

Flutter ile Uygulama Geliştirme Kursu | Android & IOS

Nesne Tabanlı Programlama

Kasım ADALAN

Elektronik ve Haberleşme Mühendisi

Android - IOS Developer and Trainer

Eğitim İçeriği

1. Sınıf (Class ve Structure) Nedir ?
2. Nesne (Object) Nedir ?
3. Fonksiyonlar - Metodlar
4. Initialization – Constructor
 - Shadowing (Gölgeleme)
5. Nullable
6. Import
7. Visibility Modifier
8. Static değişken & metodlar
9. Enumeration
9. Composition
10. Kalıtım
11. Kalıtım ilişkisinde Constructor
12. Metodları Ezme (Override)
13. PolyMorphism
14. Nesnelerin Tip Dönüşümü
15. Interface

Nesne Tabanlı Programlama

Nesne Tabanlı Programlama

- Hayatlarımız nesneler çevresinde kuruludur



- Bu nesneleri soyutlayarak yazılım projelerine yansıtırız
- Birden çok kez kullanım için nesneler soyutlanarak bilgisayar koduna dönüştülür
- Oluşan soyut taslaklara sınıf (class) denir

Sınıf (Class) Nedir ?

- Araba Analojisi

- Mühendisler yeni bir araba üretmek için öncelikle proje planları oluşturur
- Benzin emisyonu, motorun nasıl çalıştığı gibi ayrıntılar bu planlara yansıtılır
- Planlar arabanın nasıl hareket edeceği, arabayı oluşturacak parçalar gibi birçok detayı içerir
- Herhangi bir sürücünün tüm bu detayları bilmesine gerek var mıdır?
- Hayır! Yalnızca ehliyetinin olması ve arabayı sürmeyi bilmesi yeterlidir.

Nesne (Object) Nedir ?

- Nesneler sınıfların somutlaşmış halleridir
- Nesneleri durumu (state) ve davranış biçimleri vardır (behaviour)



Arabanın

-renk, hız, kapasite

-Hızlanmak ve

yavaşlamak için pedallar

} Durum (state)

} Davranış (behaviour)

- Sınıflar ise nesnelerin özellikleri ve davranışları ile ilgili ayrıntıları içerir

Araba sınıfı.

-Frene basıldığında ne olur?

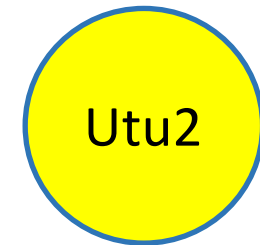
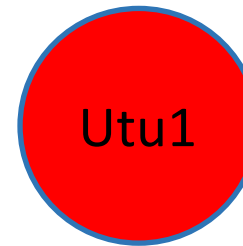
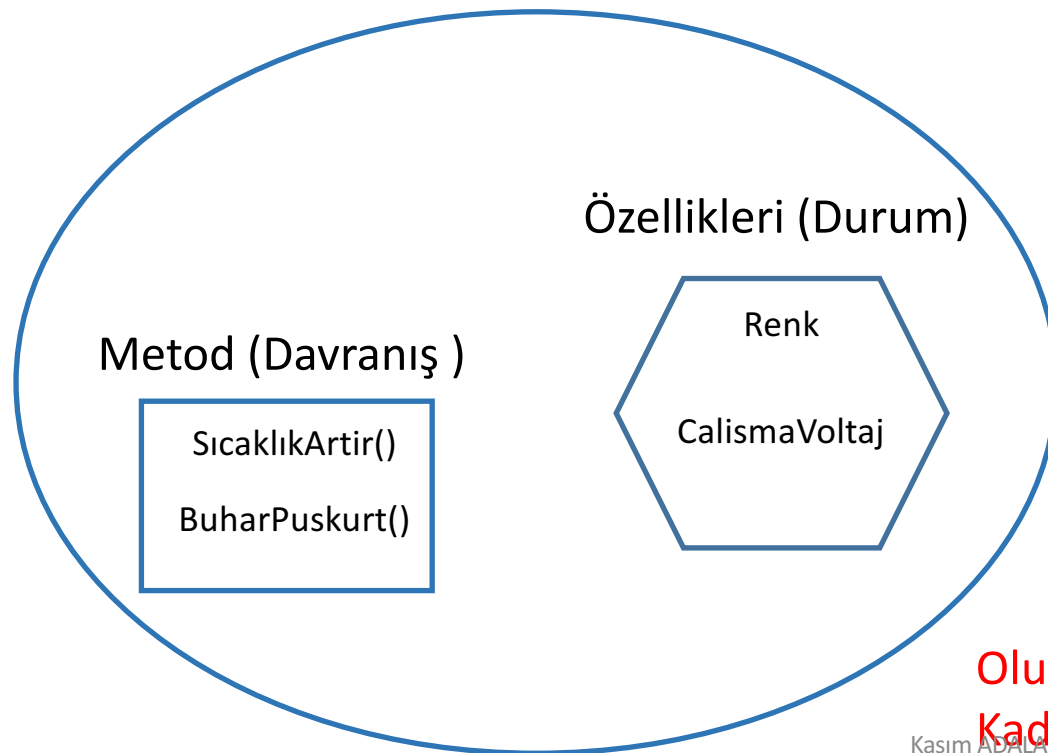


Nesnenin Durumu ve Davranışı

- **Durum (State)**: Sınıfların bir – birçok özelliği olabilir
 - Somut değişkenler (instance variables) belirler
 - Nesneyle birlikte taşınır
- **Davranış (Behavior)**: Sınıflar bir ya da birden çok metoda sahip olabilir
 - Metod program içindeki bir işi temsil eder
 - Görevlerin gerçekleştirileceği adımları tanımlar
 - Kullanıcıdan kompleks işlemleri gizler
 - Metodu çağırmak, metodun bu işlemleri gerçekleştirmesini sağlar

Nesne (Object)

Sınıf (Utu)



Oluşturduğumuz Ütü sınıfı taslağında istediğimiz
Kadar Nesne üretebiliriz.

Class Tanımlama


Keywords

İsim

```
class Araba{  
    }  
}
```

Nesne Tanımlama

```
class Araba{  
    late String renk ;  
    late int hiz;  
    late bool calisiyormu;  
}
```



late sınıf içinde değişkene değer aktarmadan değişken oluşturmamızı sağlar.

Nesnenin Adı	Nesnenin Oluşturulduğu Sınıf
<hr/>	<hr/>
var <i>bmw</i> =	Araba ();

Class Yapısına Erişim.

- Class yapısı içindeki metod ve özelliklerine erişmek için **ilk şart bulunduğu Class ' dan nesne oluşturmali.**

```
var bmw = Araba();
```

```
class Araba{  
    late String renk ;  
    late int hiz;  
    late bool calisiyormu;  
}
```

```
//Değer atama
```

```
bmw.renk = "Mavi";
```

```
bmw.hiz = 200;
```

```
bmw.calisiyormu = true;
```

```
//Değer Okuma
```

```
print(bmw.renk); //Mavi
```

```
print(bmw.hiz); //200
```

```
print(bmw.calisiyormu); //true
```

Class Yapısına Erişim.

- Her nesne kendine ait özelliklere erişebilir ve veri aktarımı yapabilir.

```
var bmw = Araba();
```

```
//Değer atama
```

```
bmw.renk = "Mavi";
```

```
bmw.hiz = 200;
```

```
bmw.calisiyormu = true;
```

```
//Değer Okuma
```

```
print(bmw.renk);//Mavi
```

```
print(bmw.hiz);//200
```

```
print(bmw.calisiyormu);//true
```

```
var limuzin = Araba();
```

```
//Değer atama
```

```
limuzin.renk = "Siyah";
```

```
limuzin.hiz = 0;
```

```
limuzin.calisiyormu = false;
```

```
//Değer Okuma
```

```
print(limuzin.renk);//Siyah
```

```
print(limuzin.hiz);//0
```

```
print(limuzin.calisiyormu);//false
```

Fonksiyonlar - Metodlar

Fonksiyonlar

- Belirli işlemleri temsil eden yapılardır.
- Kullanma amacımız tekrarlanan kod yapılarının önüne geçmektir.
- Programlamayı daha pratik bir hale getirir.
- Kodun okunmasına faydası vardır.

```
dönüş türü  fonksiyon adı  ( Parametre ) {  
  
    //Kodlama buraya yazılır  
  
    return dönüş değeri  
}
```

Class Metodlarına Erişim.

```
class Otobus{
    late int kapasite;
    late String nereden;
    late String nereye;
    late int mevcutYolcu;

    void yolcuAl(int yolcuSayisi){
        mevcutYolcu = mevcutYolcu + yolcuSayisi;
    }

    void yolcuIndir(int yolcuSayisi){
        mevcutYolcu = mevcutYolcu - yolcuSayisi;
    }

    void bilgiAl(){
        print("Kapasite : $kapasite");
        print("Nereden : $nereden");
        print("Nereye : $nereye");
        print("Mevcut Yolcu : $mevcutYolcu");
    }
}
```

```
var kamilKoc = Otobus();
```

```
kamilKoc.kapasite = 50;
kamilKoc.nerden = "Bursa";
kamilKoc.nereye = "Ankara";
kamilKoc.mevcutYolcu = 10;
```

```
kamilKoc.bilgiAl();
```

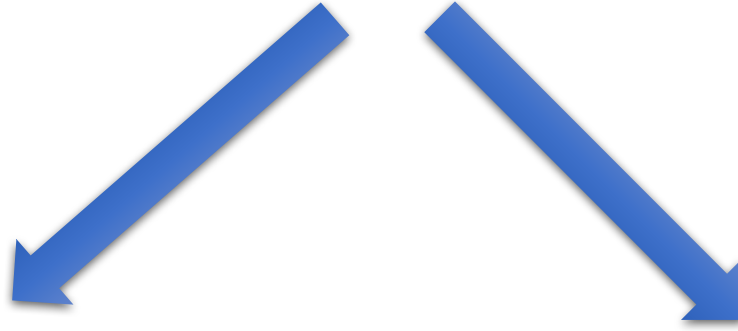
```
kamilKoc.yolcuIndir(5);
```

```
kamilKoc.bilgiAl();
```

```
kamilKoc.yolcuAl(20);
```

```
kamilKoc.bilgiAl();
```

Foksiyonlar



Geri dönüş değeri olan

Geri dönüş değeri olmayan

Geri Dönüş değeri olmayan fonksiyonlar

- Sadece yaptırılmak istenen işlemi yapan metodu kullanana veri döndürmeyen fonksiyonlardır.

```
void selamla(){  
    String sonuc = "Merhaba ahmet";  
    print(sonuc);  
}
```

```
selamla();
```

Geri Dönüş değeri olan fonksiyonlar

- Yapılan işlem sonucunda metodu kullanana veri dönüşü yapan fonksiyonlardır.

```
String selamla1(){  
    String sonuc = "Merhaba ahmet";  
    return sonuc;  
}
```

```
String gelenSonuc = selamla1();  
print(gelenSonuc);
```

Fonksiyonların Parametre Alması

- Parametre fonksiyonlara dışarıdan verilen değerlerdir.
- Her fonksiyonun parametresi olmak zorunda değildir.
- Parametreler tanımlaması değişkeni tanımlar gibidir.
- Parametreler , virgül ile birden fazla tanımlanabilir. (parametre)

• **String** **isim**



parametre türü parametre adı

```
void selamla2(String isim){  
    String sonuc = "Merhaba $isim";  
    print(sonuc);  
}
```

```
selamla2("Zeynep");
```

Fonksiyonların Parametre Alması

- Birden fazla parametre kullanılabilir.

```
int toplama2(int sayi1,int sayi2) {  
    int toplam = sayi1 + sayi2;  
    return toplam;  
}
```

```
int t2 = toplama2(100,200);  
print("Toplama2 : $t2");
```

Nesne Tabanlı ÖDEVLER

Ödevler

1. Parametre olarak girilen dereceyi fahrenheit'a dönüştürdükten sonra geri döndüren bir metod yazınız. $T_{(^{\circ}\text{F})} = T_{(^{\circ}\text{C})} \times 1.8 + 32$
2. Kenarları parametre olarak girilen ve dikdörtgenin çevresini hesaplayan bir metod yazınız..
3. Parametre olarak girilen sayının faktoriyel değerini hesaplayıp geri döndüren metodu yazınız.
4. Parametre olarak girilen kelime ve harf için harfin kelime içinde kaç adet olduğunu gösteren bir metod yazınız.

Ödevler

5. Parametre olarak girilen kenar sayısına göre iç açılar toplamını hesaplayıp sonucu geri gönderen metod yazınız.

Formül n: kenar sayısı $(n-2).180$

6 . Parametre olarak girilen gün sayısına göre maaş hesabı yapan ve elde edilen değeri geri döndüren metod yazınız.

1 Günde 8 saat çalışılabilir.

Çalışma saat ücreti : 10 tl

Mesai saat ücreti : 20tl

160 saat üzeri mesai sayılır.

7. Parametre olarak girilen kota miktarına göre ücreti hesaplayarak geri döndüren metodu yazınız.

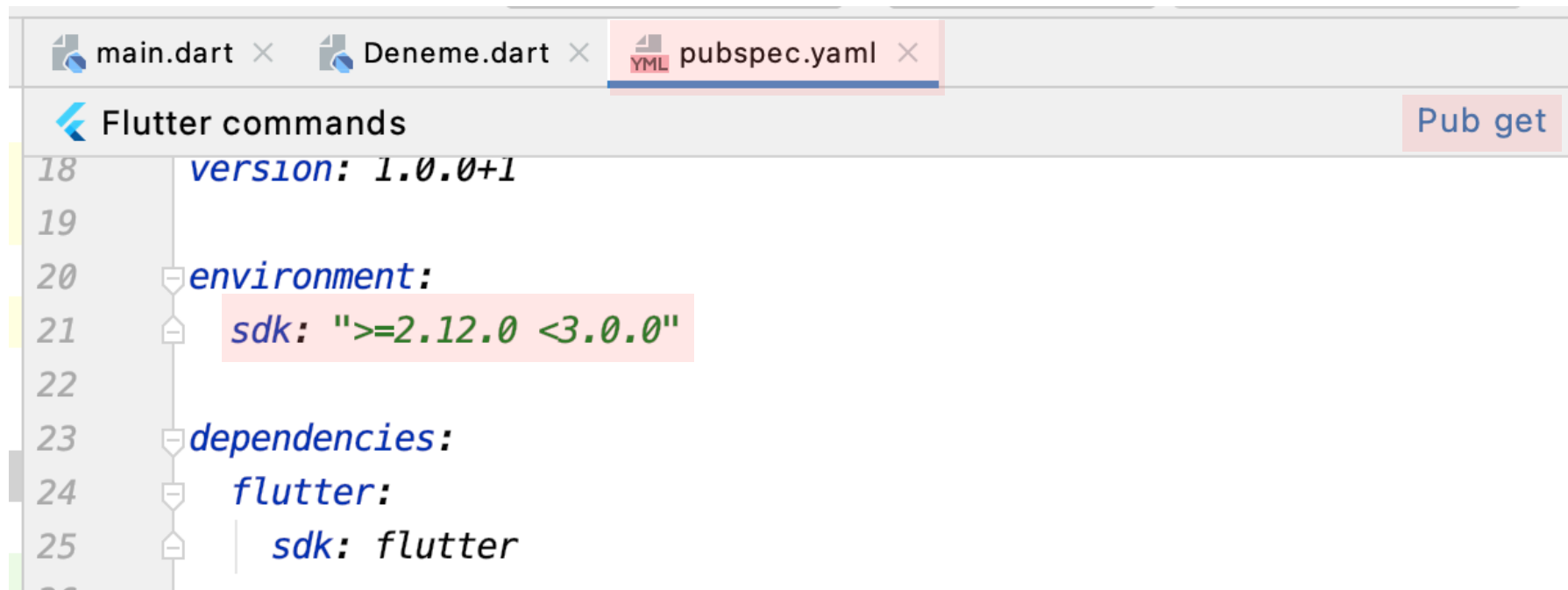
- 50GB 100 TL
- Kota aşımından sonra her 1GB 4 TL

Null Safety

Null Safety ?

- Kullandığımız değişkenler null olabilir.
- Eğer değişken null olabilirse bu noktada dikkatli olmalıyız.
- Uygulamaların çökmesi büyük oranda null olan değişkenlerden kaynaklanmaktadır.
- Null olabilecek değişken kullanımını daha kontrollü yapmak için null safety kullanılmaktadır.
- Mobil uygulama geliştirme için kullanılan bütün modern dillerde bu özellik yer almaktadır.
- Null Safety özelliğini kullanmak için değişken türünden sonra ? işareti kullanılır.
- Bu özelliğe sahip değişken daha sonra if ile null kontrolü yapılarak kullanılması önerilir.
- Null safety özelliği olan değişkenden sonra ! işareti kullanılırsa , null olmayacağını ben kodlamama güveniyorum demektir.
- Null safety özelliği olan değişkenden sonra ? kullanılırsa , null olduğunda hata oluşmaz korumaya alır , null olmaz ise normal çalışmasını gerçekleştirir.

Null Safety – Aktif Etme



The screenshot shows an IDE window with three tabs: `main.dart`, `Deneme.dart`, and `pubspec.yaml`. The `pubspec.yaml` tab is active and shows the following content:

```
18  version: 1.0.0+1
19
20  environment:
21    sdk: ">=2.12.0 <3.0.0"
22
23  dependencies:
24    flutter:
25      sdk: flutter
```

On the right side of the editor, there is a button labeled "Pub get".

2.12.0 üzeri sdk versiyonu olmalıdır.

//Tanımlama

//String isim = null; //Bir ifade null olacaksa nullable olmalıdır.

String? mesaj = null; //? işareti ile nullable yapılır.

mesaj = "Merhaba"; //Nullable ifadeye sonradan değer aktarılabilir.

//Tanımlama sonrası kullanım

String? isim = null;

//Yöntem 1 : ? null ise çökme olmaz , null olmazsa çalışır.

print("Sonuç 1 : \${isim?.toUpperCase()}");

//Yöntem 2 : ! Uyarıdan kurtarır ama hata olma ihtimali vardır.

//Ayrıca eğer değişkenin içinde veri varsa ! gerekli değildir program uyarı verecektir.

print("Sonuç 2 : \${isim!.toUpperCase()}");

//Yöntem 3 : if ile null kontrolü yapmak

if(isim != null){

print("Sonuç 3 : \${isim.toUpperCase()}");

//if kontrolü sonrasında nullable özelliği direk kullanabiliriz.

}else{

print("isim null ve işlem yapılmadı");

}

late

- Hafıza yönetimini verimli hale getirmek için kullanılır.
- Null safety özelliği ile sınıf içinde değer atamadan değişken oluşturamayız.
- Bunu çözenin yolu late keyword'u kullanmaktır.
- Değişken tanımlandığı anda hafızada yer ayrılmasındansa, değişkeni ilk kullandığımız anda hafızada yer ayrılmasını sağlar.

```
class LateKullanimi {  
    //int x; //Bu şekilde içerisinde değer olmadan atama yapamayız.  
  
    late int y; //late ile değer atamadan tanımlama yapabiliriz.  
  
    int z = 10; //Değer atayarak her zaman çalışabiliriz.  
}
```

Sınıf İçinde Değişken Oluşturma

- Sınıfın constructorı varsa ve içinde değişken ataması yapılabilirse late kullanmaya gerek yoktur.

```
class Kisiler{  
    int kisi_no;  
    String kisi_ad;  
  
    Kisiler( this.kisi_no, this.kisi_ad);  
  
}
```

Constructor

Constructor

- Bir sınıftan (class) nesne oluşturmak için gerekli olan yapıdır.
 - Varsayılan olarak biz oluşturmasakta oluşturulur.
- Nesne oluşurken istenilen kodlamalar bu metod içinde yapılabilir.

```
class Kisiler{  
    late String ad;  
    late int yas;
```

```
    Kisiler(){
```

```
    }
```

```
}
```

```
var kisi = Kisiler();
```

Bu parantezler boş
constructor'ı temsil
etmektedirler.

```
kisi.ad = "Ahmet";
```

```
kisi.yas = 23;
```

```
print(kisi.ad);
```

```
print(kisi.yas);
```

Dolu Constructor

```
class Kisiler{  
    String ad;  
    int yas;  
  
    Kisiler(this.ad, this.yas);  
}  
  
var kisi = Kisiler("Ahmet", 23);  
print(kisi.ad);  
print(kisi.yas);
```

Bu parantezler dolu constructor'ı temsil etmektedirler.

Bir sınıfta hem boş hem dolu constructor olamaz.

Dolu constructor varsa late kullanılmasına gerek yoktur.

Dolu Constructor

Zorunlu Parametre Adı Yazdırma

```
class Kisiler{  
    String ad;  
    int yas;  
  
    Kisiler({required this.ad, required this.yas});  
}
```

parametre alanının başına ve sonuna
süslü parantez ve alanların başına
required eklenirse zorunlu olarak
parametre adı girilmesi beklenir.

```
var kisi = Kisiler(ad:"Ahmet",yas:23);
```

Nesne oluştururken
parametre adını girmek
zorundayız.

```
var kisi = Kisiler("Ahmet",23);
```

parametre adı
girilmezse hata alırsınız.

Paketler & Import

Paketler & Import

- **Paketler** birden fazla sınıfı kümelediğimiz yapılardır.
- **Paketler** ile daha düzenli projeler oluşturulabilir.
- **Import** bir sınıfı başka bir sınıf içinde kullanıyorsa o sınıfı **import** etmeliyiz.
- Bir sınıfın tüm uzantısı kullanıldığında **import'a** gerek yoktur.



```
import 'package:nesne_tabanli_programlama/paket1/A.dart';
```

```
class B{  
  
    var a = A();  
  
}
```

```
class A{  
  
}
```

Visibility Modifier

Visibility Modifier (Erişim)

<ul style="list-style-type: none">• Public<ul style="list-style-type: none">• Sınıf• Metod• Değişken• (Bütün sınıflardan ve paketlerden ulaşılabilir.)	<ul style="list-style-type: none">• Private<ul style="list-style-type: none">• Metod• Değişken• (Yalnızca tanımlandıkları sınıftan ulaşılabilir.)
--	--

Not : Hiçbir access modifier kullanılmıyorsa public anlamına gelir.

Visibility Modifier Örnek

```
class A{  
    late int publicDegisken;  
    late int _privateDegisken;  
}
```

Private ifadesi _ ile eklenir.

```
var a = A();
```

```
a.publicDegisken = 1 ;
```

```
a.privateDegisken = 2;
```

```
print(a.publicDegisken);
```

```
print(a.privateDegisken);
```

Private ifadeye dışardan erişilemez.

Static Değişkenler ve Metodlar

Static Değişkenler ve Metodlar

- Bir değişkenin veya metodun, bulunduğu sınıftan (class) nesne oluşturmaya gerek kalmadan erişilmek istenirse kullanılır.

```
class Asinifi{  
  
    static int degisken = 10;  
  
    static final double oran = 10.45;  
  
    static void metod() {  
        | print("Merhaba");  
    }  
  
}
```

```
print(Asinifi.degisken);
```

```
Asinifi.degisken = 100;  
print(Asinifi.degisken);
```

```
print(Asinifi.oran);
```

```
Asinifi.metod();
```

Enumeration

Enumeration

- **enum** ifadesi gösterilir.
- Parametrelerde kullanılır.
- Verilerin eşleşmesi sonucunda bir işlem yapılır.
- Kodlama yapan yazılımcıyı detaydan kurtarmaktadır.

```
enum Renkler{  
    Beyaz, Siyah  
}
```

```
var renk = Renkler.Beyaz;  
  
switch(renk){  
    case Renkler.Beyaz:{  
        print("#FFFFFF");  
    }  
    break;  
  
    case Renkler.Siyah:{  
        print("#000000");  
    }  
    break;  
}
```

Enumeration - Örnek

```
enum KonserveBoyut{  
    Kucuk, Orta, Buyuk  
}
```

```
void ucretAl(KonserveBoyut boyut){  
  
    switch(boyut){  
        case KonserveBoyut.Kucuk:{  
            print(20*30);  
        }  
        break;  
  
        case KonserveBoyut.Orta:{  
            print(30*30);  
        }  
        break;  
  
        case KonserveBoyut.Buyuk:{  
            print(40*30);  
        }  
        break;  
    }  
}  
  
ucretAl(KonserveBoyut.Orta);
```


Composition

Composition

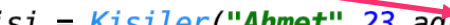
- Başka sınıflardan (class) oluşmuş nesneler bir sınıfın değişkeni olabilir.

```
class Adres{  
    String il;  
    String ilce;  
  
    Adres(this.il, this.ilce);  
}
```

```
class Kisiler{  
    String ad;  
    int yas;  
    Adres adres;  
  
    Kisiler(this.ad, this.yas, this.adres);  
}
```



```
var adres = Adres("Bursa", "Osmangazi");  
  
var kisi = Kisiler("Ahmet", 23, adres);  
  
print("Kişi ad : ${kisi.ad}");  
print("Kişi yaş : ${kisi.yas}");  
print("Kişi il : ${kisi.adres.il}");  
print("Kişi ilçe : ${kisi.adres.ilce}");
```



Composition

Kategoriler Tablosu

kategori_id	kategori_ad
1	Dram
2	Komedi
3	Bilim Kurgu

Yonetmenler Tablosu

yonetmen_id	yonetmen_ad
1	Nuri Bilge Ceylan
2	Quetin Tarantino
3	2013

Filmler Tablosu

film_id	film_ad	film_yil	kategori_id	yonetmen_id
1	Django	2013	1	2
2	Inception	2006	3	3

```
class Kategoriler {  
    int kategori_id;  
    String kategori_ad;
```

```
Kategoriler(this.kategori_id, this.kategori_ad);  
}
```

```
class Yonetmenler {  
    int yonetmen_id;  
    String yonetmen_ad;
```

```
Yonetmenler(this.yonetmen_id, this.yonetmen_ad);  
}
```

```
class Filmler {  
    int film_id;  
    String film_ad;  
    int film_yil;  
    Kategoriler kategori;  
    Yonetmenler yonetmen;
```

```
Filmler(this.film_id, this.film_ad, this.film_yil, this.kategori, this.yonetmen);  
}
```

```
var k1 = Kategoriler(1,"Dram");  
var k2 = Kategoriler(2,"Komedi");
```

```
var y1 = Yonetmenler(1,"Nuri Bilge Ceylan");  
var y2 = Yonetmenler(2,"Quentin Tarantino");
```

```
var f1 = Filmler(1,"Django",2013,k1,y2);
```

```
print("Film id : ${f1.film_id}");  
print("Film ad : ${f1.film_ad}");  
print("Film yıl : ${f1.film_yil}");  
print("Film kategori : ${f1.kategori.kategori_ad}");  
print("Film yönetmen : ${f1.yonetmen.yonetmen_ad}");
```


Kalıtım (Inheritance)

OOP Kuralı – Kalıtım (Inheritance)

- Mevcut bir sınıftan başka bir sınıf türetmek için kullanılır.
- Kodun tekrar kullanılabilirliğini artırır.
- Sadece **class** için geçerlidir.
- Super class **extends** kelimesi ile subclass'a eklenir.
- Bir sınıfın tek kalıtımı olabilir.
- Bir sınıfa birden fazla sınıf kalıtım yolu ile bağlanamaz.
- Üst sınıfa **superclass** denir.
- Alt sınıfa **subclass** denir.

```
class Arac {  
  
}
```

```
class Araba extends Arac {  
  
}
```

```
class Nissan extends Araba {  
  
}
```



Örnek :

```
class Arac {  
    String renk;  
    String vites;  
  
    Arac(this.renk, this.vites);  
}
```

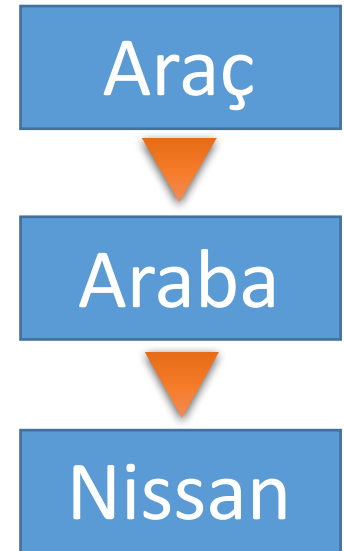
```
class Araba extends Arac {  
    String kasaTipi;  
  
    Araba(this.kasaTipi, String renk, String vites) : super(renk, vites);  
}
```

```
class Nissan extends Araba {  
    String model;  
  
    Nissan(this.model, String kasaTipi, String renk, String vites) : super(kasaTipi, renk, vites);  
}
```

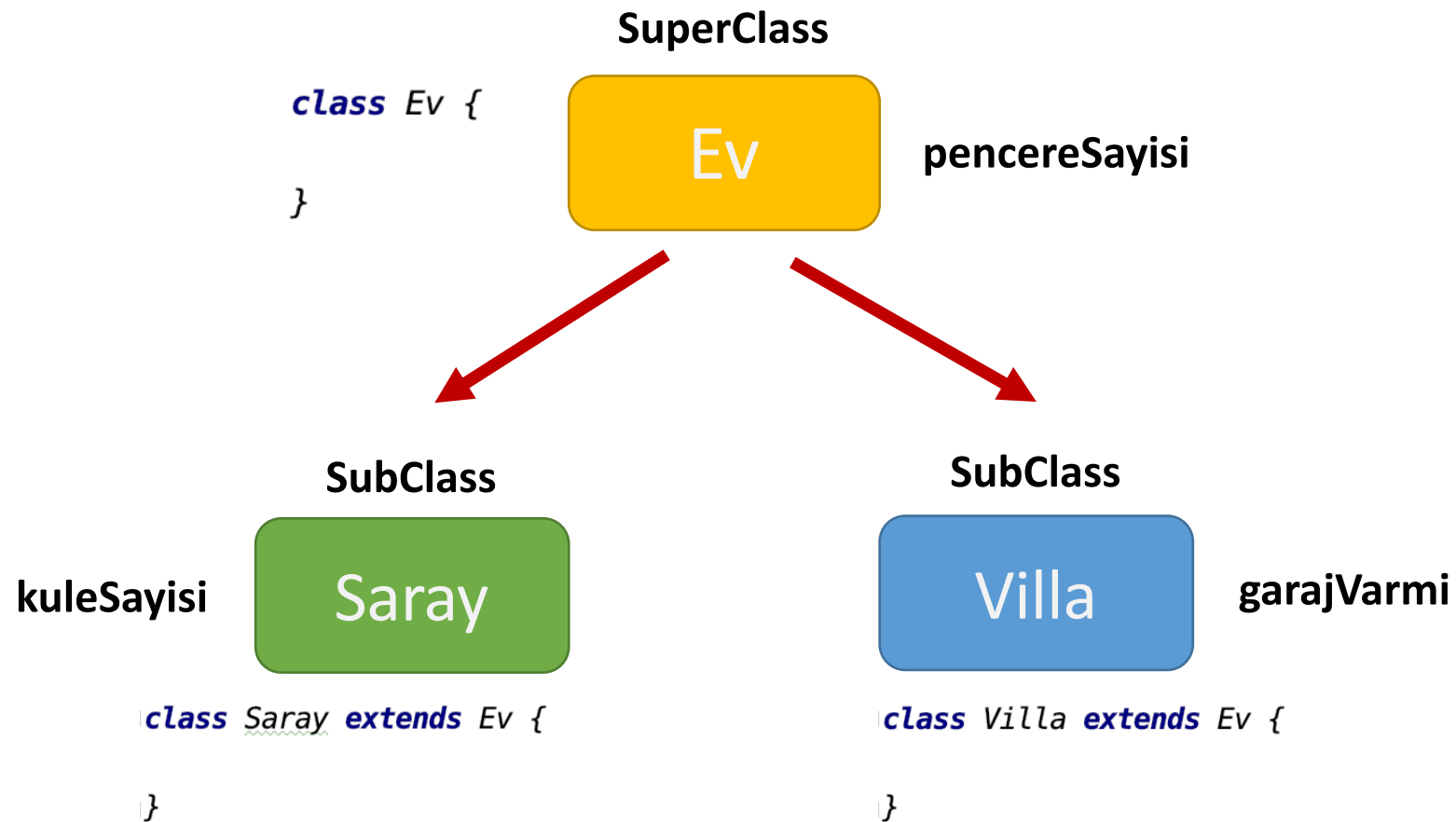
```
var araba = Araba("Sedan", "Kırmızı", "Otomatik");
```

```
print(araba.kasaTipi);  
print(araba.renk);  
print(araba.vites);
```

Kalıtım yoluyla oluşturulan sınıfın constructor'ı üst sınıfın özelliklerini almalıdır.



Kalıtım Hiyerarşisi Örnek



Kalıtım Hiyerarşisi Örnek

```
class Ev {  
    int pencereSayisi;  
  
    Ev(this.pencereSayisi);  
}
```

```
class Saray extends Ev {  
    int kuleSayisi;  
  
    Saray(this.kuleSayisi,int pencereSayisi) : super(pencereSayisi);  
}
```

```
class Villa extends Ev {  
    bool garajVarmi;  
  
    Villa(this.garajVarmi,int pencereSayisi) : super(pencereSayisi);  
}
```

```
var topkapiSarayi = Saray(10,100);  
var bogazVilla = Villa(true,20);
```

```
print(topkapiSarayi.kuleSayisi); //Kendi değişkeni  
print(topkapiSarayi.pencereSayisi); //Kalıtım ile gelen değişken
```

```
print(topkapiSarayi.garajVarmi); //Saray ile Villa arasında bir kalıtım ilişkisi yok.
```

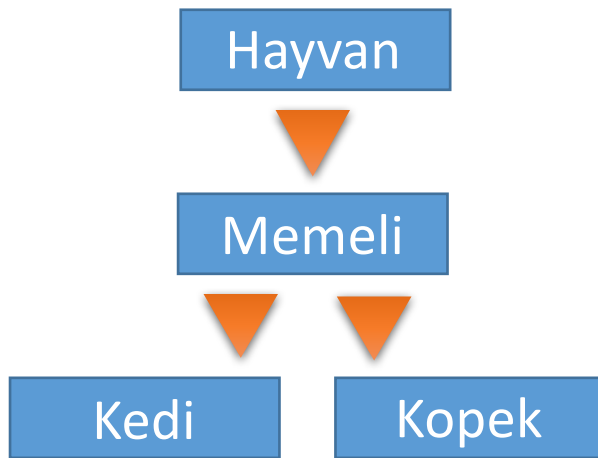
```
print(bogazVilla.garajVarmi); //Kendi değişkeni  
print(bogazVilla.pencereSayisi); //Kalıtım ile gelen değişken
```

```
print(bogazVilla.kuleSayisi); //Villa ile Saray arasında bir kalıtım ilişkisi yok.
```

Override

Metodları Ezme : Overriding

- Kalıtım ilişkisinde üst sınıfın metodlarının alt sınıf tarafından tekrar kullanılmasıdır.



```
class Hayvan {  
    void sesCikar(){  
        print("Ses yok");  
    }  
}
```

```
class Memeli extends Hayvan {  
}
```

```
class Kedi extends Memeli {  
    @override  
    void sesCikar() {  
        print("Miyav Miyav");  
    }  
}
```

```
class Kopek extends Memeli {  
    @override  
    void sesCikar() {  
        print("Hav Hav");  
    }  
}
```

```
var hayvan = Hayvan();  
hayvan.sesCikar();  
//Üst sınıf kendi metodunu çalıştırır.
```

Sesim Yok

```
var memeli = Memeli();  
memeli.sesCikar();  
//Alt sınıfta bu metod yoksa üst sınıfı çalıştırır.
```

Sesim Yok

```
var kedi = Kedi();  
kedi.sesCikar();  
//Kedi sınıfı üst sınıfın metodu override  
//ettiği için kendi metodunu çalıştırır.
```

Miyav Miyav

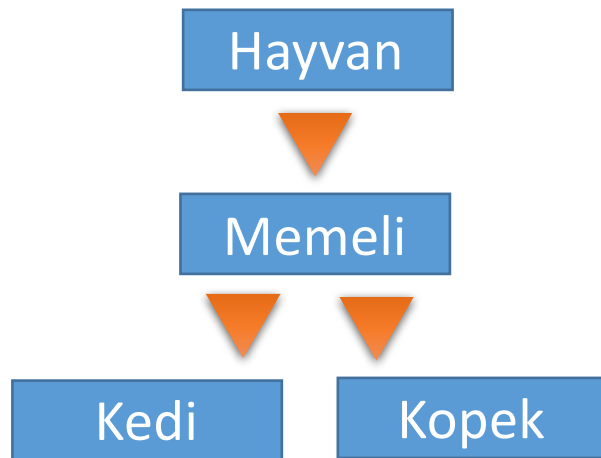
```
var kopek = Kopek();  
kopek.sesCikar();  
//Kopek sınıfı üst sınıfın metodu override  
//ettiği için kendi metodunu çalıştırır.
```

Hav Hav

PolyMorphism

PolyMorphism

- PolyMorphism olması için iki sınıf arasında kalıtım ilişkisi olmalıdır.
- Daha kapsayıcı bir kullanım sağlamak için kullanılır.
- Özellikle metodların parametrelerinde PolyMorphism kullanılarak daha kapsayıcı veriler alınabilir.
- Superclass gibi görünüp subclass gibi davranır.

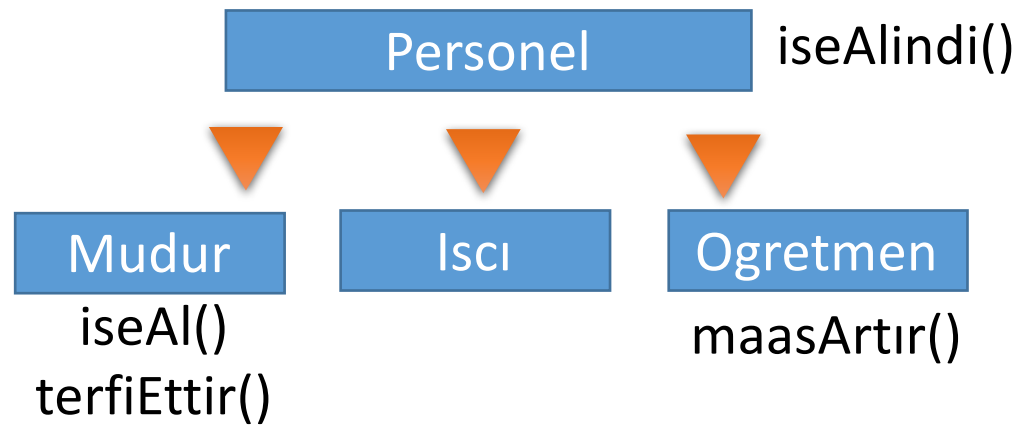


superclass *subclass*
Hayvan h = Kopek();

h.sesCikar();

Hav Hav

PolyMorphism Örnek



PolyMorphism Örnek

```
class Personel {  
    void iseAlindi(){  
        print("Personel mutlu");  
    }  
}  
  
class Mudur extends Personel {  
    void iseAl(Personel p){  
        p.iseAlindi();  
    }  
}  
  
class Isci extends Personel{  
}  
  
class Ogretmen extends Personel{  
}
```

KULLANIM

```
Personel ogretmen = Ogretmen();
```

```
Personel isci = Isci();
```

```
var mudur = Mudur();
```

```
mudur.iseAl(isci);      Personel Mutlu  
mudur.iseAl(ogretmen); Personel Mutlu
```

Özellikle metodların parametrelerinde
PolyMorphism kullanılarak daha
kapsayıcı veriler alınabilir.

PolyMorphism Örnek

```
class Personel {  
    void iseAlindi(){  
        print("Personel mutlu");  
    }  
}  
  
class Mudur extends Personel {  
    void iseAl(Personel p){  
        p.iseAlindi();  
    }  
}  
  
void terfiEttir(Personel p){  
    (p as Ogretmen).maasArttir();  
}  
  
class Isci extends Personel{  
}  
  
class Ogretmen extends Personel{  
    void maasArttir(){  
        print("Maaş arttı.Öğretmen Mutlu");  
    }  
}  
  
KULLANIM  
  
Personel ogretmen = Ogretmen();  
  
Personel isci = Isci();  
  
var mudur = Mudur();  
  
mudur.terfiEttir(ogretmen);  
mudur.terfiEttir(isci);
```

Casting hatası oluşacaktır çünkü isci Ogretmen sınıfına dönüşemez.

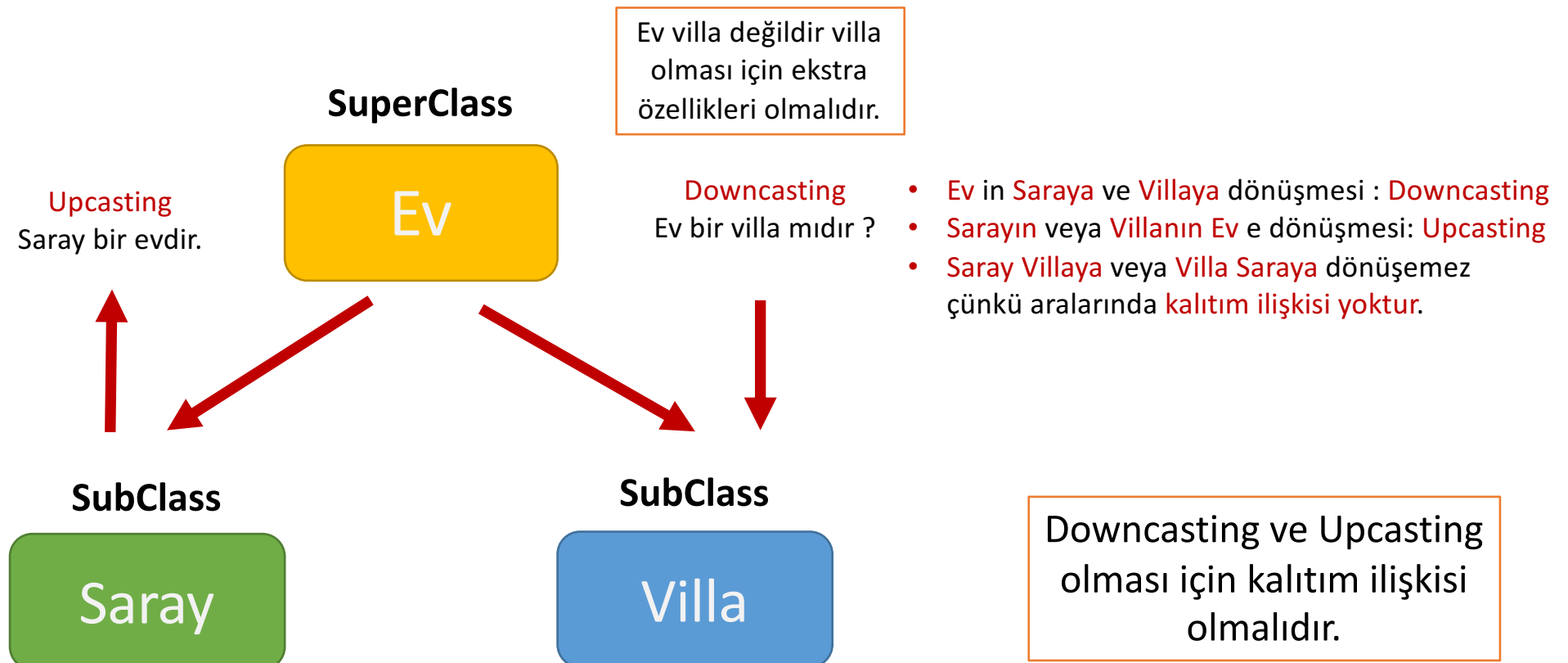
Nesnelerin Tip Dönüşümü

Tip Kontrolü - **is**

- Tip kontrolü **is** ile yapılabilir.is true false şeklinde bilgi verir.

```
var saray = Saray(3,30);  
  
if(saray is Saray){  
    print("Saraydır");  
}else{  
    print("Saray Değildir");  
}
```

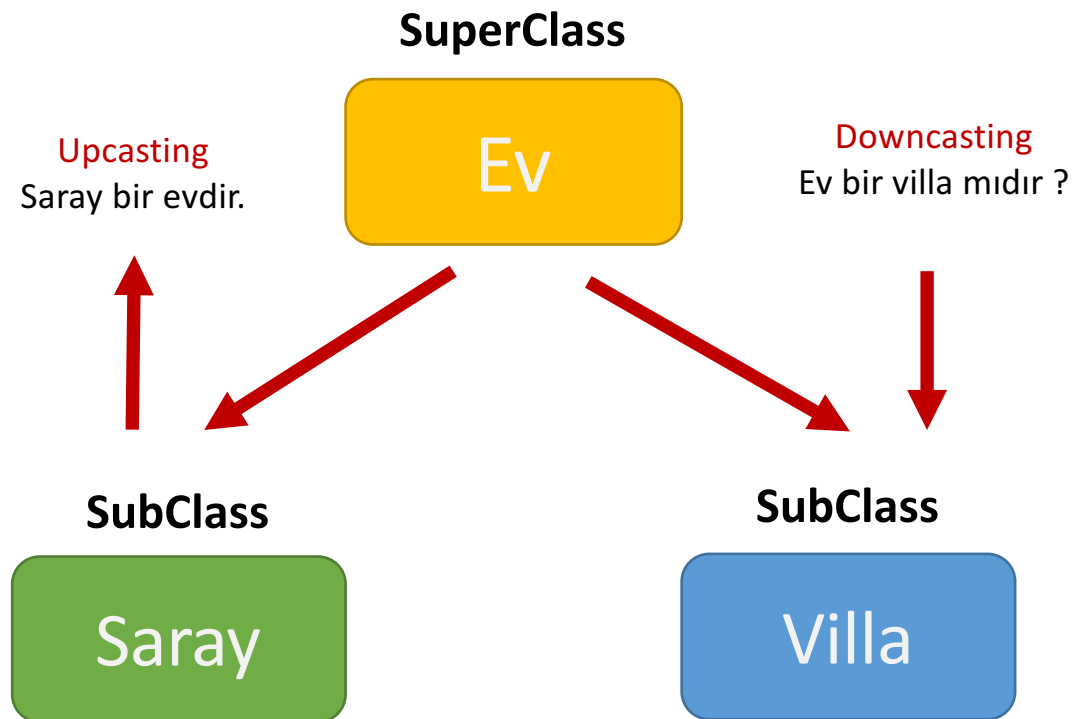
Downcasting – UpCasting



Upcasting

```
var saray = Saray(10,200);
```

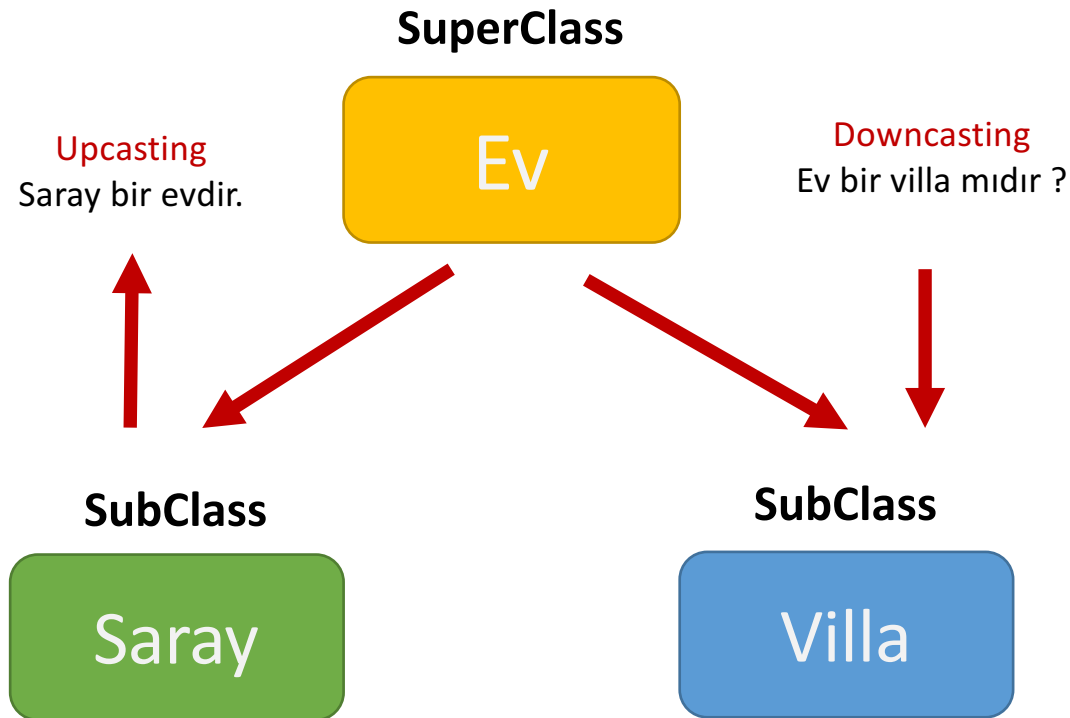
```
Ev ev = saray ;  
//Upcasting
```



Downcasting – **as**

```
var ev = Ev(5);
```

```
Saray saray = ev as Saray;  
//Downcasting
```



PolyMorphism Casting Hatası Önleme

Tip Kontrolü **is**

```
class Mudur extends Personel {  
    void iseAl(Personel p){  
        p.iseAlindi();  
    }  
  
    void terfiEttir(Personel p){  
        if( p is Ogretmen){  
            p.maasArttir();  
        }  
  
        if( p is Isci){  
            print("İşçiler terfi alamaz");  
        }  
    }  
}
```

KULLANIM

```
Personel ogretmen = Ogretmen();
```

```
Personel isci = Isci();
```

```
var mudur = Mudur();
```

```
mudur.terfiEttir(ogretmen);
```

```
mudur.terfiEttir(isci);
```

```
Maaş arttı.Öğretmen Mutlu :)  
İşçiler terfi alamaz
```

Interface

Interface

- Class yapısında kullanılabilir.
- Bir sınıf birden fazla interface alabilir.
- **implements** ile eklenirler.
- Hazır taslaklar gibi düşünebilirsiniz.
- Interface'ler sınıflara özellik katar.

```
abstract class Interface1{  
    late int degisken;  
  
    void metod1();  
  
    String metod2();  
}
```

```
class ClassA implements Interface1 {  
    @override  
    int degisken = 10 ;  
  
    @override  
    void metod1() {  
        print("Interface Merhaba");  
    }  
  
    @override  
    String metod2() {  
        return "Interface Çalışması";  
    }  
}
```

Interface Örnek

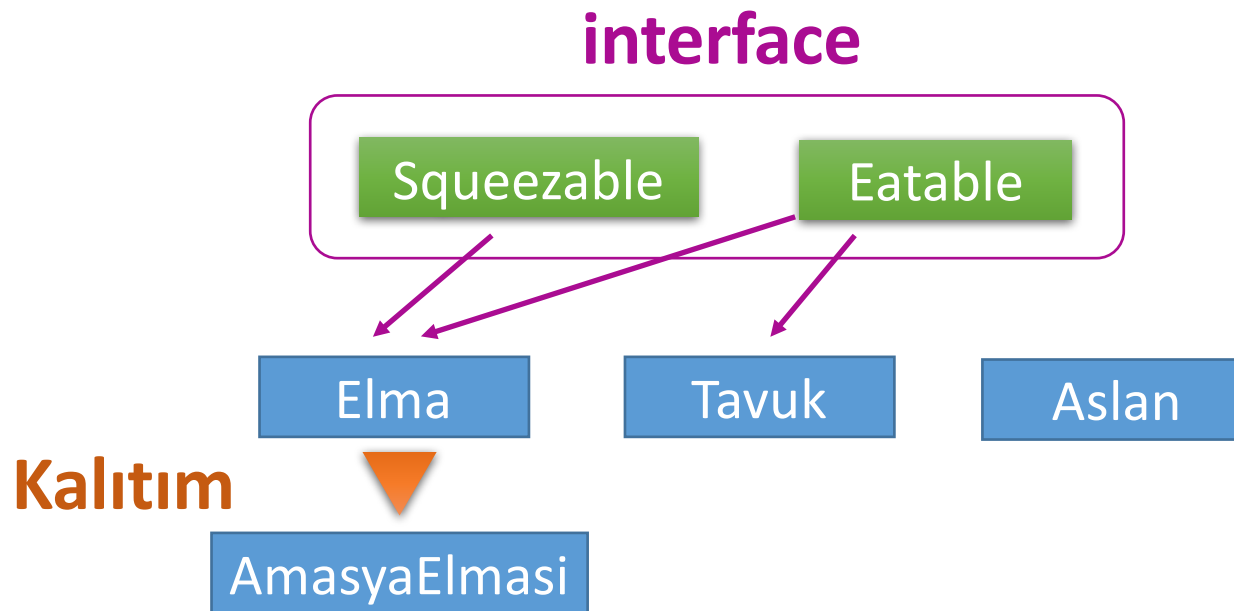
```
abstract class Interface1{  
    late int degisken;  
  
    void metod1();  
  
    String metod2();  
}
```

```
void main(){  
  
    var a = ClassA();  
  
    print(a.degisken);  
    a.metod1();  
    print(a.metod2());  
}
```

```
class ClassA implements Interface1 {  
    @override  
    int degisken = 10 ;  
  
    @override  
    void metod1() {  
        print("Interface Merhaba");  
    }  
  
    @override  
    String metod2() {  
        return "Interface Çalışması";  
    }  
}
```

```
10  
Interface Merhaba  
Interface çalışması
```

Interface Örnek



Interface Örnek

```
abstract class Squeezable {  
    void howToSqueeze();  
}
```

```
abstract class Eatable {  
    void howToEat();  
}
```

```
class Elma implements Squeezable, Eatable {  
    @override  
    void howToEat() {  
        print("Dilimle ve ye");  
    }  
  
    @override  
    void howToSqueeze() {  
        print("Blendır ile sık");  
    }  
}
```

```
class Tavuk implements Eatable {  
    @override  
    void howToEat() {  
        print("Fırında kızart");  
    }  
}
```

```
class Aslan {  
}
```

```
var aslan = Aslan();  
Elma amasyaElmasi = AmasyaElmasi();  
var elma = Elma();  
Eatable tavuk = Tavuk();
```

```
class AmasyaElmasi extends Elma {  
    @override  
    void howToEat() {  
        print("Yıka ve ye");  
    }  
}
```

Teşekkürler...



kasım-adalan



kasimadalan@gmail.com



kasimadalan