

Bölüm 4: Sarmalama

Java ile Nesne Yönelimli Programlama

Bilgi saklama

Ön ve son koşullar

Erişim belirleyiciler

Getter ve setter

Arayüz ve gerçekleştirme

Paketler ve import

Ad çakışmaları

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 gerçekleştirmeyi ayırmak

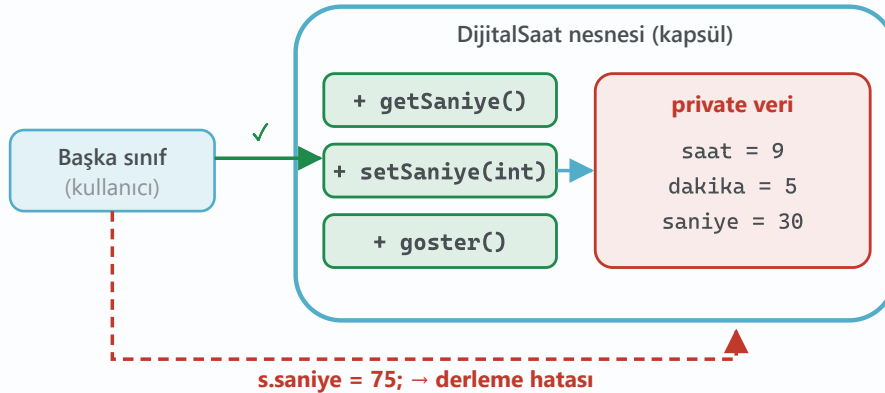
Sınıfın dış yüzünü değiştirmeden iç yapısını değiştirebilmek; `interface` ve `implements` ile tanışmak.

Paket kullanmak

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

Sarmalama (Encapsulation)

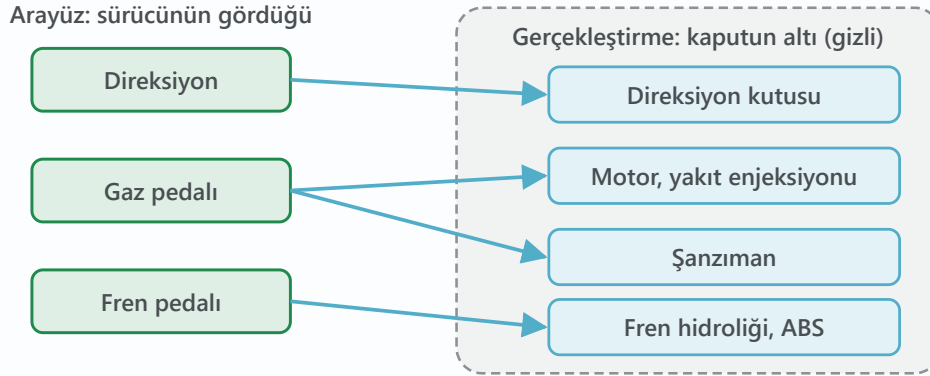
- **Sarmalama** (encapsulation): bir nesnenin **verisini** (alanlar) ve bu veriyi işleyen **metotları** tek birimde, sınıfta toplamak.
- Veri dışarıya **kapatılır** (`private`); dışarıdan yalnızca sınıfın sunduğu **metotlar** çağrılabilir.
- Benzetme: ilaç kapsülü. İçindeki toz görünmez, yalnızca kapsülü 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 metod, iç ayrıntıları bilinmeden de kullanılabilmelidir: metodun **ne işe yaradığını** bilmek yeterli olmalıdır.
- Arabada fren, gaz pedalı ve direksiyonu görür, kullanırsınız; bu parçaların **iç mekaniğini** bilmeyiz.



Aynı fikir koddadır: `System.out.println` metodunu sürekli kullanırsınız 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

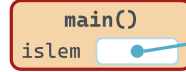
```
Bağlantı açıldı  
Eklendi: Ayşe  
Bağlantı kapatıldı
```

- `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.

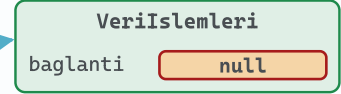
Tek Çağrının Arkası: Adım Adım

```
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) {  
13         ▶ VeriIslemleri islem = new VeriIslemleri();  
14         islem.veriEkle("Ayşe");  
15     }  
16 }
```

Yığın (stack)



Heap



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

Adım 1 / 7

ÇIKTI

DEĞİŞKENLER

`islem = →VeriIslemleri`

ÇAĞRI YIĞINI

`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);  
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 }
```

ÇIKTI

DEĞİŞKENLER

veri = "Ayşe"

Yığın (stack)

veriEkle
veri "Ayşe"

main()
islem

Heap

VeriIslemleri
baglanti null

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

Adım 2 / 7

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();  
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 }
```

ÇIKTI

DEĞİŞKENLER

veri = "Ayşe"

baglanti = →nesne

Yığın (stack)

veriEkle
veri "Ayşe"main()
islem

Heap

VeriIslemleri
baglantiVeritabaniBaglantisi
(Ana bu nesneyi hiç görmez)

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

Adım 3 / 7

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();
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 }
```

ÇIKTI

Bağlantı açıldı

DEĞİŞKENLER

veri = "Ayşe"

Yığın (stack)

veriEkle
veri "Ayşe"

main()
islem

Heap

VeriIslemleri
baglanti

VeritabaniBaglantisi
(Ana bu nesneyi hiç görmez)

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

Adım 4 / 7

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();
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 }
```

ÇIKTI

```
Bağlantı açıldı
Eklendi: Ayşe
```

DEĞİŞKENLER

```
veri = "Ayşe"
```

Yığın (stack)

veriEkle
veri "Ayşe"

main()
islem

Heap

VeriIslemleri
baglanti

VeritabaniBaglantisi
(Ana bu nesneyi hiç görmez)

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

Adım 5 / 7

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
4     // kullanılabilir.
5     public void veriEkle(String veri) {
6         baglanti = new VeritabaniBaglantisi();
7         baglanti.baglan();
8         baglanti.veriEkle(veri);
9         baglanti.kapat();
10    }
11 }
12 public class Ana {
13     public static void main(String[] args) {
14         VeriIslemleri islem = new VeriIslemleri();
15         islem.veriEkle("Ayşe");
16     }
17 }
```

ÇIKTI

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

DEĞİŞKENLER

```
veri = "Ayşe"
```

Yığın (stack)

veriEkle
veri "Ayşe"

main()
islem

Heap

VeriIslemleri
baglanti

VeritabaniBaglantisi
(Ana bu nesneyi hiç görmez)

Bağlantı kapatılır. Üç adımın hepsi tek çağrının arkasında saklı.

Adım 6 / 7

ÇAĞRI YIĞINI

```
baglanti.kapat()
```

```
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);  
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 }
```

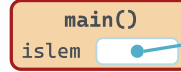
ÇIKTI

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

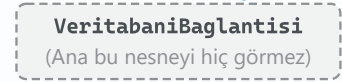
DEĞİŞKENLER

islem = →VeriIslemleri

Yığın (stack)



Heap



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

Adım 7 / 7

ÇAĞRI YIĞINI

main()

ÖN VE SON KOŞULLAR

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.

ÖN VE SON KOŞULLAR

Sözleşme: Tam Örnek

```
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;  
    }  
}
```

```
// 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());
```

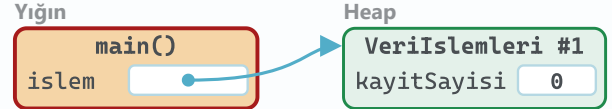
ÇIKTI

```
true  
false  
false  
1
```

- `/** ... */` bir **Javadoc** yorumudur: metodun nasıl kullanılacağını anlatır.
- Ön koşul bozulursa metod hiçbir şey eklemeyiz, `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:  
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());
```



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

Adım 1 / 8

ÇIKTI



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());
```

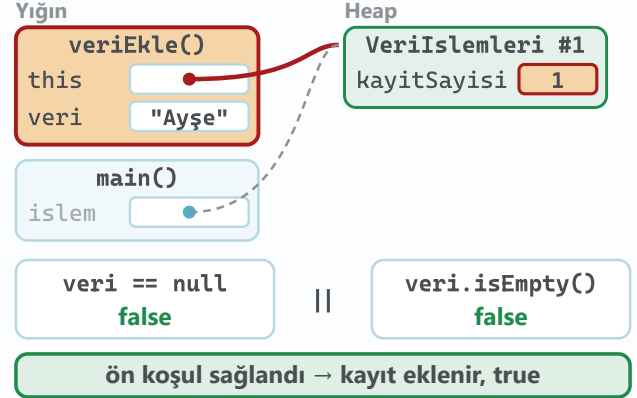


Adım 2 / 8

ÇIKTI

Sözleşme: Kod İzleme

```
1 public boolean veriEkle(String veri) {  
2     if (veri == null || veri.isEmpty()) {  
3         return false; // ön koşul bozuk  
4     }  
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());
```



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

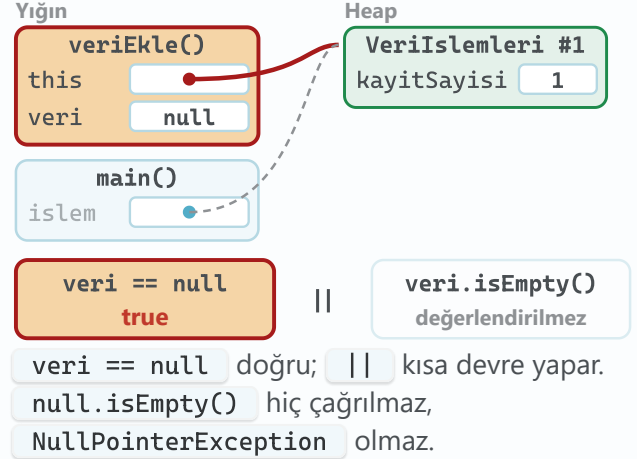
Adım 3 / 8

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());
```



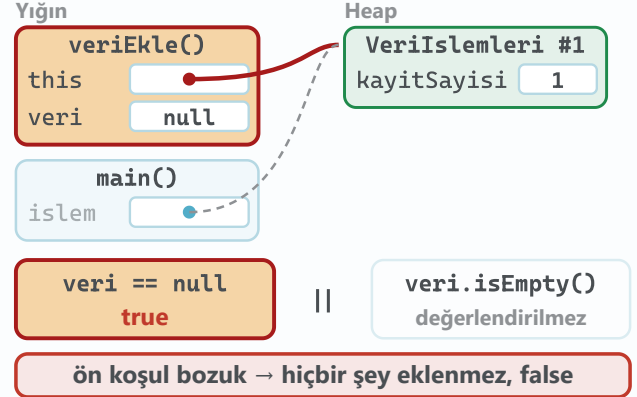
Adım 4 / 8

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: metod nesneye dokunmadan **false** döndürür.

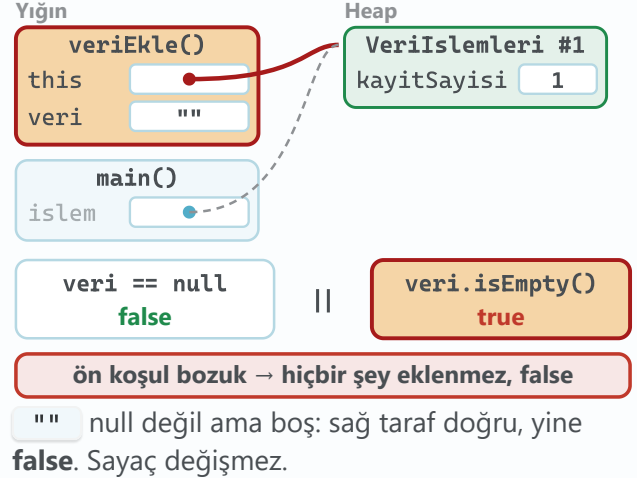
Adım 5 / 8

ÇIKTI

```
true  
false
```

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());
```



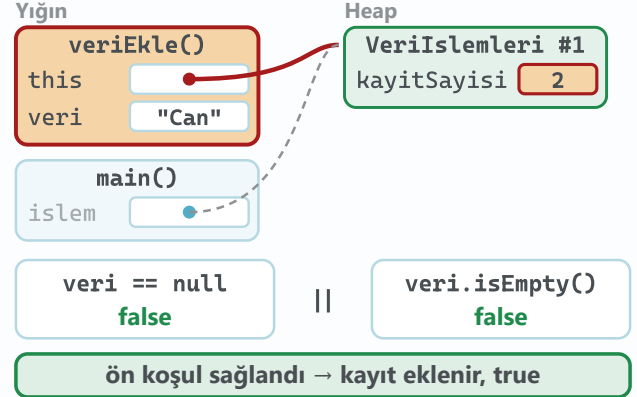
Adım 6 / 8

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     }  
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());
```



Geçerli veri: sayaç 2 olur, **true**.

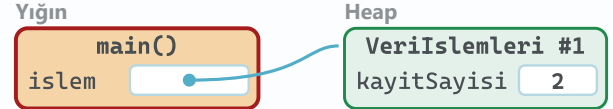
Adım 7 / 8

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
```

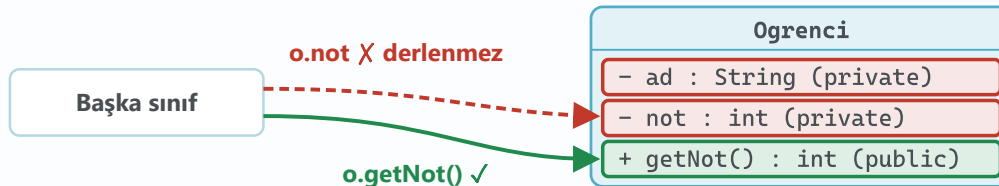
Erişim Belirleyiciler

- **Erişim belirleyici** (access modifier): bir sınıfa, alana veya metoda **kimin** erişebileceğini belirleyen anahtar kelime.
- **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 metod
        return not;
    }
}
```

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



Dört Erişim Düzeyi

Belirleyici	Aynı sınıf	Aynı paket	Alt sınıf (başka paket)	Her yer
<code>public</code>	✓	✓	✓	✓
<code>protected</code>	✓	✓	✓	✗
(yazılmazsa) paket erişimi	✓	✓	✗	✗
<code>private</code>	✓	✗	✗	✗

- Belirleyici yazılmazsa **paket erişimi** (package-private, "default") geçerlidir: yalnızca aynı paketteki sınıflar erişir.
- `protected` alt sınıflar içindir; ayrıntısı **Kalıtım** bölümündedir.
- En kısıtlayıcı olanı seçin: önce `private`, gerçekten gerekirse daha açık bir düzey.
- Üst düzey bir sınıf yalnızca `public` ya da paket erişimli olabilir.

public: her yer

protected: + başka paketteki alt sınıflar

paket erişimi: aynı paket

private: aynı sınıf

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.



DijitalSaat Sınıfı

```
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;  
        }  
    }  
}
```

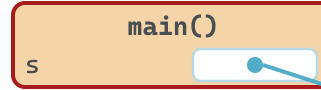
DijitalSaat
- saat: int = 0 - dakika: int = 0 - saniye: int = 0
+ getSaniye(): int + setSaniye(int): void

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

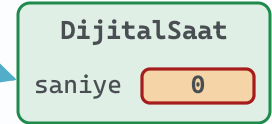
Setter Koruması: Adım Adım

```
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());
```

Yığın (stack)



Heap



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

Adım 1 / 6

ÇIKTI



ÇAĞRI YİĞİNİ

main()

DEĞİŞKENLER

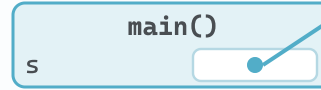
s = →DijitalSaat

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) {  
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());
```

ÇIKTI

Yığın (stack)



Heap

 $45 \geq 0 \ \&\& \ 45 < 60 \rightarrow \text{doğru}$

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

Adım 2 / 6

ÇAĞRI YİĞİNİ

s.setSaniye(45)

main()

DEĞİŞKENLER

this = nesne

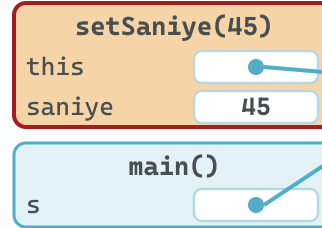
saniye = 45

Setter Koruması: Adım Adım

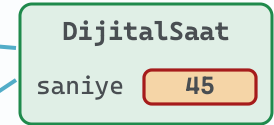
```
1 public class DijitalSaat {  
2     private int saniye = 0;  
3     public int getSaniye() {  
4         return this.saniye;  
5     }  
6     public void setSaniye(int saniye) {  
7         if (saniye >= 0 && saniye < 60) {  
8             this.saniye = saniye;  
9         }  
10    }  
11 }  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

ÇIKTI

Yığın (stack)



Heap



this.saniye = saniye; soldaki **nesnenin alanı**, sağdaki **parametredir**. Alan 45 olur.

Adım 3 / 6

ÇAĞRI YIĞINI

s.setSaniye(45)

main()

DEĞİŞKENLER

this = →nesne

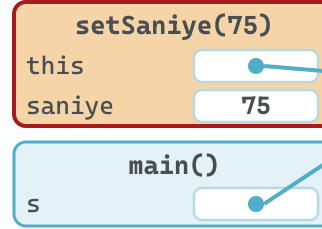
saniye = 45

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) {  
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());
```

ÇIKTI

Yığın (stack)



Heap

**75 < 60 → yanlış**

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

Adım 4 / 6

ÇAĞRI YİĞİNİ

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

DEĞİŞKENLER

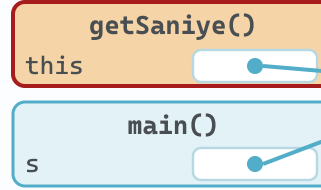
`this = → nesne``saniye = 75`

Setter Koruması: Adım Adım

```
1 public class DijitalSaat {  
2     private int saniye = 0;  
3     public int getSaniye() {  
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);  
16 System.out.println(s.getSaniye());
```

ÇIKTI

Yığın (stack)



Heap



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

Adım 5 / 6

ÇAĞRI YIĞINI

s.getSaniye()

main()

DEĞİŞKENLER

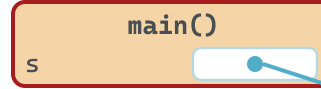
this = nesne

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());
```

Yığın (stack)



Heap



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

Adım 6 / 6

ÇIKTI

45

ÇAĞRI YİĞİNİ

main()

DEĞİŞKENLER

s = →DijitalSaat

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()` :

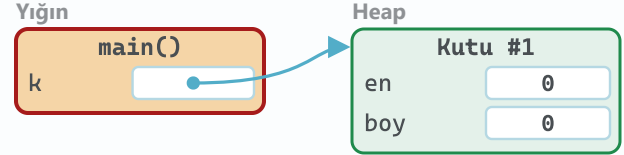
ÇIKTI

0

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

Gölgeleme: Kod İzleme

```
1 public class Kutu {  
2     private int en, boy;  
3     public void setEn(int en) {  
4         en = en;          // hatalı: this yok  
5     }  
6     public void setBoy(int boy) {  
7         this.boy = boy;    // doğru  
8     }  
9     // getEn() ve getBoy() erişimcileri  
10 }  
11 // main içinde:  
▶ Kutu k = new Kutu();  
13 k.setEn(7);  
14 k.setBoy(4);  
15 System.out.println(k.getEn() + " x " + k.getBoy());
```



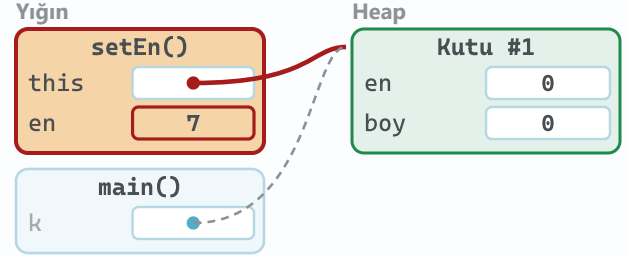
Yeni **Kutu** : iki alan da varsayılan değer 0 ile başlar.

Adım 1 / 7

ÇIKTI

Gölgeleme: Kod İzleme

```
1 public class Kutu {  
2     private int en, boy;  
3     ▶ public void setEn(int en) {  
4         en = en;           // hatalı: this yok  
5     }  
6     public void setBoy(int boy) {  
7         this.boy = boy;    // doğru  
8     }  
9     // getEn() ve getBoy() erişimcileri  
10 }  
11 // main içinde:  
12 Kutu k = new Kutu();  
13 ▶ k.setEn(7);  
14 k.setBoy(4);  
15 System.out.println(k.getEn() + " x " + k.getBoy());
```



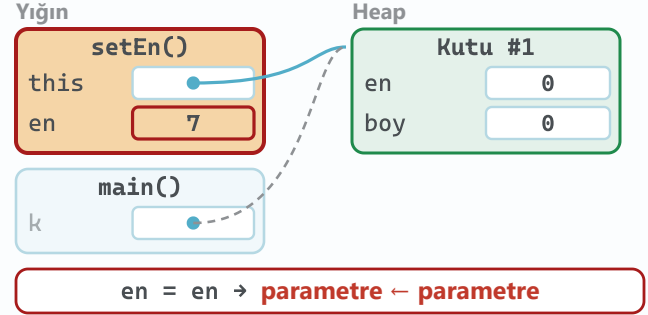
Çerçevede **en** adlı bir **parametre** var (7). Aynı adlı alan, nesnenin içinde: iki ayrı değişken.

Adım 2 / 7

ÇIKTI

Gölgeleme: Kod İzleme

```
1 public class Kutu {  
2     private int en, boy;  
3     public void setEn(int en) {  
4         en = en; // hatalı: this yok  
5     }  
6     public void setBoy(int boy) {  
7         this.boy = boy; // doğru  
8     }  
9     // getEn() ve getBoy() erişimcileri  
10 }  
11 // main içinde:  
12 Kutu k = new Kutu();  
13 k.setEn(7);  
14 k.setBoy(4);  
15 System.out.println(k.getEn() + " x " + k.getBoy());
```



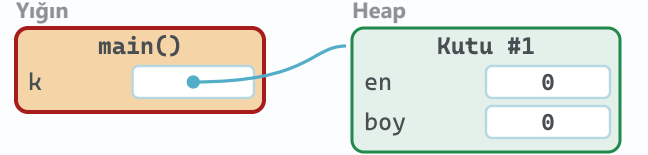
Metot içinde **en** adı en yakın tanımı, yani parametreyi gösterir: parametre alanı **gölgeler**. Atama 5'i yine parametreye yazar.

Adım 3 / 7

ÇIKTI

Gölgeleme: Kod İzleme

```
1 public class Kutu {  
2     private int en, boy;  
3     public void setEn(int en) {  
4         en = en;          // hatalı: this yok  
5     }  
6     public void setBoy(int boy) {  
7         this.boy = boy;    // doğru  
8     }  
9     // getEn() ve getBoy() erişimcileri  
10 }  
11 // main içinde:  
12 Kutu k = new Kutu();  
13 k.setEn(7);  
14 k.setBoy(4);  
15 System.out.println(k.getEn() + " x " + k.getBoy());
```



alan **hâlâ 0**; parametre çerçeveye silindi

Metot biter, çerçeve ve parametre silinir. Nesnedeki

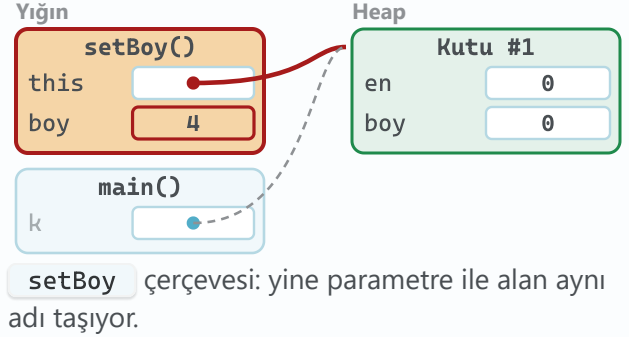
en hiç değişmedi; derleyici hata vermedi.

Adım 4 / 7

ÇIKTI

Gölgeleme: Kod İzleme

```
1 public class Kutu {  
2     private int en, boy;  
3     public void setEn(int en) {  
4         en = en;          // hatalı: this yok  
5     }  
6     ▶ public void setBoy(int boy) {  
7         this.boy = boy;    // doğru  
8     }  
9     // getEn() ve getBoy() erişimcileri  
10 }  
11 // main içinde:  
12 Kutu k = new Kutu();  
13 k.setEn(7);  
14 ▶ k.setBoy(4);  
15 System.out.println(k.getEn() + " x " + k.getBoy());
```

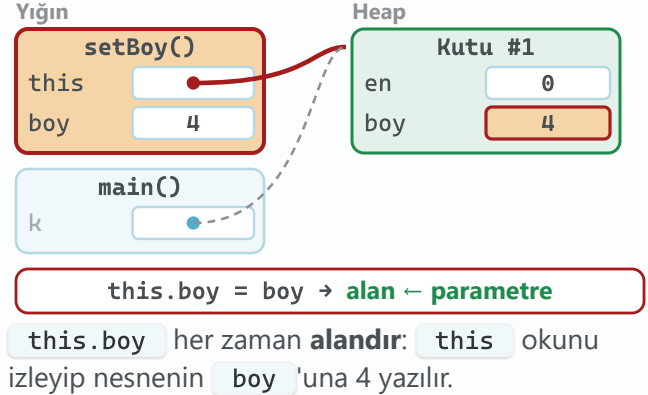


Adım 5 / 7

ÇIKTI

Gölgeleme: Kod İzleme

```
1 public class Kutu {  
2     private int en, boy;  
3     public void setEn(int en) {  
4         en = en;          // hatalı: this yok  
5     }  
6     public void setBoy(int boy) {  
7         this.boy = boy;    // doğru  
8     }  
9     // getEn() ve getBoy() erişimcileri  
10 }  
11 // main içinde:  
12 Kutu k = new Kutu();  
13 k.setEn(7);  
14 k.setBoy(4);  
15 System.out.println(k.getEn() + " x " + k.getBoy());
```

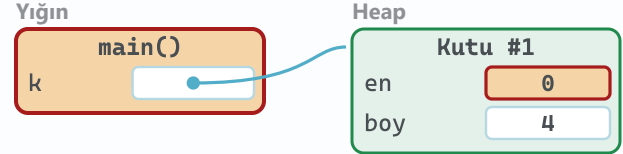


Adım 6 / 7

ÇIKTI

Gölgeleme: Kod İzleme

```
1 public class Kutu {  
2     private int en, boy;  
3     public void setEn(int en) {  
4         en = en;          // hatalı: this yok  
5     }  
6     public void setBoy(int boy) {  
7         this.boy = boy;    // doğru  
8     }  
9     // getEn() ve getBoy() erişimcileri  
10 }  
11 // main içinde:  
12 Kutu k = new Kutu();  
13 k.setEn(7);  
14 k.setBoy(4);  
▶ System.out.println(k.getEn() + " x " + k.getBoy());
```



Çıktı **0 x 4**: `setEn` hiçbir şey yapmadı. Setter'larda `this.` yazma alışkanlığı bu hatayı önler.

Adım 7 / 7

ÇIKTI

0 x 4

Erişim Metotları

```
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);  
    }  
}
```

```
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: `isActive()` .

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.

Deneyin: Setter Koruması

alan **saniye** ▼ değer **75** **setter ile ata** **alan public olsaydı** **sıfırla**

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

saat
9

dakika
5

saniye
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 bozursa 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 **yeral** değişkendir: yalnızca metod 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.");  
    }  
}
```

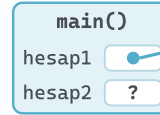
- `transfer` , aynı sınıftan **başka bir nesnenin** metodunu çağırır: `hedefHesap` üzerinde `paraYatir` .
- Hedef hesabın `bakiye` alanına doğrudan yazmak da mümkündür (aynı sınıf, `private` sınıf düzeyindedir), ama metot çağırarak kuralların tek yerde kalmasını sağlar.



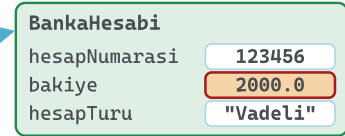
İki Hesap, Bir Transfer: Adım Adım

```
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());
```

Yığın (stack)



Heap



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

Adım 1 / 9

ÇIKTI

ÇAĞRI YIĞINI

main()

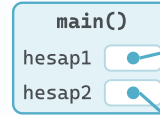
DEĞİŞKENLER

hesap1 = →BankaHesabi

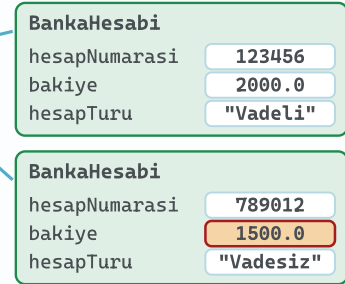
İki Hesap, Bir Transfer: Adım Adım

```
1 // main içinde:
2 BankaHesabi hesap1 =
3   new BankaHesabi(123456, 2000.0, "Vadeli");
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());
```

Yığın (stack)



Heap



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

Adım 2 / 9

ÇIKTI

ÇAĞRI YIĞINI

main()

DEĞİŞKENLER

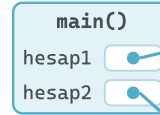
hesap1 = →BankaHesabi

hesap2 = →BankaHesabi

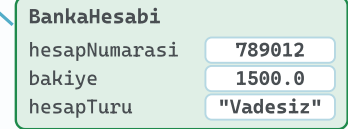
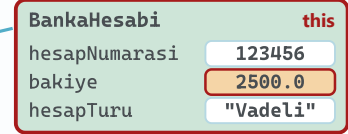
İki Hesap, Bir Transfer: Adım Adım

```
1 // main içinde:
2 BankaHesabi hesap1 =
3   new BankaHesabi(123456, 2000.0, "Vadeli");
4 BankaHesabi hesap2 =
5   new BankaHesabi(789012, 1500.0, "Vadesiz");
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());
```

Yığın (stack)



Heap



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()

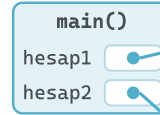
DEĞİŞKENLER

miktar = 500.0

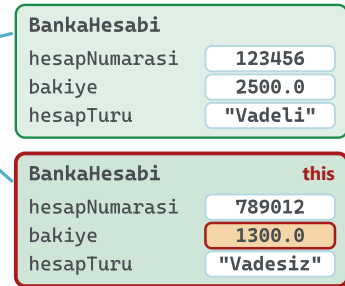
İki Hesap, Bir Transfer: Adım Adım

```
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);
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());
```

Yığın (stack)



Heap



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()`

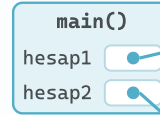
DEĞİŞKENLER

`miktar = 200.0`

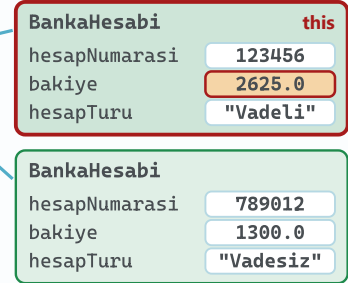
İki Hesap, Bir Transfer: Adım Adım

```
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);
8 ► hesap1.faizEkle(5.0);
9 hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```

Yığın (stack)



Heap



Hesap vadeli: $\text{faiz} = 2500.0 \times 5 / 100 = 125.0$, 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

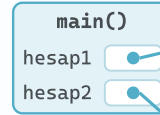
oran = 5.0

faiz = 125.0

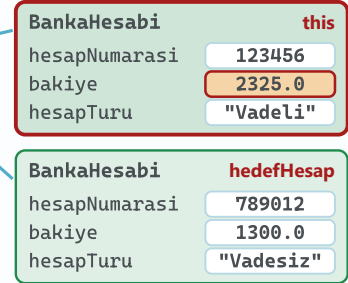
İki Hesap, Bir Transfer: Adım Adım

```
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);
9 ► hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```

Yığın (stack)



Heap



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

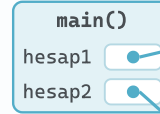
hedefHesap = →hesap2

ÖRNEK: BANKA HESABI · KODU ADIM ADIM ÇALIŞTIR

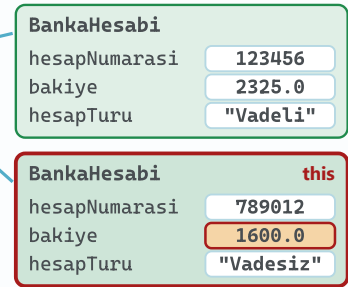
İki Hesap, Bir Transfer: Adım Adım

```
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);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```

Yığın (stack)



Heap



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()

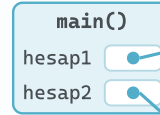
DEĞİŞKENLER

miktar = 300.0

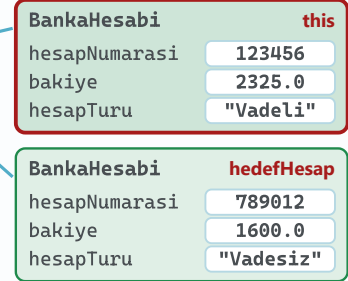
İki Hesap, Bir Transfer: Adım Adım

```
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);
9 ► hesap1.transfer(hesap2, 300.0);
10 System.out.println(hesap1.getBakiye());
11 System.out.println(hesap2.getBakiye());
```

Yığın (stack)



Heap



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()

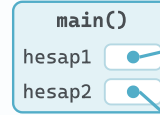
DEĞİŞKENLER

miktar = 300.0

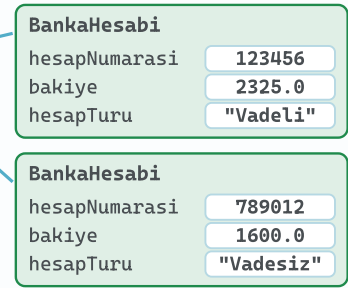
İki Hesap, Bir Transfer: Adım Adım

```
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());
```

Yığın (stack)



Heap



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()

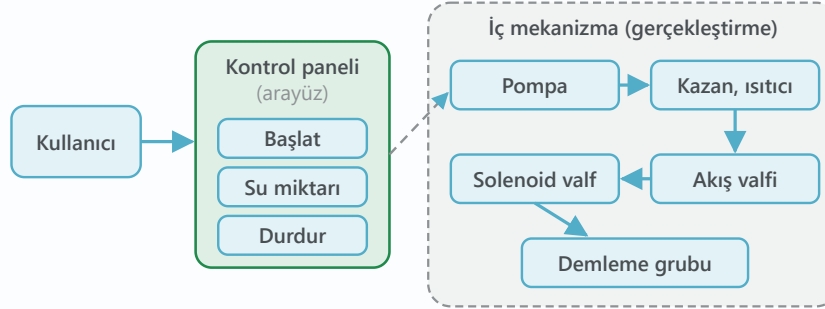
DEĞİŞKENLER

hesap1 = →BankaHesabi

hesap2 = →BankaHesabi

Sınıf Tasarımı

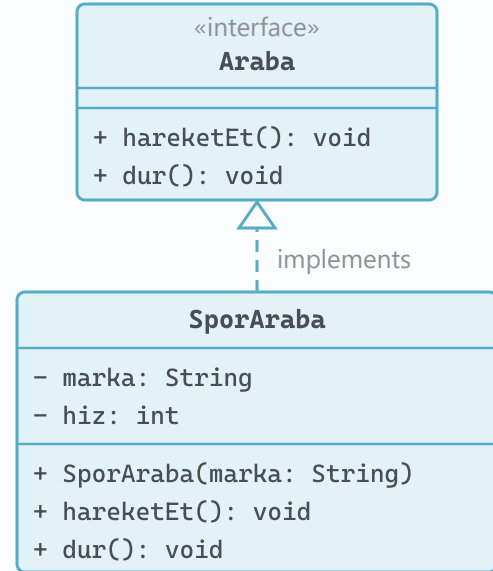
- Sınıf tanımı iki kısma ayrılır: **arayüz** (interface) ve **gerçekleştirme** (implementation).
- Arayüz, sınıfın **ne yaptığı** ve **nasıl kullanılacağı**ı belirtir: public metotların adları, parametreleri, dönüş tipleri ve yorumları.
- Gerçekleştirme, sınıfın iç mekanik ayrıntılarını ve özel (private) alanlarını içerir. Bunlar yalnızca sınıfın içinde önemlidir.
- **Gerçekleştirme değişse bile arayüz değişmeyebilir**; öz alanlar sınıfın dışına gizlidir.



Arayüz

```
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.
- Arayüzler ayrıntılı olarak **Soyut Sınıflar** bölümünde ele alınır.



Sınıf Gerçekleştirilmesi

```
public class SporAraba implements Araba {  
    // Sınıf gerçekleştirimi: 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.");  
    }  
}
```

// 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.

Gerçekleştirmeyi 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 Gerçekleştirme: 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
▶ DigitalSaat s = new DigitalSaat(); // 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());
```

Yığın



Heap



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

Adım 1 / 7

ÇIKTI



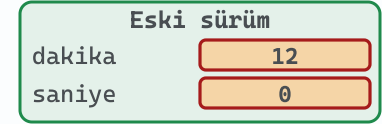
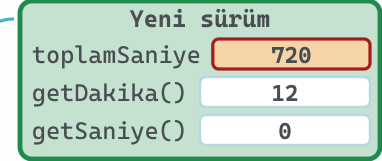
Aynı Arayüz, Yeni Gerçekleştirme: Kod İzleme

```
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());
```

Yığın



Heap



$$12 \cdot 60 = 720$$

setDakika(12) : fark 12 dakika, $12 \cdot 60 = 720$ saniye eklenir. Eski sürüm doğrudan dakika = 12 yazar.

Adım 2 / 7

ÇIKTI



Aynı Arayüz, Yeni Gerçekleştirme: 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());
```

Yığın

main()

s

Heap

Yeni sürüm

toplamSaniye 720

getDakika() 12

getSaniye() 0

Eski sürüm

dakika 12

saniye 0

45 geçerli (0-59)

setSaniye(45) : değer aralıkta, koşul doğru.

Adım 3 / 7

ÇIKTI

Aynı Arayüz, Yeni Gerçekleştirme: 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());
```

Yığın

main()

s

Heap

Yeni sürüm

toplamSaniye 765

getDakika() 12

getSaniye() 45

Eski sürüm

dakika 12

saniye 45

$$720 + (45 - 720 \% 60) = 720 + 45 = 765$$

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

Adım 4 / 7

ÇIKTI

Aynı Arayüz, Yeni Gerçekleştirme: 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());
```

Yığın

main()

s

Heap

Yeni sürüm

toplamSaniye 765

getDakika() 12

getSaniye() 45

Eski sürüm

dakika 12

saniye 45

75 < 60 **false** → değişiklik yok

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

Adım 5 / 7

ÇIKTI

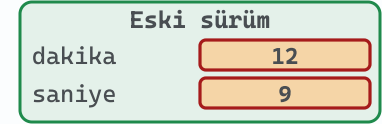
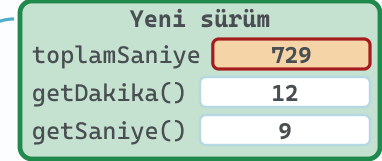
Aynı Arayüz, Yeni Gerçekleştirme: Kod İzleme

```
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());
```

Yığın



Heap



$$765 + (9 - 45) = 729$$

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

Adım 6 / 7

ÇIKTI



Aynı Arayüz, Yeni Gerçekleştirme: 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());
```

Yığın

main()

s

Heap

Yeni sürüm

toplamSaniye 729

getDakika() 12

getSaniye() 9

Eski sürüm

dakika 12

saniye 9

$$729 / 60 \% 60 = 12 \cdot 729 \% 60 = 9$$

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
```

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.

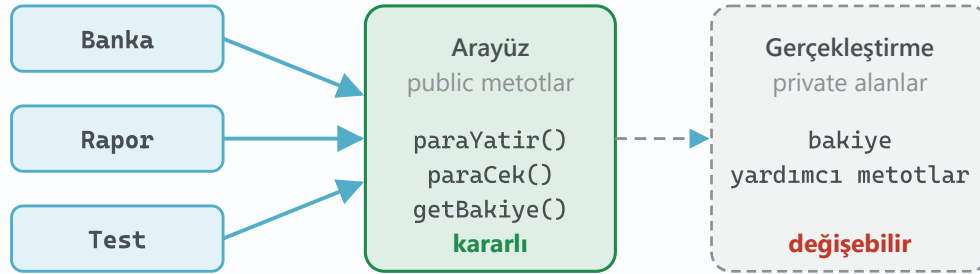
Geçerli durum (değişmez)

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` çağırın kod aynı kalır.

Gözlemler



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

PAKETLER

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` .

PAKETLER

Paket Örneği

```
package banka;

public class Hesap {
    private double bakiye;

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

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

```
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 kullanırır.

```
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`.

PAKETLER

Paket Erişimi

```
package banka;

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

```
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 iştir**; `Hesap` onu kullanabilir, `uygulama` paketi kullanamaz.
- Paket de bir sarmalama düzeyidir: sınıflar dışarıya kapatılır, yalnızca `public` olanlar paketin arayüzünü oluşturur.

Metot Adı Çakışmaları

```
public class Yazici {  
    static void yazdir(int x)          { System.out.println("int: " + x); }  
    static void yazdir(double x)       { System.out.println("double: " + x); }  
    static void yazdir(String s)       { System.out.println("String: " + s); }  
    static void yazdir(int a, int b)   { System.out.println("iki int: " + a + ", " + b); }  
  
    public static void main(String[] args) {  
        yazdir(7);           // int: 7  
        yazdir(7.5);         // double: 7.5  
        yazdir("yedi");      // String: yedi  
        yazdir(3, 4);        // iki int: 3, 4  
        yazdir('A');         // int: 65      (char → int)  
        yazdir(5L);          // double: 5.0   (long → double)  
        yazdir(2.5f);        // double: 2.5   (float → double)  
    }  
}
```

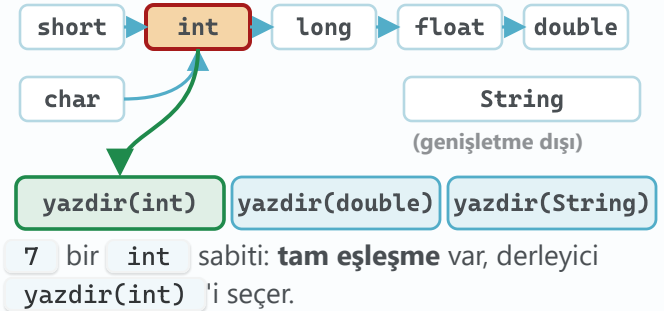
- Aynı adlı birden fazla metot olabilir (**aşırı yükleme**, overloading); parametre **sayısı ve tipleri** ayırt eder. (Yorumlar gerçek çıktıdır.)
- Tam eşleşme yoksa **genişletme** dönüşümü denenir: `char` → `int`, `long` → `double`.
- Hiçbiri uymazsa derleme hatası: `yazdir(true)` → `no suitable method found for yazdir(boolean)`
- Dönüş tipi tek başına ayırt etmez; parametre listesi farklı olmalıdır.

Aşırı Yükleme Seçimi: Kod İzleme

```
▶ static void yazdir(int x) {  
▶   System.out.println("int: " + x); }  
3 static void yazdir(double x) {  
4   System.out.println("double: " + x); }  
5 static void yazdir(String s) {  
6   System.out.println("String: " + s); }  
7 // main içinde:  
▶ yazdir(7);  
9 yazdir('B');  
10 yazdir(9L);  
11 yazdir(7.5f);  
12 yazdir("7");  
13 short k = 3; yazdir(k);
```

argüman: 7 (int)

genişletme:



Adım 1 / 6

ÇIKTI

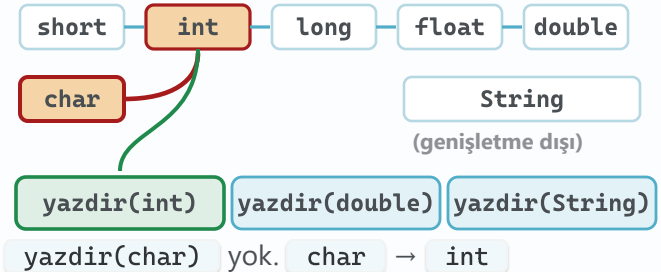
```
int: 7
```

Aşırı Yükleme Seçimi: Kod İzleme

```
▶ static void yazdir(int x) {  
▶     System.out.println("int: " + x); }  
3 static void yazdir(double x) {  
4     System.out.println("double: " + x); }  
5 static void yazdir(String s) {  
6     System.out.println("String: " + s); }  
7 // main içinde:  
8 yazdir(7);  
▶ yazdir('B');  
10 yazdir(9L);  
11 yazdir(7.5f);  
12 yazdir("7");  
13 short k = 3; yazdir(k);
```

argüman: 'B' (char)

genişletme:



Adım 2 / 6

ÇIKTI

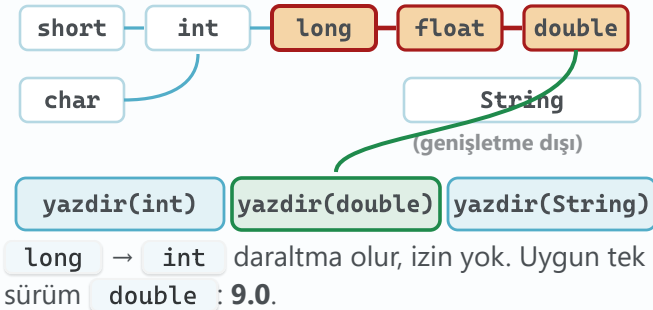
```
int: 7  
int: 66
```

Aşırı Yükleme Seçimi: Kod İzleme

```
1 static void yazdir(int x) {  
2     System.out.println("int: " + x); }  
▶ static void yazdir(double x) {  
▶     System.out.println("double: " + x); }  
5 static void yazdir(String s) {  
6     System.out.println("String: " + s); }  
7 // main içinde:  
8 yazdir(7);  
9 yazdir('B');  
▶ yazdir(9L);  
11 yazdir(7.5f);  
12 yazdir("7");  
13 short k = 3; yazdir(k);
```

argüman: 9L (long)

genişletme:



Adım 3 / 6

ÇIKTI

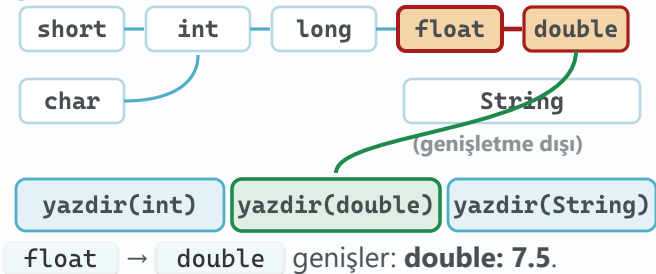
```
int: 7  
int: 66  
double: 9.0
```

Aşırı Yükleme Seçimi: Kod İzleme

```
1 static void yazdir(int x) {  
2     System.out.println("int: " + x); }  
▶ static void yazdir(double x) {  
▶     System.out.println("double: " + x); }  
5 static void yazdir(String s) {  
6     System.out.println("String: " + s); }  
7 // main içinde:  
8 yazdir(7);  
9 yazdir('B');  
10 yazdir(9L);  
▶ yazdir(7.5f);  
12 yazdir("7");  
13 short k = 3; yazdir(k);
```

argüman: 7.5f (float)

genişletme:



Adım 4 / 6

ÇIKTI

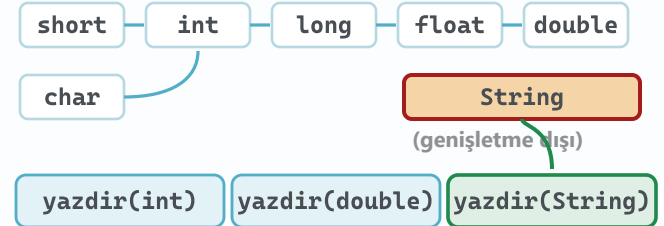
```
int: 7  
int: 66  
double: 9.0  
double: 7.5
```

Aşırı Yükleme Seçimi: Kod İzleme

```
1 static void yazdir(int x) {  
2     System.out.println("int: " + x); }  
3 static void yazdir(double x) {  
4     System.out.println("double: " + x); }  
▶ static void yazdir(String s) {  
▶     System.out.println("String: " + s); }  
7 // main içinde:  
8 yazdir(7);  
9 yazdir('B');  
10 yazdir(9L);  
11 yazdir(7.5f);  
▶ yazdir("7");  
13 short k = 3; yazdir(k);
```

argüman: "7" (String)

genişletme:



Tırnaklı "7" bir String 'tir; sayısal sürümlere hiç uymaz.

Adım 5 / 6

ÇIKTI

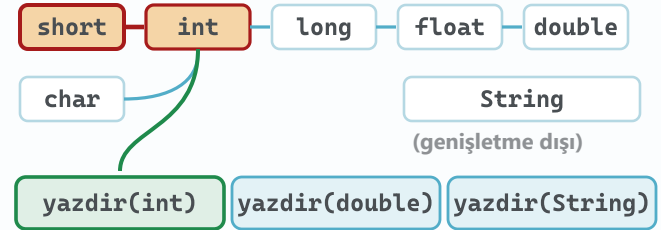
```
int: 7  
int: 66  
double: 9.0  
double: 7.5  
String: 7
```

Aşırı Yükleme Seçimi: Kod İzleme

```
▶ static void yazdir(int x) {  
▶   System.out.println("int: " + x); }  
3 static void yazdir(double x) {  
4   System.out.println("double: " + x); }  
5 static void yazdir(String s) {  
6   System.out.println("String: " + s); }  
7 // main içinde:  
8 yazdir(7);  
9 yazdir('B');  
10 yazdir(9L);  
11 yazdir(7.5f);  
12 yazdir("7");  
▶ short k = 3; yazdir(k);
```

argüman: k (short)

genişletme:



short hem int 'e hem double 'a
genişleyebilir; derleyici **en yakın** (en özel) olanı seçer:
int .

Adım 6 / 6

ÇIKTI

```
int: 7  
int: 66  
double: 9.0  
double: 7.5  
String: 7  
int: 3
```

Sınıf Adı Çakışmaları

```
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

```
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: birini `import` edin, diğerini **tam adıyla** yazın.

ÖZET

Özet

Kavram	Anlamı	Java'da
Sarmalama	Veri + metotları tek birimde toplamak, iç ayrıntıyı gizlemek	<code>private</code> alanlar, <code>public</code> 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 + <code>if</code> kontrolü
Erişim belirleyici	Kimin erişebileceği	<code>public</code> , <code>protected</code> , (paket), <code>private</code>
Getter / setter	Alanı okuyan / doğrulayarak değiştiren metot	<code>getSaniye()</code> , <code>setSaniye(int)</code> , <code>this.saniye = saniye</code>
Arayüz / gerçekleştirme	Sınıfın dış yüzü / iç mekanizması	<code>interface Araba</code> , <code>class SporAraba</code> <code>implements Araba</code>
Paket, import	Klasördeki sınıf grubu; başka paketten kısa adla kullanım	<code>package banka;</code> , <code>import</code> <code>java.util.Scanner;</code>
Ad çakışması	Aynı adlı metot / sınıf	Parametre listesi / tam ad <code>java.sql.Date</code>

ÖZET

Sık Yapılan Hatalar (1)

private alana dışarıdan erişmek

error: saniye has private access in DijitalSaat

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

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 kuralını setter'a koyun.

ÖZET

Sık Yapılan Hatalar (2)

import unutmak

error: cannot find symbol
symbol: class Scanner

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

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

Ya sınıfı `public` yapın ya da aynı pakette kullanın.

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

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

Alıştırmalar (1)

1. `Ogrenci` : `private` `ad` ve `not` ; not yalnızca 0–100 arası kabul edilsin; `getMi()` `not` $\geq$ 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

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: Sarmalama

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