Bölüm 4.1: Nesneler ve Sınıflar

Java ile Nesne Yönelimli Programlama

Sınıf ve nesne | Yapıcı metotlar | Alan ve metot türleri | static üyeler | Erişim ve veri gizleme | Referanslar ve this | == ve equals | Wrapper sınıflar | record ve enum | İç sınıflar | Sınıf türleri

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Sınıf ve nesne

Sınıfı bir taslak, nesneyi o taslaktan üretilmiş somut bir örnek olarak açıklamak; alan ve metotlarıyla bir sınıf yazmak.

Nesne oluşturmak

`new` işlecini ve yapıcı metotları (constructor) kullanmak; varsayılan değerleri ve varsayılan yapıcıyı bilmek.

Örnek ve sınıf üyeleri

Örneğe ait ve `static` alan/metotları ayırt etmek; ilklendirici bloklarının çalışma sırasını izlemek.

Referanslar

Değişkenin nesneyi değil **referansı** tuttuğunu bellek çizimiyle göstermek; `this`, `==` ve `equals` farkını açıklamak.

Yapıcı zinciri ve erişim

`this(...)` ile yapıcıları zincirlemek; erişim belirleyicilerin ve veri gizlemenin amacını açıklamak.

Özel sınıf türleri

Wrapper sınıfları, `record`, `enum` ve iç sınıfları tanımak; `RasyonelSayi` örneğiyle bir sınıfı baştan tasarlamak.

Nesne Yönelimli Programlama

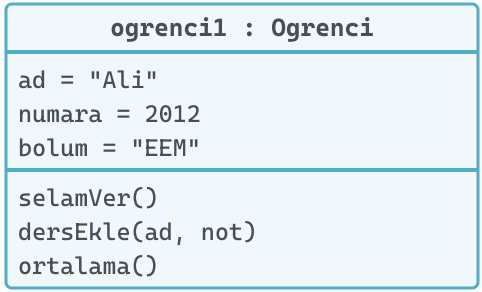
- **Nesne yönelimli programlama** (Object-Oriented Programming, OOP), programı **birbiriyle etkileşen nesneler** olarak kuran bir yaklaşımdır.
- **Nesne** (object), gerçek dünyada açıkça tanımlanabilen bir **varlığı** temsil eder: bir öğrenci, bir masa, bir çember, ekrandaki bir düğme, hatta bir kredi.
- Her nesnenin **durumu** (verileri) ve **davranışı** (yapabildikleri) vardır.

Benzetme: bir okul, öğrenci, öğretmen ve ders nesnelerinin birbirine **mesaj göndermesiyle** (metot çağırmasıyla) işler.

Nesne	Alanlar (durum)	Davranışlar
Öğrenci	ad, numara, bölüm	derseKaydol(), notGor()
Masa	en, boy, malzeme	tasi()
Çember	merkez, yarıçap	alan(), cevre()
Düğme	yazı, renk, etkin mi	tikla()
Kredi	tutar, faiz, vade	taksitHesapla()

Nesneler: Alan ve Davranış

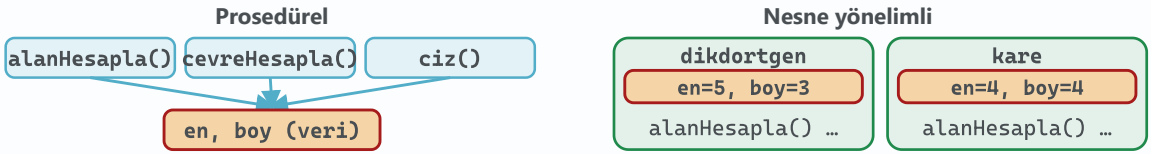
- Bir sınıfın **üyeleri** (members) vardır: **alanlar** (fields), **metotlar** (methods) ve **yapıcılar** (constructors).
- Nesneler **veri saklar**: bu veri parçalarına **alan** (field) denir.
- Nesneler **işlem yapar**: bu işlemlere **davranış** (metot, method) denir.
- Veriyi ve o veri üzerinde çalışan kodu bir araya getirmek, programı daha **düzenli**, **yeniden kullanılabilir** ve **sezgisel** yapar.



Prosedürel ve Nesne Yönelimli

	Prosedürel (yordamsal) yaklaşım	Nesne yönelimli yaklaşım
Temel birim	Fonksiyon (metot)	Nesne (veri + metotlar)
Veri	Ayrı değişkenlerde; her fonksiyon erişebilir	Nesnenin içinde; dışarıya kontrollü açılır
Örnek	`alanHesapla(en, boy)`	`dikdortgen.alanHesapla()`
Büyük programlar	Değişiklik birçok fonksiyonu etkiler	Değişiklik çoğunlukla bir sınıfın içinde kalır
Yeniden kullanım	Fonksiyonları kopyalayarak	Sınıfları kullanarak ve kalıtımla

Şimdiye kadar yazdığımız `static` metotlu programlar prosedürel tarzdaydı. Java'da her kod bir sınıfın içindedir; nesne yönelimli programlamada sınıflar **nesne üretmek** için kullanılır.



NESNE YÖNELİMLİ PROGRAMLAMA

Üç Temel Özellik

Sarmalama (Encapsulation)

Nesne, alan ve davranışlarının ayrıntısını diğer nesnelerden **gizleyebilir**. Dışarıya karşı kapalı bir kutu gibidir; yalnızca izin verilen metotlarla kullanılır.

Kalıtım (Inheritance)

Bir sınıf, başka bir sınıfın alan ve metotlarını **kalıtımla alabilir**. Kodun yeniden kullanımını ve sınıf hiyerarşisini sağlar.

Çok Biçimlilik (Polymorphism)

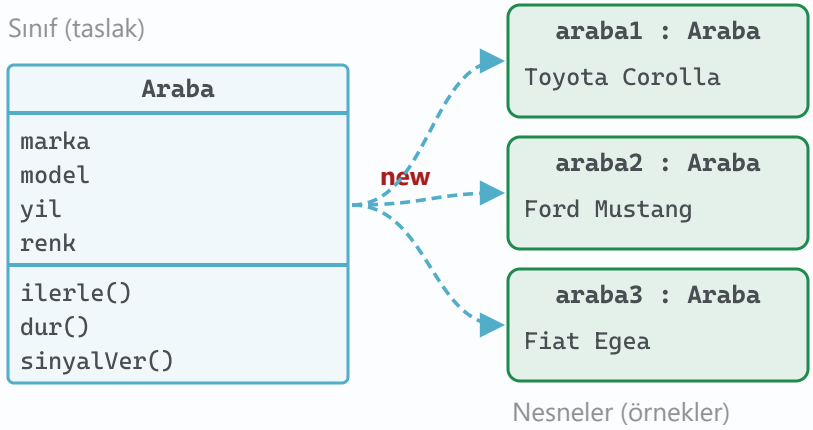
Bir nesne **birden fazla biçim** (tip) alabilir; aynı çağrı farklı nesnelerde farklı davranır. Esneklik ve dinamik davranış sağlar.

Bunlara çoğu zaman dördüncü olarak **soyutlama** (abstraction) eklenir: gereksiz ayrıntıyı gizleyip yalnızca önemli özellikleri göstermek.

SINIF VE NESNE

Sınıf: Nesnenin Taslağı

- **Sınıf** (class), bir nesnenin **taslağıdır** (blueprint); bir sınıftan farklı verileri taşıyan birçok nesne üretilebilir.
- Sınıf, nesnenin taşıyacağı **verileri** (alanlar) ve yapabileceği **işlemleri** (metotlar) belirtir.
- Bir nesne, bir sınıfın **örneğidir** (instance): sınıfın bellekteki somut hâlidir.
- Sınıflar, gerçek dünyadaki varlıkları **modellemeyi** sağlar: bir `Araba` sınıfı her otomobili temsil edebilir; her marka ve model bu sınıftan üretilen bir nesnedir.



Benzetme: sınıf kurabiye kalıbı, nesneler o kalıpla kesilen kurabiyelerdir.

Örnek: Araba Sınıfı

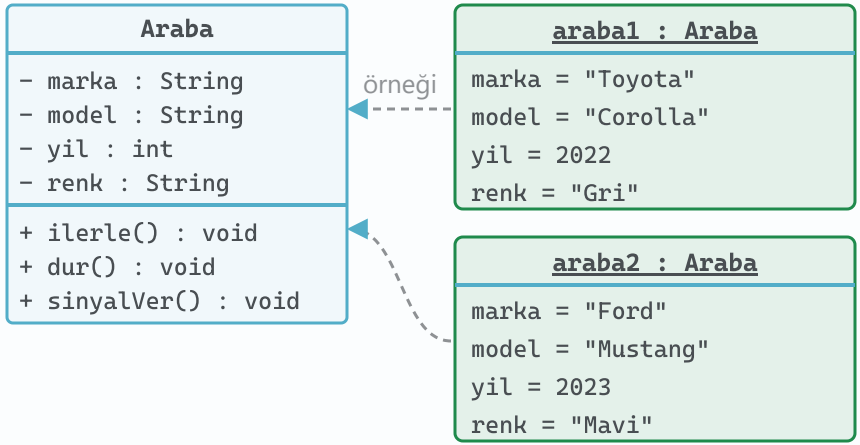
Alanlar (veriler)

Marka, Model, Yıl, Renk

Metotlar (işlemler)

Hareket et (ilerle), Dur, Sinyal ver

Aynı sınıftan üretilen iki nesne aynı **alan adlarına** ama farklı **değerlere** sahiptir.



UML sınıf diyagramı: ad / alanlar / metotlar; `-` private, `+` public. Nesne adları altı çizili yazılır.

SINIF VE NESNE

Sınıf Tanımlama

```
public class Araba {
  // Alanlar (fields)
  String marka;
  String model;
  int yil;
  String renk;

  // Metotlar (methods)
  void ilerle() {
    System.out.println(marka + " ilerliyor.");
  }

  void dur() {
    System.out.println(marka + " durdu.");
  }

  void sinyalVer() {
    System.out.println(marka + " sinyal veriyor.");
  }
}
```

- `class` anahtar sözcüğü ve sınıf adıyla başlar; gövde `{ }` arasındadır.
- Sınıf adı **büyük harfle** başlar (`Araba` , `BankaHesabi`); `public` bir sınıf, **aynı adlı dosyada** (`Araba.java`) bulunmalıdır.
- Alanlar sınıf gövdesinde, metotların **dışında** tanımlanır; her metot bu alanları doğrudan kullanabilir.
- Metotların önünde `static` **yok**: her biri belirli bir arabanın (nesnenin) metodudur.
- Bu sınıfta `main` yok; tek başına çalışmaz, başka bir sınıf onu kullanır.

Sınıfın Kullanımı

```
public class AracTesti {
  public static void main(String[] args) {
    Araba araba1 = new Araba();
    araba1.marka = "Toyota";
    araba1.model = "Corolla";
    araba1.yil = 2022;
    araba1.renk = "Gri";

    Araba araba2 = new Araba();
    araba2.marka = "Ford";
    araba2.model = "Mustang";
    araba2.yil = 2023;
    araba2.renk = "Mavi";

    araba1.ilerle();
    araba2.sinyalVer();
    araba1.dur();
  }
}
```

ÇIKTI

```
Toyota ilerliyor.
Ford sinyal veriyor.
Toyota durdu.
```

- `new Araba()` bir nesne üretir; `araba1` onu gösterir.
- `araba1.marka = "Toyota";` : nokta (`.`) ile nesnenin alanına erişilir.
- `araba1.ilerle()` : metot, **araba1'in** verileriyle çalışır; `araba2.sinyalVer()` ise araba2'ninkilerle.
- Alanları dışarıdan böyle doğrudan değiştirmek iyi bir alışkanlık değildir; doğrusu **yapıcı** ve **erişim metotları** kullanmaktır.

SINIF VE NESNE

Sınıfın Üyeleri

Alanlar (fields)

Nesnelerin taşıyabileceği **verileri** temsil eder. Örnek: bir `Ogrenci` sınıfında `ad` ve `soyad`.

Metotlar (methods)

Nesnelerin yapabileceği **işlemleri** tanımlar. Örnek: bir `Kopek` sınıfında `havla()` ve `kos()`.

Erişim düzeyi

Her üye `private`, `protected`, varsayılan (yazılmaz) veya `public` olabilir: kimin kullanabileceğini belirler.

Sahiplik

Her üye **nesneye ait** (instance, varsayılan) veya `static` ile **sınıfa ait** olabilir.

Yapıcı metotlar da sınıfın içinde yazılır; ama yalnızca nesne oluşturulurken çalışırlar ve teknik olarak metot değil, ayrı bir üye türüdür. Sınıfın içinde başka sınıflar (iç sınıflar) da tanımlanabilir.

Erişim Belirleyiciler ve Veri Gizleme

- **Erişim belirleyiciler** (`public` , `protected` , `private` , yazılmazsa paket erişimi), bir üyeye sınıfın **dışından** kimin erişebileceğini belirtir.
- Genel kural: alanlar `private` , dışarıya sunulan metotlar `public` .
- Alanın değeri bir **erişimci** (getter, `get...`) ile okunur, bir **değiştirici** (setter, `set...`) ile değiştirilir; değiştirici geçersiz değeri reddeder.
- Bu düzen **veri gizlemedir** (data hiding): alanlara yalnızca kendi sınıfının metotları doğrudan erişir. Sarmalamanın (encapsulation) temelidir.

```
public class BankaHesabi {
  private double hesapBakiyesi;   // gizli alan
  public double getHesapBakiyesi() {
    return hesapBakiyesi;
  }
  public void setHesapBakiyesi(double b) {
    if (b >= 0) hesapBakiyesi = b;  // geçersizi reddet
  }
}
// başka bir sınıfta:
hesap.hesapBakiyesi = -500;
```

```
error: hesapBakiyesi has private access in BankaHesabi
```

Yapıcı Metotlar

- **Yapıcı metot** (constructor), nesne oluşturmak için kullanılan **özel bir metottur**; `new` ile nesne oluşturulduğunda **otomatik** çağrılır.
- Görevi, alanlara **ilk değerlerini** vermek ve varsa başka hazırlık işlerini yapmaktır.
- Adı **sınıf adıyla aynıdır** (büyük-küçük harf dahil).
- **Dönüş tipi yoktur**, `void` bile yazılmaz; değer döndüremez.
- Genellikle `public` tanımlanır; nesne oluşturmayı kısıtlamak için `private` da olabilir (ör. singleton deseni).
- Bir sınıfın **birden çok** yapıcısı olabilir (farklı parametrelerle).

```
public class Ogrenci {
  private String ad;
  private int numara;

  // Yapıcı: sınıfla aynı ad, dönüş türü yok
  public Ogrenci(String ad, int numara) {
    this.ad = ad;
    this.numara = numara;
  }
}

// Kullanımı:
Ogrenci ogr = new Ogrenci("Ali", 2012);
```

Nesne Nasıl Oluşturulur?

```
SinifAdi nesneAdi = new SinifAdi(parametreler);

Araba corolla = new Araba("Toyota", "Corolla", 2022);
Scanner klavye = new Scanner(System.in);
Kitap kitap = new Kitap();
```

```
import java.util.Scanner;
public class Olusturma {
  public static void main(String[] args) {
    Scanner klavye = new Scanner(System.in);
    StringBuilder metin = new StringBuilder("Java");
    String baslik = "Nesneler";           // kısayol
    String yazar = new String("Sercan");  // uzun yol
    metin.append(" 21");
    System.out.println(metin + " / " + baslik);
    System.out.println(yazar.length());
    System.out.println(klavye.nextInt() * 2);
  }
}
```

1. `new`, heap'te nesne için **yer ayırır**; alanlar varsayılan değerleri alır.
2. **Yapıcı metot** çağrılır, ilk değerleri verir.
3. `new` ifadesinin sonucu nesnenin **referansıdır**; değişkene atanır.

ÇIKTI

```
(girdi: 21)
Java 21 / Nesneler
6
42
```

Kısayol: `String baslik = "Nesneler";` String nesnesi oluşturur; yalnızca `String baslik;` ise sadece referans tanımlar.

Yapıcı Örneği: Araba

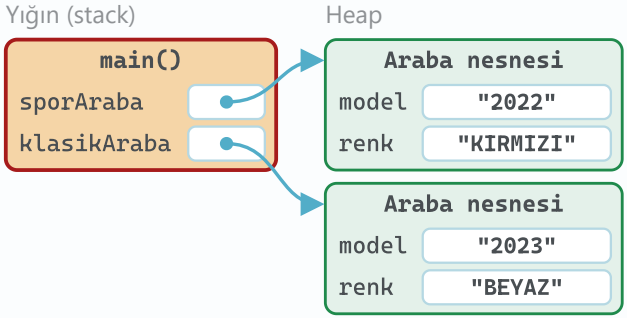
```java
public class Araba {
  private String model;
  private String renk;

  public Araba(String model, String renk) {
    this.model = model;
    this.renk = renk;
  }

  public void bilgiYazdir() {
    System.out.println(model + " - " + renk);
  }

}
```

```java
public static void main(String[] args) {
  Araba sporAraba = new Araba("2022", "KIRMIZI");
  Araba klasikAraba = new Araba("2023", "BEYAZ");
  sporAraba.bilgiYazdir();
  klasikAraba.bilgiYazdir();
}
```



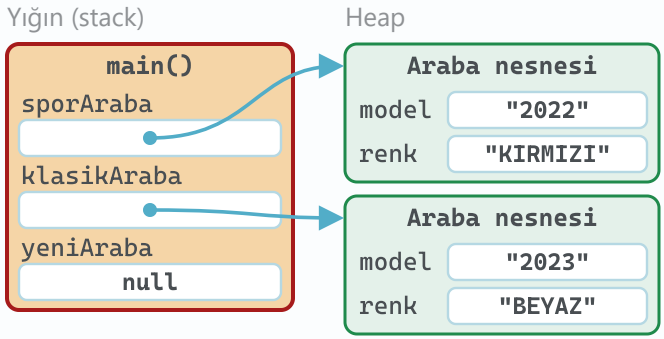
ÇIKTI

```
2022 - KIRMIZI
2023 - BEYAZ
```

Referanslar

- Bir nesne **doğrudan** kullanılamaz; ona yalnızca **referans** (reference, başvuru) ile erişilir. Değişkenlerde nesnenin kendisi değil, referansı saklanır.
- Referans, nesnenin bellekte nerede olduğunu gösteren bir **adres** gibi düşünülebilir.
- `int`, `double`, `boolean` gibi **ilkel tipli** değişkenler değeri kendileri tutar; sınıf tipinden değişkenler (`Araba`, `String`, `Scanner`) **referans** tutar.
- Hiçbir nesneyi göstermeyen referansın değeri `null`'dır.

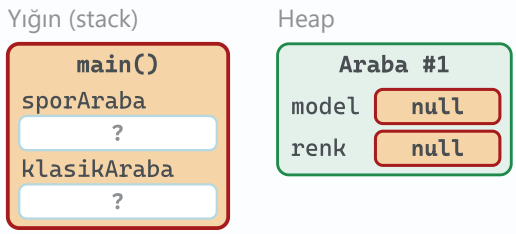
Benzetme: değişken bir **televizyon kumandası**, nesne televizyondur. Kumandayı kopyalamak ikinci bir televizyon yaratmaz.



Yerel değişkenler yığında (stack), nesneler heap'te bulunur; ok = referans.

new Araba(...): Adım Adım

```
1  public class Araba {
2    private String model;
3    private String renk;
4
5    public Araba(String model, String renk) {
6      this.model = model;
7      this.renk = renk;
8    }
9
10   public void bilgiYazdir() {
11     System.out.println(model + " - " + renk);
12   }
13
14   public static void main(String[] args) {
▶      Araba sporAraba = new Araba("2022", "KIRMIZI");
16     Araba klasikAraba = new Araba("2023", "BEYAZ");
17     sporAaraba.bilgiYazdir();
18     klasikAraba.bilgiYazdir();
19   }
20 }
```



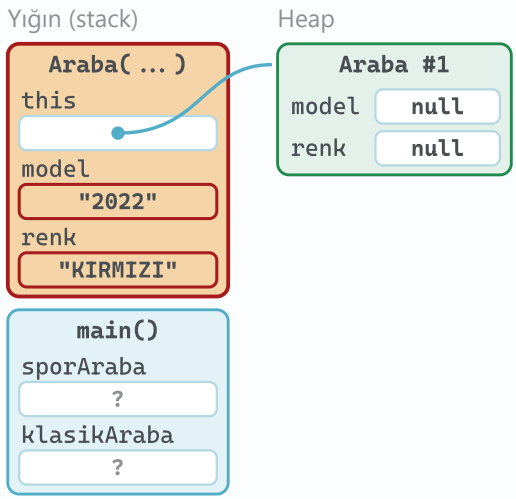
`new` heap'te yeni bir nesne ayırır; `String` alanlar önce varsayılan değer `null` alır.

Adım 1 / 8

ÇIKTI

new Araba(...): Adım Adım

```
1  public class Araba {
2    private String model;
3    private String renk;
4
▶    public Araba(String model, String renk) {
6      this.model = model;
7      this.renk = renk;
8    }
9
10   public void bilgiYazdir() {
11     System.out.println(model + " - " + renk);
12   }
13
14   public static void main(String[] args) {
15     Araba sporAraba = new Araba("2022", "KIRMIZI");
16     Araba klasikAraba = new Araba("2023", "BEYAZ");
17     sporAraba.bilgiYazdir();
18     klasikAraba.bilgiYazdir();
19   }
20 }
```



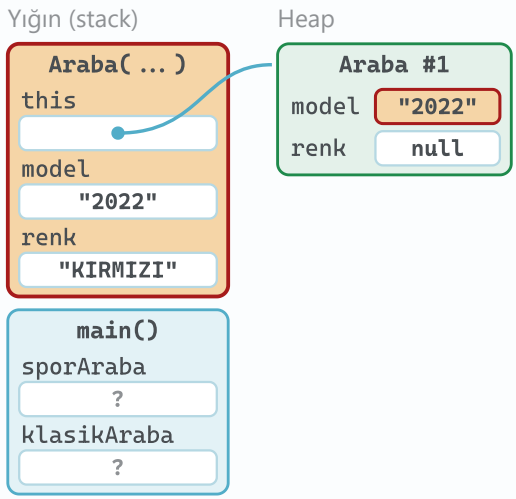
Yapıcı çağrılır: parametreler argümanların kopyasını alır, `this` yeni nesneyi gösterir.

Adım 2 / 8

ÇIKTI

new Araba(...): Adım Adım

```
1  public class Araba {
2    private String model;
3    private String renk;
4
5    public Araba(String model, String renk) {
▶      this.model = model;
7      this.renk = renk;
8    }
9
10   public void bilgiYazdir() {
11     System.out.println(model + " - " + renk);
12   }
13
14   public static void main(String[] args) {
15     Araba sporAraba = new Araba("2022", "KIRMIZI");
16     Araba klasikAraba = new Araba("2023", "BEYAZ");
17     sporAraba.bilgiYazdir();
18     klasikAraba.bilgiYazdir();
19   }
20 }
```



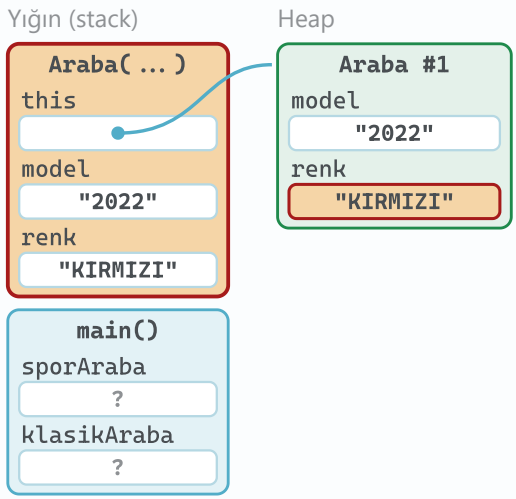
`this.model = model` : soldaki nesnenin alanı, sağdaki parametredir.

Adım 3 / 8

ÇIKTI

new Araba(...): Adım Adım

```
1  public class Araba {
2    private String model;
3    private String renk;
4
5    public Araba(String model, String renk) {
6      this.model = model;
▶      this.renk = renk;
8    }
9
10   public void bilgiYazdir() {
11     System.out.println(model + " - " + renk);
12   }
13
14   public static void main(String[] args) {
15     Araba sporAraba = new Araba("2022", "KIRMIZI");
16     Araba klasikAraba = new Araba("2023", "BEYAZ");
17     sporAraba.bilgiYazdir();
18     klasikAraba.bilgiYazdir();
19   }
20 }
```



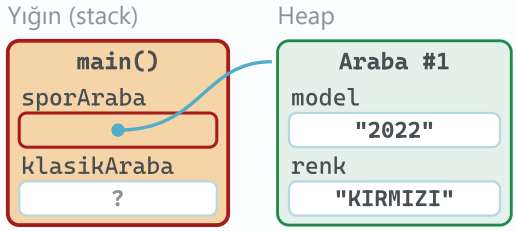
`this.renk = renk` : nesne artık hazır.

Adım 4 / 8

ÇIKTI

new Araba(...): Adım Adım

```
1  public class Araba {
2    private String model;
3    private String renk;
4
5    public Araba(String model, String renk) {
6      this.model = model;
7      this.renk = renk;
8    }
9
10   public void bilgiYazdir() {
11     System.out.println(model + " - " + renk);
12   }
13
14   public static void main(String[] args) {
▶      Araba sporAraba = new Araba("2022", "KIRMIZI");
16     Araba klasikAraba = new Araba("2023", "BEYAZ");
17     sporAraba.bilgiYazdir();
18     klasikAraba.bilgiYazdir();
19   }
20 }
```



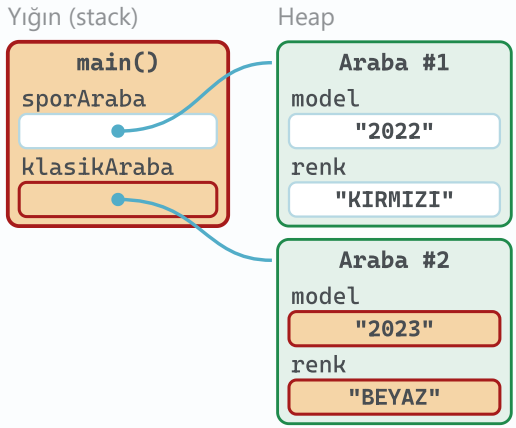
Yapıcı biter, çerçevesi silinir. `new` ifadesinin değeri olan **referans** `sporAraba` 'ya atanır.

Adım 5 / 8

ÇIKTI

new Araba(...): Adım Adım

```
1  public class Araba {
2    private String model;
3    private String renk;
4
5    public Araba(String model, String renk) {
6      this.model = model;
7      this.renk = renk;
8    }
9
10   public void bilgiYazdir() {
11     System.out.println(model + " - " + renk);
12   }
13
14   public static void main(String[] args) {
15     Araba sporAraba = new Araba("2022", "KIRMIZI");
▶      Araba klasikAraba = new Araba("2023", "BEYAZ");
17     sporAraba.bilgiYazdir();
18     klasikAraba.bilgiYazdir();
19   }
20 }
```



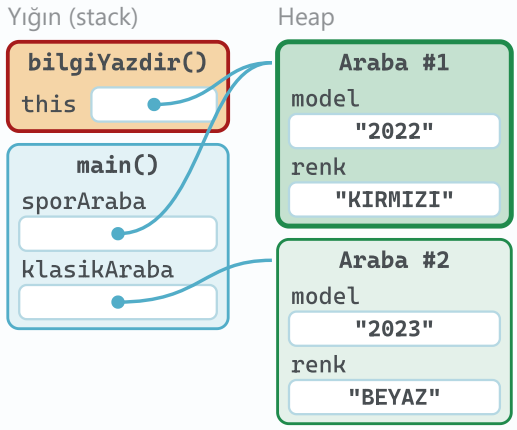
Aynı adımlar ikinci kez: **ayrı** bir nesne oluşur, kendi `model` ve `renk` kopyalarıyla.

Adım 6 / 8

ÇIKTI

new Araba(...): Adım Adım

```
1  public class Araba {
2    private String model;
3    private String renk;
4
5    public Araba(String model, String renk) {
6      this.model = model;
7      this.renk = renk;
8    }
9
10   public void bilgiYazdir() {
▶      System.out.println(model + " - " + renk);
12   }
13
14   public static void main(String[] args) {
15     Araba sporAraba = new Araba("2022", "KIRMIZI");
16     Araba klasikAraba = new Araba("2023", "BEYAZ");
▶      sporAraba.bilgiYazdir();
18     klasikAraba.bilgiYazdir();
19   }
20 }
```



`sporAraba.bilgiYazdir()` : metotta `this` = #1, bu yüzden #1'in alanları yazılır.

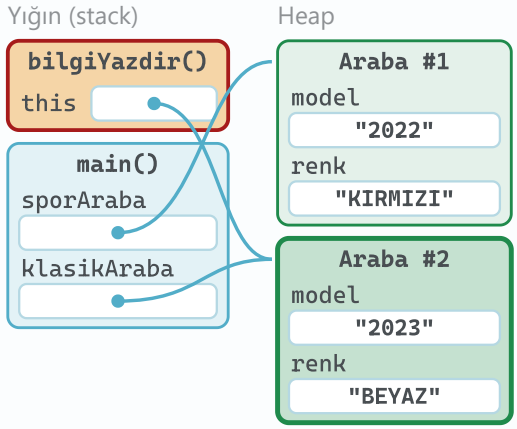
Adım 7 / 8

ÇIKTI

```
2022 - KIRMIZI
```

new Araba(...): Adım Adım

```
1  public class Araba {
2    private String model;
3    private String renk;
4
5    public Araba(String model, String renk) {
6      this.model = model;
7      this.renk = renk;
8    }
9
10   public void bilgiYazdir() {
▶      System.out.println(model + " - " + renk);
12   }
13
14   public static void main(String[] args) {
15     Araba sporAraba = new Araba("2022", "KIRMIZI");
16     Araba klasikAraba = new Araba("2023", "BEYAZ");
17     sporAraba.bilgiYazdir();
▶      klasikAraba.bilgiYazdir();
19   }
20 }
```



Aynı metot, bu kez `this` = #2. Kod tek, veriler nesne başına ayrıdır.

Adım 8 / 8

ÇIKTI

```
2022 - KIRMIZI
2023 - BEYAZ
```

Varsayılan Yapıcı Metot

- Sınıfa hiç yapıcı yazılmamışsa derleyici **varsayılan yapıcıyı** (default constructor) ekler: parametresiz, gövdesi boş. Nesne oluşturulurken her zaman bir yapıcı çağrılır.
- Değer verilmeyen **alanlar** varsayılan değer alır (yerel değişkenler almaz):

Alan tipi	Varsayılan değer
`byte short int long`	`0`
`float double`	`0.0`
`boolean`	`false`
`char`	`'\u0000'` (kod 0)
Referans (`String` , diziler, nesneler)	`null`

```
class Nokta3 {
  int x;
  double agirlik;
  boolean gorunur;
  char harf;
  String ad;
}
public class Varsayilan {
  public static void main(String[] args) {
    Nokta3 n = new Nokta3();   // varsayılan yapıcı
    System.out.println(n.x + " " + n.agirlik);
    System.out.println(n.gorunur + " " + n.ad);
    System.out.println((int) n.harf);
  }
}
```

ÇIKTI

```
0 0.0
false null
0
```

YAPICI METOTLAR

Parametresiz ve Parametreli Yapıcı

```
public class Arac {
  private String marka;
  private String model;
  private int yil;
  public Arac() {                  // parametresiz yapıcı
    marka = "Bilinmiyor";
    model = "Bilinmiyor";
    yil = 0;
  }
  public Arac(String marka, String model, int yil) {
    this.marka = marka;
    this.model = model;
    this.yil = yil;
  }
  public void bilgiYazdir() {
    System.out.println(marka + " " + model + " " + yil);
  }
  public static void main(String[] args) {
    Arac a1 = new Arac();
    Arac a2 = new Arac("Renault", "Clio", 2021);
    a1.bilgiYazdir();
    a2.bilgiYazdir();
  }
}
```

ÇIKTI

```
Bilinmiyor Bilinmiyor 0
Renault Clio 2021
```

- `Arac()`, kendi yazdığımız **parametresiz** yapıcıdır; alanları anlamlı başlangıç değerlerine ayarlar.
- İki yapıcı aynı adı taşır, parametreleri farklıdır: yapıcı **aşırı yükleme**. Hangisinin çalışacağını argümanlar belirler.
- Sınıfa **herhangi bir** yapıcı yazıldığında derleyici artık varsayılan yapıcı eklemez. Yalnızca parametreli yapıcı varsa `new Arac()` derlenmez: `constructor Arac in class Arac cannot be applied to given types`.

ALANLAR

Alanlar ve Türleri

Alanlar, sınıfın içinde (metotların dışında) tanımlanan veri değişkenleridir. İki temel türü vardır:

	Örneğe ait alan (instance field)	**Sınıfa ait alan (static field)**
Tanım	`private String ad;`	`private static int toplamHesapSayisi;`
Kopya sayısı	Her nesnede **ayrı** bir kopya	Sınıfta **tek** kopya, tüm nesneler paylaşır
Ne zaman oluşur?	Her `new` ile	Sınıf belleğe yüklenirken, bir kez
Erişim	`nesne.ad`	`SinifAdi.toplamHesapSayisi`
Örnek	`Ogrenci` : ad, soyad	`BankaHesabi` : toplamHesapSayisi

- **Sabit** (`final`) alanlara değer bir kez verilir, sonra değiştirilemez. Sınıf sabitleri genellikle `static final` ve BÜYÜK_HARF yazılır: `Math.PI` .
- **Sınıfa ait ilklendiriciler** (`static { }`) sınıf yüklenirken bir kez; **örneğe ait ilklendiriciler** (`{ }`) her nesne oluşturulduğunda çalışır.

static Sayaç: Adım Adım

```
1  public class BankaHesabi {
2    private String hesapSahibi;
▶   private static int toplamHesapSayisi = 0;
4
5    public BankaHesabi(String hesapSahibi) {
6      this.hesapSahibi = hesapSahibi;
7      toplamHesapSayisi++;
8    }
9
10   public static int getToplamHesapSayisi() {
11     return toplamHesapSayisi;
12   }
13
14   public static void main(String[] args) {
15     BankaHesabi h1 = new BankaHesabi("Ali");
16     BankaHesabi h2 = new BankaHesabi("Ayşe");
17     BankaHesabi h3 = new BankaHesabi("Can");
18     System.out.println(h2.hesapSahibi);
19     int sayi = BankaHesabi.getToplamHesapSayisi();
20     System.out.println(sayi);
21   }
22 }
```

Yığın (stack) Heap



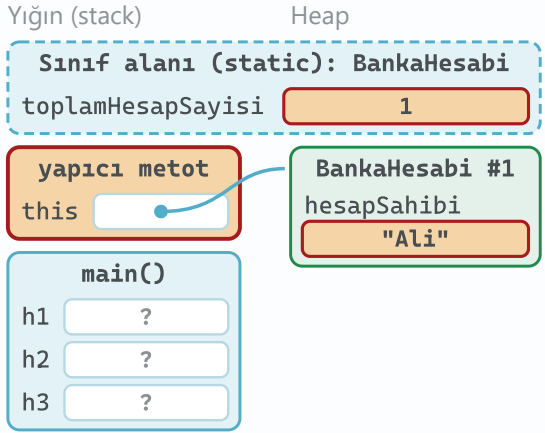
Sınıf belleğe yüklenirken `static` alan **bir kez** oluşur ve 0 olur. Henüz hiç nesne yok.

Adım 1 / 8

ÇIKTI

static Sayaç: Adım Adım

```
1  public class BankaHesabi {
2    private String hesapSahibi;
3    private static int toplamHesapSayisi = 0;
4
▶    public BankaHesabi(String hesapSahibi) {
▶      this.hesapSahibi = hesapSahibi;
▶      toplamHesapSayisi++;
8    }
9
10   public static int getToplamHesapSayisi() {
11     return toplamHesapSayisi;
12   }
13
14   public static void main(String[] args) {
15     BankaHesabi h1 = new BankaHesabi("Ali");
16     BankaHesabi h2 = new BankaHesabi("Ayşe");
17     BankaHesabi h3 = new BankaHesabi("Can");
18     System.out.println(h2.hesapSahibi);
19     int sayi = BankaHesabi.getToplamHesapSayisi();
20     System.out.println(sayi);
21   }
22 }
```



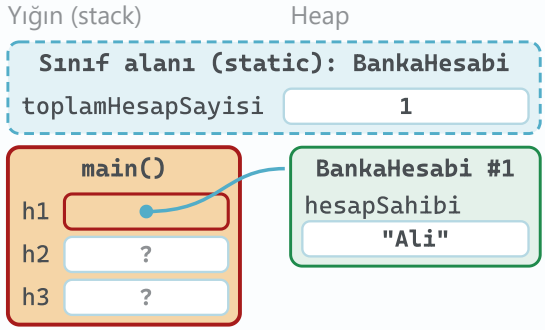
İlk nesne: kendi `hesapSahibi` kopyası "Ali" olur; ortak sayaç 1'e çıkar.

Adım 2 / 8

ÇIKTI

static Sayaç: Adım Adım

```
1  public class BankaHesabi {
2    private String hesapSahibi;
3    private static int toplamHesapSayisi = 0;
4
5    public BankaHesabi(String hesapSahibi) {
6      this.hesapSahibi = hesapSahibi;
7      toplamHesapSayisi++;
8    }
9
10   public static int getToplamHesapSayisi() {
11     return toplamHesapSayisi;
12   }
13
14   public static void main(String[] args) {
▶      BankaHesabi h1 = new BankaHesabi("Ali");
16     BankaHesabi h2 = new BankaHesabi("Ayşe");
17     BankaHesabi h3 = new BankaHesabi("Can");
18     System.out.println(h2.hesapSahibi);
19     int sayi = BankaHesabi.getToplamHesapSayisi();
20     System.out.println(sayi);
21   }
22 }
```



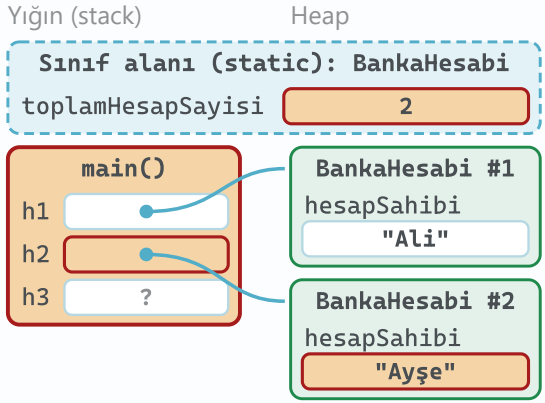
`h1` birinci nesneyi gösterir.

Adım 3 / 8

ÇIKTI

static Sayaç: Adım Adım

```
1  public class BankaHesabi {
2    private String hesapSahibi;
3    private static int toplamHesapSayisi = 0;
4
5    public BankaHesabi(String hesapSahibi) {
6      this.hesapSahibi = hesapSahibi;
7      toplamHesapSayisi++;
8    }
9
10   public static int getToplamHesapSayisi() {
11     return toplamHesapSayisi;
12   }
13
14   public static void main(String[] args) {
15     BankaHesabi h1 = new BankaHesabi("Ali");
▶      BankaHesabi h2 = new BankaHesabi("Ayşe");
17     BankaHesabi h3 = new BankaHesabi("Can");
18     System.out.println(h2.hesapSahibi);
19     int sayi = BankaHesabi.getToplamHesapSayisi();
20     System.out.println(sayi);
21   }
22 }
```



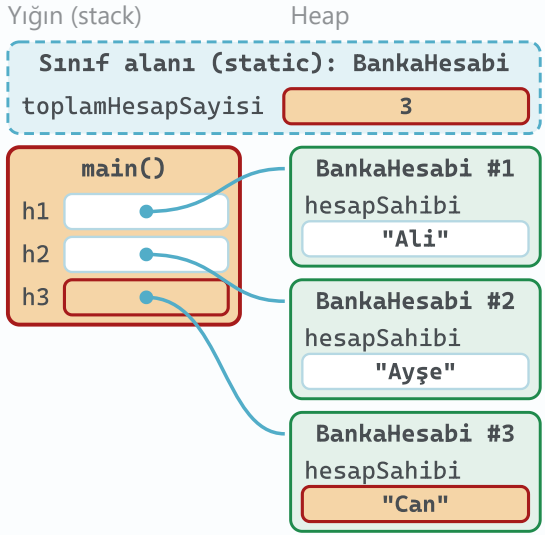
İkinci nesnenin **ayrı** bir `hesapSahibi` alanı var; sayaç ise **aynı** alan: 2.

Adım 4 / 8

ÇIKTI

static Sayaç: Adım Adım

```
1  public class BankaHesabi {
2    private String hesapSahibi;
3    private static int toplamHesapSayisi = 0;
4
5    public BankaHesabi(String hesapSahibi) {
6      this.hesapSahibi = hesapSahibi;
7      toplamHesapSayisi++;
8    }
9
10   public static int getToplamHesapSayisi() {
11     return toplamHesapSayisi;
12   }
13
14   public static void main(String[] args) {
15     BankaHesabi h1 = new BankaHesabi("Ali");
16     BankaHesabi h2 = new BankaHesabi("Ayşe");
▶      BankaHesabi h3 = new BankaHesabi("Can");
18     System.out.println(h2.hesapSahibi);
19     int sayi = BankaHesabi.getToplamHesapSayisi();
20     System.out.println(sayi);
21   }
22 }
```



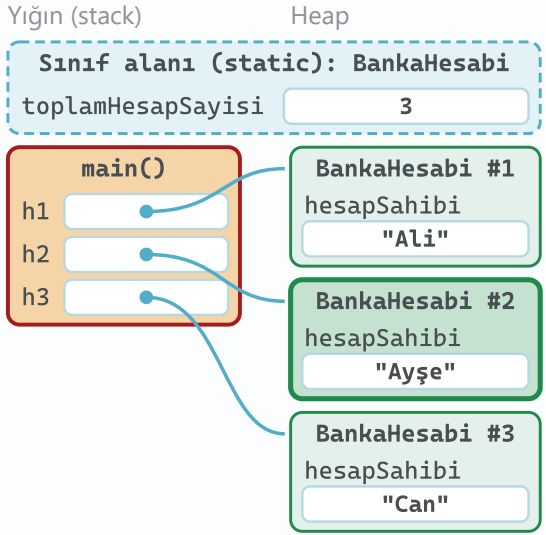
Üç nesne, üç `hesapSahibi` ; ama tek `toplamHesapSayisi` = 3.

Adım 5 / 8

ÇIKTI

static Sayaç: Adım Adım

```
1  public class BankaHesabi {
2    private String hesapSahibi;
3    private static int toplamHesapSayisi = 0;
4
5    public BankaHesabi(String hesapSahibi) {
6      this.hesapSahibi = hesapSahibi;
7      toplamHesapSayisi++;
8    }
9
10   public static int getToplamHesapSayisi() {
11     return toplamHesapSayisi;
12   }
13
14   public static void main(String[] args) {
15     BankaHesabi h1 = new BankaHesabi("Ali");
16     BankaHesabi h2 = new BankaHesabi("Ayşe");
17     BankaHesabi h3 = new BankaHesabi("Can");
▶      System.out.println(h2.hesapSahibi);
19     int sayi = BankaHesabi.getToplamHesapSayisi();
20     System.out.println(sayi);
21   }
22 }
```



`h2.hesapSahibi` : örnek alanı nesne üzerinden okunur.

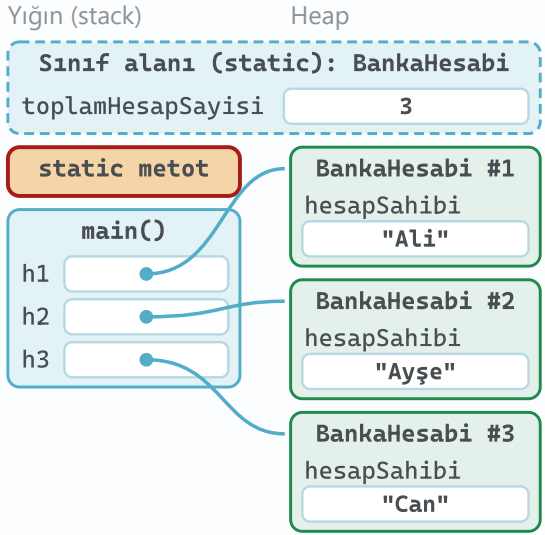
Adım 6 / 8

ÇIKTI

```
Ayşe
```

static Sayaç: Adım Adım

```
1  public class BankaHesabi {
2    private String hesapSahibi;
3    private static int toplamHesapSayisi = 0;
4
5    public BankaHesabi(String hesapSahibi) {
6      this.hesapSahibi = hesapSahibi;
7      toplamHesapSayisi++;
8    }
9
▶    public static int getToplamHesapSayisi() {
▶      return toplamHesapSayisi;
12   }
13
14   public static void main(String[] args) {
15     BankaHesabi h1 = new BankaHesabi("Ali");
16     BankaHesabi h2 = new BankaHesabi("Ayşe");
17     BankaHesabi h3 = new BankaHesabi("Can");
18     System.out.println(h2.hesapSahibi);
▶      int sayi = BankaHesabi.getToplamHesapSayisi();
20     System.out.println(sayi);
21   }
22 }
```



`static` metot **sınıf adıyla** çağrılır; `this` 'i yoktur, yalnızca static alanı okur.

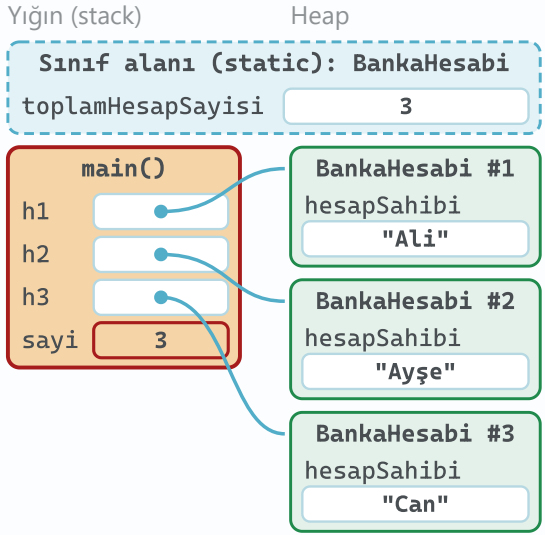
Adım 7 / 8

ÇIKTI

```
Ayşe
```

static Sayaç: Adım Adım

```
1  public class BankaHesabi {
2    private String hesapSahibi;
3    private static int toplamHesapSayisi = 0;
4
5    public BankaHesabi(String hesapSahibi) {
6      this.hesapSahibi = hesapSahibi;
7      toplamHesapSayisi++;
8    }
9
10   public static int getToplamHesapSayisi() {
11     return toplamHesapSayisi;
12   }
13
14   public static void main(String[] args) {
15     BankaHesabi h1 = new BankaHesabi("Ali");
16     BankaHesabi h2 = new BankaHesabi("Ayşe");
17     BankaHesabi h3 = new BankaHesabi("Can");
18     System.out.println(h2.hesapSahibi);
19     int sayi = BankaHesabi.getToplamHesapSayisi();
▶      System.out.println(sayi);
21   }
22 }
```



Dönen 3, `sayi` 'ya atanır ve yazdırılır.

Adım 8 / 8

ÇIKTI

```
Ayşe
3
```

ALANLAR

Alan Tanımlamaları

```java
public class Kitap {
  // Örneğe ait alanlar
  private String baslik;
  private String yazar;
  private int sayfaSayisi;
  // Sınıfa ait alan
  private static int toplamKitapSayisi;
  // final tanımlanmış bir alan (sabit)
  public static final double PI = 3.14159265359;
}
```

```java
public class Alanlar {
  public int a;                          // genel
  int b = 1, c = 2;                      // varsayılan (paket)
  private double x;                      // özel
  private static int xx;                 // statik, özel
  public static int y;                   // statik, genel
  public final int CONST1 = 5;           // değiştirilemez
  private static final int CONST2 = 6;   // statik sabit
}
```

- Söz dizimi: `[erişim] [static] [final] tür ad [= ilkDeğer];`
- Aynı satırda birden çok alan: `int b = 1, c = 2;`
- `PI` burada örnek olarak tanımlandı; Java'da hazır bir sabit vardır: `Math.PI`.
- `final` alana yeniden değer atamak derleme hatasıdır: `error: cannot assign a value to final variable CONST1`

ALANLAR

İlklendirici Blokları

Sınıfa ait ilklendirici (static initializer)

`static { ... }` bloğu, sınıf belleğe **ilk kez yüklenirken**, yalnızca **bir kez** çalışır. Static alanlara karmaşık ilk değer vermek için kullanılır.

Örneğe ait ilklendirici (instance initializer)

`{ ... }` bloğu, **her nesne oluşturulduğunda**, yapıcının gövdesinden **önce** çalışır. Tüm yapıcılarda ortak olan işleri yapar.

Bir nesne oluşturulurken sıra: **(1)** sınıf ilk kez kullanılıyorsa static alanlar ve `static` bloklar → **(2)** `new` ile bellek ayrılır, alanlar varsayılan değer alır → **(3)** alan ilk değerleri ve örnek ilklendirici blokları (yazılış sırasıyla) → **(4)** yapıcının gövdesi.

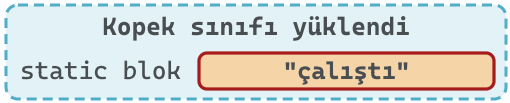
`Kopek` sınıfında her ikisi de var. Not: `havla()` 'nın döndürdüğü değer yazdırılmıyor; `System.out.println(d2.havla());` yazılsaydı son satır "Gece haw haw haw!" olurdu.



Kopek: İlklendirici Sırası

```
1  public class Kopek {
2    String adi;                  // alan
3    Kopek(String adi) {          // yapıcı metot
4      this.adi = adi;
5      System.out.println(adi);
6    }
7    String havla() {             // örneğe ait metot
8      return adi + " haw haw haw!";
9    }
10   static {                     // sınıfa ait ilklendirici
▶      System.out.println("Köpekler!");
12   }
13   {                            // örneğe ait ilklendirici
14     System.out.println("Köpek oluşturuluyor.");
15   }
16   public static void main(String... strings) {
17     var d1 = new Kopek("Kara");
18     var d2 = new Kopek("Gece");
19     d2.havla();
20   }
21 }
```

Yığın (stack) Heap



`main` çalışmadan önce sınıf yüklenir: `static` blok **bir kez** çalışır. (`var`, Java 10+: tipi sağdan çıkarır; `String...` = `String[]`.)

Adım 1 / 7

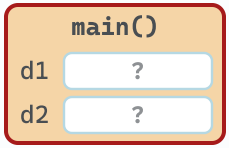
ÇIKTI

```
Köpekler!
```

Kopek: İlklendirici Sırası

```
1  public class Kopek {
2    String adi;                  // alan
3    Kopek(String adi) {          // yapıcı metot
4      this.adi = adi;
5      System.out.println(adi);
6    }
7    String havla() {             // örneğe ait metot
8      return adi + " haw haw haw!";
9    }
10   static {                     // sınıfa ait ilklendirici
11     System.out.println("Köpekler!");
12   }
13   {                            // örneğe ait ilklendirici
14     System.out.println("Köpek oluşturuluyor.");
15   }
16   public static void main(String... strings) {
▶      var d1 = new Kopek("Kara");
18     var d2 = new Kopek("Gece");
19     d2.havla();
20   }
21 }
```

Yığın (stack)



Heap



`new Kopek("Kara")` : nesne için yer ayrılır, `adi` = null.

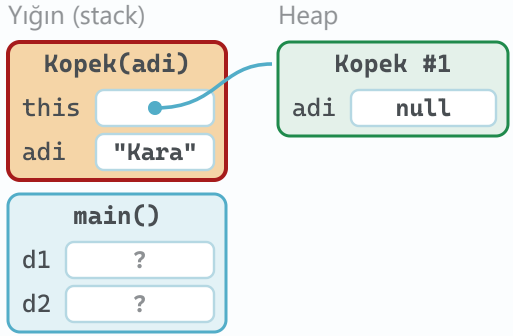
Adım 2 / 7

ÇIKTI

```
Köpekler!
```

Kopek: İlklendirici Sırası

```
1  public class Kopek {
2    String adi;                  // alan
3    Kopek(String adi) {          // yapıcı metot
4      this.adi = adi;
5      System.out.println(adi);
6    }
7    String havla() {             // örneğe ait metot
8      return adi + " haw haw haw!";
9    }
10   static {                     // sınıfa ait ilklendirici
11     System.out.println("Köpekler!");
12   }
13   {                            // örneğe ait ilklendirici
▶      System.out.println("Köpek oluşturuluyor.");
15   }
16   public static void main(String... strings) {
17     var d1 = new Kopek("Kara");
18     var d2 = new Kopek("Gece");
19     d2.havla();
20   }
21 }
```



Yapıcının gövdesinden **önce** örnek ilklendirici bloğu çalışır.

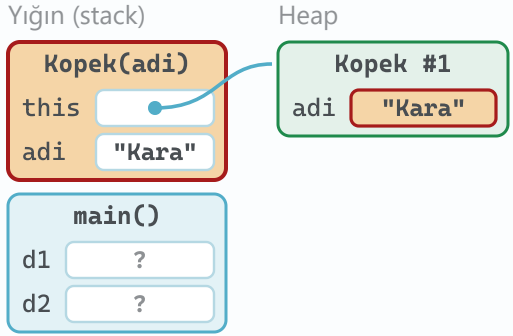
Adım 3 / 7

ÇIKTI

```
Köpekler!
Köpek oluşturuluyor.
```

Kopek: İlklendirici Sırası

```
1  public class Kopek {
2    String adi;                  // alan
3    Kopek(String adi) {          // yapıcı metot
4      this.adi = adi;
5      System.out.println(adi);
6    }
7    String havla() {             // örneğe ait metot
8      return adi + " haw haw haw!";
9    }
10   static {                     // sınıfa ait ilklendirici
11     System.out.println("Köpekler!");
12   }
13   {                            // örneğe ait ilklendirici
14     System.out.println("Köpek oluşturuluyor.");
15   }
16   public static void main(String... strings) {
17     var d1 = new Kopek("Kara");
18     var d2 = new Kopek("Gece");
19     d2.havla();
20   }
21 }
```



Şimdi yapıcı gövdesi: `this.adi = adi` .

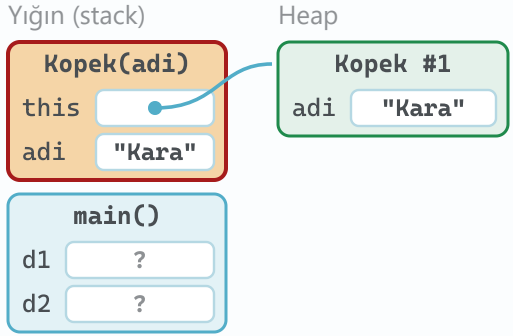
Adım 4 / 7

ÇIKTI

```
Köpekler!
Köpek oluşturuluyor.
```

Kopek: İlklendirici Sırası

```
1  public class Kopek {
2    String adi;                  // alan
3    Kopek(String adi) {          // yapıcı metot
4      this.adi = adi;
▶      System.out.println(adi);
6    }
7    String havla() {             // örneğe ait metot
8      return adi + " haw haw haw!";
9    }
10   static {                     // sınıfa ait ilklendirici
11     System.out.println("Köpekler!");
12   }
13   {                            // örneğe ait ilklendirici
14     System.out.println("Köpek oluşturuluyor.");
15   }
16   public static void main(String... strings) {
17     var d1 = new Kopek("Kara");
18     var d2 = new Kopek("Gece");
19     d2.havla();
20   }
21 }
```



Yapıcı adı yazdırır: **Kara**.

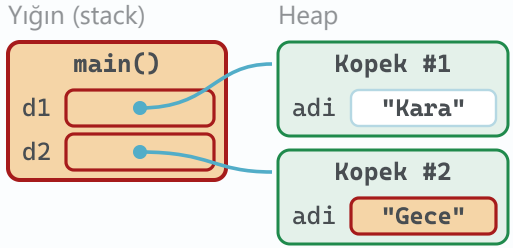
Adım 5 / 7

ÇIKTI

```
Köpekler!
Köpek oluşturuluyor.
Kara
```

Kopek: İlklendirici Sırası

```
1  public class Kopek {
2    String adi;                  // alan
3    Kopek(String adi) {          // yapıcı metot
4      this.adi = adi;
5      System.out.println(adi);
6    }
7    String havla() {             // örneğe ait metot
8      return adi + " haw haw haw!";
9    }
10   static {                     // sınıfa ait ilklendirici
11     System.out.println("Köpekler!");
12   }
13   {                            // örneğe ait ilklendirici
14     System.out.println("Köpek oluşturuluyor.");
15   }
16   public static void main(String... strings) {
17     var d1 = new Kopek("Kara");
▶      var d2 = new Kopek("Gece");
19     d2.havla();
20   }
21 }
```



İkinci nesne: ilklendirici + yapıcı yeniden; `static` blok **tekrar çalışmaz**.

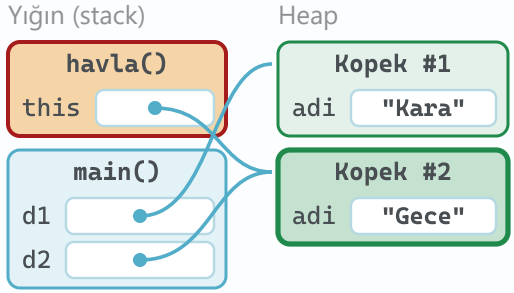
Adım 6 / 7

ÇIKTI

```
Köpekler!
Köpek oluşturuluyor.
Kara
Köpek oluşturuluyor.
Gece
```

Kopek: İlklendirici Sırası

```
1  public class Kopek {
2    String adi;                  // alan
3    Kopek(String adi) {          // yapıcı metot
4      this.adi = adi;
5      System.out.println(adi);
6    }
7    String havla() {             // örneğe ait metot
▶      return adi + " haw haw haw!";
9    }
10   static {                     // sınıfa ait ilklendirici
11     System.out.println("Köpekler!");
12   }
13   {                            // örneğe ait ilklendirici
14     System.out.println("Köpek oluşturuluyor.");
15   }
16   public static void main(String... strings) {
17     var d1 = new Kopek("Kara");
18     var d2 = new Kopek("Gece");
▶      d2.havla();
20   }
21 }
```



`havla()` bir metin döndürür ama değer **kullanılmıyor**: hiçbir şey yazılmaz.

Adım 7 / 7

ÇIKTI

```
Köpekler!
Köpek oluşturuluyor.
Kara
Köpek oluşturuluyor.
Gece
```

Metot Türleri

- **Metotlar**, sınıf içinde tanımlanan, belirli bir işi yapan kod bloklarıdır; sınıfın **davranışını** tanımlar.
- **Örneğe ait metotlar** (instance methods) bir **nesneyle** ilişkilidir: `telefon.aramaYap(...)`. O nesnenin alanlarını kullanır.
- **Sınıfa ait metotlar** (`static`) **sınıfın kendisiyle** ilişkilidir, tüm nesnelerde ortaktır: `CepTelefonu` üzerinden.
- Static metotlar belirli bir nesneye bağlı olmadığı için nesneye özgü verilere (`model` gibi) **doğrudan erişemez**.

```
public class CepTelefonu {
  private String model;
  private static String isletimSistemi = "Sürüm 1";

  // Örneğe ait metot
  public void aramaYap(String numara) {
    System.out.println(model + " arıyor: " + numara);
  }

  // Örneğe ait metot
  public void mesajGonder(String alici, String mesaj) {
    System.out.println(alici + " <- " + mesaj);
  }

  // Sınıfa ait metot (static)
  public static void sistemiGuncelle(String yeniSurum) {
    isletimSistemi = yeniSurum;
  }
}
```

Metot Örneği: Eposta

```
public class Eposta {
  public boolean gonder(String alici, String konu,
              String icerik) {
    // Gerçek gönderim yerine bir benzetim
    if (alici == null || !alici.contains("@")) {
      return false;                // başarısız
    }
    System.out.println(alici + " <- " + konu);
    return true;                   // başarılı
  }

  public static void main(String[] args) {
    Eposta e = new Eposta();
    boolean s1 = e.gonder("ali@ornek.com", "Ödev", "Ek");
    boolean s2 = e.gonder("ali.ornek.com", "Ödev", "Ek");
    System.out.println(s1 + " " + s2);
  }
}
```

gonder metodu

- Ad: `gonder`
- Parametreler: üç `String`: alıcı, konu, içerik
- Dönüş tipi: `boolean`, gönderim başarılı mı?

ÇIKTI

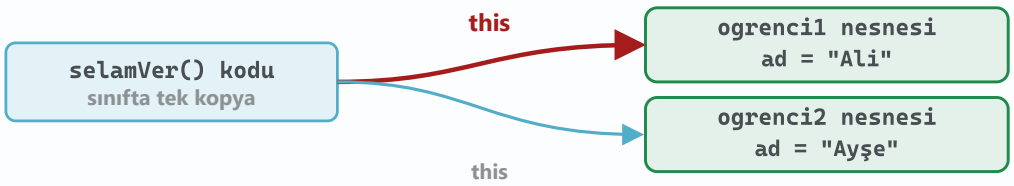
```
ali@ornek.com <- Ödev
true false
```

`boolean` döndüren metodun **her yolu** bir `return` ile bitmelidir; gövdesi yalnızca yorum olan bir metot `missing return statement` hatası verir.

Örneğe Ait Alanlar ve Metotlar

- Sınıf içinde `static` anahtar sözcüğü **olmadan** tanımlanırlar.
- Her nesne, alanların **kendi kopyasına** sahiptir; nesneler farklı verilere sahip olabilir.
- Alanlar nesnenin **özelliklerini**, metotlar o nesnenin **davranışını** temsil eder.
- Kullanabilmek için önce **nesne oluşturmak** gerekir: `new`, sonra `nesne.metot()`.
- Metot **kodu** tüm nesneler için tektir; her çağrıda hangi nesnenin verisiyle çalışacağını `this` belirler.

	Örneğe ait	**`static`**
Kime ait?	Nesneye	Sınıfa
Çağrı	`nesne.metot()`	`Sinif.metot()`
Nesne gerekir mi?	Evet	Hayır
`this` var mı?	Evet	Hayır
Örnek	`str.length()`	`Math.sqrt(x)`



METOTLAR

Nokta İşleci

```java
public class Telefon {
  String numara;
  public Telefon(String numara) {
    this.numara = numara;
  }
  public void aramaYap(Telefon telefon) {
    System.out.println(numara + " arıyor: " + telefon.numara);
  }
  public void mesajGonder(Telefon telefon, String mesaj) {
    System.out.println(telefon.numara + " <- " + mesaj);
  }
  public void kameraAc() {
    System.out.println(numara + " kamera açtı.");
  }
  public static void main(String[] args) {
    Telefon telefon1 = new Telefon("5551234567");
    Telefon telefon2 = new Telefon("5051234567");
    telefon1.aramaYap(telefon2);
    telefon2.mesajGonder(telefon1, "Merhaba!");
    telefon1.kameraAc();
  }
}
```

ÇIKTI

```
5551234567 arıyor: 5051234567
5551234567 <- Merhaba!
5551234567 kamera açtı.
```

- Nesne oluşturulduktan sonra alan ve metotlarına **nokta işleci** ile erişilir: `nesne.metot(...)`, `nesne.alan`.
- `telefon1.aramaYap(...)`: **telefon1** arar; metotta `numara` telefon1'in, `telefon.numara` telefon2'nin numarasıdır.
- Bir nesne başka bir nesneyi **parametre** olarak alabilir.

METOTLAR

Dikdörtgen Sınıfı

```
public class Dikdortgen {
  private double uzunluk;     // dikdörtgenin uzunluğu
  private double genislik;    // dikdörtgenin genişliği

  // Yapıcı metot
  public Dikdortgen(double uzunluk, double genislik) {
    this.uzunluk = uzunluk;
    this.genislik = genislik;
  }

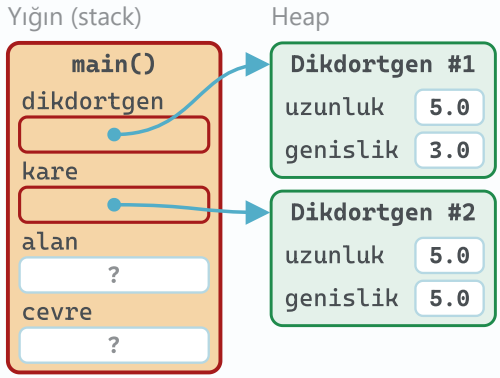
  public double alanHesapla() {      // alanı hesaplar
    return uzunluk * genislik;
  }

  public double cevreHesapla() {     // çevreyi hesaplar
    return 2 * (uzunluk + genislik);
  }
}
```

- Alanlar `private`: dışarıdan yalnızca metotlarla kullanılır.
- Yapıcı, iki kenarı alır ve nesneye kaydeder.
- `alanHesapla()` ve `cevreHesapla()` parametre almaz: ihtiyaç duydukları veriler **nesnenin içinde**.
- Prosedürel karşılığı `alanHesapla(5.0, 3.0)` olurdu; nesneyle `dikdortgen.alanHesapla()`.

Dikdörtgen: Adım Adım

```
1  public class DikdortgenTesti {
2    public static void main(String[] args) {
▶      Dikdortgen dikdortgen = new Dikdortgen(5.0, 3.0);
▶      Dikdortgen kare = new Dikdortgen(5.0, 5.0);
5      double alan = dikdortgen.alanHesapla();
6      double cevre = dikdortgen.cevreHesapla();
7      System.out.println("Alanı: " + alan
8          + " Çevresi: " + cevre);
9      alan = kare.alanHesapla();
10     cevre = kare.cevreHesapla();
11     System.out.println("Alanı: " + alan
12         + " Çevresi: " + cevre);
13   }
14 }
```



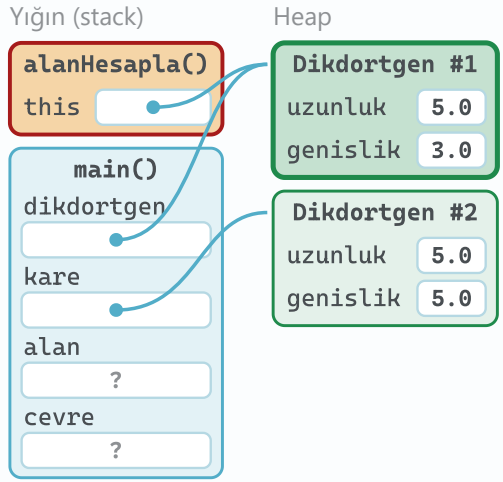
İki nesne, her biri kendi `uzunluk` ve `genislik` değerleriyle.

Adım 1 / 6

ÇIKTI

Dikdörtgen: Adım Adım

```
1  public class DikdortgenTesti {
2    public static void main(String[] args) {
3      Dikdortgen dikdortgen = new Dikdortgen(5.0, 3.0);
4      Dikdortgen kare = new Dikdortgen(5.0, 5.0);
▶      double alan = dikdortgen.alanHesapla();
6      double cevre = dikdortgen.cevreHesapla();
7      System.out.println("Alanı: " + alan
8          + " Çevresi: " + cevre);
9      alan = kare.alanHesapla();
10     cevre = kare.cevreHesapla();
11     System.out.println("Alanı: " + alan
12         + " Çevresi: " + cevre);
13   }
14 }
```



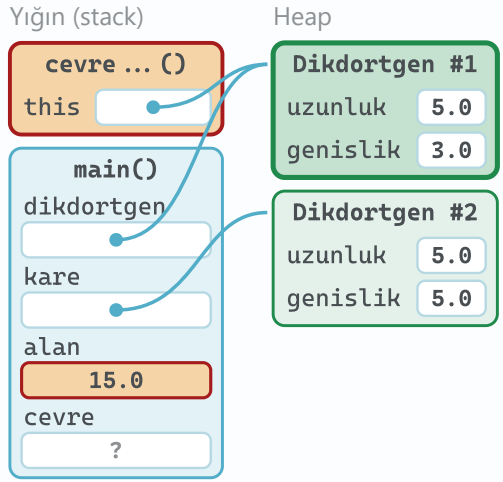
`dikdortgen.alanHesapla()` : `this` = #1, sonuç 5.0 × 3.0 = 15.0.

Adım 2 / 6

ÇIKTI

Dikdörtgen: Adım Adım

```
1  public class DikdortgenTesti {
2    public static void main(String[] args) {
3      Dikdortgen dikdortgen = new Dikdortgen(5.0, 3.0);
4      Dikdortgen kare = new Dikdortgen(5.0, 5.0);
5      double alan = dikdortgen.alanHesapla();
▶      double cevre = dikdortgen.cevreHesapla();
7      System.out.println("Alanı: " + alan
8          + " Çevresi: " + cevre);
9      alan = kare.alanHesapla();
10     cevre = kare.cevreHesapla();
11     System.out.println("Alanı: " + alan
12         + " Çevresi: " + cevre);
13   }
14 }
```



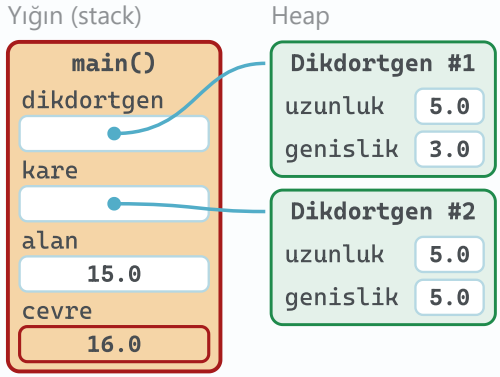
`alan` = 15.0. `cevreHesapla()` : 2 × (5.0 + 3.0) = 16.0.

Adım 3 / 6

ÇIKTI

Dikdörtgen: Adım Adım

```
1  public class DikdortgenTesti {
2    public static void main(String[] args) {
3      Dikdortgen dikdortgen = new Dikdortgen(5.0, 3.0);
4      Dikdortgen kare = new Dikdortgen(5.0, 5.0);
5      double alan = dikdortgen.alanHesapla();
6      double cevre = dikdortgen.cevreHesapla();
▶      System.out.println("Alanı: " + alan
▶          + " Çevresi: " + cevre);
9      alan = kare.alanHesapla();
10     cevre = kare.cevreHesapla();
11     System.out.println("Alanı: " + alan
12         + " Çevresi: " + cevre);
13   }
14 }
```



`double` değerler `15.0` biçiminde yazılır.

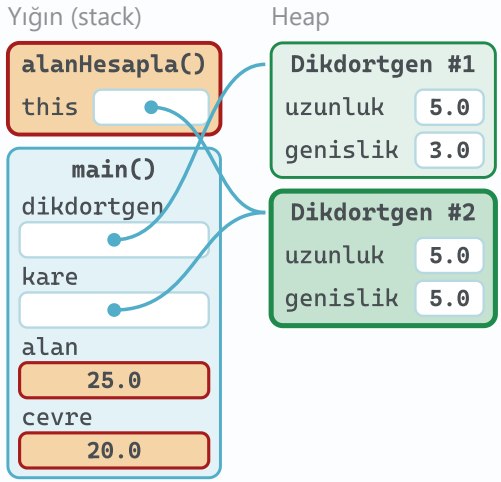
Adım 4 / 6

ÇIKTI

```
Alanı: 15.0 Çevresi: 16.0
```

Dikdörtgen: Adım Adım

```
1  public class DikdortgenTesti {
2    public static void main(String[] args) {
3      Dikdortgen dikdortgen = new Dikdortgen(5.0, 3.0);
4      Dikdortgen kare = new Dikdortgen(5.0, 5.0);
5      double alan = dikdortgen.alanHesapla();
6      double cevre = dikdortgen.cevreHesapla();
7      System.out.println("Alanı: " + alan
8          + " Çevresi: " + cevre);
▶      alan = kare.alanHesapla();
▶      cevre = kare.cevreHesapla();
11     System.out.println("Alanı: " + alan
12         + " Çevresi: " + cevre);
13   }
14 }
```



Aynı metotlar, bu kez `this` = #2 (kare): 25.0 ve 20.0. Eski değerlerin üzerine yazılır.

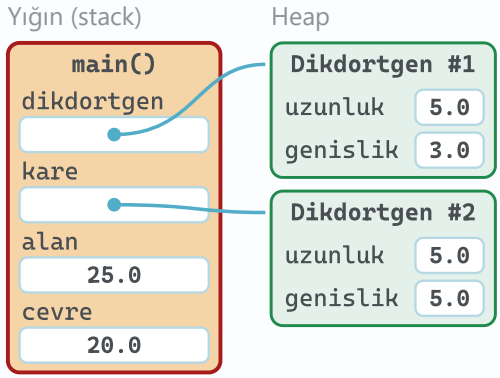
Adım 5 / 6

ÇIKTI

```
Alanı: 15.0 Çevresi: 16.0
```

Dikdörtgen: Adım Adım

```
1  public class DikdortgenTesti {
2    public static void main(String[] args) {
3      Dikdortgen dikdortgen = new Dikdortgen(5.0, 3.0);
4      Dikdortgen kare = new Dikdortgen(5.0, 5.0);
5      double alan = dikdortgen.alanHesapla();
6      double cevre = dikdortgen.cevreHesapla();
7      System.out.println("Alanı: " + alan
8          + " Çevresi: " + cevre);
9      alan = kare.alanHesapla();
10     cevre = kare.cevreHesapla();
▶      System.out.println("Alanı: " + alan
▶          + " Çevresi: " + cevre);
13   }
14 }
```



İkinci satır yazdırılır. Uzun bir deyim iki satıra bölünebilir; `;` gelene kadar sürer.

Adım 6 / 6

ÇIKTI

```
Alanı: 15.0 Çevresi: 16.0
Alanı: 25.0 Çevresi: 20.0
```

STATIC ÜYELER

static Değişkenler ve Metotlar

```java
public class Matematik {
  // Statik toplama metodu
  public static int toplama(int sayi1, int sayi2) {
    return sayi1 + sayi2;
  }

  public static void main(String[] args) {
    int sonuc = Matematik.toplama(5, 3);
    System.out.println("Toplama Sonucu: " + sonuc);
    System.out.println(Math.max(4, 9));
    System.out.println(Math.sqrt(16.0));
  }
}
```

- `static` anahtar sözcüğüyle tanımlanır; **sınıfa** aittir.
- static üyelere erişmek için **nesne oluşturmaya gerek yoktur**; sınıf adı ve nokta kullanılır: `SinifAdi.alanAdi`, `SinifAdi.metotAdi(argumanlar)`.
- Nesne üzerinden de erişilebilir (`nesne.alanAdi`), ama **tercih edilmez**: okuyan kişi üyenin nesneye ait olduğunu sanır.
- Hazır örnekler: `Math.max()`, `Math.sqrt()`, `Math.PI`, `Integer.parseInt()` ve `main`.

ÇIKTI

```
Toplama Sonucu: 8
9
4.0
```

static Kuralları

Nereden →	Örnek alan/metot kullanımı	static alan/metot kullanımı
Örneğe ait metottan	✓ doğrudan (`ad` , `this.ad`)	✓ doğrudan
`static` metottan (`main` dahil)	✗ ancak bir nesne üzerinden (`ogr.ad`)	✓ doğrudan

```java
public class Selam {
  private String ad = "Ali";
  public void selamVer() {
    System.out.println("Merhaba " + ad);
  }
  public static void main(String[] args) {
    selamVer();
  }
}
```

```
error: non-static method selamVer() cannot be
referenced from a static context
```

```java
public class Selam2 {
  private String ad = "Ali";
  public static void yaz() {
    System.out.println(ad);
  }
}
```

```
error: non-static variable ad cannot be referenced
from a static context
```

Çözüm: çağrı nesne üzerinden yapılır (`new Selam().selamVer();`) ya da metot `static` yapılır.

Referans Atama

- Java'da nesneler **referanslar** (adresler) aracılığıyla işlenir.
- `araba2 = araba1;` nesneyi **kopyalamaz**, referansı kopyalar: iki değişken **aynı nesneyi** işaret eder (takma ad, alias).
- Biri üzerinden yapılan değişiklik diğerinden de görünür.

Kütüphane etiketi

Raftaki etiket kitabın kendisi değil, **yerini** gösterir.

Araba anahtarı

Anahtar arabayı işaret eder; iki anahtar aynı arabayı açar.

```
public class Araba {
  private String marka;

  public Araba(String marka) {
    this.marka = marka;
  }

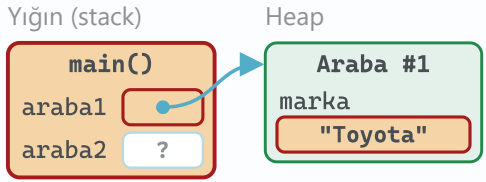
  public String getMarka() {
    return marka;
  }

  public void setMarka(String marka) {
    this.marka = marka;
  }

}
```

Referans Atama: Adım Adım

```
1  public static void main(String[] args) {
2    Araba araba1 = new Araba("Toyota");
3    Araba araba2 = araba1;
4    System.out.println("Araba 1: " + araba1.getMarka());
5    System.out.println("Araba 2: " + araba2.getMarka());
6    araba2.setMarka("Ford");
7    System.out.println("Araba 1: " + araba1.getMarka());
8    araba1 = null;
9    System.out.println("Araba 2: " + araba2.getMarka());
10 }
```



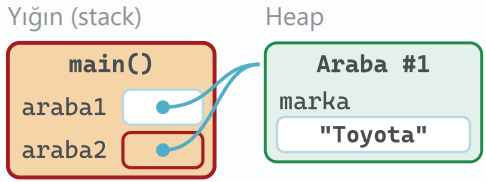
Bir nesne oluşturulur; `araba1` onu gösterir.

Adım 1 / 7

ÇIKTI

Referans Atama: Adım Adım

```
1  public static void main(String[] args) {
2    Araba araba1 = new Araba("Toyota");
▶    Araba araba2 = araba1;
4    System.out.println("Araba 1: " + araba1.getMarka());
5    System.out.println("Araba 2: " + araba2.getMarka());
6    araba2.setMarka("Ford");
7    System.out.println("Araba 1: " + araba1.getMarka());
8    araba1 = null;
9    System.out.println("Araba 2: " + araba2.getMarka());
10 }
```



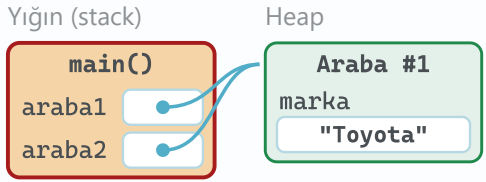
`araba2 = araba1` : referans kopyalanır. Hâlâ **tek** nesne var.

Adım 2 / 7

ÇIKTI

Referans Atama: Adım Adım

```
1  public static void main(String[] args) {
2    Araba araba1 = new Araba("Toyota");
3    Araba araba2 = araba1;
▶    System.out.println("Araba 1: " + araba1.getMarka());
▶    System.out.println("Araba 2: " + araba2.getMarka());
6    araba2.setMarka("Ford");
7    System.out.println("Araba 1: " + araba1.getMarka());
8    araba1 = null;
9    System.out.println("Araba 2: " + araba2.getMarka());
10 }
```



İki değişken aynı nesneyi okur: ikisi de Toyota.

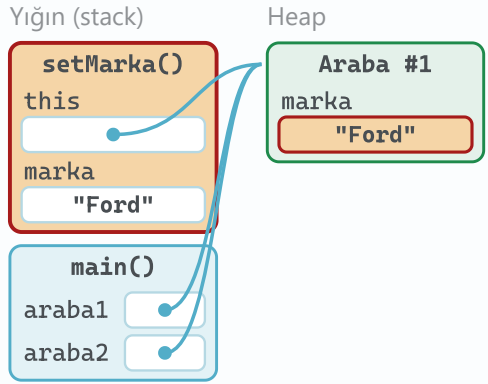
Adım 3 / 7

ÇIKTI

```
Araba 1: Toyota
Araba 2: Toyota
```

Referans Atama: Adım Adım

```
1  public static void main(String[] args) {
2    Araba araba1 = new Araba("Toyota");
3    Araba araba2 = araba1;
4    System.out.println("Araba 1: " + araba1.getMarka());
5    System.out.println("Araba 2: " + araba2.getMarka());
▶    araba2.setMarka("Ford");
7    System.out.println("Araba 1: " + araba1.getMarka());
8    araba1 = null;
9    System.out.println("Araba 2: " + araba2.getMarka());
10 }
```



`araba2.setMarka("Ford")` ortak nesneyi değiştirir.

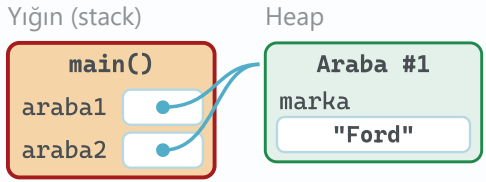
Adım 4 / 7

ÇIKTI

```
Araba 1: Toyota
Araba 2: Toyota
```

Referans Atama: Adım Adım

```
1  public static void main(String[] args) {
2    Araba araba1 = new Araba("Toyota");
3    Araba araba2 = araba1;
4    System.out.println("Araba 1: " + araba1.getMarka());
5    System.out.println("Araba 2: " + araba2.getMarka());
6    araba2.setMarka("Ford");
▶    System.out.println("Araba 1: " + araba1.getMarka());
8    araba1 = null;
9    System.out.println("Araba 2: " + araba2.getMarka());
10 }
```



`araba1` üzerinden okuyunca da **Ford**: değişiklik paylaşılır.

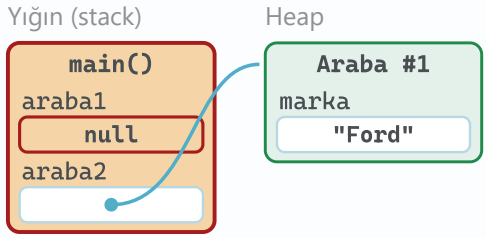
Adım 5 / 7

ÇIKTI

```
Araba 1: Toyota
Araba 2: Toyota
Araba 1: Ford
```

Referans Atama: Adım Adım

```
1  public static void main(String[] args) {
2    Araba araba1 = new Araba("Toyota");
3    Araba araba2 = araba1;
4    System.out.println("Araba 1: " + araba1.getMarka());
5    System.out.println("Araba 2: " + araba2.getMarka());
6    araba2.setMarka("Ford");
7    System.out.println("Araba 1: " + araba1.getMarka());
▶    araba1 = null;
9    System.out.println("Araba 2: " + araba2.getMarka());
10 }
```



`araba1 = null` : yalnızca bu değişkenin bağı kopar; nesne silinmez, `araba2` onu göstermeye devam eder.

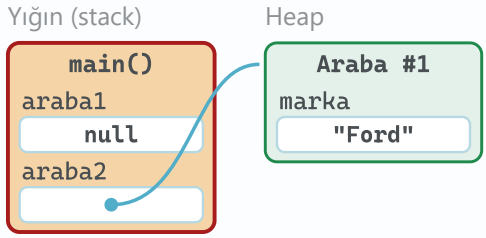
Adım 6 / 7

ÇIKTI

```
Araba 1: Toyota
Araba 2: Toyota
Araba 1: Ford
```

Referans Atama: Adım Adım

```
1  public static void main(String[] args) {
2    Araba araba1 = new Araba("Toyota");
3    Araba araba2 = araba1;
4    System.out.println("Araba 1: " + araba1.getMarka());
5    System.out.println("Araba 2: " + araba2.getMarka());
6    araba2.setMarka("Ford");
7    System.out.println("Araba 1: " + araba1.getMarka());
8    araba1 = null;
▶    System.out.println("Araba 2: " + araba2.getMarka());
10 }
```



Nesneyi gösteren son referans da kaybolursa nesne **çöp toplayıcı** (garbage collector) tarafından silinir.

Adım 7 / 7

ÇIKTI

```
Araba 1: Toyota
Araba 2: Toyota
Araba 1: Ford
Araba 2: Ford
```

null ve NullPointerException

```
public class Bos {
  String ad;

  public static void main(String[] args) {
    Bos b = null;
    System.out.println(b.ad.length());
  }
}
```

```
Exception in thread "main"
java.lang.NullPointerException: Cannot read
field "ad" because "<local1>" is null
    at Bos.main(Bos.java:6)
```

Yerel değişken hiç değer verilmeden kullanılırsa program **derlenmez**: `error: variable sb might not have been initialized`

- `null`, referans değişkeninin **hiçbir nesneyi göstermediği** anlamına gelir.
- `null` bir referans üzerinden alan okumak ya da metot çağırmak çalışma zamanında `NullPointerException` fırlatır.
- Önlem: kullanmadan önce denetim: `if (b != null) { ... }`
- Hiçbir referansın göstermediği nesneler JVM'in **çöp toplayıcısı** tarafından otomatik silinir; Java'da nesneyi elle silmek gerekmez.



this Anahtar Sözcüğü

```java
public class Golge {
  private String ad;

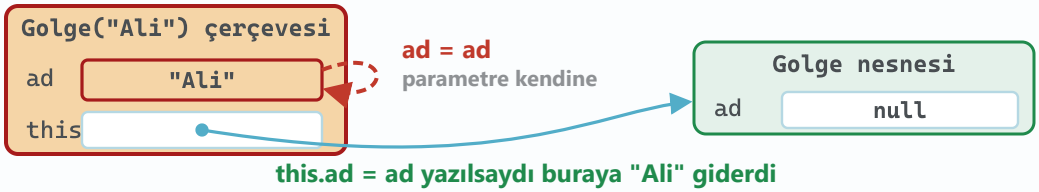
  public Golge(String ad) {
    ad = ad;            // this unutuldu!
  }

  public static void main(String[] args) {
    Golge g = new Golge("Ali");
    System.out.println(g.ad);
  }
}
```

ÇIKTI

```
null
```

- `this`, metodun **çağrıldığı nesneyi** işaret eden referanstır.
- `static` metotlarda **kullanılamaz** (orada nesne yoktur).
- En sık yapıcılarda, alan adı ile parametre adı **aynı** olduğunda kullanılır: `this.ad = ad;`
- Soldaki kodda `this` unutulmuş: `ad = ad;` parametreyi kendine atar, alan `null` kalır. Derleyici hata vermez!
- `this(...)` ise aynı sınıfın başka bir yapıcısını çağırır.



this ile Nesneyi Aktarmak

```java
public class Takim {
  private String ad;
  public Takim(String ad) {
    this.ad = ad;
  }
  public void oyuncuEkle(Oyuncu oyuncu) {
    oyuncu.takimaKatil(this);   // bu nesne
  }
  public String getAd() {
    return ad;
  }
  public static void main(String[] args) {
    Takim takim = new Takim("Giresunspor");
    Oyuncu oyuncu = new Oyuncu("Mert");
    takim.oyuncuEkle(oyuncu);
  }
}
```

- `this`, o anki nesneyi başka bir metoda **parametre** olarak iletmek için kullanılır.
- `takim.oyuncuEkle(oyuncu)` içinde `this` = `takim` nesnesi; `takimaKatil(this)` ile oyuncuya verilir.
- Aynı dosyada ikinci sınıf (`public` değil):

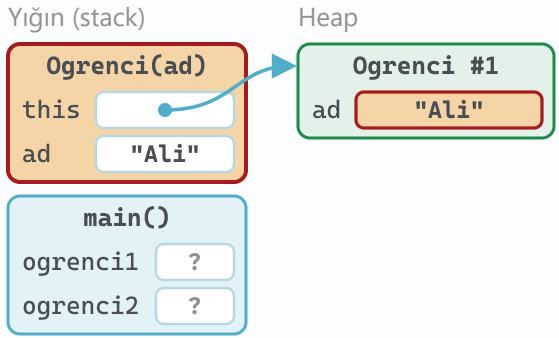
```java
class Oyuncu {
  private String ad;
  Oyuncu(String ad) {
    this.ad = ad;
  }
  void takimaKatil(Takim takim) {
    String t = takim.getAd();
    System.out.println(ad + " -> " + t);
  }
}
```

ÇIKTI

```
Mert -> Giresunspor
```

this ile Selam: Adım Adım

```
1  public class Ogrenci {
2    private String ad;
3
▶    public Ogrenci(String ad) {
▶      this.ad = ad;
6    }
7
8    public void selamVer() {
9      System.out.println("Merhaba, ben " + this.ad);
10   }
11
12   public static void main(String[] args) {
▶      Ogrenci ogrenci1 = new Ogrenci("Ali");
14     Ogrenci ogrenci2 = new Ogrenci("Ayşe");
15     ogrenci1.selamVer();
16     ogrenci2.selamVer();
17   }
18 }
```



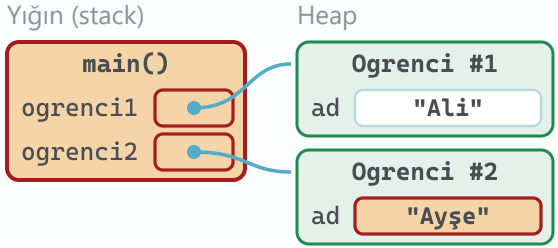
Yapıcıda `this` yeni nesnedir: `this.ad` (alan) = `ad` (parametre).

Adım 1 / 4

ÇIKTI

this ile Selam: Adım Adım

```
1  public class Ogrenci {
2    private String ad;
3
4    public Ogrenci(String ad) {
5      this.ad = ad;
6    }
7
8    public void selamVer() {
9      System.out.println("Merhaba, ben " + this.ad);
10   }
11
12   public static void main(String[] args) {
13     Ogrenci ogrenci1 = new Ogrenci("Ali");
▶      Ogrenci ogrenci2 = new Ogrenci("Ayşe");
15     ogrenci1.selamVer();
16     ogrenci2.selamVer();
17   }
18 }
```



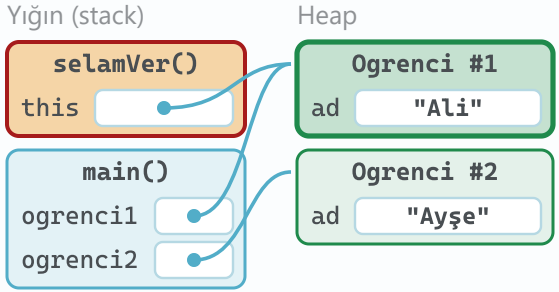
İkinci nesne de aynı yapıcıyla oluşur; o sırada `this` = #2 idi.

Adım 2 / 4

ÇIKTI

this ile Selam: Adım Adım

```
1  public class Ogrenci {
2    private String ad;
3
4    public Ogrenci(String ad) {
5      this.ad = ad;
6    }
7
▶    public void selamVer() {
▶      System.out.println("Merhaba, ben " + this.ad);
10   }
11
12   public static void main(String[] args) {
13     Ogrenci ogrenci1 = new Ogrenci("Ali");
14     Ogrenci ogrenci2 = new Ogrenci("Ayşe");
▶      ogrenci1.selamVer();
16     ogrenci2.selamVer();
17   }
18 }
```



`ogrenci1.selamVer()` : noktanın solundaki nesne `this` olur → "Ali".

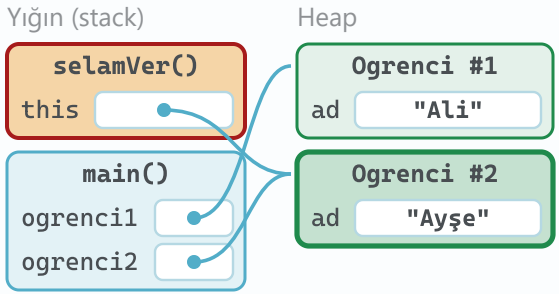
Adım 3 / 4

ÇIKTI

```
Merhaba, ben Ali
```

this ile Selam: Adım Adım

```
1  public class Ogrenci {
2    private String ad;
3
4    public Ogrenci(String ad) {
5      this.ad = ad;
6    }
7
▶  8    public void selamVer() {
▶  9      System.out.println("Merhaba, ben " + this.ad);
10   }
11
12   public static void main(String[] args) {
13     Ogrenci ogrenci1 = new Ogrenci("Ali");
14     Ogrenci ogrenci2 = new Ogrenci("Ayşe");
15     ogrenci1.selamVer();
▶ 16     ogrenci2.selamVer();
17   }
18 }
```



Aynı kod, farklı `this` → "Ayşe". Burada `this.ad` yerine `ad` yazmak da aynıdır.

Adım 4 / 4

ÇIKTI

```
Merhaba, ben Ali
Merhaba, ben Ayşe
```

`this(...)` ile Yapıcı Zinciri

- `this(...)` aynı sınıfın **başka bir yapıcısını** çağırır. Ortak ilk değer atamaları tek bir yapıcıda toplanır; öteki yapıcılar ona eksik argümanları ekleyerek devreder.
- `this(...)` yapıcının **ilk** satırı olmalıdır; başka bir deyimden sonra yazılırsa derlenmez.
- Çağrılan yapıcının gövdesi önce biter, sonra çağıranın kalan satırları çalışır; zincir kaç yapıcıdan geçerse geçsin tek bir nesne kurulur.
- Yapıcılar birbirini döngüsel çağırırsa derleyici bunu yakalar.

```
class Calisan {
  private String ad;
  private double maas;
  Calisan(String ad) {
    this(ad, 20000.0);   // aynı sınıfın yapıcısı
  }
  Calisan(String ad, double maas) {
    this.ad = ad;        // ilk değerler tek yerde
    this.maas = maas;
  }
}
```

İlk satırda değil: `A() { x = 1; this(2); }`

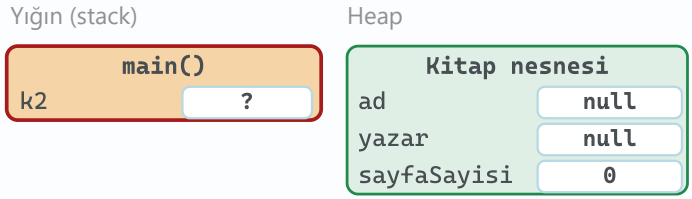
```
error: call to this must be first statement in
constructor
```

Döngü: `A() { this(1); }` ve `A(int x) { this(); }`

```
error: recursive constructor invocation
```

new Kitap("Nutuk", "Atatürk"): Adım Adım

```
1  class Kitap {
2    String ad;
3    String yazar;
4    int sayfaSayisi;
5    public Kitap(String ad) {
6      this.ad = ad;
7    }
8    public Kitap(String ad, String yazar) {
9      this(ad);  // yapıcı zincirleme
10     this.yazar = yazar;
11   }
12 }
13 // main içinde:
▶  Kitap k2 = new Kitap("Nutuk", "Atatürk");
15 System.out.println(k2.ad + " / "
16     + k2.yazar + " / " + k2.sayfaSayisi);
```



`new` heap'te yeni bir nesne ayırır; alanlar varsayılan değerleri alır: `null`, `null`, `0`.

Adım 1 / 7

ÇIKTI

ÇAĞRI YIĞINI

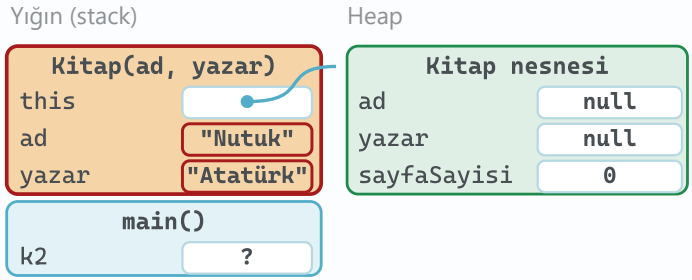
`main()`

DEĞİŞKENLER

`k2 = ?`

new Kitap("Nutuk", "Atatürk"): Adım Adım

```
1  class Kitap {
2    String ad;
3    String yazar;
4    int sayfaSayisi;
5    public Kitap(String ad) {
6      this.ad = ad;
7    }
▶   public Kitap(String ad, String yazar) {
9      this(ad);  // yapıcı zincirleme
10     this.yazar = yazar;
11   }
12 }
13 // main içinde:
14 Kitap k2 = new Kitap("Nutuk", "Atatürk");
15 System.out.println(k2.ad + " / "
16     + k2.yazar + " / " + k2.sayfaSayisi);
```



Argümanlar iki `String` olduğu için **iki parametreli** yapıcı çalışır. `this`, yeni nesneyi gösterir.

Adım 2 / 7

ÇIKTI

DEĞİŞKENLER

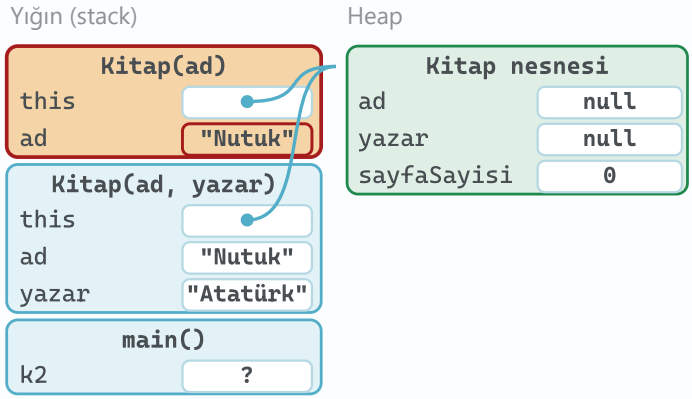
`ad = "Nutuk"` `yazar = "Atatürk"`

ÇAĞRI YIĞINI

- `Kitap(ad, yazar)`
- `main()`

new Kitap("Nutuk", "Atatürk"): Adım Adım

```
1  class Kitap {
2    String ad;
3    String yazar;
4    int sayfaSayisi;
5    public Kitap(String ad) {
6      this.ad = ad;
7    }
8    public Kitap(String ad, String yazar) {
▶      this(ad);  // yapıcı zincirleme
10     this.yazar = yazar;
11   }
12 }
13 // main içinde:
14 Kitap k2 = new Kitap("Nutuk", "Atatürk");
15 System.out.println(k2.ad + " / "
16     + k2.yazar + " / " + k2.sayfaSayisi);
```



`this(ad)` : **zincirleme**. Tek parametreli yapıcı, aynı nesne için çağrılır.

Adım 3 / 7

ÇIKTI

DEĞİŞKENLER

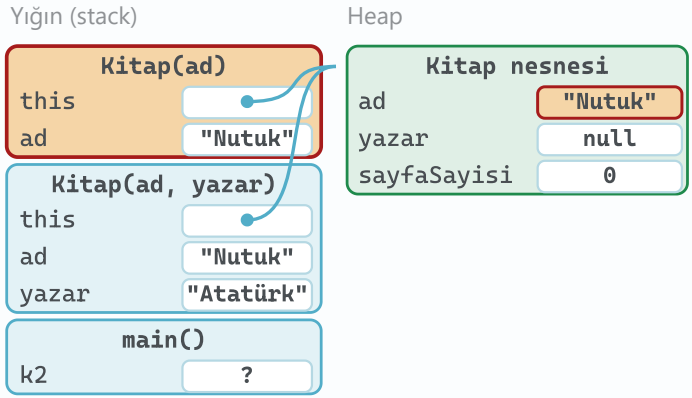
`ad = "Nutuk"`

ÇAĞRI YIĞINI

- `Kitap(ad)`
- `Kitap(ad, yazar)`
- `main()`

new Kitap("Nutuk", "Atatürk"): Adım Adım

```
1  class Kitap {
2    String ad;
3    String yazar;
4    int sayfaSayisi;
5    public Kitap(String ad) {
▶      this.ad = ad;
7    }
8    public Kitap(String ad, String yazar) {
9      this(ad);  // yapıcı zincirleme
10     this.yazar = yazar;
11   }
12 }
13 // main içinde:
14 Kitap k2 = new Kitap("Nutuk", "Atatürk");
15 System.out.println(k2.ad + " / "
16     + k2.yazar + " / " + k2.sayfaSayisi);
```



`this.ad = ad` : parametredeki değer nesnenin `ad` alanına yazılır.

Adım 4 / 7

ÇIKTI

DEĞİŞKENLER

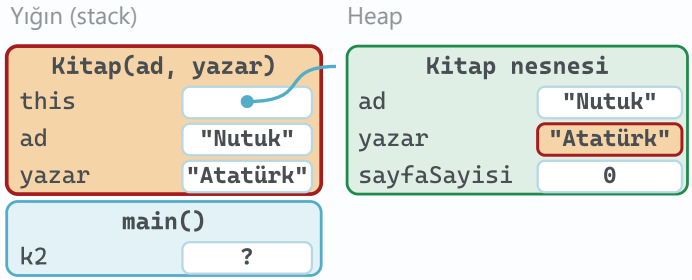
`ad = "Nutuk"`

ÇAĞRI YIĞINI

- **Kitap(ad)**
- Kitap(ad, yazar)
- main()

new Kitap("Nutuk", "Atatürk"): Adım Adım

```
1  class Kitap {
2    String ad;
3    String yazar;
4    int sayfaSayisi;
5    public Kitap(String ad) {
6      this.ad = ad;
7    }
8    public Kitap(String ad, String yazar) {
9      this(ad);  // yapıcı zincirleme
▶      this.yazar = yazar;
11   }
12 }
13 // main içinde:
14 Kitap k2 = new Kitap("Nutuk", "Atatürk");
15 System.out.println(k2.ad + " / "
16     + k2.yazar + " / " + k2.sayfaSayisi);
```



İç yapıcı bitti. Kalan satır: `this.yazar = yazar`.

Adım 5 / 7

ÇIKTI

DEĞİŞKENLER

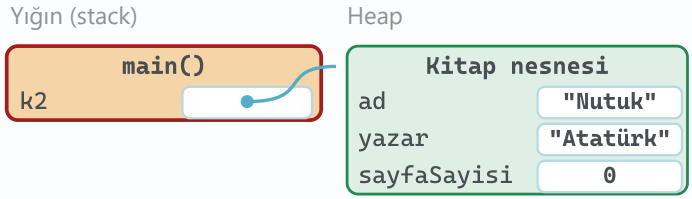
`ad = "Nutuk"` `yazar = "Atatürk"`

ÇAĞRI YIĞINI

- `Kitap(ad, yazar)`
- `main()`

new Kitap("Nutuk", "Atatürk"): Adım Adım

```
1  class Kitap {
2    String ad;
3    String yazar;
4    int sayfaSayisi;
5    public Kitap(String ad) {
6      this.ad = ad;
7    }
8    public Kitap(String ad, String yazar) {
9      this(ad);  // yapıcı zincirleme
10     this.yazar = yazar;
11   }
12 }
13 // main içinde:
▶ Kitap k2 = new Kitap("Nutuk", "Atatürk");
15 System.out.println(k2.ad + " / "
16     + k2.yazar + " / " + k2.sayfaSayisi);
```



Yapıcılar bitti; `new` ifadesinin değeri nesnenin **referansıdır**. `k2` artık nesneyi gösterir.

Adım 6 / 7

ÇIKTI

ÇAĞRI YIĞINI

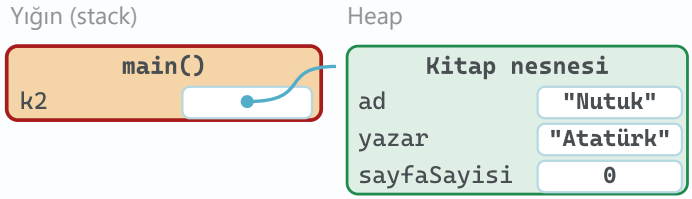
`main()`

DEĞİŞKENLER

`k2 = →Kitap`

new Kitap("Nutuk", "Atatürk"): Adım Adım

```
1  class Kitap {
2    String ad;
3    String yazar;
4    int sayfaSayisi;
5    public Kitap(String ad) {
6      this.ad = ad;
7    }
8    public Kitap(String ad, String yazar) {
9      this(ad);  // yapıcı zincirleme
10     this.yazar = yazar;
11   }
12 }
13 // main içinde:
14 Kitap k2 = new Kitap("Nutuk", "Atatürk");
▶  System.out.println(k2.ad + " / "
▶      + k2.yazar + " / " + k2.sayfaSayisi);
```



Nokta işleciyle alanlar okunur: **Nutuk / Atatürk / 0**.

Adım 7 / 7

ÇIKTI

```
Nutuk / Atatürk / 0
```

ÇAĞRI YIĞINI

`main()`

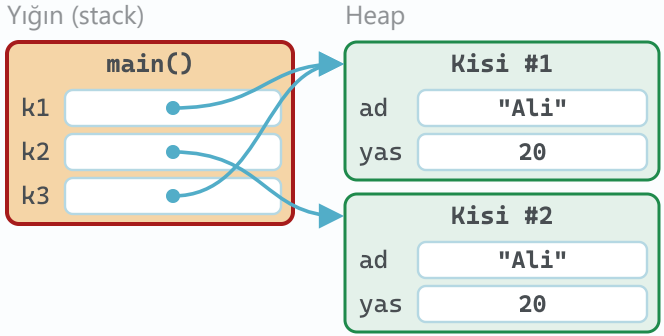
DEĞİŞKENLER

`k2 = →Kitap`

NESNELERİ KARŞILAŞTIRMA

== ve equals()

- Değişkenler yalnızca nesnelere ait **referansları** saklar.
- `==` işleci nesnelerin alanları yerine **bellek adreslerini** karşılaştırır: iki değişken **aynı nesneyi** mi gösteriyor?
- İki nesnenin içerik eşitliği için tüm özellikleri karşılaştıran `equals()` kullanılır; kendi sınıflarımızda **yeniden yazılmalıdır**.
- `String` ve wrapper'lar da nesnedir: klavyeden okunan "Java" ile `"Java"` sabiti `==` ile **false**, `equals` ile **true** verir.
- `equals` , `hashCode` , `toString` ve `getClass` her sınıfa `Object` 'ten gelir; `equals` yeniden yazılınca `hashCode` da yazılmalıdır.



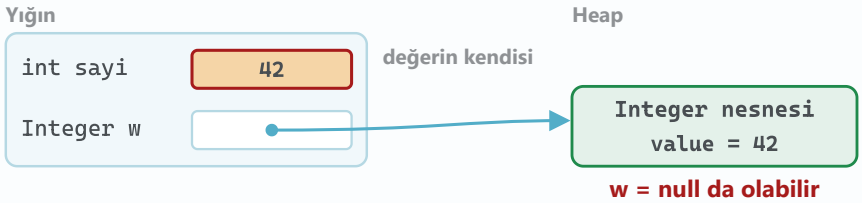
İfade	Sonuç
`k1 == k3`	true: aynı nesne
`k1 == k2`	false: farklı nesneler
`k1.equals(k2)`	true: aynı içerik*

* `equals` yeniden yazıldıysa; yoksa `Object` 'in sürümü `==` gibi davranır.

Wrapper Sınıflar

İlkel tip	Wrapper sınıf
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

- **Wrapper** (sarmalayıcı) sınıflar, temel veri tiplerini **nesne** olarak temsil eder.
- Bazı yerlerde nesne gerekir: `ArrayList<Integer>` gibi jenerik koleksiyonlar ilkel tip alamaz.
- İlkel tipe `null` atanamaz; wrapper değişkene atanabilir ("değer yok").
- **Yardımcı metotlar** sağlar: `Integer.parseInt("12")`, `Integer.MAX_VALUE`, `Character.isDigit(c)`.
- İlkel tipler daha **hızlı** ve az bellek kullanır; gerekmedikçe ilkel tip tercih edilir.
- **Autoboxing** ve **unboxing** dönüşümlerine dikkat edilmelidir.



Autoboxing ve Unboxing

```java
public class Sarmalayici {
  public static void main(String[] args) {
    // int tipinde bir değişken
    int sayi = 42;
    // Integer wrapper sınıfı kullanımı
    Integer wrapperSayi = Integer.valueOf(sayi);
    // Autoboxing (otomatik kutulama)
    Integer wrapperSayi2 = sayi;
    // Unboxing (açık ve otomatik)
    int sayi2 = wrapperSayi2.intValue();
    int sayi3 = wrapperSayi2;
    System.out.println(sayi2 + sayi3);
    System.out.println(wrapperSayi.equals(wrapperSayi2));
    System.out.println(Integer.parseInt("123") + 1);
    System.out.println(Integer.MAX_VALUE);
    System.out.println(Character.isDigit('7'));
    System.out.println(Double.parseDouble("2.5") * 2);
    Integer bos = null;      // int bos = null; derlenmez
    System.out.println(bos);
  }
}
```

- **Autoboxing**: ilkel tip → wrapper, **otomatik**. **Unboxing**: wrapper → ilkel tip.
- Açık biçimleri: `Integer.valueOf(sayi)`, `w.intValue()`.

ÇIKTI

```
84
true
124
2147483647
true
5.0
null
```

Wrapper Tuzakları

```java
import java.util.ArrayList;
public class Tuzaklar {
  public static void main(String[] args) {
    ArrayList<Integer> liste = new ArrayList<>();
    liste.add(5);                  // autoboxing: 5 -> Integer
    int ilk = liste.get(0);        // unboxing: Integer -> int
    System.out.println(ilk + 1);
    Integer a = 127, b = 127;
    Integer c = 128, d = 128;
    System.out.println(a == b);
    System.out.println(c == d);
    System.out.println(c.equals(d));
    Integer bos = null;
    int x = bos;                   // NullPointerException
  }
}
```

```
Exception in thread "main" java.lang.NullPointerException: Cannot
invoke "java.lang.Integer.intValue()" because "<local7>" is null
```

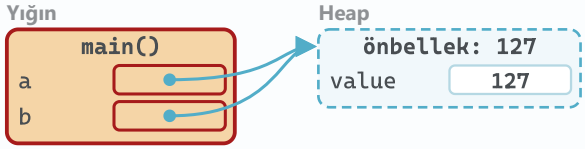
ÇIKTI

```
6
true
false
true
```

- Wrapper nesnelerini `==` ile karşılaştırmak yanlıştır: `==` referans karşılaştırır. −128…127 arası değerler önbellekten **aynı nesne** olarak gelir; 128 için **false**.
- Karşılaştırmada her zaman `equals()` kullanılır.
- `null` bir wrapper'ı açmak (unboxing) `NullPointerException` fırlatır.

Integer Önbelleği: Kod İzleme

```
▶ Integer a = 127, b = 127;
2 Integer c = 128, d = 128;
3 System.out.println(a == b);
4 System.out.println(c == d);
5 System.out.println(c.equals(d));
6 int t = c + d;
7 System.out.println(t);
8 Integer bos = null;
9 int x = bos;
```



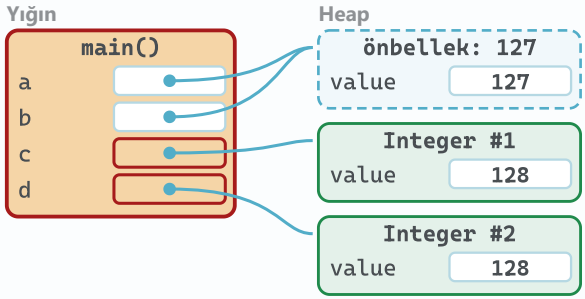
Otomatik kutulama `Integer.valueOf(127)` çağırır. −128…127 arası değerler önbellekte hazırdır: `a` ve `b` **aynı** nesneyi alır.

Adım 1 / 9

ÇIKTI

Integer Önbelleği: Kod İzleme

```
1 Integer a = 127, b = 127;
▶ Integer c = 128, d = 128;
3 System.out.println(a == b);
4 System.out.println(c == d);
5 System.out.println(c.equals(d));
6 int t = c + d;
7 System.out.println(t);
8 Integer bos = null;
9 int x = bos;
```



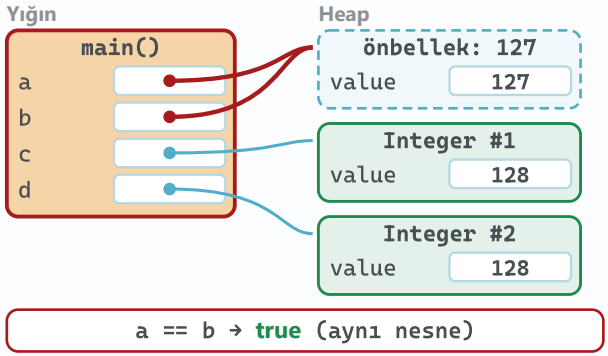
128 önbellek aralığının dışında: `valueOf` her seferinde **yeni** bir nesne oluşturur.

Adım 2 / 9

ÇIKTI

Integer Önbelleği: Kod İzleme

```
1 Integer a = 127, b = 127;
2 Integer c = 128, d = 128;
▶ System.out.println(a == b);
4 System.out.println(c == d);
5 System.out.println(c.equals(d));
6 int t = c + d;
7 System.out.println(t);
8 Integer bos = null;
9 int x = bos;
```



`==` referansları karşılaştırır; iki ok aynı nesneye gider: **true**.

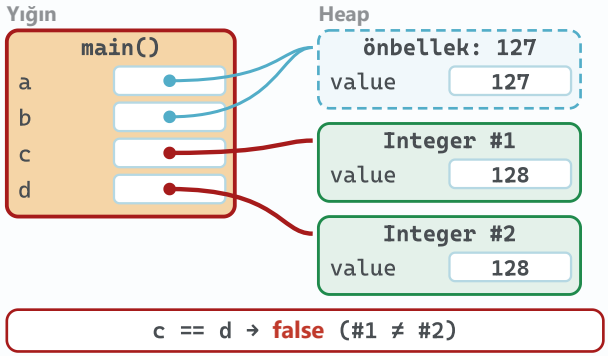
Adım 3 / 9

ÇIKTI

```
true
```

Integer Önbelleği: Kod İzleme

```
1 Integer a = 127, b = 127;
2 Integer c = 128, d = 128;
3 System.out.println(a == b);
▶ System.out.println(c == d);
5 System.out.println(c.equals(d));
6 int t = c + d;
7 System.out.println(t);
8 Integer bos = null;
9 int x = bos;
```



Değerler aynı (128) ama nesneler farklı: **false**. Wrapper'ları `==` ile karşılaştırmanın tuzağı budur.

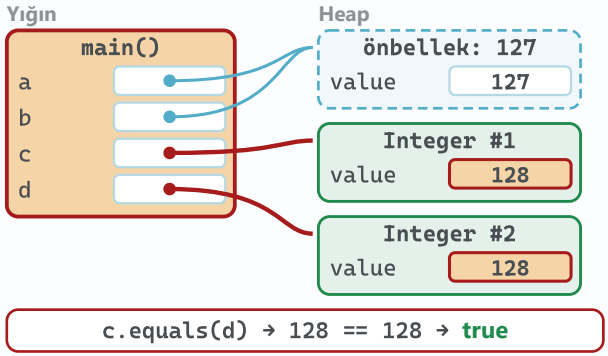
Adım 4 / 9

ÇIKTI

```
true
false
```

Integer Önbelleği: Kod İzleme

```
1 Integer a = 127, b = 127;
2 Integer c = 128, d = 128;
3 System.out.println(a == b);
4 System.out.println(c == d);
▶ System.out.println(c.equals(d));
6 int t = c + d;
7 System.out.println(t);
8 Integer bos = null;
9 int x = bos;
```



`Integer.equals` içerideki `int` değerleri karşılaştırır: **true**.

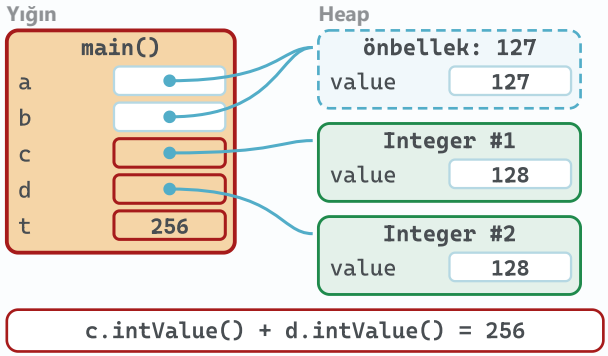
Adım 5 / 9

ÇIKTI

```
true
false
true
```

Integer Önbelleği: Kod İzleme

```
1 Integer a = 127, b = 127;
2 Integer c = 128, d = 128;
3 System.out.println(a == b);
4 System.out.println(c == d);
5 System.out.println(c.equals(d));
▶ int t = c + d;
7 System.out.println(t);
8 Integer bos = null;
9 int x = bos;
```



`+` nesnelerle çalışmaz: ikisi de kutudan çıkarılır (`intValue()`). `t` ilkel bir `int` tir, değeri yığında durur.

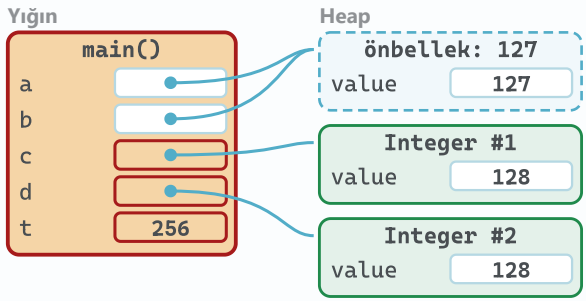
Adım 6 / 9

ÇIKTI

```
true
false
true
```

Integer Önbelleği: Kod İzleme

```
1 Integer a = 127, b = 127;
2 Integer c = 128, d = 128;
3 System.out.println(a == b);
4 System.out.println(c == d);
5 System.out.println(c.equals(d));
6 int t = c + d;
▶ System.out.println(t);
8 Integer bos = null;
9 int x = bos;
```



İlkel `t` doğrudan yazdırılır: **256**.

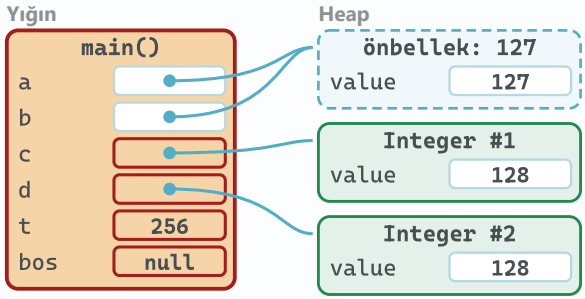
Adım 7 / 9

ÇIKTI

```
true
false
true
256
```

Integer Önbelleği: Kod İzleme

```
1 Integer a = 127, b = 127;
2 Integer c = 128, d = 128;
3 System.out.println(a == b);
4 System.out.println(c == d);
5 System.out.println(c.equals(d));
6 int t = c + d;
7 System.out.println(t);
▶ Integer bos = null;
9 int x = bos;
```



Wrapper değişkene `null` atanabilir; `int bos = null` ise derlenmezdi.

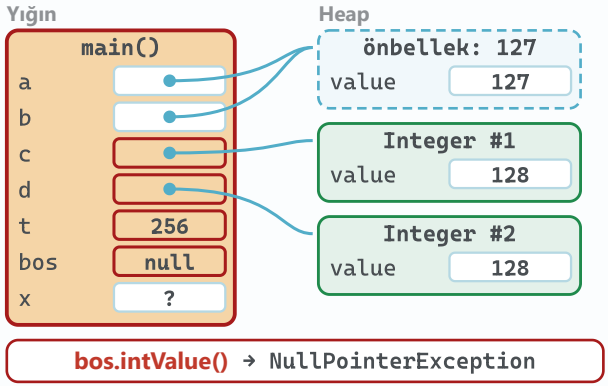
Adım 8 / 9

ÇIKTI

```
true
false
true
256
```

Integer Önbelleği: Kod İzleme

```
1 Integer a = 127, b = 127;
2 Integer c = 128, d = 128;
3 System.out.println(a == b);
4 System.out.println(c == d);
5 System.out.println(c.equals(d));
6 int t = c + d;
7 System.out.println(t);
8 Integer bos = null;
▶ int x = bos;
```



`int x = bos` gizlice `bos.intValue()` olur. Gösterilen nesne yok: **NullPointerException**, program durur.

Adım 9 / 9

ÇIKTI

```
true
false
true
256
Exception in thread "main"
java.lang.NullPointerException
```

Kayıtlar (Records)

- **Kayıt** (record, Java 16+), yalnızca veri taşıyan sınıfları **kısa ve temiz** yazmak için tasarlanmış bir sınıf türüdür.
- **Değiştirilemez** (immutable) nesneleri temsil eder: tüm alanlar `private final` 'dır; **set metodu yoktur**.
- Derleyici otomatik üretir: yapıcı, erişim metotları `x()` ve `y()` (adları `get` ile başlamaz), `equals`, `hashCode`, `toString`.

```
record Nokta(double x, double y) { }
```

Bu tek satır; iki alan, yapıcı, iki erişimci ve üç `Object` metodu içeren yaklaşık 30 satırlık bir sınıfa denktir.

```
public class KayitTesti {
  public static void main(String[] args) {
    Nokta nokta1 = new Nokta(1.0, 2.5);
    Nokta nokta2 = new Nokta(1.0, 2.5);
    boolean esitMi = nokta1.equals(nokta2);
    System.out.println(esitMi);
    System.out.println(nokta1 == nokta2);
    System.out.println(nokta1);
    System.out.println(nokta1.x() + nokta1.y());
  }
}
```

ÇIKTI

```
true
false
Nokta[x=1.0, y=2.5]
3.5
```

```
n.x = 3.0;   →   error: x has private access in
Nokta
```

enum (Numaralandırma Tipi)

```
enum HaftaGunleri {
  PAZARTESI, SALI, CARSAMBA, PERSEMBE,
  CUMA, CUMARTESI, PAZAR
}
public class GunTesti {
  public static void main(String[] args) {
    HaftaGunleri[] tumGunler = HaftaGunleri.values();
    HaftaGunleri persembe = HaftaGunleri.valueOf("PERSEMBE");
    System.out.println(tumGunler.length + " gün");
    System.out.println(persembe + " " + persembe.ordinal());
    for (HaftaGunleri gun : tumGunler) {
      boolean haftaSonu = gun == HaftaGunleri.CUMARTESI
          || gun == HaftaGunleri.PAZAR;
      if (haftaSonu) {
        System.out.println(gun + ": hafta sonu");
      }
    }
  }
}
```

ÇIKTI

```
7 gün
PERSEMBE 3
CUMARTESI: hafta sonu
PAZAR: hafta sonu
```

- **enum**, **sabit ve sınırlı** değerleri temsil eden özel bir türdür.
- `values()` tüm değerleri dizi olarak verir; üzerinde **döngü** kurulabilir.
- `valueOf` metinden değere çevirir; `ordinal()` 0'dan başlayan sıra. Değerler `==` ile karşılaştırılır.

İç Sınıflar (Inner Classes)

- **İç sınıf**, bir sınıfın içinde tanımlanan sınıftır. Dış sınıfın alanlarına ve metotlarına (`private` olanlar dahil) doğrudan erişebilir.
- Yalnızca bir sınıfın işine yarayan yardımcı sınıfları gruplamak için kullanılır; grafik arayüzlerde olay dinleyici (listener) sınıfları için çok yaygındır.

Tür	Nerede tanımlanır?	Nasıl oluşturulur?
static iç sınıf	Sınıf gövdesinde, `static` ile	Dış sınıf nesnesi olmadan: `new Outer.StaticInner()`
Üye iç sınıf	Sınıf gövdesinde, `static` olmadan	Bir dış sınıf nesnesi üzerinden: `outer.new MemberInner()`
Yerel iç sınıf	Bir metodun içinde	Yalnızca o metodun içinden
Anonim iç sınıf	Bir ifadenin içinde, **adsız**	Tanımlandığı yerde tek nesne; bir arayüzü veya soyut sınıfı uygular

static İç Sınıf

```java
public class Outer {
  // Dış sınıfın içinde statik bir iç sınıf
  static class StaticInner {
    void display() {
      System.out.println("Statik İç Sınıf");
    }
  }

  public static void main(String[] args) {
    // Dış sınıf örneği oluşturmadan erişim
    Outer.StaticInner staticInner = new Outer.StaticInner();
    staticInner.display();
  }
}
```

ÇIKTI

```
Statik İç Sınıf
```

- `static` anahtar sözcüğüyle tanımlanır.
- Dış sınıfın bir **nesnesi olmadan** oluşturulabilir: `new Outer.StaticInner()`.
- Dış sınıfın yalnızca `static` üyelerine doğrudan erişebilir.

Üye İç Sınıf

```java
public class Outer {
  private int sayi = 5;          // dış sınıfın özel alanı

  class MemberInner {            // üye iç sınıf
    void display() {
      System.out.println("Üye İç Sınıf, sayi = " + sayi);
    }
  }

  public static void main(String[] args) {
    // Dış sınıf örneği üzerinden üye iç sınıfa erişim
    Outer outer = new Outer();
    Outer.MemberInner memberInner = outer.new MemberInner();
    memberInner.display();
  }
}
```

ÇIKTI

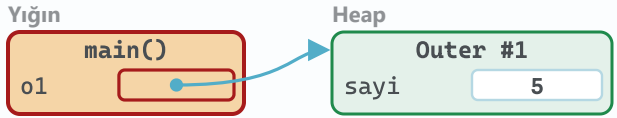
```
Üye İç Sınıf, sayi = 5
```

- Dış sınıfın bir **üyesi** olarak (`static` olmadan) tanımlanır.
- Her üye iç sınıf nesnesi bir **dış nesneye bağlıdır**: `outer.new MemberInner()`. Bu yüzden `private sayi` 'yi okuyabilir.
- `main` içinde doğrudan `new MemberInner()` derlenmez:

```
error: non-static variable this
cannot be referenced from a
static context
```

Üye İç Sınıf: Kod İzleme

```
1  public class Outer {
2      private int sayi = 5;
3      class MemberInner {
4          void display() {
5              System.out.println("sayi = " + sayi);
6          }
7      }
8      public static void main(String[] args) {
▶          Outer o1 = new Outer();
10         Outer o2 = new Outer();
11         o2.sayi = 8;
12         Outer.MemberInner m = o2.new MemberInner();
13         m.display();
14     }
15 }
```



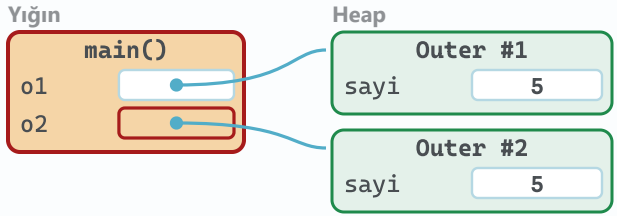
İlk dış nesne: kendi `sayi` alanı 5.

Adım 1 / 6

ÇIKTI

Üye İç Sınıf: Kod İzleme

```
1  public class Outer {
2      private int sayi = 5;
3      class MemberInner {
4          void display() {
5              System.out.println("sayi = " + sayi);
6          }
7      }
8      public static void main(String[] args) {
9          Outer o1 = new Outer();
▶          Outer o2 = new Outer();
11         o2.sayi = 8;
12         Outer.MemberInner m = o2.new MemberInner();
13         m.display();
14     }
15 }
```



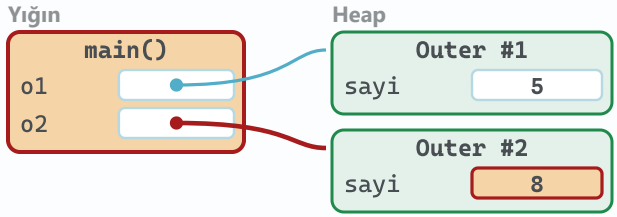
İkinci dış nesne; o da 5 ile başlar. İki ayrı `sayi` var.

Adım 2 / 6

ÇIKTI

Üye İç Sınıf: Kod İzleme

```
1  public class Outer {
2      private int sayi = 5;
3      class MemberInner {
4          void display() {
5              System.out.println("sayi = " + sayi);
6          }
7      }
8      public static void main(String[] args) {
9          Outer o1 = new Outer();
10         Outer o2 = new Outer();
▶          o2.sayi = 8;
12         Outer.MemberInner m = o2.new MemberInner();
13         m.display();
14     }
15 }
```



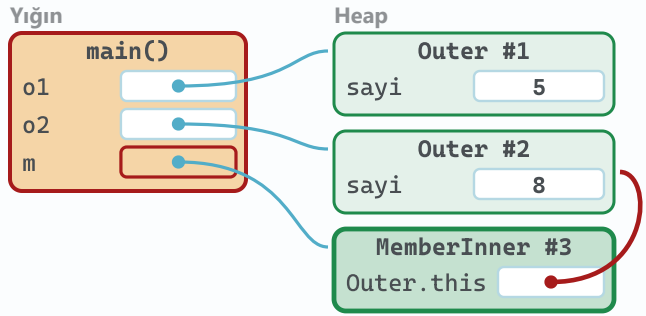
`main` de `Outer`'ın içinde olduğu için `private` alana yazabilir: yalnızca #2 değişir.

Adım 3 / 6

ÇIKTI

Üye İç Sınıf: Kod İzleme

```
1  public class Outer {
2      private int sayi = 5;
3      class MemberInner {
4          void display() {
5              System.out.println("sayi = " + sayi);
6          }
7      }
8      public static void main(String[] args) {
9          Outer o1 = new Outer();
10         Outer o2 = new Outer();
11         o2.sayi = 8;
▶          Outer.MemberInner m = o2.new MemberInner();
13         m.display();
14     }
15 }
```



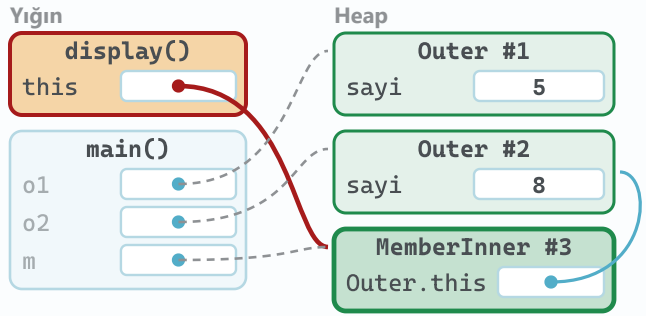
`o2.new MemberInner()` : iç nesne, onu oluşturan dış nesnenin referansını gizli bir alanda (`Outer.this`) saklar.

Adım 4 / 6

ÇIKTI

Üye İç Sınıf: Kod İzleme

```
1  public class Outer {
2      private int sayi = 5;
3      class MemberInner {
▶          void display() {
5              System.out.println("sayi = " + sayi);
6          }
7      }
8      public static void main(String[] args) {
9          Outer o1 = new Outer();
10         Outer o2 = new Outer();
11         o2.sayi = 8;
12         Outer.MemberInner m = o2.new MemberInner();
▶          m.display();
14     }
15 }
```



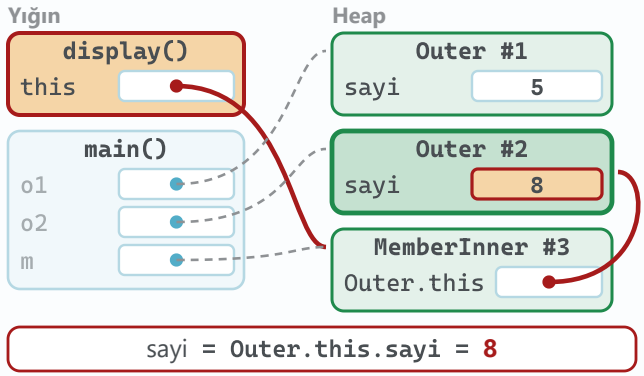
`m.display()` : metotta `this` iç nesnedir (#3). İç sınıfın kendi `sayi` alanı yok.

Adım 5 / 6

ÇIKTI

Üye İç Sınıf: Kod İzleme

```
1  public class Outer {
2      private int sayi = 5;
3      class MemberInner {
4          void display() {
▶              System.out.println("sayi = " + sayi);
6          }
7      }
8      public static void main(String[] args) {
9          Outer o1 = new Outer();
10         Outer o2 = new Outer();
11         o2.sayi = 8;
12         Outer.MemberInner m = o2.new MemberInner();
13         m.display();
14     }
15 }
```



`sayi` , `Outer.this.sayi` demektir: ok #2'ye gider, **8** yazılır. `o1.new MemberInner()` olsaydı 5 yazılırdı.

Adım 6 / 6

ÇIKTI

```
sayi = 8
```

Yerel İç Sınıf

```
public class Outer {
  void outerMethod() {
    class LocalInner {          // metodun içinde tanımlanır
      void display() {
        System.out.println("Yerel İç Sınıf");
      }
    }
    // Yerel iç sınıfa yalnızca bu metodun içinden erişilir
    LocalInner localInner = new LocalInner();
    localInner.display();
  }

  public static void main(String[] args) {
    Outer outer = new Outer();
    outer.outerMethod();
  }
}
```

ÇIKTI

```
Yerel İç Sınıf
```

- Bir **metodun içinde** tanımlanır; yalnızca o metodun içinden kullanılabilir.
- Metot dışında `LocalInner` adı bilinmez.
- Bir metodun içinde birkaç kez gereken küçük bir yardımcı tip için kullanılır.

Anonim İç Sınıf

```java
interface MyInterface {
  void myMethod();
}

public class Outer {
  public static void main(String[] args) {
    // Genellikle arayüzleri veya soyut sınıfları uygular
    MyInterface myInterface = new MyInterface() {
      @Override
      public void myMethod() {
        System.out.println("Anonim İç Sınıf");
      }
    };
    myInterface.myMethod();
  }
}
```

ÇIKTI

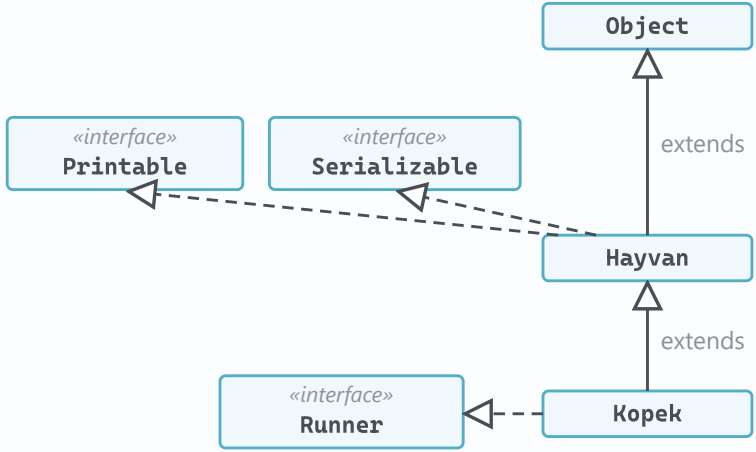
```
Anonim İç Sınıf
```

- **Adı olmayan** sınıftır; tanımlandığı yerde tek bir nesnesi oluşturulur.
- Genellikle bir **arayüzü** (interface) veya soyut sınıfı uygular: `new ArayuzAdi() { ... };` Sondaki `;` atama deyimine aittir.
- Tek metotlu arayüzler için Java 8+ **lambda** daha kısadır: `() -> ...`

Sınıflar ve Tipler

```
interface Printable { ... }
interface Runner { ... }
class Hayvan implements Printable, Serializable { ... }
class Kopek extends Hayvan implements Runner { ... }
var kazanan = new Kopek(...);
```

- Bir nesne **tek bir sınıfa** ait olabilir, ama **birçok türe** sahip olabilir.
- `kazanan` yalnızca `Kopek` sınıfına aittir.
- Ama tipleri: `Kopek`, `Runner`, `Hayvan`, `Printable`, `Serializable`, `Object`; hepsi için `instanceof` true verir.
- Bu, **çok biçimliliğin** temelidir.



Düz çizgi + boş üçgen: kalıtım (extends); kesikli: arayüzü uygulama (implements).

SINIF TÜRLERİ VE TASARIM

Sınıf Çeşitleri

Sınıf, bir nesnenin yapısını ve davranışını tanımlayan bir şablondur. Amaçlarına göre farklı sınıf çeşitleri vardır:

Çeşit	Özellik	Örnek
Numaralandırma (enum)	Sabit bir nesne kümesine sahiptir	`enum Gun { PAZARTESI, SALI, ... }`
Singleton	Yalnızca **bir** nesnesi oluşturulabilir	`private` yapıcı + `static` tek nesne
Soyut (abstract)	Kendi nesnesi oluşturulamaz	`abstract class Sekil`
Mühürlü (sealed, Java 17+)	Alt sınıfları sabit bir kümedir	`sealed interface Hareket permits Yuru, Kos`
Final	Alt sınıf **türetilemez**	`final class String`
Veri (record)	Nesneleri değiştirilemez (immutable)	`record Nokta(double x, double y)`
Araç (utility)	Yalnızca static metotlar içerir	`Math`, `Arrays`
Uygulama (application)	`main` metodunu içerir	Programın başladığı sınıf

Sınıf Çeşitleri: Örnek

```
final class Araclar {              // araç ve final sınıf
  private Araclar() { }          // nesne oluşturulamaz
  static int kare(int x) { return x * x; }
}
class Ayarlar {                    // singleton
  private static final Ayarlar TEK = new Ayarlar();
  private Ayarlar() { }
  static Ayarlar getir() { return TEK; }
}
abstract class Sekil {             // soyut sınıf
  abstract double alan();
}
// mühürlü (sealed, Java 17+)
sealed interface Hareket permits Yuru, Kos { }
final class Yuru implements Hareket { }
final class Kos implements Hareket { }
public class Cesitler {            // uygulama sınıfı
  public static void main(String[] args) {
    System.out.println(Araclar.kare(7));
    System.out.println(Ayarlar.getir() == Ayarlar.getir());
  }
}
```

ÇIKTI

```
49
true
```

- `Araclar` : `private` yapıcı ile nesne oluşturulamaz, yalnızca static metot; `final` ile türetilemez.
- `Ayarlar` : tek nesne her seferinde aynıdır, bu yüzden `==` true.
- `new Sekil()` derleme hatası verir (soyut); `Hareket` 'i yalnızca `Yuru` ve `Kos` uygulayabilir.

Tasarım Örneği: RasyonelSayi

```
public class RasyonelSayi {
  // Temsil: gizli alanlar
  private int pay, payda;
  // Tanımlama: public yapıcı ve metotlar
  public RasyonelSayi(int pay, int payda) {
    int e = ebob(Math.abs(pay), Math.abs(payda));
    this.pay = pay / e;
    this.payda = payda / e;
  }
  public RasyonelSayi topla(RasyonelSayi d) {
    int yeniPay = pay * d.payda + d.pay * payda;
    return new RasyonelSayi(yeniPay, payda * d.payda);
  }
  @Override
  public String toString() { return pay + "/" + payda; }
  // Gerçekleme ayrıntısı: gizli yardımcı metot
  private static int ebob(int a, int b) {
    while (b != 0) { int k = a % b; a = b; b = k; }
    return a;
  }
}
```

- **Tanımlama**: `public` yapıcı, `topla`, `toString`.
- **Temsil**: `private int pay, payda`; yapıcı kesri sadeleştirir.
- **Gerçekleme**: metot gövdeleri ve gizli `ebob` yardımcısı (Öklid algoritması).
- `topla` mevcut nesneyi değiştirmez, **yeni** bir nesne döndürür.

`a = new RasyonelSayi(8, 10)`, `b = new RasyonelSayi(2, 3)`; `println(a)` ve `println(a.topla(b))`:

ÇIKTI

```
4/5
22/15
```

RasyonelSayi: Kod İzleme

```
▶ public RasyonelSayi(int pay, int payda) {
2     int e = ebob(Math.abs(pay), Math.abs(payda));
3     this.pay = pay / e;
4     this.payda = payda / e;
5 }
6 private static int ebob(int a, int b) {
7     while (b != 0) { int k = a % b; a = b; b = k; }
8     return a;
9 }
10 public RasyonelSayi topla(RasyonelSayi d) {
11     int yeniPay = pay * d.payda + d.pay * payda;
12     return new RasyonelSayi(yeniPay, payda * d.payda);
13 }
▶ RasyonelSayi a = new RasyonelSayi(6, 8);   // main
15 RasyonelSayi b = new RasyonelSayi(1, 6);
16 System.out.println(a.topla(b) + " " + a);
```

Yığın

RasyonelSayi()

pay	6
payda	8
e	?

main()

a	?

Heap

Rasyonel #1

pay	0
payda	0

`new RasyonelSayi(6, 8)` : nesnenin alanları önce 0; yapıcının parametreleri 6 ve 8.

Adım 1 / 10

ÇIKTI

RasyonelSayi: Kod İzleme

```
1  public RasyonelSayi(int pay, int payda) {
▶      int e = ebob(Math.abs(pay), Math.abs(payda));
3      this.pay = pay / e;
4      this.payda = payda / e;
5  }
▶  private static int ebob(int a, int b) {
7      while (b != 0) { int k = a % b; a = b; b = k; }
8      return a;
9  }
10 public RasyonelSayi topla(RasyonelSayi d) {
11     int yeniPay = pay * d.payda + d.pay * payda;
12     return new RasyonelSayi(yeniPay, payda * d.payda);
13 }
14 RasyonelSayi a = new RasyonelSayi(6, 8);   // main
15 RasyonelSayi b = new RasyonelSayi(1, 6);
16 System.out.println(a.topla(b) + " " + a);
```

Yığın

ebob()

a	6
b	8
k	–

RasyonelSayi()

pay	6
payda	8
e	?

main()

a	?

Heap

Rasyonel #1

pay	0
payda	0

Gizli yardımcı `ebob(6, 8)` çağrılır (Öklid algoritması). Kullanıcı bu metodu göremez.

Adım 2 / 10

ÇIKTI

RasyonelSayi: Kod İzleme

```
1  public RasyonelSayi(int pay, int payda) {
2      int e = ebob(Math.abs(pay), Math.abs(payda));
3      this.pay = pay / e;
4      this.payda = payda / e;
5  }
6  private static int ebob(int a, int b) {
▶      while (b != 0) { int k = a % b; a = b; b = k; }
8      return a;
9  }
10 public RasyonelSayi topla(RasyonelSayi d) {
11     int yeniPay = pay * d.payda + d.pay * payda;
12     return new RasyonelSayi(yeniPay, payda * d.payda);
13 }
14 RasyonelSayi a = new RasyonelSayi(6, 8);   // main
15 RasyonelSayi b = new RasyonelSayi(1, 6);
16 System.out.println(a.topla(b) + " " + a);
```

Yığın

ebob()

a	8
b	6
k	6

RasyonelSayi()

pay	6
payda	8
e	?

main()

a	?

Heap

Rasyonel #1

pay	0
payda	0

6 % 8 = 6; a ← 8, b ← 6. İlk turda sayılar yer değiştirir.

Adım 3 / 10

ÇIKTI

RasyonelSayi: Kod İzleme

```
1  public RasyonelSayi(int pay, int payda) {
2      int e = ebob(Math.abs(pay), Math.abs(payda));
3      this.pay = pay / e;
4      this.payda = payda / e;
5  }
6  private static int ebob(int a, int b) {
▶      while (b != 0) { int k = a % b; a = b; b = k; }
8      return a;
9  }
10 public RasyonelSayi topla(RasyonelSayi d) {
11     int yeniPay = pay * d.payda + d.pay * payda;
12     return new RasyonelSayi(yeniPay, payda * d.payda);
13 }
14 RasyonelSayi a = new RasyonelSayi(6, 8);   // main
15 RasyonelSayi b = new RasyonelSayi(1, 6);
16 System.out.println(a.topla(b) + " " + a);
```

Yığın

ebob()

a	6
b	2
k	2

RasyonelSayi()

pay	6
payda	8
e	?

main()

a	?

Heap

Rasyonel #1

pay	0
payda	0

8 % 6 = 2; a ← 6, b ← 2.

Adım 4 / 10

ÇIKTI

RasyonelSayi: Kod İzleme

```
1  public RasyonelSayi(int pay, int payda) {
2      int e = ebob(Math.abs(pay), Math.abs(payda));
3      this.pay = pay / e;
4      this.payda = payda / e;
5  }
6  private static int ebob(int a, int b) {
▶      while (b != 0) { int k = a % b; a = b; b = k; }
8      return a;
9  }
10 public RasyonelSayi topla(RasyonelSayi d) {
11     int yeniPay = pay * d.payda + d.pay * payda;
12     return new RasyonelSayi(yeniPay, payda * d.payda);
13 }
14 RasyonelSayi a = new RasyonelSayi(6, 8);   // main
15 RasyonelSayi b = new RasyonelSayi(1, 6);
16 System.out.println(a.topla(b) + " " + a);
```

Yığın

ebob()

a	2
b	0
k	0

RasyonelSayi()

pay	6
payda	8
e	?

main()

a	?

Heap

Rasyonel #1

pay	0
payda	0

6 % 2 = 0; a ← 2, b ← 0. `b == 0` : döngü biter.

Adım 5 / 10

ÇIKTI

RasyonelSayi: Kod İzleme

```
1  public RasyonelSayi(int pay, int payda) {
2      int e = ebob(Math.abs(pay), Math.abs(payda));
▶      this.pay = pay / e;
▶      this.payda = payda / e;
5  }
6  private static int ebob(int a, int b) {
7      while (b != 0) { int k = a % b; a = b; b = k; }
▶      return a;
9  }
10 public RasyonelSayi topla(RasyonelSayi d) {
11     int yeniPay = pay * d.payda + d.pay * payda;
12     return new RasyonelSayi(yeniPay, payda * d.payda);
13 }
14 RasyonelSayi a = new RasyonelSayi(6, 8);   // main
15 RasyonelSayi b = new RasyonelSayi(1, 6);
16 System.out.println(a.topla(b) + " " + a);
```

Yığın

RasyonelSayi()

pay	6
payda	8
e	2

main()

a	?

Heap

Rasyonel #1

pay	3
payda	4

`6/8 → 6/2 = 3, 8/2 = 4`

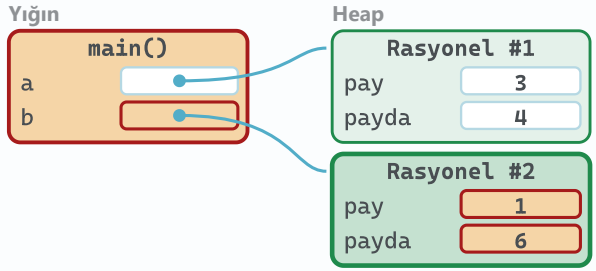
`ebob` 2 döndürür; yapıcı kesri sadeleştirerek saklar: nesne **3/4**. Temsil her zaman sade kalır.

Adım 6 / 10

ÇIKTI

RasyonelSayi: Kod İzleme

```
1  public RasyonelSayi(int pay, int payda) {
2      int e = ebob(Math.abs(pay), Math.abs(payda));
3      this.pay = pay / e;
4      this.payda = payda / e;
5  }
6  private static int ebob(int a, int b) {
7      while (b != 0) { int k = a % b; a = b; b = k; }
8      return a;
9  }
10 public RasyonelSayi topla(RasyonelSayi d) {
11     int yeniPay = pay * d.payda + d.pay * payda;
12     return new RasyonelSayi(yeniPay, payda * d.payda);
13 }
14 RasyonelSayi a = new RasyonelSayi(6, 8);   // main
▶ RasyonelSayi b = new RasyonelSayi(1, 6);
16 System.out.println(a.topla(b) + " " + a);
```



`ebob(1, 6) = 1 → 1/6`

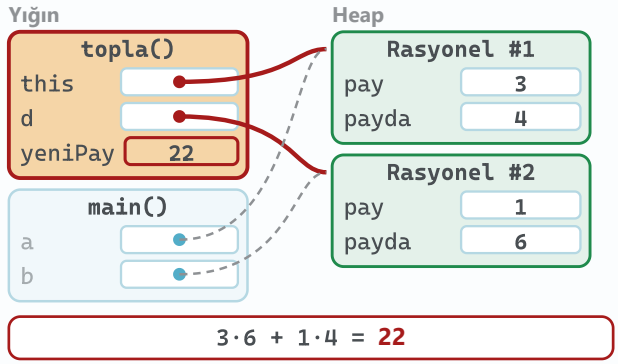
Aynı adımlar `(1, 6)` için: `ebob` 1 döndürür, kesir zaten sade: **1/6**.

Adım 7 / 10

ÇIKTI

RasyonelSayi: Kod İzleme

```
 1 public RasyonelSayi(int pay, int payda) {
 2     int e = ebob(Math.abs(pay), Math.abs(payda));
 3     this.pay = pay / e;
 4     this.payda = payda / e;
 5 }
 6 private static int ebob(int a, int b) {
 7     while (b != 0) { int k = a % b; a = b; b = k; }
 8     return a;
 9 }
▶ public RasyonelSayi topla(RasyonelSayi d) {
▶     int yeniPay = pay * d.payda + d.pay * payda;
12     return new RasyonelSayi(yeniPay, payda * d.payda);
13 }
14 RasyonelSayi a = new RasyonelSayi(6, 8);   // main
15 RasyonelSayi b = new RasyonelSayi(1, 6);
▶ System.out.println(a.topla(b) + " " + a);
```



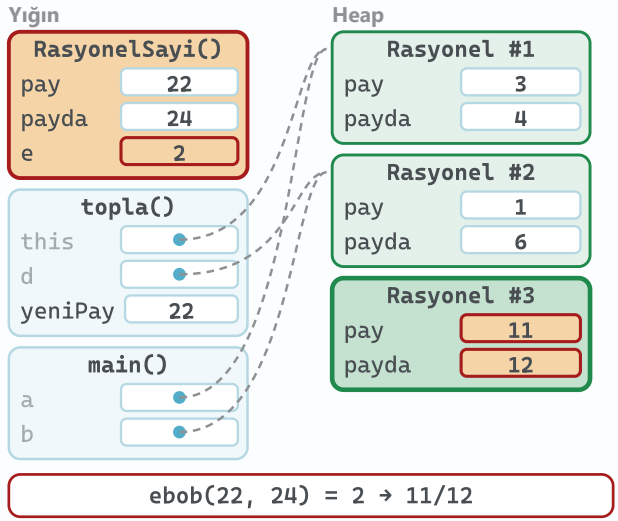
`a.topla(b)` : `this` = #1, `d` = #2. Aynı sınıf olduğu için `d.payda` gibi private alanlar okunabilir.

Adım 8 / 10

ÇIKTI

RasyonelSayi: Kod İzleme

```
▶ public RasyonelSayi(int pay, int payda) {
▶     int e = ebob(Math.abs(pay), Math.abs(payda));
3     this.pay = pay / e;
4     this.payda = payda / e;
5 }
6 private static int ebob(int a, int b) {
7     while (b != 0) { int k = a % b; a = b; b = k; }
8     return a;
9 }
10 public RasyonelSayi topla(RasyonelSayi d) {
11     int yeniPay = pay * d.payda + d.pay * payda;
▶     return new RasyonelSayi(yeniPay, payda * d.payda);
13 }
14 RasyonelSayi a = new RasyonelSayi(6, 8);   // main
15 RasyonelSayi b = new RasyonelSayi(1, 6);
16 System.out.println(a.topla(b) + " " + a);
```



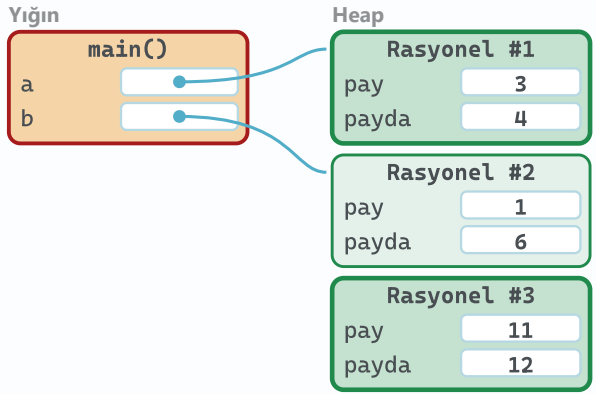
`ebob(22, 24) = 2 → 11/12`

`new RasyonelSayi(22, 24)` : `ebob` 24, 22, 2, 0 adımlarıyla 2 bulur; yeni nesne **11/12**.

ÇIKTI

RasyonelSayi: Kod İzleme

```
1  public RasyonelSayi(int pay, int payda) {
2      int e = ebob(Math.abs(pay), Math.abs(payda));
3      this.pay = pay / e;
4      this.payda = payda / e;
5  }
6  private static int ebob(int a, int b) {
7      while (b != 0) { int k = a % b; a = b; b = k; }
8      return a;
9  }
10 public RasyonelSayi topla(RasyonelSayi d) {
11     int yeniPay = pay * d.payda + d.pay * payda;
12     return new RasyonelSayi(yeniPay, payda * d.payda);
13 }
14 RasyonelSayi a = new RasyonelSayi(6, 8);   // main
15 RasyonelSayi b = new RasyonelSayi(1, 6);
▶  System.out.println(a.topla(b) + " " + a);
```



`topla()` **yeni** nesne döndürdü; a hâlâ 3/4

Önce sonuç (#3), sonra `a` yazdırılır: `topla` mevcut nesneyi değiştirmez, yeni nesne döndürür.

Adım 10 / 10

ÇIKTI

```
11/12 3/4
```

ÖZET

Özet

Kavram	Özü
Sınıf / nesne	Sınıf taslaktır; nesne `new` ile üretilen örneğidir.
Yapıcı	Sınıfla aynı ad, dönüş tipi yok; yazılmazsa varsayılan yapıcı eklenir.
Örnek / static üye	Örnek alanı her nesnede ayrı; static alan sınıfta tek ve ortak.
İlklendiriciler	`static { }` bir kez; `{ }` her nesnede, yapıcı gövdesinden önce.
Erişim ve gizleme	Alanlar `private`; get/set ile kontrollü erişim.
Referans	Değişken nesneyi değil referansı tutar; atama referansı kopyalar.
`this` / `this(...)`	Metodun çağrıldığı nesne (static metotta yok) / aynı sınıfın başka yapıcısı, ilk satırda.
`==` / `equals`	Aynı nesne mi? / aynı içerik mi?
Wrapper	`int` ↔ `Integer`: autoboxing ve unboxing.
record / enum	Değiştirilemez veri sınıfı / sabit değer kümesi.
İç sınıflar	static, üye, yerel, anonim.

ÖZET

İlgili Sayfalar

Yığın ve Heap Gezgini

Simülatör: `new` ile oluşan nesneler, referans atama, `this` ve metot çerçeveleri yığında ve heap'te adım adım.

Çalışma Soruları

Yapıcı zinciri, ilklendirme sırası, `static` üyeler, referanslar, `==` ve `equals`, wrapper, `enum` ve iç sınıf soruları.

ÖZET

Sık Yapılan Hatalar (1)

Örnek metodu main'den nesnesiz çağırmak

```
error: non-static method selamVer() cannot be referenced from a static context
```

Çağrı bir nesne üzerinden yapılır (`new Selam().selamVer()`); metot gerçekten nesneye bağlı değilse `static` yapılır.

Yapıcıya void yazmak

```
error: constructor Hata1 in class Hata1 cannot be applied to given types;
```

`public void Hata1(String m)` yapıcı değil, sıradan metottur. Yapıcının dönüş tipi olmaz.

Yalnızca parametreli yapıcı varken new Urun()

```
error: constructor Urun in class Urun cannot be applied to given types;
```

Bir yapıcı yazılınca varsayılan yapıcı eklenmez; gerekiyorsa parametresiz yapıcı da yazılır.

new unutmak

```
error: cannot find symbol symbol: method Urun(String)
```

Nesne oluşturmak için `new Urun("Kalem")` yazılmalıdır.

ÖZET

Sık Yapılan Hatalar (2)

private alana dışarıdan erişmek

```
error: hesapBakiyesi has private access in BankaHesabi
```

Alana sınıfın get/set metotlarıyla erişilir.

static metotta this

```
error: non-static variable this cannot be referenced from a static context
```

Static metodun bir nesnesi yoktur.

this unutmak (gölgeleme)

```
ad = ad; // derlenir, alan null kalır
```

Parametre alanı gölgeler; doğrusu `this.ad = ad;`.

this(...) ilk satırda değil

```
error: call to this must be first statement in constructor
```

Öteki yapıcıya devretme yapıcının ilk deyimi olmalıdır.

ÖZET

Sık Yapılan Hatalar (3)

Nesneleri == ile karşılaştırmak

`k1 == k2 // false`

İçerik `equals()` ile karşılaştırılır; `String` ve wrapper'lar dahil.

null referansı kullanmak

`NullPointerException`

Nesne oluşturulmadan veya `null` atandıktan sonra metot çağrılamaz; önce `null` denetimi yapılır.

Record alanını değiştirmeye çalışmak

`error: x has private access in Nokta`

Record değiştirilemezdir; yeni değer için yeni bir nesne oluşturulur.

Üye iç sınıfı static main'de oluşturmak

`error: non-static variable this cannot be referenced from a static context`

Doğrusu `outer.new MemberInner()`; ya da iç sınıf `static` yapılır.

ÖZET

Örnek Problemler (1)

Problem 1. Ne yazdırır?

```
class Sayac {
  static int toplam = 0;
  int adet = 0;
  void arttir() { toplam++; adet++; }
}
public class Problem1 {
  public static void main(String[] args) {
    Sayac a = new Sayac(), b = new Sayac();
    a.arttir(); a.arttir(); b.arttir();
    System.out.print(a.adet + " " + b.adet);
    System.out.println(" " + Sayac.toplam);
  }
}
```

Problem 2. Ne yazdırır?

```
class Kutu { int deger; }
public class Problem2 {
  public static void main(String[] args) {
    Kutu k1 = new Kutu();
    k1.deger = 5;
    Kutu k2 = k1;
    k2.deger = 9;
    Kutu k3 = new Kutu();
    k3.deger = 9;
    System.out.println(k1.deger);
    System.out.println(k1 == k2);
    System.out.println(k2 == k3);
  }
}
```

▶ **Cevapları göster**

ÖZET

Örnek Problemler (2)

Problem 3. Ne yazdırır?

```
class A {
  static { System.out.print("S"); }
  { System.out.print("I"); }
  A() { System.out.print("C"); }
}
public class Problem3 {
  public static void main(String[] args) {
    System.out.print("M");
    new A();
    new A();
    System.out.println();
  }
}
```

Problem 4. Ne yazdırır?

```
class Kutu { int deger; }
public class Problem4 {
  static void degistir(Kutu k) {
    k.deger = 100;
    k = new Kutu();
    k.deger = 7;
  }
  public static void main(String[] args) {
    Kutu x = new Kutu();
    x.deger = 1;
    degistir(x);
    System.out.println(x.deger);
  }
}
```

▶ **Cevapları göster**

Örnek Problemler (3)

Problem 5. Ne yazdırır?

```
record Para(int lira) { }
public class Problem5 {
  public static void main(String[] args) {
    Para p1 = new Para(10);
    Para p2 = new Para(10);
    Integer i1 = 1000, i2 = 1000;
    System.out.println(p1.equals(p2));
    System.out.println(p1 == p2);
    System.out.println(i1.equals(i2));
    System.out.println(p1);
  }
}
```

Problem 6. Ne yazdırır?

```
class Nokta {
  int x, y;
  Nokta(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
public class Problem6 {
  public static void main(String[] args) {
    Nokta a = new Nokta(1, 2);
    Nokta b = new Nokta(1, 2);
    System.out.println(a.equals(b));
    System.out.println(a.x == b.x);
  }
}
```

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. `Kitap` sınıfı: `private` `baslik`, `yazar`, `sayfaSayisi`; üç parametreli yapıcı, get metotları ve `toString()`.
2. `Kitap`'a static `toplamKitapSayisi` sayacı ve yalnızca pozitif değeri kabul eden `setSayfaSayisi` ekleyin.
3. `Daire` sınıfı: `yaricap`, `alan()`, `cevre()`. `new Daire(3)` için iki basamakla yazdırın.
4. `Mevsim` enum'unun değerlerini sıra numaralarıyla yazdırın.
5. `OgrenciKaydi(String ad, int numara)` record'unu yazıp bir nesnesini yazdırın.

▶ **Cevaplar / ipuçları**

ÖZET

Kaynaklar

- Oracle, *The Java Tutorials*: "Classes and Objects", "Nested Classes", "Enum Types", "Record Classes", "Autoboxing and Unboxing".
- J. Gosling ve diğ., *The Java Language Specification, Java SE 21 Edition* (Bölüm 8 Classes: 8.3 Fields, 8.6–8.7 Initializers, 8.8 Constructors, 8.9 Enum, 8.10 Record).
- P. Deitel ve H. Deitel, *Java: How to Program, Early Objects*, 11. baskı, Pearson, 2017 (Bölüm 3 ve 8: Classes and Objects).
- K. Sierra ve B. Bates, *Head First Java*, 3. baskı, O'Reilly, 2022.
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12. baskı, 2022 (Bölüm 4: Objects and Classes).
- A. B. Downey ve C. Mayfield, *Think Java*, 2. baskı, O'Reilly, 2019 (Bölüm 10–11).
- J. Bloch, *Effective Java*, 3. baskı, Addison-Wesley, 2018 (equals, hashCode, toString maddeleri).

Son

Bölüm 4.1: Nesneler ve Sınıflar

Sonraki bölüm: Bölüm 4.2: Sarmalama · Terim sözlüğü