1906003172019

Tasarım Desenleri

Dr. Öğr. Üy. Önder EYECİOĞLU
Bilgisayar Mühendisliği

Hafta	İşlenecek Konu
1	Bölüm 1: Yazılım Tasarımı, Yazılım Örüntüleri ve UML
	1.1. Yazılım geliştirme süreci
	1.2. Yazılım Tasarımı ve Tasarım Örüntüleri
2	1.3. UML
	1.3.1. Sınıf Diyagramı (Class Diagram)
	1.4. Java Dökümantasyonu (Javadoc)
3	Bölüm 2: Nesnesel Kavramlar
	2.1. Modülerlik
	2.2. Soyutlama (abstraction)
	2.2.1. Genelleştirme ve Özelleştirme
4	2.3. Sınıf Tasarımı
	2.3.1. Bilgi Saklama
	2.4. Kalıt (inheritance)
5	2.5. Tür Değiştirme (Type Casting)
	2.6. Soyut Sınıf (Abstract Class)
	2.7. Soyut Metotlar
6	2.8. Polimorfizm (polimorphism)
	2.9. Arayüz (interface)

Hafta	İşlenecek Konu
	2.12.4. Java Collections
	2.12.5. Java Generics
	2.12.6. Java Nesnelerinin Karşılaştırılması
9	Bölüm 3: Tasarım Örüntüleri
	3.1. Tasarım Örüntüsü Nedir?
	3.2. İncelenecek Tasarım Örüntüleri ve Tasarım Kategorileri
10	3.3. Gözlemci Örüntüsü (Observer Pattern)
	3.4. Dekoratör Örüntüsü (Decorator Pattern)
	3.5. Strateji Örüntüsü (Strategy Pattern)
	3.5.1. Soyut Fabrika Örüntüsü (Abstract Factory Pattern)
11	3.6. Tekli Örüntü (Singleton Pattern)
	3.7. Komut Örüntüsü (Command Pattern)
	3.8. Adaptör Örüntüsü (Adapter Pattern)
12	3.9. Fasat Örüntüsü (Façade Pattern)
	3.10. Kalıp Metodu Örüntüsü (Template Method Pattern)

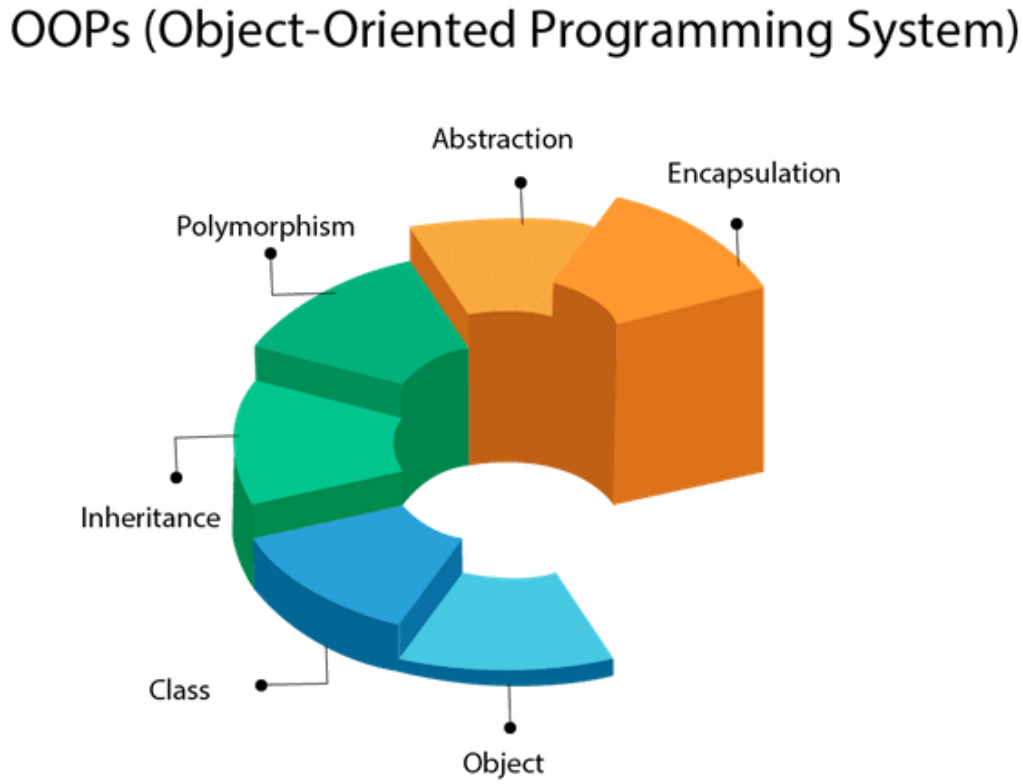
Bölüm Hedefi

Nesnesel tasarımda kullanılan temel kavramlar Java dili kullanılarak açıklanacaktır. Bu kavramlar bilgi saklama, kalıt, soyut sınıf, arayüz, polimorfizm, nesnelerin yaşam döngüsü, sınıf ve nesne ilişkisi ve Java'da kullanılan diğer nesnesel yapılardan oluşmaktadır. Son olarak da Java kütüphaneleri kısaca gözden geçirilecektir.

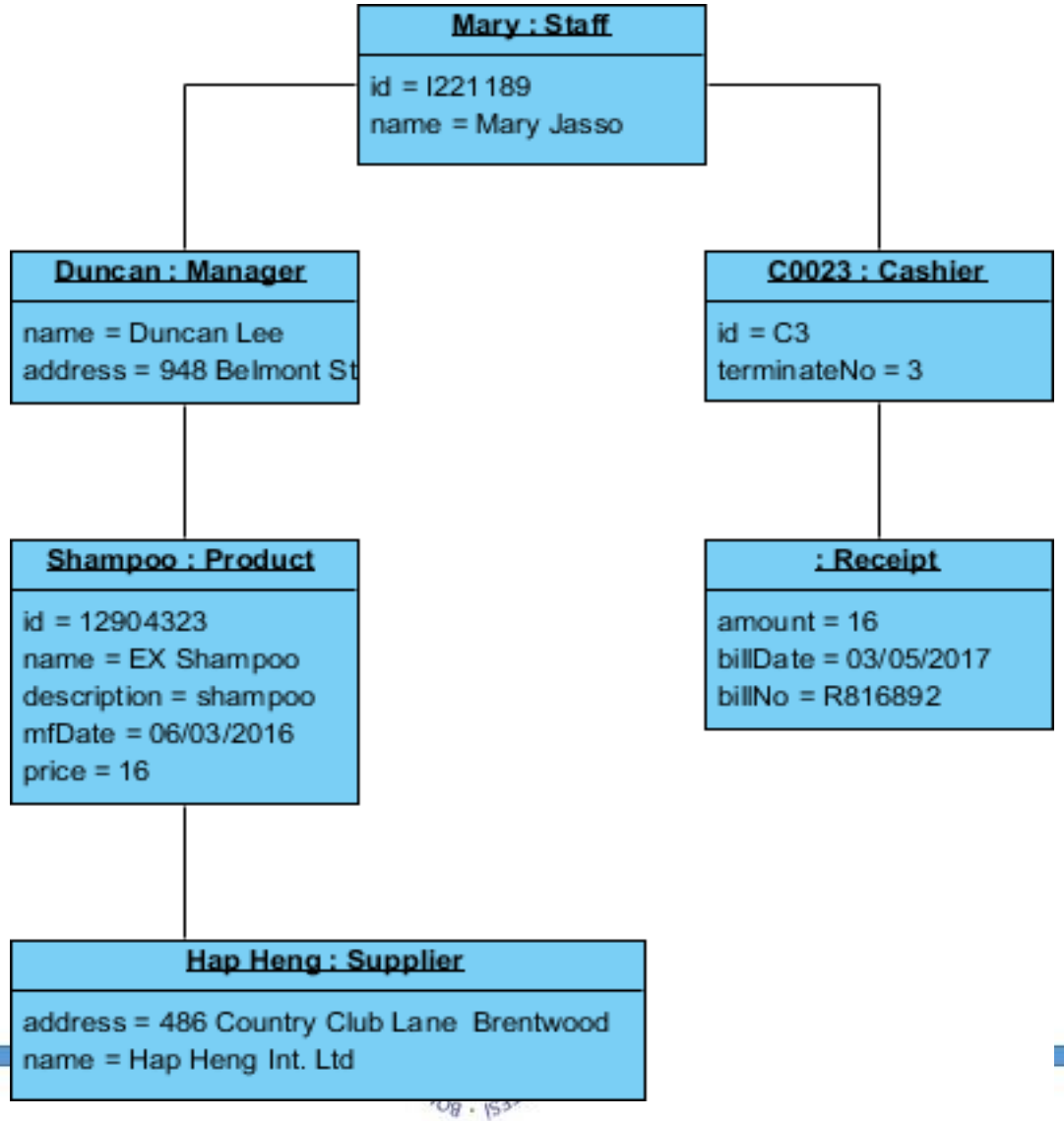
Nesne Tabanlı Programlama Kavramları

Java, Nesne Tabanlı bir Dildir. Nesneye Yönelik özelliğe sahip bir dil olarak Java, aşağıdaki temel kavramları destekler:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation



Nesne (Oject)


Mary : Staff
id = I221189
name = Mary Jasso
Duncan : Manager
name = Duncan Lee
address = 948 Belmont St
C0023 : Cashier
id = C3
terminateNo = 3
Shampoo : Product
id = 12904323
name = EX Shampoo
description = shampoo
mfDate = 06/03/2016
price = 16
: Receipt
amount = 16
billDate = 03/05/2017
billNo = R816892
Hap Heng : Supplier
address = 486 Country Club Lane Brentwood
name = Hap Heng Int. Ltd

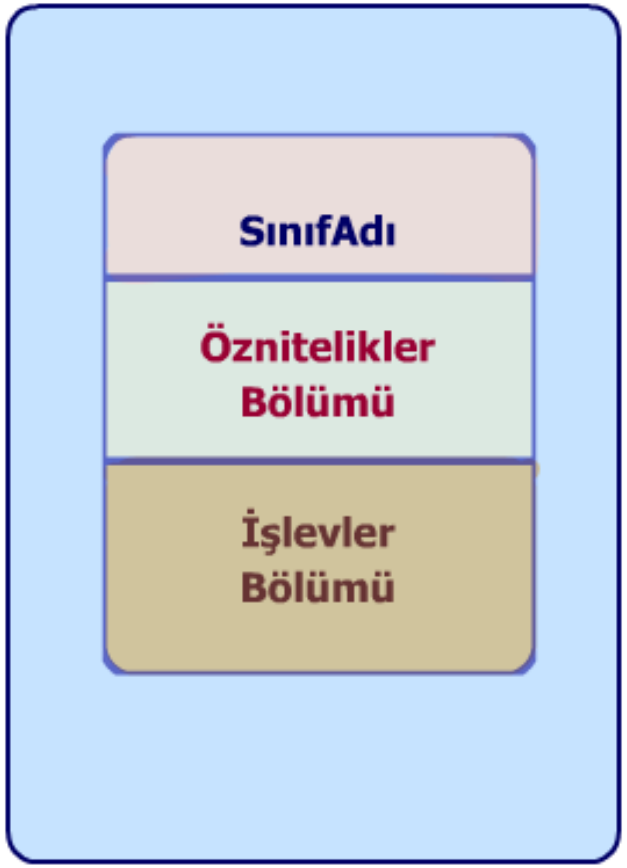
Sınıf Diyagramları

- Kavramsal (conceptual) bir model olan Sınıf Diyagramları, hem gereksinimlerin müşteriye özetlenmesinde hem de tasarımda kullanılmaktadır.

- Nesne tabanlı sistemlerin doğru tasarlanması tamamen tasarımcının tecrübe ve bilgisi ile ilgilidir, bu sebeple tasarlanacak Sınıf Diyagramlarının, **türetme** (**inheritance**), **soyut sınıflar**, **sınıf hiyerarşileri**, **tasarım örnekleri** (**design pattern**) gibi detaylarının düşünülmesi gerekmektedir.

Sınıf Diyagramları

- Sınıf Diyagramları UML 'in en sık kullanılan diyagram türüdür.
- Sınıflar nesne tabanlı programlama mantığından yola çıkarak tasarlanmıştır.
- Sınıf diyagramları bir sistem içerisindeki nesne tiplerini ve birbirleri ile olan ilişkileri tanımlamak için kullanılırlar.
- UML'de sistem yapısını anlatmak için Sınıf Diyagramlarını kullanırız. Sınıf Diyagramları, aralarında bizim belirlediğimiz ilişkileri içeren sınıflar topluluğudur.
- UML' de bir sınıf yanda görüldüğü gibi 3 bölmeden oluşan dikdörtgen ile ifade edilir. Bunlara ek olarak notlar (notes) ve Kısıtlar (Constraints) tanımlanabilir.
- Dikdörtgenin en üstünde sınıfın adı vardır.
- Sınıfın sahip olduğu **Öznitelikler** SınıfAdı'nın hemen altına ikinci bölmeye yazılırken, son bölmeye de sınıfın sahip olduğu **İşlevler** yazılır. Genel bir kullanım şekli olarak sınıflara isim verilirken her kelimenin ilk harfi büyük yazılır.



Sınıf Diyagramları

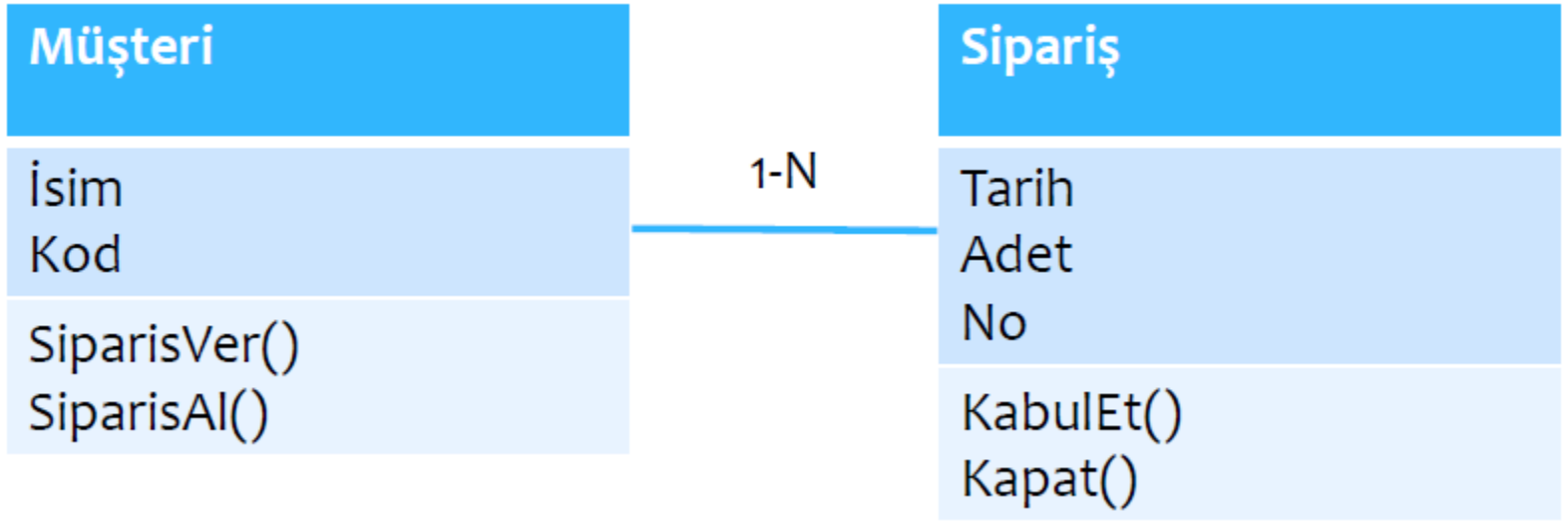
ERİŞİM

- **Public:**diğer sınıflar erişebilir. UML'de **+** sembolü ile gösterilir.
- **Protected:**aynı paketteki (*package*) diğer sınıflar ve bütün alt sınıflar (*subclasses*) tarafında erişilebilir. UML'de**#**sembolü ile gösterilir.
- **Package:**aynı paketteki (*package*) diğer sınıflar tarafında erişilebilir. UML'de **~**sembolü ile gösterilir.
- **Private:**yalnızca içinde bulunduğu sınıf tarafından erişilebilir (diğer sınıflar erişemezler). UML'de -sembolü ile gösterilir.

Sınıf Diyagramları

Sınıflar Arası İlişki (Assocation)

- UML'de sınıflar arasındaki ilişki, iki sınıf arasına düz bir çizgi çekilerek ve bu çizginin üzerine ilişkinin türü yazılarak gösterilir.

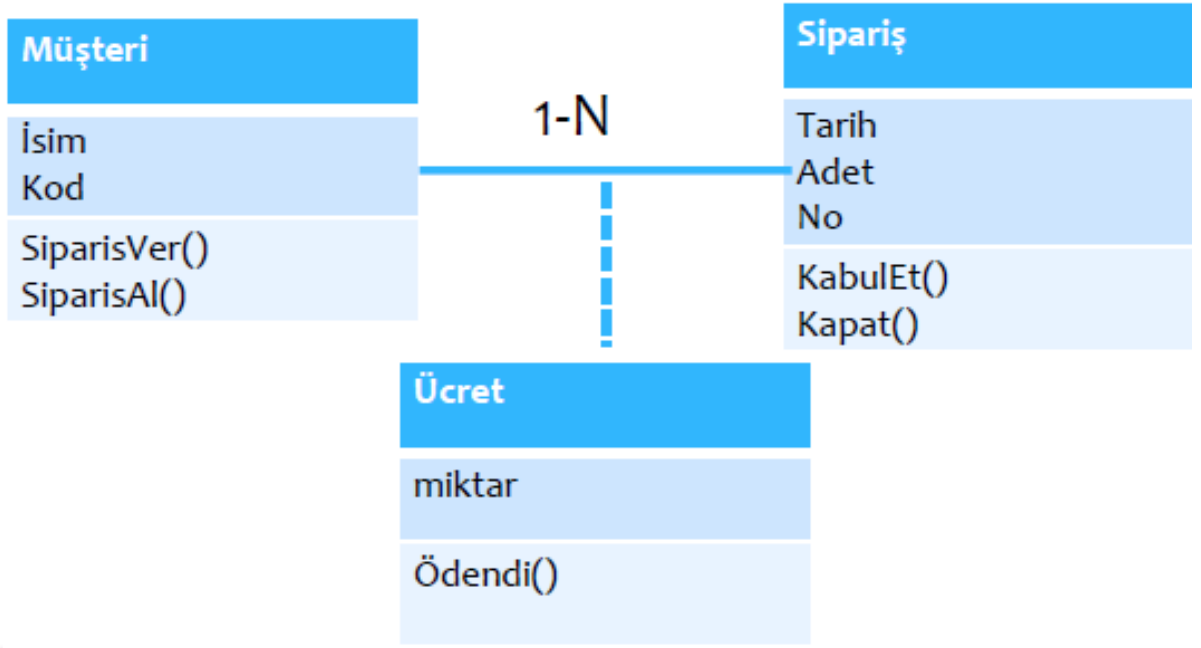


Sınıf Diyagramları

İlişki Sınıfları

- Bazı durumlarda sınıflar arasındaki ilişki, bir çizgiyle belirtebilecekşekilde basit olmayabilir.
- Bu durumda ilişki sınıfları kullanılır.
- İlişki sınıfları bildigimiz sınıflarla aynıdır.
- Sınıflar arasındaki ilişki eğer bir sınıf türüyle belirleniyorsa UML ile gösterilmesi gerekir.

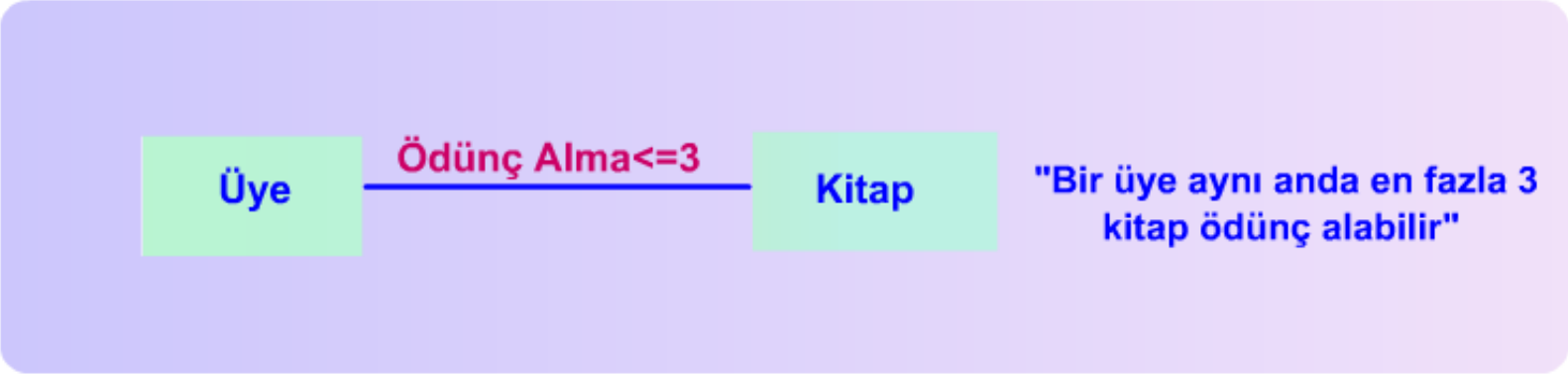
- Müşteri ile Sipariş sınıfı arasında ilişki vardır. Fakat müşteri satın alırken Ücret ödemek zorundadır
- Bu ilişkiyi göstermek için Ücret sınıfı ilişki ile kesikli çizgi ile birleştirilir.



Sınıf Diyagramları

Kısıtlar

Bazı durumlarda belirtilen ilişkinin bir kurala uyması gerekebilir. Bu durumda ilişki çizgisinin yanına kısıtlar (constraints) yazılır. Örneğimizde bir üye aynı anda en fazla 3 kitap ödünç alabilir olsun.



Sınıf Diyagramları

İlişki Tipleri

İlişkiler her zaman bire-bir olmak zorunda değildirler. Eğer bir sınıf, n tane başka bir sınıf ile ilişkiliyse biz buna bire-çok ilişiki deriz. Örneğin bir dersi 30 öğrenci alıyorsa Öğretmen ile Öğrenci sınıfları arasında 1-30 bir ilişki vardır. Çizelgede bunu gösterirken Öğretmen sınıfına 1, Öğrenci sınıfına ise 30 yazarız. Gösterimi aşağıdaki gibidir:

Sınıf Diyagramları

İlişki Tipleri

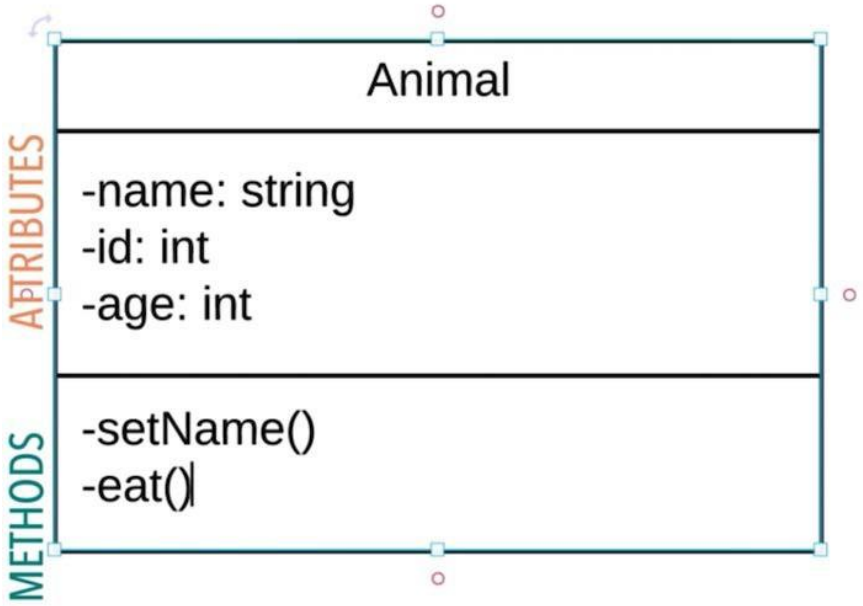
Temel ilişki tipleri aşağıdaki gibi listelenebilir:

1	Bire-bir
2	Bire-çok
3	Bire-bir veya daha fazla
4	Bire-sıfır veya bir
5	Bire-sınırlı aralık (Örneğin: bire-[0,20] aralığı)
6	Bire-n (UML'de birden çok ifadesini kullanmak için * simgesi kullanılır)
7	Bire-beş ya da Bire-sekiz

Sınıf (Class)

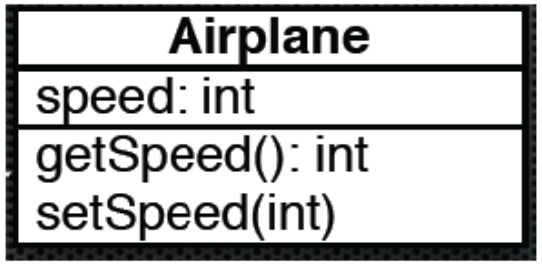
Nesnelerimizi oluşturabilmek için önce nesnelerin ait oldukları sınıfları tanımlamamız gerekiyor. Bir sınıf, aynı zamanda, kendisinden aynı türde nesne oluşturabileceğiniz bir plan olarak da tanımlanabilir. Sınıf herhangi bir yer kaplamaz.

Sınıf denilen sey aslında birden fazla metod ve degiskeni birarada tutan bir yapıdır. Ancak bu degisken ve metodların da aynı göreve iliskin olması beklenir.



Sınıf (Class)

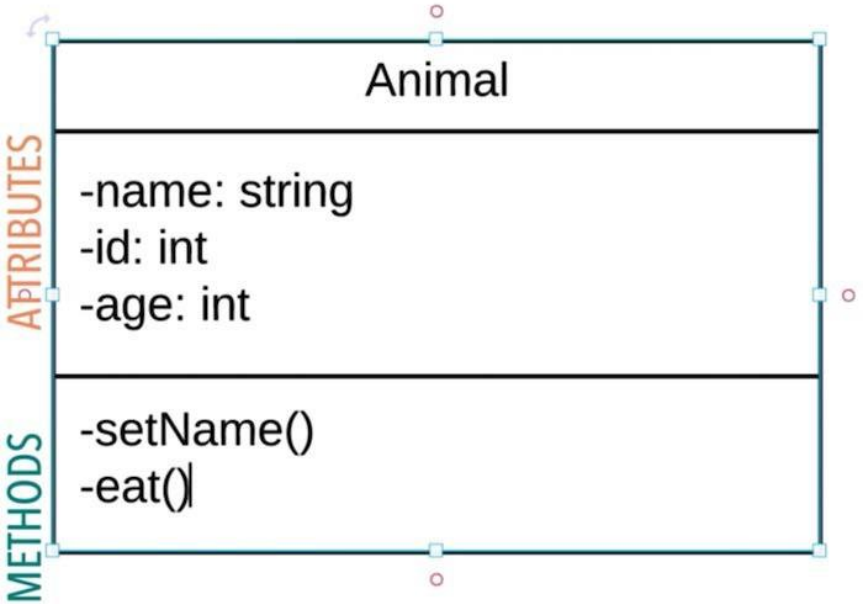
```java
public class Airplane {

    private int speed;

    public Airplane(int speed) {
        this.speed = speed;
    }

    public int getSpeed() {
        return speed;
    }

    public void setSpeed(int speed) {
        this.speed = speed;
    }

}
```



Sınıf (Class)

Sınıf(class) aşağıdakileritanımlar:

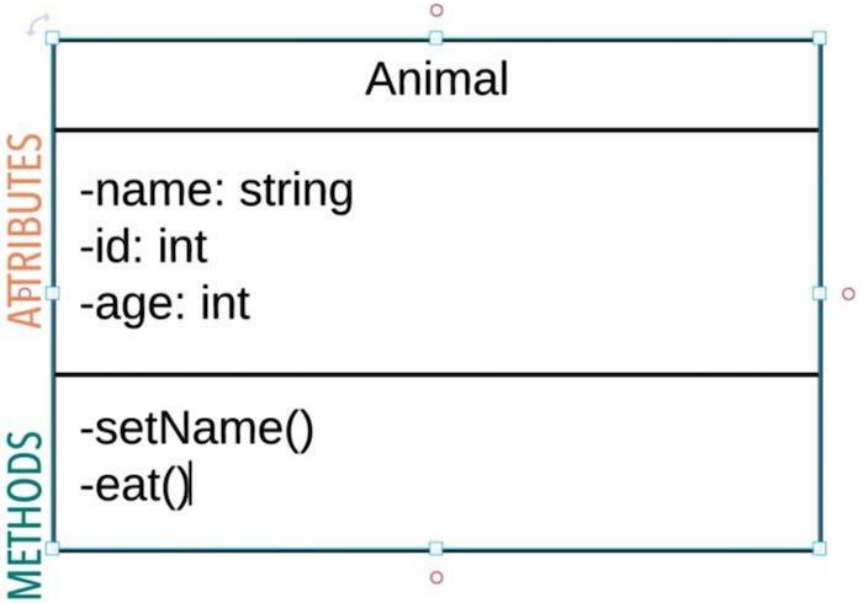
- Öznitelik(Atrtribute):Nesne özelliklerini tanımlayan değişkenler, adları ve türleri ile.
- Metotlar(methods): Metot adları, döndürdüğü tür, parametreleri, ve metotu gerçekleştiren program kodu



Sınıf (Class)

Özellik (property) NYP bağlamında, nesnenin durumunu oluşturan, nesnenin ait olduğu sınıfın içine kodlanmış herbir değişkene verilen isimdir. Özellik, Java bağlamında nitelik (attribute) olarak adlandırılır.

Davranış (behaviour) NYP bağlamında, nesnenin durumunda değişiklik yapabilen, nesnenin ait olduğu sınıfın içine kodlanmış herbir işleve verilen isimdir. Davranış, Java bağlamında yöntem (method) olarak adlandırılır.



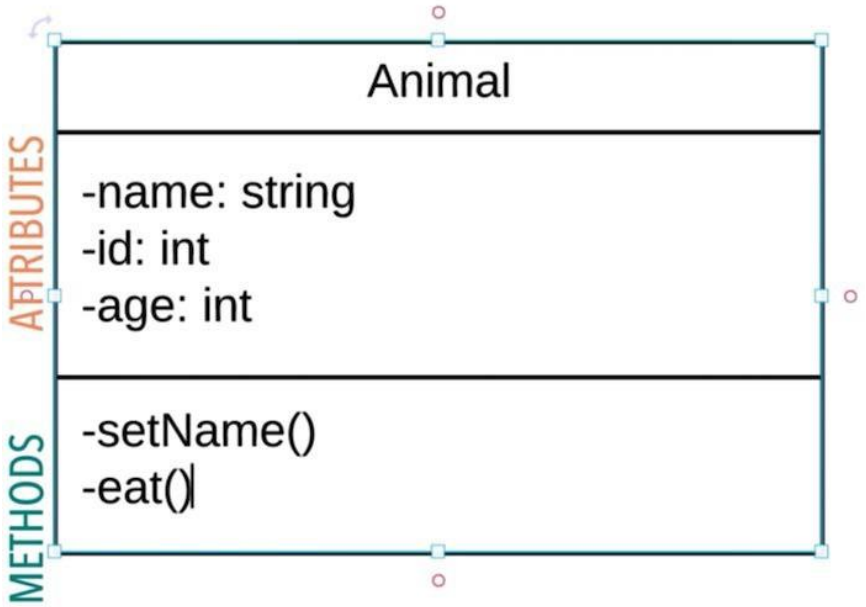
Sınıf (Class)

Public: diğer sınıflar erişebilir. UML'de +sembolü ile gösterilir.

Protected: aynı paketteki (package) diğer sınıflar ve bütün alt sınıflar (subclasses) tarafında erişilebilir. UML'de#sembolü ile gösterilir.

Package: aynı paketteki (package) diğer sınıflar tarafında erişilebilir. UML'de ~sembolü ile gösterilir.

Private: yalnızca içinde bulunduğu sınıf tarafından erişilebilir (diğer sınıflar erişemezler). UML'de -sembolü ile gösterilir.



Sınıf (Class)

Bir sınıf, aşağıdaki değişken türlerinden herhangi birini içerebilir.

Yerel değişkenler - Yöntemler, yapıcılar veya bloklar içinde tanımlanan değişkenlere yerel değişkenler denir. Değişken, yöntem içinde bildirilecek ve başlatılacak ve değişken, yöntem tamamlandığında yok edilecektir.

Örnek değişkenler - Örnek değişkenleri, bir sınıf içindeki ancak herhangi bir yöntemin dışındaki değişkenlerdir. Bu değişkenler, sınıf başlatıldığında başlatılır. Örnek değişkenlerine, söz konusu sınıfın herhangi bir yöntemi, kurucusu veya bloğunun içinden erişilebilir.

Sınıf değişkenleri - Sınıf değişkenleri, bir sınıf içinde, herhangi bir yöntemin dışında, static anahtar sözcüğüyle bildirilen değişkenlerdir.

Nesne Oluşturma

Sınıfın bir tanım, nesnenin ise o tanıma göre oluşturulmuş fiziksel bir varlık olduğunu tekrar hatırlayalım. Bu ifade üzerinde biraz durursak şu sonuca varabiliriz: Nesne oluşturulmadığı sürece bellekte fiziksel olarak oluşturulmuş bir varlık yoktur; sınıfın nitelikleri için bellekten yer ayrılmamıştır.

new işleci

Herhangi bir sınıfa ait bir nesne oluşturulabilmesi için **new** işleci kullanılır. **new**, belirtilen sınıfa ait bir nesne oluşturup bu nesnenin bellekteki adresini (referansını) döndürür.

```
Zaman zamanNesnesi = new Zaman();
```

```
Zaman zamanNesnesi;

zamanNesnesi = new Zaman();
```

KALITIM (INHERITANCE)

Nesneler arasında ortak özellikler varsa, bunu her sınıfta belirtmek yerine ortak özellikleri bir sınıfta toplayıp, diğer sınıfları da ortak sınıftan türeterek ve yeni özellikler ekleyerek organizasyonu daha etkin hale getirmeye, nesne yönelimli programlamada ***kalıtım (inheritance)*** denir.

Kalıtıma;

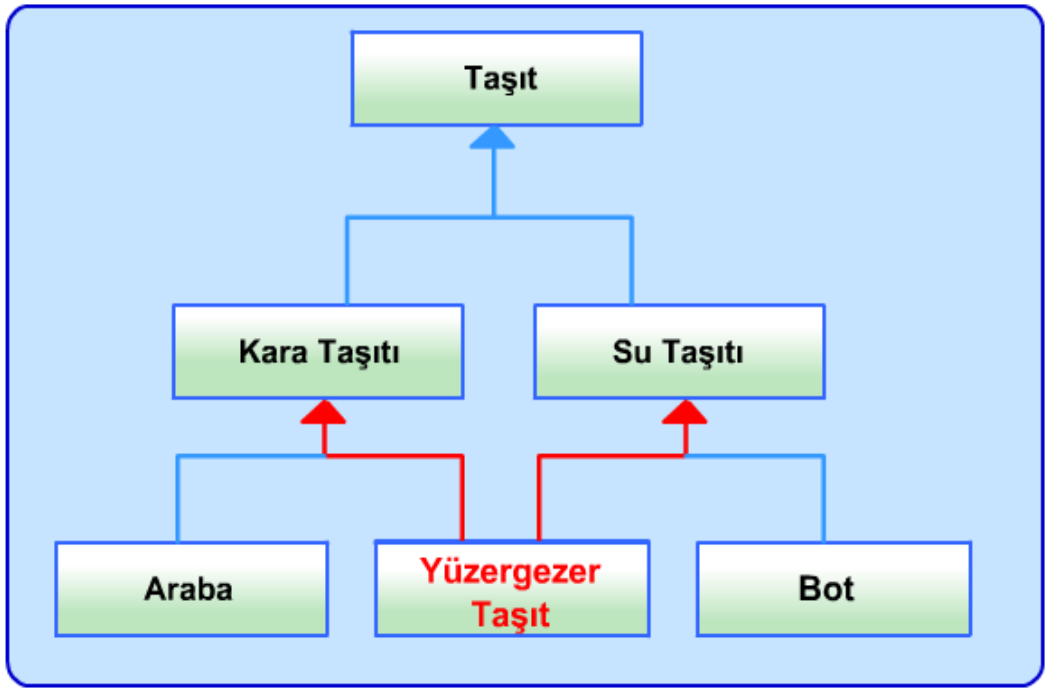
- Nesnenin Görevini arttırarak, yeni üye fonksiyonları ilave etmek
- Bir sınıfın üye fonksiyonlarından birinin görevlerinin farklı biçimde yerine getirecek şekilde değiştirmek

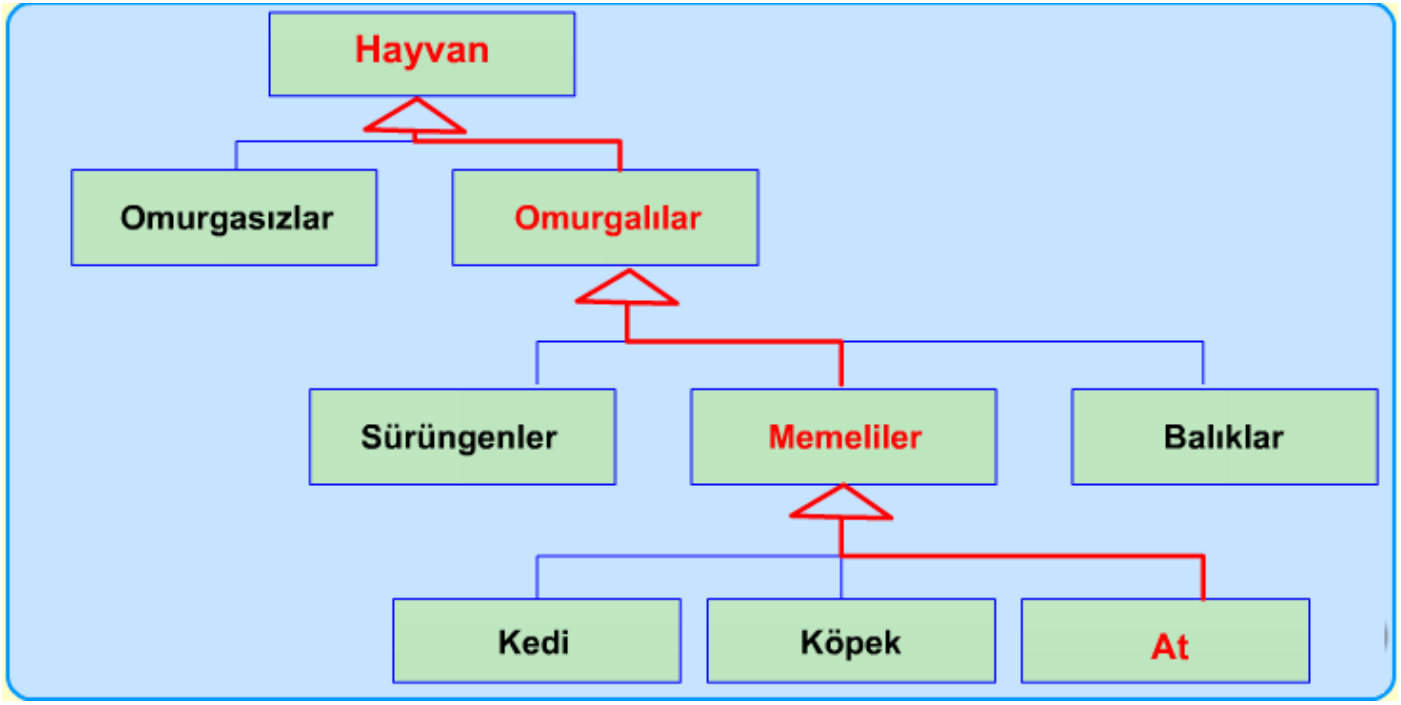
Türetme yapabilmek için öncelikle, tanımlanmış en az bir sınıfın mevcut olması gerekir. Türetme işleminde kullanılan bu mevcut sınıfa taban sınıf (base class), türeme sonucunda ortaya çıkacak yeni sınıfa ise türemiş sınıf (derivated class) denir.

KALITIM (INHERITANCE)

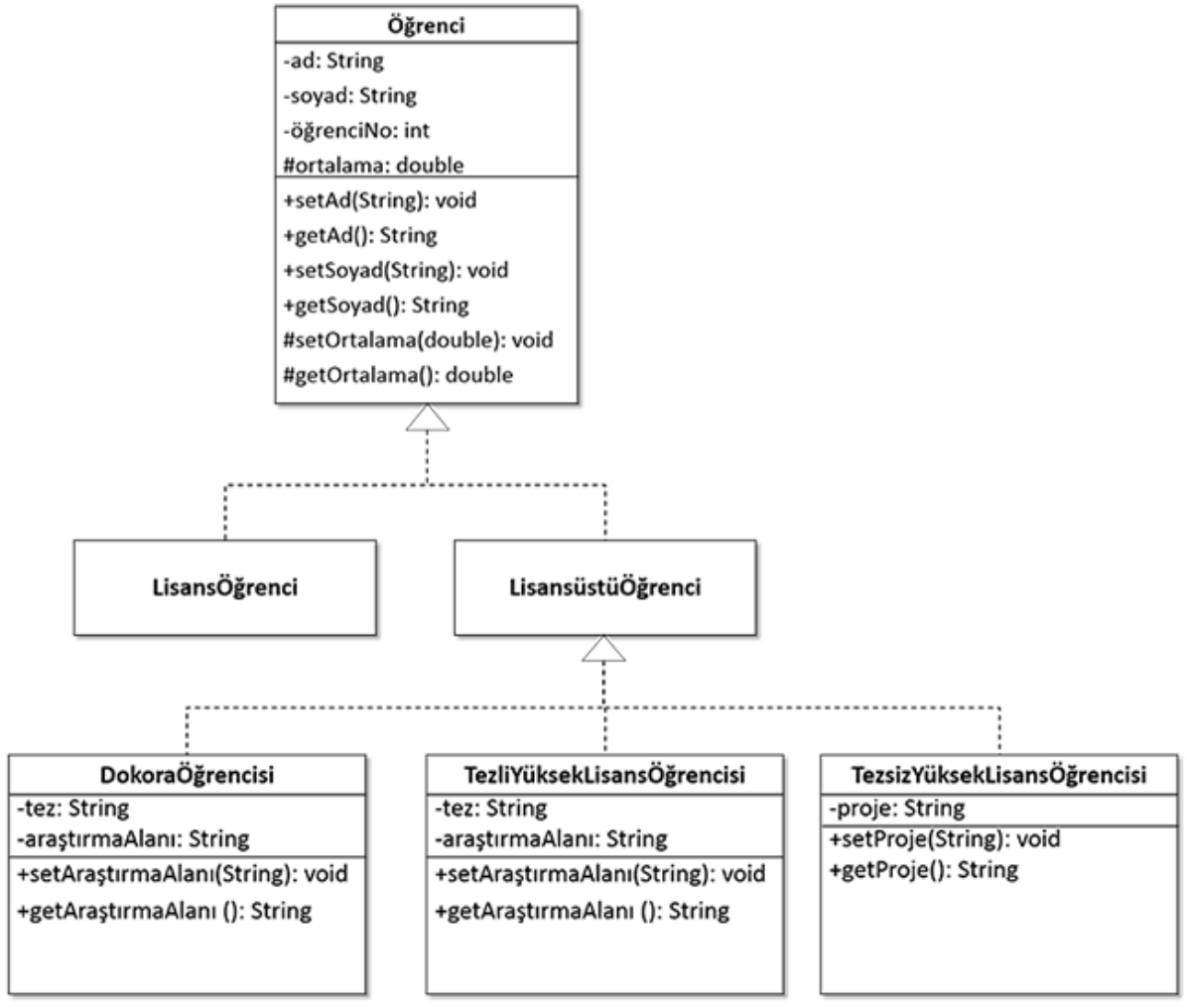
- Bir sınıf özelleştirilerek daha özel bir sınıf oluşturulabilir. Özelleştirme (specialization) için yeni sınıfa ek özellikler ve ek metotlar (işlevler) eklenebilir.
- Özelleştirilerek oluşturulan yeni sınıfa “alt sınıf” (subclass) denir.
- Kendisinden özelleştirme yapılan asıl sınıfa ise “üst sınıf” (superclass) denir.
- Verilen bir veya daha çok sınıftan (alt sınıflar) genelleştirme işlemi yapılarak (generalization) bir üst sınıf oluşturulabilir. Bu durumda tüm alt sınıfların ortak özellikleri ve işlevleri (metotlar) üst sınıfta toplanır.
- Alt/üst sınıflar arasında “kalıtım” (inheritance) ilişkisi mevcuttur. Alt sınıf, üst sınıfın özelliklerini ve metotlarını kalıtsal olarak, belirli kurallara göre, alır (kullanabilir). Tersi olmaz, yani üst sınıf, alt sınıftan hiçbir şey kullanamaz.

KALITIM (INHERITANCE)





KALITIM (INHERITANCE)



KALITIM (INHERITANCE)

```
public class Öğrenci {
  // ...
}
public class LisansÜstüÖğrenci extends Öğrenci {
  // ...
}
public class TezliYüksekLisansÖğrencisi extends LisansÜstüÖğrenci {
   // ...
}
public class TezsizYüksekLisansÖğrencisi extends LisansÜstüÖğrenci {
  // ...
}
public class DoktoraÖğrencisi extends LisansÜstüÖğrenci {
  // ...
}
public class LisansÖğrencisi extends Öğrenci {
  // ...
}
```

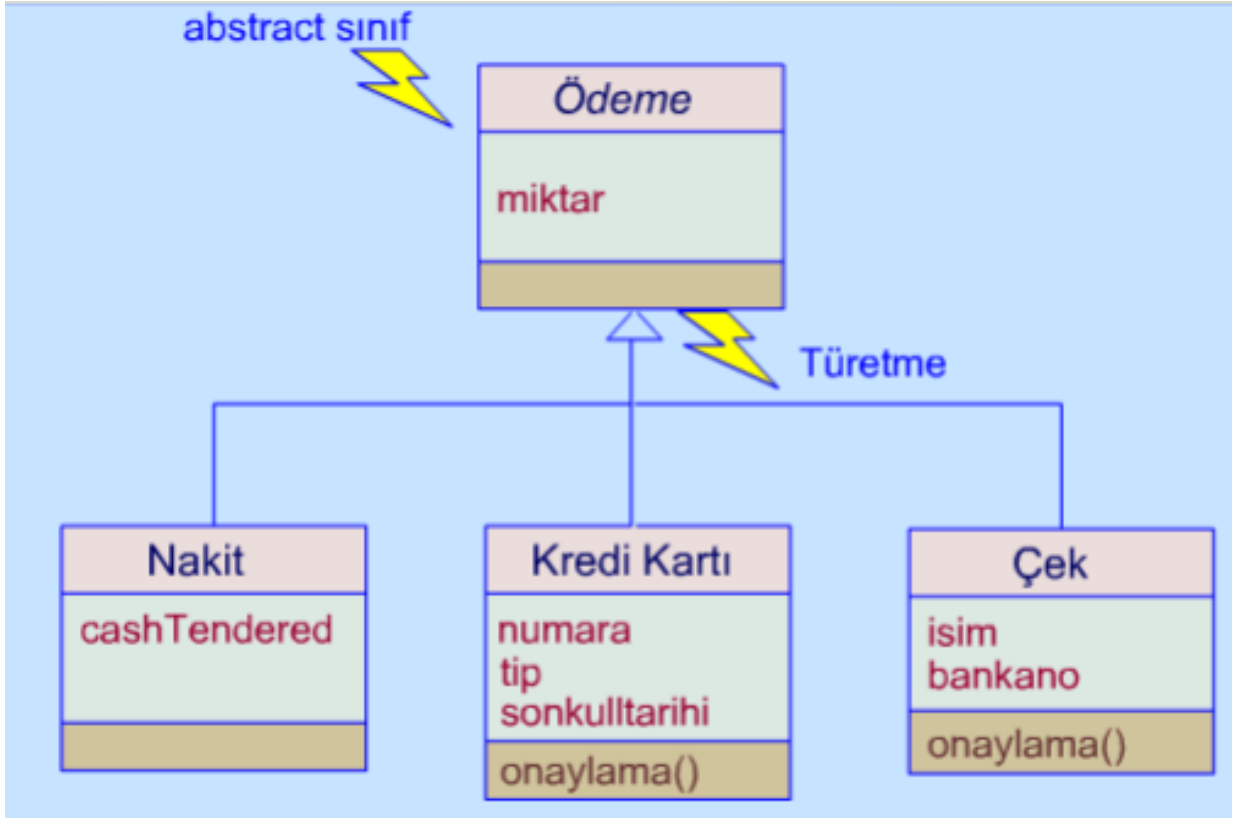
KALITIM (INHERITANCE)

- Bir sınıf özelleştirilerek daha özel bir sınıf oluşturulabilir. Özelleştirme (specialization) için yeni sınıfa ek özellikler ve ek metotlar (işlevler) eklenebilir.
- Özelleştirilerek oluşturulan yeni sınıfa “alt sınıf” (subclass) denir.
- Kendisinden özelleştirme yapılan asıl sınıfa ise “üst sınıf” (superclass) denir.
- Verilen bir veya daha çok sınıftan (alt sınıflar) genelleştirme işlemi yapılarak (generalization) bir üst sınıf oluşturulabilir. Bu durumda tüm alt sınıfların ortak özellikleri ve işlevleri (metotlar) üst sınıfta toplanır.
- Alt/üst sınıflar arasında “kalıtım” (inheritance) ilişkisi mevcuttur. Alt sınıf, üst sınıfın özelliklerini ve metotlarını kalıtsal olarak, belirli kurallara göre, alır (kullanabilir). Tersi olmaz, yani üst sınıf, alt sınıftan hiçbir şey kullanamaz.

KALITIM (INHERITANCE)

Soyut (Abstrct) Sınıflar

- Eğer ortak özelliklerin toplandığı sınıftan gerçek nesneler türetilmesini engellemek istiyorsak **Soyut Sınıf (Abstract Class)** oluştururuz. UML'de bir sınıfın Soyut (Abstract) olması için sınıf ismini italik yazarız.
- Aşağıdaki örnekte Ödeme sınıfını Soyut (Abstract) Sınıfa örnek verebiliriz. Ödeme nakit, çekle ya da kredi kartıyla yapılabilir. Üçü için de birer sınıf yaratılır, özellik ve işlevleri anlatılır. Ama üçünün de ortak özelliği olan ne kadar ödeme yapılacağı bilgisi için Ödeme soyut sınıfı yaratılır. Diğer üç sınıf bu sınıftan türetilir.



Tür Dönüştürme

Bir programlama dilinde önemli bir kavram, tip kavramıdır.

Bir tür, bir dizi değeri ve bu değerlerle gerçekleştirilebilecek işlemleri belirtir. Örneğin, int türü,

tüm 32 bit tam sayıları ve bunlar üzerindeki aritmetik işlemleri belirtir.

Bir sınıf türü, onlara uygulanabilecek yöntemlerle birlikte bir dizi nesneyi belirtir.

Java'daki her tür aşağıdakilerden biridir:

1. İlkel bir tür (int, short, long, byte, char, float, double, boolean)
2. Sınıf türü
3. Arayüz (Interface) türü
4. Dizi (Array) türü
5. NULL tür

Tür Dönüştürme

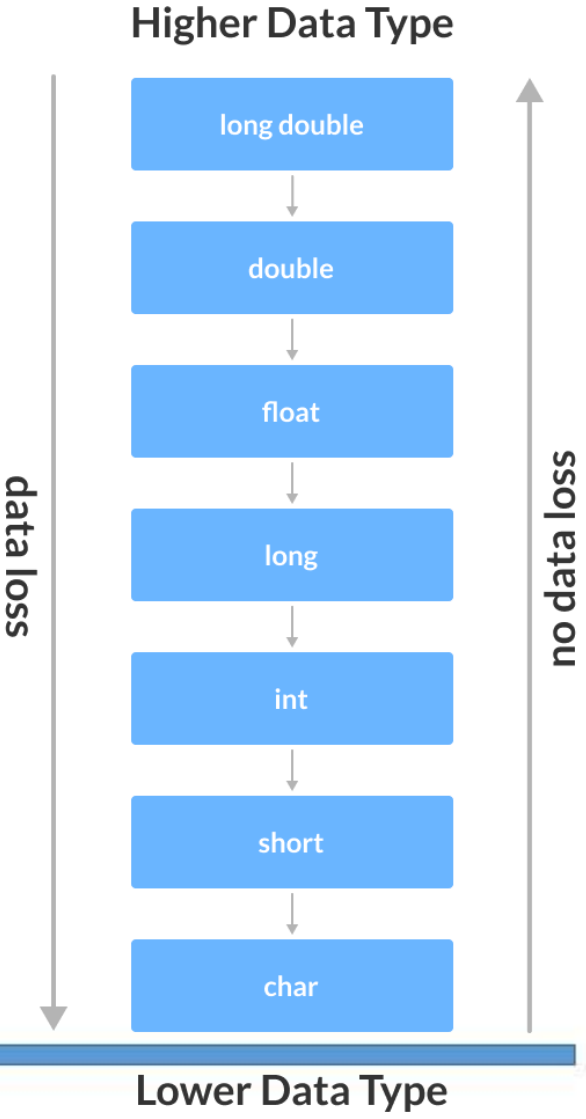
Örtülü tür dönüştürme (Zorlamalar) /Implicit Type Conversion

Karma modlu ifadeleri etkinleştiren programlama dilleri, örtük işlenen türü dönüşümleri için kuralları tanımlamalıdır.

Zorlama, türler arasında otomatik bir dönüşüm olarak tanımlanır.

```
//Implicit Type Conversion: C++
int intType2;
intType2 = 10.55 + 10.55;
std::cout << intType2<< std::endl;
```

```
// Implicit Type Conversion: Java (Error)
int intType2;
intType2=10.99+10.99;
System.out.println(intType2);
```



Tür Dökümü/ Type Casting

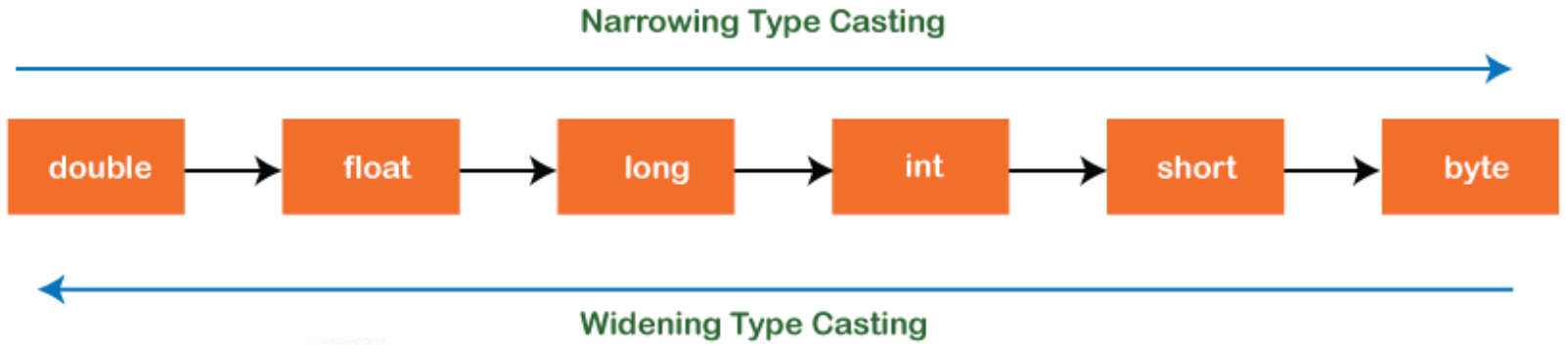
Açık tür dönüştürme/Implicit Type Conversion

Kullanıcı verileri bir türden diğerine manuel olarak değiştirdiğinde, bu açık dönüştürme olarak bilinir . Bu tür dönüştürme, tür dökümü (type casting) olarak da bilinir.

Java'da tür dökümü , bir veri türünü manuel ve otomatik olarak her iki şekilde başka bir veri türüne dönüştüren bir yöntem veya işlemdir.

İki tür döküm türü vardır:

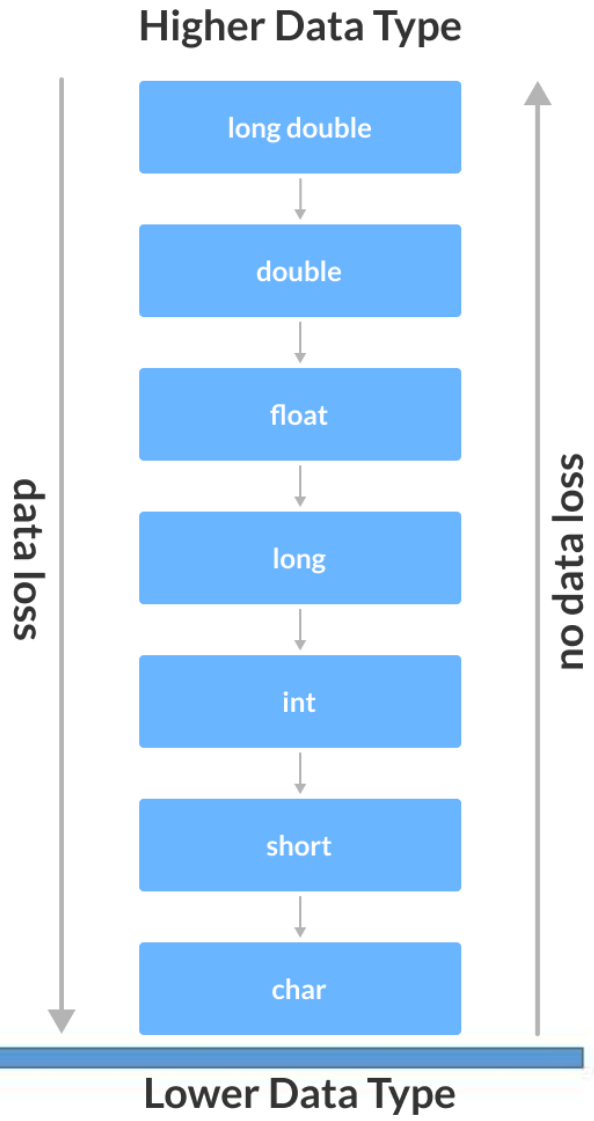
1. Genişletme Tipi Döküm
2. Daralan Tip Döküm



Tür Dökümü/ Type Casting

1. Genişletme Tipi Tür Dökümü (Upcasting)
2. Daralan Tipi Tür Dökümü (Downcasting)

Daraltma dönüştürmesi, bir değeri, orijinal türün tüm değerlerinin yaklaşık değerlerini bile depolayamayan bir türe dönüştürür. Örneğin, Java'da bir double'ı "float"a dönüştürmek daraltıcı bir dönüşümdür, çünkü double aralığı kayan noktanınkinden çok daha büyüktür. Genişleyen bir dönüştürme, bir değeri, orijinal türün tüm değerlerinin en azından yaklaşık değerlerini içerebilen bir türe dönüştürür. Örneğin, Java'da bir int'yi bir kayan nokta değerine dönüştürmek, bir genişletme dönüşümüdür



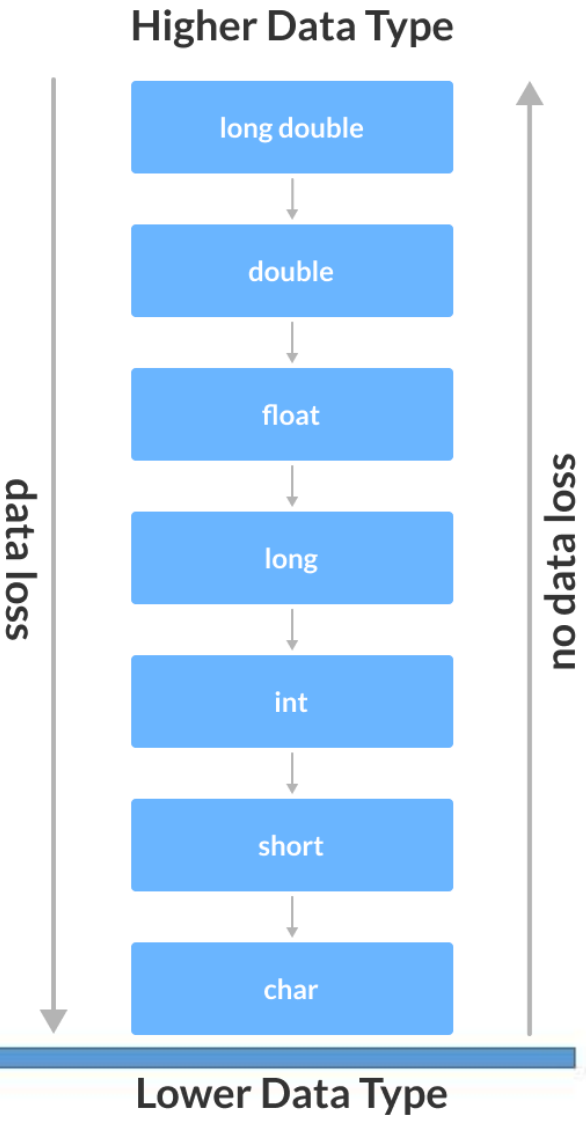
Tür Dökümü/ Type Casting

Dönüştürme Sırasında Veri Kaybı

Genişletme dönüşümleri neredeyse her zaman güvenlidir, yani dönüştürülen değerin yaklaşık büyüklüğü korunur. Dönüşümleri daraltmak her zaman güvenli değildir; bazen süreçte dönüştürülen değerin büyüklüğü değişir.

```
double doubleType1 = 10.99;
System.out.println("The double value: " + doubleType1);
// convert into int type
int intType1 = (int)doubleType1;
System.out.println("The integer value: " + intType1);
```

```
The double value: 10.99
The integer value: 10
```



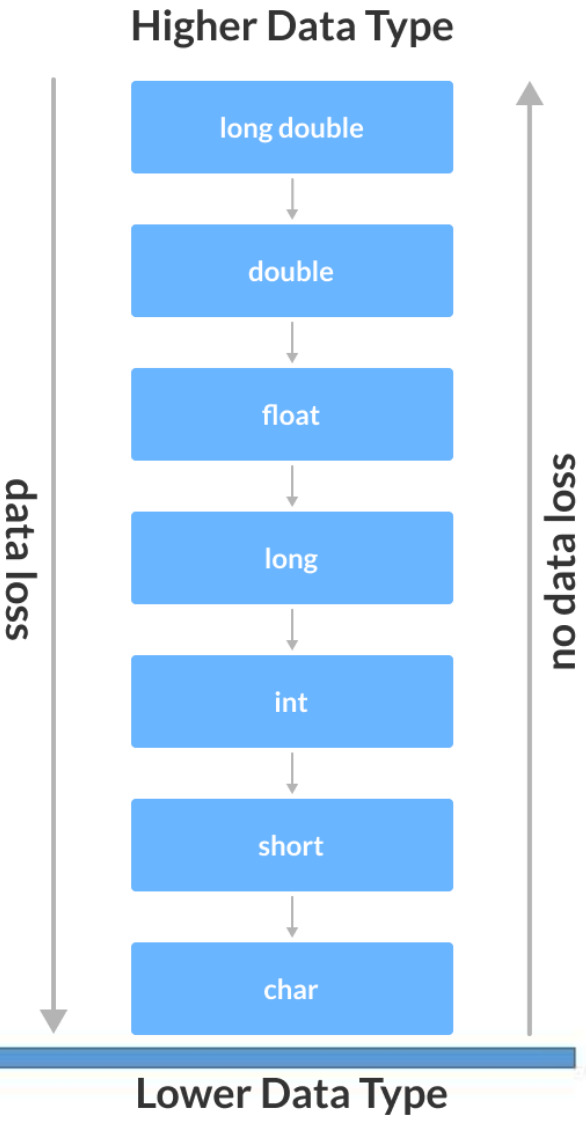
Tür Dökümü/ Type Casting

Dönüştürme Sırasında Veri Kaybı

Genişletme dönüşümleri neredeyse her zaman güvenlidir, yani dönüştürülen değerin yaklaşık büyüklüğü korunur. Dönüşümleri daraltmak her zaman güvenli değildir; bazen süreçte dönüştürülen değerin büyüklüğü değişir.

```
double doubleType1 = 10.99;
System.out.println("The double value: " + doubleType1);
// convert into int type
int intType1 = (int)doubleType1;
System.out.println("The integer value: " + intType1);
```

```
The double value: 10.99
The integer value: 10
```

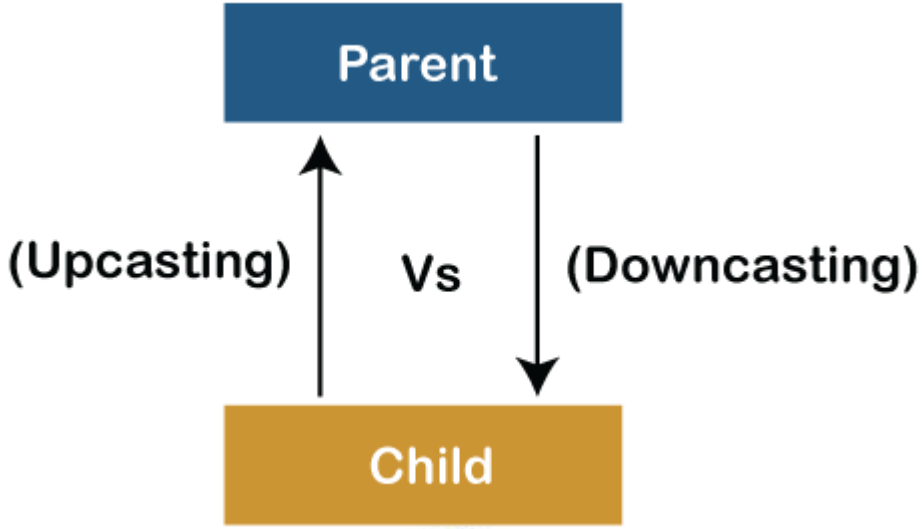


Tür Dökümü/ Type Casting

Sınıf Tipi Tür dökümü

Sınıf tipi döküm yapmak için aşağıdaki iki kuralı takip etmeliyiz:

1. Sınıflar “IS-A-Relationship” olmalıdır
2. Bir nesne, içinde yayınlayacağı bir sınıfın özelliğine sahip olmalıdır.



Tür Dökümü/ Type Casting

Sınıf Tipi Tür dökümü

Genişletme Tipi Tür Dökümü (Upcasting)

Kalıtım ağacına yukarı yönde bir süper türe bir alt tür atmaktır. Bir alt sınıf nesnesine bir üst sınıf referans değişkeni tarafından atıfta bulunulduğunda, bunu yapmak için hiçbir çabanın harcanmadığı otomatik bir prosedürdür. Bunu dinamik polimorfizm ile ilişkilendirebiliriz.

```
// Implicit upcasting in Java (polymorphism)
parent obj = new child(10,20);
// Calling the show() method to execute
obj.show();
```

```
run:
Child show method is called:10,20
BUILD SUCCESSFUL (total time: 0 seconds)
```

Tür Dökümü/ Type Casting

Sınıf Tipi Tür dökümü

Daralma Tipi Tür Dökümü (Downcasting)

Aşağıya yayın, alt sınıf türü, ana sınıfın nesnesine atıfta bulunduğunda, aşağı yayın olarak bilindiği prosedürü ifade eder. Doğrudan gerçekleştirilirse, çalışma zamanında ClassCastException oluşturulduğu için derleyici bir hata verir. Bu, yalnızca instanceof operatörünün kullanımıyla elde edilebilir. Halihazırda yukarıya yayınlanmış olan nesne, bu nesne yalnızca aşağı yayın gerçekleştirilebilir.

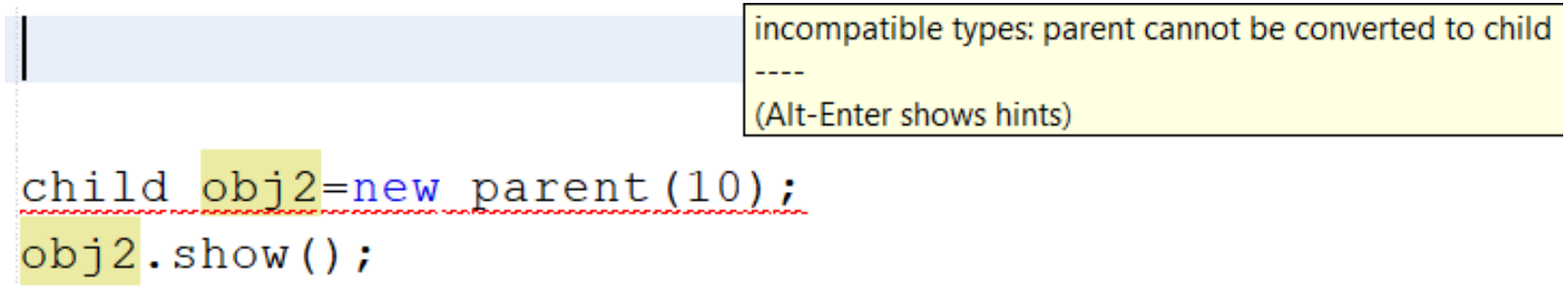
```
incompatible types: parent cannot be converted to child
----
(Alt-Enter shows hints)
```

```
child obj2=new parent(10);
obj2.show();
```

Tür Dökümü/ Type Casting

Sınıf Tipi Tür dökümü

Daralma Tipi Tür Dökümü (Downcasting)

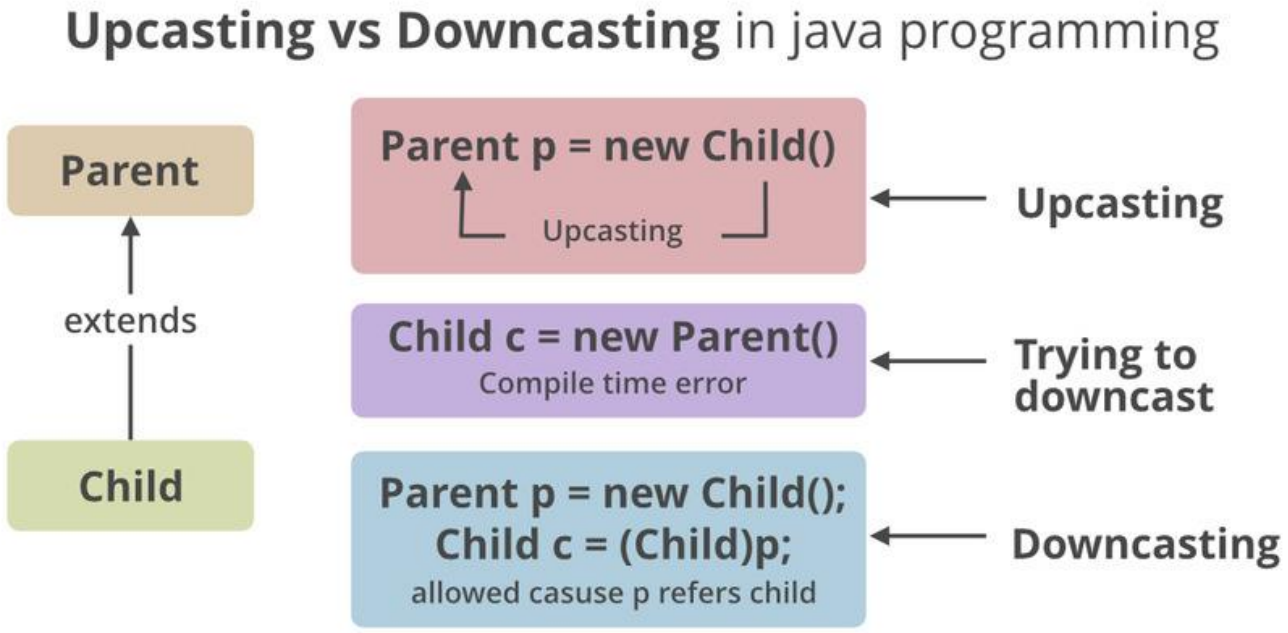


Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - Erroneous tree type: <any>
at javaclasstypecasting.JavaClassTypeCasting.main(JavaClassTypeCasting.java:25)
C:\Users\Önder\AppData\Local\NetBeans\Cache\12.6\executor-snippets\run.xml:111: The following error occurred while executing this line:
C:\Users\Önder\AppData\Local\NetBeans\Cache\12.6\executor-snippets\run.xml:68: Java returned: 1

Sınıf Tipi Tür dökümü

Daralma Tipi Tür Dökümü (Downcasting)

```
//Explicit Downcasting
parent obj1=new child(10,20);
child obj2=(child)obj1;
obj2.show();
```



Sınıf Tipi Tür dökümü

Daralma Tipi Tür Dökümü (Downcasting)

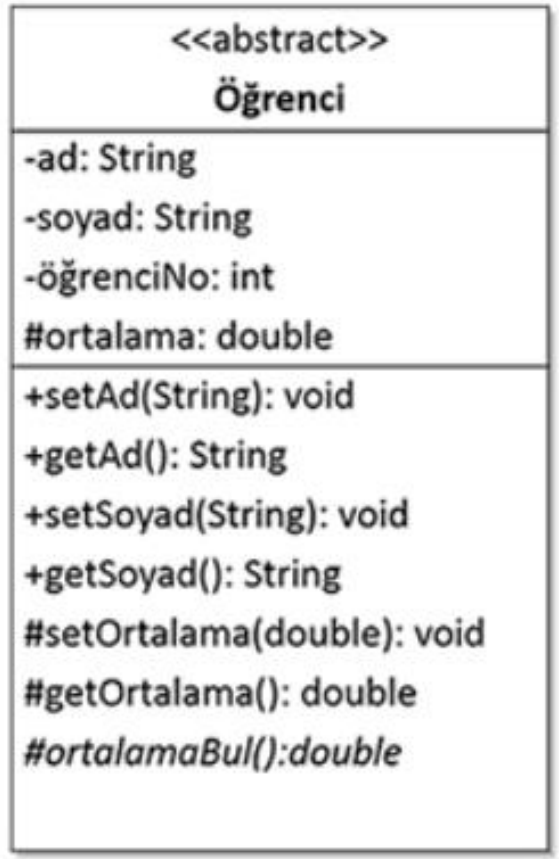
Java instanceof operatörü , nesnenin belirtilen tipte (sınıf veya alt sınıf veya arayüz) bir örnek olup olmadığını test etmek için kullanılır.

Java'daki instanceof, örneği türle karşılaştırdığı için tür karşılaştırma operatörü olarak da bilinir . Doğru veya yanlış döndürür. Instanceof operatörünü null değeri olan herhangi bir değişkene uygularsak, false döndürür.

```
//Explicit Downcasting
parent obj1=new child(10,20);
System.out.println(obj1 instanceof child);
System.out.println(obj1 instanceof parent);
if(obj1 instanceof child){
    child obj2=(child)obj1;
    obj2.show();
}
```

Java Soyut Sınıfı ve Soyut Yöntemler

- Bazı sınıfların **soyut sınıf (abstract class)** olarak tanımlanmaları daha anlamlıdır.
- Soyut sınıflardan nesne türetilemez. Bunlar üst sınıflardır ve alt sınıflar için genelleştirme yapmayı sağlarlar. Örneğin yukarıda verilen Öğrenci sınıfı, soyut sınıf olmaya aday bir sınıftır. Şayet Öğrenci sınıfından doğrudan nesne üretilmeyecekse, öğrenci nesneleri sadece yüksek lisans ve doktora öğrencileri olabiliyorlarsa bu durumda Öğrenci sınıfı soyut sınıf olarak tanımlanabilir.
- UML'de soyut sınıflar, sınıf ismi önüne <> sözcüğü eklenerek gösterilir.



Java Soyut Sınıfı ve Soyut Yöntemler

- Nesne yönelimli programlamada da soyutlama, uygulama detaylarını kullanıcıdan gizleme sürecidir, kullanıcıya sadece işlevsellik sağlanacaktır. Başka bir deyişle, kullanıcı nesnenin nasıl yaptığı yerine ne yaptığı hakkında bilgiye sahip olacaktır.

- Java'da soyutlama, Soyut sınıflar ve arayüzler kullanılarak gerçekleştirilir.

```
// create an abstract class
abstract class Language {
  // fields and methods
}
...

// try to create an object Language
// throws an error
Language obj = new Language();
```

Java Soyut Sınıfı ve Soyut Yöntemler

- Soyut bir sınıf, hem normal yöntemlere hem de soyut yöntemlere sahip olabilir.,

```
abstract class Language [

  // abstract method
  abstract void method1();

  // regular method
  void method2() {
    System.out.println("This is regular method");
  }
}
```

Java Soyut Sınıfı ve Soyut Yöntemler

Bildiriminde **abstract** anahtar sözcüğünü içeren bir sınıf , soyut sınıf olarak bilinir.

- Soyut sınıflar soyut yöntemler içerebilir veya içermeyebilir , yani gövdesiz yöntemler (public void get ();)
- Ancak, bir sınıfın en az bir soyut yöntemi varsa, o zaman sınıfın soyut olarak bildirilmesi gerekir .
- Bir sınıf soyut olarak bildirilmişse, somutlaştırılamaz.
- Soyut bir sınıfı kullanmak için, onu başka bir sınıftan miras almanız, içindeki soyut yöntemlere uygulamalar sağlamanız gerekir.
- Soyut bir sınıfı miras alırsanız, içindeki tüm soyut yöntemlere uygulamalar sağlamanız gerekir.

Java Soyut Sınıfı ve Soyut Yöntemler

Soyut Yöntemler

Bir sınıfın belirli bir yöntemi içermesini istiyorsanız, ancak bu yöntemin gerçek uygulamasının alt sınıflar tarafından belirlenmesini istiyorsanız, yöntemi ana sınıfta bir soyut olarak bildirebilirsiniz.

abstract anahtar sözcüğü, yöntemi soyut olarak bildirmek için kullanılır.

Sen yerleştirmek zorunda soyut yöntem bildiriminde yöntem adından önce anahtar kelime.

Soyut bir yöntem, bir yöntem imzası içerir ancak yöntem gövdesi içermez.

Küme parantezleri yerine, soyut bir yöntemin sonunda bir semoi iki nokta üst üste (;) olacaktır.

Java Soyut Sınıfı ve Soyut Yöntemler

Soyut Yöntemler

```
public abstract class Employee {
   private String name;
   private String address;
   private int number;

   public abstract double computePay();
   // Remainder of class definition
}
```

```
/* File name : Salary.java */
public class Salary extends Employee {
   private double salary;   // Annual salary

   public double computePay() {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }
   // Remainder of class definition
}
```

Java Soyut Sınıfı ve Soyut Yöntemler

- abstract Anahtar kelimeyi soyut sınıflar ve yöntemler oluşturmak için kullanırız .
- Soyut bir yöntemin herhangi bir uygulaması yoktur (yöntem gövdesi).
- Soyut yöntemler içeren bir sınıf da soyut olmalıdır.
- Soyut bir sınıfın nesnelerini oluşturamayız.
- Soyut bir sınıfın özelliklerini uygulamak için, ondan alt sınıfları miras alır ve alt sınıfın nesnelerini yaratırız.
- Bir alt sınıf, soyut bir sınıfın tüm soyut yöntemlerini geçersiz kılmalıdır. Ancak, alt sınıf soyut olarak bildirilirse, soyut yöntemleri geçersiz kılmak zorunlu değildir.
- Soyut sınıfın referansını kullanarak soyut bir sınıfın statik niteliklerine ve yöntemlerine erişebiliriz.

Polimorfizm

- Polimorfizm, bir nesnenin birçok form alma yeteneğidir. OOP'de polimorfizmin en yaygın kullanımı, bir alt sınıf nesnesine başvurmak için bir ana sınıf başvurusu kullanıldığında ortaya çıkar.

- Birden fazla IS-A testini geçebilen herhangi bir Java nesnesi polimorfik olarak kabul edilir. Java'da, herhangi bir nesne kendi türü ve Object sınıfı için IS-A testini geçeceğinden tüm Java nesneleri çok biçimlidir.

- Bir nesneye erişmenin tek olası yolunun bir referans değişkeni aracılığıyla olduğunu bilmek önemlidir. Bir referans değişkeni yalnızca tek tipte olabilir. Bir kez bildirildikten sonra, bir referans değişkeninin türü değiştirilemez.

- Referans değişkeni, nihai olarak bildirilmemesi koşuluyla diğer nesnelere yeniden atanabilir. Referans değişkeninin türü, nesnede başlatabileceği yöntemleri belirleyecektir.

- Bir referans değişkeni, bildirilen türündeki herhangi bir nesneye veya bildirilen türünün herhangi bir alt türüne başvurabilir. Bir referans değişkeni, bir sınıf veya arabirim türü olarak bildirilebilir.

Polimorfizm

Java'da polimorfizmi aşağıdaki yollarla elde edebiliriz:

1. Yöntemi Geçersiz Kılma (Method overriding)
2. Yöntem Aşırı Yükleme (Method overloading)
3. Operatör Aşırı Yüklemesi (Operator overloading)

Aggregation

Bir sınıfın bir varlık başvurusu varsa, bu Toplama olarak bilinir. Toplama, HAS-A ilişkisini temsil eder.

Bir durumu düşünün, Çalışan nesnesi id, ad, e-posta kimliği vb. Birçok bilgiyi içerir. Aşağıda verilen şehir, eyalet, ülke, posta kodu vb. Kendi bilgilerini içeren adres adlı bir nesne daha içerir.

```
class  Employee {
int  id;
Dize adı;
Adres adresi; // Adres bir sınıftır
...
}
```

Method Overridding

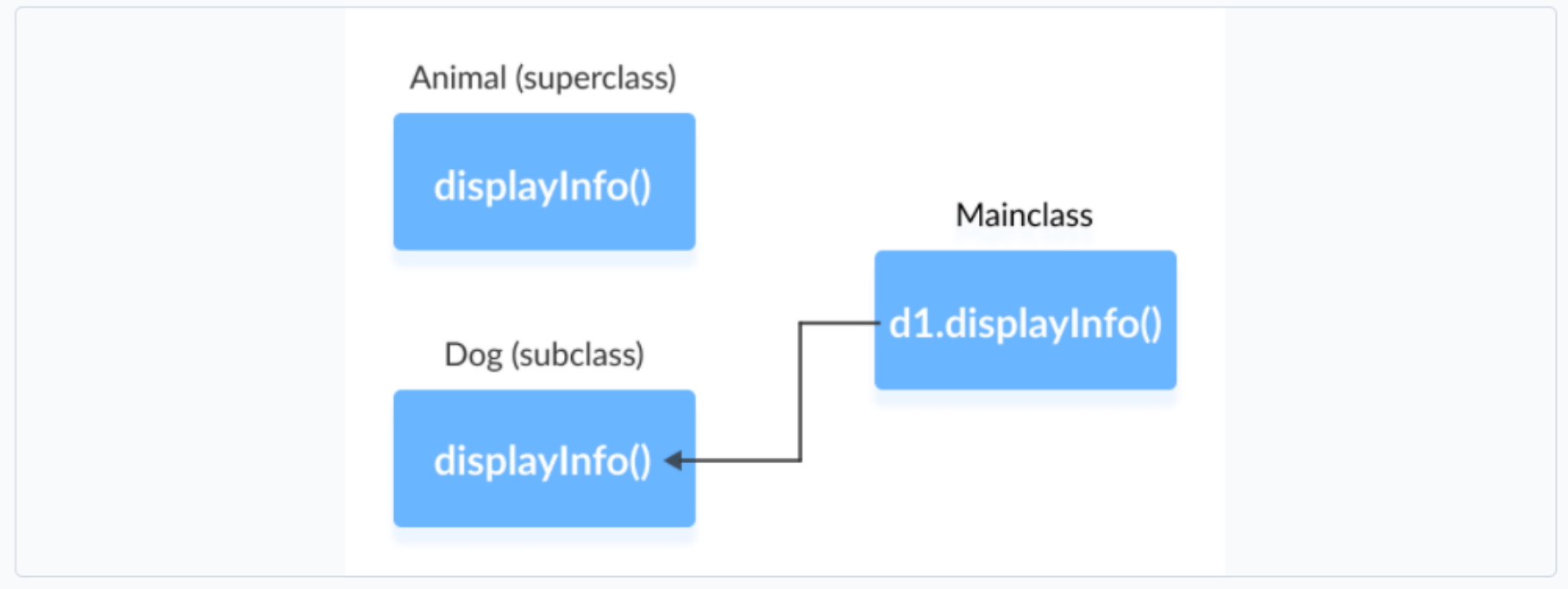
kalıtımı öğrendik. Kalıtım, mevcut bir sınıftan (üst sınıf) yeni bir sınıf (alt sınıf) türetmemize izin veren bir OOP özelliğidir. Alt sınıf, üst sınıfın niteliklerini ve yöntemlerini miras alır.

Şimdi, aynı yöntem hem üst sınıf sınıfında hem de alt sınıf sınıfında tanımlanırsa, alt sınıf sınıfının yöntemi, üst sınıfın yöntemini geçersiz kılar. Bu, yöntemi geçersiz kılma olarak bilinir.

Method Overridding

```
class Animal {
    public void displayInfo() {
        System.out.println("I am an animal.");
    }
}

class Dog extends Animal {
    @Override
    public void displayInfo() {
        System.out.println("I am a dog.");
    }
}

class Main {
    public static void main(String[] args) {
        Dog d1 = new Dog();
        d1.displayInfo();
    }
}
```

Method Overridding



Method Overridding

Yöntemi Geçersiz Kılma Kuralları

- Bağımsız değişken listesi, geçersiz kılınan yöntemle tamamen aynı olmalıdır.
- Dönüş türü, üst sınıftaki orijinal geçersiz kılınan yöntemde bildirilen dönüş türünün aynısı veya alt türü olmalıdır.
- Erişim düzeyi, geçersiz kılınan yöntemin erişim düzeyinden daha kısıtlayıcı olamaz. Örneğin: Üst sınıf yöntemi genel olarak bildirilirse, alt sınıftaki geçersiz kılma yöntemi özel veya korumalı olamaz.
- Örnek yöntemleri, yalnızca alt sınıf tarafından miras alınmışlarsa geçersiz kılınabilir.
- Son olarak belirtilen bir yöntem geçersiz kılınamaz.
- Statik olarak bildirilen bir yöntem geçersiz kılınamaz, ancak yeniden bildirilebilir.
- Bir yöntem miras alınamıyorsa, geçersiz kılınamaz.

Method Overridding

Yöntemi Geçersiz Kılma Kuralları

- Örneğin üst sınıfıyla aynı pakette bulunan bir alt sınıf, özel veya son olarak bildirilmemiş herhangi bir üst sınıf yöntemini geçersiz kılabilir.
- Farklı bir paketteki bir alt sınıf, yalnızca genel veya korumalı olarak ilan edilen nihai olmayan yöntemleri geçersiz kılabilir.
- Geçersiz kılan bir yöntem, geçersiz kılınan yöntemin istisnalar atıp atmadığına bakılmaksızın, herhangi bir onaysız istisna atabilir. Ancak, geçersiz kılma yöntemi, geçersiz kılınan yöntem tarafından bildirilenlerden yeni veya daha geniş olan kontrol edilmiş istisnaları atmamalıdır. Geçersiz kılma yöntemi, geçersiz kılınan yönteme göre daha dar veya daha az istisna atabilir.
- Oluşturucular geçersiz kılınamaz.

Java Geçersiz Kılmada «super» Anahtar Kelimesi

```
class Animal {
   public void displayInfo() {
      System.out.println("I am an animal.");
   }
}

class Dog extends Animal {
   public void displayInfo() {
      super.displayInfo();
      System.out.println("I am a dog.");
   }
}

class Main {
   public static void main(String[] args) {
      Dog d1 = new Dog();
      d1.displayInfo();
   }
}
```

Yöntemi Geçersiz Kılmadaki Tanımlayıcılara Erişim

- Üst sınıfta ve alt sınıflarında bildirilen aynı yöntemin farklı erişim belirleyicileri olabilir. Ancak bir kısıtlama var.

- Bu erişim tanımlayıcılarını yalnızca üst sınıfın erişim tanımlayıcısından daha büyük erişim sağlayan alt sınıflarda kullanabiliriz. Örneğin,

- Üst sınıftaki bir yöntemin myClass() bildirildiğini varsayalım protected. Daha sonra, myClass()alt sınıftaki aynı yöntem ya public da olabilir protected, ama private olamaz

Java Soyut Sınıfı ve Soyut Yöntemler

- nesne yönelimli programlamada da soyutlama, uygulama detaylarını kullanıcıdan gizleme sürecidir, kullanıcıya sadece işlevsellik sağlanacaktır. Başka bir deyişle, kullanıcı nesnenin nasıl yaptığı yerine ne yaptığı hakkında bilgiye sahip olacaktır.

- Java'da soyutlama, Soyut sınıflar ve arayüzler kullanılarak gerçekleştirilir.

```
// create an abstract class
abstract class Language {
  // fields and methods
}
...

// try to create an object Language
// throws an error
Language obj = new Language();
```

Java Soyut Sınıfı ve Soyut Yöntemler

- Soyut bir sınıf, hem normal yöntemlere hem de soyut yöntemlere sahip olabilir.,

```
abstract class Language [

  // abstract method
  abstract void method1();

  // regular method
  void method2() {
    System.out.println("This is regular method");
  }
}
```

Java Soyut Sınıfı ve Soyut Yöntemler

Bildiriminde abstract anahtar sözcüğünü içeren bir sınıf , soyut sınıf olarak bilinir.

- Soyut sınıflar soyut yöntemler içerebilir veya içermeyebilir , yani gövdesiz yöntemler (public void get ();)
- Ancak, bir sınıfın en az bir soyut yöntemi varsa, o zaman sınıfın soyut olarak bildirilmesi gerekir .
- Bir sınıf soyut olarak bildirilmişse, somutlaştırılamaz.
- Soyut bir sınıfı kullanmak için, onu başka bir sınıftan miras almanız, içindeki soyut yöntemlere uygulamalar sağlamanız gerekir.
- Soyut bir sınıfı miras alırsanız, içindeki tüm soyut yöntemlere uygulamalar sağlamanız gerekir.

Java Soyut Sınıfı ve Soyut Yöntemler

Soyut Yöntemler

Bir sınıfın belirli bir yöntemi içermesini istiyorsanız, ancak bu yöntemin gerçek uygulamasının alt sınıflar tarafından belirlenmesini istiyorsanız, yöntemi ana sınıfta bir soyut olarak bildirebilirsiniz.

abstract anahtar sözcüğü, yöntemi soyut olarak bildirmek için kullanılır.

Sen yerleştirmek zorunda soyut yöntem bildiriminde yöntem adından önce anahtar kelime.

Soyut bir yöntem, bir yöntem imzası içerir ancak yöntem gövdesi içermez.

Küme parantezleri yerine, soyut bir yöntemin sonunda bir semoi iki nokta üst üste (;) olacaktır.

Java Soyut Sınıfı ve Soyut Yöntemler

Soyut Yöntemler

```
public abstract class Employee {
   private String name;
   private String address;
   private int number;

   public abstract double computePay();
   // Remainder of class definition
}
```

```
/* File name : Salary.java */
public class Salary extends Employee {
   private double salary;   // Annual salary

   public double computePay() {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }
   // Remainder of class definition
}
```

Encapsulation- Kapsülleme

Kapsülleme, dört temel OOP konseptinden biridir. Diğer üçü kalıtım, çok biçimlilik ve soyutlamadır.

Java'da kapsülleme, verileri (değişkenleri) ve verileri (yöntemleri) tek bir birim olarak birlikte hareket eden kodu sarma mekanizmasıdır. Kapsüllemede, bir sınıfın değişkenleri diğer sınıflardan gizlenir ve bunlara yalnızca mevcut sınıflarının yöntemleri aracılığıyla erişilebilir. Bu nedenle, veri gizleme olarak da bilinir.

Java'da kapsülleme elde etmek için -

- Bir sınıfın değişkenlerini özel olarak bildirilir.
- Değişken değerlerini değiştirmek ve görüntülemek için genel ayarlayıcı ve alıcı yöntemleri sağlanır.

Encapsulation- Kapsülleme

- Kapsülleme, ilgili alanların ve yöntemlerin bir araya toplanması anlamına gelir. Bu, veri gizlemeyi sağlamak için kullanılabilir. Kapsülleme kendi başına veri saklama değildir.
- Java'da kapsülleme, ilgili alanları ve yöntemleri bir arada tutmamıza yardımcı olur, bu da kodumuzu daha temiz ve okunması kolay hale getirir.
- Veri alanlarımızın değerlerini kontrol etmeye yardımcı olur.
- Bir sınıfın alanları salt okunur veya salt okunur hale getirilebilir.
- Bir sınıf, alanlarında depolananlar üzerinde tam kontrole sahip olabilir.

Encapsulation- Kapsülleme

```
class Person {
  private int age;

  public void setAge(int age) {
    if (age >= 0) {
      this.age = age;
    }
  }
}
```

Erişim Düzenleyiciler

- **Private**: Bir özel değiştiricinin erişim düzeyi yalnızca sınıfın içindedir. Sınıf dışından erişilemez.
- **Default**: Varsayılan bir değiştiricinin erişim düzeyi yalnızca paketin içindedir. Paket dışından erişilemez. Herhangi bir erişim seviyesi belirtmezseniz, bu varsayılan olacaktır.
- **Protected**: Korumalı bir değiştiricinin erişim düzeyi paketin içinde ve alt sınıf aracılığıyla paketin dışındadır. Alt sınıf yapmazsanız paket dışından erişilemez.
- **Public**: Bir genel değiştiricinin erişim düzeyi her yerdedir. Sınıf içinden, sınıf dışından, paket içinden ve paket dışından erişilebilir

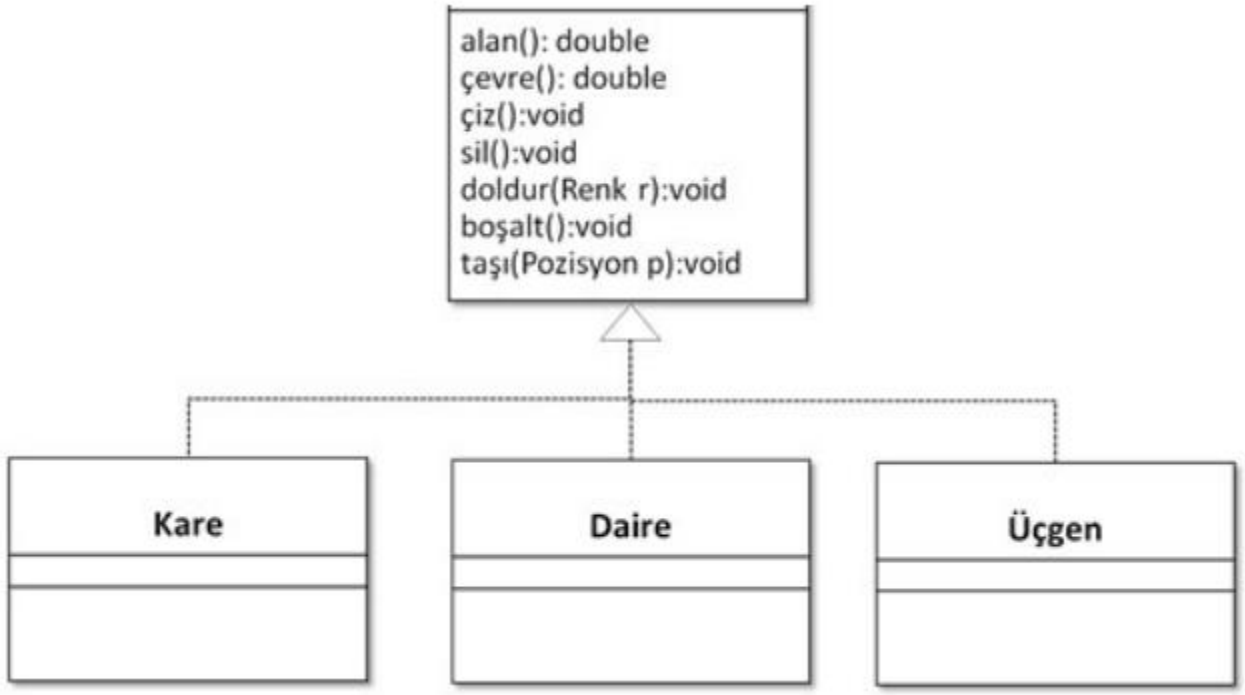
Erişim Düzenleyiciler

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Interfaces-Arayüzler

- Arayüz, gövdesi olmayan bir grup yöntem içeren tamamen soyut bir sınıftır.
- Arayüz, Java'da bir referans türüdür. Sınıfa benzer. Soyut yöntemler koleksiyonudur. Bir sınıf bir arabirim uygular, dolayısıyla arabirimin soyut yöntemlerini miras alır.
- Soyut yöntemlerin yanı sıra, bir arabirim ayrıca sabitler, varsayılan yöntemler, statik yöntemler ve iç içe türler içerebilir. Yöntem gövdeleri yalnızca varsayılan yöntemler ve statik yöntemler için mevcuttur.
- Arayüz yazmak, sınıf yazmaya benzer. Ancak bir sınıf, bir nesnenin niteliklerini ve davranışlarını tanımlar. Ve bir arayüz, bir sınıfın uyguladığı davranışları içerir.
- Arabirimi uygulayan sınıf soyut olmadığı sürece, arabirimin tüm yöntemlerinin sınıfta tanımlanması gerekir.

Interfaces-Arayüzler



Interfaces-Arayüzler

```
interface <interface_name>{

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

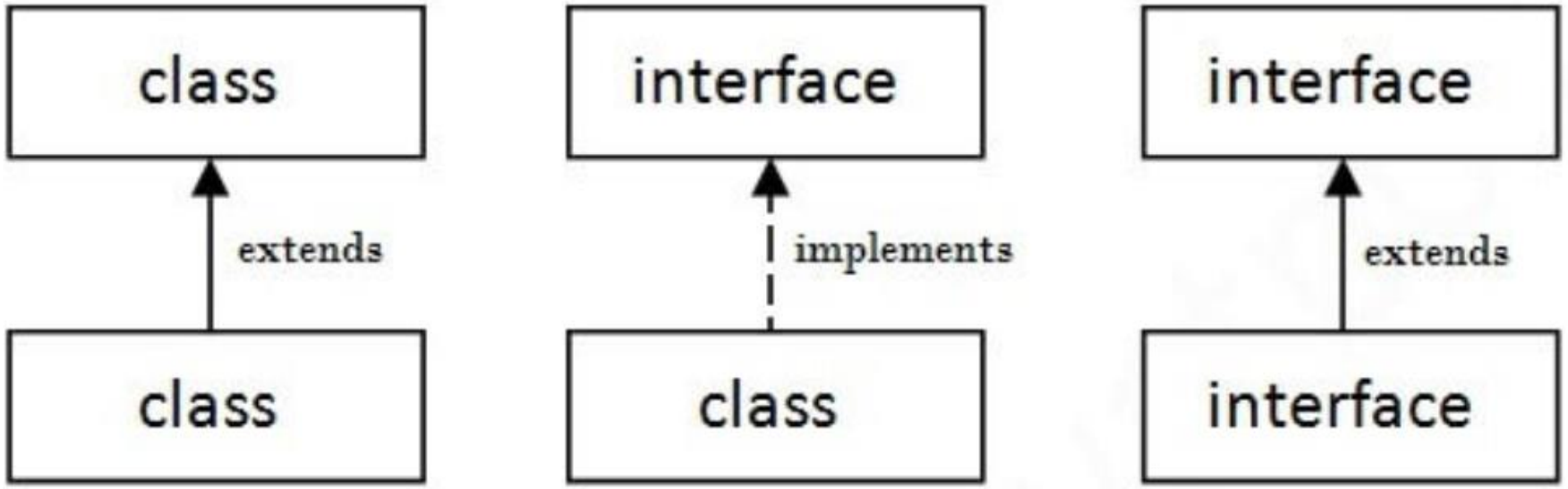
- Bir arayüz, herhangi bir sayıda yöntem içerebilir.
- Bir arayüz örtük olarak soyuttur. Bir arayüz bildirirken abstract anahtar sözcüğünü kullanmanıza gerek yoktur.
- Bir arayüzdeki her yöntem de dolaylı olarak soyuttur, bu nedenle soyut anahtar sözcüğe gerek yoktur.
- Bir arayüzdeki yöntemler örtük olarak geneldir.

Interfaces-Arayüzler

Bir arayüz, bir sınıftan birkaç yönden farklıdır:

- Bir arabirimi örnekleyemezsiniz.
- Bir arayüz herhangi bir kurucu içermez.
- Bir arayüzdeki tüm yöntemler soyuttur.
- Bir arayüz, örnek alanları içeremez. Bir arabirimde görünebilecek alanların hem statik hem de son olarak bildirilmesi gerekir.
- Bir arayüz bir sınıf tarafından genişletilmez; bir sınıf tarafından uygulanmaktadır.
- Bir arayüz birden çok arayüzü genişletebilir.

Interfaces-Arayüzler



Interfaces-Arayüzler

```
interface A {
  // members of A
}

interface B {
  // members of B
}

class C implements A, B {
  // abstract members of A
  // abstract members of B
}
```

Interfaces-Arayüzler

```
interface A {
    ...
}
interface B {
    ...
}

interface C extends A, B {
    ...
}
```

Interfaces-Arayüzler- default motodlar

```
public default void getSides() {
   // body of getSides()
}
```

```
interface Drawable{
void draw();
default void msg(){System.out.println("default method");}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class TestInterfaceDefault{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
d.msg();
}}
```

Interfaces-Arayüzler- Static motodlar

```
interface Drawable{
void draw();
static int cube(int x){return x*x*x;}
}
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}

class TestInterfaceStatic{
public static void main(String args[]){
Drawable d=new Rectangle();
d.draw();
System.out.println(Drawable.cube(3));
}}
```

Interfaces-Arayüzler- Nested Interface

```
interface printable{
 void print();
 interface MessagePrintable{
   void msg();
 }
}
```