Bölüm 4.2: Sarmalama

Java ile Nesne Yönelimli Programlama

Bilgi saklama | Ön ve son koşullar | Erişim belirleyiciler | Getter ve setter | Arayüz ve gerçekleme | Değiştirilemez sınıflar | Paketler ve import

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Sarmalamayı açıklamak
Veriyi ve onu işleyen metotları tek sınıfta toplamayı, iç ayrıntıları gizlemeyi (bilgi saklama) ve "ne / nasıl" ayrımını örnekle anlatmak.

Metot sözleşmesi yazmak
Bir metodun **ön koşulunu** ve **son koşulunu** yorum satırıyla belirtmek ve kodda kontrol etmek.

Erişimi doğru seçmek
`public`, `private`, `protected` ve paket erişiminin kimlere açık olduğunu tablo ile söylemek.

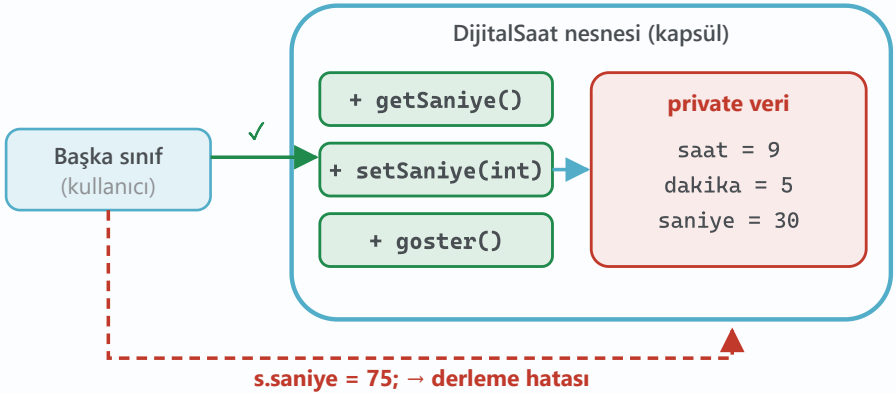
Getter ve setter yazmak
Alanları `private` yapıp doğrulama yapan erişim metotları yazmak; `this.alan = alan` kalıbını kullanmak.

Arayüz ile gerçeklemeyi ayırmak
Sınıfın dış yüzünü değiştirmeden iç yapısını değiştirebilmek; `interface` ve `implements` ile tanışmak.

Paket kullanmak
`package` ve `import` ile sınıfları gruplamak, derleyip çalıştırmak; sınıf adı çakışmalarını çözmek.

Sarmalama (Encapsulation)

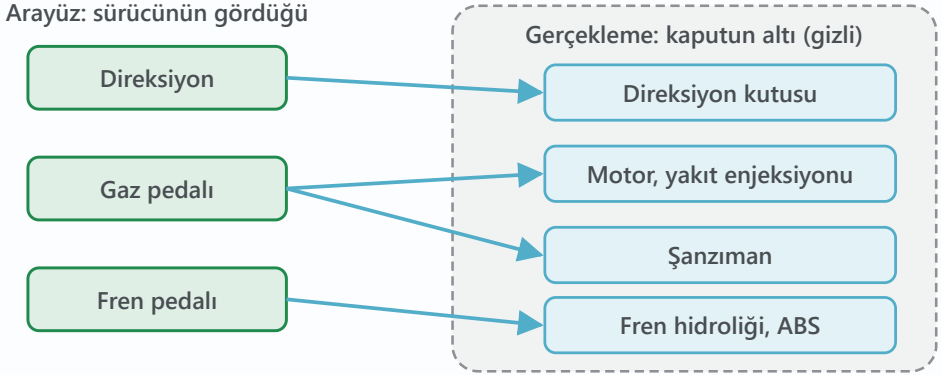
- **Sarmalama** (encapsulation): bir nesnenin **verisini** (alanlar) ve bu veriyi işleyen **metotları** tek birimde, sınıfta toplamak.
- Veri dışarıya **kapatılır** (`private`); dışarıdan yalnızca sınıfın sunduğu **metotlar** çağrılabilir.
- Benzetme: ilaç kapsülü. İçindeki toz görünmez, yalnızca kapsülü yutarsınız. Sınıf da kendi verisinin **bekçisidir**: geçersiz bir değer (ör. 75. saniye) içeri giremez.



Sarmalama, nesne yönelimli programlamanın dört temel ilkesinden biridir (diğerleri: kalıtım, çok biçimlilik, soyutlama).

Bilgi Saklama

- **Bilgi saklama** (information hiding), **soyutlama** (abstraction) olarak da adlandırılır. Tasarım, **ne** yapıldığını **nasıl** yapıldığından ayırmalıdır.
- Bir metot, iç ayrıntıları bilinmeden de kullanılabilmelidir: metodun **ne işe yaradığını** bilmek yeterli olmalıdır.
- Arabada fren, gaz pedalı ve direksiyonu görür, kullanırız; bu parçaların **iç mekaniğini** bilmeyiz.



Aynı fikir koddadır: `System.out.println` metodunu sürekli kullanırız ama ekrana yazmanın nasıl gerçekleştiğini bilmemiz gerekmez.

Bilgi Saklama: Kod

```
public class VeriIslemleri {
    private VeritabaniBaglantisi baglanti;
    // metodun iç yapısı bilinmeden de kullanılabilir.
    public void veriEkle(String veri) {
        baglanti = new VeritabaniBaglantisi();
        baglanti.baglan();
        baglanti.veriEkle(veri);
        baglanti.kapat();
    }
}
```

```
public class Ana {
    public static void main(String[] args) {
        VeriIslemleri islem = new VeriIslemleri();
        islem.veriEkle("Ayşe");
    }
}
```

ÇIKTI

```
Bağlantı açıldı
Eklendi: Ayşe
Bağlantı kapatıldı
```

- `Ana` sınıfı yalnızca `islem.veriEkle("Ayşe")` der: **ne** istediğini söyler.
- Bağlantıyı açmak, veriyi yazmak, kapatmak (**nasıl**) `VeriIslemleri` içinde gizlidir.
- `baglanti` alanı `private` : dışarıdan görülemez, değiştirilemez.
- `VeritabaniBaglantisi` burada ekrana yazan basit bir sınıftır; gerçek bir veri tabanı sürücüsü olsaydı bile `Ana` değişmezdi.

Tek Çağrının Arkası: Adım Adım

```
 1 public class VeriIslemleri {
 2   private VeritabaniBaglantisi baglanti;
 3   // metodun iç yapısı bilinmeden de
   kullanılabilir.
 4   public void veriEkle(String veri) {
 5     baglanti = new VeritabaniBaglantisi();
 6     baglanti.baglan();
 7     baglanti.veriEkle(veri);
 8     baglanti.kapat();
 9   }
10 }
11 public class Ana {
12   public static void main(String[] args) {
▶      VeriIslemleri islem = new VeriIslemleri();
14     islem.veriEkle("Ayşe");
15   }
16 }
```



`new VeriIslemleri()` heap'te bir nesne oluşturur; `islem` bu nesneyi **gösterir** (referans). `baglanti` henüz `null`.

Adım 1 / 7

ÇIKTI

ÇAĞRI YIĞINI

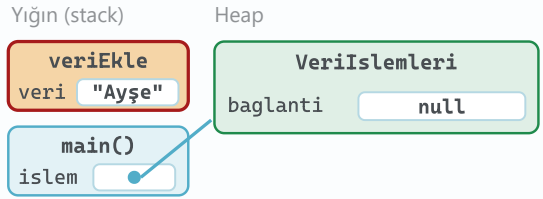
`main()`

DEĞİŞKENLER

`islem = →VeriIslemleri`

Tek Çağrının Arkası: Adım Adım

```
1  public class VeriIslemleri {
2    private VeritabaniBaglantisi baglanti;
3    // metodun iç yapısı bilinmeden de
   kullanılabilir.
▶    public void veriEkle(String veri) {
5      baglanti = new VeritabaniBaglantisi();
6      baglanti.baglan();
7      baglanti.veriEkle(veri);
8      baglanti.kapat();
9    }
10 }
11 public class Ana {
12   public static void main(String[] args) {
13     VeriIslemleri islem = new VeriIslemleri();
14     islem.veriEkle("Ayşe");
15   }
16 }
```



Ana'nın yaptığı tek şey bu çağrıdır. Yeni bir çerçeve açılır, `veri` parametresi "Ayşe" olur.

Adım 2 / 7

ÇIKTI

DEĞİŞKENLER

`veri = "Ayşe"`

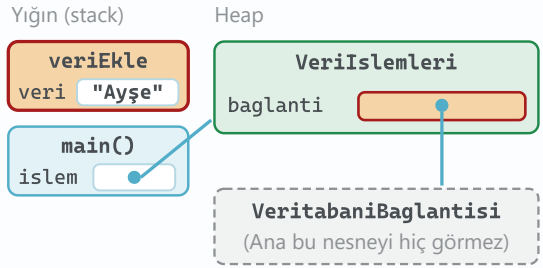
ÇAĞRI YIĞINI

`islem.veriEkle("Ayşe")`

`main()`

Tek Çağrının Arkası: Adım Adım

```
1  public class VeriIslemleri {
2    private VeritabaniBaglantisi baglanti;
3    // metodun iç yapısı bilinmeden de
   kullanılabilir.
4    public void veriEkle(String veri) {
▶      baglanti = new VeritabaniBaglantisi();
6      baglanti.baglan();
7      baglanti.veriEkle(veri);
8      baglanti.kapat();
9    }
10 }
11 public class Ana {
12   public static void main(String[] args) {
13     VeriIslemleri islem = new VeriIslemleri();
14     islem.veriEkle("Ayşe");
15   }
16 }
```



Gizli iş başlar: ikinci nesne oluşur, `private` `baglanti` onu gösterir.

Adım 3 / 7

ÇIKTI

DEĞİŞKENLER

`veri = "Ayşe"` `baglanti = →nesne`

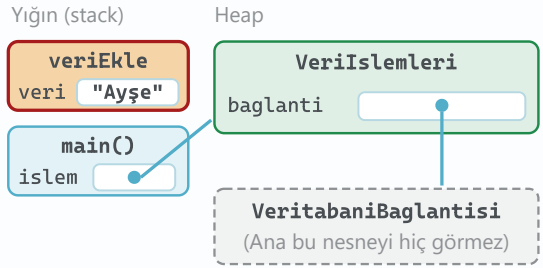
ÇAĞRI YIĞINI

`islem.veriEkle("Ayşe")`

`main()`

Tek Çağrının Arkası: Adım Adım

```
1  public class VeriIslemleri {
2    private VeritabaniBaglantisi baglanti;
3    // metodun iç yapısı bilinmeden de
   kullanılabilir.
4    public void veriEkle(String veri) {
5      baglanti = new VeritabaniBaglantisi();
▶      baglanti.baglan();
7      baglanti.veriEkle(veri);
8      baglanti.kapat();
9    }
10 }
11 public class Ana {
12   public static void main(String[] args) {
13     VeriIslemleri islem = new VeriIslemleri();
14     islem.veriEkle("Ayşe");
15   }
16 }
```



`baglan()` çağrılır: yığında üç çerçeve var. `Ana` bundan habersiz.

Adım 4 / 7

ÇIKTI

```
Bağlantı açıldı
```

DEĞİŞKENLER

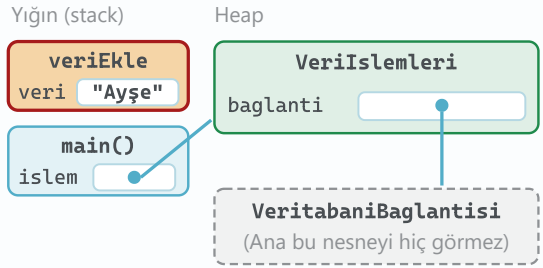
`veri = "Ayşe"`

ÇAĞRI YIĞINI

- `baglanti.baglan()`
- `islem.veriEkle("Ayşe")`
- `main()`

Tek Çağrının Arkası: Adım Adım

```
1  public class VeriIslemleri {
2    private VeritabaniBaglantisi baglanti;
3    // metodun iç yapısı bilinmeden de
   kullanılabilir.
4    public void veriEkle(String veri) {
5      baglanti = new VeritabaniBaglantisi();
6      baglanti.baglan();
▶      baglanti.veriEkle(veri);
8      baglanti.kapat();
9    }
10 }
11 public class Ana {
12   public static void main(String[] args) {
13     VeriIslemleri islem = new VeriIslemleri();
14     islem.veriEkle("Ayşe");
15   }
16 }
```



Veri, bağlantı nesnesinin `veriEkle` metoduna gider: aynı ad, **başka sınıf**.

Adım 5 / 7

ÇIKTI

```
Bağlantı açıldı
Eklendi: Ayşe
```

DEĞİŞKENLER

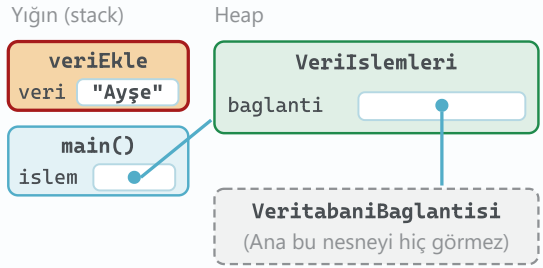
`veri = "Ayşe"`

ÇAĞRI YIĞINI

- **`baglanti.veriEkle(veri)`**
- `islem.veriEkle("Ayşe")`
- `main()`

Tek Çağrının Arkası: Adım Adım

```
 1 public class VeriIslemleri {
 2   private VeritabaniBaglantisi baglanti;
 3   // metodun iç yapısı bilinmeden de
   kullanılabilir.
 4   public void veriEkle(String veri) {
 5     baglanti = new VeritabaniBaglantisi();
 6     baglanti.baglan();
 7     baglanti.veriEkle(veri);
 ▶     baglanti.kapat();
 9   }
10 }
11 public class Ana {
12   public static void main(String[] args) {
13     VeriIslemleri islem = new VeriIslemleri();
14     islem.veriEkle("Ayşe");
15   }
16 }
```



Bağlantı kapatılır. Üç adımın hepsi tek çağrının arkasında saklı.

Adım 6 / 7

ÇIKTI

```
Bağlantı açıldı
Eklendi: Ayşe
Bağlantı kapatıldı
```

ÇAĞRI YIĞINI

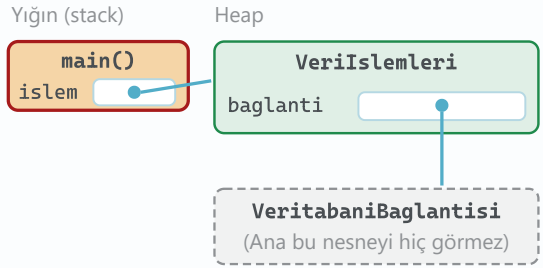
- `baglanti.kapat()`
- `islem.veriEkle("Ayşe")`
- `main()`

DEĞİŞKENLER

`veri = "Ayşe"`

Tek Çağrının Arkası: Adım Adım

```
1  public class VeriIslemleri {
2    private VeritabaniBaglantisi baglanti;
3    // metodun iç yapısı bilinmeden de
   kullanılabilir.
4    public void veriEkle(String veri) {
5      baglanti = new VeritabaniBaglantisi();
6      baglanti.baglan();
7      baglanti.veriEkle(veri);
8      baglanti.kapat();
9    }
10 }
11 public class Ana {
12   public static void main(String[] args) {
13     VeriIslemleri islem = new VeriIslemleri();
14     islem.veriEkle("Ayşe");
▶    }
16 }
```



`main` 'e dönülür. Kullanıcı tek satır yazdı, ayrıntıyı bilmedi: **bilgi saklama**.

Adım 7 / 7

ÇIKTI

```
Bağlantı açıldı
Eklendi: Ayşe
Bağlantı kapatıldı
```

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`islem = →VeriIslemleri`

Ön ve Son Koşullar

Ön koşul (precondition)

Metot çağrılmadan **önce** doğru olduğu varsayılan koşul. Sağlamak **çağıranın** sorumluluğudur.

```
public class VeriIslemleri {
    // Ön koşul: veri != null
    public void veriEkle(String veri) {
        // verinin null olup olmadığını kontrol et
        // ...
    }
}
```

Son koşul (postcondition)

Ön koşul sağlandıysa, metot bittikten **sonra** doğru olacağı garanti edilen durum. Sağlamak **metodun** sorumluluğudur.

```
public class VeriIslemleri {
    // Son koşul: veri tabanına veri eklendiyse true döner
    public boolean veriEkle(String veri) {
        // veriyi ekleyip eklemediğini kontrol et
        // ...
        return true;
    }
}
```

İkisi birlikte metodun **sözleşmesidir** (contract): "Sen bana boş olmayan bir veri ver, ben de onu ekleyip `true` döndüreyim." Kullanıcı yalnızca sözleşmeyi bilir, gövdeyi bilmez.

Sözleşme: Tam Örnek

```java
public class VeriIslemleri {
  private int kayitSayisi = 0;

  /**
   * Veri tabanına bir kayıt ekler.
   * Ön koşul: veri null değil ve boş değil.
   * Son koşul: eklendiyse true döner,
   *            kayitSayisi 1 artar.
   */
  public boolean veriEkle(String veri) {
    if (veri == null || veri.isEmpty()) {
      return false;   // ön koşul bozuk
    }
    // ... veriyi veri tabanına ekle ...
    kayitSayisi++;
    return true;      // son koşul sağlandı
  }

  public int getKayitSayisi() {
    return kayitSayisi;
  }
}
```

```java
// main içinde:
VeriIslemleri islem = new VeriIslemleri();
System.out.println(islem.veriEkle("Ali"));
System.out.println(islem.veriEkle(""));
System.out.println(islem.veriEkle(null));
System.out.println(islem.getKayitSayisi());
```

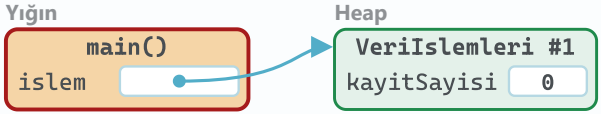
ÇIKTI

```
true
false
false
1
```

- `/** ... */` bir **Javadoc** yorumudur: metodun nasıl kullanılacağını anlatır.
- Ön koşul bozulursa metot hiçbir şey eklemez, `false` döndürür.

Sözleşme: Kod İzleme

```
 1 public boolean veriEkle(String veri) {
 2     if (veri == null || veri.isEmpty()) {
 3         return false;   // ön koşul bozuk
 4     }
 5     kayitSayisi++;
 6     return true;        // son koşul sağlandı
 7 }
 8 // main içinde:
▶ VeriIslemleri islem = new VeriIslemleri();
10 System.out.println(islem.veriEkle("Ayşe"));
11 System.out.println(islem.veriEkle(null));
12 System.out.println(islem.veriEkle(""));
13 System.out.println(islem.veriEkle("Can"));
14 System.out.println(islem.getKayitSayisi());
```



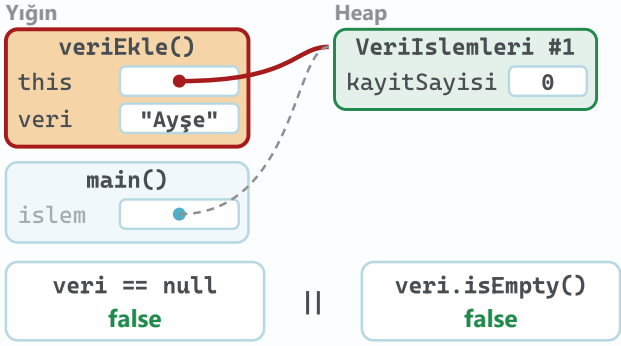
Nesne yeni oluştu: `kayitSayisi = 0`.
Sözleşme: `veri` null ve boş olmamalı.

Adım 1 / 8

ÇIKTI

Sözleşme: Kod İzleme

```
1  public boolean veriEkle(String veri) {
▶      if (veri == null || veri.isEmpty()) {
3          return false;   // ön koşul bozuk
4      }
5      kayitSayisi++;
6      return true;        // son koşul sağlandı
7  }
8  // main içinde:
9  VeriIslemleri islem = new VeriIslemleri();
▶  System.out.println(islem.veriEkle("Ayşe"));
11 System.out.println(islem.veriEkle(null));
12 System.out.println(islem.veriEkle(""));
13 System.out.println(islem.veriEkle("Can"));
14 System.out.println(islem.getKayitSayisi());
```



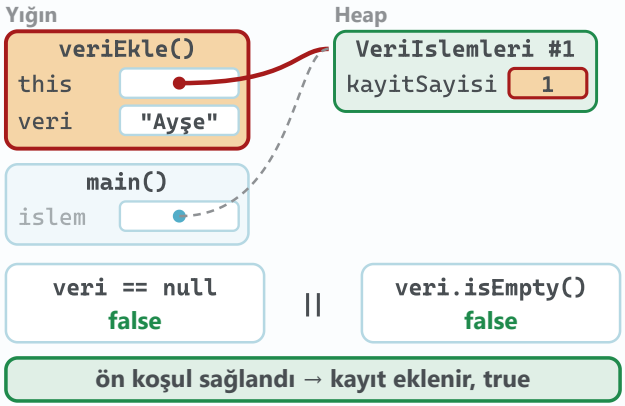
`"Ayşe"` null değil ve boş değil: koşul yanlış, `if` bloğu atlanır.

Adım 2 / 8

ÇIKTI

Sözleşme: Kod İzleme

```
1  public boolean veriEkle(String veri) {
2      if (veri == null || veri.isEmpty()) {
3          return false;   // ön koşul bozuk
4      }
▶      kayitSayisi++;
▶      return true;        // son koşul sağlandı
7  }
8  // main içinde:
9  VeriIslemleri islem = new VeriIslemleri();
10 System.out.println(islem.veriEkle("Ayşe"));
11 System.out.println(islem.veriEkle(null));
12 System.out.println(islem.veriEkle(""));
13 System.out.println(islem.veriEkle("Can"));
14 System.out.println(islem.getKayitSayisi());
```



Son koşul yerine getirilir: sayaç 1 artar ve **true** döner.

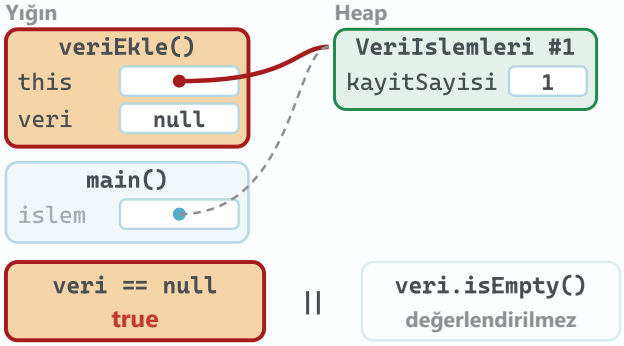
Adım 3 / 8

ÇIKTI

```
true
```

Sözleşme: Kod İzleme

```
1  public boolean veriEkle(String veri) {
▶      if (veri == null || veri.isEmpty()) {
3          return false;   // ön koşul bozuk
4      }
5      kayitSayisi++;
6      return true;        // son koşul sağlandı
7  }
8  // main içinde:
9  VeriIslemleri islem = new VeriIslemleri();
10 System.out.println(islem.veriEkle("Ayşe"));
▶  System.out.println(islem.veriEkle(null));
12 System.out.println(islem.veriEkle(""));
13 System.out.println(islem.veriEkle("Can"));
14 System.out.println(islem.getKayitSayisi());
```



`veri == null` doğru; `||` kısa devre yapar. `null.isEmpty()` hiç çağrılmaz, `NullPointerException` olmaz.

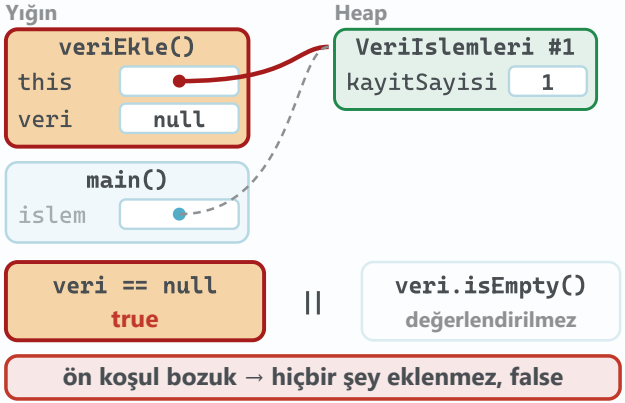
Adım 4 / 8

ÇIKTI

```
true
```

Sözleşme: Kod İzleme

```
1  public boolean veriEkle(String veri) {
2      if (veri == null || veri.isEmpty()) {
3          return false;   // ön koşul bozuk
4      }
5      kayitSayisi++;
6      return true;        // son koşul sağlandı
7  }
8  // main içinde:
9  VeriIslemleri islem = new VeriIslemleri();
10 System.out.println(islem.veriEkle("Ayşe"));
11 System.out.println(islem.veriEkle(null));
12 System.out.println(islem.veriEkle(""));
13 System.out.println(islem.veriEkle("Can"));
14 System.out.println(islem.getKayitSayisi());
```



Ön koşul bozuk: metot nesneye dokunmadan **false** döndürür.

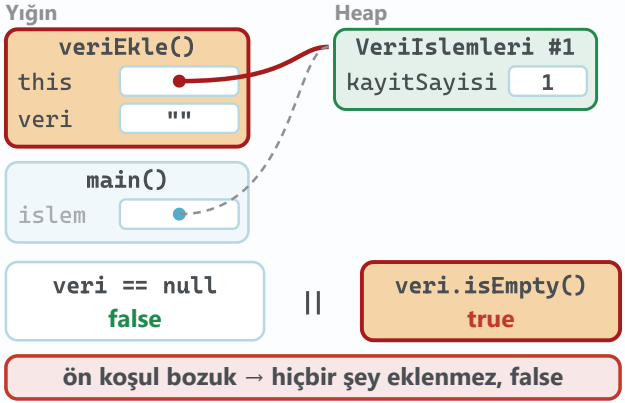
Adım 5 / 8

ÇIKTI

```
true
false
```

Sözleşme: Kod İzleme

```
1  public boolean veriEkle(String veri) {
▶      if (veri == null || veri.isEmpty()) {
▶          return false;   // ön koşul bozuk
4      }
5      kayitSayisi++;
6      return true;        // son koşul sağlandı
7  }
8  // main içinde:
9  VeriIslemleri islem = new VeriIslemleri();
10 System.out.println(islem.veriEkle("Ayşe"));
11 System.out.println(islem.veriEkle(null));
▶  System.out.println(islem.veriEkle(""));
13 System.out.println(islem.veriEkle("Can"));
14 System.out.println(islem.getKayitSayisi());
```



`""` null değil ama boş: sağ taraf doğru, yine **false**. Sayaç değişmez.

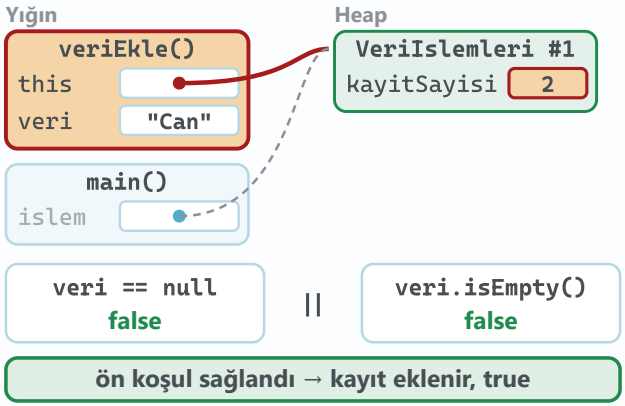
Adım 6 / 8

ÇIKTI

```
true
false
false
```

Sözleşme: Kod İzleme

```
1  public boolean veriEkle(String veri) {
2      if (veri == null || veri.isEmpty()) {
3          return false;   // ön koşul bozuk
4      }
▶      kayitSayisi++;
▶      return true;        // son koşul sağlandı
7  }
8  // main içinde:
9  VeriIslemleri islem = new VeriIslemleri();
10 System.out.println(islem.veriEkle("Ayşe"));
11 System.out.println(islem.veriEkle(null));
12 System.out.println(islem.veriEkle(""));
▶  System.out.println(islem.veriEkle("Can"));
14 System.out.println(islem.getKayitSayisi());
```



Geçerli veri: sayaç 2 olur, **true**.

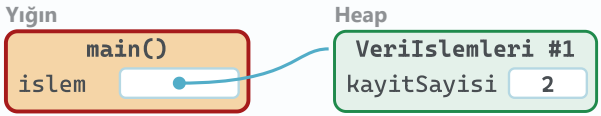
Adım 7 / 8

ÇIKTI

```
true
false
false
true
```

Sözleşme: Kod İzleme

```
1  public boolean veriEkle(String veri) {
2      if (veri == null || veri.isEmpty()) {
3          return false;   // ön koşul bozuk
4      }
5      kayitSayisi++;
6      return true;        // son koşul sağlandı
7  }
8  // main içinde:
9  VeriIslemleri islem = new VeriIslemleri();
10 System.out.println(islem.veriEkle("Ayşe"));
11 System.out.println(islem.veriEkle(null));
12 System.out.println(islem.veriEkle(""));
13 System.out.println(islem.veriEkle("Can"));
▶  System.out.println(islem.getKayitSayisi());
```



Dört çağrıdan yalnızca sözleşmeye uyan ikisi kayıt ekledi: **2**. Nesnenin durumu hiçbir çağrıda bozulmadı.

Adım 8 / 8

ÇIKTI

```
true
false
false
true
2
```

Ön Koşul Bozulunca Ne Yapmalı?

Yaklaşım	Örnek	Çağıran ne fark eder?
Sessizce yok saymak	`setSaniye(75)` hiçbir şey yapmaz	Hiçbir şey; hata gözden kaçabilir
Durum döndürmek	`veriEkle(null)` → `false`	Dönüş değerine bakarsa fark eder
İstisna fırlatmak	`throw new IllegalArgumentException(...)`	Yakalamazsa program durur; hata kaybolmaz

```java
public void setSaniye(int saniye) {
    if (saniye < 0 || saniye > 59)
        throw new IllegalArgumentException(
            "saniye 0-59 olmalı: " + saniye);
    this.saniye = saniye;
}
// main içinde:
DijitalSaat s = new DijitalSaat();
try {
    s.setSaniye(30);
    s.setSaniye(75);
    System.out.println("buraya gelinmez");
} catch (IllegalArgumentException e) {
    System.out.println("Hata: " + e.getMessage());
}
System.out.println(s.getSaniye());
```

```
Hata: saniye 0-59 olmalı: 75
30
```

- İstisna, hatalı değeri **çağrının yapıldığı yerde** görünür kılar; nesnenin durumu değişmez (30 kalır).
- `public` metotlarda argüman hatası için Java'nın alışılmış seçimi `IllegalArgumentException`'dır.
- Seçilen yaklaşım, sözleşmenin parçası olarak belgelenmelidir.

Sözleşmeyi Javadoc ile Yazmak

```java
/**
 * Hesaptan para çeker.
 * @param miktar çekilecek tutar; {@code 0 < miktar <= bakiye} olmalı
 * @return işlemden sonraki bakiye (eski bakiye - miktar)
 * @throws IllegalArgumentException miktar bu aralıkta değilse
 */
public double paraCek(double miktar) {
    if (miktar <= 0 || miktar > bakiye)
        throw new IllegalArgumentException("Geçersiz miktar: " + miktar);
    bakiye -= miktar;
    return bakiye;
}
```

Etiketler

`@param` : her parametre ve ön koşulu. `@return` : dönüş değeri, yani son koşul. `@throws` : sözleşme bozulunca ne olacağı.

Belge üretmek

`javadoc -d belge BankaHesabi.java` komutu bu yorumlardan HTML belgeler üretir. Java kütüphanesinin çevrim içi belgeleri de böyle oluşturulur.

Ne yazılır?

Metodun **ne** yaptığı ve kullanım koşulları. **Nasıl** yaptığı yazılmaz: o, gizlenen gerçeklemedir ve değişebilir.

Kimin için?

Gövdeyi hiç görmeyecek olan kullanıcı için. Sınıfın arayüzü, imzalar ve bu yorumların toplamıdır.

`assert` ile Değişmezi Denetlemek

```
class Sayac {
    private int deger;   // değişmez: 0 <= deger <= 9
    void arttir() {
        deger = deger + 1;   // hata: 9'dan sonra 10
        assert deger >= 0 && deger <= 9 : "deger=" + deger;
    }
}
// main içinde:
Sayac s = new Sayac();
for (int i = 0; i < 10; i++) s.arttir();
System.out.println("bitti");
```

`java Ana`

```
bitti
```

`java -ea Ana`

```
Exception in thread "main" java.lang.AssertionError:
deger=10
        at Sayac.arttir(Ana.java:5)
```

- `assert koşul : mesaj;` koşul yanlışsa `AssertionError` fırlatır. Sınıf değişmezini ve son koşulları **geliştirme sırasında** denetlemek içindir.
- Denetimler varsayılan olarak **kapalıdır**; `-ea` (enable assertions) seçeneğiyle açılır. Kapalıyken hata sessizce geçer: `bitti`.
- `public` metodun argümanları `assert` ile denetlenmez: kapalıyken denetim hiç yapılmaz. Argüman için istisna, iç tutarlılık için `assert`.

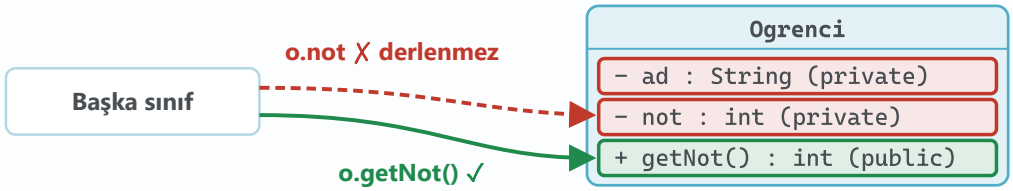
Erişim Belirleyiciler

- **Erişim belirleyici** (access modifier): bir sınıfa, alana veya metoda **kimin** erişebileceğini belirleyen anahtar sözcük.
- `public` : diğer sınıflar tarafından doğrudan erişilebilir.
- `private` : yalnızca sınıfın **içinden** erişilebilir; dışarıdan erişim derleyici tarafından engellenir.
- Sınıflar genellikle `public` tanımlanır.
- Sınıfın alanları genellikle `private` tanımlanır.
- Alanlara erişmek ve onları değiştirmek için `public` **metotlar** kullanılır.

```
public class Ogrenci {       // herkese açık sınıf
    private String ad;       // gizli alan
    private int not;         // gizli alan

    public int getNot() {    // açık metot
        return not;
    }
}
```

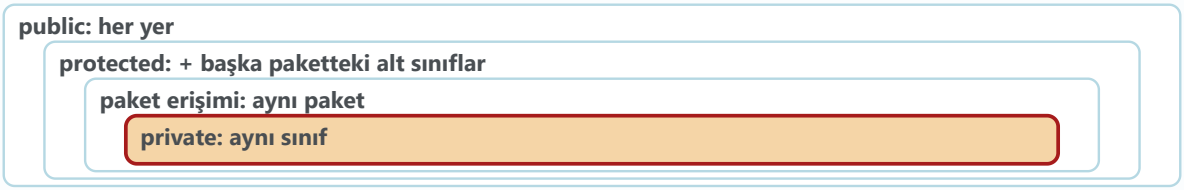
Kural: **alanlar gizli, davranışlar açık**.



Dört Erişim Düzeyi

Belirleyici	Aynı sınıf	Aynı paket	Alt sınıf (başka paket)	Her yer
`public`	✓	✓	✓	✓
`protected`	✓	✓	✓	✗
(yazılmazsa) paket erişimi	✓	✓	✗	✗
`private`	✓	✗	✗	✗

- Belirleyici yazılmazsa **paket erişimi** (package-private, "default") geçerlidir: yalnızca aynı paketteki sınıflar erişir.
- `protected` alt sınıflar içindir: başka paketteki alt sınıf da erişir.
- Kural olarak en kısıtlayıcı düzey tercih edilir: önce `private`, gerçekten gerekirse daha açık bir düzey.
- Üst düzey bir sınıf yalnızca `public` ya da paket erişimli olabilir.

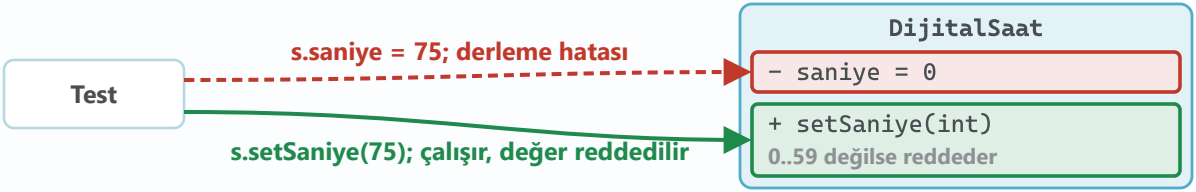


private Alana Erişim Denemesi

```
public class Test {
    public static void main(String[] args) {
        DijitalSaat s = new DijitalSaat();
        s.saniye = 75;
    }
}
```

```
Test.java:4: error: saniye has private access
in DijitalSaat
        s.saniye = 75;
         ^
1 error
```

- `saniye`, `DijitalSaat` içinde `private` tanımlıdır.
- Başka bir sınıftan `s.saniye` yazmak **derleme hatasıdır**: program hiç çalışmaz.
- Alan `public` olsaydı `s.saniye = 75;` sessizce kabul edilir, saat **geçersiz** bir duruma düşerdi.
- Çözüm: değişikliği sınıfın kendi metoduna yaptırmak: `s.setSaniye(75);` Metot geçersiz değeri reddeder.



`private` Sınıf Düzeyindedir

```
class Kesir {
    private int pay, payda;
    Kesir(int pay, int payda) {
        this.pay = pay;
        this.payda = payda;
    }
    Kesir carp(Kesir diger) {
        // diger'in private alanları: izinli
        return new Kesir(pay * diger.pay,
                         payda * diger.payda);
    }
    @Override public String toString() {
        return pay + "/" + payda;
    }
}
// main içinde:
Kesir k = new Kesir(2, 3).carp(new Kesir(3, 4));
System.out.println(k);
```


- Erişim denetimi **nesneye göre değil, sınıfa göre** yapılır: `Kesir`'in kodu, **her** `Kesir` nesnesinin `private` alanlarını okuyabilir.
- `carp`, `equals`, `compareTo` gibi iki nesneyi birlikte işleyen metotlar bu sayede getter gerektirmez.
- Başka bir sınıfta aynı satır derlenmez:

```
error: pay has private access in Kesir
```

Sarmalamanın sınırı sınıfın kaynak kodudur. Sınıfın içi tek bir yazarın sorumluluğundadır; korunan, dışarıdaki kullanıcılardır.

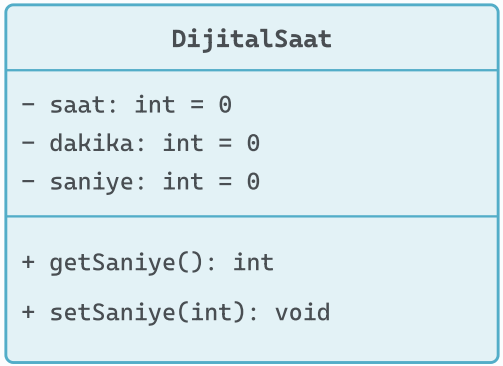
GETTER VE SETTER

DijitalSaat Sınıfı

```java
public class DijitalSaat {
    private int saat = 0;
    private int dakika = 0;
    private int saniye = 0;

    public int getSaniye() {
        return this.saniye;
    }

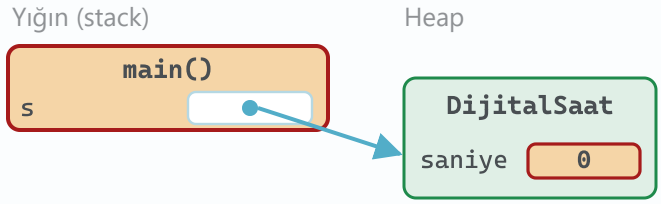
    public void setSaniye(int saniye) {
        if (saniye >= 0 && saniye < 60) {
            this.saniye = saniye;
        }
    }
}
```



- **Getter** (erişimci): alanın değerini döndürür.
- **Setter** (değiştirici): yeni değeri **önce kontrol eder**, geçerliyse atar.
- UML'de `-` private, `+` public demektir.

Setter Koruması: Adım Adım

```
1  public class DijitalSaat {
2    private int saniye = 0;
3    public int getSaniye() {
4      return this.saniye;
5    }
6    public void setSaniye(int saniye) {
7      if (saniye >= 0 && saniye < 60) {
8        this.saniye = saniye;
9      }
10   }
11 }
12 // SaatTest.main içinde:
▶ DijitalSaat s = new DijitalSaat();
14 s.setSaniye(45);
15 s.setSaniye(75);
16 System.out.println(s.getSaniye());
```



Nesne oluşturulur, `saniye` alanı **0** ile başlar. `s` nesneyi gösterir.

Adım 1 / 6

ÇIKTI

ÇAĞRI YIĞINI

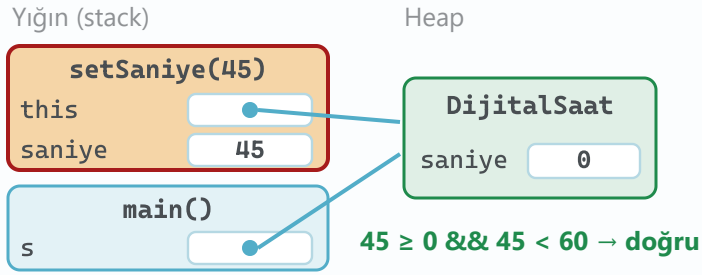
`main()`

DEĞİŞKENLER

`s = →DijitalSaat`

Setter Koruması: Adım Adım

```
1  public class DijitalSaat {
2    private int saniye = 0;
3    public int getSaniye() {
4      return this.saniye;
5    }
6    public void setSaniye(int saniye) {
▶      if (saniye >= 0 && saniye < 60) {
8        this.saniye = saniye;
9      }
10   }
11 }
12 // SaatTest.main içinde:
13 DijitalSaat s = new DijitalSaat();
14 s.setSaniye(45);
15 s.setSaniye(75);
16 System.out.println(s.getSaniye());
```



`setSaniye(45)` : parametre `saniye` = 45. `this` , metodu çağıran nesnedir. Koşul doğru.

Adım 2 / 6

ÇIKTI

ÇAĞRI YIĞINI

`s.setSaniye(45)`

`main()`

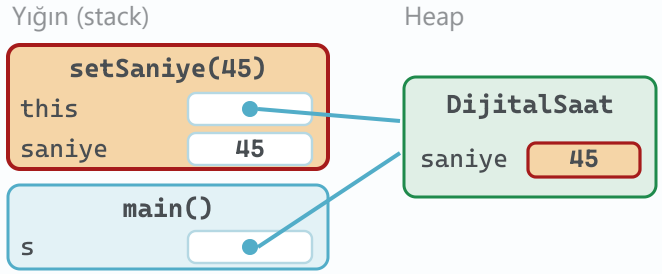
DEĞİŞKENLER

`this` = →nesne

`saniye` = 45

Setter Koruması: Adım Adım

```
1  public class DijitalSaat {
2    private int saniye = 0;
3    public int getSaniye() {
4      return this.saniye;
5    }
6    public void setSaniye(int saniye) {
7      if (saniye >= 0 && saniye < 60) {
▶        this.saniye = saniye;
9      }
10   }
11 }
12 // SaatTest.main içinde:
13 DijitalSaat s = new DijitalSaat();
14 s.setSaniye(45);
15 s.setSaniye(75);
16 System.out.println(s.getSaniye());
```



`this.saniye = saniye;` soldaki **nesnenin alanı**, sağdaki **parametre**dir. Alan 45 olur.

Adım 3 / 6

ÇIKTI

ÇAĞRI YIĞINI

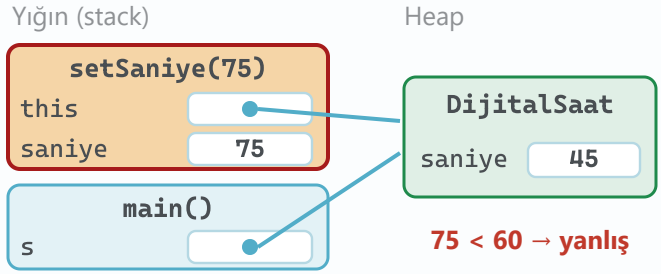
- `s.setSaniye(45)`
- `main()`

DEĞİŞKENLER

- `this = →nesne`
- `saniye = 45`

Setter Koruması: Adım Adım

```
1  public class DijitalSaat {
2    private int saniye = 0;
3    public int getSaniye() {
4      return this.saniye;
5    }
6    public void setSaniye(int saniye) {
▶      if (saniye >= 0 && saniye < 60) {
8        this.saniye = saniye;
9      }
10   }
11 }
12 // SaatTest.main içinde:
13 DijitalSaat s = new DijitalSaat();
14 s.setSaniye(45);
15 s.setSaniye(75);
16 System.out.println(s.getSaniye());
```



`setSaniye(75)` : koşul **yanlış**, atama satırı atlanır. Geçersiz değer nesneye **giremez**.

Adım 4 / 6

ÇIKTI

ÇAĞRI YIĞINI

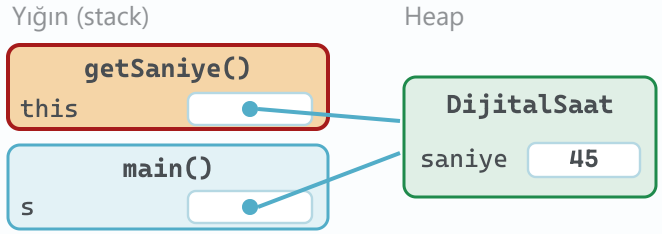
- `s.setSaniye(75)`
- `main()`

DEĞİŞKENLER

- `this = →nesne`
- `saniye = 75`

Setter Koruması: Adım Adım

```
1  public class DijitalSaat {
2    private int saniye = 0;
3    public int getSaniye() {
▶      return this.saniye;
5    }
6    public void setSaniye(int saniye) {
7      if (saniye >= 0 && saniye < 60) {
8        this.saniye = saniye;
9      }
10   }
11 }
12 // SaatTest.main içinde:
13 DijitalSaat s = new DijitalSaat();
14 s.setSaniye(45);
15 s.setSaniye(75);
16 System.out.println(s.getSaniye());
```



`getSaniye()` alanın değerini döndürür: 45. Alan hâlâ geçerli.

Adım 5 / 6

ÇIKTI

ÇAĞRI YIĞINI

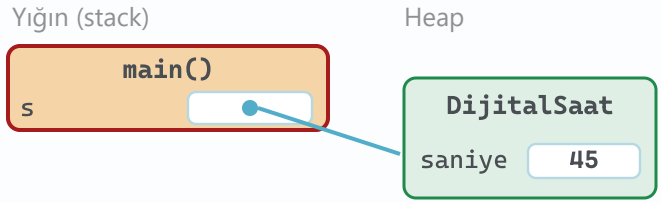
- `s.getSaniye()`
- `main()`

DEĞİŞKENLER

- `this = →nesne`
- `dönüş = 45`

Setter Koruması: Adım Adım

```
1  public class DijitalSaat {
2    private int saniye = 0;
3    public int getSaniye() {
4      return this.saniye;
5    }
6    public void setSaniye(int saniye) {
7      if (saniye >= 0 && saniye < 60) {
8        this.saniye = saniye;
9      }
10   }
11 }
12 // SaatTest.main içinde:
13 DijitalSaat s = new DijitalSaat();
14 s.setSaniye(45);
15 s.setSaniye(75);
▶  System.out.println(s.getSaniye());
```



Ekrana **45** yazılır. Nesnenin durumu baştan sona **geçerli** kaldı: sarmalamanın amacı budur.

Adım 6 / 6

ÇIKTI

```
45
```

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`s = →DijitalSaat`

GETTER VE SETTER

this ve Gölgeleme

Doğru

```
public void setSaniye(int saniye) {
    if (saniye >= 0 && saniye < 60) {
        this.saniye = saniye;
    }
}
```

Hatalı (ama derlenir!)

```
public class Kutu {
    private int en;
    public int getEn() {
        return en;
    }
    public void setEn(int en) {
        en = en;
    }
}
```

- Parametrenin adı alanla aynıysa, metot içinde `saniye` adı **parametreyi** gösterir: parametre alanı **gölgeler** (shadowing).
- `this`, metodu çağıran **nesnenin kendisidir**. `this.saniye` her zaman alandır.
- Hatalı sürümde `en = en;` parametreyi kendisine atar; alan değişmez. `k.setEn(5)` sonrası `k.getEn()`:

ÇIKTI

```
0
```

Derleyici hata vermez; hata ancak çalıştırınca fark edilir. Bu yüzden setter'larda `this.` alışkanlığı önemlidir.

GETTER VE SETTER

Erişim Metotları

```java
public class DijitalSaat {
  private int saat = 0;      // 0..23
  private int dakika = 0;    // 0..59
  private int saniye = 0;    // 0..59

  public int getSaat() { return saat; }
  public int getDakika() { return dakika; }
  public int getSaniye() { return saniye; }

  public void setSaat(int saat) {
    if (saat >= 0 && saat < 24) this.saat = saat;
  }
  public void setDakika(int dakika) {
    if (dakika >= 0 && dakika < 60) this.dakika = dakika;
  }
  public void setSaniye(int saniye) {
    if (saniye >= 0 && saniye < 60) this.saniye = saniye;
  }

  public void goster() {
    System.out.printf("%02d:%02d:%02d%n",
        saat, dakika, saniye);
  }
}
```

```java
DijitalSaat s = new DijitalSaat();
s.setSaat(9);
s.setDakika(5);
s.setSaniye(30);
s.goster();
s.setSaat(24);     // reddedilir
s.setDakika(-3);   // reddedilir
s.goster();
```

ÇIKTI

```
09:05:30
09:05:30
```

- Adlandırma: `dakika` → `getDakika()`, `setDakika(int)`.
- `boolean` alanda erişimci `is` ile başlar: `isAktif()`.

GETTER VE SETTER

Neden public Alan Değil?

public alan

```
public class Saat {
    public int saniye;
}
// başka bir sınıfta:
s.saniye = -5;   // kabul edilir!
```

- Her yerden, her değer yazılabilir.
- Kural (0–59) her kullanıcıya bırakılır; biri unutursa nesne bozulur.
- Alanın adı/tipi değişirse onu kullanan **tüm** sınıflar değişir.

private alan + metot

```
public class Saat {
    private int saniye;
    public void setSaniye(int s) {
        if (s >= 0 && s < 60) saniye = s;
    }
}
```

+ Kural **tek bir yerde**, sınıfın içinde kontrol edilir.
+ Salt okunur alan mümkün: yalnızca getter yazılır.
+ İç yapı değişse de metotların imzası aynı kalabilir.

Java 16+: yalnızca veri taşıyan değiştirilemez sınıflar için `record Nokta(int x, int y) { }` yazılabilir; alanlar otomatik `private final` olur, erişimciler `x()` ve `y()` adını alır.

GETTER VE SETTER

Setter Koruması: Etkileşimli Deneme

alan saniye değer 75 setter ile ata alan public olsaydı sıfırla

```
s.setSaat(9); s.setDakika(5); s.setSaniye(30);
```

saat	dakika	saniye
9	5	30

09:05:30

Bir alan ve değer seçip iki yolu karşılaştırın.

"setter ile ata" `s.setSaniye(75)` gibi çağırır: geçersiz değer reddedilir. "alan public olsaydı" `s.saniye = 75` gibi doğrudan yazar: nesne geçersiz duruma düşer (kırmızı). Geçerli aralıklar: saat 0–23, dakika ve saniye 0–59.

ÖRNEK: BANKA HESABI

BankaHesabi: Alanlar

```
public class BankaHesabi {
  private int hesapNumarasi;  // Hesap numarası
  private double bakiye;      // Hesap bakiyesi
  private String hesapTuru;   // "Vadesiz" / "Vadeli"

  public BankaHesabi(int hesapNumarasi,
      double bakiye, String hesapTuru) {
    this.hesapNumarasi = hesapNumarasi;
    this.bakiye = bakiye;
    this.hesapTuru = hesapTuru;
  }

  public int getHesapNumarasi() {
    return hesapNumarasi;
  }
  public double getBakiye() {
    return bakiye;
  }
  public String getHesapTuru() {
    return hesapTuru;
  }
  // ... para yatırma ve çekme metotları
```

- Üç alan da `private`: hesap numarasını veya bakiyeyi kimse doğrudan değiştiremez.
- **Yapıcı** (constructor) nesneyi ilk değerleriyle kurar; `this.bakiye = bakiye` kalıbı.
- Yalnızca getter var, setter yok: `setBakiye` olmadığı için bakiye yalnızca **işlemlerle** değişir.

ÖRNEK: BANKA HESABI

BankaHesabi: Yatırma ve Çekme

```
// ...
// Para yatırma işlemi
public void paraYatir(double miktar) {
  if (miktar > 0) {
    bakiye += miktar;
    System.out.println(miktar
        + " TL yatırıldı. Yeni bakiye: " + bakiye);
  } else {
    System.out.println("Yatırılan miktar geçersiz.");
  }
}

// Para çekme işlemi
public void paraCek(double miktar) {
  if (miktar <= bakiye && miktar > 0) {
    bakiye -= miktar;
    System.out.println(miktar
        + " TL çekildi. Kalan bakiye: " + bakiye);
  } else if (miktar > bakiye) {
    System.out.println("Yetersiz bakiye.");
  } else {
    System.out.println("Çekilen miktar geçersiz.");
  }
}
```

- Her işlem önce **ön koşulu** kontrol eder: `miktar > 0`, çekmede ayrıca `miktar ≤ bakiye`.
- Koşul bozuksa bakiye **değişmez**, yalnızca bir mesaj yazılır.
- Bakiye hiçbir yoldan eksiye düşemez: bu, sınıfın **değişmezidir**.

ÖRNEK: BANKA HESABI

BankaHesabi: Faiz

```
// ...
// Faiz ekleme (yalnızca vadeli hesap)
public void faizEkle(double oran) {
  if (hesapTuru.equals("Vadeli") && oran > 0) {
    double faiz = bakiye * oran / 100;
    bakiye += faiz;
    System.out.println("Faiz ekleniyor: " + faiz
        + " TL. Yeni bakiye: " + bakiye);
  } else if (!hesapTuru.equals("Vadeli")) {
    System.out.println(
        "Faiz yalnızca vadeli hesaplarda uygulanabilir.");
  } else {
    System.out.println("Geçersiz faiz oranı.");
  }
}
```

- Faiz yalnızca vadeli hesapta ve pozitif oranla eklenir.
- Metinler `==` ile değil, `equals` ile karşılaştırılır.
- `faiz` bir **yerel** değişkendir: yalnızca metot çalışırken vardır, nesnenin parçası değildir.

BankaHesabi: Transfer

```
  // ...
  // Başka bir hesaba transfer
  public void transfer(BankaHesabi hedefHesap,
                       double miktar) {
    if (miktar <= bakiye && miktar > 0) {
      bakiye -= miktar;
      hedefHesap.paraYatir(miktar);
      System.out.println(miktar
          + " TL transfer edildi. Yeni bakiye: " + bakiye);
    } else {
      System.out.println(
          "Yetersiz bakiye veya geçersiz miktar.");
    }
  }
}
```

- `transfer`, aynı sınıftan **başka bir nesnenin** metodunu çağırır: `hedefHesap` üzerinde `paraYatir`.
- Hedef hesabın `bakiye` alanına doğrudan yazmak da mümkündü (aynı sınıf, `private` sınıf düzeyindedir), ama metot çağırmak kuralların tek yerde kalmasını sağlar.

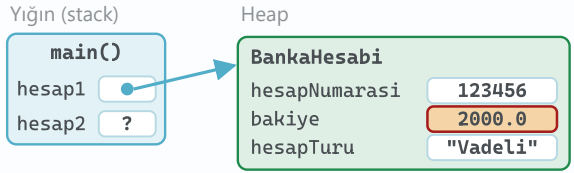


transfer(hesap2, 300.0): bakiye -= 300

paraYatir kendi kuralıyla: bakiye += 300

İki Hesap, Bir Transfer: Adım Adım

```
1  // main içinde:
▶  BankaHesabi hesap1 =
▶      new BankaHesabi(123456, 2000.0, "Vadeli");
4  BankaHesabi hesap2 =
5      new BankaHesabi(789012, 1500.0, "Vadesiz");
6  hesap1.paraYatir(500.0);
7  hesap2.paraCek(200.0);
8  hesap1.faizEkle(5.0);
9  hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```



Yapıcı çalışır: ilk nesne heap'te 2000.0 bakiye ile kurulur.

Adım 1 / 9

ÇIKTI

ÇAĞRI YIĞINI

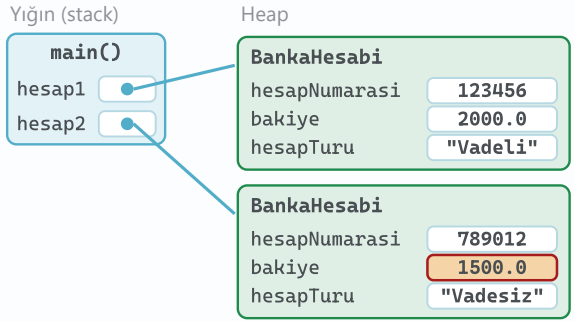
`main()`

DEĞİŞKENLER

`hesap1 = →BankaHesabi`

İki Hesap, Bir Transfer: Adım Adım

```
1  // main içinde:
2  BankaHesabi hesap1 =
3      new BankaHesabi(123456, 2000.0, "Vadeli");
▶  BankaHesabi hesap2 =
▶      new BankaHesabi(789012, 1500.0, "Vadesiz");
6  hesap1.paraYatir(500.0);
7  hesap2.paraCek(200.0);
8  hesap1.faizEkle(5.0);
9  hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```



İkinci, **bağımsız** nesne. Her nesnenin kendi `bakiye` alanı vardır.

Adım 2 / 9

ÇIKTI

ÇAĞRI YIĞINI

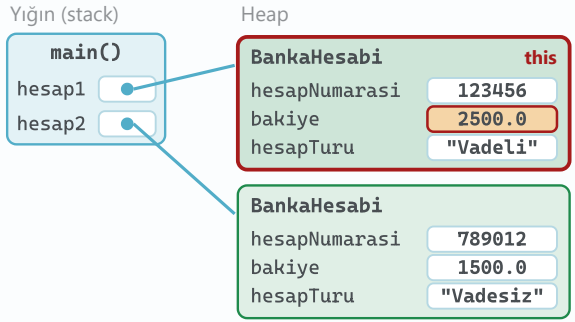
`main()`

DEĞİŞKENLER

`hesap1 = →BankaHesabi` `hesap2 = →BankaHesabi`

İki Hesap, Bir Transfer: Adım Adım

```
1  // main içinde:
2  BankaHesabi hesap1 =
3      new BankaHesabi(123456, 2000.0, "Vadeli");
4  BankaHesabi hesap2 =
5      new BankaHesabi(789012, 1500.0, "Vadesiz");
▶  hesap1.paraYatir(500.0);
7  hesap2.paraCek(200.0);
8  hesap1.faizEkle(5.0);
9  hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```



`this` = hesap1. `miktar > 0` doğru: `bakiye += miktar` → 2500.0.

Adım 3 / 9

ÇIKTI

```
500.0 TL yatırıldı. Yeni bakiye: 2500.0
```

ÇAĞRI YIĞINI

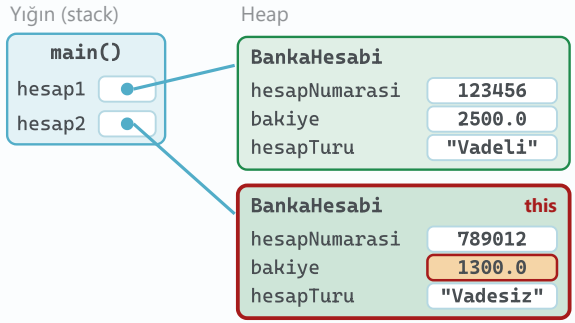
- `hesap1.paraYatir(500.0)`
- `main()`

DEĞİŞKENLER

`miktar = 500.0`

İki Hesap, Bir Transfer: Adım Adım

```
1  // main içinde:
2  BankaHesabi hesap1 =
3      new BankaHesabi(123456, 2000.0, "Vadeli");
4  BankaHesabi hesap2 =
5      new BankaHesabi(789012, 1500.0, "Vadesiz");
6  hesap1.paraYatir(500.0);
▶  hesap2.paraCek(200.0);
8  hesap1.faizEkle(5.0);
9  hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```



Bu kez `this` = hesap2. 200 ≤ 1500: bakiye 1300.0 olur; hesap1 etkilenmez.

Adım 4 / 9

ÇIKTI

```
500.0 TL yatırıldı. Yeni bakiye: 2500.0
200.0 TL çekildi. Kalan bakiye: 1300.0
```

ÇAĞRI YIĞINI

`hesap2.paraCek(200.0)`

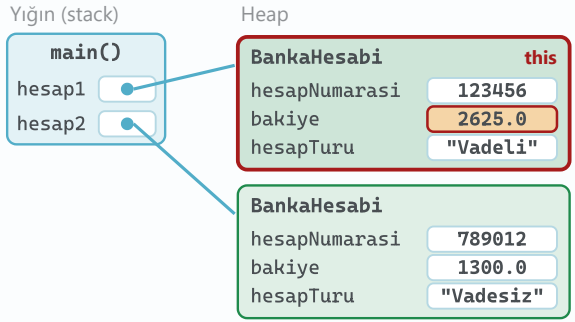
`main()`

DEĞİŞKENLER

`miktar = 200.0`

İki Hesap, Bir Transfer: Adım Adım

```
1  // main içinde:
2  BankaHesabi hesap1 =
3      new BankaHesabi(123456, 2000.0, "Vadeli");
4  BankaHesabi hesap2 =
5      new BankaHesabi(789012, 1500.0, "Vadesiz");
6  hesap1.paraYatir(500.0);
7  hesap2.paraCek(200.0);
▶ hesap1.faizEkle(5.0);
9  hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```



Hesap vadeli: `faiz = 2500.0 × 5 / 100 = 125.0`, bakiye 2625.0.

Adım 5 / 9

ÇIKTI

```
500.0 TL yatırıldı. Yeni bakiye: 2500.0
200.0 TL çekildi. Kalan bakiye: 1300.0
Faiz ekleniyor: 125.0 TL. Yeni bakiye: 2625.0
```

ÇAĞRI YIĞINI

`hesap1.faizEkle(5.0)`

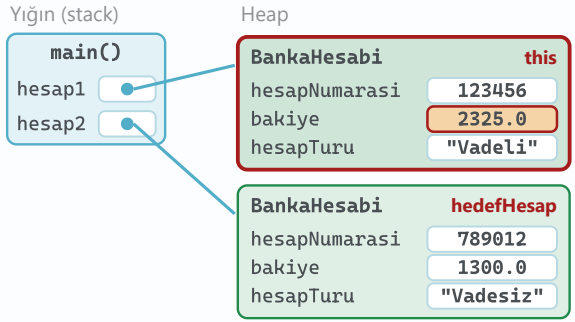
`main()`

DEĞİŞKENLER

`oran = 5.0` `faiz = 125.0`

İki Hesap, Bir Transfer: Adım Adım

```
1  // main içinde:
2  BankaHesabi hesap1 =
3      new BankaHesabi(123456, 2000.0, "Vadeli");
4  BankaHesabi hesap2 =
5      new BankaHesabi(789012, 1500.0, "Vadesiz");
6  hesap1.paraYatir(500.0);
7  hesap2.paraCek(200.0);
8  hesap1.faizEkle(5.0);
▶  hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```



`transfer` : koşul doğru, önce kendi bakiyesinden düşer: 2625.0 − 300.0 = 2325.0.

Adım 6 / 9

ÇIKTI

```
500.0 TL yatırıldı. Yeni bakiye: 2500.0
200.0 TL çekildi. Kalan bakiye: 1300.0
Faiz ekleniyor: 125.0 TL. Yeni bakiye: 2625.0
```

ÇAĞRI YIĞINI

`hesap1.transfer(hesap2, 300.0)`

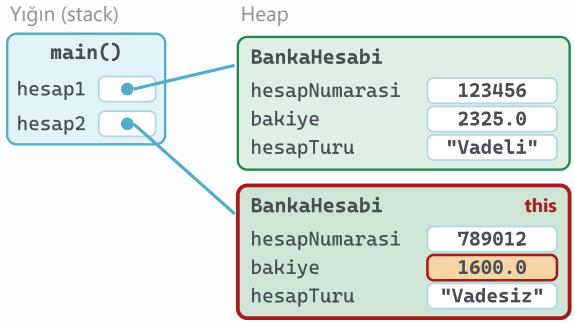
`main()`

DEĞİŞKENLER

`miktar = 300.0` `hedefHesap = →hesap2`

İki Hesap, Bir Transfer: Adım Adım

```
1  // main içinde:
2  BankaHesabi hesap1 =
3      new BankaHesabi(123456, 2000.0, "Vadeli");
4  BankaHesabi hesap2 =
5      new BankaHesabi(789012, 1500.0, "Vadesiz");
6  hesap1.paraYatir(500.0);
7  hesap2.paraCek(200.0);
8  hesap1.faizEkle(5.0);
▶  hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```



`hedefHesap.paraYatir(miktar)` : yığında 3 çerçeve; şimdi `this` = hesap2, bakiye 1600.0.

Adım 7 / 9

ÇIKTI

```
500.0 TL yatırıldı. Yeni bakiye: 2500.0
200.0 TL çekildi. Kalan bakiye: 1300.0
Faiz ekleniyor: 125.0 TL. Yeni bakiye: 2625.0
300.0 TL yatırıldı. Yeni bakiye: 1600.0
```

ÇAĞRI YIĞINI

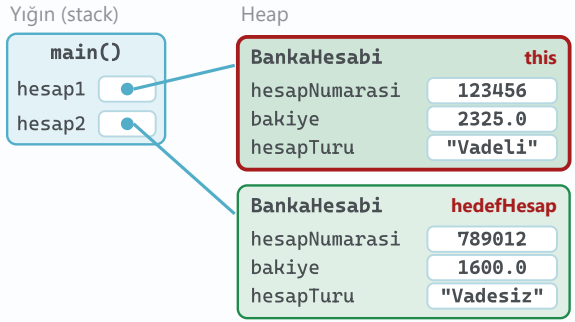
- `hesap2.paraYatir(300.0)`
- `hesap1.transfer(hesap2, 300.0)`
- `main()`

DEĞİŞKENLER

`miktar = 300.0`

İki Hesap, Bir Transfer: Adım Adım

```
1  // main içinde:
2  BankaHesabi hesap1 =
3      new BankaHesabi(123456, 2000.0, "Vadeli");
4  BankaHesabi hesap2 =
5      new BankaHesabi(789012, 1500.0, "Vadesiz");
6  hesap1.paraYatir(500.0);
7  hesap2.paraCek(200.0);
8  hesap1.faizEkle(5.0);
▶  hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```



`paraYatir` biter, `transfer` kaldığı yerden devam eder ve kendi mesajını yazar.

Adım 8 / 9

ÇIKTI

```
500.0 TL yatırıldı. Yeni bakiye: 2500.0
200.0 TL çekildi. Kalan bakiye: 1300.0
Faiz ekleniyor: 125.0 TL. Yeni bakiye: 2625.0
300.0 TL yatırıldı. Yeni bakiye: 1600.0
300.0 TL transfer edildi. Yeni bakiye: 2325.0
```

ÇAĞRI YIĞINI

`hesap1.transfer(hesap2, 300.0)`

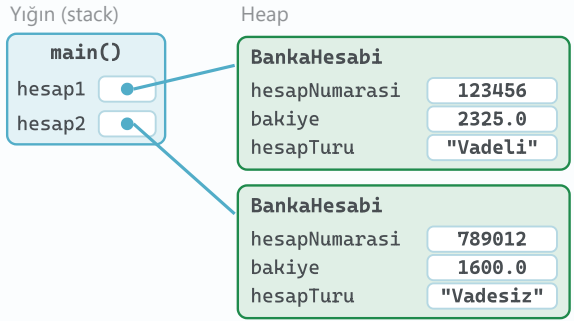
`main()`

DEĞİŞKENLER

`miktar = 300.0`

İki Hesap, Bir Transfer: Adım Adım

```
1 // main içinde:
2 BankaHesabi hesap1 =
3     new BankaHesabi(123456, 2000.0, "Vadeli");
4 BankaHesabi hesap2 =
5     new BankaHesabi(789012, 1500.0, "Vadesiz");
6 hesap1.paraYatir(500.0);
7 hesap2.paraCek(200.0);
8 hesap1.faizEkle(5.0);
9 hesap1.transfer(hesap2, 300.0);
▶ System.out.println(hesap1.getBakiye());
▶ System.out.println(hesap2.getBakiye());
```



Getter'lar son bakiyeleri okur. Toplam para korunur: 2325 + 1600 = 3925 = 2625 + 1300.

Adım 9 / 9

ÇIKTI

```
500.0 TL yatırıldı. Yeni bakiye: 2500.0
200.0 TL çekildi. Kalan bakiye: 1300.0
Faiz ekleniyor: 125.0 TL. Yeni bakiye: 2625.0
300.0 TL yatırıldı. Yeni bakiye: 1600.0
300.0 TL transfer edildi. Yeni bakiye: 2325.0
2325.0
1600.0
```

ÇAĞRI YIĞINI

`main()`

DEĞİŞKENLER

`hesap1 = →BankaHesabi` `hesap2 = →BankaHesabi`

ÖRNEK: BANKA HESABI

Yapıcı da Doğrulamalı

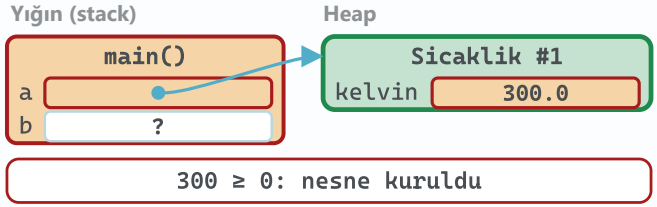
- Setter'lar ve işlem metotları bakiyeyi korur; ama örnekteki yapıcı gelen değeri **denetlemeden** atar.
- `new BankaHesabi(1, -500.0, "Vadesiz")` eksi bakiyeli bir hesap kurar; `"vadeli"` (küçük harf) yazılırsa hesap faiz alamaz ve kimse fark etmez.
- Sınıf değişmezi nesnenin **doğduğu anda** sağlanmalıdır: yapıcı da bir ön koşul denetleyicisidir.
- Yapıcı istisna fırlatırsa `new` ifadesi hiçbir referans üretmez: geçersiz nesne kullanıcının eline hiç geçmez.

```
public BankaHesabi(int hesapNumarasi,
        double bakiye, String hesapTuru) {
    if (bakiye < 0)
        throw new IllegalArgumentException(
            "Başlangıç bakiyesi negatif olamaz");
    if (!hesapTuru.equals("Vadeli")
            && !hesapTuru.equals("Vadesiz"))
        throw new IllegalArgumentException(
            "Bilinmeyen hesap türü: " + hesapTuru);
    this.hesapNumarasi = hesapNumarasi;
    this.bakiye = bakiye;
    this.hesapTuru = hesapTuru;
}
```

Yapıcı, nesnenin kurulduğu tek kapıdır: kurallar orada bir kez uygulanırsa sınıfın öteki metotları gelen nesnenin geçerli olduğuna güvenebilir.

Yapıcıda Doğrulama: Kod İzleme

```
 1 class Sicaklik {
 2     private final double kelvin;
 3     public Sicaklik(double kelvin) {
 4         if (kelvin < 0)
 5             throw new IllegalArgumentException(
 6                 "Kelvin negatif olamaz");
 7         this.kelvin = kelvin;
 8     }
 9     public double getKelvin() { return kelvin; }
10 }
11 // main içinde:
▶ Sicaklik a = new Sicaklik(300);
13 Sicaklik b = null;
14 try {
15     b = new Sicaklik(-5);
16 } catch (IllegalArgumentException e) {
17     System.out.println("Reddedildi: "
18         + e.getMessage());
19 }
20 System.out.println(a.getKelvin() + " " + b);
```



300 geçerli: koşul yanlış, `kelvin` atanır ve `a` yeni nesneyi gösterir.

Adım 1 / 6

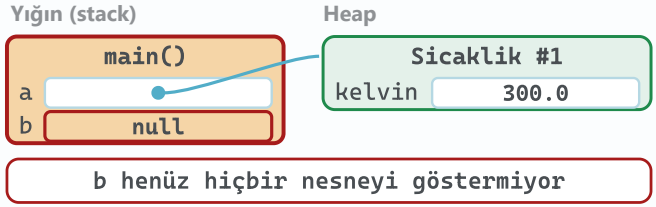
ÇIKTI

DEĞİŞKENLER

`a = →#1`

Yapıcıda Doğrulama: Kod İzleme

```
 1 class Sicaklik {
 2     private final double kelvin;
 3     public Sicaklik(double kelvin) {
 4         if (kelvin < 0)
 5             throw new IllegalArgumentException(
 6                 "Kelvin negatif olamaz");
 7         this.kelvin = kelvin;
 8     }
 9     public double getKelvin() { return kelvin; }
10 }
11 // main içinde:
12 Sicaklik a = new Sicaklik(300);
▶ Sicaklik b = null;
14 try {
15     b = new Sicaklik(-5);
16 } catch (IllegalArgumentException e) {
17     System.out.println("Reddedildi: "
18         + e.getMessage());
19 }
20 System.out.println(a.getKelvin() + " " + b);
```



b henüz hiçbir nesneyi göstermiyor

`b` açıkça `null` ile başlatılır; `catch` sonrasında da okunabilmesi için.

Adım 2 / 6

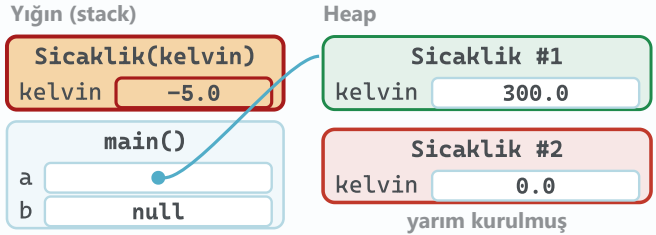
ÇIKTI

DEĞİŞKENLER

`a = →#1` `b = null`

Yapıcıda Doğrulama: Kod İzleme

```
 1 class Sicaklik {
 2     private final double kelvin;
 3     public Sicaklik(double kelvin) {
 4         if (kelvin < 0)
 5             throw new IllegalArgumentException(
 6                 "Kelvin negatif olamaz");
 7         this.kelvin = kelvin;
 8     }
 9     public double getKelvin() { return kelvin; }
10 }
11 // main içinde:
12 Sicaklik a = new Sicaklik(300);
13 Sicaklik b = null;
14 try {
15     b = new Sicaklik(-5);
16 } catch (IllegalArgumentException e) {
17     System.out.println("Reddedildi: "
18         + e.getMessage());
19 }
20 System.out.println(a.getKelvin() + " " + b);
```



`new yer ayırdı, yapıcı çalışıyor`

`new` ikinci nesneye yer ayırır (`kelvin` varsayılan 0.0) ve yapıcıyı çağırır. Parametre −5.0.

Adım 3 / 6

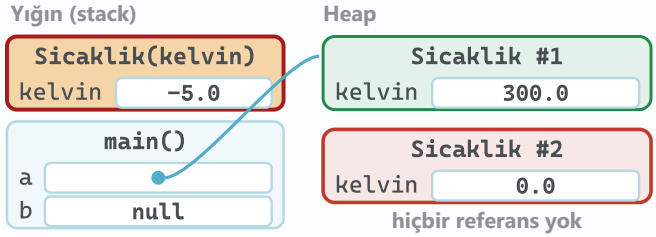
ÇIKTI

DEĞİŞKENLER

`a = →#1` `b = null` `kelvin = -5.0`

Yapıcıda Doğrulama: Kod İzleme

```
1  class Sicaklik {
2      private final double kelvin;
3      public Sicaklik(double kelvin) {
▶          if (kelvin < 0)
▶              throw new IllegalArgumentException(
6                  "Kelvin negatif olamaz");
7          this.kelvin = kelvin;
8      }
9      public double getKelvin() { return kelvin; }
10 }
11 // main içinde:
12 Sicaklik a = new Sicaklik(300);
13 Sicaklik b = null;
14 try {
15     b = new Sicaklik(-5);
16 } catch (IllegalArgumentException e) {
17     System.out.println("Reddedildi: "
18         + e.getMessage());
19 }
20 System.out.println(a.getKelvin() + " " + b);
```



`throw: yapıcı yarıda kesilir`

Ön koşul bozuk: `throw` yapıcıyı burada keser. Atama satırı hiç çalışmaz.

Adım 4 / 6

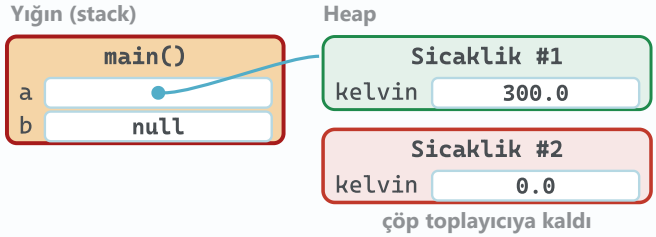
ÇIKTI

DEĞİŞKENLER

`a = →#1` `b = null` `kelvin = -5.0`

Yapıcıda Doğrulama: Kod İzleme

```
1  class Sicaklik {
2      private final double kelvin;
3      public Sicaklik(double kelvin) {
4          if (kelvin < 0)
5              throw new IllegalArgumentException(
6                  "Kelvin negatif olamaz");
7          this.kelvin = kelvin;
8      }
9      public double getKelvin() { return kelvin; }
10 }
11 // main içinde:
12 Sicaklik a = new Sicaklik(300);
13 Sicaklik b = null;
14 try {
15     b = new Sicaklik(-5);
▶ } catch (IllegalArgumentException e) {
▶     System.out.println("Reddedildi: "
▶         + e.getMessage());
19 }
20 System.out.println(a.getKelvin() + " " + b);
```



`atama yapılmadı: b hâlâ null`

Kontrol `catch` bloğuna geçer. `b = ...` ataması yarıda kaldığı için `b` değişmedi; yarım nesneye ulaşan hiçbir referans yok.

Adım 5 / 6

ÇIKTI

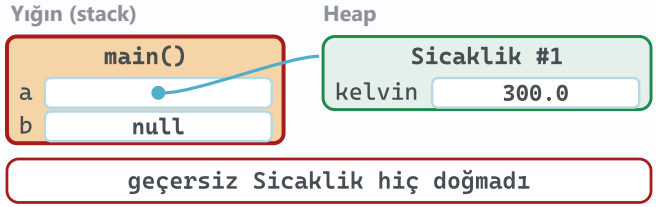
```
Reddedildi: Kelvin negatif olamaz
```

DEĞİŞKENLER

`a = →#1` `b = null`

Yapıcıda Doğrulama: Kod İzleme

```
1  class Sicaklik {
2      private final double kelvin;
3      public Sicaklik(double kelvin) {
4          if (kelvin < 0)
5              throw new IllegalArgumentException(
6                  "Kelvin negatif olamaz");
7          this.kelvin = kelvin;
8      }
9      public double getKelvin() { return kelvin; }
10 }
11 // main içinde:
12 Sicaklik a = new Sicaklik(300);
13 Sicaklik b = null;
14 try {
15     b = new Sicaklik(-5);
16 } catch (IllegalArgumentException e) {
17     System.out.println("Reddedildi: "
18         + e.getMessage());
19 }
▶  System.out.println(a.getKelvin() + " " + b);
```



Programda yalnızca geçerli `Sicaklik` nesneleri kullanılabilir: değişmez (`kelvin ≥ 0`) her nesnede doğrudur.

Adım 6 / 6

ÇIKTI

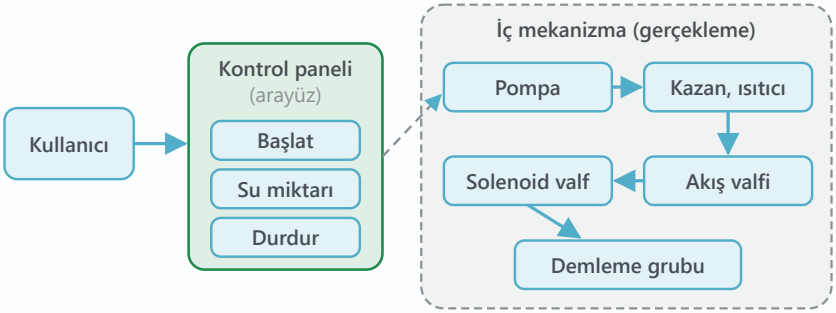
```
Reddedildi: Kelvin negatif olamaz
300.0 null
```

DEĞİŞKENLER

`a = →#1` `b = null`

Sınıf Tasarımı

- Sınıf tanımı iki kısma ayrılır: **arayüz** (interface) ve **gerçekleme** (implementation).
- Arayüz, sınıfın **ne yaptığını** ve **nasıl kullanılacağını** belirtir: public metotların adları, parametreleri, dönüş tipleri ve yorumları.
- Gerçekleme, sınıfın iç mekanik ayrıntılarını ve özel (private) alanlarını içerir. Bunlar yalnızca sınıfın içinde önemlidir.
- **Gerçekleme değişse bile arayüz değişmeyebilir**; öz alanlar sınıfın dışına gizlidir.
- Üç unsur: **tanımlama** (specification: yapıcı ve metotların imzaları), **temsil** (representation: veriyi tutan `private` alanlar) ve **gerçekleme** (implementation: yapıcı ve metot gövdeleri). Kullanıcı yalnızca tanımlamayı bilir; temsil ve gerçekleme değişse de kodu bozulmaz.

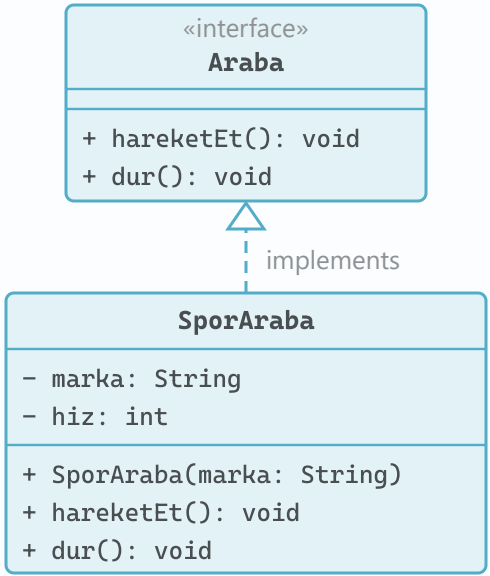


Arayüz

```java
public interface Araba {
    // Sınıf arayüzü: NE yapılır?
    public void hareketEt();

    public void dur();
}
```

- Java'da `interface`, bir sınıfın **ne** yapacağını söyleyen sözleşmedir: metotların yalnızca imzası vardır, gövdesi yoktur.
- Arabayı kullanan kişi yalnızca `hareketEt()` ve `dur()` metotlarını bilir.



SINIF TASARIMI

Sınıf Gerçeklemesi

```java
public class SporAraba implements Araba {
  // Sınıf gerçeklemesi: NASIL yapılır?
  private String marka;
  private int hiz;

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

  public void hareketEt() {
    hiz = 100;
    System.out.println(marka + " hareket ediyor: "
        + hiz + " km/sa");
  }

  public void dur() {
    hiz = 0;
    System.out.println(marka + " durdu.");
  }
}
```

```java
// main içinde:
Araba araba = new SporAraba("Rüzgar");
araba.hareketEt();
araba.dur();
```

ÇIKTI

```
Rüzgar hareket ediyor: 100 km/sa
Rüzgar durdu.
```

- `implements Araba` : sınıf, arayüzdeki **her** metodu yazmak zorundadır.
- Arayüz ile sınıf aynı pakette **aynı adı** taşıyamaz (`duplicate class: Araba`); bu yüzden sınıfın adı `SporAraba` .
- `Ana` değişkeni `Araba` tipinde tanımlar: yalnızca arayüze bağımlıdır.

Gerçeklemeyi Değiştirmek

Eski: iki ayrı alan

```
public class DijitalSaat {
  private int dakika = 0;
  private int saniye = 0;
  public int getDakika() {
    return dakika;
  }
  public int getSaniye() {
    return saniye;
  }
  public void setDakika(int dakika) {
    if (dakika >= 0 && dakika < 60) {
      this.dakika = dakika;
    }
  }
  public void setSaniye(int saniye) {
    if (saniye >= 0 && saniye < 60) {
      this.saniye = saniye;
    }
  }
}
```

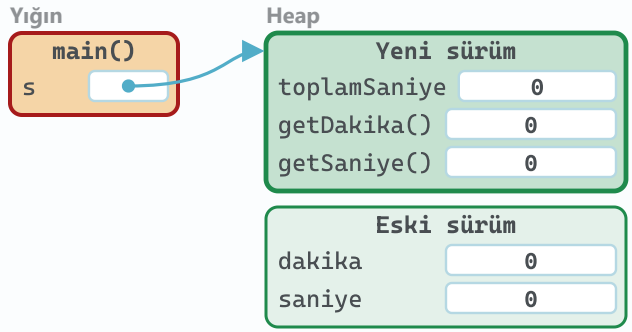
Yeni: tek alan

```
public class DijitalSaat {
  // gece yarısından beri geçen saniye
  private int toplamSaniye = 0;
  public int getDakika() {
    return toplamSaniye / 60 % 60;
  }
  public int getSaniye() {
    return toplamSaniye % 60;
  }
  public void setDakika(int dakika) {
    if (dakika >= 0 && dakika < 60) {
      int fark = dakika - getDakika();
      toplamSaniye += fark * 60;
    }
  }
  public void setSaniye(int saniye) {
    if (saniye >= 0 && saniye < 60) {
      toplamSaniye += saniye - getSaniye();
    }
  }
}
```

Kullanan kod değişmez: `setDakika(12); setSaniye(45); setSaniye(75);` sonrası iki sürüm de **12:45** verir.

Aynı Arayüz, Yeni Gerçekleme: Kod İzleme

```
 1 private int toplamSaniye = 0;  // tek alan
 2 public int getSaniye() {
 3     return toplamSaniye % 60;
 4 }
 5 public void setSaniye(int saniye) {
 6     if (saniye >= 0 && saniye < 60)
 7         toplamSaniye += saniye - getSaniye();
 8 }
 9 // getDakika(): toplamSaniye / 60 % 60
▶ DijitalSaat s = new DijitalSaat(); // main
11 s.setDakika(12);
12 s.setSaniye(45);
13 s.setSaniye(75);
14 s.setSaniye(9);
15 System.out.println(s.getDakika() + ":"
16     + s.getSaniye());
```



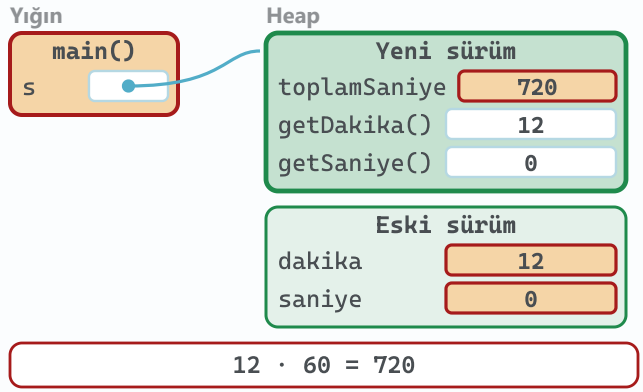
Yeni sürüm tek bir alan tutar; eski sürüm (karşılaştırma için) iki alan. Dışarıdan ikisi de 00:00.

Adım 1 / 7

ÇIKTI

Aynı Arayüz, Yeni Gerçekleme: Kod İzleme

```
 1 private int toplamSaniye = 0;  // tek alan
 2 public int getSaniye() {
 3     return toplamSaniye % 60;
 4 }
 5 public void setSaniye(int saniye) {
 6     if (saniye >= 0 && saniye < 60)
 7         toplamSaniye += saniye - getSaniye();
 8 }
 9 // getDakika(): toplamSaniye / 60 % 60
10 DijitalSaat s = new DijitalSaat(); // main
▶ s.setDakika(12);
12 s.setSaniye(45);
13 s.setSaniye(75);
14 s.setSaniye(9);
15 System.out.println(s.getDakika() + ":"
16     + s.getSaniye());
```



`12 · 60 = 720`

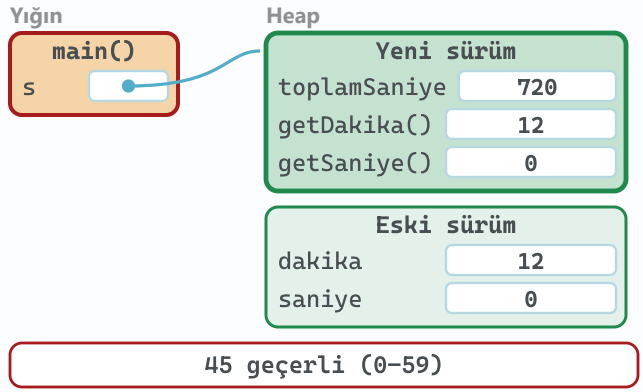
`setDakika(12)` : fark 12 dakika, 12 · 60 = 720 saniye eklenir. Eski sürüm doğrudan `dakika = 12` yazar.

Adım 2 / 7

ÇIKTI

Aynı Arayüz, Yeni Gerçekleme: Kod İzleme

```
 1 private int toplamSaniye = 0;  // tek alan
 2 public int getSaniye() {
 3     return toplamSaniye % 60;
 4 }
 ▶ public void setSaniye(int saniye) {
 ▶     if (saniye >= 0 && saniye < 60)
 7         toplamSaniye += saniye - getSaniye();
 8 }
 9 // getDakika(): toplamSaniye / 60 % 60
10 DijitalSaat s = new DijitalSaat(); // main
11 s.setDakika(12);
 ▶ s.setSaniye(45);
13 s.setSaniye(75);
14 s.setSaniye(9);
15 System.out.println(s.getDakika() + ":"
16     + s.getSaniye());
```



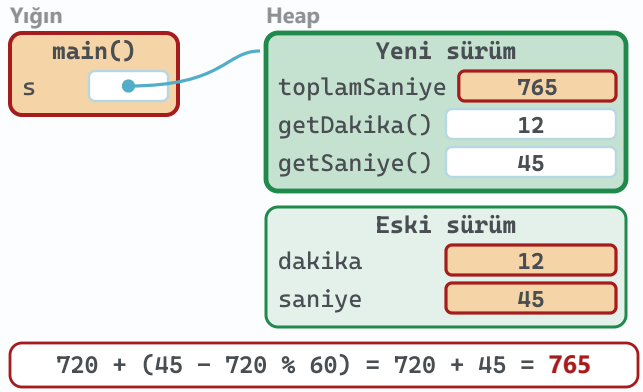
`setSaniye(45)` : değer aralıkta, koşul doğru.

Adım 3 / 7

ÇIKTI

Aynı Arayüz, Yeni Gerçekleme: Kod İzleme

```
 1 private int toplamSaniye = 0;  // tek alan
 2 public int getSaniye() {
 3     return toplamSaniye % 60;
 4 }
 5 public void setSaniye(int saniye) {
 6     if (saniye >= 0 && saniye < 60)
 7         toplamSaniye += saniye - getSaniye();
 8 }
 9 // getDakika(): toplamSaniye / 60 % 60
10 DijitalSaat s = new DijitalSaat(); // main
11 s.setDakika(12);
12 s.setSaniye(45);
13 s.setSaniye(75);
14 s.setSaniye(9);
15 System.out.println(s.getDakika() + ":"
16     + s.getSaniye());
```



`720 + (45 − 720 % 60) = 720 + 45 = 765`

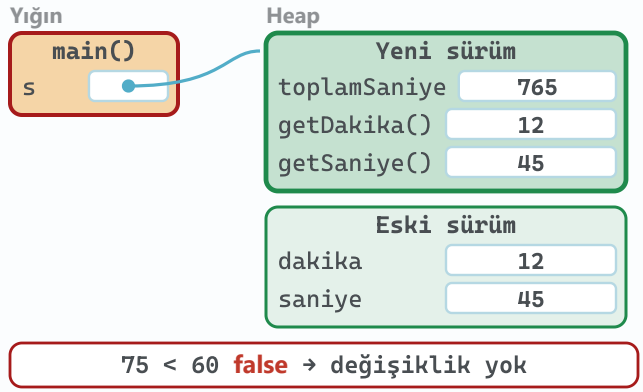
Önce `getSaniye()` = 720 % 60 = 0; fark 45 eklenir: **765**. Dışarıdan bakan için 12:45.

Adım 4 / 7

ÇIKTI

Aynı Arayüz, Yeni Gerçekleme: Kod İzleme

```
 1 private int toplamSaniye = 0;  // tek alan
 2 public int getSaniye() {
 3     return toplamSaniye % 60;
 4 }
 5 public void setSaniye(int saniye) {
 6     if (saniye >= 0 && saniye < 60)
 7         toplamSaniye += saniye - getSaniye();
 8 }
 9 // getDakika(): toplamSaniye / 60 % 60
10 DijitalSaat s = new DijitalSaat(); // main
11 s.setDakika(12);
12 s.setSaniye(45);
13 s.setSaniye(75);
14 s.setSaniye(9);
15 System.out.println(s.getDakika() + ":"
16     + s.getSaniye());
```



`75 < 60` **false** → değişiklik yok

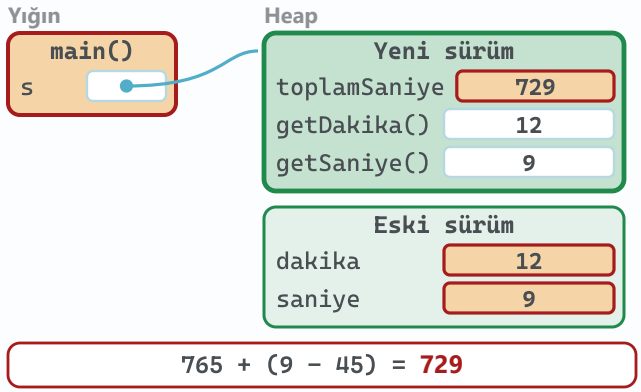
75 geçersiz: iki sürüm de reddeder. Kural (0–59) sınıfın içinde kaldı.

Adım 5 / 7

ÇIKTI

Aynı Arayüz, Yeni Gerçekleme: Kod İzleme

```
 1 private int toplamSaniye = 0;  // tek alan
 2 public int getSaniye() {
 3     return toplamSaniye % 60;
 4 }
 5 public void setSaniye(int saniye) {
 6     if (saniye >= 0 && saniye < 60)
▶          toplamSaniye += saniye - getSaniye();
 8 }
 9 // getDakika(): toplamSaniye / 60 % 60
10 DijitalSaat s = new DijitalSaat(); // main
11 s.setDakika(12);
12 s.setSaniye(45);
13 s.setSaniye(75);
▶ s.setSaniye(9);
15 System.out.println(s.getDakika() + ":"
16     + s.getSaniye());
```



765 + (9 − 45) = **729**

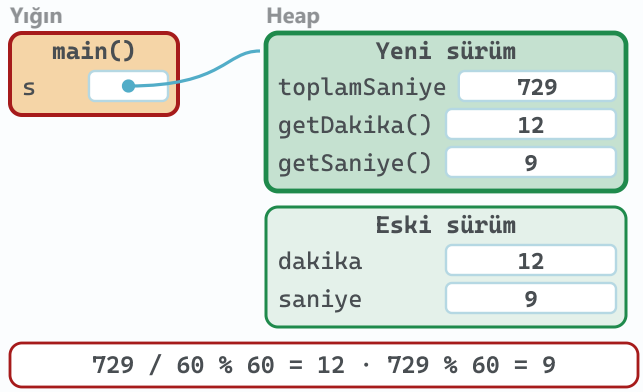
Fark negatif olabilir: 9 − 45 = −36, toplam **729**. Dakika yine 12.

Adım 6 / 7

ÇIKTI

Aynı Arayüz, Yeni Gerçekleme: Kod İzleme

```
 1 private int toplamSaniye = 0;  // tek alan
 2 public int getSaniye() {
 3     return toplamSaniye % 60;
 4 }
 5 public void setSaniye(int saniye) {
 6     if (saniye >= 0 && saniye < 60)
 7         toplamSaniye += saniye - getSaniye();
 8 }
 9 // getDakika(): toplamSaniye / 60 % 60
10 DijitalSaat s = new DijitalSaat(); // main
11 s.setDakika(12);
12 s.setSaniye(45);
13 s.setSaniye(75);
14 s.setSaniye(9);
▶ System.out.println(s.getDakika() + ":"
▶     + s.getSaniye());
```



`729 / 60 % 60 = 12 · 729 % 60 = 9`

Çıktı **12:9**; eski sürüm de aynısını verirdi. İç temsil değişti, `main` tek harf değişmedi.

Adım 7 / 7

ÇIKTI

```
12:9
```

Tavsiyeler

1. Sınıf içindeki tüm değişkenler (alanlar) `private` tanımlanmalı.
2. Değişkenlere erişim için **gerektiği kadar** `public` erişim metodu sağlanmalı (her alana otomatik setter yazmak şart değildir).
3. Her metodun nasıl kullanılacağını belirten yorum satırları eklenmeli: ne yapar, ön ve son koşulları neler.
4. Yalnızca sınıf içinde kullanılan **yardımcı metotlar** `private` tanımlanmalı.

```
public class Ogrenci {
    private int not;
    private boolean gecerliMi(int deger) {
        return deger >= 0 && deger <= 100;
    }
    public void setNot(int not) {
        if (gecerliMi(not)) this.not = not;
    }
}
```

Başka sınıftan `o.gecerliMi(50)` çağrılırsa:

```
error: gecerliMi(int) has private access in
Ogrenci
```

SINIF TASARIMI

Sor Değil, Söyle

Her alana bir getter ve bir setter yazmak sarmalama **değildir**: kurallar yine sınıfın dışında uygulanıyorsa veri yalnızca bir adım uzağa taşınmıştır.

Sormak: kural kullanan kodda

```
// her para çeken yerde aynı kontrol
if (h.getBakiye() >= 300) {
    h.setBakiye(h.getBakiye() - 300);
}
// ...ve not ekleyen her yerde
o.setNot(Math.min(100, o.getNot() + 5));
```

- Kural her çağrı yerinde tekrar eder; biri unutursa nesne bozulur.
- `setBakiye` yazıldığı anda herkes bakiyeyi istediği değere çekebilir.

Söylemek: kural nesnenin içinde

```
h.paraCek(300);   // kural BankaHesabi'nda
o.notEkle(5);     // 100 sınırı Ogrenci'de
```

- Nesneye verisini sorup onun yerine karar vermek yerine, ne istendiği **söylenir**; kararı veriyi tutan nesne verir.
- Setter yalnızca alanın gerçekten serbestçe değişebildiği durumlarda yazılır. `BankaHesabi` 'nda bu yüzden `setBakiye` yoktur.

Sınama sorusu: “Bu kararı vermek için gereken veri kimde?” Cevap nesnenin kendisiyse karar da onun metodunda olmalıdır.

SINIF TASARIMI

Sarmalamanın Faydaları

Soyutlama

Nesne ile kullanıcısı arasındaki ilişkiyi sadeleştirir: kullanıcı yalnızca metotları bilir.

Koruma

Nesneyi kullanıcının istenmeyen, hatalı erişimlerine karşı korur.

Değiştirilebilirlik

Arayüzü değiştirmeden sınıfın iç yapısı değiştirilebilir; kullanan kod etkilenmez.

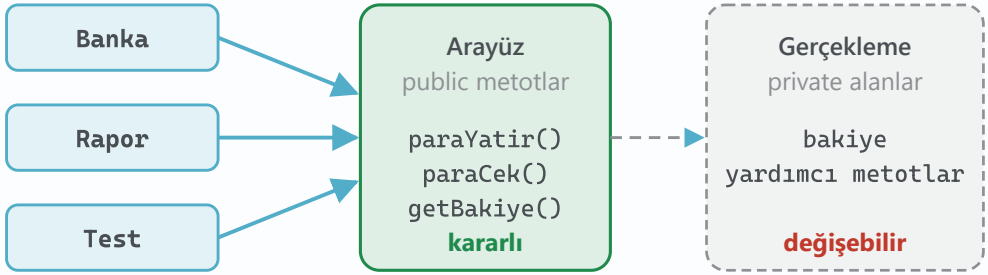
Geçerli durum (değişmez)

Nesnenin durumu her zaman kurallara uyar (sınıf değişmezi, class invariant):

- yalnızca pozitif koordinatlı noktalar,
- saniye ve dakika 0–59 arasında,
- saat negatif olamaz.

Bu bölümün örneklerinde: `setSaniye(75)` reddedilir, `paraCek` bakiyeyi asla eksiye düşürmez, `BankaHesabi` 'nın iç yapısı değişse bile `paraYatir` çağıran kod aynı kalır.

Gözlemler



- Arayüzler, gerçeklemeye göre çok **daha nadir** değişir: bu, kodun **kararlı** olmasını sağlar.
- Gerçekleme değiştiğinde, ona **doğrudan** bağımlı olan kodların da değişmesi gerekir.
- Gerçeklemeyi gizlemek, diğer sınıfların bağımlılığını azaltır: değişiklik sınıfın içinde kalır (gevşek bağlılık, loose coupling).

Sızan Referans

```
public class Karne {
    private final int[] notlar;
    public Karne(int[] notlar) {
        this.notlar = notlar; // aynı dizi
    }
    public int[] getNotlar() {
        return notlar;        // iç dizi
    }
    public double ortalama() {
        int t = 0;
        for (int n : notlar) t += n;
        return (double) t / notlar.length;
    }
}
```

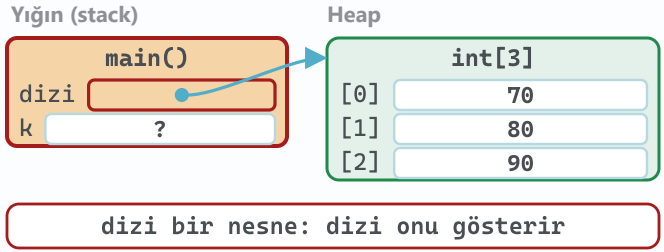
```
int[] dizi = {70, 80, 90};
Karne k = new Karne(dizi);
dizi[0] = 0;                // dışarıdaki dizi
k.getNotlar()[1] = -100;    // getter'dan gelen
System.out.println(k.ortalama());
```

```
-3.3333333333333335
```

- Alan `private final`, setter yok; yine de notlar dışarıdan değişti, hatta −100 gibi geçersiz bir not girdi.
- Sebep: dizi bir **nesnedir**. Yapıcı ve getter yalnızca **referansı** paylaştı; nesneye üç ayrı yerden ulaşılıyor.
- `final` yalnızca referansın başka diziyi göstermesini engeller, dizinin **içeriğini** korumaz.

Sızan Referans: Kod İzleme

```
1  public Karne(int[] notlar) {
2      this.notlar = notlar;
3  }
4  public int[] getNotlar() {
5      return notlar;
6  }
7  // main içinde:
▶ int[] dizi = {70, 80, 90};
9  Karne k = new Karne(dizi);
10 dizi[0] = 0;
11 k.getNotlar()[1] = -100;
12 System.out.println(k.ortalama());
```



Dizi heap'te bir nesnedir; `dizi` değişkeni yalnızca onun referansını tutar.

Adım 1 / 5

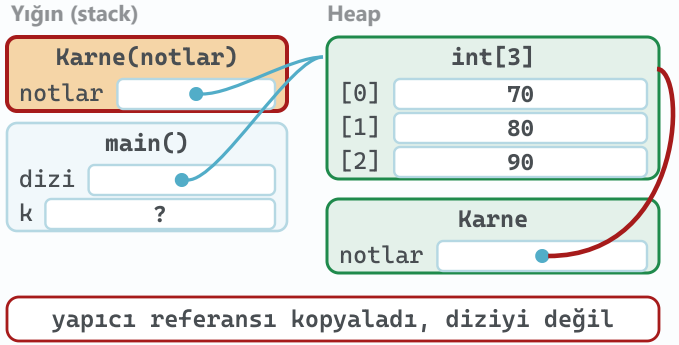
ÇIKTI

DEĞİŞKENLER

`dizi = →[70, 80, 90]`

Sızan Referans: Kod İzleme

```
1  public Karne(int[] notlar) {
2      this.notlar = notlar;
3  }
4  public int[] getNotlar() {
5      return notlar;
6  }
7  // main içinde:
8  int[] dizi = {70, 80, 90};
9  Karne k = new Karne(dizi);
10 dizi[0] = 0;
11 k.getNotlar()[1] = -100;
12 System.out.println(k.ortalama());
```



Yapıcı parametredeki **referansı** alana yazar. Artık `Karne` 'nin `private` alanı ile `main` 'in `dizi` değişkeni aynı diziyi gösterir.

Adım 2 / 5

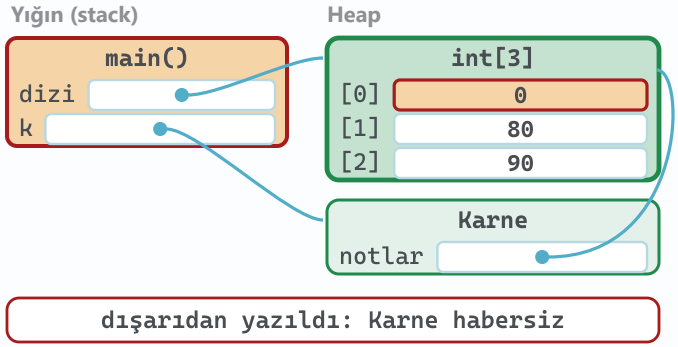
ÇIKTI

DEĞİŞKENLER

`dizi = →[70, 80, 90]` `k = →Karne`

Sızan Referans: Kod İzleme

```
1  public Karne(int[] notlar) {
2      this.notlar = notlar;
3  }
4  public int[] getNotlar() {
5      return notlar;
6  }
7  // main içinde:
8  int[] dizi = {70, 80, 90};
9  Karne k = new Karne(dizi);
▶  dizi[0] = 0;
11 k.getNotlar()[1] = -100;
12 System.out.println(k.ortalama());
```



`dizi[0] = 0` Karne'nin hiçbir metodundan geçmez, ama Karne'nin gördüğü not değişti.

Adım 3 / 5

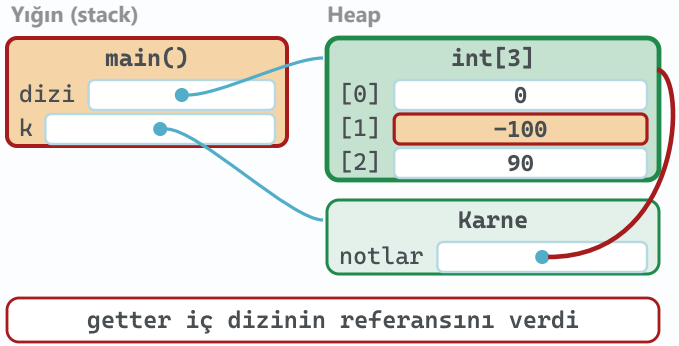
ÇIKTI

DEĞİŞKENLER

`dizi = →[0, 80, 90]` `k = →Karne`

Sızan Referans: Kod İzleme

```
1  public Karne(int[] notlar) {
2      this.notlar = notlar;
3  }
4  public int[] getNotlar() {
▶      return notlar;
6  }
7  // main içinde:
8  int[] dizi = {70, 80, 90};
9  Karne k = new Karne(dizi);
10 dizi[0] = 0;
▶ k.getNotlar()[1] = -100;
12 System.out.println(k.ortalama());
```



`getNotlar()` iç dizinin kendisini döndürür; dönen referansla −100 yazılır. Geçersiz not, hiçbir denetimden geçmeden nesneye girdi.

Adım 4 / 5

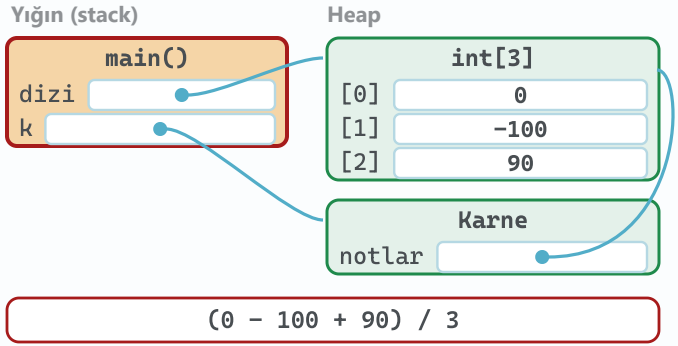
ÇIKTI

DEĞİŞKENLER

`dizi = →[0, -100, 90]` `k = →Karne`

Sızan Referans: Kod İzleme

```
1  public Karne(int[] notlar) {
2      this.notlar = notlar;
3  }
4  public int[] getNotlar() {
5      return notlar;
6  }
7  // main içinde:
8  int[] dizi = {70, 80, 90};
9  Karne k = new Karne(dizi);
10 dizi[0] = 0;
11 k.getNotlar()[1] = -100;
▶  System.out.println(k.ortalama());
```



Ortalama (0 − 100 + 90) / 3 ≈ −3,33. `private` yazmak yetmedi: iç nesneye giden referans dışarı **sızdı**.

Adım 5 / 5

ÇIKTI

```
-3.3333333333333335
```

DEĞİŞKENLER

`dizi = →[0, -100, 90]` `k = →Karne`

Savunmacı Kopyalama

```
public class Karne {
    private final int[] notlar;
    public Karne(int[] notlar) {
        this.notlar = notlar.clone(); // girişte
    }
    public int[] getNotlar() {
        return notlar.clone();        // çıkışta
    }
    // ortalama() aynı
}
```

Aynı `main` ile

```
80.0
```

- **Savunmacı kopya** (defensive copy): değiştirilebilir bir nesne sınıfa **girerken** ve sınıftan **çıkarken** kopyalanır. İç diziye ulaşan tek referans sınıfın kendi alanı kalır.
- Yapıcıdaki kopya, çağıranın elindeki diziyi sonradan değiştirmesine karşı; getter'daki kopya, dönen diziyi değiştirmesine karşı korur.
- Listeler için `new ArrayList<>(liste)` ya da salt okunur görünüm `Collections.unmodifiableList(liste)` kullanılır.
- `String`, `Integer` gibi değiştirilemez nesneler kopyalanmaz: paylaşılmaları zararsızdır.

Kopyalamanın bir maliyeti vardır; büyük veride salt okunur görünüm ya da baştan değiştirilemez bir tip seçmek daha ucuzdur.

Değiştirilemez Sınıf Tasarlamak

```
public final class Para {        // alt sınıf yok
    private final long kurus;     // bir kez atanır
    private final String birim;
    public Para(long kurus, String birim) {
        this.kurus = kurus;
        this.birim = birim;
    }
    public Para ekle(Para diger) {   // setter yerine
        if (!birim.equals(diger.birim))
            throw new IllegalArgumentException("Birim farklı");
        return new Para(kurus + diger.kurus, birim);
    }
    @Override public String toString() {
        return kurus / 100 + ","
            + String.format("%02d", kurus % 100) + " " + birim;
    }
}
```

Kurallar

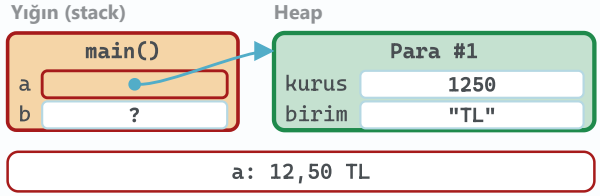
1. Setter yok; durum yalnızca yapıcıda kurulur.
2. Alanlar `private final`.
3. Sınıf `final`: değiştirilebilir bir alt sınıf yazılamasın.
4. "Değiştiren" metotlar yeni bir nesne döndürür.
5. Değiştirilebilir alanlar varsa savunmacı kopya.

Kazanç

- Değişmez yalnızca yapıcıda denetlenir; sonra hiç bozulamaz.
- Nesne kopyalanmadan paylaşılabilir, `HashMap` anahtarı olabilir, iş parçacıkları arasında güvenlidir.
- `String` ve `record` bu kuralların hazır örnekleridir.

Değiştirilemez `Para`: Kod İzleme

```
 1 public Para ekle(Para diger) {
 2     if (!birim.equals(diger.birim))
 3         throw new IllegalArgumentException(
 4             "Birim farklı");
 5     return new Para(kurus + diger.kurus, birim);
 6 }
 7 // main içinde:
▶ Para a = new Para(1250, "TL");
 9 a.ekle(new Para(100, "TL"));  // sonuç atıldı
10 Para b = a.ekle(new Para(375, "TL"));
11 System.out.println(a + " | " + b);
```



İlk nesne kurulur. Alanları `final`: bundan sonra hiçbir metot onları değiştiremez.

Adım 1 / 5

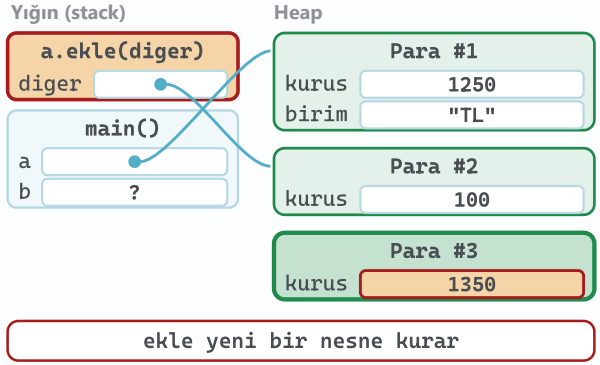
ÇIKTI

DEĞİŞKENLER

`a = 12,50 TL`

Değiştirilemez `Para`: Kod İzleme

```
1  public Para ekle(Para diger) {
2      if (!birim.equals(diger.birim))
3          throw new IllegalArgumentException(
4              "Birim farklı");
▶      return new Para(kurus + diger.kurus, birim);
6  }
7  // main içinde:
8  Para a = new Para(1250, "TL");
▶  a.ekle(new Para(100, "TL"));  // sonuç atıldı
10 Para b = a.ekle(new Para(375, "TL"));
11 System.out.println(a + " | " + b);
```



`ekle`, `a`'yı değiştirmez; toplamı taşıyan **yeni** bir nesne (#3) kurup döndürür.

Adım 2 / 5

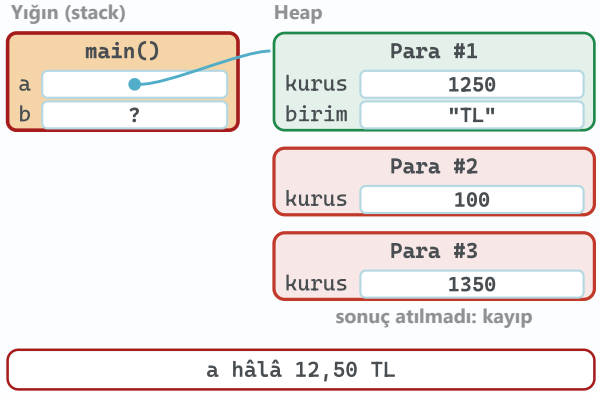
ÇIKTI

DEĞİŞKENLER

`a = 12,50 TL`

Değiştirilemez `Para`: Kod İzleme

```
 1 public Para ekle(Para diger) {
 2     if (!birim.equals(diger.birim))
 3         throw new IllegalArgumentException(
 4             "Birim farklı");
 5     return new Para(kurus + diger.kurus, birim);
 6 }
 7 // main içinde:
 8 Para a = new Para(1250, "TL");
▶ a.ekle(new Para(100, "TL"));  // sonuç atıldı
10 Para b = a.ekle(new Para(375, "TL"));
11 System.out.println(a + " | " + b);
```



Dönen değer bir değişkene atanmadı: #3 kaybolur. Değiştirilemez nesnelerde en sık yapılan hata budur; `a` değişmedi.

Adım 3 / 5

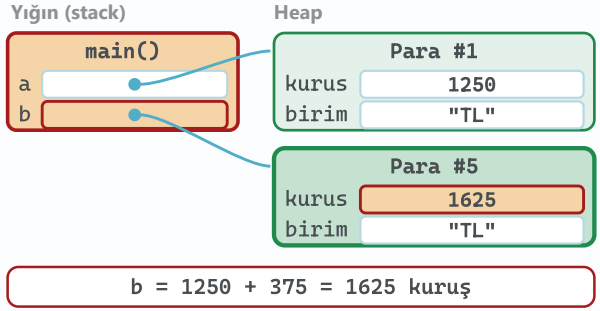
ÇIKTI

DEĞİŞKENLER

`a = 12,50 TL`

Değiştirilemez `Para`: Kod İzleme

```
1  public Para ekle(Para diger) {
2      if (!birim.equals(diger.birim))
3          throw new IllegalArgumentException(
4              "Birim farklı");
▶      return new Para(kurus + diger.kurus, birim);
6  }
7  // main içinde:
8  Para a = new Para(1250, "TL");
9  a.ekle(new Para(100, "TL"));  // sonuç atıldı
▶  Para b = a.ekle(new Para(375, "TL"));
11 System.out.println(a + " | " + b);
```



Bu kez sonuç `b` 'ye atanır: 1250 + 375 = 1625 kuruş. Argüman olarak kurulan #4 de artık kullanılmaz.

Adım 4 / 5

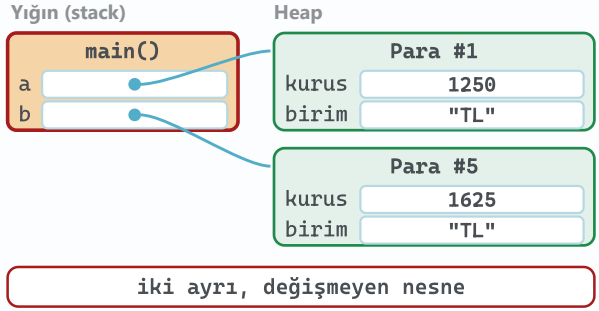
ÇIKTI

DEĞİŞKENLER

`a = 12,50 TL` `b = 16,25 TL`

Değiştirilemez `Para`: Kod İzleme

```
1  public Para ekle(Para diger) {
2      if (!birim.equals(diger.birim))
3          throw new IllegalArgumentException(
4              "Birim farklı");
5      return new Para(kurus + diger.kurus, birim);
6  }
7  // main içinde:
8  Para a = new Para(1250, "TL");
9  a.ekle(new Para(100, "TL"));  // sonuç atıldı
10 Para b = a.ekle(new Para(375, "TL"));
▶  System.out.println(a + " | " + b);
```



`a` ilk değerini korudu. `String` 'in `toUpperCase()` ya da `concat()` metotları da aynı biçimde çalışır.

Adım 5 / 5

ÇIKTI

```
12,50 TL | 16,25 TL
```

DEĞİŞKENLER

`a` = **12,50 TL** `b` = **16,25 TL**

Paketler

- **Paket** (package): bir klasör içinde gruplanmış sınıflar topluluğu. Klasörün adı paketin adıdır.
- Her `public` sınıf ayrı bir dosyada, sınıfla aynı adlı dosyada tanımlanır.
- Dosyanın ilk satırı ait olduğu paketi bildirir: `package paket_adi;`
- Derleyici ve JVM paketleri **sınıf yolu** (classpath) köklerinden başlayarak arar; paket adı bu kökten itibaren klasör yolunu söyler. Varsayılan sınıf yolu bulunulan dizindir (`.`).
- İç içe klasörlerde `/` veya `\` yerine nokta kullanılır: `okul/yonetim` → `okul.yonetim` .
- Paket adları küçük harfle yazılır; Java'nın kendi paketleri: `java.lang` , `java.util` , `java.io` ...

```
proje/
├── src/
│   ├── banka/
│   │   ├── Hesap.java       // package banka;
│   │   └── Yardimci.java    // package banka;
│   └── uygulama/
│       └── Ana.java         // package uygulama;
└── out/                     // derleme çıktısı
    ├── banka/
    │   ├── Hesap.class
    │   └── Yardimci.class
    └── uygulama/
        └── Ana.class
```

Sınıfın tam adı paketini de içerir: `banka.Hesap` , `uygulama.Ana` .

Paket Örneği

```java
package banka;

public class Hesap {
    private double bakiye;

    public void yatir(double miktar) {
        if (miktar > 0) {
            bakiye += miktar;
        }
    }

    public double getBakiye() {
        return bakiye;
    }
}
```

```java
package uygulama;

import banka.Hesap;

public class Ana {
    public static void main(String[] args) {
        Hesap h = new Hesap();
        h.yatir(250.0);
        System.out.println("Bakiye: " + h.getBakiye());
    }
}
```

KOMUT SATIRI (PROJE/ İÇİNDE)

```
> javac -d out src/banka/Hesap.java
src/uygulama/Ana.java
> java -cp out uygulama.Ana
Bakiye: 250.0
```

- `-d out` : .class dosyaları paket klasörleriyle `out/` altına yazılır.
- `-cp out` : sınıf yolu; çalıştırırken sınıfın **tam adı** verilir.

İçe Aktarma (import)

- Başka paketteki bir sınıfı kısa adıyla kullanmak için: `import paket_adi.SinifAdi;` Paketin tümü: `import java.util.*;` (alt paketler hariç).
- `import` yazmadan **tam ad** da kullanılabilir: `java.util.Random`.
- `java.lang` (`String`, `Math`, `System`) ve aynı paketteki sınıflar otomatik görünür.
- Sıra: önce `package`, sonra `import`'lar, sonra sınıf. `import static` static üyeyi sınıf adı olmadan kullandırır.

```
import java.util.Scanner;
import static java.lang.Math.sqrt;

public class ImportOrnek {
  public static void main(String[] args) {
    Scanner giris = new Scanner(System.in);
    int sayi = giris.nextInt();
    java.util.Random rastgele =
        new java.util.Random(42);
    System.out.println(sqrt(sayi) + " "
        + Math.max(sayi, 10) + " "
        + rastgele.nextInt(100));
  }
}
```

ÇIKTI (GİRDİ: 49)

```
7.0 49 30
```

- `Scanner`: import ile kısa ad.
- `java.util.Random`: tam ad, import yok.
- `sqrt`: `import static`; `Math.max`: `java.lang`.

Paket Erişimi

```java
package banka;

class Yardimci {      // public değil
    static double kdv(double tutar) {
        return tutar * 0.20;
    }
}
```

```java
package uygulama;

import banka.Yardimci;

public class Ana2 {
}
```

```
Ana2.java:3: error: Yardimci is not public in
banka;
cannot be accessed from outside package
import banka.Yardimci;
            ^
```

- Belirleyicisi olmayan sınıf ve üyeler **paket erişimlidir**: yalnızca kendi paketinden görülür.
- `Yardimci`, `banka` paketinin **iç işidir**; `Hesap` onu kullanabilir, `uygulama` paketi kullanamaz.
- Paket de bir sarmalama düzeyidir: sınıflar dışarıya kapatılır, yalnızca `public` olanlar paketin arayüzünü oluşturur.

Paket Adlandırma ve Alt Paketler

- Ad çakışmasını önlemek için kurumlar paket adına **ters çevrilmiş alan adıyla** başlar: `giresun.edu.tr` → `tr.edu.giresun.banka`.
- Noktalı adlar klasörlere karşılık gelir: `tr/edu/giresun/banka/`.
- **Alt paket ayrı bir pakettir.** `tr.edu.giresun.banka.rapor`, `banka`'nın paket erişimli üyelerini göremez; adların benzemesi bir erişim hakkı vermez.
- `import tr.edu.giresun.banka.*;` alt paketleri içe aktarmaz: `import java.util.*;` da `java.util.function`'daki sınıfları getirmez.

```java
// tr/edu/giresun/banka/Yardimci.java
package tr.edu.giresun.banka;
class Yardimci {                    // paket erişimli
    static double kdv(double t) { return t * 0.20; }
}
```

```java
// tr/edu/giresun/banka/rapor/Rapor.java
package tr.edu.giresun.banka.rapor;
import tr.edu.giresun.banka.Yardimci;
public class Rapor {
    double hesapla() { return Yardimci.kdv(100); }
}
```

```
error: Yardimci is not public in
tr.edu.giresun.banka; cannot be accessed from
outside package
```

Paket adı ile klasör yolu birebir eşleşmezse derleyici sınıfı bulamaz; bu yüzden IDE'ler klasörleri paket adından oluşturur.

Sınıf Adı Çakışmaları

```java
import java.util.*;
import java.sql.*;

public class Belirsiz {
    public static void main(String[] args) {
        Date tarih = null;
    }
}
```

```
Belirsiz.java:6: error: reference to Date is
ambiguous
  both class java.sql.Date in java.sql and
  class java.util.Date in java.util match
```

```java
import java.util.Date;

public class TarihOrnegi {
  public static void main(String[] args) {
    Date baslangic = new Date(0);  // java.util
    java.sql.Date kayit =          // tam ad
        java.sql.Date.valueOf("2024-09-16");
    System.out.println(baslangic.getTime());
    System.out.println(kayit);
  }
}
```

ÇIKTI

```
0
2024-09-16
```

- İki farklı paketteki sınıflar aynı adı taşıyabilir: `java.util.Date` ve `java.sql.Date`.
- Belirsizliği gidermek için **paket adı** kullanılır: biri `import` edilir, diğeri **tam adıyla** yazılır.

Özet

Kavram	Anlamı	Java'da
Sarmalama	Veri + metotları tek birimde toplamak, iç ayrıntıyı gizlemek	`private` alanlar, `public` metotlar
Bilgi saklama	"Ne" ile "nasıl"ı ayırmak	Kullanıcı yalnızca metot imzasını bilir
Ön / son koşul	Çağrıdan önce varsayılan / sonra garanti edilen durum	Javadoc yorumu + `if` kontrolü
Erişim belirleyici	Kimin erişebileceği	`public`, `protected`, (paket), `private`
Getter / setter	Alanı okuyan / doğrulayarak değiştiren metot	`getSaniye()`, `setSaniye(int)`, `this.saniye = saniye`
Arayüz / gerçekleme	Sınıfın dış yüzü / iç mekanizması	`interface Araba`, `class SporAraba implements Araba`
Paket, import	Klasördeki sınıf grubu; başka paketten kısa adla kullanım	`package banka;`, `import java.util.Scanner;`
Sınıf adı çakışması	Farklı paketlerde aynı adlı sınıf	Tam ad: `java.sql.Date`

ÖZET

İlgili Sayfalar

Erişim Belirleyiciler ve Sarmalama

Simülatör: `public`, `protected`, paket erişimi ve `private` üyelere hangi sınıftan erişilebildiği; setter ile korunan alan.

Çalışma Soruları

Bilgi saklama, sözleşme, erişim düzeyleri, getter ve setter, banka hesabı, gerçeklemeyi değiştirmek ve paket soruları.

ÖZET

Sık Yapılan Hatalar (1)

private alana dışarıdan erişmek

```
error: saniye has private access in DijitalSaat
```

`s.saniye = 75;` yerine `s.setSaniye(75);` yazılır.

Setter'da this unutmak

```
en = en;   // derlenir, alan değişmez
```

Parametre alanı gölgeler; `getEn()` 0 döndürür. Doğrusu `this.en = en;`

Getter'ı void yazmak

```
error: incompatible types: unexpected return value
```

`public void getEn()` değer döndüremez; dönüş tipi `int` olmalı.

Setter'ı doğrulamasız yazmak

```
public void setSaniye(int s) { saniye = s; }
```

Bu, alanı public yapmaktan farksızdır; geçerlilik kuralı setter'da denetlenir.

ÖZET

Sık Yapılan Hatalar (2)

import unutmak

```
error: cannot find symbol
  symbol:   class Scanner
```

Dosyanın başına `import java.util.Scanner;` eklenir.

public sınıf ≠ dosya adı

```
error: class Hesap is public, should be
declared in a file named Hesap.java
```

Dosya adı, public sınıfın adıyla birebir aynı olmalı.

Paket erişimli sınıfı dışarıdan kullanmak

```
error: Yardimci is not public in banka;
cannot be accessed from outside package
```

Sınıf ya `public` yapılır ya da yalnız aynı paketten kullanılır.

Arayüz metodunu yazmamak

```
error: SporAraba is not abstract and does
not override abstract method dur() in Araba
```

`implements` eden sınıf arayüzdeki tüm metotları yazmalıdır.

ÖZET

Sık Yapılan Hatalar (3)

İç nesneyi getter ile vermek

`return notlar;` değiştirilebilir diziyi ya da listeyi dışarı açar. Kopya ya da salt okunur görünüm döndürülür.

final alanı değiştirilemez sanmak

`private final int[] notlar` : referans sabittir, dizinin elemanları değil.

Yeni nesneyi atamayı unutmak

`a.ekle(p);` ya da `s.toUpperCase();` tek başına hiçbir şeyi değiştirmez: `a = a.ekle(p);`

Yapıcıda doğrulamamak

Setter'lar denetlese de `new BankaHesabi(1, -500.0, ...)` geçersiz nesne kurar. Yapıcı da kuralı uygulamalı.

assert ile argüman denetlemek

`-ea` verilmezse denetim hiç çalışmaz. `public` metot argümanları için `IllegalArgumentException`.

Alt paketin içe aktarıldığını sanmak

`import java.util.*;` sonrası `Function` için: `cannot find symbol`. `java.util.function` ayrıca içe aktarılır.

Her alana setter yazmak

Kural kullanan koda kayar. Önce davranış (`paraCek`, `notEkle`) tasarlanır; setter gerekmeyebilir.

Hatayı sessizce yutmak

Geçersiz değeri hiçbir iz bırakmadan yok saymak hatayı gizler; ya sözleşmede belirtilir ya da istisna fırlatılır.

ÖZET

Örnek Problemler (1)

```
DijitalSaat t = new DijitalSaat();
t.setSaat(25);
t.setDakika(30);
t.setSaniye(-1);
t.setSaniye(59);
t.goster();
```

```
DijitalSaat a = new DijitalSaat();
DijitalSaat b = a;
a.setDakika(15);
b.setDakika(45);
b = new DijitalSaat();
b.setDakika(10);
System.out.println(a.getDakika() + " " + b.getDakika());
```

"Erişim Metotları" slaytındaki `DijitalSaat` sınıfı kullanılıyor (saat 0–23, dakika/saniye 0–59).

1. Soldaki kod ne yazdırır?
2. Sağdaki kod ne yazdırır? Kaç tane `DijitalSaat` nesnesi oluşturuldu?

▶ **Cevapları göster**

ÖZET

Örnek Problemler (2)

```
public class Test {
    public static void main(String[] args) {
        DijitalSaat s = new DijitalSaat();
        s.setSaniye(30);                    // A
        int x = s.saniye;                   // B
        System.out.println(s.getSaniye());  // C
        s.getSaniye() = 10;                 // D
        s.setSaniye("40");                  // E
    }
}
```

A–E satırlarından hangileri **derleme hatası** verir? Neden?

▶ **Cevapları göster**

ÖZET

Örnek Problemler (3)

```
BankaHesabi h = new BankaHesabi(111, 1000.0, "Vadesiz");
h.paraCek(5000.0);
h.paraYatir(-50.0);
h.faizEkle(3.0);
h.paraCek(250.0);
BankaHesabi v = new BankaHesabi(222, 100.0, "Vadeli");
h.transfer(v, 1000.0);
h.transfer(v, 750.0);
System.out.println(h.getBakiye() + " " + v.getBakiye());
```

`BankaHesabi` sınıfıyla bu kod ne yazdırır? Son satırda hangi bakiyeler görülür?

▶ **Cevabı göster**

ÖZET

Örnek Problemler (4)

A) Kopyalamayan `Karne` ile

```
int[] d = {50, 60};
Karne k = new Karne(d);
int[] g = k.getNotlar();
g[0] = 100;
d[1] = 100;
System.out.println(k.ortalama());
```

B) `Para` sınıfıyla

```
Para p = new Para(100, "TL");
p.ekle(p);
p = p.ekle(p);
p.ekle(new Para(1, "TL"));
System.out.println(p);
```

1. A ne yazdırır? Savunmacı kopyalı `Karne` ile ne yazdırır?
2. B ne yazdırır?
3. On kez `arttir()` çağıran `Sayac` programı `-ea` olmadan ve `-ea` ile çalıştırılırsa ne olur? Hangisi hatayı gösterir?
4. `Kesir` dışındaki bir sınıfta `new Kesir(1, 2).pay` derlenir mi?

▶ **Cevapları göster**

ÖZET

Alıştırmalar (1)

1. `Ogrenci`: `private` `ad` ve `not`; not yalnızca 0–100 arası kabul edilsin; `gectiMi()` not ≥ 50 ise `true` döndürsün.
2. `Nokta`: yalnızca negatif olmayan koordinatları kabul etsin; `uzaklik()` orijine uzaklığı döndürsün.
3. `Termometre`: santigrat değeri saklasın (−273.15 altını reddetsin); `getFahrenheit()` ayrı bir alan **olmadan** hesaplasın.

▶ **Cevaplar / ipuçları**

ÖZET

Alıştırmalar (2)

4. `Sayac` : yalnızca `arttir()` , `sifirla()` , `getDeger()` metotları olsun. Neden setter yok?
5. `Ogrenci` 'yi `okul` paketine taşıyın, `uygulama.Ana` 'dan kullanın; derleme ve çalıştırma komutlarını yazın.
6. `BankaHesabi` 'na yalnızca getter'ı olan `islemSayisi` ekleyin; başarılı her yatırma/çekmede 1 artsın. Banka örneğinin (izleme slaytı) sonunda iki hesabın değeri ne olur?

▶ **Cevaplar / ipuçları**

ÖZET

Kaynaklar

- Oracle, *The Java Tutorials*: "Controlling Access to Members of a Class", "Using the this Keyword", "Packages", "Interfaces".
- J. Gosling ve diğ., *The Java Language Specification, Java SE 21 Edition* (6.6 Access Control, 7 Packages and Modules, 7.5 Import Declarations, 15.12.2 metot seçimi).
- 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 (Bölüm 4: encapsulation).
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12. baskı, 2022 (Bölüm 4: Objects and Classes).
- J. Bloch, *Effective Java*, 3. baskı, Addison-Wesley, 2018 (Madde 15–16: erişimi en aza indirmek, public alan yerine erişim metodu).
- A. B. Downey ve C. Mayfield, *Think Java*, 2. baskı, O'Reilly, 2019 (Bölüm 11: Designing Classes).

Son

Bölüm 4.2: Sarmalama

Sonraki bölüm: Bölüm 4.3: Kalıtım · Terim sözlüğü