

Bölüm 4: Kalıtım

Java ile Nesne Yönelimli Programlama

Üst ve alt sınıf

extends

super ve yapıcı zinciri

Geçersiz kılma ve @Override

protected

final

is-a ve has-a

Çoklu kalıtım

Object sınıfı

toString, equals, hashCode

instanceof

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

Öğrenme Çıktıları

Kavram

Kalıtımı (inheritance), üst sınıf ve alt sınıf kavramlarını tanımlamak; "is-a" ilişkisini tanımak.

Yapıcılar

`super(...)` ve `super.metot()` kullanmak; yapıcı zincirinin çalışma sırasını izlemek.

Erişim ve sınırlama

`protected` erişimini ve `final` ile kalıtımın nasıl sınırlandığını açıklamak.

Söz dizimi

`extends` ile alt sınıf yazmak; neyin kalıtımla alındığını, neyin alınmadığını açıklamak.

Geçersiz kılma

Metodu geçersiz kılmak (overriding), aşırı yüklemekten (overloading) ayırmak, `@Override` kullanmak.

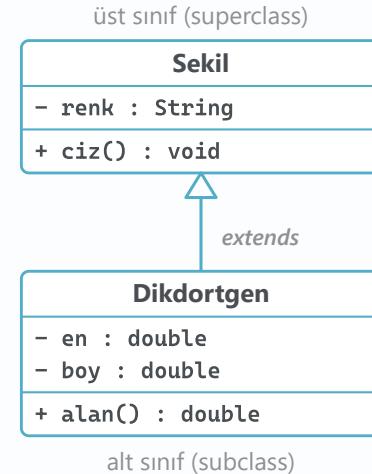
Object sınıfı

`toString`, `equals` ve `hashCode` metotlarını doğru yazmak; `==` ile `equals` farkını bilmek.

Kalıtım Nedir?

- **Kalıtım (Inheritance):** mevcut bir sınıfın alan ve metotlarını kalıtımla alıp yeni veya değiştirilmiş yeteneklerle zenginleştirerek **yeni bir sınıf** oluşturmaktır.
- Mevcut sınıf: **üst sınıf** (superclass).
- Yeni sınıf: **alt sınıf** (subclass).
- Alt sınıfa yeni alan ve metotlar eklenebilir; kalıtımla alınan alanlar **yeniden tanımlanmaz**.

Benzetme: “Her dikdörtgen bir şekildir.”
Dikdörtgen, şeklin her şeyine sahiptir; üstüne kendi en, boy ve alan() bilgisini ekler.



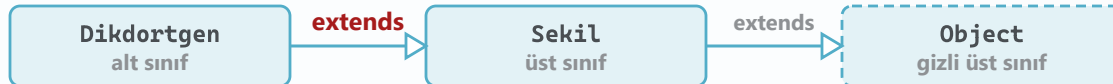
KALITIM KAVRAMI

extends ile Alt Sınıf

```
// Temel sınıf (üst sınıf)
public class Sekil {
    // Temel sınıfa ait nitelik ve metotlar
    // ...
}

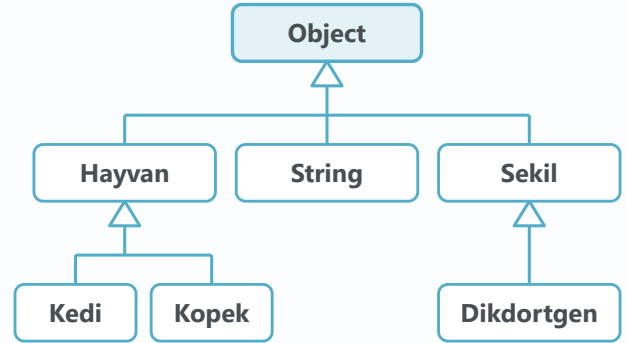
// Yeni sınıf (alt sınıf), Sekil'den türer
public class Dikdortgen extends Sekil {
    // Yeni sınıfa ait nitelik ve metotlar
    // ...
}
```

- Söz dizimi: `class AltSinif extends UstSinif` . `extends` : "genişletir".
- `extends` sonrasına **yalnızca bir** sınıf yazılabilir.
- Alt sınıf, üst sınıfın `public` ve `protected` üyelerini kendi üyesiymiş gibi kullanır.
- Üst sınıftaki aynı adlı alanı alt sınıfta yeniden tanımlamak onu **gizler** (hiding); iki ayrı değişken oluşur ve kafa karıştırır. Yapılmamalıdır.
- `extends` yazılmazsa sınıf, gizlice `Object` sınıfından türer.



Alt ve Üst Sınıf

- Alt sınıf, üst sınıfın davranışlarını sergiler; **ek olarak** kendine özgü davranışlar da sergiler.
- Kalıtım bu yüzden **özelleştirme** (specialization) olarak da adlandırılır: alt sınıf daha **özel** bir grubu temsil eder.
- Her kedi bir hayvandır, ama her hayvan kedi değildir.
- Java sınıf hiyerarşisi, `java.lang` paketindeki `Object` sınıfı ile başlar.
- Her sınıf, **doğrudan** veya **dolaylı** olarak `Object` sınıfından kalıtımla alır.



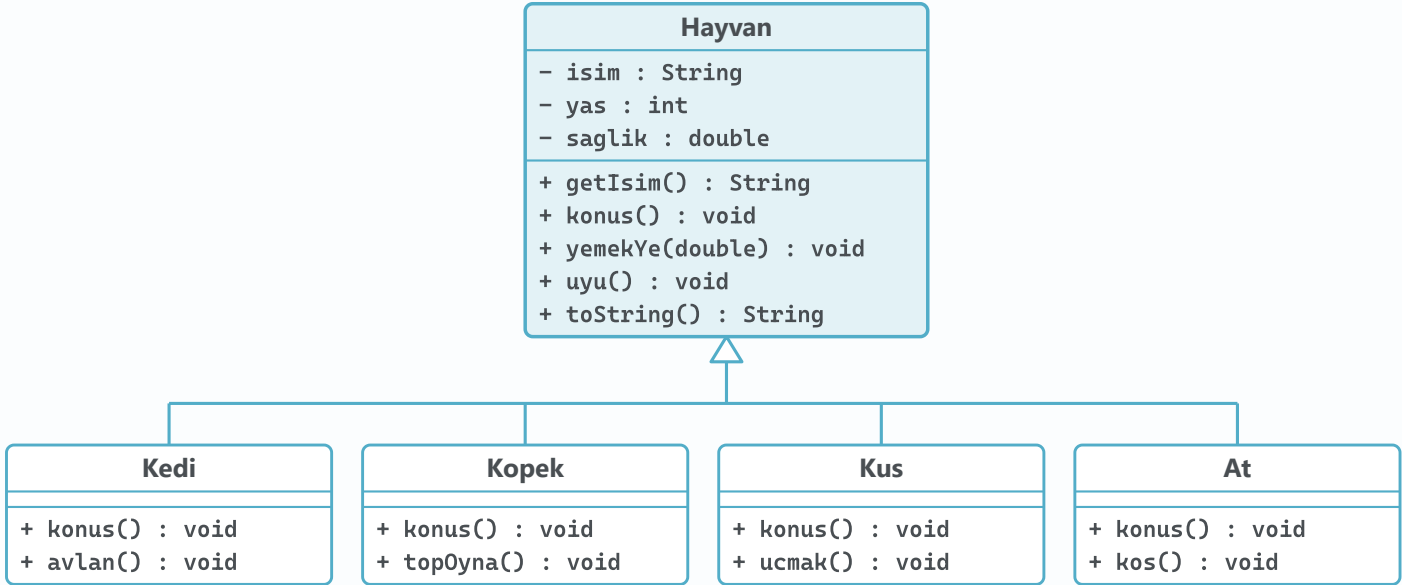
yukarı: genel · aşağı: özel

Neler Kalıtımla Alınır?

Üye	Alt sınıfa geçer mi?	Not
<code>public</code> / <code>protected</code> alan ve metotlar	Evet	Alt sınıfta doğrudan kullanılır.
Belirteçsiz (paket erişimli) üyeler	Yalnızca aynı paketteyse	Başka paketteki alt sınıf erişemez.
<code>private</code> alanlar	Nesnede yer alır , ama doğrudan erişilemez	Üst sınıfın <code>public</code> / <code>protected</code> metotlarıyla (getter, setter) kullanılır.
<code>private</code> metotlar	Hayır	Görünmez; geçersiz kılınamaz.
Yapıcılar (constructors)	Hayır	Kalıtımla alınmaz; alt sınıf yapıcısı <code>super(...)</code> ile çağırır.
<code>static</code> üyeler	Erişilebilir	Sınıfa aittir; geçersiz kılınmaz, yalnızca gizlenebilir.

Kısaca: alt sınıf nesnesinin içinde üst sınıfın bütün alanları vardır; hangisine **doğrudan** erişilebileceğini erişim belirteci belirler.

Hayvan Sınıf Hiyerarşisi



Ortak alan ve metotlar **bir kez**, **Hayvan** sınıfında yazılır. Her alt sınıf **konus()** metodunu kendine göre **geçersiz kılar** ve kendine özgü bir metot ekler. UML'de kalıtım, ucu üst sınıfa dönük **içi boş üçgen** okla gösterilir.

Hayvan ve Kopek Sınıfları

```
public class Hayvan {  
    private String isim;      // ortak nitelikler  
    private int yas;  
    private double saglik;  
  
    public Hayvan(String isim, int yas) {  
        this.isim = isim;  
        this.yas = yas;  
        this.saglik = 100.0;  
    }  
    public String getIsim() { return isim; } // private isim'e erişim yolu  
    public void konus() {  
        System.out.println("Bu hayvan konuşur.");  
    }  
    public void uyu() {  
        saglik += 5;  
        System.out.println(isim + " uyudu. Sağlık: " + saglik);  
    }  
}
```

Üst sınıf **Hayvan** : bütün hayvanlarda ortak olan alanlar ve davranışlar.

Hayvan ve Kopek Sınıfları

```
public class Kopek extends Hayvan {  
    public Kopek(String isim, int yas) {  
        super(isim, yas);    // üst sınıf yapıcısı alanları doldurur  
    }  
    @Override  
    public void konus() {    // miras alınan davranışı değiştirir  
        System.out.println(getIsim() + " havlar.");  
    }  
    public void topOyna() { // yalnızca Kopek'te olan davranış  
        System.out.println(getIsim() + " top oynuyor!");  
    }  
}
```

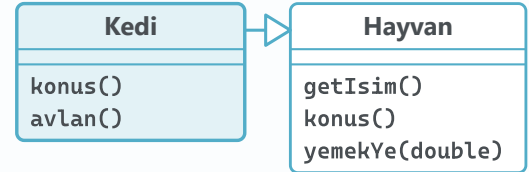
```
Kopek kopek = new Kopek("Karabaş", 4);  
kopek.konus();    // Kopek'in sürümü  
kopek.topOyna();  // Kopek'e özgü  
kopek.uyu();      // Hayvan'dan miras
```

Karabaş havlar.
Karabaş top oynuyor!
Karabaş uyudu. Sağlık: 105.0

Alt sınıf **Kopek** yalnızca **farkı** yazar; **uyu()** hiç yazılmadığı hâlde çalışır.

Hangi Metot Çalışır?

```
1 class Hayvan {
2     private String isim;
3     private double saglik = 100.0;
4     Hayvan(String isim) { this.isim = isim; }
5     String getIsim() { return isim; }
6     void konus() { System.out.println("Ses yok."); }
7     void yemekYe(double m) {
8         saglik += m;
9         System.out.println(isim + ": " + saglik);
10    }
11 }
12 class Kedi extends Hayvan {
13     Kedi(String isim) { super(isim); }
14     @Override void konus() {
15         System.out.println(getIsim() + " miyavlar.");
16     }
17     void avlan() {
18         System.out.println(getIsim() + " avlandı!");
19     }
20 }
21 ▶ Kedi kedi = new Kedi("Mavi"); // main
22 kedi.konus();
23 kedi.avlan();
24 kedi.yemekYe(5.0);
```



`new Kedi("Mavi")` : nesne oluştur. İçinde Hayvan'dan gelen `isim` ve `saglik` alanları da vardır.

Adım 1 / 8

ÇAĞRI YİĞİNİ

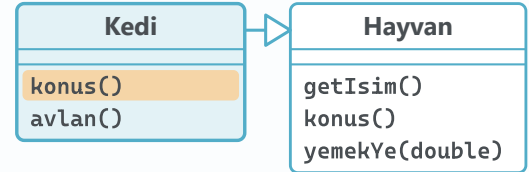
`main()`

ÇIKTI



Hangi Metot Çalışır?

```
1 class Hayvan {
2     private String isim;
3     private double saglik = 100.0;
4     Hayvan(String isim) { this.isim = isim; }
5     String getIsim() { return isim; }
6     void konus() { System.out.println("Ses yok."); }
7     void yemekYe(double m) {
8         saglik += m;
9         System.out.println(isim + ": " + saglik);
10    }
11 }
12 class Kedi extends Hayvan {
13     Kedi(String isim) { super(isim); }
14     @Override void konus() {
15         System.out.println(getIsim() + " miyavlar.");
16     }
17     void avlan() {
18         System.out.println(getIsim() + " avlandı!");
19     }
20 }
21 Kedi kedi = new Kedi("Mavi"); // main
22 ► kedi.konus();
23 kedi.avlan();
24 kedi.yemekYe(5.0);
```



✓ Kedi'de bulundu

`kedi.konus()` : arama nesnenin sınıfı

`Kedi` 'de başlar. `konus()` orada var:
geçersiz kılınmış sürüm seçilir.

Adım 2 / 8

ÇAĞRI YIĞINI

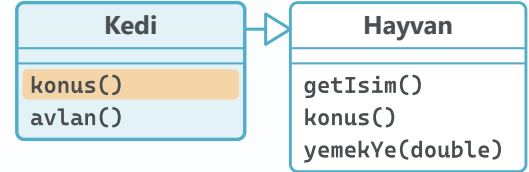
`main()`

ÇIKTI



Hangi Metot Çalışır?

```
1 class Hayvan {
2     private String isim;
3     private double saglik = 100.0;
4     Hayvan(String isim) { this.isim = isim; }
5     String getIsim() { return isim; }
6     void konus() { System.out.println("Ses yok."); }
7     void yemekYe(double m) {
8         saglik += m;
9         System.out.println(isim + ": " + saglik);
10    }
11 }
12 class Kedi extends Hayvan {
13     Kedi(String isim) { super(isim); }
14     @Override void konus() {
15         ▶ System.out.println(getIsim() + " miyavlar.");
16     }
17     void avlan() {
18         System.out.println(getIsim() + " avlandı!");
19     }
20 }
21 Kedi kedi = new Kedi("Mavi"); // main
22 kedi.konus();
23 kedi.avlan();
24 kedi.yemekYe(5.0);
```



✓ Kedi'de bulundu

getIsim() Hayvan'dan kalıtımla gelir;
private isim 'e onun aracılığıyla ulaşılır.

Adım 3 / 8

ÇAĞRI YİĞİNİ

Kedi.konus()

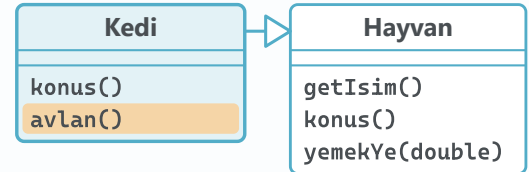
main()

ÇIKTI

Mavi miyavlar.

Hangi Metot Çalışır?

```
1 class Hayvan {
2     private String isim;
3     private double saglik = 100.0;
4     Hayvan(String isim) { this.isim = isim; }
5     String getIsim() { return isim; }
6     void konus() { System.out.println("Ses yok."); }
7     void yemekYe(double m) {
8         saglik += m;
9         System.out.println(isim + ": " + saglik);
10    }
11 }
12 class Kedi extends Hayvan {
13     Kedi(String isim) { super(isim); }
14     @Override void konus() {
15         System.out.println(getIsim() + " miyavlar.");
16     }
17     void avlan() {
18         System.out.println(getIsim() + " avlandı!");
19     }
20 }
21 Kedi kedi = new Kedi("Mavi");    // main
22 kedi.konus();
23 ► kedi.avlan();
24 kedi.yemekYe(5.0);
```



✓ Kedi'de bulundu

`kedi.avlan()` : Kedi'ye özgü metot;
Kedi'de bulunur.

Adım 4 / 8

ÇAĞRI YİĞİNİ

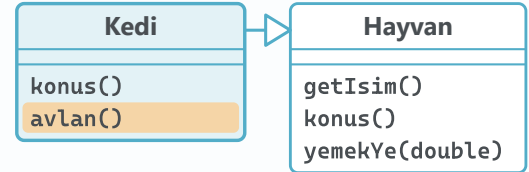
`main()`

ÇIKTI

Mavi miyavlar.

Hangi Metot Çalışır?

```
1 class Hayvan {
2     private String isim;
3     private double saglik = 100.0;
4     Hayvan(String isim) { this.isim = isim; }
5     String getIsim() { return isim; }
6     void konus() { System.out.println("Ses yok."); }
7     void yemekYe(double m) {
8         saglik += m;
9         System.out.println(isim + ": " + saglik);
10    }
11 }
12 class Kedi extends Hayvan {
13     Kedi(String isim) { super(isim); }
14     @Override void konus() {
15         System.out.println(getIsim() + " miyavlar.");
16     }
17     void avlan() {
18         ▶ System.out.println(getIsim() + " avlandı!");
19     }
20 }
21 Kedi kedi = new Kedi("Mavi"); // main
22 kedi.konus();
23 kedi.avlan();
24 kedi.yemekYe(5.0);
```



✓ Kedi'de bulundu

avlan() gövdesi çalışır ve ikinci satır yazılır.

Adım 5 / 8

ÇAĞRI YIĞINI

Kedi.avlan()

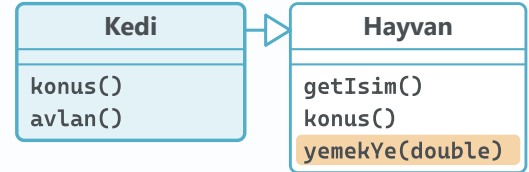
main()

ÇIKTI

```
Mavi miyavlar.
Mavi avlandı!
```

Hangi Metot Çalışır?

```
1 class Hayvan {
2     private String isim;
3     private double saglik = 100.0;
4     Hayvan(String isim) { this.isim = isim; }
5     String getIsim() { return isim; }
6     void konus() { System.out.println("Ses yok."); }
7     void yemekYe(double m) {
8         saglik += m;
9         System.out.println(isim + ": " + saglik);
10    }
11 }
12 class Kedi extends Hayvan {
13     Kedi(String isim) { super(isim); }
14     @Override void konus() {
15         System.out.println(getIsim() + " miyavlar.");
16     }
17     void avlan() {
18         System.out.println(getIsim() + " avlandı!");
19     }
20 }
21 Kedi kedi = new Kedi("Mavi");    // main
22 kedi.konus();
23 kedi.avlan();
24 ► kedi.yemekYe(5.0);
```



X Kedi'de yok → üste ✓ Hayvan'da bulundu

kedi.yemekYe(5.0) : Kedi'de yok → bir üst sınıfa, **Hayvan** 'a bakılır ve orada bulunur.

Adım 6 / 8

ÇAĞRI YİĞİNİ

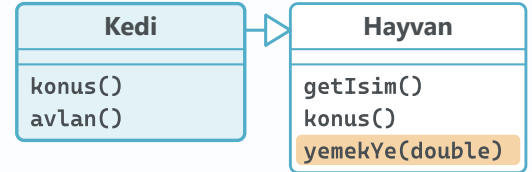
main()

ÇIKTI

```
Mavi miyavlar.
Mavi avlandı!
```

Hangi Metot Çalışır?

```
1 class Hayvan {
2     private String isim;
3     private double saglik = 100.0;
4     Hayvan(String isim) { this.isim = isim; }
5     String getIsim() { return isim; }
6     void konus() { System.out.println("Ses yok."); }
7     void yemekYe(double m) {
8         saglik += m;
9         System.out.println(isim + ": " + saglik);
10    }
11 }
12 class Kedi extends Hayvan {
13     Kedi(String isim) { super(isim); }
14     @Override void konus() {
15         System.out.println(getIsim() + " miyavlar.");
16     }
17     void avlan() {
18         System.out.println(getIsim() + " avlandı!");
19     }
20 }
21 Kedi kedi = new Kedi("Mavi"); // main
22 kedi.konus();
23 kedi.avlan();
24 kedi.yemekYe(5.0);
```



✓ Hayvan'da bulundu

Kalıtımla alınan metot, **aynı nesnenin** `saglik` alanını günceller: $100.0 + 5.0$.

Adım 7 / 8

ÇAĞRI YİĞİNİ

Hayvan.yemekYe(5.0)

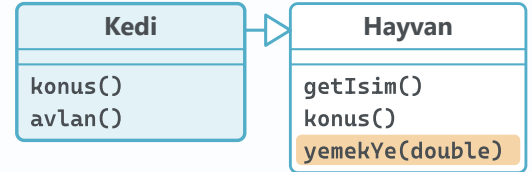
main()

ÇIKTI

```
Mavi miyavlar.
Mavi avlandı!
```

Hangi Metot Çalışır?

```
1 class Hayvan {
2     private String isim;
3     private double saglik = 100.0;
4     Hayvan(String isim) { this.isim = isim; }
5     String getIsim() { return isim; }
6     void konus() { System.out.println("Ses yok."); }
7     void yemekYe(double m) {
8         saglik += m;
9         System.out.println(isim + ": " + saglik);
10    }
11 }
12 class Kedi extends Hayvan {
13     Kedi(String isim) { super(isim); }
14     @Override void konus() {
15         System.out.println(getIsim() + " miyavlar.");
16     }
17     void avlan() {
18         System.out.println(getIsim() + " avlandı!");
19     }
20 }
21 Kedi kedi = new Kedi("Mavi");    // main
22 kedi.konus();
23 kedi.avlan();
24 kedi.yemekYe(5.0);
```



✓ Hayvan'da bulundu

Sonuç yazılır. Kural: metot **önce nesnenin kendi sınıfında**, yoksa yukarı doğru üst sınıflarda aranır.

Adım 8 / 8

ÇAĞRI YIĞINI

Hayvan.yemekYe(5.0)

main()

ÇIKTI

```
Mavi miyavlar.
Mavi avlandı!
Mavi: 105.0
```

Kalıtımın Avantajları

Yeniden kullanım

Üst sınıfın metot ve alanları alt sınıfta yeniden yazılmadan kullanılır.

Genişletme

Mevcut sınıf, kodu değiştirilmeden yeni özelliklerle genişletilir.

Değiştirme

Üst sınıfın metotları alt sınıfta **geçersiz kılınarak** (overriding) değiştirilebilir.

HesapMakinesi

- hafiza : double
+ topla(int, int) : double
+ carp(int, int) : double
+ hafizayiGoruntule() : double



GelismisHesapMakinesi

+ us(double, int) : double
+ faktoriyel(int) : long

```
public class GelismisHesapMakinesi
    extends HesapMakinesi {
    // Yeni yetenek: üs alma
    public double us(double taban, int us) {
        double sonuc = 1.0;
        for (int i = 0; i < us; i++) {
            sonuc *= taban;
        }
        return sonuc;
    }
    // faktoriyel(int) benzer biçimde eklenir
}

// main içinde
GelismisHesapMakinesi h = new GelismisHesapMakinesi();
System.out.println(h.topla(5, 3));           // 8.0 (miras)
System.out.println(h.hafizayiGoruntule());   // 8.0 (miras)
System.out.println(h.us(2, 5));              // 32.0 (yeni)
```

super Anahtar Kelimesi

- `super(...)` : alt sınıfın yapıcısı içinde **üst sınıfın yapıcısını** çağırır; üst sınıfın alanlarına ilk değerleri o atar.
- `super(...)` yapıcının **ilk satırı** olmalıdır.
- Yazılmazsa derleyici gizlice `super();` (parametresiz) ekler.
- `super.metot()` : üst sınıfın metoduna nokta (.) ile erişir; geçersiz kılınmış bir metodun içinden üst sınıfın sürümünü çağırmak için kullanılır.
- İlke: **her sınıf yalnızca kendi verilerini yönetmelidir**. Kedi, Hayvan'ın `isim` alanına kendisi atama yapmaz; işi `super(isim)` 'e bırakır.

```
class Kedi extends Hayvan {  
    private boolean evcil;  
  
    Kedi(String isim, boolean evcil) {  
        super(isim);           // Hayvan(String)  
        this.evcil = evcil;    // kendi verisi  
    }  
  
    @Override  
    public String toString() {  
        return super.toString() // Hayvan'ınki  
            + ", Evcil: " + evcil;  
    }  
}  
  
// main içinde  
System.out.println(new Kedi("Mavi", true));
```

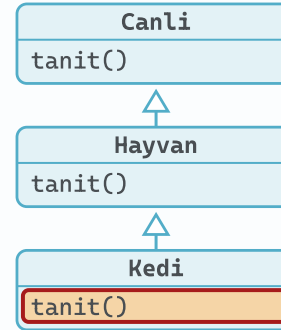
ÇIKTI

```
İsim: Mavi, Sağlık: 100.0, Evcil: true
```

Hayvan'ın `toString()` metodu isim ve sağlık bilgisini döndürür; Kedi yalnızca kendi alanını ekler.

super.tanıt() Zinciri: Kod İzleme

```
1 class Canli {  
2     void tanıt() { System.out.println("Canlıyım"); }  
3 }  
4 class Hayvan extends Canli {  
5     void tanıt() {  
6         super.tanıt();  
7         System.out.println("Hayvanım");  
8     }  
9 }  
10 class Kedi extends Hayvan {  
11     void tanıt() {  
12         super.tanıt();  
13         System.out.println("Kediyim");  
14     }  
15 }  
▶ new Kedi().tanıt(); // main
```



Çağrı yığını

Kedi.tanıt()

main()

nesne: Kedi #1

arama nesnenin sınıfından başlar: Kedi.tanıt()

Nesne bir **Kedi** ; arama en alttan başlar ve **Kedi.tanıt()** bulunur.

Adım 1 / 7

ÇIKTI



super.tanıt() Zinciri: Kod İzleme

```
1 class Canli {  
2     void tanıt() { System.out.println("Canlıyım"); }  
3 }  
4 class Hayvan extends Canli {  
5     void tanıt() {  
6         super.tanıt();  
7         System.out.println("Hayvanım");  
8     }  
9 }  
10 class Kedi extends Hayvan {  
11     void tanıt() {  
12         ▶ super.tanıt();  
13         System.out.println("Kediyim");  
14     }  
15 }  
16 new Kedi().tanıt(); // main
```



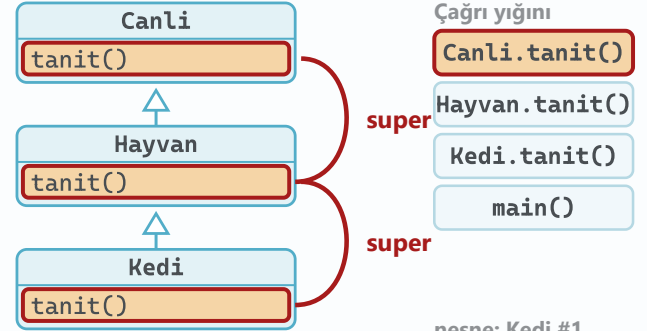
Adım 2 / 7

ÇIKTI



super.tanıt() Zinciri: Kod İzleme

```
1 class Canli {  
2     void tanıt() { System.out.println("Canlıyım"); }  
3 }  
4 class Hayvan extends Canli {  
5     void tanıt() {  
6         super.tanıt();  
7         System.out.println("Hayvanım");  
8     }  
9 }  
10 class Kedi extends Hayvan {  
11     void tanıt() {  
12         super.tanıt();  
13         System.out.println("Kediyim");  
14     }  
15 }  
16 new Kedi().tanıt(); // main
```



Nesne hâlâ bir **Kedi** olsa da **Hayvan** içindeki **super** sabit olarak **Canli** 'yi gösterir: dinamik bağlama **yok**.

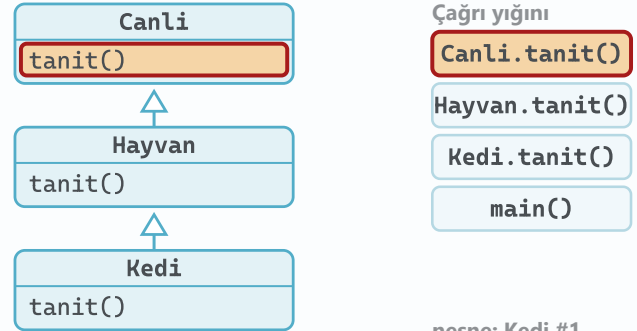
Adım 3 / 7

ÇIKTI



super.tanıt() Zinciri: Kod İzleme

```
1 class Canli {  
2     void tanıt() { System.out.println("Canlıyım"); }  
3 }  
4 class Hayvan extends Canli {  
5     void tanıt() {  
6         super.tanıt();  
7         System.out.println("Hayvanım");  
8     }  
9 }  
10 class Kedi extends Hayvan {  
11     void tanıt() {  
12         super.tanıt();  
13         System.out.println("Kediyim");  
14     }  
15 }  
16 new Kedi().tanıt(); // main
```



nesne: Kedi #1

Canlıyım yazılır, `Canli.tanıt()` döner

En üstteki sürüm çalışır ve ilk satırı yazar; sonra döner.

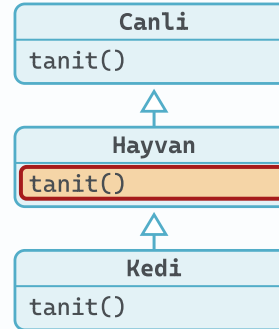
Adım 4 / 7

ÇIKTI

Canlıyım

super.tanıt() Zinciri: Kod İzleme

```
1 class Canli {
2     void tanıt() { System.out.println("Canlıyım"); }
3 }
4 class Hayvan extends Canli {
5     void tanıt() {
6         super.tanıt();
7         System.out.println("Hayvanım");
8     }
9 }
10 class Kedi extends Hayvan {
11     void tanıt() {
12         super.tanıt();
13         System.out.println("Kediyim");
14     }
15 }
16 new Kedi().tanıt(); // main
```



Çağrı yığını

Hayvan.tanıt()

Kedi.tanıt()

main()

nesne: Kedi #1

Hayvan.tanıt() kaldığı yerden sürer:
Hayvanım.

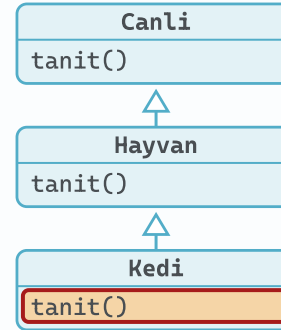
Adım 5 / 7

ÇIKTI

Canlıyım
Hayvanım

super.tanıt() Zinciri: Kod İzleme

```
1 class Canli {  
2     void tanıt() { System.out.println("Canlıyım"); }  
3 }  
4 class Hayvan extends Canli {  
5     void tanıt() {  
6         super.tanıt();  
7         System.out.println("Hayvanım");  
8     }  
9 }  
10 class Kedi extends Hayvan {  
11     void tanıt() {  
12         super.tanıt();  
13         System.out.println("Kediyim");  
14     }  
15 }  
16 new Kedi().tanıt(); // main
```



Çağrı yığını

Kedi.tanıt()

main()

nesne: Kedi #1

Son olarak **Kedi.tanıt()** kendi satırını yazar.

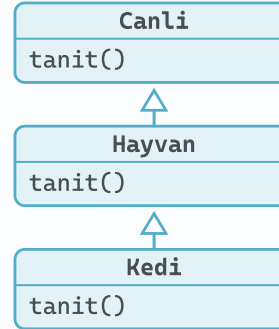
Adım 6 / 7

ÇIKTI

```
Canlıyım  
Hayvanım  
Kediyim
```

super.tanıt() Zinciri: Kod İzleme

```
1 class Canli {  
2     void tanıt() { System.out.println("Canlıyım"); }  
3 }  
4 class Hayvan extends Canli {  
5     void tanıt() {  
6         super.tanıt();  
7         System.out.println("Hayvanım");  
8     }  
9 }  
10 class Kedi extends Hayvan {  
11     void tanıt() {  
12         super.tanıt();  
13         System.out.println("Kediyim");  
14     }  
15 }  
▶ new Kedi().tanıt(); // main
```



Çağrı yığını

main()

nesne: Kedi #1

çağrı aşağıdan yukarı, çıktı yukarıdan aşağı

Çağrılar Kedi → Hayvan → Canli yönünde gitti, çıktılar ters sırada geldi: her sınıf önce üst sınıfın işini yaptırıp sonra kendi farkını ekledi.

Adım 7 / 7

ÇIKTI

```
Canlıyım  
Hayvanım  
Kediyim
```

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {  
2     private String isim;  
3     private double saglik;  
4     Hayvan(String isim) {  
5         super(); // Object() çağrılır  
6         this.isim = isim;  
7         saglik = 100.0;  
8         System.out.println("Hayvan yapıcısı");  
9     }  
10 }  
11 class Kedi extends Hayvan {  
12     private boolean evcil;  
13     Kedi(String isim, boolean evcil) {  
14         super(isim);  
15         this.evcil = evcil;  
16         System.out.println("Kedi yapıcısı");  
17     }  
18 }  
19 // main içinde  
▶ Kedi kedi = new Kedi("Mavi", true);  
21 System.out.println("Nesne hazır");
```

kedi

-

Kedi nesnesi

Hayvan'dan gelen

isim = null

saglik = 0.0

Kedi'nin kendi

evcil = false

`new` : önce bellekte nesneye yer ayrılır; **tüm** alanlar (Hayvan'ınkiler dahil) varsayılan değerleri alır.

Adım 1 / 11

ÇAĞRI YİĞİNİ

main()

ÇIKTI

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {
2     private String isim;
3     private double saglik;
4     Hayvan(String isim) {
5         super(); // Object() çağrılır
6         this.isim = isim;
7         saglik = 100.0;
8         System.out.println("Hayvan yapıcısı");
9     }
10 }
11 class Kedi extends Hayvan {
12     private boolean evcil;
13     ► Kedi(String isim, boolean evcil) {
14         super(isim);
15         this.evcil = evcil;
16         System.out.println("Kedi yapıcısı");
17     }
18 }
19 // main içinde
20 Kedi kedi = new Kedi("Mavi", true);
21 System.out.println("Nesne hazır");
```

kedi

-

Kedi nesnesi

Hayvan'dan gelen

isim = null

saglik = 0.0

Kedi'nin kendi

evcil = false

Kedi yapıcısı çağrılır. Kendi işine başlamadan önce üst sınıf kısmı hazırlanmalıdır.

Adım 2 / 11

ÇAĞRI YİĞİNİ

Kedi("Mavi", true)

main()

ÇIKTI

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {
2     private String isim;
3     private double saglik;
4     Hayvan(String isim) {
5         super(); // Object() çağrılır
6         this.isim = isim;
7         saglik = 100.0;
8         System.out.println("Hayvan yapıcısı");
9     }
10 }
11 class Kedi extends Hayvan {
12     private boolean evcil;
13     Kedi(String isim, boolean evcil) {
14         ▶ super(isim);
15         this.evcil = evcil;
16         System.out.println("Kedi yapıcısı");
17     }
18 }
19 // main içinde
20 Kedi kedi = new Kedi("Mavi", true);
21 System.out.println("Nesne hazır");
```

kedi

–

Kedi nesnesi

Hayvan'dan gelen

isim = null

saglik = 0.0

Kedi'nin kendi

evcil = false

`super(isim)` : Hayvan yapıcısı çağrılır; Kedi yapıcısı bekler.

Adım 3 / 11

ÇAĞRI YIĞINI

`Kedi("Mavi", true)`

`main()`

ÇIKTI

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {  
2     private String isim;  
3     private double saglik;  
4     Hayvan(String isim) {  
5         ▶ super(); // Object() çağrılır  
6         this.isim = isim;  
7         saglik = 100.0;  
8         System.out.println("Hayvan yapıcısı");  
9     }  
10 }  
11 class Kedi extends Hayvan {  
12     private boolean evcil;  
13     Kedi(String isim, boolean evcil) {  
14         super(isim);  
15         this.evcil = evcil;  
16         System.out.println("Kedi yapıcısı");  
17     }  
18 }  
19 // main içinde  
20 Kedi kedi = new Kedi("Mavi", true);  
21 System.out.println("Nesne hazır");
```

kedi

-

Kedi nesnesi

Hayvan'dan gelen

isim = null

saglik = 0.0

Kedi'nin kendi

evcil = false

Hayvan da önce kendi üst sınıfını çağırır: `Object()`. Her zincir `Object`'te biter.

Adım 4 / 11

ÇAĞRI YIĞINI

`Object()``Hayvan("Mavi")``Kedi("Mavi", true)``main()`

ÇIKTI

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {
2     private String isim;
3     private double saglik;
4     Hayvan(String isim) {
5         super(); // Object() çağrılır
6         this.isim = isim;
7         saglik = 100.0;
8         System.out.println("Hayvan yapıcısı");
9     }
10 }
11 class Kedi extends Hayvan {
12     private boolean evcil;
13     Kedi(String isim, boolean evcil) {
14         super(isim);
15         this.evcil = evcil;
16         System.out.println("Kedi yapıcısı");
17     }
18 }
19 // main içinde
20 Kedi kedi = new Kedi("Mavi", true);
21 System.out.println("Nesne hazır");
```

kedi

-

Kedi nesnesi

Hayvan'dan gelen

isim = "Mavi"

saglik = 0.0

Kedi'nin kendi

evcil = false

Object() döndü. Hayvan kendi alanını doldurur: `isim = "Mavi"`.

Adım 5 / 11

ÇAĞRI YIĞINI

Hayvan("Mavi")

Kedi("Mavi", true)

main()

ÇIKTI

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {
2     private String isim;
3     private double saglik;
4     Hayvan(String isim) {
5         super(); // Object() çağrılır
6         this.isim = isim;
7         saglik = 100.0;
8         System.out.println("Hayvan yapıcısı");
9     }
10 }
11 class Kedi extends Hayvan {
12     private boolean evcil;
13     Kedi(String isim, boolean evcil) {
14         super(isim);
15         this.evcil = evcil;
16         System.out.println("Kedi yapıcısı");
17     }
18 }
19 // main içinde
20 Kedi kedi = new Kedi("Mavi", true);
21 System.out.println("Nesne hazır");
```

kedi

-

saglik = 100.0 .

Kedi nesnesi

Hayvan'dan gelen
isim = "Mavi"
saglik = 100.0

Kedi'nin kendi
evcil = false

Adım 6 / 11

ÇAĞRI YIĞINI

Hayvan("Mavi")

Kedi("Mavi", true)

main()

ÇIKTI

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {
2     private String isim;
3     private double saglik;
4     Hayvan(String isim) {
5         super(); // Object() çağrılır
6         this.isim = isim;
7         saglik = 100.0;
8         ▶ System.out.println("Hayvan yapıcısı");
9     }
10 }
11 class Kedi extends Hayvan {
12     private boolean evcil;
13     Kedi(String isim, boolean evcil) {
14         super(isim);
15         this.evcil = evcil;
16         System.out.println("Kedi yapıcısı");
17     }
18 }
19 // main içinde
20 Kedi kedi = new Kedi("Mavi", true);
21 System.out.println("Nesne hazır");
```

kedi

-

Kedi nesnesi

Hayvan'dan gelen
isim = "Mavi"
saglik = 100.0

Kedi'nin kendi
evcil = false

Hayvan yapıcısı mesajını yazar ve döner: **önce üst sınıf** yazdı.

Adım 7 / 11

ÇAĞRI YIĞINI

Hayvan("Mavi")

Kedi("Mavi", true)

main()

ÇIKTI

Hayvan yapıcısı

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {
2     private String isim;
3     private double saglik;
4     Hayvan(String isim) {
5         super(); // Object() çağrılır
6         this.isim = isim;
7         saglik = 100.0;
8         System.out.println("Hayvan yapıcısı");
9     }
10 }
11 class Kedi extends Hayvan {
12     private boolean evcil;
13     Kedi(String isim, boolean evcil) {
14         super(isim);
15         ▶ this.evcil = evcil;
16         System.out.println("Kedi yapıcısı");
17     }
18 }
19 // main içinde
20 Kedi kedi = new Kedi("Mavi", true);
21 System.out.println("Nesne hazır");
```

kedi

-

Kedi nesnesi

Hayvan'dan gelen
isim = "Mavi"
saglik = 100.0

Kedi'nin kendi
evcil = true

Kedi yapıcısı kaldığı yerden devam eder: `evcil = true` .

Adım 8 / 11

ÇAĞRI YIĞINI

`Kedi("Mavi", true)`

`main()`

ÇIKTI

Hayvan yapıcısı

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {  
2     private String isim;  
3     private double saglik;  
4     Hayvan(String isim) {  
5         super(); // Object() çağrılır  
6         this.isim = isim;  
7         saglik = 100.0;  
8         System.out.println("Hayvan yapıcısı");  
9     }  
10 }  
11 class Kedi extends Hayvan {  
12     private boolean evcil;  
13     Kedi(String isim, boolean evcil) {  
14         super(isim);  
15         this.evcil = evcil;  
16         System.out.println("Kedi yapıcısı");  
17     }  
18 }  
19 // main içinde  
20 Kedi kedi = new Kedi("Mavi", true);  
21 System.out.println("Nesne hazır");
```

kedi

-

Kedi nesnesi

Hayvan'dan gelen
isim = "Mavi"
saglik = 100.0

Kedi'nin kendi
evcil = true

Kedi yapıcısı mesajını yazar.

Adım 9 / 11

ÇAĞRI YIĞINI

Kedi("Mavi", true)

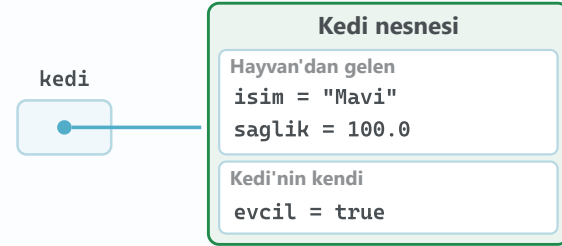
main()

ÇIKTI

Hayvan yapıcısı
Kedi yapıcısı

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {  
2     private String isim;  
3     private double saglik;  
4     Hayvan(String isim) {  
5         super(); // Object() çağrılır  
6         this.isim = isim;  
7         saglik = 100.0;  
8         System.out.println("Hayvan yapıcısı");  
9     }  
10 }  
11 class Kedi extends Hayvan {  
12     private boolean evcil;  
13     Kedi(String isim, boolean evcil) {  
14         super(isim);  
15         this.evcil = evcil;  
16         System.out.println("Kedi yapıcısı");  
17     }  
18 }  
19 // main içinde  
20 ► Kedi kedi = new Kedi("Mavi", true);  
21 System.out.println("Nesne hazır");
```



Yapıcı bitti; nesnenin adresi `kedi` değişkenine atanır.

Adım 10 / 11

ÇAĞRI YİĞİNİ

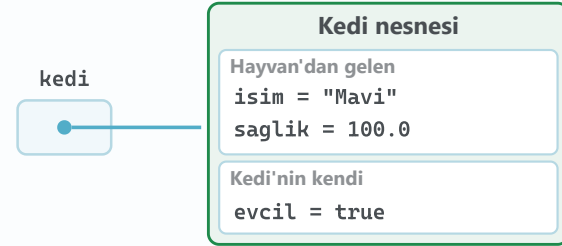
`main()`

ÇIKTI

```
Hayvan yapıcısı  
Kedi yapıcısı
```

Yapıcı Zinciri: `new Kedi("Mavi", true)`

```
1 class Hayvan {
2     private String isim;
3     private double saglik;
4     Hayvan(String isim) {
5         super(); // Object() çağrılır
6         this.isim = isim;
7         saglik = 100.0;
8         System.out.println("Hayvan yapıcısı");
9     }
10 }
11 class Kedi extends Hayvan {
12     private boolean evcil;
13     Kedi(String isim, boolean evcil) {
14         super(isim);
15         this.evcil = evcil;
16         System.out.println("Kedi yapıcısı");
17     }
18 }
19 // main içinde
20 Kedi kedi = new Kedi("Mavi", true);
▶ System.out.println("Nesne hazır");
```



Yapıcılar **yukarıdan aşağıya** tamamlanır: Object → Hayvan → Kedi.

Adım 11 / 11

ÇAĞRI YİĞİNİ

`main()`

ÇIKTI

```
Hayvan yapıcısı
Kedi yapıcısı
Nesne hazır
```

Gizli super() ve Hatalar

1) ÜST SINIFTA PARAMETRESİZ YAPICI YOK

```
class Hayvan {  
    private String isim;  
    Hayvan(String isim) { this.isim = isim; }  
}  
class Kedi extends Hayvan {  
    Kedi(String isim) {  
        // derleyici buraya super(); ekler  
        System.out.println("Kedi");  
    }  
}
```

error: constructor Hayvan in class Hayvan
cannot be applied to given types;
required: String
found: no arguments

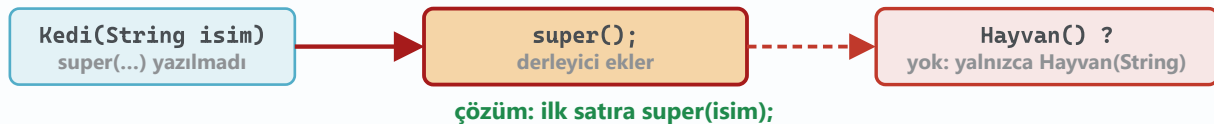
2) SUPER(...) İLK SATIRDA DEĞİL

```
Kedi(String isim) {  
    System.out.println("Kedi");  
    super(isim);  
}
```

error: call to super must be first
statement in constructor

Çözüm: alt sınıf yapıcısının ilk satırında uygun üst sınıf yapıcısını açıkça çağırın: `super(isim);`

Hiç yapıcı yazılmamış sınıfa derleyici parametresiz bir varsayılan yapıcı (default constructor) ekler; bir yapıcı yazıldığı anda bu eklenmez.



Metodu Geçersiz Kılma

Geçersiz kılma (overriding): alt sınıfın, üst sınıftan kalıtımla aldığı bir metodu **aynı imzayla** yeniden yazarak kendi davranışını vermesidir.

- Ad ve parametre listesi **aynı** olmalı.
- Dönüş tipi aynı (veya onun alt tipi) olmalı.
- Erişim **daraltılamaz**: `public` metod `public` kalmalı.
- `final`, `static` ve `private` metodlar geçersiz kılınamaz.

```
class Sekil {  
    public void ciz() {  
        System.out.println("Şekil çiziliyor.");  
    }  
}  
  
class Dikdortgen extends Sekil {  
    @Override  
    public void ciz() {  
        System.out.println("Dikdörtgen çiziliyor.");  
    }  
}  
  
// main içinde  
new Sekil().ciz();  
new Dikdortgen().ciz();
```

Şekil çiziliyor.
Dikdörtgen çiziliyor.

```
// alt sınıfta: void ciz()  
error: ciz() in Dikdortgen cannot override  
      ciz() in Sekil  
      attempting to assign weaker access  
      privileges; was public
```

```
// alt sınıfta: public int ciz()  
error: ciz() in Dikdortgen cannot override  
      ciz() in Sekil  
      return type int is not compatible  
      with void
```

Geçersiz Kılma ve Aşırı Yükleme

	Geçersiz kılma (overriding)	Aşırı yükleme (overloading)
Nerede?	Alt sınıfta, kalıtımla alınan metot için	Aynı sınıfta (veya alt sınıfta)
Ad	Aynı	Aynı
Parametreler	Aynı	Farklı (sayı ya da tip)
Dönüş tipi	Aynı veya alt tipi	Serbest
Amaç	Kalıtımla alınan davranışı değiştirmek	Aynı işi farklı girdilerle yapmak
Seçim zamanı	Çalışma zamanı (nesnenin gerçek tipi)	Derleme zamanı (argüman tipleri)

```
class Dikdortgen extends Sekil {  
    @Override  
    public void ciz() { ... }           // geçersiz kılma  
    public void ciz(String renk) { ... } // aşırı yükleme  
}
```

d.ciz("Mavi") ÇIKTISI

Mavi dikdörtgen çiziliyor.

Her iki kavram da bir sonraki bölümün konusu olan **çok biçimliliğin** (polymorphism) temelidir.

@Override Etiketi

- `@Override` bir **etikettir** (annotation): metodun, üst sınıftan kalıtımla alınan bir metodu geçersiz kıldığını bildirir.
- Derleyiciye bunu **kontrol etmesini** söyler: üst sınıfta böyle bir metot yoksa derleme hatası verir.
- Kodu daha okunabilir yapar; diğer geliştiricilere metodun üst sınıftan geldiğini gösterir.
- Zorunlu değildir, ama **her zaman yazılması** önerilir: yazım hatalarını hemen yakalar.

```
public class Dikdortgen extends Sekil {  
    // miras alınan metodu geçersiz kıl  
    @Override  
    public void ciz() {  
        System.out.println("Dikdörtgen çiziliyor.");  
    }  
}
```

YAZIM HATASI: `Ciz` (BÜYÜK C)

```
@Override  
public void Ciz() { ... }
```

error: method does not override or implement
a method from a supertype

`@Override` olmasaydı `Ciz()` sessizce yeni bir metot olurdu ve `ciz()` çağrıları hâlâ Sekil'in sürümünü çalıştırırdı.

ERİŞİM VE SINIRLAMA

protected Erişim Seviyesi

Belirteç	Aynı sınıf	Aynı paket	Alt sınıf (başka paket)	Her yer
<code>private</code>	✓	✗	✗	✗
(belirteç yok: paket)	✓	✓	✗	✗
<code>protected</code>	✓	✓	✓	✗
<code>public</code>	✓	✓	✓	✓

- `public` : sınıfın üyelerine her yerden erişilebilir. `private` : yalnızca o sınıfın içinden.
- `protected` : ikisinin arasında bir düzeydir. Sınıfın kendisi, **alt sınıfları** ve **aynı paketteki** diğer sınıflar erişebilir.
- Bir sınıfın `private` alanlarına alt sınıf, yalnızca üst sınıftan kalıtımla alınan `public` veya `protected` metotlar aracılığıyla erişebilir.

İyi uygulama: alanlar yine `private` kalsın; alt sınıfların ihtiyaç duyduğu işlemler `protected` metotlar olarak açılsın.

ERİŞİM VE SINIRLAMA

protected: Kod Örneği

```
package canli;

public class Hayvan {
    protected String isim; // alt sınıflar
    private double saglik; // yalnızca Hayvan

    public Hayvan(String isim) {
        this.isim = isim;
        saglik = 100.0;
    }
    public double getSaglik() { return saglik; }
    protected void saglikEkle(double miktar) {
        saglik += miktar;
    }
}
```

```
package evcil; // başka paket
import canli.Hayvan;

public class Kedi extends Hayvan {
    public Kedi(String isim) { super(isim); }
    public void uyu() {
        // protected alan: izin var
        System.out.println(isim + " uyuyor.");
        // saglik += 5; // private: hata
        saglikEkle(5); // protected metot
        System.out.println("Sağlık: "
                           + getSaglik());
    }
}
```

```
new Kedi("Mavi").uyu();
```

 ÇIKTISI

```
Mavi uyuyor.
Sağlık: 105.0
```

YORUM SATIRI AÇILIRSA

```
error: saglik has private access in Hayvan
```

ERİŞİM VE SINIRLAMA

final ile Kalıtımı Sınırlamak

- `final` anahtar kelimesi kalıtımı **sınırlamak** için kullanılabilir.
- Bir **metot** `final` tanımlanırsa hiçbir alt sınıfta geçersiz kırılamaz (override).
- Bir **sınıf** `final` tanımlanırsa ondan hiçbir alt sınıf türetilemez. Örnek: `String`, `Math`, `Integer`.

```
class Hayvan {  
    public final void nefesAl() { ... }  
}  
class Kedi extends Hayvan {  
    public void nefesAl() { ... }  
}
```

error: nefesAl() in Kedi cannot override
nefesAl() in Hayvan
overridden method is final

```
final class Sabit { ... }  
class Deneme extends Sabit { }
```

error: cannot inherit from final Sabit

Neden? Güvenlik ve tutarlılık: `String` değiştirilemez olmalıdır; bir alt sınıf bu sözü bozamasın.

Değişkenlerde `final` başka bir anlam taşır: değer bir kez atanır (sabit).



Nesnenin Görünümünü Kısıtlama

- Bir nesne, **üst sınıf tipinde** bir değişkene atanabilir (yukarı dönüşüm, upcasting): her Lastik bir Parça'dır.
- Böylece dışarıya yalnızca **üst sınıfa ait** metotlar açılır.
- Nesnenin gerçek sınıfına bakılmaksızın, değişkenin tipi neyi **çağırabileceğimizi** belirler.

Parça p



Lastik nesnesi

Parça kısmı · p ile görünür
marka = "Petlas"
seriNo = "XYZ-123"
seriNumarasi()

Lastik kısmı · p ile görünmez
ebat = "225/45R17"
kislik = true
kislikMi()

```
Parça p = new Lastik("Petlas", "XYZ-123", "225/45R17", true);  
System.out.println(p.seriNumarasi()); // izin verilir → XYZ-123  
// p.kislikMi()                       // izin verilmez
```

```
error: cannot find symbol  
symbol:   method kislikMi()  
location: variable p of type Parça
```

Lastik'e özgü metoda ulaşmak için önce tip kontrolü, sonra aşağı dönüşüm (downcasting) gerekir: `instanceof` slaytına bakın.

SINIFLAR ARASI İLİŞKİLER

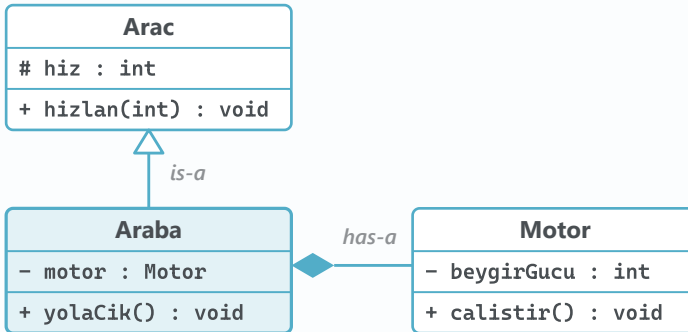
is-a ve has-a İlişkileri

Is-a (kalıtım, inheritance)

- Bir sınıfın başka bir sınıftan türetilmesidir; türetilen sınıf üst sınıfın **bir türüdür**.
- Alt sınıf nesnesi, üst sınıfın nesnesi gibi de kullanılabilir.

Has-a (oluşum, composition)

- Bir sınıfın başka bir sınıf tipinde alan içermesidir.
- Nesne, içinde diğer nesnelere **referanslar** tutar.



```
// Is-a ilişkisi (kalıtım)
public class Araba extends Arac {
    // Has-a ilişkisi (oluşum)
    private Motor motor;

    public Araba(int beygirGucu) {
        motor = new Motor(beygirGucu);
    }

    public void yolaCik() {
        motor.calistir(); // Motor'a devret
        hizlan(50);       // Arac'tan miras
        System.out.println("Hız: " + hiz
                           + " km/sa");
    }
}

// Motor: beygirGucu alanı ve calistir():
// "Motor çalıştı: " + beygirGucu + " BG"
```

```
new Araba(110).yolaCik();
```

```
Motor çalıştı: 110 BG
Hız: 50 km/sa
```

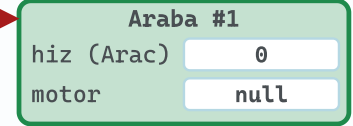
is-a ve has-a: Kod İzleme

```
1 class Motor {  
2     private int bg;  
3     Motor(int bg) { this.bg = bg; }  
4     void calistir() {  
5         System.out.println("Motor: " + bg + " BG");  
6     }  
7 }  
8 class Araba extends Arac {           // is-a  
9     private Motor motor;             // has-a  
10    Araba(int bg) { motor = new Motor(bg); }  
11    void yolaCik() {  
12        motor.calistir(); hizlan(50);  
13        System.out.println("Hız: " + hiz);  
14    }  
15 }  
▶ new Araba(90).yolaCik(); // main
```

Yığın



Heap



main()

Arac kısmı (hiz) + Araba kısmı (motor)

new Araba(90) : tek nesne, içinde üst sınıf
Arac 'tan gelen hiz ve kendi motor alanı
var (is-a).

Adım 1 / 6

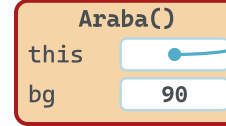
ÇIKTI



is-a ve has-a: Kod İzleme

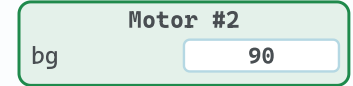
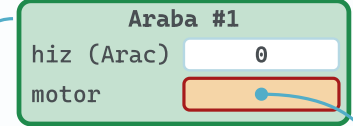
```
1 class Motor {  
2     private int bg;  
3     ▶ Motor(int bg) { this.bg = bg; }  
4     void calistir() {  
5         System.out.println("Motor: " + bg + " BG");  
6     }  
7 }  
8 class Araba extends Arac {           // is-a  
9     private Motor motor;             // has-a  
10    ▶ Araba(int bg) { motor = new Motor(bg); }  
11    void yolaCik() {  
12        motor.calistir(); hizlan(50);  
13        System.out.println("Hız: " + hiz);  
14    }  
15 }  
16 new Araba(90).yolaCik(); // main
```

Yığın



main()

Heap



Yapıcı ikinci bir nesne oluşturur: araba bir motor **değildir**, motora **sahiptir (has-a)**. Bağlantı bir referanstır.

Adım 2 / 6

ÇIKTI



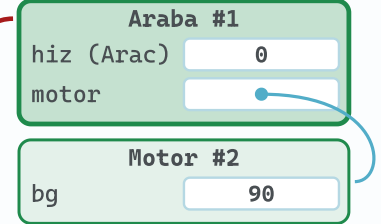
is-a ve has-a: Kod İzleme

```
1 class Motor {  
2     private int bg;  
3     Motor(int bg) { this.bg = bg; }  
4     void calistir() {  
5         System.out.println("Motor: " + bg + " BG");  
6     }  
7 }  
8 class Araba extends Arac {           // is-a  
9     private Motor motor;             // has-a  
10    Araba(int bg) { motor = new Motor(bg); }  
11    ▶ void yolaCik() {  
12        motor.calistir(); hizlan(50);  
13        System.out.println("Hız: " + hiz);  
14    }  
15 }  
▶ new Araba(90).yolaCik(); // main
```

Yığın



Heap



`yolaCik()` Araba nesnesi üzerinde çalışır.

Adım 3 / 6

ÇIKTI

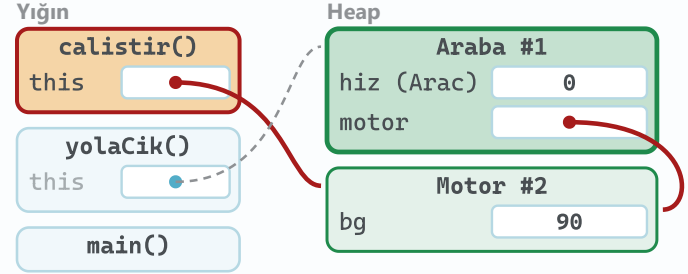


is-a ve has-a: Kod İzleme

```

1 class Motor {
2     private int bg;
3     Motor(int bg) { this.bg = bg; }
4     void calistir() {
5         System.out.println("Motor: " + bg + " BG");
6     }
7 }
8 class Araba extends Arac {           // is-a
9     private Motor motor;             // has-a
10    Araba(int bg) { motor = new Motor(bg); }
11    void yolaCik() {
12        motor.calistir(); hizlan(50);
13        System.out.println("Hız: " + hiz);
14    }
15 }
16 new Araba(90).yolaCik(); // main

```



motor.calistir() → işi **Motor nesnesi yapar**

motor.calistir() : Araba işi kendi yapmaz, referansı izleyip Motor nesnesine **devreder**.

Adım 4 / 6

ÇIKTI

Motor: 90 BG

is-a ve has-a: Kod İzleme

```
1 class Motor {  
2     private int bg;  
3     Motor(int bg) { this.bg = bg; }  
4     void calistir() {  
5         System.out.println("Motor: " + bg + " BG");  
6     }  
7 }  
8 class Araba extends Arac {           // is-a  
9     private Motor motor;             // has-a  
10    Araba(int bg) { motor = new Motor(bg); }  
11    void yolaCik() {  
12        ▶ motor.calistir(); hizlan(50);  
13        System.out.println("Hız: " + hiz);  
14    }  
15 }  
16 new Araba(90).yolaCik(); // main
```

Yığın

yolaCik()
this

main()

Heap

Araba #1
hiz (Arac) 50
motor

Motor #2
bg 90

hizlan(50): Arac'tan **kalıtımla** gelen metot

hizlan(50) Araba'da yazılmadı; Arac 'tan kalıtımla gelir ve aynı nesnenin hiz alanını 50 yapar.

Adım 5 / 6

ÇIKTI

Motor: 90 BG

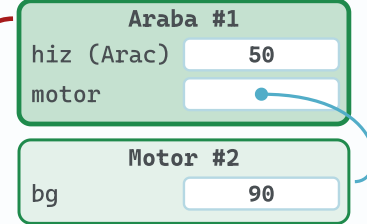
is-a ve has-a: Kod İzleme

```
1 class Motor {  
2     private int bg;  
3     Motor(int bg) { this.bg = bg; }  
4     void calistir() {  
5         System.out.println("Motor: " + bg + " BG");  
6     }  
7 }  
8 class Araba extends Arac {           // is-a  
9     private Motor motor;             // has-a  
10    Araba(int bg) { motor = new Motor(bg); }  
11    void yolaCik() {  
12        motor.calistir(); hizlan(50);  
13        System.out.println("Hız: " + hiz);  
14    }  
15 }  
16 new Araba(90).yolaCik(); // main
```

Yığın



Heap



Kalıtım (`hiz` , `hizlan`) ve oluşum (`motor`)
aynı sınıfta birlikte çalıştı.

Adım 6 / 6

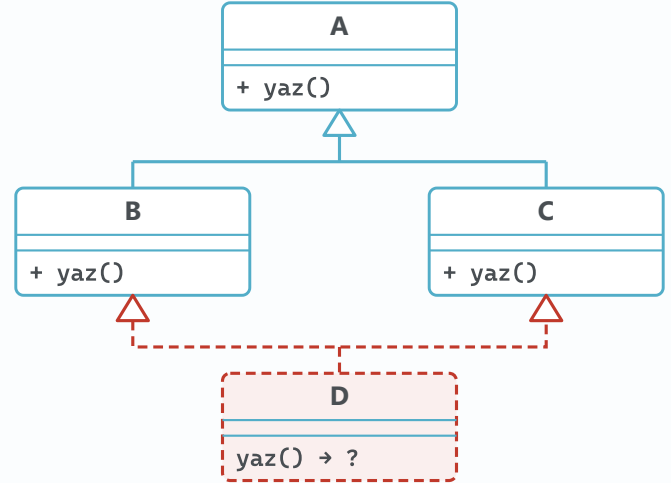
ÇIKTI

```
Motor: 90 BG  
Hız: 50
```

SINIFLAR ARASI İLİŞKİLER

Çoklu Kalıtım

- Java'da **doğrudan çoklu kalıtım desteklenmez**: bir sınıf yalnızca **bir** sınıftan türeyebilir.
- Elmas problemi** (diamond problem): B ve C `yaz()` 'ı farklı geçersiz kılırsa D hangisini almalı?
- Birden fazla sınıfın özellikleri gerekiyorsa: **oluşum** (composition) ve **arayüzler** (interface; bir sınıf birçok arayüz gerçekleştirebilir, ayrıntısı "Soyut Sınıflar" bölümünde).



```
class D extends B, C { } // geçersiz
```

```
error: '{' expected
```

```
interface Sarj { void sarjEt(); }
class ElektrikliAraba extends Araba
    implements Sarj {
    @Override
    public void sarjEt() {
        System.out.println("Şarj ediliyor.");
    }
}
```

Kalıtımla Tasarım İlkeleri

“is-a” testi

“Her X bir Y midir?” cümlesi doğru değilse kalıtım kullanmayın. Araba bir Motor **değildir**; Motor'a **sahiptir** → oluşum.

Herkes kendi verisini yönetir

Üst sınıfın alanlarına `super(...)` ve üst sınıfın metotlarıyla değer verin; alanları `private` tutun.

Genel metotları geçersiz kılın

`toString` ve `equals` (ve onunla birlikte `hashCode`) her anlamlı sınıfta geçersiz kılınmalıdır.

Ortak kavramlar soyut sınıfa

Alt sınıfların ortak kavramları **soyut** (abstract) sınıflarla temsil edilmelidir: “Hayvan” tek başına nesne olmamalıysa.

OBJECT SINIFI

Object Sınıfı

- Her sınıf `java.lang.Object` sınıfından türer; `extends` yazılmamışsa **dolaylı** olarak.
- Bu yüzden her nesnede aşağıdaki metotlar **hazır** bulunur.
- Varsayılan sürümleri genellikle işimize yaramaz; anlamlı sınıflarda geçersiz kılınırlar.

```
class Nokta { }  
// derleyicinin gördüğü:  
class Nokta extends Object { }
```

Metot	Varsayılan davranış
<code>String toString()</code>	Sınıf adı + <code>@</code> + hashCode değerinin onaltılık (hex) yazımı
<code>boolean equals(Object o)</code>	Referansları karşılaştırır: <code>this == o</code>
<code>int hashCode()</code>	Nesneye özgü bir tam sayı (genellikle her nesnede farklı)
<code>Class getClass()</code>	Nesnenin çalışma zamanındaki sınıfı (final; geçersiz kılinamaz)

OBJECT SINIFI

toString Metodu

- `public String toString()` : nesneyi metin olarak gösterir. `println(nesne)` ve `"..." + nesne` onu **kendiliğinden** çağırır.
- Geçersiz kılınmazsa sınıf adı ve nesnenin hashCode değerini içeren bir dize döner.

GEÇERSİZ KILINMADAN

```
class Nokta {  
    private int x, y;  
    Nokta(int x, int y) {  
        this.x = x; this.y = y;  
    }  
}  
// main içinde  
System.out.println(new Nokta(3, 4));
```

Nokta@28a418fc

@ sonrası değer her çalıştırmada değişebilir; okunaklı değildir.

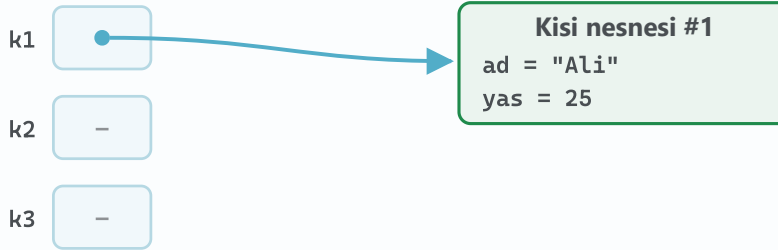
GEÇERSİZ KILINARAK

```
class Nokta {  
    private int x, y;  
    Nokta(int x, int y) {  
        this.x = x; this.y = y;  
    }  
    @Override  
    public String toString() {  
        return "Nokta[x=" + x + ", y=" + y + "];"  
    }  
}  
// main içinde  
Nokta n = new Nokta(3, 4);  
System.out.println(n);  
System.out.println("Konum: " + n);
```

Nokta[x=3, y=4]
Konum: Nokta[x=3, y=4]

OBJECT SINIFI · KODU ADIM ADIM ÇALIŞTIR

== mi, equals mı?



`new` bir nesne oluşturur; `k1` onun **adresini** (referansını) tutar.

Adım 1 / 6

```
► Kisi k1 = new Kisi("Ali", 25);  
2 Kisi k2 = new Kisi("Ali", 25);  
3 Kisi k3 = k1;  
4 System.out.println(k1 == k2);  
5 System.out.println(k1 == k3);  
6 System.out.println(k1.equals(k2));
```

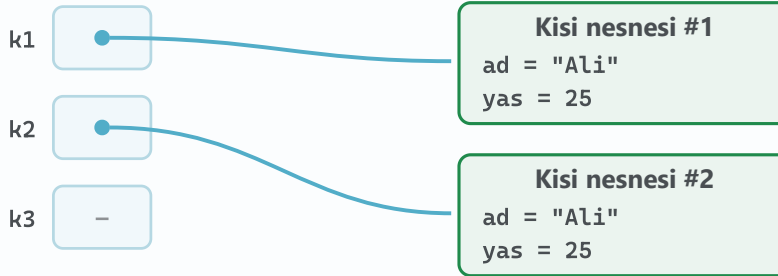
DEĞİŞKENLER

k1 = →#1

ÇIKTI

OBJECT SINIFI · KODU ADIM ADIM ÇALIŞTIR

== mi, equals mı?



İkinci `new` : içeriği aynı, ama **ayrı** bir nesne. `k2` onu gösterir.

Adım 2 / 6

```
1 Kisi k1 = new Kisi("Ali", 25);  
2 ► Kisi k2 = new Kisi("Ali", 25);  
3 Kisi k3 = k1;  
4 System.out.println(k1 == k2);  
5 System.out.println(k1 == k3);  
6 System.out.println(k1.equals(k2));
```

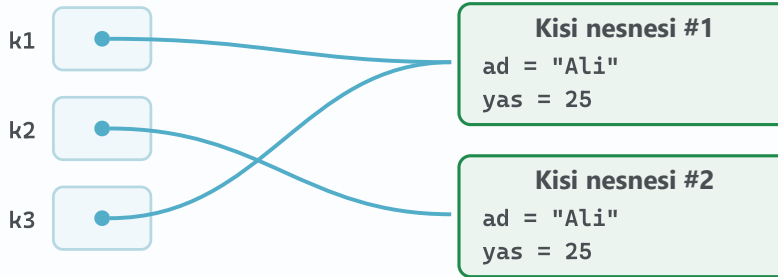
DEĞİŞKENLER

k1 = →#1 k2 = →#2

ÇIKTI

OBJECT SINIFI · KODU ADIM ADIM ÇALIŞTIR

== mi, equals mı?



`k3 = k1` nesne kopyalamaz; yalnızca adresi kopyalar. k1 ve k3 **aynı** nesneyi gösterir.

Adım 3 / 6

```
1 Kisi k1 = new Kisi("Ali", 25);  
2 Kisi k2 = new Kisi("Ali", 25);  
3 ► Kisi k3 = k1;  
4 System.out.println(k1 == k2);  
5 System.out.println(k1 == k3);  
6 System.out.println(k1.equals(k2));
```

DEĞİŞKENLER

k1 = →#1

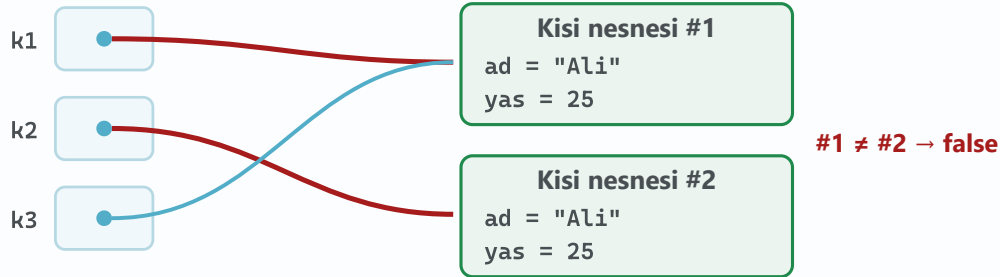
k2 = →#2

k3 = →#1

ÇIKTI

OBJECT SINIFI · KODU ADIM ADIM ÇALIŞTIR

== mi, equals mı?



== referansları karşılaştırır: k1 ve k2 farklı nesneleri gösterir → **false** .

Adım 4 / 6

```
1 Kisi k1 = new Kisi("Ali", 25);  
2 Kisi k2 = new Kisi("Ali", 25);  
3 Kisi k3 = k1;  
4 System.out.println(k1 == k2);  
5 System.out.println(k1 == k3);  
6 System.out.println(k1.equals(k2));
```

DEĞİŞKENLER

k1 = →#1

k2 = →#2

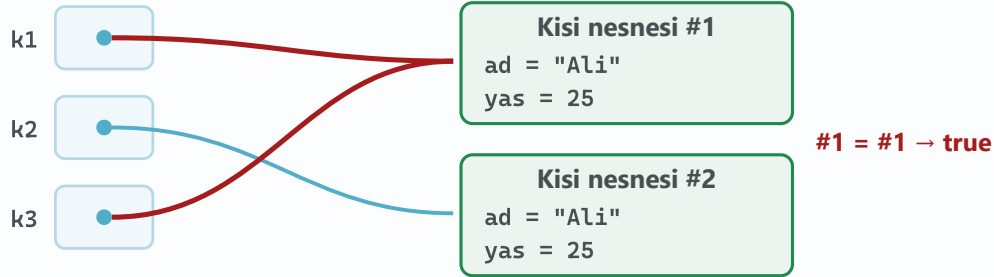
k3 = →#1

ÇIKTI

false

OBJECT SINIFI · KODU ADIM ADIM ÇALIŞTIR

== mi, equals mı?



k1 ve k3 aynı nesneyi gösterir → **true** .

Adım 5 / 6

```
1 Kisi k1 = new Kisi("Ali", 25);  
2 Kisi k2 = new Kisi("Ali", 25);  
3 Kisi k3 = k1;  
4 System.out.println(k1 == k2);  
▶ System.out.println(k1 == k3);  
6 System.out.println(k1.equals(k2));
```

DEĞİŞKENLER

k1 = →#1

k2 = →#2

k3 = →#1

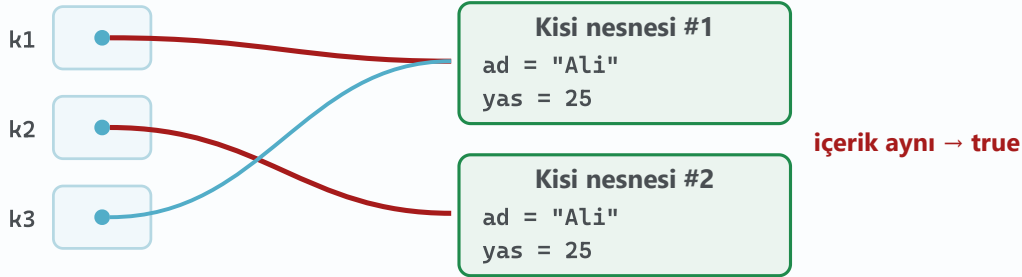
ÇIKTI

false

true

OBJECT SINIFI · KODU ADIM ADIM ÇALIŞTIR

== mi, equals mı?



`equals`, Kisi'de geçersiz kılındığı için **içeriği** (ad, yas) karşılaştırır → `true`. Geçersiz kılınmasaydı `false` olurdu.

Adım 6 / 6

```
1 Kisi k1 = new Kisi("Ali", 25);
2 Kisi k2 = new Kisi("Ali", 25);
3 Kisi k3 = k1;
4 System.out.println(k1 == k2);
5 System.out.println(k1 == k3);
▶ System.out.println(k1.equals(k2));
```

DEĞİŞKENLER

k1 = →#1

k2 = →#2

k3 = →#1

ÇIKTI

```
false
true
true
```

OBJECT SINIFI

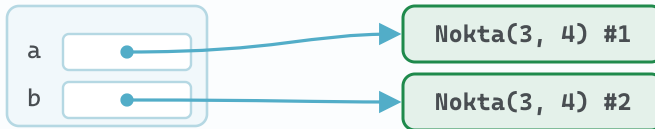
equals Metodu

- `public boolean equals(Object other)` : iki nesnenin **eşdeğer** olup olmadığını söyler.
- Varsayılan olarak yalnızca referansları karşılaştırır: geçersiz kılınmazsa `==` operatörü gibi davranır.
- Geçersiz kılınarak davranışı değiştirilir: metod iki nesnenin **durumunu** (alanlarını) karşılaştırmalı ve eşitse `true` dönmelidir.
- Parametre `Object` tipindedir; hatalı durumları engellemek için önce **tip kontrolü** yapılmalıdır.

EQUALS SÖZLEŞMESİ

Yansıma	<code>x.equals(x)</code> → true
Simetri	<code>x.equals(y)</code> ⇔ <code>y.equals(x)</code>
Geçişme	<code>x = y</code> ve <code>y = z</code> ise <code>x = z</code>
Tutarlılık	Alanlar değişmedikçe sonuç değişmez
null	<code>x.equals(null)</code> → false

String karşılaştırmasında da kural aynıdır:
`s1 == s2` değil, `s1.equals(s2)`.



`a == b` **false**
`Object.equals` (varsayılan) **false**
geçersiz kılınmış `equals` **true**

OBJECT SINIFI

instanceof Operatörü

- Bir nesnenin belirli bir sınıfa veya arayüze **uyup uymadığını** kontrol eder.
- Kullanım: `nesne instanceof Tür` ; sonuç `true` veya `false` .
- Alt sınıf nesnesi, üst sınıfın da örneğidir.
- `null` bir nesne kabul edilmez: `null instanceof Tür` her zaman `false` döner. Bu yüzden `instanceof` ayrıca null kontrolü gerektirmez.
- Aşağı dönüşümden (downcasting) önce kullanılır; yanlış dönüşüm çalışma zamanında `ClassCastException` fırlatır.

```
Exception in thread "main"  
java.lang.ClassCastException: class Parca  
cannot be cast to class Lastik
```

```
Parca p = new Lastik("Petlas", "XYZ-123",  
                    "225/45R17", true);  
  
if (p instanceof Lastik) {  
    Lastik l = (Lastik) p;    // güvenli dönüşüm  
    System.out.println(l.kislikMi());  
}  
  
System.out.println(p instanceof Parca);  
System.out.println(p instanceof Object);  
Parca bos = null;  
System.out.println(bos instanceof Parca);  
Parca duz = new Parca("Bosch", "ABC-9");  
System.out.println(duz instanceof Lastik);
```

```
true  
true  
true  
false  
false
```

JAVA 16+: ÖRÜNTÜ EŞLEME

```
if (p instanceof Lastik l) {    // kontrol + dönüşüm  
    System.out.println(l.kislikMi());  
}
```

OBJECT SINIFI

equals Metodunu Yazmak

instanceof İLE TİP KONTROLÜ

```
public boolean equals(Object nesne) {  
    if (nesne instanceof Nokta) {  
        // nesne bir Nokta nesnesidir;  
        // dönüştürüp karşılaştır  
        Nokta diger = (Nokta) nesne;  
        return x == diger.x && y == diger.y;  
    } else {  
        // nesne bir Nokta değildir;  
        // eşit olamaz  
        return false;  
    }  
}
```

- Parametre tipi **Object** olmalıdır;
`equals(Nokta n)` yazmak geçersiz kılma değil, **aşırı yüklem**dir.
- Dönüşüm (`(Nokta) nesne`) yalnızca tip kontrolünden sonra güvenlidir.

DENEME

```
Nokta a = new Nokta(3, 4);  
Nokta b = new Nokta(3, 4);  
System.out.println(a == b);  
System.out.println(a.equals(b));  
System.out.println(a.equals("3,4"));  
System.out.println(a.equals(null));
```

```
false  
true  
false  
false
```

"3,4" bir Nokta değildir → false. null instanceof Nokta da false olduğu için ayrı bir null kontrolüne gerek kalmaz.

OBJECT SINIFI

getClass() ile equals

```
public class Kisi {  
    private String ad;  
    private int yas;  
    // ... yapıcı ...  
  
    @Override  
    public boolean equals(Object obj) {  
        // aynı referanssa eşittir  
        if (this == obj) return true;  
        // null ya da farklı sınıf: eşit değil  
        if (obj == null  
            || getClass() != obj.getClass())  
            return false;  
        // alanları karşılaştır  
        Kisi kisi = (Kisi) obj;  
        return yas == kisi.yas  
            && ad.equals(kisi.ad);  
    }  
}
```

- `this == obj` : aynı nesneyse hemen `true` (hızlı yol).
- `getClass()` nesnenin **gerçek** sınıfını verir; iki tarafın sınıfı birebir aynı olmalıdır.
- İlkel tipler (`int yas`) `==` ile, nesne tipindeki alanlar (`String ad`) `equals` ile karşılaştırılır.

Tip kontrolü	Alt sınıf nesnesi eşit sayılabilir mi?
<code>instanceof</code>	Evet (Kedi nesnesi bir Hayvan'dır)
<code>getClass()</code>	Hayır (sınıflar birebir aynı olmalı)

OBJECT SINIFI

equals ve hashCode

Kural: equals geçersiz kılınıyorsa hashCode da geçersiz kılınmalıdır. Eşit nesnelerin hashCode değerleri **aynı** olmalıdır.

```
import java.util.Objects;

public class Nokta {
    private int x, y;
    public Nokta(int x, int y) {
        this.x = x; this.y = y;
    }
    @Override public boolean equals(Object o) {
        return (o instanceof Nokta n)
            && x == n.x && y == n.y;
    }
    @Override public int hashCode() {
        return Objects.hash(x, y);
    }
    @Override public String toString() {
        return "Nokta[x=" + x + ", y=" + y + "];"
    }
}
```

Neden? `HashSet` ve `HashMap` önce hashCode ile “kova” seçer, sonra `equals` çağırır. hashCode farklıysa eşit nesneler hiç karşılaştırılmaz.

```
HashSet<Nokta> kume = new HashSet<>();
kume.add(new Nokta(3, 4));
kume.add(new Nokta(3, 4));
System.out.println(kume.size() + " "
    + kume.contains(new Nokta(3, 4)));
```

Nokta sınıfında	Çıktı
yalnızca <code>equals</code>	2 false ✗
<code>equals</code> + <code>hashCode</code>	1 true ✓

`o instanceof Nokta n` örüntü eşlemesi Java 16+ özelliğidir; klasik biçim: `if (o instanceof Nokta) { Nokta n = (Nokta) o; ... }`.

HashSet İçinde equals ve hashCode: Kod İzleme

```
1 public boolean equals(Object o) {  
2     if (!(o instanceof Nokta)) return false;  
3     Nokta n = (Nokta) o;  
4     return x == n.x && y == n.y;  
5 }  
6 public int hashCode() {  
7     return Objects.hash(x, y); // 31·(31 + x) + y  
8 }  
9 // main içinde:  
► HashSet<Nokta> kume = new HashSet<>();  
11 kume.add(new Nokta(3, 4));  
12 kume.add(new Nokta(4, 3));  
13 kume.add(new Nokta(3, 4));  
14 System.out.println(kume.size());  
15 System.out.println(kume.contains(new Nokta(4, 3)));
```

kume: 16 kova (kova = hash % 16)



Boş bir `HashSet` : öğeler 16 kovadan birine konur. Kovayı `hashCode()` seçer.

Adım 1 / 7

ÇIKTI



HashSet İçinde equals ve hashCode: Kod İzleme

```
1 public boolean equals(Object o) {  
2     if (!(o instanceof Nokta)) return false;  
3     Nokta n = (Nokta) o;  
4     return x == n.x && y == n.y;  
5 }  
▶ public int hashCode() {  
▶     return Objects.hash(x, y); // 31·(31 + x) + y  
8 }  
9 // main içinde:  
10 HashSet<Nokta> kume = new HashSet<>();  
▶ kume.add(new Nokta(3, 4));  
12 kume.add(new Nokta(4, 3));  
13 kume.add(new Nokta(3, 4));  
14 System.out.println(kume.size());  
15 System.out.println(kume.contains(new Nokta(4, 3)));
```

kume: 16 kova (kova = hash % 16)



(3,4)

Nokta(3,4)

$31 \cdot 34 + 4 = 1058$

$1058 \% 16 = 2 \rightarrow$ kova boş, eklenir

`hashCode()` = $31 \cdot (31 + 3) + 4 = 1058$; $1058 \% 16 = 2$. Kova boş: nesne eklenir.

Adım 2 / 7

ÇIKTI



HashSet İçinde equals ve hashCode: Kod İzleme

```
1 public boolean equals(Object o) {  
2     if (!(o instanceof Nokta)) return false;  
3     Nokta n = (Nokta) o;  
4     return x == n.x && y == n.y;  
5 }  
6 public int hashCode() {  
7     return Objects.hash(x, y); // 31·(31 + x) + y  
8 }  
9 // main içinde:  
10 HashSet<Nokta> kume = new HashSet<>();  
11 kume.add(new Nokta(3, 4));  
12 ► kume.add(new Nokta(4, 3));  
13 kume.add(new Nokta(3, 4));  
14 System.out.println(kume.size());  
15 System.out.println(kume.contains(new Nokta(4, 3)));
```

kume: 16 kova (kova = hash % 16)



(3,4)

(4,3)

Nokta(4,3)

$31 \cdot 35 + 3 = 1088$

$1088 \% 16 = 0 \rightarrow$ kova boş, eklenir

Aynı sayılar, farklı sıra: hash **1088**, kova 0. Yine boş kova, eklenir.

Adım 3 / 7

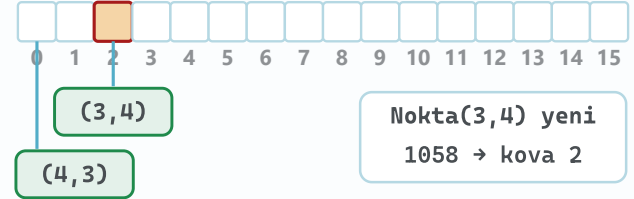
ÇIKTI



HashSet İçinde equals ve hashCode: Kod İzleme

```
1 public boolean equals(Object o) {  
2     if (!(o instanceof Nokta)) return false;  
3     Nokta n = (Nokta) o;  
4     return x == n.x && y == n.y;  
5 }  
6 public int hashCode() {  
7     return Objects.hash(x, y); // 31·(31 + x) + y  
8 }  
9 // main içinde:  
10 HashSet<Nokta> kume = new HashSet<>();  
11 kume.add(new Nokta(3, 4));  
12 kume.add(new Nokta(4, 3));  
13 ► kume.add(new Nokta(3, 4));  
14 System.out.println(kume.size());  
15 System.out.println(kume.contains(new Nokta(4, 3)));
```

kume: 16 kova (kova = hash % 16)



Yeni bir `Nokta(3, 4)` nesnesi: hash yine 1058, aynı kova 2. Kova dolu: içindekiyle `equals` sorulur.

Adım 4 / 7

ÇIKTI



HashSet İçinde equals ve hashCode: Kod İzleme

```
▶ public boolean equals(Object o) {  
2   if (!(o instanceof Nokta)) return false;  
3   Nokta n = (Nokta) o;  
▶   return x == n.x && y == n.y;  
5 }  
6 public int hashCode() {  
7   return Objects.hash(x, y); // 31·(31 + x) + y  
8 }  
9 // main içinde:  
10 HashSet<Nokta> kume = new HashSet<>();  
11 kume.add(new Nokta(3, 4));  
12 kume.add(new Nokta(4, 3));  
13 kume.add(new Nokta(3, 4));  
14 System.out.println(kume.size());  
15 System.out.println(kume.contains(new Nokta(4, 3)));
```

kume: 16 kova (kova = hash % 16)



(3,4)

(4,3)

Nokta(3,4) yeni
equals → true

3 == 3 && 4 == 4 → true: eklenmez

equals doğru: küme aynı öğeyi ikinci kez
almaz, add false döndürür.

Adım 5 / 7

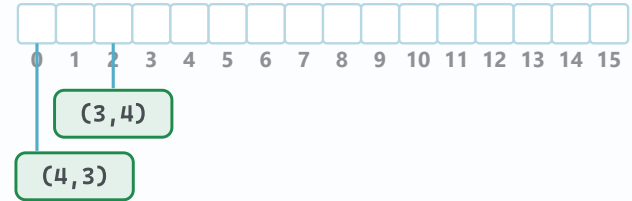
ÇIKTI



HashSet İçinde equals ve hashCode: Kod İzleme

```
1 public boolean equals(Object o) {  
2     if (!(o instanceof Nokta)) return false;  
3     Nokta n = (Nokta) o;  
4     return x == n.x && y == n.y;  
5 }  
6 public int hashCode() {  
7     return Objects.hash(x, y); // 31·(31 + x) + y  
8 }  
9 // main içinde:  
10 HashSet<Nokta> kume = new HashSet<>();  
11 kume.add(new Nokta(3, 4));  
12 kume.add(new Nokta(4, 3));  
13 kume.add(new Nokta(3, 4));  
14 ▶ System.out.println(kume.size());  
15 System.out.println(kume.contains(new Nokta(4, 3)));
```

kume: 16 kova (kova = hash % 16)



Kümede iki öge var: 2.

Adım 6 / 7

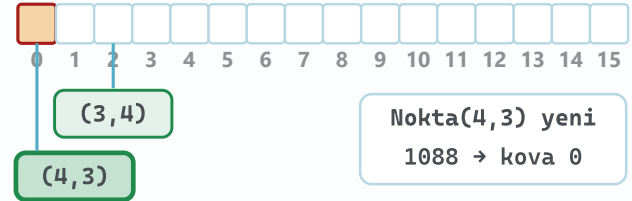
ÇIKTI

2

HashSet İçinde equals ve hashCode: Kod İzleme

```
1 public boolean equals(Object o) {  
2     if (!(o instanceof Nokta)) return false;  
3     Nokta n = (Nokta) o;  
4     return x == n.x && y == n.y;  
5 }  
6 public int hashCode() {  
7     return Objects.hash(x, y); // 31*(31 + x) + y  
8 }  
9 // main içinde:  
10 HashSet<Nokta> kume = new HashSet<>();  
11 kume.add(new Nokta(3, 4));  
12 kume.add(new Nokta(4, 3));  
13 kume.add(new Nokta(3, 4));  
14 System.out.println(kume.size());  
15 System.out.println(kume.contains(new Nokta(4, 3)));
```

kume: 16 kova (kova = hash % 16)



kova 0 → equals **true** → bulundu

hashCode yoksa: farklı kova, **equals sorulmaz**

contains da önce kovayı bulur, sonra
equals sorar: **true**. hashCode yazılmasaydı
eşit nesneler farklı kovalara düşerdi.

Adım 7 / 7

ÇIKTI

```
2  
true
```

ÖZET

Özet

Kavram	Özü
Kalıtım	<code>class Alt extends Ust</code> ; alt sınıf üst sınıfın üyelerini kalıtımla alır, yenilerini ekler (is-a)
Kalıtımla alınmayan	Yapıcılar; <code>private</code> üyeler nesnede vardır ama doğrudan erişilemez
super	<code>super(...)</code> : üst sınıf yapıcısı (ilk satır); <code>super.m()</code> : üst sınıfın sürümü
Yapıcı zinciri	Object → ... → Ust → Alt sırasıyla tamamlanır
Geçersiz kılma	Aynı imza, alt sınıfta yeni gövde; <code>@Override</code> ile derleyici denetler
protected	Sınıf + alt sınıflar + aynı paket
final	Metot: geçersiz kılınamaz; sınıf: alt sınıfı olamaz
Has-a	Oluşum: başka sınıf tipinde alan; çoklu kalıtım yerine oluşum ve arayüzler
Object	Her sınıfın atası; <code>toString</code> , <code>equals</code> , <code>hashCode</code> geçersiz kılınır
== / equals	Referans karşılaştırma / içerik karşılaştırma

Sık Yapılan Hatalar

super(...) çağırmamak

Üst sınıfta parametresiz yapıcı yoksa: `constructor`
Hayvan in class Hayvan cannot be applied to
given types .

equals(Nokta) yazmak

Parametre `Object` değilse aşırı yüklemidir;
`@Override` ile: `method does not override or
implement a method from a supertype` .

equals var, hashCode yok

Derleme hatası yoktur ama `HashSet` / `HashMap`
yanlış çalışır: aynı nokta iki kez eklenir.

Erişimi daraltmak

`public void ciz()` alt sınıfta `void ciz()`
olursa: `attempting to assign weaker access
privileges` .

private alana doğrudan erişmek

`saglik += 5;` alt sınıfta: `saglik has private
access in Hayvan` . Getter veya protected metot
kullanın.

Nesneleri == ile karşılaştırmak

İçerikleri aynı iki nesne için `==` → `false` .
`String` dahil, `equals` kullanın.

İki sınıftan türetmeye çalışmak

`class D extends B, C` → `'{' expected` .
Oluşum veya arayüz kullanın.

Kontrolsüz aşağı dönüşüm

`(Lastik) p` , p bir Lastik değilse:
`ClassCastException` . Önce `instanceof` .

ÖZET

Örnek Problemler

```
class A {  
    A() { System.out.println("A"); }  
    void yaz() { System.out.println("A.yaz"); }  
}  
class B extends A {  
    B() { System.out.println("B"); }  
    @Override void yaz() { System.out.println("B.yaz"); }  
}  
class C extends B {  
    C() { super(); System.out.println("C"); }  
    @Override void yaz() {  
        super.yaz();  
        System.out.println("C.yaz");  
    }  
}
```

1. `C c = new C();` ne yazdırır?
2. Ardından `c.yaz();` ne yazdırır?
3. `A a = new B(); a.yaz();` ne yazdırır?
4. `c instanceof A` nedir?
5. `equals` 'ı geçersiz kılmayan bir `Ogrenci` sınıfında, aynı adlı iki nesne için `o1.equals(o2)` nedir?

► Cevapları göster

ÖZET

Alıştırmalar

1. `Calisan` (`ad` , `maas`) ve ondan türeyen `Yoneticisi` (`prim`) yazın; `toplamGelir()` ve `super.toString()` kullanan bir `toString()` ekleyin.
2. `Hayvan` 'ın tek yapıcısı `Hayvan(String isim)` . `Kus(String isim, boolean ucabilir)` yapıcısını yazın.
3. `Ogrenci` (`int no` , `String ad`) için `equals` ve `hashCode` yazın.
4. Başka paketteki, alt sınıf **olmayan** bir sınıf `protected` bir alana erişebilir mi?
5. is-a mı, has-a mı? Öğrenci–Kişi, Araba–Tekerlek, Kütüphane–Kitap, Kare–Şekil.

► Cevaplar / ipuçları

Kaynaklar

- Oracle, *The Java Tutorials*: Inheritance; Overriding and Hiding Methods; Using the Keyword super; Object as a Superclass.
- J. Gosling ve diğ., *The Java Language Specification*, Java SE 21 Edition (8.1.4 Superclasses and Subclasses, 8.4.8 Inheritance, Overriding, and Hiding).
- P. Deitel ve H. Deitel, *Java: How to Program*, 11. baskı, Pearson (Bölüm 9: Object-Oriented Programming: Inheritance).
- K. Sierra ve B. Bates, *Head First Java*, 3. baskı, O'Reilly (Bölüm 7: Inheritance and Polymorphism).
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12. baskı, Pearson (Bölüm 5: Inheritance).
- J. Bloch, *Effective Java*, 3. baskı, Addison-Wesley (Madde 10–11: equals ve hashCode; Madde 18: oluşumu kalıtıma tercih edin).

Son

Bölüm 4: Kalıtım

Sonraki bölüm: Bölüm 4: Çok Biçimlilik · [Terim sözlüğü](#)