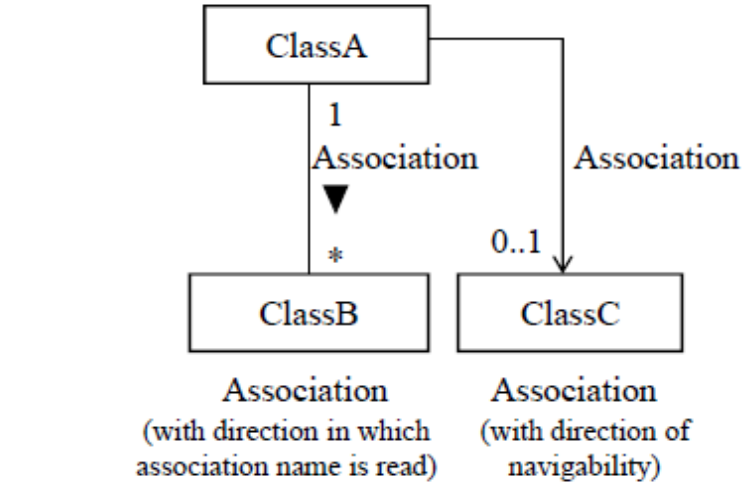
  «account number», «balance» gibi

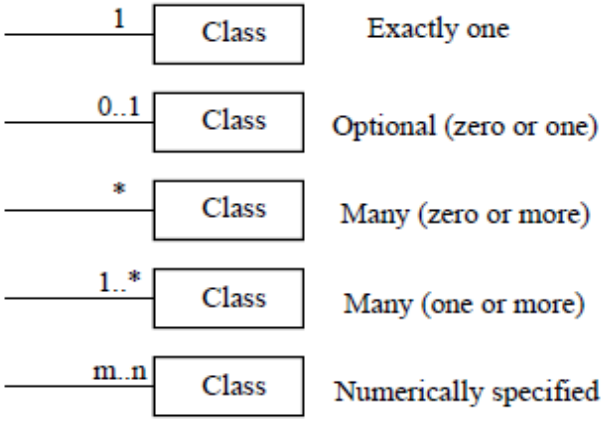
Class with attributes

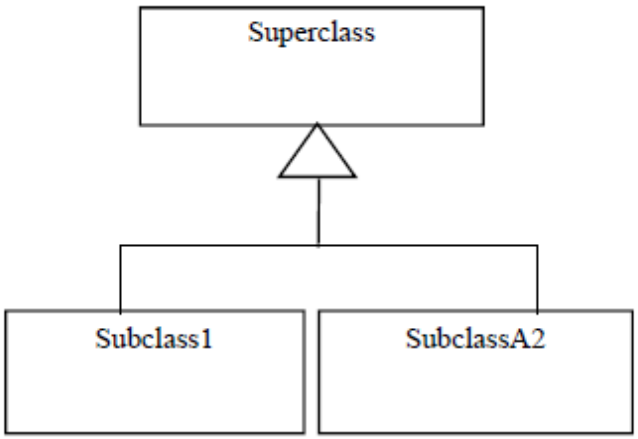


Objects with values

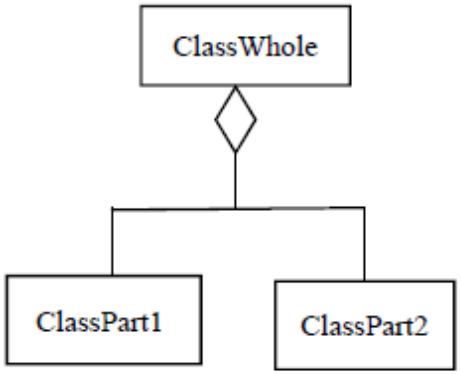
Sınıf Diyagramlarının UML Gösterimi



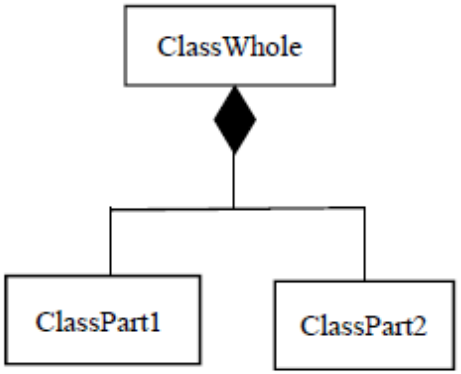




Generalization/specialization

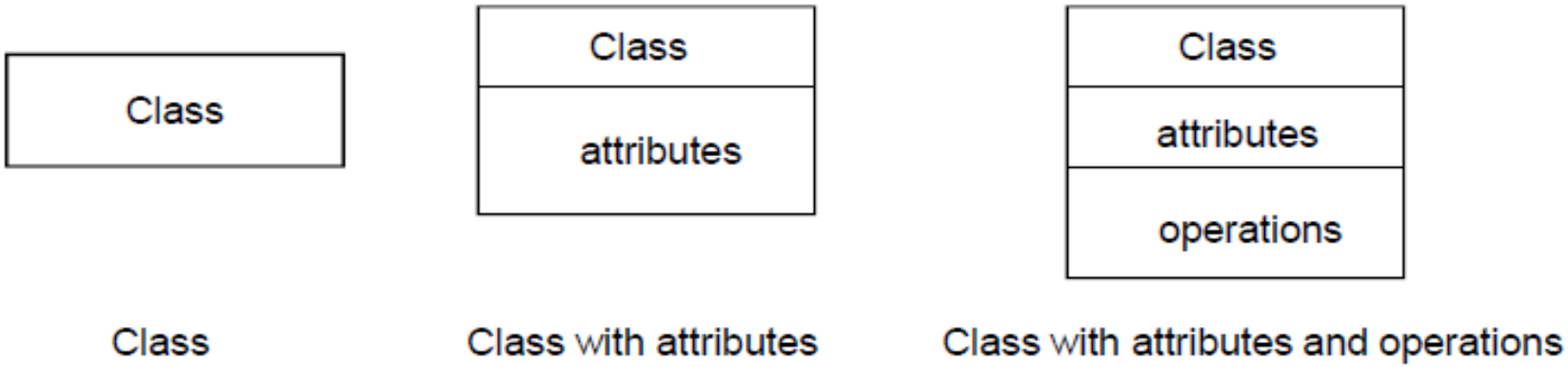


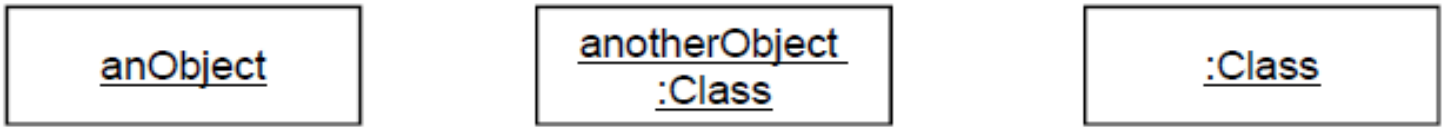
Aggregation hierarchy



Composition hierarchy

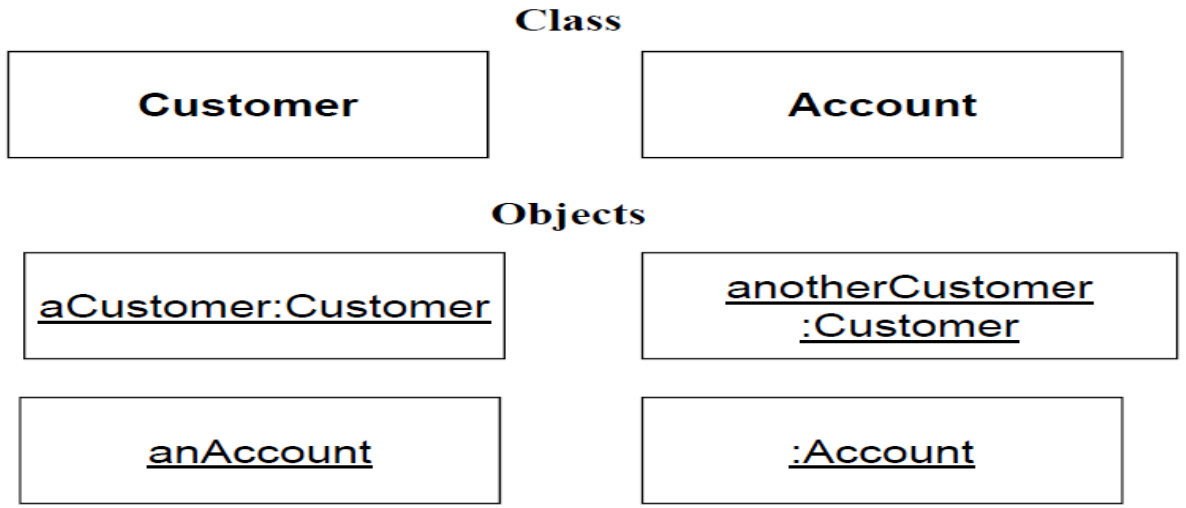
Sınıf ve Nesnelerin UML Gösterimi

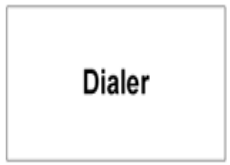




Objects

Sınıf (class) ve Nesne (Objects) Örnekleri






```
public class Dialer
{
}
```

Genel sınıf tanımı

Sınıfın öznitelik (değişkenleri) ve metotları (fonksiyonları) ile bildirimi

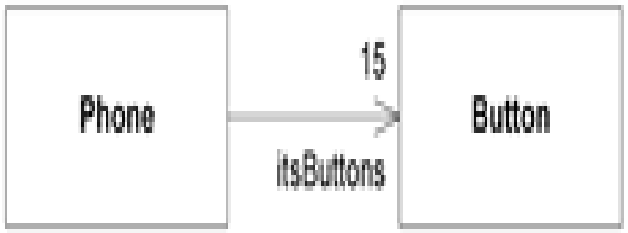

```
public class Dialer
{
  private Vector digits;
  int nDigits;
  public void digit(int n);
  protected boolean recordDigit(int n);
}
```

Sınıflar arası Birliktelikler (Associations)

- Sınıflar arasındaki yapısal ilişkiler tanımlanır (structural relationships)
- Sınıflar arası ilişkiler
  - Associations
  - Composition / Aggregation
  - Generalization / Specialization
- Analiz sırasında statik modelleme
  - Sistem «context» (içerik) sınıf diyagramları (System Context Class Diagram)
    - ✓Dışsal (harici) sınıflar ve sistem sınırlarını gösterir (Depict external classes and system boundary)
  - Varlık «Entity» sınıflarının statik modellemesi (Static Modeling of Entity classes)
    - ✓Verilerin depolandığı sınıflar

Association I

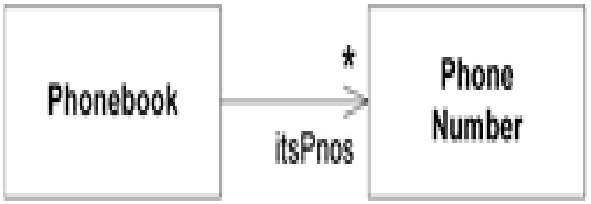
- Sınıflar arası birlikteliklerde (associations) çoğunlukla bir sınıfın örnekleri diğer nesneleri referans olarak alır.
- Phone ve Button sınıfları arasındaki birliktelikte okun yönü Phone sınıfının Button sınıfına bir referans gerçekleştirdiği görülmektedir.
  - Phone sınıfının bir değişeninin tipi diğer sınıftır
- Okun yanındaki isim sınıfın örnek değişkenidir (instance variable); sayı ise kaç tane referans tutulacağını ifade eder (15 elemanlı dizi anlamındadır).




```
public class Phone
{
  private Button itsButtons[15];
}
```

Association II

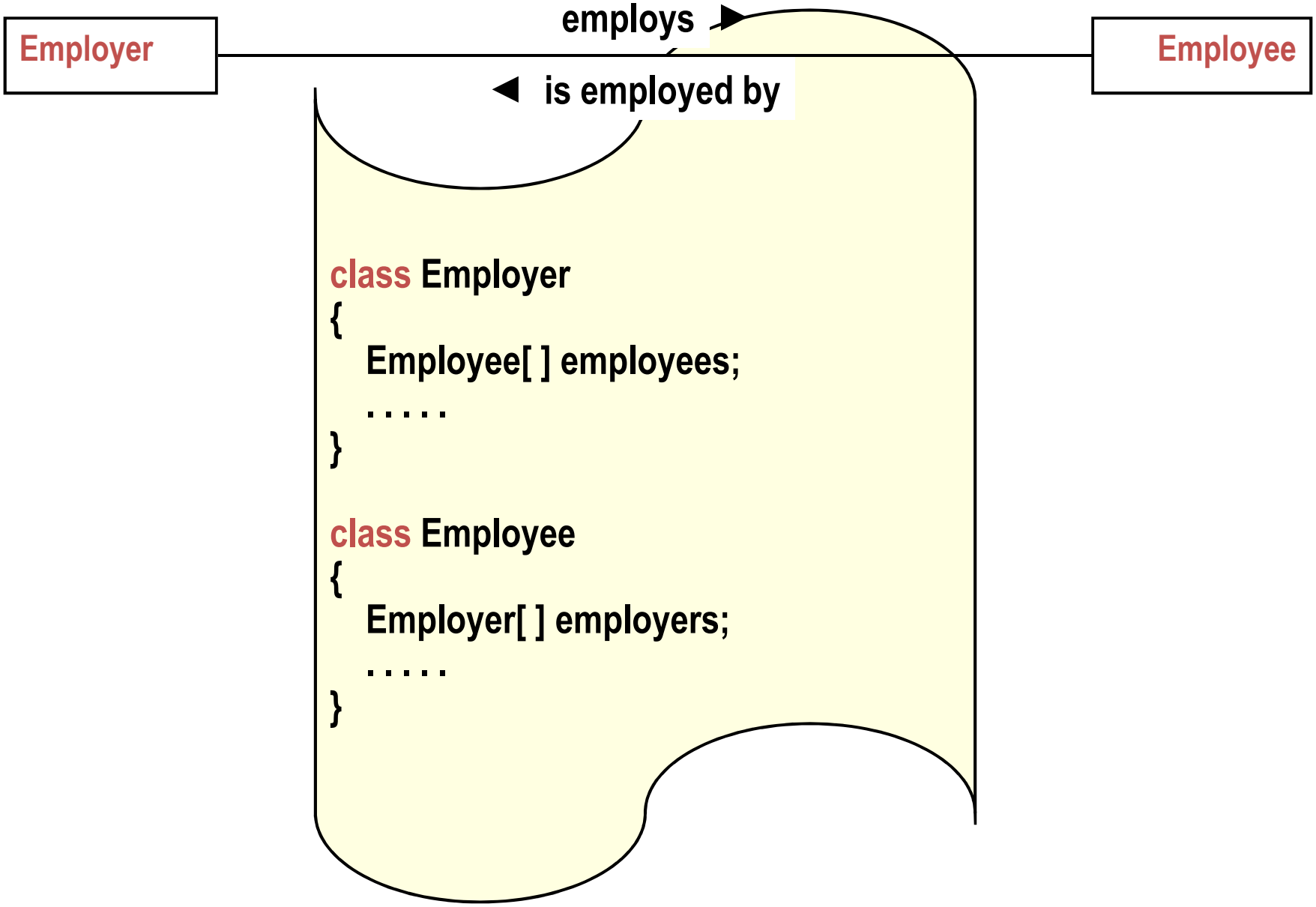
- Bir telefon rehberinde sıfır veya daha fazla sayıda (*) telefon numarası bulunur.
- Nesneye yönelik tasarımda bu ilişki HASA (has a) ve ISA (is a)ilişkisi şekline ifade edilir.



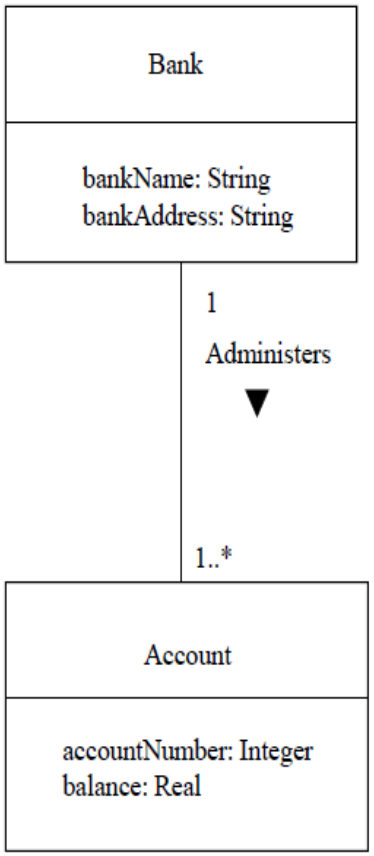

```
public class Phonebook
{
  private Vector itsPnos;
}
```

PhoneNumber

Association : UML Gösterimi



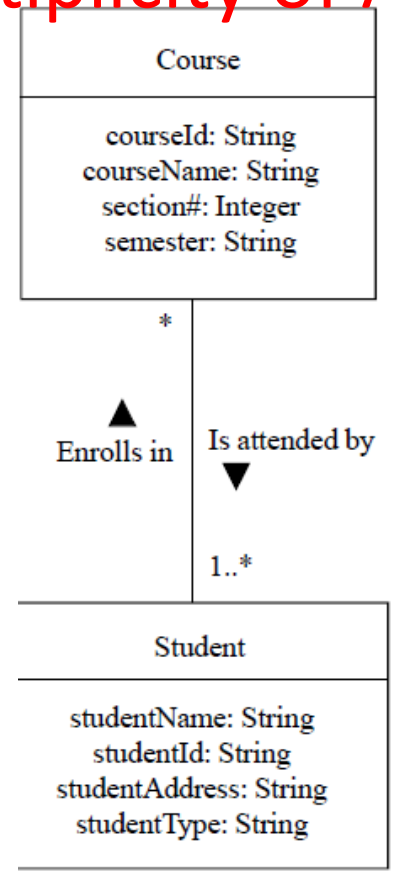
Birlikteliklerde Çokluklar (Multiplicity of Associations)



One -to-many association (1..*)

Her bankada *1 veya daha fazla hesap bulunur.*

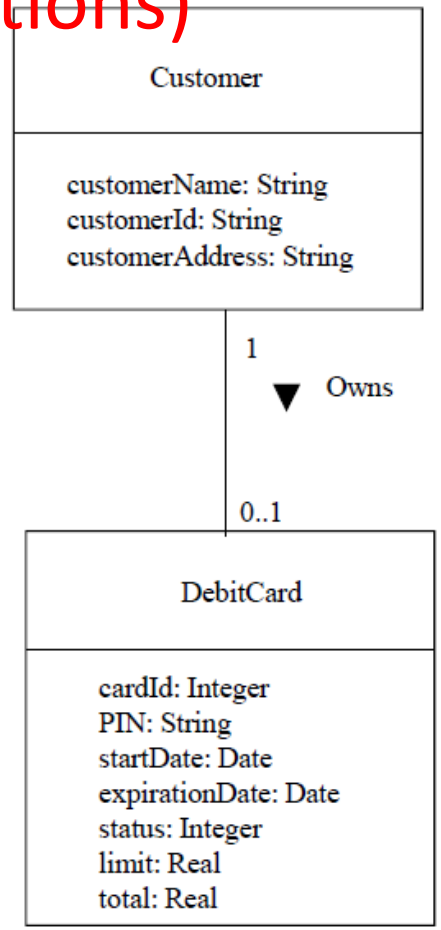
Her hesap *bir bankaya aittir.*



Many-to-Many association

Her ders *1 veya daha fazla sayıda (1..*)öğrenci tarafından alınır.*

Her öğrenci sıfır veya daha fazla (*) sayıda derse kayıt olur.

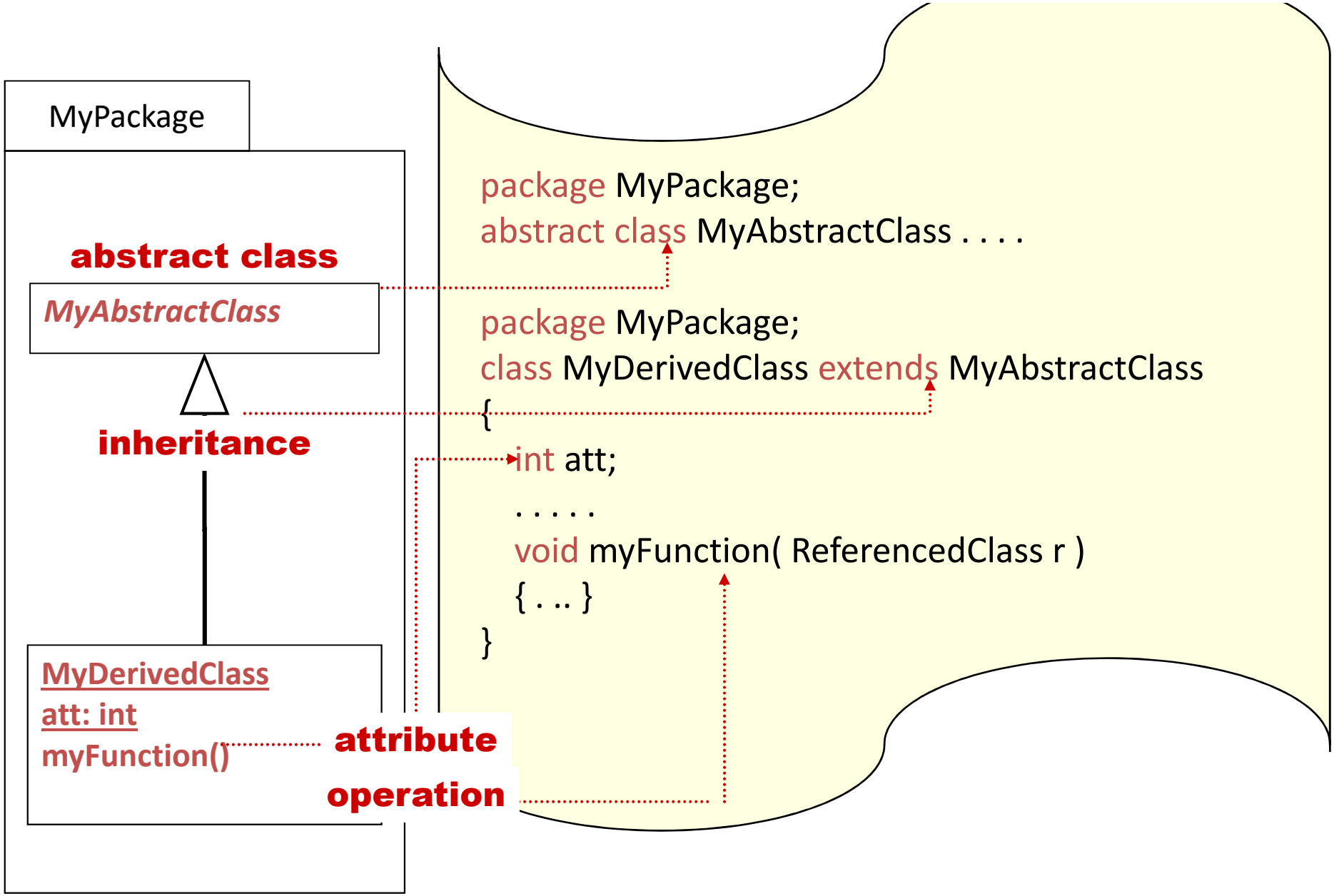


Optional association (0 ya da 1)

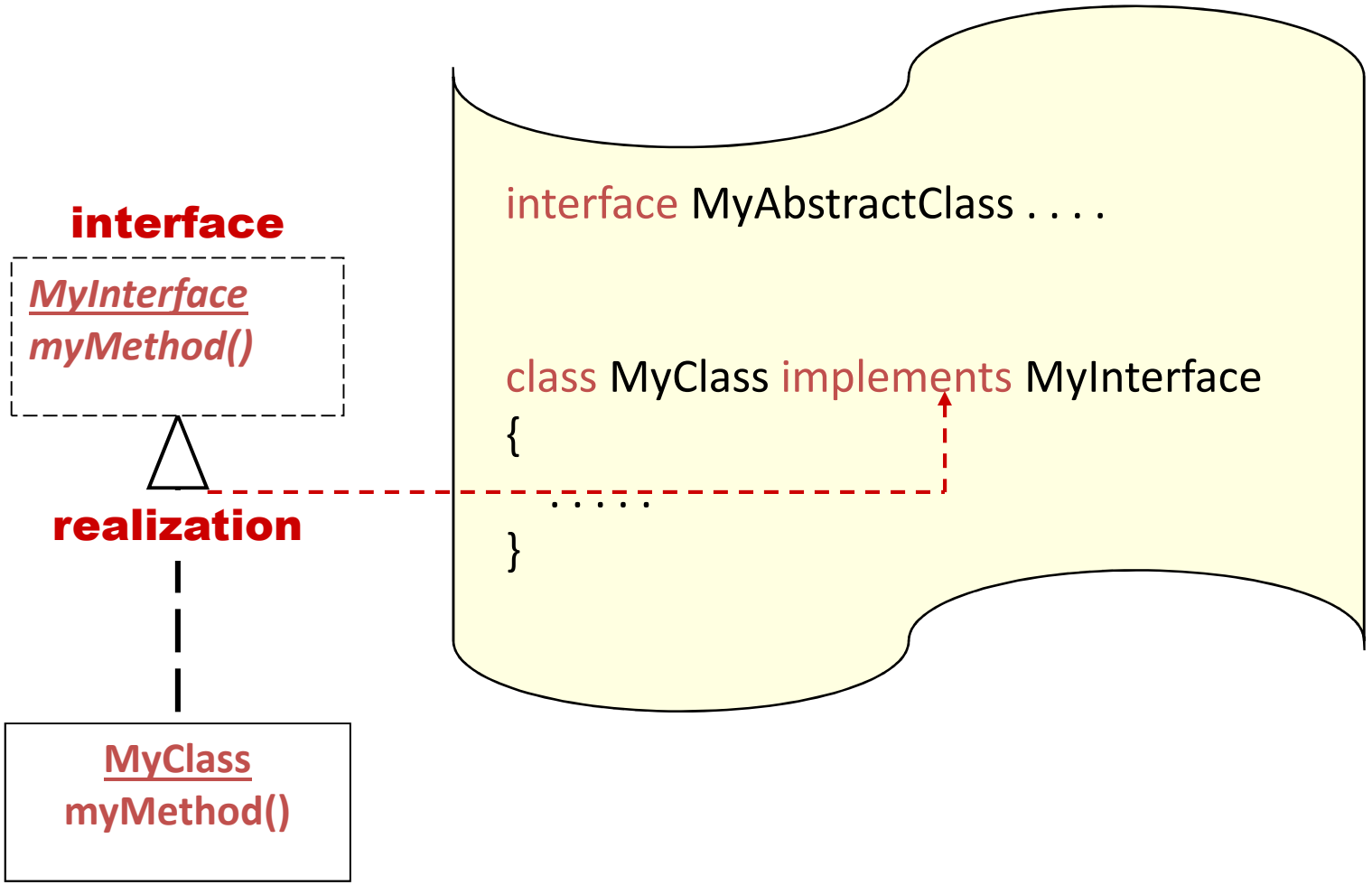
Her müşterinin *kartı yoktur ya da bir tane vardır (0..1)*

Her kart *bir müşteriye aittir.*

Inheritance : UML Notation



Interfaces: UML Notation

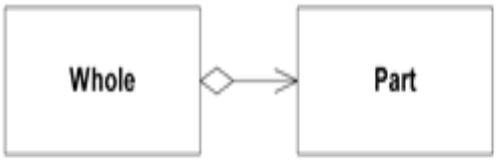


Aggregation İlişkisi

- *Aggregation* "has a« birliktelik (association) ilişkisinin bir başka şeklidir.
- «Aggregation» ilişkisi «association» ilişkisinden daha güçlüdür.
- «Aggregation» «part-whole» ya da (part-of relationship) birlikteliğini tanımlar.

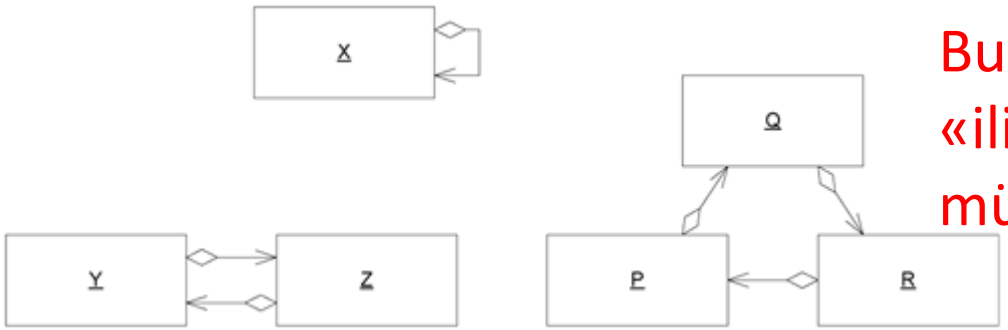
Aggregation Hiyerarşisi

- «Aggregation» "whole/part" bütünün parçası ilişkisi olarak betimlenir.




```
public class Whole
{
  private Part itsPart;
}
```

- Bir bütün(whole) kendisinin bir parçası olamaz (A whole cannot be its own part)
- Bu nedenle de bu ilişkinin örnekleri döngüsel olarak gerçekleşemez.
- Tek bir nesne kendisi ile ilişkili olamaz (aggregation ilişkisi).
- İki nesne karşılıklı olarak birbirleri ile «aggregation» birlikteliği oluşturmaz.
- Üç nesne bir «aggregation» ilişkisi döngüsü oluşturmaz.



Bu üç « aggregation «ilişkisinin gerçekleşmesi mümkün değildir.

Aggregation Hiyerarşisi

- Parça nesneleri (part objects) birbirlerinden bağımsız olarak oluşturulabilirler (created) ve bağımsız olarak silinebilirler (deleted independently)

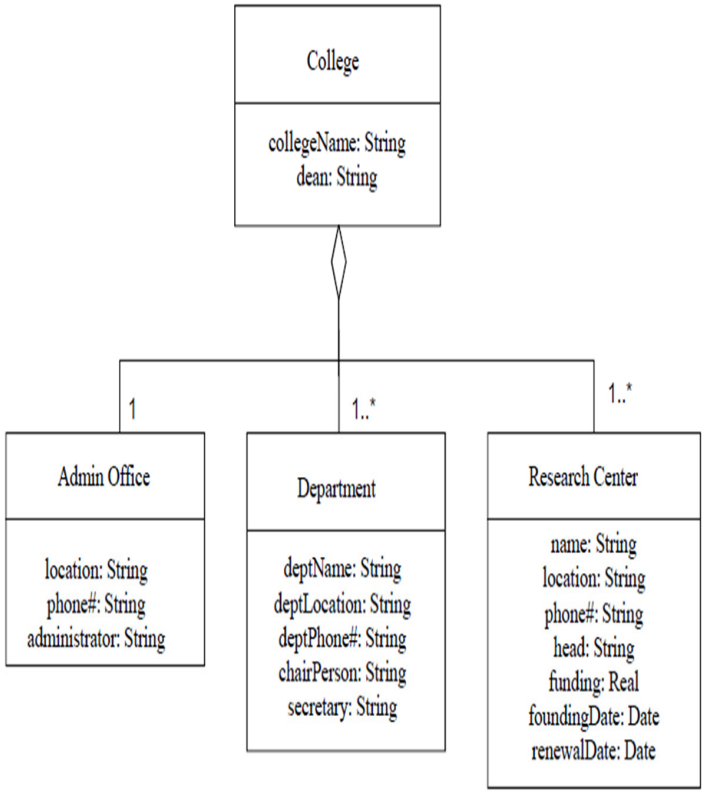
Aggregate class - College

Part classes –

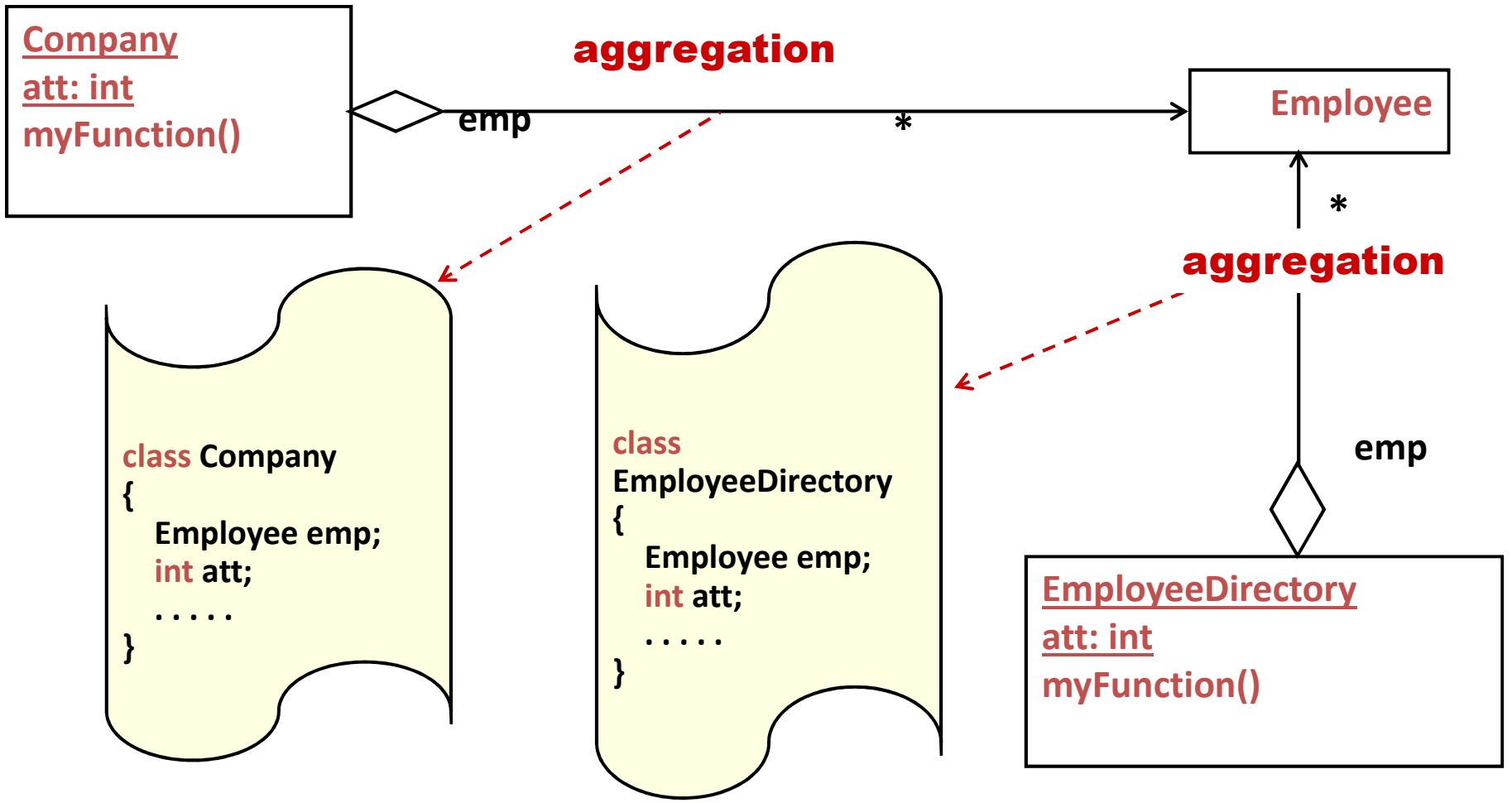
AdminOffice IS PART OF College,

Department IS PART OF College,

ResearchCenter IS PART OF College



Aggregation : UML Gösterimi



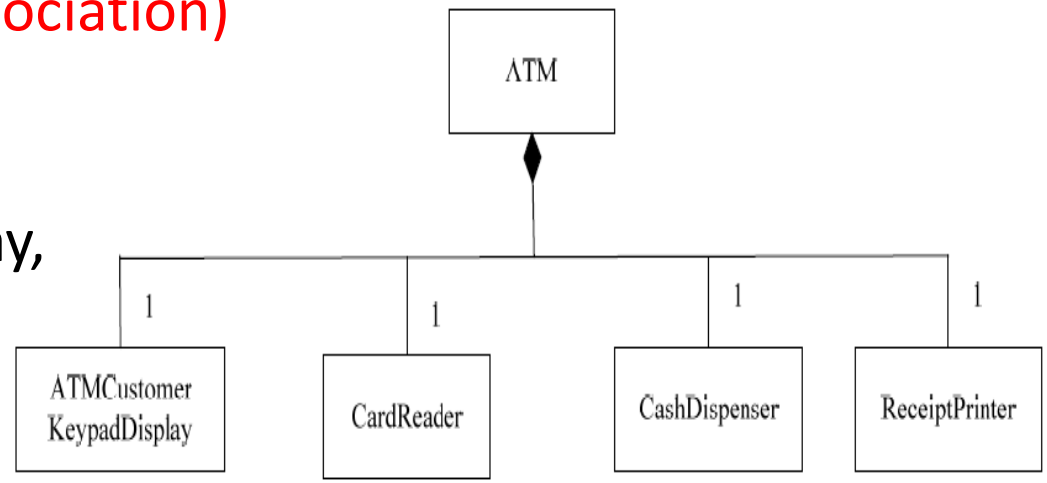
«Composition» Hiyerarşisi

- Bütün(whole) ve parça nesneleri (part objects) birlikte tanımlanırlar (created), işlemlerini gerçekleştirirler (live) ve birlikte yok olurlar (die).
- Genellikle fiziksel bir birliktelik oluştururlar (physical association)

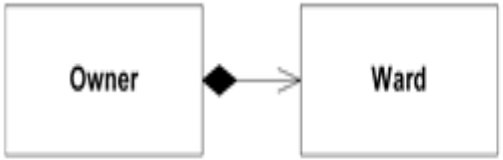
Composite class - ATM

Part classes

ATMCustomerKeypadDisplay,
CarReader,
CashDispenser,
ReceiptPrinter

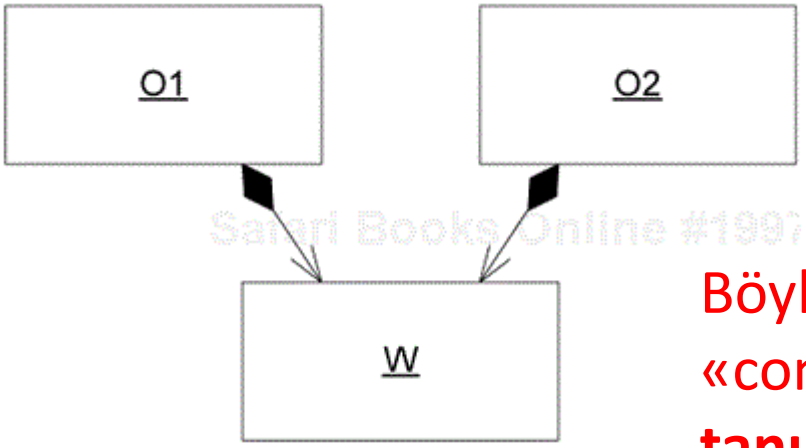


«Composition» Hiyerarşisi



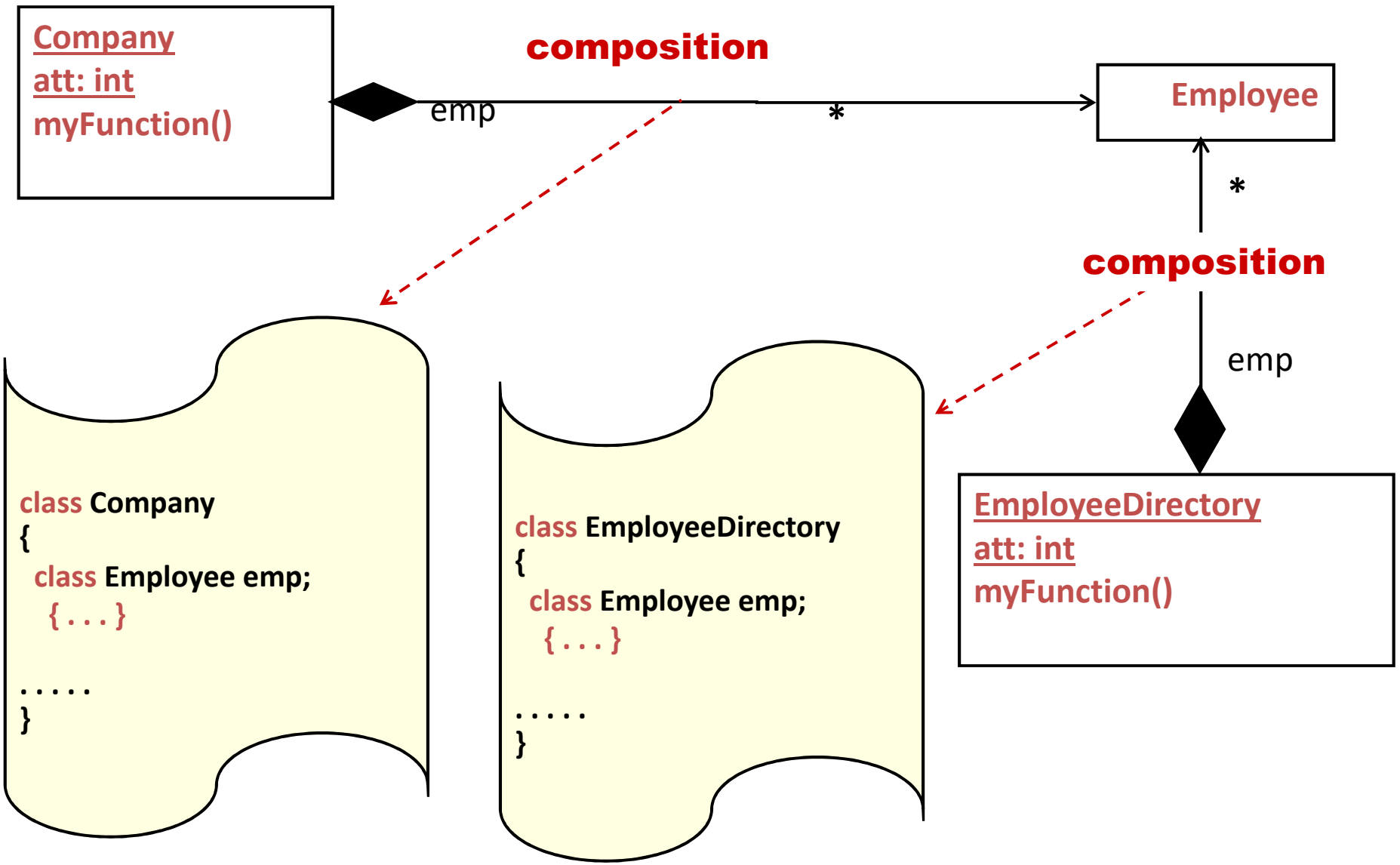

```
public class Owner
{
  private Ward itsWard;
}
```

« Composition» , «aggregation» ilişkisinin özel bir halidir.

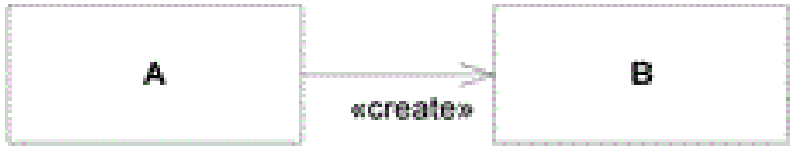


Böyle bir
«composition» ilişkisi
tanımlanamaz.

Composition: UML Gösterimi

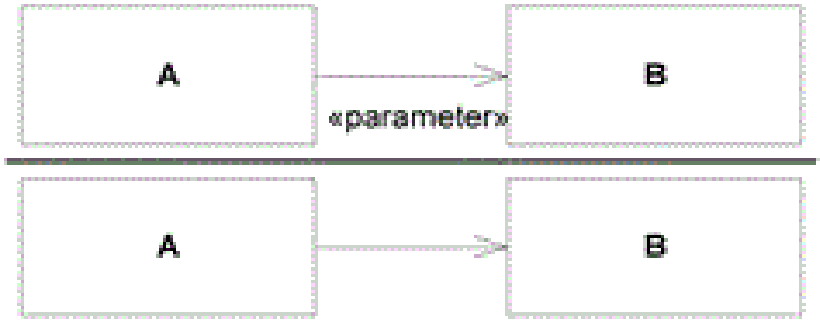


Association Stereotypes

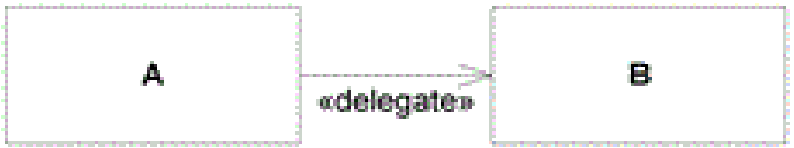



```
public class A {
  public B makeB() {
    return new B();
  }
}
```


```
public class A {
  public void f() {
    B b = new B();
    // use b
  }
}
```




```
public class A {
  public void f(B b) {
    // use b;
  }
}
```



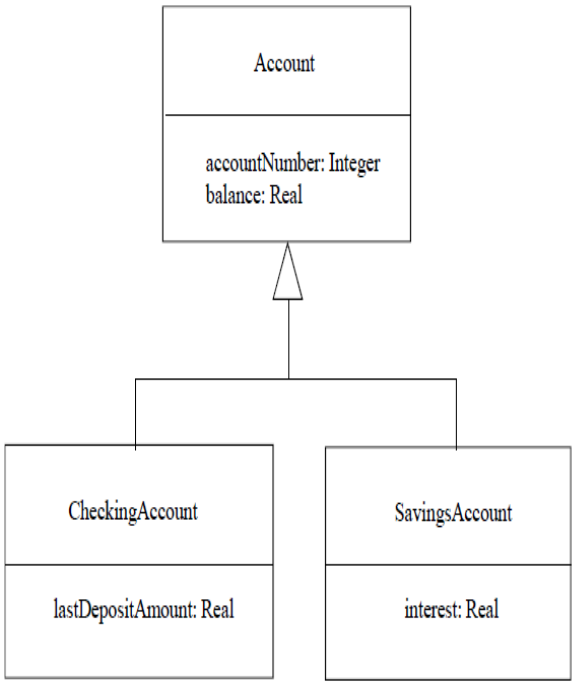

```
public class A {
  private B itsB;
  public void f() {
    itsB.f();
  }
}
```

Generalization / Specialization Hierarchy

- Bazı sınıflar benzerdir; fakat birbirlerine denk değildir.
- Bu sınıfların bazı ortak öznitelikleri (attributes) vardır; fakat farklı öznitelikleri de mevcuttur.
- Ortak olan öznitelik (Common attributes) Generalized Class (superclass) şeklinde üst sınıfta soyut olarak tanımlanırlar.

  Account (Account number, Balance)
- Alt sınıf (subclass) ve üst sınıf (superclass) arasında IS A bağıntısı vardır.

  Savings Account IS A Account

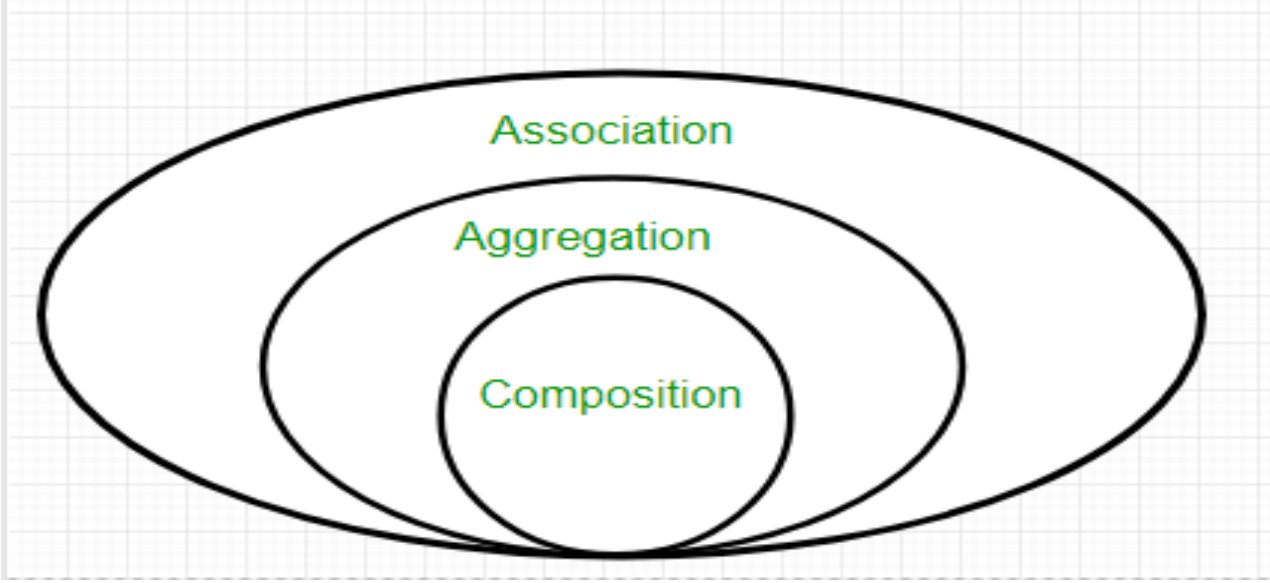


Aggregation bağıntısı Java kodu

```
public class Subject {
private String name;
public void setName(String name)  {
this.name = name;
}
public String getName()
{
return name;
}
}
public class Student {
private Subject[] studyAreas = new Subject[10];
//the rest of the Student class
}
```

Each class referenced is considered to be part-of the aggregate class.

Sınıflar ararası İlişkilerin Hiyerarşisi



```
public class Job {
   private String role;
   private long salary;
   private int id;
public String getRole()
   {    return role;          }
public void setRole(String role)
    {    this.role = role;   }
public long getSalary()
   {   return salary;    }
public void setSalary(long salary)
   {  this.salary = salary ;  }
public int getId()
    {  return id;  }
 public void setId(int id)
    {  this.id = id;   }
                    }
```

```
public class Person {

   //composition has-a relationship
   private Job job;

   public Person(){
      this.job=new Job();
      job.setSalary(1000L);
   }
   public long getSalary()
{     return job.getSalary();  }
                          }
```

```
public class TestPerson {

   public static void main(String[] args)   {
        Person person = new Person();
         long salary = person.getSalary();  }
                          }
```

Composition ilişkisine ait Java kodu

Aynı package içerisinde tanımlanmıştır.