

# Swift OOP Eğitimi

## Nesne Tabanlı Programlama

Kasım ADALAN

Elektronik ve Haberleşme Mühendisi  
Freelance Software Developer

# Eğitim içeriği

1. Sınıf ( Class ve Structure ) Nedir ?
2. Nesne ( Object ) Nedir ?
3. Özellikler
4. Fonksiyonlar - Metodlar
5. Initialization – Constructor
  - Shadowing ( Gölgeleme )
6. Class vs Structure Farkı
7. Import
8. Static değişken & metodlar
9. Enumeration
10. Composition
11. Kalıtım
12. Kalıtım İlişkisinde Constructor
13. Metodları Ezme ( Override)
14. PolyMorphism
15. Nesnelerin Tip Dönüşümü
16. Extension
17. Protocol
18. Closure
19. Nesne Tabanlı Listeme
  - Array
  - Set
  - Dictionary

# 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 dönüştürülür
- 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 } Durum (state)  
-Hızlanmak ve  
yavaşlamak için pedallar } Davranış (behaviour)

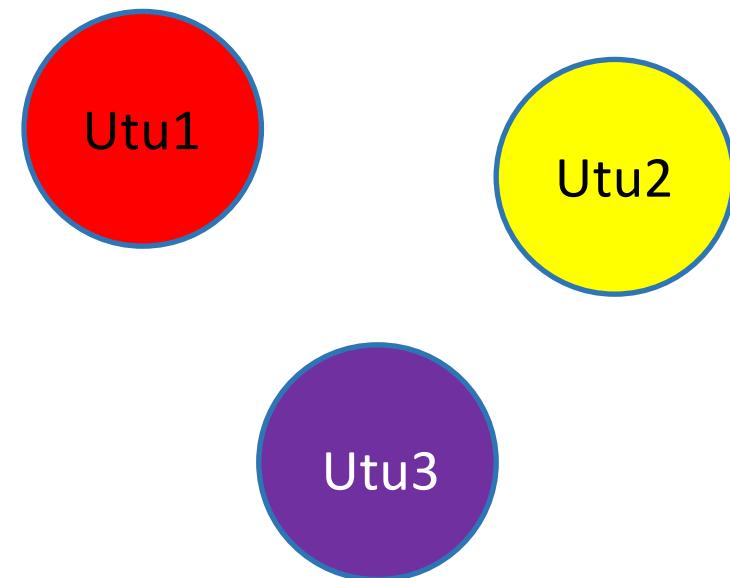
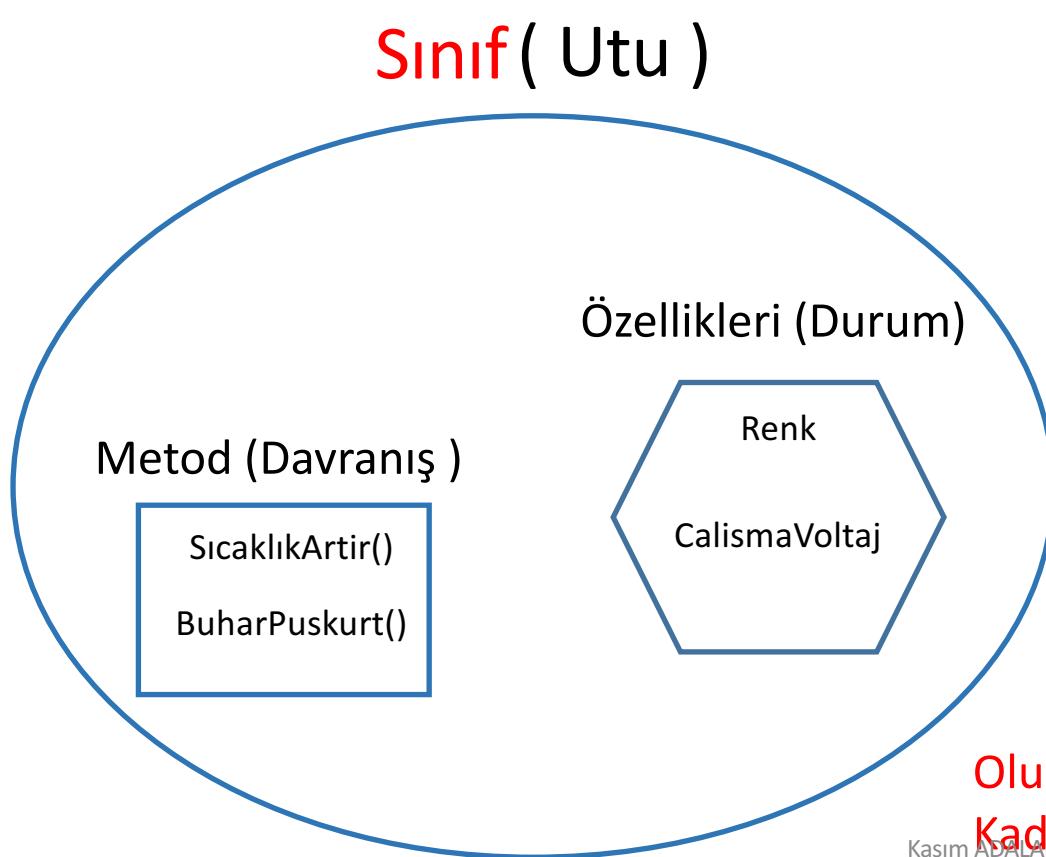
- Sınıflar ise nesnelerin özellikleri ve davranışları ile ilgili ayrıntıları içerir Araba sınıfı.  
-Frene basıldığında ne olur?



# Nesnenin Durumu ve Davranışı

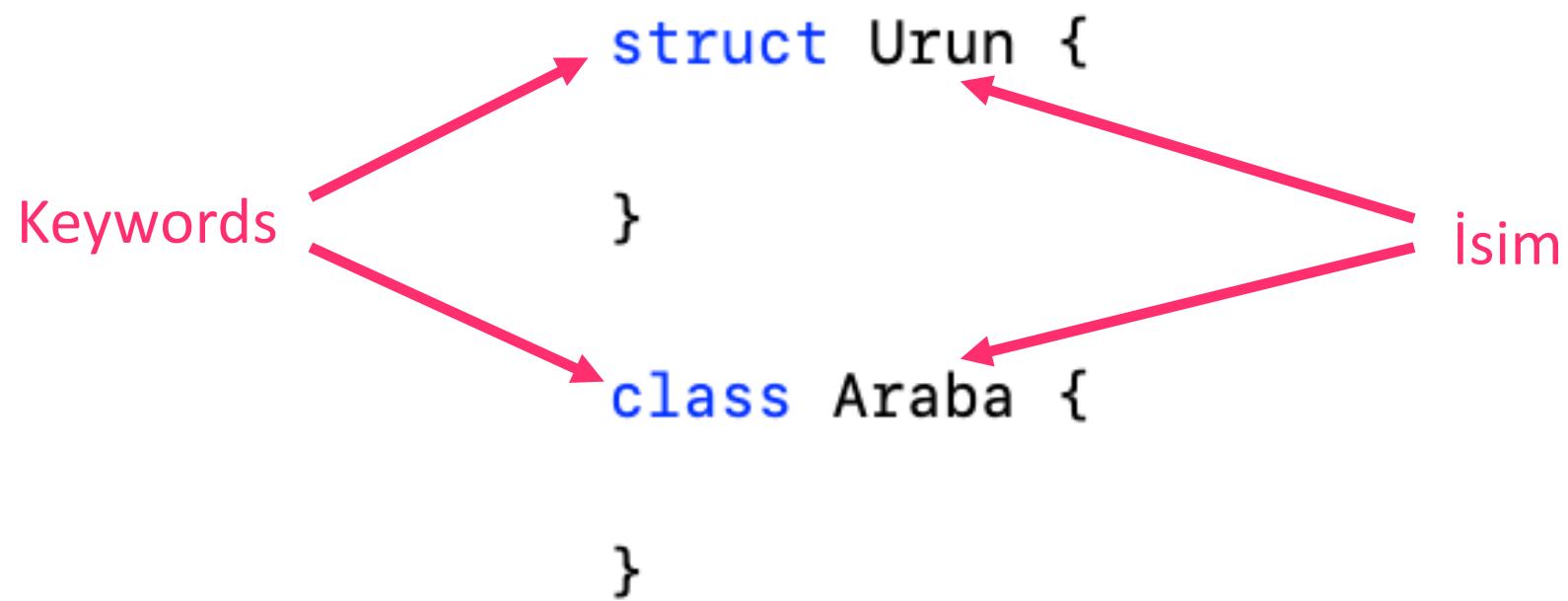
- **Durum (State):** Sınıfların bir – birçok özelliği olabilir
  - Somut değişkenler (instance variables) belirler
  - Nesneyle birlikte taşınır
- **Davranış (Behavior):** Sınıflar bir ya da birden çok metoda sahip olabilir
  - Metod program içindeki bir işi temsil eder
    - Görevlerin gerçekleştirileceği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

# Nesne ( Object )



Oluşturduğumuz Ütü sınıfı taslağında istediğimiz  
Kadar Nesne üretebiliriz.

# Class ve Structure Tanımlama



# Class ve Structure Tanımlama

## Dolu Özellikler

```
struct Urun {  
    var ad = "ütü"  
    var fiyat = 49.99  
}  
  
class Araba {  
    var renk = "kırmızı"  
    var kapasite = 4  
}
```

## Boş Özellikler

```
struct Urun {  
    var ad:String?  
    var fiyat:Double?  
}  
  
class Araba {  
    var renk:String?  
    var kapasite:Int?  
}
```

Boş özellik oluştururken  
? işaretini optional  
olduğunu bu değerin  
boş bırakılabilceğini  
belirtir.

# Nesne Tanımlama

```
struct Urun {  
    var ad:String?  
    var fiyat:Double?  
}
```

```
class Araba {  
    var renk:String?  
    var kapasite:Int?  
}
```

Nesnenin  
Adı

Nesnenin  
Oluşturulduğu  
Sınıf

var laptop = Urun()

var bmw = Araba()

# Class ve Structure Yapısına Erişim.

- Class veya Structure yapısı içindeki metod ve özelliklerine erişmek için ilk şart bulunduğu Class veya Structure ' dan nesne oluşturmalı.

```
struct Urun {  
    var ad:String?  
    var fiyat:Double?  
}
```

```
class Araba {  
    var renk:String?  
    var kapasite:Int?  
}
```

```
var laptop = Urun()  
  
//veri atama  
laptop.ad = "macbook"  
laptop.fiyat = 11987.67  
//atanan değerleri aynı nesne ile alma  
print(laptop.ad!)  
print(laptop.fiyat!)
```

```
var bmw = Araba()  
  
//veri atama  
bmw.renk = "kırmızı"  
bmw.kapasite = 4  
//atanan değerleri aynı nesne ile alma  
print(bmw.renk!)  
print(bmw.kapasite!)
```

# Class ve Structure Yapısına Erişim.

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

```
var laptop = Urun()

//veri atama
laptop.ad = "macbook"
laptop.fiyat = 11987.67
//atanan değerleri aynı nesne ile alma
print(laptop.ad!)
print(laptop.fiyat!)
```

```
var bmw = Araba()

//veri atama
bmw.renk = "kırmızı"
bmw.kapasite = 4
//atanan değerleri aynı nesne ile alma
print(bmw.renk!)
print(bmw.kapasite!)
```

```
var tv = Urun()

//veri atama
tv.ad = "Samsung"
tv.fiyat = 8987.67
//atanan değerleri aynı nesne ile alma
print(tv.ad!)
print(tv.fiyat!)
```

```
var limuzin = Araba()

//veri atama
limuzin.renk = "beyaz"
limuzin.kapasite = 8
//atanan değerleri aynı nesne ile alma
print(limuzin.renk!)
print(limuzin.kapasite!)
```

# Özellikler

# Lazy Özelliği

- Depolama önceliği olarak ikinci plana atılır.
- `var` yani variable olan değişkenlerden önce kullanılır.
- lazy olacak değişkenin başlangıç değeri olmalıdır. Boş bir değişken olamaz.

```
class Ornek {  
    lazy var no = 39 // `var` gereklidir.  
}  
  
var ilkornek = Ornek()  
print(ilkornek.no)
```

# Hesaplama Özelliği

- Depolama amaçlı değil hesaplama amaçlı kullanılan değişkenlerdir.
- Sınıfın içindeki değişkenlere bağlılığı ile işlemler yapar.
- Closure yapısına kullanılmaktadır. { }

```
class Matematik {  
    var x = 10  
    var y = 20  
  
    var topla:Int {  
        get { //İşlemin sonucunu almak için kullanılır.  
            return x+y  
        }  
    }  
}  
  
var nesne = Matematik()  
  
print(nesne.topla)
```

# Hesaplama Özelliği – ( Get ve Set )

- Get , özelliğin okunabilmesini sağlıyor.
- Set , özelliğin yeni veri aktarımı

```
class Maas {  
    var maas:Double = 10000  
    var bonus = 1.10  
  
    var haftalıkMaasHesaplama:Double {  
        get { //İşlemin sonucunu almak için kullanılır.  
            return (maas * bonus)/52  
        }  
  
        set(yeniHaftalıkMaas) { //İşlemin sonucunu almak için kullanılır.  
            self.maas = yeniHaftalıkMaas * 52  
        }  
    }  
  
    var maas1 = Maas()  
  
    print(maas1.haftalıkMaasHesaplama) //211.53846153846155  
  
    maas1.haftalıkMaasHesaplama = 600 //haftalık maas set edilir.|  
  
    print(maas1.maas) // Veri set edildikten sonra maas : 31200.0
```

# Fonksiyonlar - Metodlar

# Fonksiyonlar

- Belirli işlemleri temsil eden yapılardır.
- Kullanma amacımız tekrarlanan kod yapılarının önüne geçmektir.
- **func** kelimesi tanımlanırlar.
- Programlamayı daha pratik bir hale getirir.
- Kodun okunmasına faydası vardır.
- Class veya Structure içinde yer alan fonksiyonlara denir.
- Özellikler gibi bulunduğu Class veya Structure'dan nesne oluşturulursa erişilebilir.

```
func fonksiyon adı ( Parametre ) -> dönüş türü {
```

```
//Kodlama buraya yazılır
```

```
return dönüş değeri
```

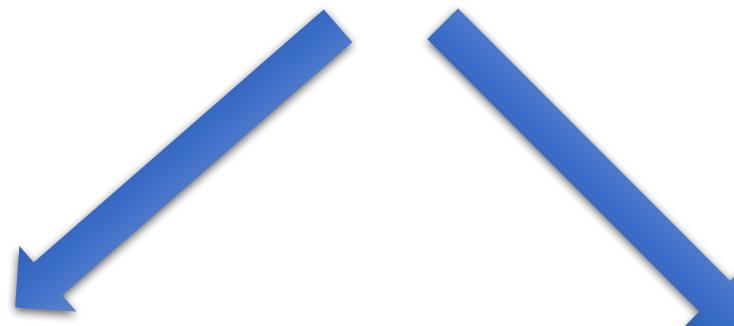
```
}
```

# Class ve Structure Metodlarına Erişim.

```
class Araba {  
    var renk:String?  
    var kapasite:Int?  
  
    func calistir(){  
        print("Araba çalıştı")  
    }  
    func durdur(){  
        print("Araba durdu")  
    }  
}
```

```
var bmw = Araba()  
  
//veri atama  
bmw.renk = "kırmızı"  
bmw.kapasite = 4  
//atanan değerleri aynı nesne ile alma  
print(bmw.renk!)  
print(bmw.kapasite!)  
  
bmw.calistir()  
bmw.durdur()
```

## Foksiyonlar



Geri dönüş değeri olan

Geri dönüş değeri olmayan

# Geri Dönüş değeri olmayan fonksiyonlar

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

```
func selamla() {  
    let sonuc = "Merhaba Ahmet"  
    print(sonuc)  
}  
  
selamla()
```

# Geri Dönüş değeri olan fonksiyonlar

- Yapılan işlem sonucunda metodun kullanana veri dönüşü yapan fonksiyonlardır.

```
func selamla() -> String {  
    let sonuc = "Merhaba Ahmet"  
    return sonuc  
}
```

```
let gelenSonuc = selamla()  
  
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.
- **: işaretinin** ile ismi ve türü belirlenir.

**mesaj : String**

parametre adı      parametre türü

```
func selamla(isim: String) {  
    let sonuc = "Merhaba \(isim)"  
    print(sonuc)  
}  
  
selamla(isim : "Ahmet")
```

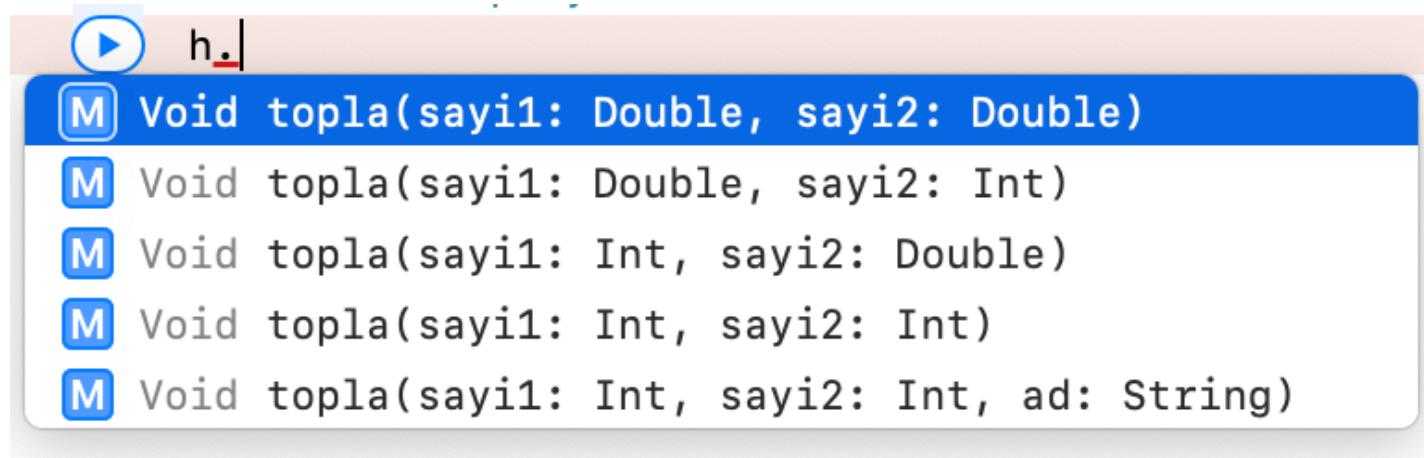
# Fonksiyonların Parametre Alması

- Birden fazla parametre kullanılabilir.

```
func topla(sayi1: Int, sayi2: Int) -> Int {  
    return sayi1+sayi2  
}  
  
let gelenSonuc = topla(sayi1 : 20,sayi2 : 30)  
  
print(gelenSonuc)
```

# Metodların Aşırı Yüklenmesi : Overloading

- Metodların aynı isimde tekrar kullanılması.
- Tek şart metodun parametre dizilimindeki türlerin farklı olmasıdır.
- Metodların kullanımında parametre çeşitliliği sağlar.



# Metodların Aşırı Yüklenmesi : Overloading

```
class Hesaplayici {  
    func topla(sayi1:Int,sayi2:Int){  
        print("Toplam : \(sayi1 + sayi2)")  
    }  
    func topla(sayi1:Double,sayi2:Int){  
        print("Toplam : \(sayi1 + Double(sayi2))")  
    }  
    func topla(sayi1:Int,sayi2:Double){  
        print("Toplam : \(Double(sayi1) + sayi2)")  
    }  
    func topla(sayi1:Double,sayi2:Double){  
        print("Toplam : \(sayi1 + sayi2)")  
    }  
    func topla(sayi1:Int,sayi2:Int,ad:String){  
        print("Toplama yapan \(ad) , sonuç : \(sayi1 + sayi2)")  
    }  
}  
  
var h = Hesaplayici()  
h.topla(sayi1: 4, sayi2: 5, ad: "ahmet")
```

# Variadic Parametreler

- İstenildiği kadar veri girilebilen parametre

```
func toplamVariadiac(sayilar:Int...) -> Int {  
    var toplam = 0  
    for s in sayilar {  
        toplam = toplam + s  
    }  
    return toplam  
}  
  
var v1 = toplamVariadiac(sayilar: 1,2,3,4,5)//15  
print(v1)  
  
var v2 = toplamVariadiac(sayilar: 10,28,32,122,43)//303  
print(v2)
```

# Birden Fazla Dönüş Değeri

```
func islem (sayilar:[Int]) -> (toplam:Int,faktoriyel:Int){  
    var toplam = 0  
    var faktoriyel = 1  
    for s in sayilar {  
        toplam = toplam + s  
        faktoriyel = faktoriyel * s  
    }  
    return (toplam,faktoriyel)  
}  
  
var dizi = [1,2,3,4,5]  
  
let b1 = islem(sayilar:dizi)  
  
print("Toplam : \(b1.toplam)")  
print("Toplam : \(b1.faktoriyel)")
```

# Geri dönüş değerinin Optional yapılması

- Fonksiyonu kullanacak kişiye geri dönen değerin nil veya dolu olabileceğini bu gelen değeri kontrol etmesini söyler.
- ? işaretini kullanılır.

```
func hesapla(sayi1:Int,sayi2:Int) -> Int? {  
    let islem = sayi1*2 + sayi2*4  
    return islem  
}
```

```
var a = hesapla(sayi1: 10, sayi2: 20)  
print("İşlem Sonucu : \(a!)")
```

## Foksiyonlarda Global ve Local parametreler

```
func hesapla1(sayi1 s1:Int,sayi2 s2:Int) -> Int {  
    let islem = s1*2 + s2*4  
    return islem  
}
```

```
var a1 = hesapla1(sayi1: 10, sayi2: 20)  
print("İşlem Sonucu : \(a1)")
```

# Local parametre Global parametreyi etkilemesi

Geri dönüşü parametre üzerinden yapıyorlar

```
func degistir(a: inout Int, b: inout Int) {  
    let t = a  
    a = b  
    b = t  
}  
  
var x = 2  
var y = 10  
degistir(a: &x, b: &y)  
print("x : \(x), y : \(y)") // x : 10 , y: 2
```

a ve b içine x ve y değişkenlerinin değerleri aktarılır.

a ve b üzerinde olan değişiklikler işlem bittikten sonra x ve y geri döner.  
Bu şekilde local globali etkilemiş olur.

# 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  $(n-2) \cdot 180$

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

# Initialization - Constructor

# Initialization - Constructor

- Bir sınıfından ( class veya structure ) nesne oluşturmak için gereklili olan yapıdır.
- **init** kelimesi ile tanımlanır.
- Bir sınıfından ( class veya structure ) nesne oluşturma işleminde parametre alabilir.

```
class Kisiler {  
    var ad:String?  
    var yas:Int?  
  
    init() {  
        var kisi1 = Kisiler()  
        kisi1.ad = "Ahmet"  
        kisi1.yas = 18  
  
    }  
    init(ad:String,yas:Int) {  
        self.ad = ad  
        self.yas = yas  
    }  
}
```

```
var kisi1 = Kisiler()  
kisi1.ad = "Ahmet"  
kisi1.yas = 18
```

```
var kisi2 = Kisiler(ad:"Mehmet",yas:20)
```

# Shadowing - Gölgeleme

- Bir metod içinde aynı isimde oluşturulmuş global değişkeni lokal değişken gölgelemektedir.
- Bu gölgelemeyi engellemek için metod içinde global değişken self kelimesi tanımlanır.

```
class Kisiler {  
    var ad:String?  
    var yas:Int?  
  
    init() {  
  
    }  
    init(ad:String,yas:Int) {  
        self.ad = ad  
        self.yas = yas  
    }  
}
```

# Class vs Structure Farkı

# Class vs Structure

- Her iki yapıda benzer özelliklere sahiptir.
  - Class , referans tipidir.
  - Structure , değer tipidir.
  - Structure için miras özelliği yoktur.
- 
- **Referans Tipi** : Ali ile Mehmet ortak bir dosya kullanıyorlar. Ali bu dosya içindeki veriyi değiştirirse Mehmet'e bu değişiklikten etkilenir.
  - **Değer Tipi** : Ali veya Mehmet dosyadaki bilgiyi değiştirirlerse ikisi de bu değişiklikten etkilenmez ve herkesin dosyası kendisinde olur.

# Referans Tipi Örnek

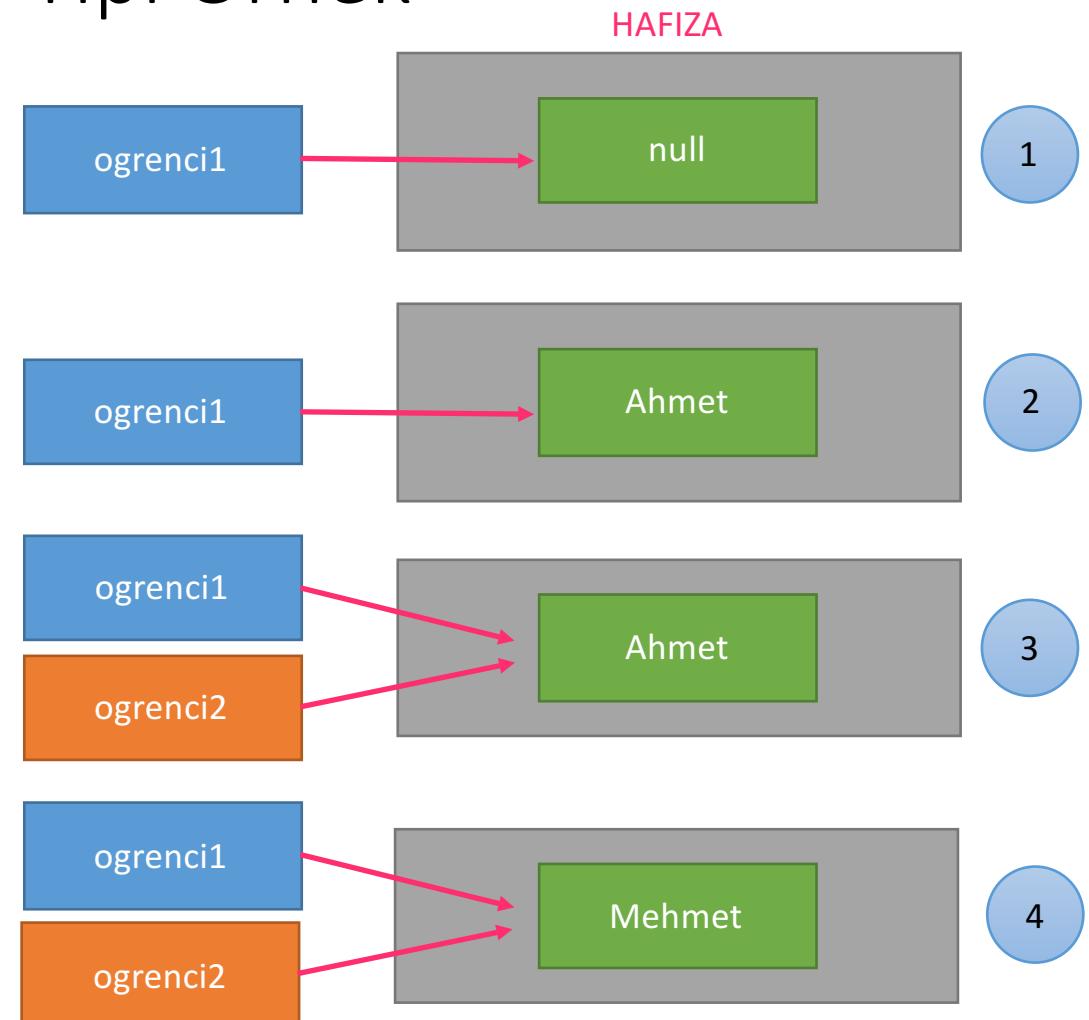
- Hafıza yönetiminde iki nesnenin aynı yeri işaret etmesidir.

```
class Ogrenci {  
    var ad:String?  
}
```

1 `var ogrenci1 = Ogrenci()`  
2 `ogrenci1.ad = "Ahmet"`

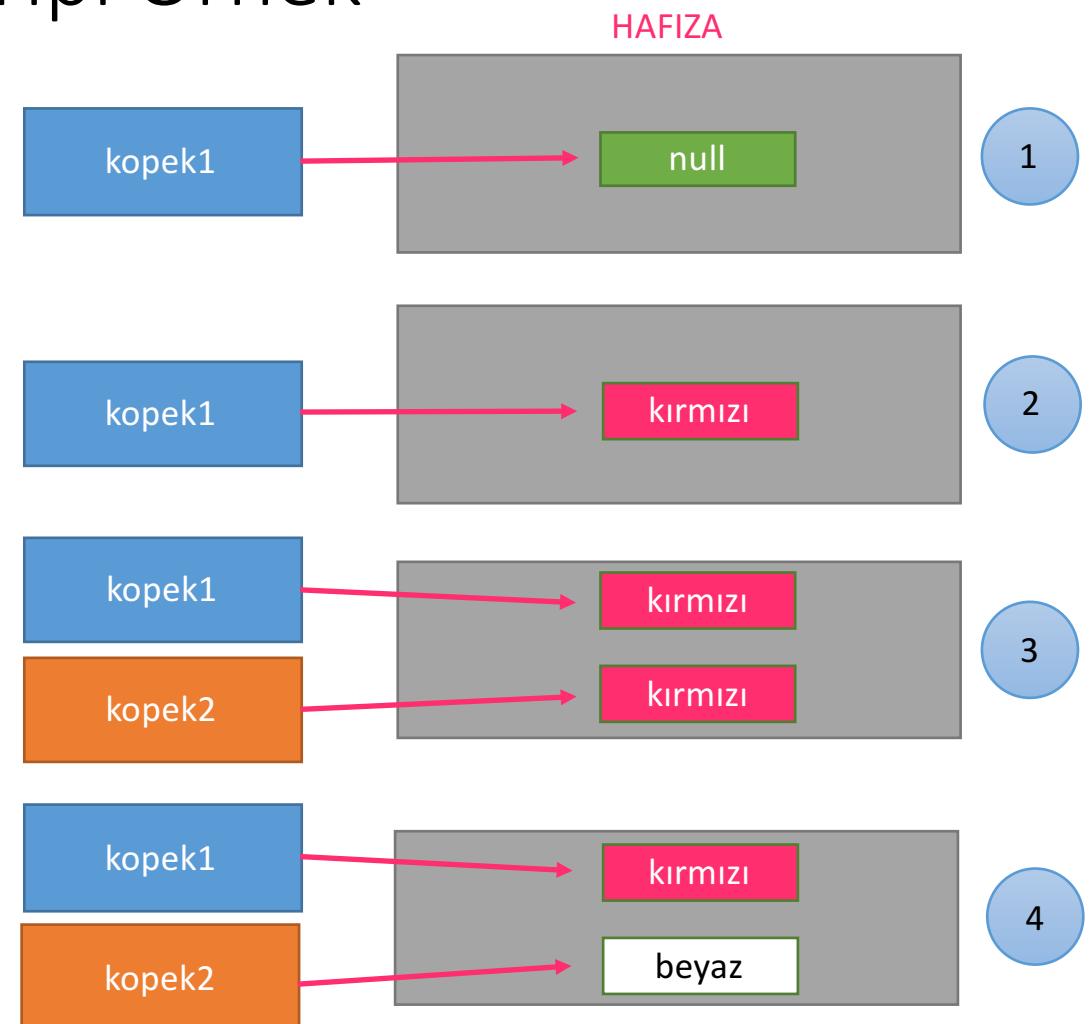
3 `var ogrenci2 = ogrenci1`  
4 `ogrenci2.ad = "Mehmet"`

```
print(ogrenci1.ad!) //Mehmet
```



# Değer Tipi Örnek

```
struct Kopek {  
    var renk:String?  
}  
  
1 var kopek1 = Kopek()  
2 kopek1.renk = "kırmızı"  
  
3 var kopek2 = kopek1  
4 kopek2.renk = "beyaz"  
  
print(kopek1.renk!) //kırmızı
```



# Import

# Import

- **import** kodlama yaparken ihtiyaç duyacağımız sınıfların kodlama yaptığımız sınıfta kullanılmasına izin verir.
- Örn : Harita işlemleri için MapKit sınıflarını import etmeliyiz.

```
import UIKit  
import Foundation  
import MapKit
```

# Access Control – Erişim Kontrolü

- <https://medium.com/@kishannakum/learn-access-control-in-swift-4-ff5e05b15904>

# Static Değişkenler ve Metodlar

# Static Değişkenler ve Metodlar

- Bir değişkenin veya metodun, bulunduğu sınıfın ( class veya structure ) nesne oluşturmaya gerek kalmadan erişilmek istenirse kullanılı

```
class Asinifi{  
    static var x = 10  
  
    static func metod(){  
        print("Merhaba")  
    }  
  
    print(Asinifi.x)  
    Asinifi.metod()  
  
}  
  
struct B{  
    static var x = 10  
  
    static func metod(){  
        print("Merhaba")  
    }  
  
    print(B.x)  
    B.metod()  
}
```

# Enumeration

# Enumeration

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

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

# Enumeration : Parametreli

```
enum Ogrenci {  
    case Ad(String)  
    case Notlar(Int,Int,Int)  
}  
  
var bilgi1 = Ogrenci.Ad("Ahmet")  
var bilgi2 = Ogrenci.Notlar(98,97,95)  
  
switch bilgi2 {  
    case .Ad(let ad):  
        print("Öğrenci Adı : \(ad).")  
    case .Notlar(let not1, let not2, let not3):  
        print("Öğrenci Notları : \(not1), \(not2), \(not3).")  
}
```

Çıktı

Öğrenci Notları : 98,97,95.

# Composition

# Composition

- Başka sınıflardan ( class veya structure ) olmuş nesneler bir sınıfın değişkeni olabilir.

```
class Adres {  
    var il:String?  
    var ilce:String?  
  
    init(il:String,ilce:String) {  
        self.il = il  
        self.ilce = ilce  
    }  
}
```

```
class Kisiler {  
    var ad:String?  
    var yas:Int?  
    var adres:Adres?  
  
    init(ad:String,yas:Int,adres:Adres) {  
        self.ad = ad  
        self.yas = yas  
        self.adres = adres  
    }  
}
```

```
var adres = Adres(il: "Bursa", ilce: "Osmangazi")  
  
var kisi = Kisiler(ad: "Ahmet", yas: 20, adres: adres)  
  
print("Kisi ad    : \$(kisi.ad!)")  
print("Kisi yaş   : \$(kisi.yas!)")  
print("Adre il    : \$(kisi.adres!.il!)")  
print("Adre ilce  : \$(kisi.adres!.ilce!)")
```

```
Kisi ad    : Ahmet  
Kisi yaş   : 20  
Adre il    : Bursa  
Adre ilce  : Osmangazi
```

# 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	Quentin 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

```

struct Filmler {
    var film_id:Int?
    var film_ad:String?
    var film_yil:Int?
    var kategori:Kategoriler?
    var yonetmen:Yonetmenler?

    init(film_id:Int,film_ad:String,film_yil:Int,kategori:Kategoriler,yonetmen:Yonetmenler) {
        self.film_id = film_id
        self.film_ad = film_ad
        self.film_yil = film_yil
        self.kategori = kategori
        self.yonetmen = yonetmen
    }
}

struct Kategoriler {
    var kategori_id:Int?
    var kategori_ad:String?

    init(kategori_id:Int,kategori_ad:String) {
        self.kategori_id = kategori_id
        self.kategori_ad = kategori_ad
    }
}

struct Yonetmenler {
    var yonetmen_id:Int?
    var yonetmen_ad:String?

    init(yonetmen_id:Int,yonetmen_ad:String) {
        self.yonetmen_id = yonetmen_id
        self.yonetmen_ad = yonetmen_ad
    }
}

var k1 = Kategoriler(kategori_id: 1, kategori_ad: "Dram")
var k2 = Kategoriler(kategori_id: 2, kategori_ad: "Komedi")
var k3 = Kategoriler(kategori_id: 3, kategori_ad: "Bilim Kurgu")

var y1 = Yonetmenler(yonetmen_id: 1, yonetmen_ad: "Nuri Bilge Ceylan")
var y2 = Yonetmenler(yonetmen_id: 2, yonetmen_ad: "Quentin Tarantino")
var y3 = Yonetmenler(yonetmen_id: 3, yonetmen_ad: "Christopher Nolan")

var f1 = Filmler(film_id: 1, film_ad: "Django", film_yil: 2013, kategori: k1, yonetmen: y2)
var f2 = Filmler(film_id: 2, film_ad: "Inception", film_yil: 2006, kategori: k3, yonetmen: y3)

print("Film id      : \uf7f5(f1.film_id!)")
print("Film ad       : \uf7f5(f1.film_ad!)")
print("Film yil      : \uf7f5(f1.film_yil!)")
print("Film kategori : \uf7f5(f1.kategori!.kategori_ad!)")
print("Film yönetmen : \uf7f5(f1.yonetmen!.yonetmen_ad!)")

```

Film id : 1
Film ad : Django
Film yil : 2013
Film kategori : Dram
Film yönetmen : Quentin Tarantino

# Kalıtım ( Inheritance )

# OOP Kuralı – Kalıtım ( Inheritance )

- Mevcut bir sınıfın başka bir sınıf türetmek için kullanılır.
- Kodun tekrar kullanabilirliğini artırır.
- Sadece **class** için geçerlidir.
  - : işaretini ile tanımlanır.
- Bir sınıfın tek kalıtımı olabilir.
- Bir sınıfa birden fazla sınıf kalıtım yolu **class Araba:Arac{ }** ile bağlanamaz.
- Üst sınıfa **superclass** denir.
- Alt sınıfa **subclass** denir.

```
class Arac{  
}
```



```
class Araba:Arac{  
}
```



```
class Nissan:Araba{  
}
```



```

class Arac{
    var renk:String?
    var vites:String?

    init(renk:String,vites:String) {
        self.renk = renk
        self.vites = vites
    }
}

class Araba:Arac{
    var kasaTipi:String?
    Kalıtım yoluyla oluşturulan
    sınıfın constructor'ı üst sınıfın
    özelliklerini almalıdır.

    init(renk: String, vites: String,kasaTipi:String) {
        self.kasaTipi = kasaTipi
        super.init(renk: renk, vites: vites)
    }
}

class Nissan:Araba{
    var model:String?

    init(renk: String, vites: String, kasaTipi: String,model:String) {
        self.model = model
        super.init(renk: renk, vites: vites, kasaTipi: kasaTipi)
    }
}

```

## Örnek :

Araç

Araba

Nissan

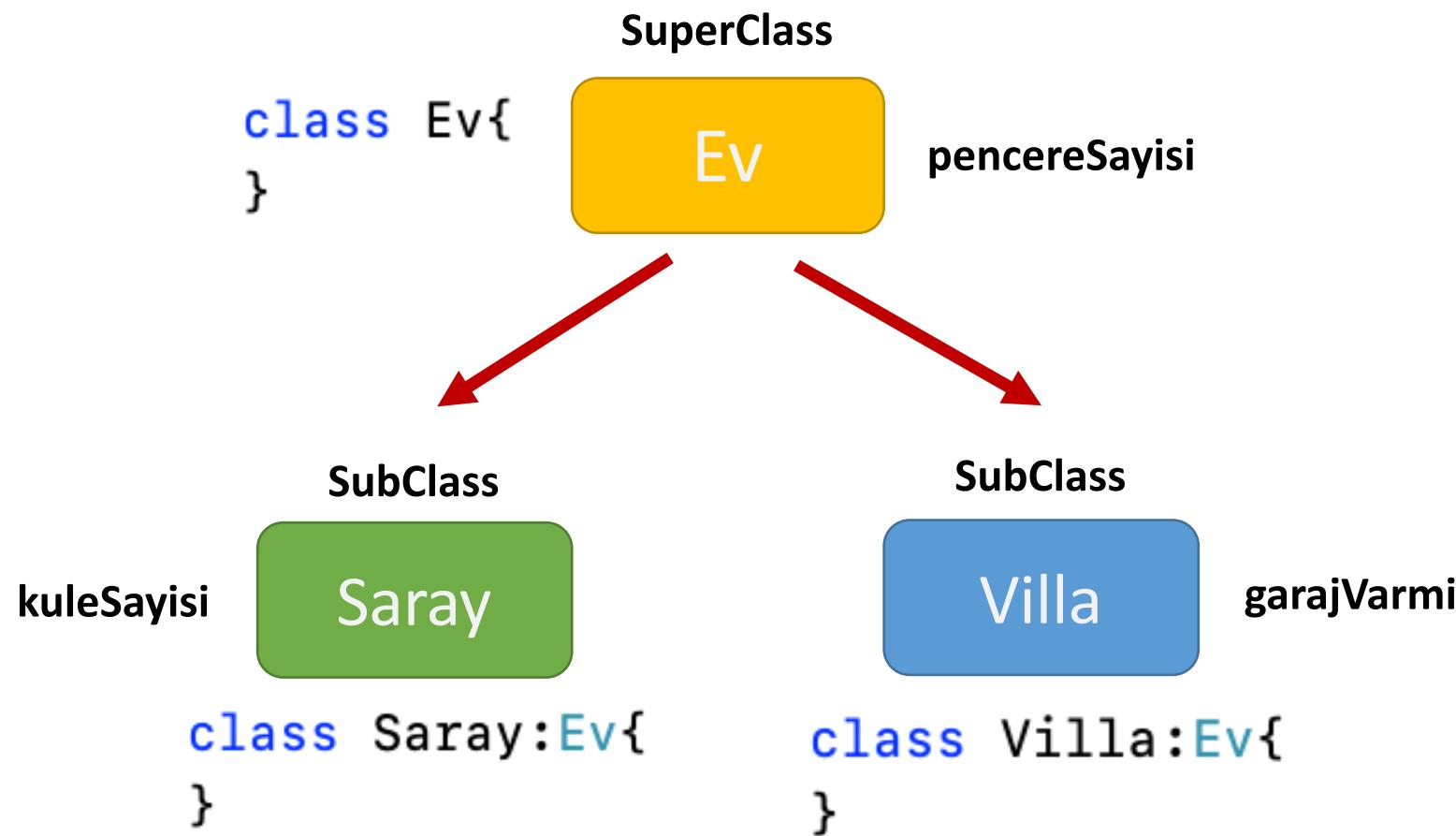
super ile üst sınıfa  
constructor ile alınan  
değişkenler gönderilir.

```

var araba = Araba(renk: "Kırmızı", vites: "Otomatik", kasaTipi: "Sedan")
print(araba.renk!)
print(araba.vites!)
print(araba.kasaTipi!)
print(araba.model) //Üst sınıf alt sınıf'a erişemez.

```

# Kalıtım Hiyerarşisi Örnek



# Kalıtım Hiyerarşisi Örnek

```
class Ev{  
    var pencereSayisi:Int?  
  
    init(pencereSayisi:Int) {  
        self.pencereSayisi = pencereSayisi  
    }  
}  
  
  
  
class Saray:Ev{  
    var kuleSayisi:Int?  
  
    init(pencereSayisi: Int,kuleSayisi:Int) {  
        self.kuleSayisi = kuleSayisi  
        super.init(pencereSayisi: pencereSayisi)  
    }  
}  
  
class Villa:Ev{  
    var garajVarmi:Bool?  
  
    init(pencereSayisi: Int,garajVarmi:Bool) {  
        self.garajVarmi = garajVarmi  
        super.init(pencereSayisi: pencereSayisi)  
    }  
}
```

# Kalıtım Hiyerarşisi Örnek

```
class Ev{  
    var pencereSayisi:Int?  
  
    init(pencereSayisi:Int) {  
        self.pencereSayisi = pencereSayisi  
    }  
}  
  
class Saray:Ev{  
    var kuleSayisi:Int?  
  
    init(pencereSayisi: Int,kuleSayisi:Int) {  
        self.kuleSayisi = kuleSayisi  
        super.init(pencereSayisi: pencereSayisi)  
    }  
}  
  
class Villa:Ev{  
    var garajVarmi:Bool?  
  
    init(pencereSayisi: Int,garajVarmi:Bool) {  
        self.garajVarmi = garajVarmi  
        super.init(pencereSayisi: pencereSayisi)  
    }  
}  
  
var topkapiSarayi = Saray(pencereSayisi: 30, kuleSayisi: 5)  
var bogazVilla = Villa(pencereSayisi: 4, garajVarmi: true)  
  
print(topkapiSarayi.pencereSayisi!) //30  
print(topkapiSarayi.kuleSayisi!) //5  
  
print(topkapiSarayi.garajVarmi!)  
//Saray ile Villa arasında Kalıtım olmadığı için erişim olmaz  
  
print(bogazVilla.pencereSayisi!) //4  
print(bogazVilla.garajVarmi!) //true  
  
print(bogazVilla.kuleSayisi!)  
//Saray ile Villa arasında Kalıtım olmadığı için erişim olmaz
```

# Kalıtım ilişkisinde Constructor

# Kalıtım ilişkisinde Initialization - Constructor

- **super** kelimesi ile üst sınıfın constructor'ına ( init ) erişilir.

```
class Arac{  
    var renk:String?  
    var vites:String?  
  
    init(renk:String,vites:String) {  
        self.renk = renk  
        self.vites = vites  
    }  
  
    super ile üst sınıfa  
    constructor ile alınan  
    değişkenler gönderilir.  
}
```

```
class Araba:Arac{  
    var kasaTipi:String?  
  
    init(renk: String, vites: String,kasaTipi:String) {  
        self.kasaTipi = kasaTipi  
        super.init(renk: renk, vites: vites)  
    }  
  
    class Nissan:Araba{  
        var model:String?  
  
        init(renk: String, vites: String, kasaTipi: String,model:String) {  
            self.model = model  
            super.init(renk: renk, vites: vites, kasaTipi: kasaTipi)  
        }  
    }  
}
```

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

Aslında bu kod yapısı üst sınıfın constructor'ını çalıştırır  
yani üst sınıftan nesne oluşturur.

# Kalıtım İlişkisinde Initialization - Constructor

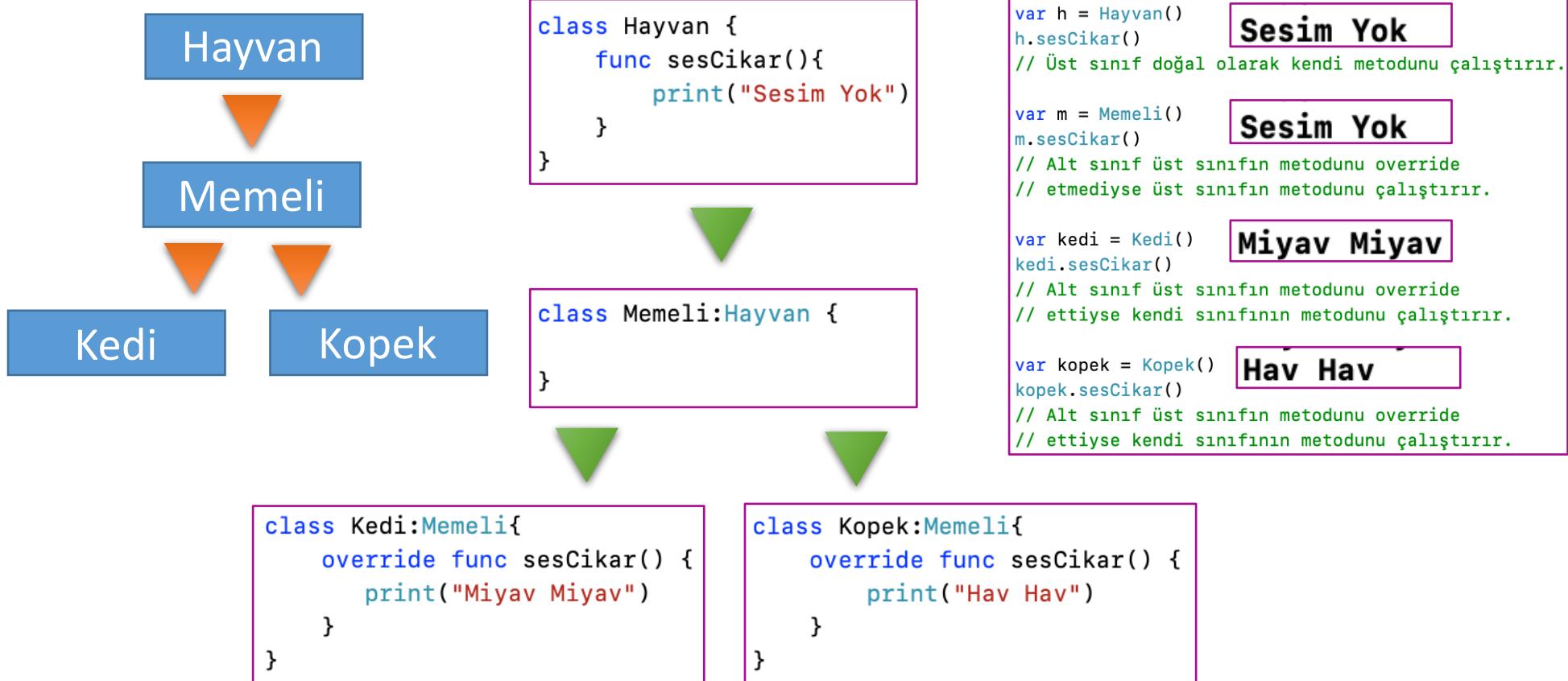
Örnek



# Override

# Metodları Ezme : Overriding

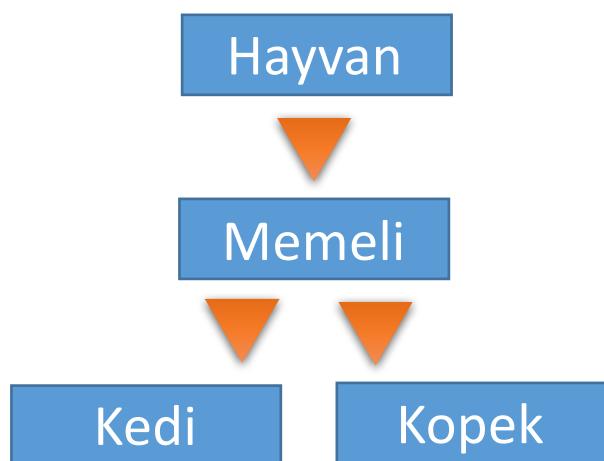
- Kalıtım ilişkisinde üst sınıfın metodlarının alt sınıf tarafından tekrar kullanılmasıdır.



# PolyMorphism

# PolyMorphism

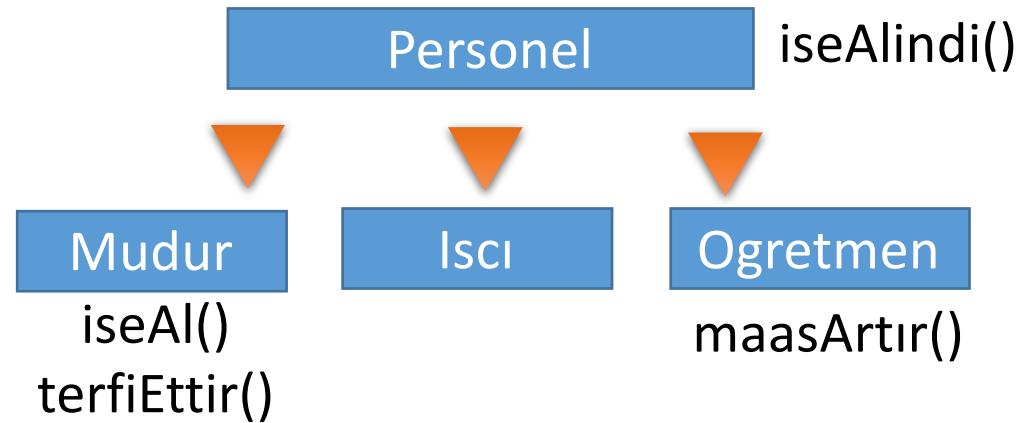
- PolyMorphism olması için iki sınıf arasında kalıtım ilişkisi olmalıdır.
- Daha kapsayıcı bir kullanım sağlamak için kullanılır.
- Özellikle metodların parametrelerinde PolyMorphism kullanılarak daha kapsayıcı veriler alınabilir.
- Superclass gibi görünüp subclass gibi davranır.



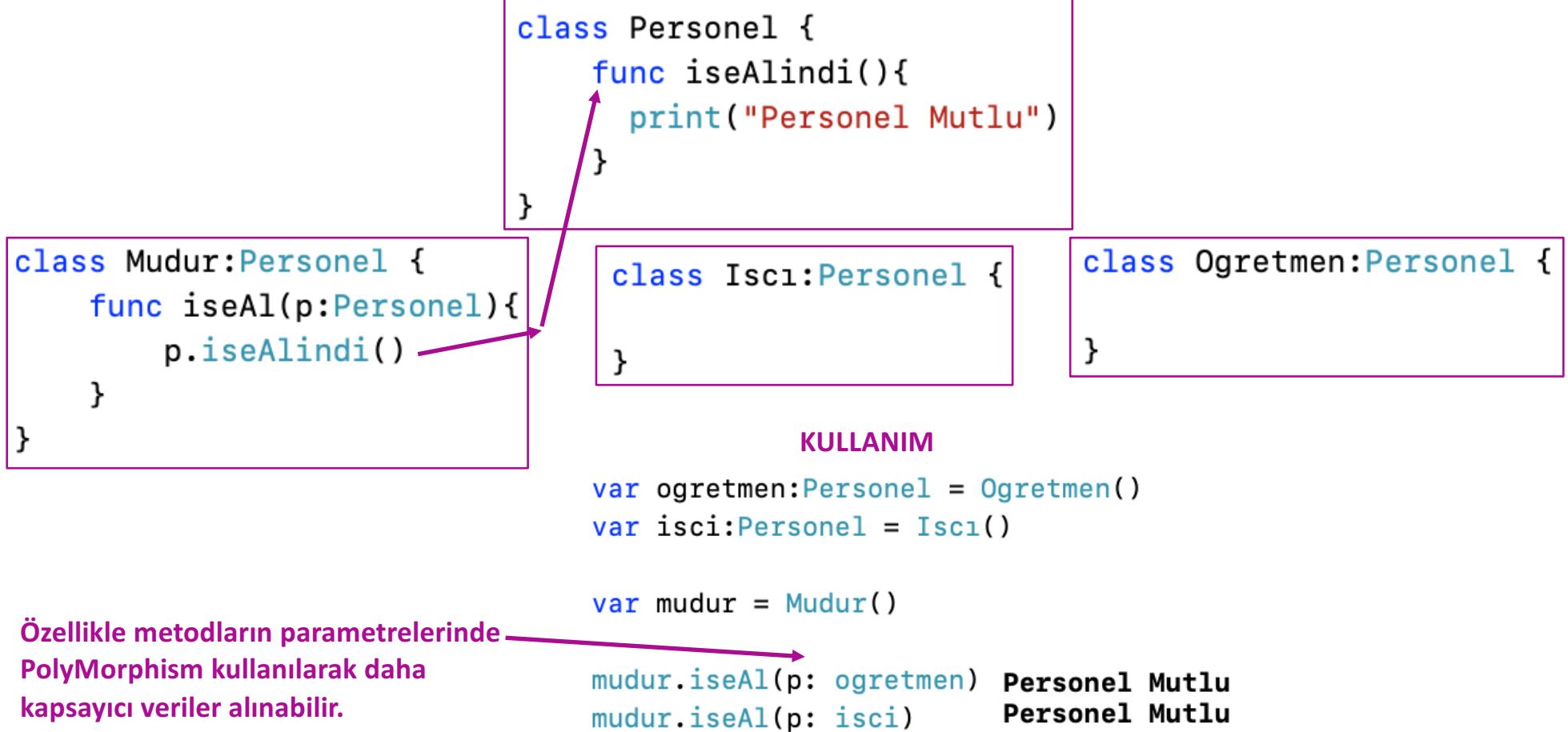
```
//POLYMORFİSM
var hayvan:Hayvan = Kopek()
hayvan.sesCikar()
```

Hav Hav

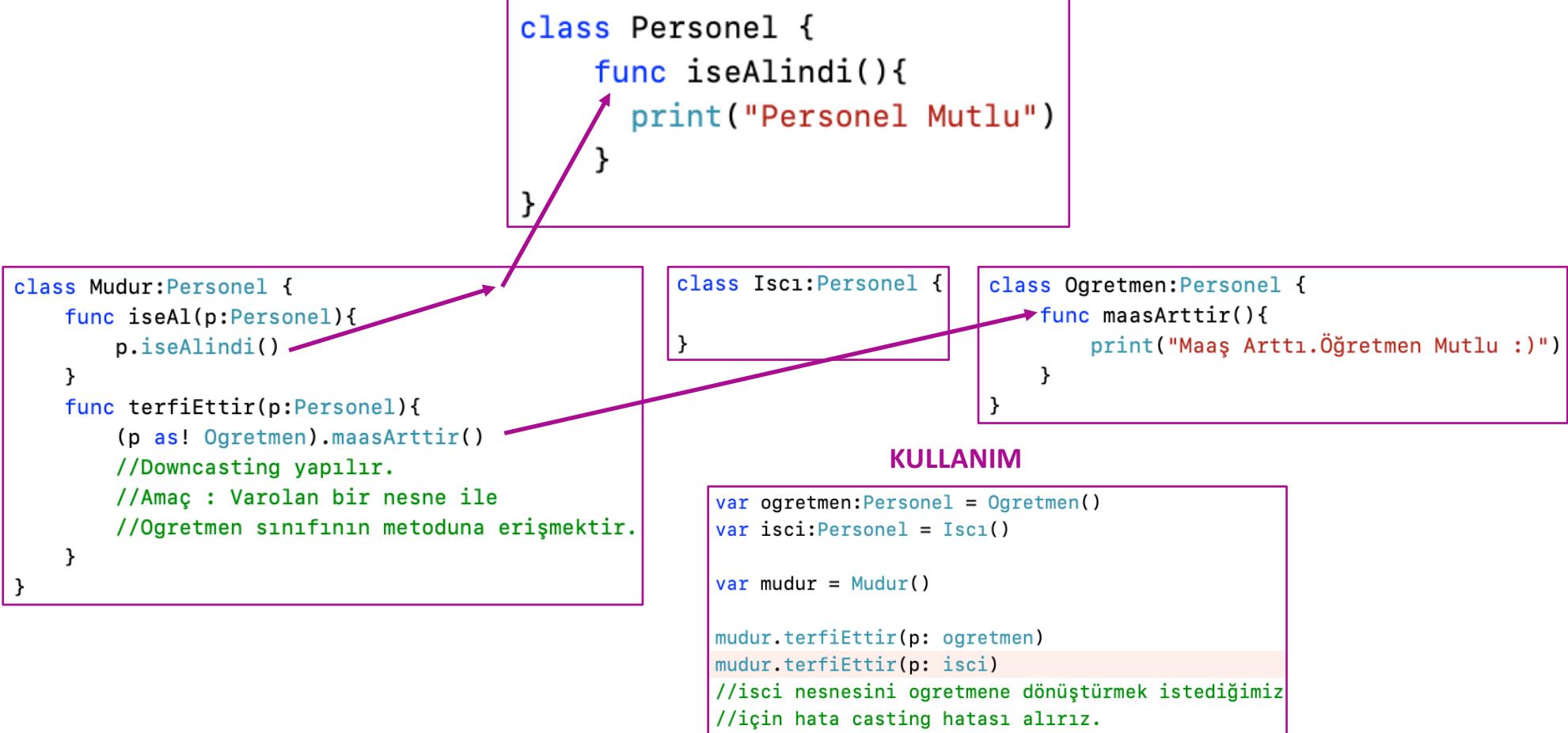
# PolyMorphism Örnek



# PolyMorphism Örnek



# PolyMorphism Örnek



# Nesnelerin Tip Dönüşümü

## is, as, as?, as!

- **is** : (**Type Checking**) Tip kontrolü için kullanılır.(javadaki instance of metodu gibi)
- **as** : (**Upcasting**) bir tipi başka bir tipe dönüştürmek için kullanılır
- **as !** : (**Force Downcasting**) bir tipi başka bir tipe dönüştürmek için kullanılır. Dönüşüm sırasında başarısız olursa hata, başarılı ise değeri dönüştürür.
- **as ?** : (**For Optional downcasting**) Dönüşüm sırasında optional bir veri geliyorsa. Dönüşüm sırasında başarısız olursa nil , başarılı ise değeri dönüştürür.

# Tip Kontrolü - **is**

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

```
let topkapi:Saray = Saray(pencereSayisi: 300, kuleSayisi: 1)

if topkapi is Saray {
    print("Bu saraydır!")
} else {
    print("Bu saray değildir!")
}
```

ÇIKTI :

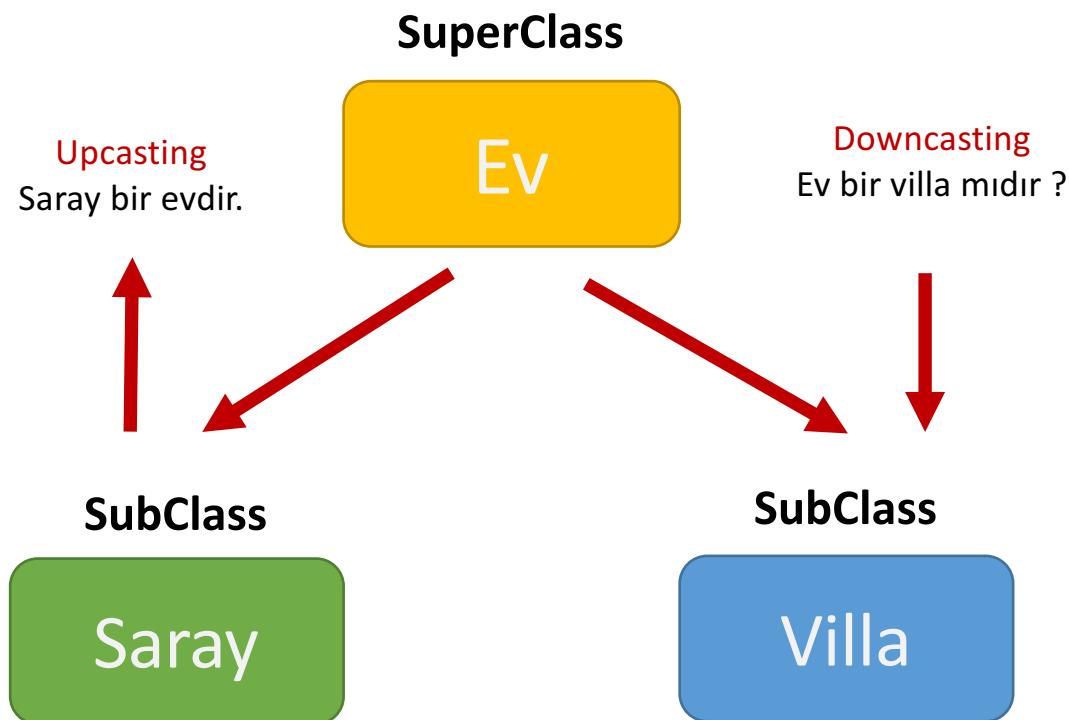
Bu saraydır!

# Downcasting – UpCasting



## Upcasting - **as**

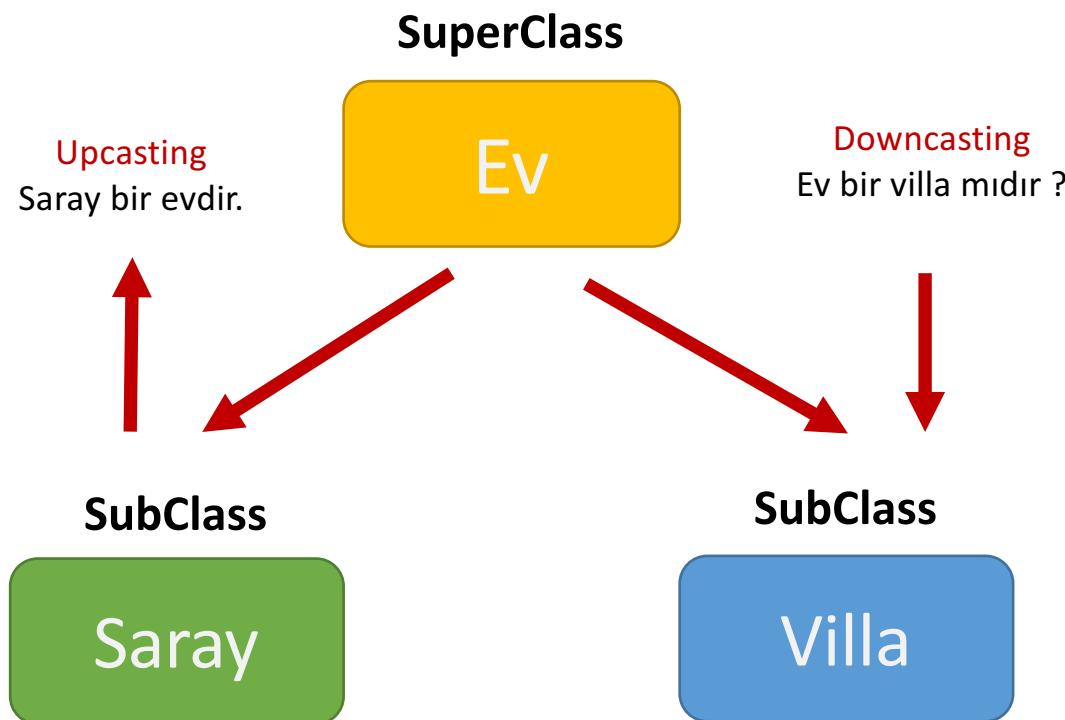
```
let ev1:Ev = Saray(pencereSayisi: 200, kuleSayisi: 4) as Ev
```



# Downcasting – as!

```
var ev = Ev(pencereSayisi: 5)
```

```
var saray:Saray = ev as! Saray
```

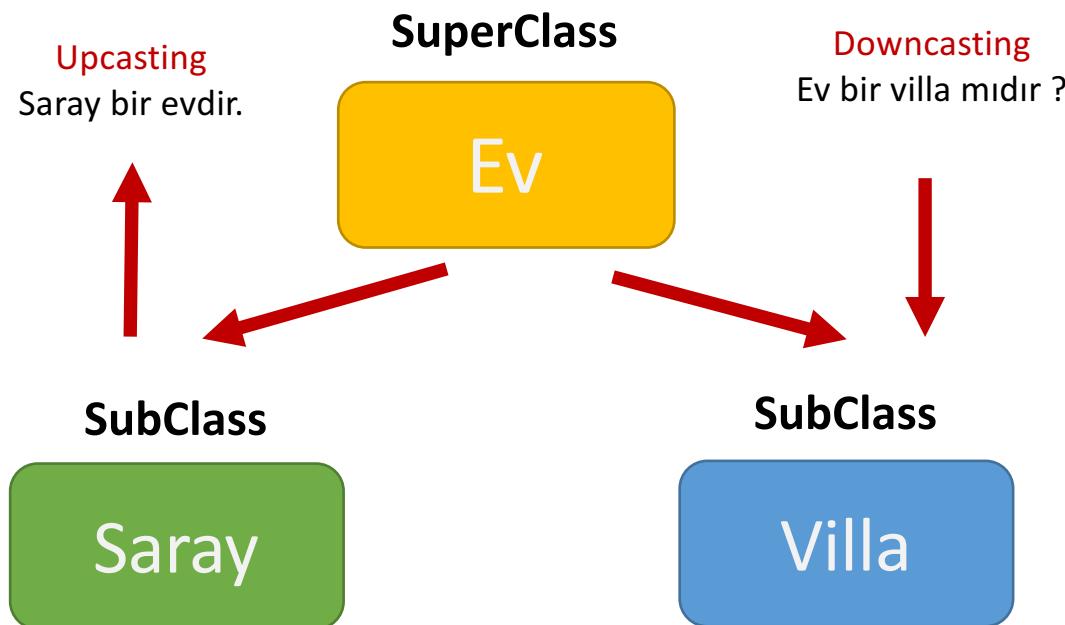


# Downcasting – as? – Optional değer

```
var ev = Ev(pencereSayisi: 5)
```

```
var saray:Saray? = ev as? Saray
```

Dönüşüm sırasında optional bir veri geliyorsa. Dönüşüm sırasında başarısız olursa nil , başarılı ise değeri dönecektir.



# Tip Dönüşümünde ?? ile varsayılan değer tanımlama

```
let arkadasListesi = d.array(forKey: "liste") as? [String] ?? [String]()
//as? den sonra gelen dönüştürülecek tip.
//?? den sonra gelen userdefaulttan veri alırken sıkıntı olursa
//varsayılan olarak boş bir string liste aktarılır.
```

```
let sehirlerListe = d.dictionary(forKey: "dict") as? [String:String] ?? [String:String]()

```

Dictionary dönüşümü                              Boş Dictionary  
yani varsayılan

# PolyMorphism Casting Hatası Önleme

## Tip Kontrolü **is**

```
class Mudur:Personel {  
    func iseAl(p:Personel){  
        p.iseAlindi()  
    }  
    func terfiEttir(p:Personel){  
  
        if p is Ogretmen{  
            (p as! Ogretmen).maasArattir()  
        }  
  
        if p is Isci{  
            print("İşçiler terfi alamaz")  
        }  
    }  
}
```

### KULLANIM

```
var ogretmen:Personel = Ogretmen()  
varisci:Personel = Isci()  
  
var mudur = Mudur()  
  
mudur.terfiEttir(p: ogretmen)  
mudur.terfiEttir(p:isci)  
//isci nesnesini ogretmene dönüştürmek istediğimiz  
//için hata casting hatası alırız.
```

Maaş Arttı. Öğretmen Mutlu :)  
İşçiler terfi alamaz

# Extension

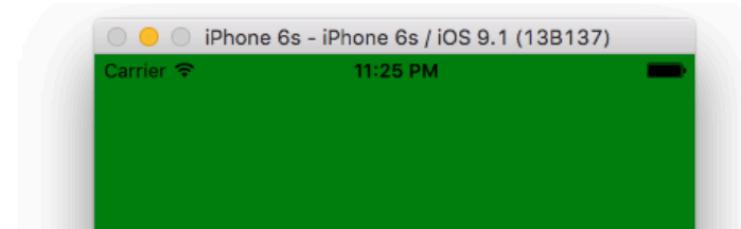
# Extension

- Var olan bir yapıyı kendimize göre genişletmemize yaramaktadır.
- Bu oluşturduğumuz yapının detaylarının görülmesini engelleyerek kod koruması sağlar.
- Bir swift dosya içine yazılarak her yerden erişilebilir.
- Barındırdığı bütün metodlar ve değişkenlerin dönüş tipi extension türünde olmalıdır.
- Örn: Yeni bir renk oluşturmak gibi.

```
extension UIColor {  
  
    var customGreen: UIColor {  
        let darkGreen = 0x008110  
        return UIColor.rgb(fromHex: darkGreen)  
    }  
  
    func rgb(fromHex: Int) -> UIColor {  
  
        let red = CGFloat((fromHex & 0xFF0000) >> 16) / 0xFF  
        let green = CGFloat((fromHex & 0x00FF00) >> 8) / 0xFF  
        let blue = CGFloat(fromHex & 0x0000FF) / 0xFF  
        let alpha = CGFloat(1.0)  
  
        return UIColor(red: red, green: green, blue: blue, alpha: alpha)  
    }  
}
```

extension Tür {  
 Kodlama Alanı }

Kullanım  
view.backgroundColor = UIColor.customGreen



# Extension - Değişken Örnek

```
//Metre dönüşümü
extension Double {
    var km : Double { return self * 1000.0 }
    var m : Double { return self }
    var cm : Double { return self / 100.0 }
    var mm : Double { return self / 1000.0 }
    //self dışarıdan verilen parametreyi temsil eder
}

print("10 cm : \(10.cm) metredir.")
print("20 km : \(20.km) metredir.")

let sayı = 30.0
//Verilen sayı double olmalıdır.
print("30 mm : \(sayı.mm) metredir.")
```

Bu değişkenler sadece  
türü Double değişkenler  
için çalışır.

10 cm : 0.1 metredir.  
20 km : 20000.0 metredir.  
30 mm : 0.03 metredir.

# Extension – Metod Örnek

```
extension String {  
  
    func yerdegistir(yeniHarf: String, eskiHarf: String) -> String {  
        return self.replacingOccurrences(of: yeniHarf, with: eskiHarf)  
    }  
}  
let str = "ankara".yerdegistir(yeniHarf: "a", eskiHarf: "e")  
print(str) // enkere  
  
var meyve = "incir"  
  
let str1 = meyve.yerdegistir(yeniHarf: "i", eskiHarf: "x")  
print(str1) // xncxr
```

Bu metodlar sadece türü  
String değişkenler için çalışır.

# Protocol

# Protocol

- Hem class hem structure yapısında kullanılabilir.
- Bir sınıf ( class ve structure ) birden fazla protocol alabilir.

```
class ViewController: UIViewController, UITableViewDataSource,  
UITableViewDelegate { }
```

- : ile eklenirler.
- Hazır taslaklar gibi düşünebilirsiniz.
- Javada **interface** olarak bilinir.
- Protocoller sınıflara özellik katar.

```
protocol Protocol1 {  
    var degisken : Int { get set }  
    func metod1()  
    func metod2() -> String  
}
```

# Protocol Örnek

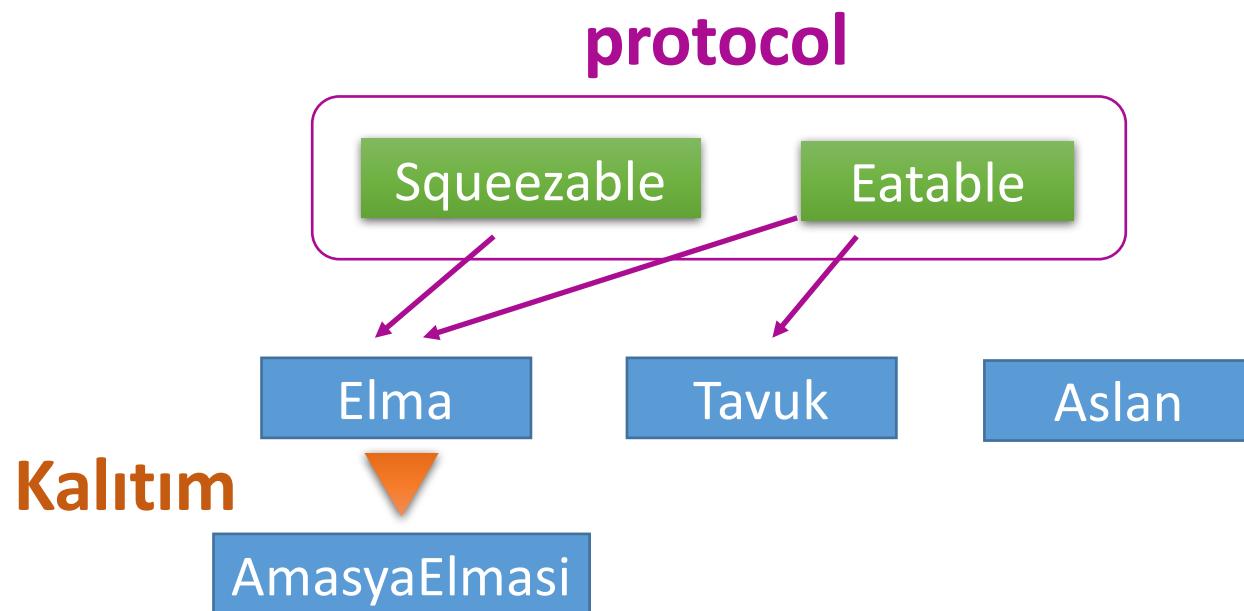
```
protocol Protocol1 {  
  
    var degisken : Int { get set }  
  
    func metod1()  
    func metod2() -> String  
}
```

```
class ClassA:Protocol1 {  
  
    var degisken = 10  
  
    func metod1() {  
        print("Protocol Merhaba")  
    }  
  
    func metod2() -> String {  
        return "Protocol Çalışması"  
    }  
}
```

```
var a = ClassA()  
  
print(a.degisken)  
a.metod1()  
print(a.metod2())
```

10  
Protocol Merhaba  
Protocol Çalışması

# Protocol Örnek



# Protocol Örnek

```
protocol Squeezable {  
    func howToSqueeze()  
}
```

```
protocol Eatable {  
    func howToEat()  
}
```

```
class Elma:Eatable,Squeezable{  
    func howToEat() {  
        print("Dilimle ve ye")  
    }  
    func howToSqueeze() {  
        print("Blendır ile sık")  
    }  
}
```

```
class Tavuk:Eatable{  
    func howToEat() {  
        print("Fırında kızart")  
    }  
}
```

```
class Aslan{  
}
```

```
class AmasyaElmasi:Elma{  
    override func howToEat() {  
        print("Yıka ve ye")  
    } //Bu metod protocol ile değil  
    //kalıtım ile geldi  
}
```

```
var aslan = Aslan()  
var amasyaElmasi:Elma = AmasyaElmasi()  
var elma = Elma()  
var tavuk:Eatable = Tavuk()  
  
var nesneler = [aslan,amasyaElmasi,elma,tavuk] as [Any]  
  
for nesne in nesneler {  
    if nesne is Eatable {  
        (nesne as! Eatable).howToEat()  
    }  
  
    if nesne is Squeezable {  
        (nesne as! Squeezable).howToSqueeze()  
    }  
}
```

Yıka ve ye  
Blendır ile sık  
Dilimle ve ye  
Blendır ile sık  
Fırında kızart

# Closure

# Closure

- Basit anlamda parantezler ile bir arada tutulan hazır kodlama yapılarıdır.
- Fonksiyonlar gibi çalışan kod yapılarıdır.
- Java ve diğer dillerde yer alan lambda yapısı ile aynıdır.



```
{  
    ( parametre ) -> dönüş tipi in  
        kod ifadesi  
}
```

# Closure Örnek : Geri Dönüş Olmayan

```
let ifade = {  
    print("Closure Konusuna Hoşgeldin")  
}  
  
ifade()
```

Çıktı

## Closure Konusuna Hoşgeldin

# Closure Örnek : Geri Dönüş Olan

```
let toplama = {  
    (sayi1: Int, sayi2: Int) -> Int in  
    return sayi1+sayi2  
}  
  
let sonuc = toplama(10, 20)  
  
print (sonuc)
```

Çıktı

# Closure Örnek : Sıralama Şartları Yazma

Bu örnekte yer alan sayı1 ve sayı2 dizi içinde sırayla gelecek olan sayıları temsil etmektedir.

```
var sayılar:[Int] = [5, 10, -6, 75, 20]
let siralama1 = sayılar.sorted(by: { sayı1, sayı2 in sayı1 > sayı2 })
let siralama2 = sayılar.sorted(by: { n1, n2 in n1 < n2 })
print(siralama1)//Büyükten küçüğe sıralama
print(siralama2)//Küçükten büyüğe sıralama
```

## Çıktı

```
[ 75, 20, 10, 5, -6 ]
[-6, 5, 10, 20, 75]
```

Not : Temsili ifadeler parametreleri temsil eder ve istenilen isim verilebilir fakat verilmiş isimlere göre kodlama yapılmalıdır.

## Closure Örnek : Kısaltma

```
var sayilar:[Int] = [5, 10, -6, 75, 20]

let siralama1 = sayilar.sorted(by: { $0 > $1 })

let siralama2 = sayilar.sorted(by: < ) Parantez olmadan ifade yazımı.

print(siralama1)//Büyükten küçüğe sıralama

print(siralama2)//Küçükten büyüğe sıralama
```

\$ işaretini closure a gelen ilk ve son ifadeyi temsil etmektedir.

Çıktı

```
[ 75, 20, 10, 5, -6 ]
[-6, 5, 10, 20, 75]
```

# Nesne Tabanlı Listeme

# Array

# Array [ ]

- Aynı türde verileri bir arada tutar.
- İndeks numaraları 0 dan başlar.
- Array tanımlarken türü belirtilmelidir.

```
var meyveler:[String] = ["Çilek", "Muz", "Elma", "Kivi", "Kiraz"]
```

meyveler	
İndeks	Değer
0	Çilek
1	Muz
2	Elma
3	Kivi
4	Kiraz

# Örnek

```
class Ogrenci{
    var no:Int?
    var ad:String?
    var sinif:String?

    init(no:Int,ad:String,sinif:String) {
        self.no = no
        self.ad = ad
        self.sinif = sinif
    }
}

var o1 = Ogrenci(no: 100, ad: "Ahmet", sinif: "11F")
var o2 = Ogrenci(no: 90, ad: "Zeynep", sinif: "10R")
var o3 = Ogrenci(no: 130, ad: "Ceyda", sinif: "12A")
var o4 = Ogrenci(no: 150, ad: "Mehmet",sinif: "9Z")
var o5 = Ogrenci(no: 110, ad: "Yasin", sinif: "11F")

var ogrenciListesi = [Ogrenci]()

ogrenciListesi.append(o1)
ogrenciListesi.append(o2)
ogrenciListesi.append(o3)
ogrenciListesi.append(o4)
ogrenciListesi.append(o5)

for ogrenci in ogrenciListesi {
    print("*****")
    print("Öğrenci No : \(\ogrenci.no!\"")
    print("Öğrenci Ad : \(\ogrenci.ad!\"")
    print("Öğrenci Sınıf : \(\ogrenci.sinif!\"")
}
```

## Array sort()

```
print("Sayısal Büyüktен Küçüğe")
let siralamaArray1 = kisilerArray.sorted(by: {$0.kisiNo > $1.kisiNo} )
```

```
print("Sayısal Küçükten Büyüge")
let siralamaArray2 = kisilerArray.sorted(by: {$0.kisiNo < $1.kisiNo} )
```

```
print("Harf Küçükten Büyüge")
let siralamaArray3 = kisilerArray.sorted(by: {$0.kisiAd < $1.kisiAd} )
```

# Örnek

```
class Kisiler {
    var kisiNo:Int?
    var kisiAd:String?

    init(kisiNo:Int,kisiAd:String) {
        self.kisiAd = kisiAd
        self.kisiNo = kisiNo
    }
}

let kisi1 = Kisiler(kisiNo: 1,kisiAd: "Ahmet")
let kisi2 = Kisiler(kisiNo: 2,kisiAd: "Zeynep")
let kisi3 = Kisiler(kisiNo: 3,kisiAd: "Berna")

var kisilerArray = [Kisiler]()

kisilerArray.append(kisi1)
kisilerArray.append(kisi2)
kisilerArray.append(kisi3)
```

Önce  
1 - Ahmet  
2 - Zeynep  
3 - Berna  
Sayısal Büyüktен Küçüğe  
3 - Berna  
2 - Zeynep  
1 - Ahmet  
Sayısal Küçükten Büyüge  
1 - Ahmet  
2 - Zeynep  
3 - Berna  
Harf Küçükten Büyüge  
1 - Ahmet  
3 - Berna  
2 - Zeynep

```
print("Önce")
for k in kisilerArray {
    print("\(k.kisiNo) - \(k.kisiAd)")
}

print("Sayısal Büyüktен Küçüğe")
let siralamaArray1 = kisilerArray.sorted(by: {$0.kisiNo > $1.kisiNo} )

for k in siralamaArray1 {
    print("\(k.kisiNo) - \(k.kisiAd)")
}

print("Sayısal Küçükten Büyüge")
let siralamaArray2 = kisilerArray.sorted(by: {$0.kisiNo < $1.kisiNo} )

for k in siralamaArray2 {
    print("\(k.kisiNo) - \(k.kisiAd)")
}

print("Harf Küçükten Büyüge")
let siralamaArray3 = kisilerArray.sorted(by: {$0.kisiAd < $1.kisiAd} )

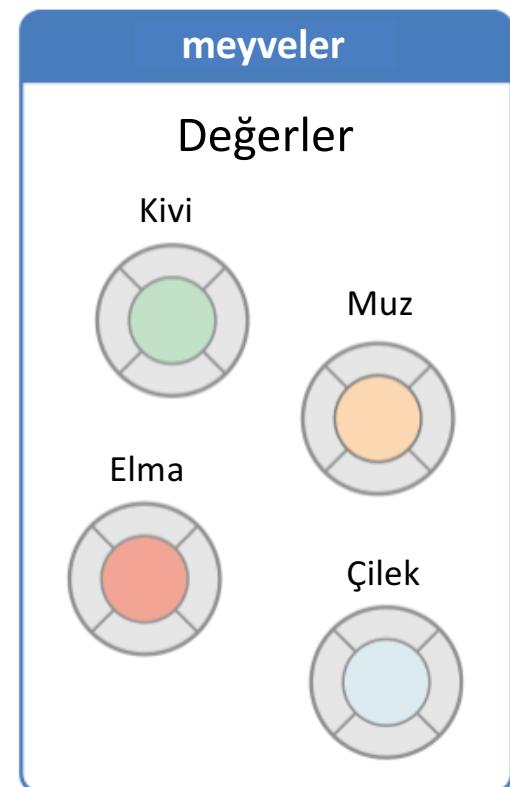
for k in siralamaArray3 {
    print("\(k.kisiNo) - \(k.kisiAd)")
}
```

# Set

# Set

- Array ile aynı özelliklere sahiptir.
- İçine eklenen veriler düzensiz rasgele yerleştirilir.
- İndeks değerlerinin takibi zordur.

```
let meyveler: Set = ["Çilek", "Muz", "Elma", "Kivi"]
```



## Nesne Tabanlı - Set

- Set yapı itibarı ile içine insert edilen verileri rasgele sıralamaktadır.
- Bu rasgele sıralama int,string içeren set gibi ifadelerde kolaylıkla yapılabiliyor.
- Fakat set içine nesne yerleştirildiğinde nesne içindeki hangi değişkene göre bu rasgele sıralamayı yapacağını belirtmemiz gerekiyor.
- Bundan dolayı protocol olarak **Equatable,Hashable** kullanılır.
- Örn : Öğrencinin nosuna göre mi ? adına göre mi ? sınıfına göre mi ? sıralama yapılacak

```
var o1 = Ogrenci(no: 100, ad: "Ahmet", sinif: "11F")
var o2 = Ogrenci(no: 90, ad: "Zeynep", sinif: "10R")
var o3 = Ogrenci(no: 130, ad: "Ceyda", sinif: "12A")
var o4 = Ogrenci(no: 150, ad: "Mehmet",sinif: "9Z")
var o5 = Ogrenci(no: 110, ad: "Yasin", sinif: "11F")
```

# Örnek

```
class Ogrenci: Equatable, Hashable{
    var no:Int?
    var ad:String?
    var sinif:String?

    init(no:Int, ad:String, sinif:String) {
        self.no = no
        self.ad = ad
        self.sinif = sinif
    }

    var hashCode: Int { //Hashable için gerekli metod
        get {
            return no.hashCode
            //Öğrenci no suna göre
        }
    }

    static func == (lhs: Ogrenci, rhs: Ogrenci) -> Bool {
        return lhs.no == rhs.no ////Öğrenci no suna göre
    } //Equatable için gerekli metod
}
```

```
//NOT : SET içinde nesne kullanılacak ise nesnenin ait olduğu sınıf
//protocol olarak Equatable, Hashable özelliğine sahip olmalıdır.

var o1 = Ogrenci(no: 100, ad: "Ahmet", sinif: "11F")
var o2 = Ogrenci(no: 90, ad: "Zeynep", sinif: "10R")
var o3 = Ogrenci(no: 130, ad: "Ceyda", sinif: "12A")
var o4 = Ogrenci(no: 150, ad: "Mehmet", sinif: "9Z")
var o5 = Ogrenci(no: 110, ad: "Yasin", sinif: "11F")

var ogrenciListesi = Set <Ogrenci>() //SET

ogrenciListesi.insert(o1)
ogrenciListesi.insert(o2)
ogrenciListesi.insert(o3)
ogrenciListesi.insert(o4)
ogrenciListesi.insert(o5)

for ogrenci in ogrenciListesi {
    print("*****")
    print("Öğrenci No : \(ogrenci.no!)")
    print("Öğrenci Ad : \(ogrenci.ad!)")
    print("Öğrenci Sınıf : \(ogrenci.sinif!)")
}
```

# Dictionary

# Dictionary

- Boş dictionary [:] temsil edilir.
- Javadaki hash map yapısının aynısıdır.
- Key ve value ilişkisi vardır.

```
var dic1 = [Int:String]()
```

```
var dic2 = [3.14:"Pi", 2.71:"e"]
```

```
var dic3:[Int:String] = [1:"Bir", 2:"İki", 3:"Üç"]
```

# Örnek

```
class Ogrenci{  
    var no:Int?  
    var ad:String?  
    var sinif:String?  
  
    init(no:Int,ad:String,sinif:String) {  
        self.no = no  
        self.ad = ad  
        self.sinif = sinif  
    }  
}  
  
var o1 = Ogrenci(no: 100, ad: "Ahmet", sinif: "11F")  
var o2 = Ogrenci(no: 90, ad: "Zeynep", sinif: "10R")  
var o3 = Ogrenci(no: 130, ad: "Ceyda", sinif: "12A")  
var o4 = Ogrenci(no: 150, ad: "Mehmet",sinif: "9Z")  
var o5 = Ogrenci(no: 110, ad: "Yasin", sinif: "11F")  
  
var ogrenciListesi = [Int:Ogrenci](){//DICTIONARY  
    ogrenciListesi[o1.no!] = o1 //veri ekleme  
    ogrenciListesi[o2.no!] = o2  
    ogrenciListesi[o3.no!] = o3  
    ogrenciListesi[o4.no!] = o4  
    ogrenciListesi[o5.no!] = o5  
  
    for ( ogrenciNo,nesne) in ogrenciListesi {  
        print("*****")  
        print("Öğrenci No : \(ogrenciNo)")  
        print("Öğrenci Ad : \(nesne.ad!)")  
        print("Öğrenci Sınıf : \(nesne.sinif!)")  
    }  
}
```

Teşekkürler...



kasim-adalan



kasimadalan@gmail.com



kasimadalan