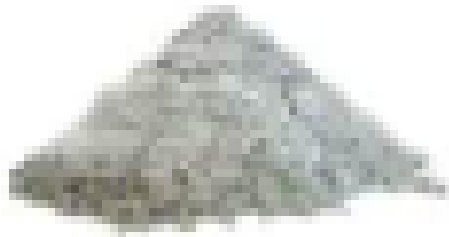
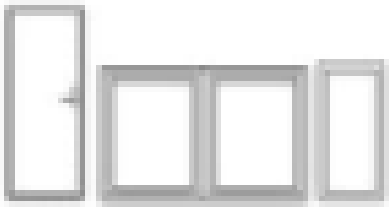




# SINIFLAR

CLASS

# Object Oriented Programming



# SINIFLAR

## Tuğla Sınıfı

Özellikler; en- boy-yükseklik  
İşlevler; hacim

d1

en: 10  
boy: 15  
yükseklik:8

d2

en: 20  
boy: 30  
yükseklik:10

d3

en: 5  
boy: 7  
yükseklik:3

# SINIFLAR

Erisim Belirleyici class **sinifAdi()**

{

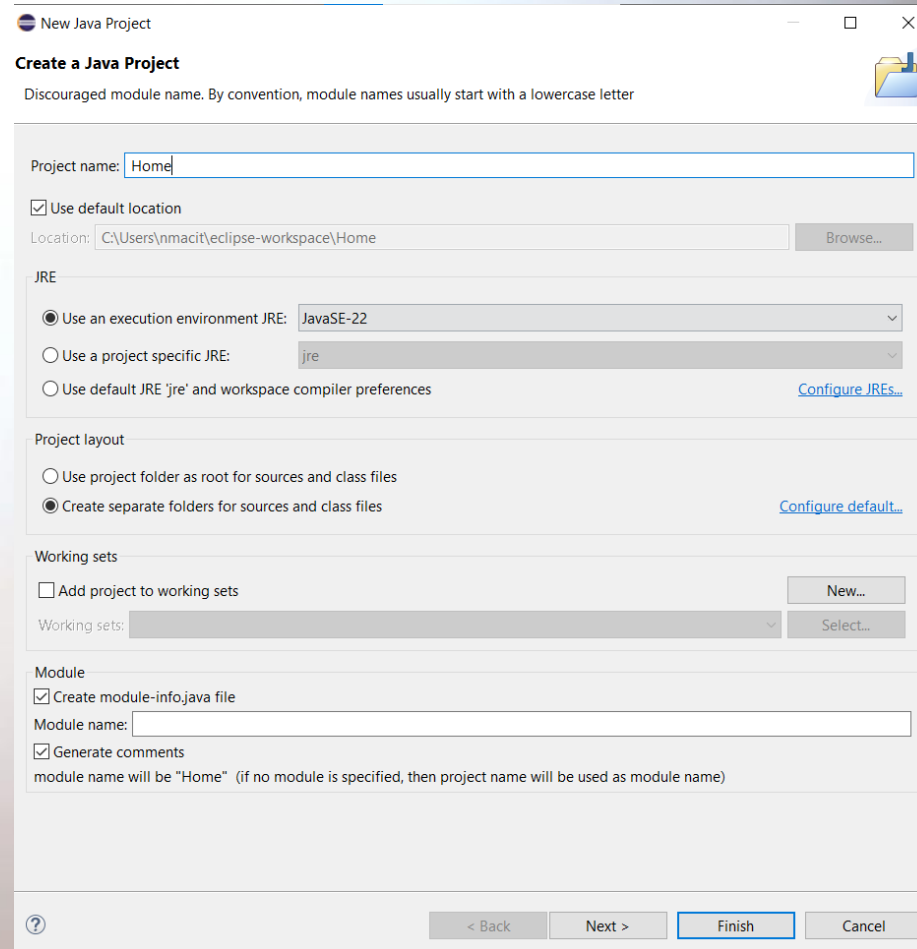
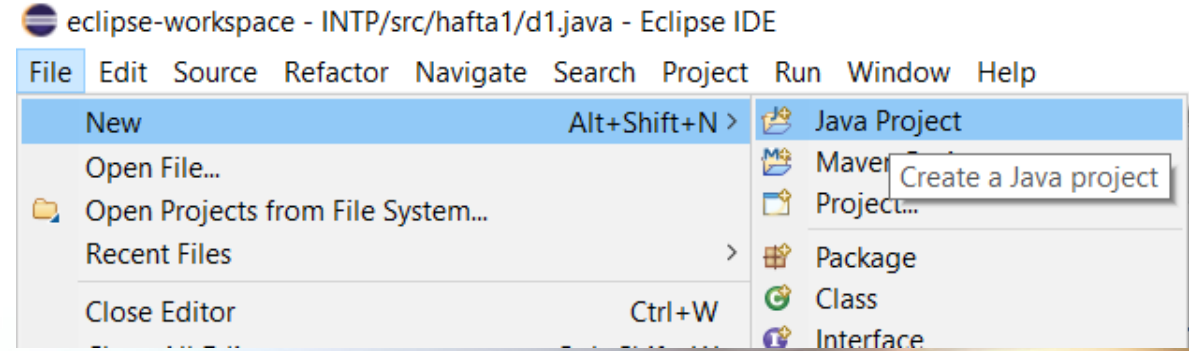
..

Sınıf ile ilgili parametreler

}

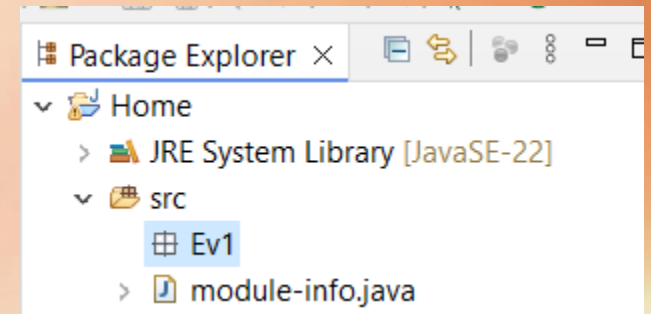
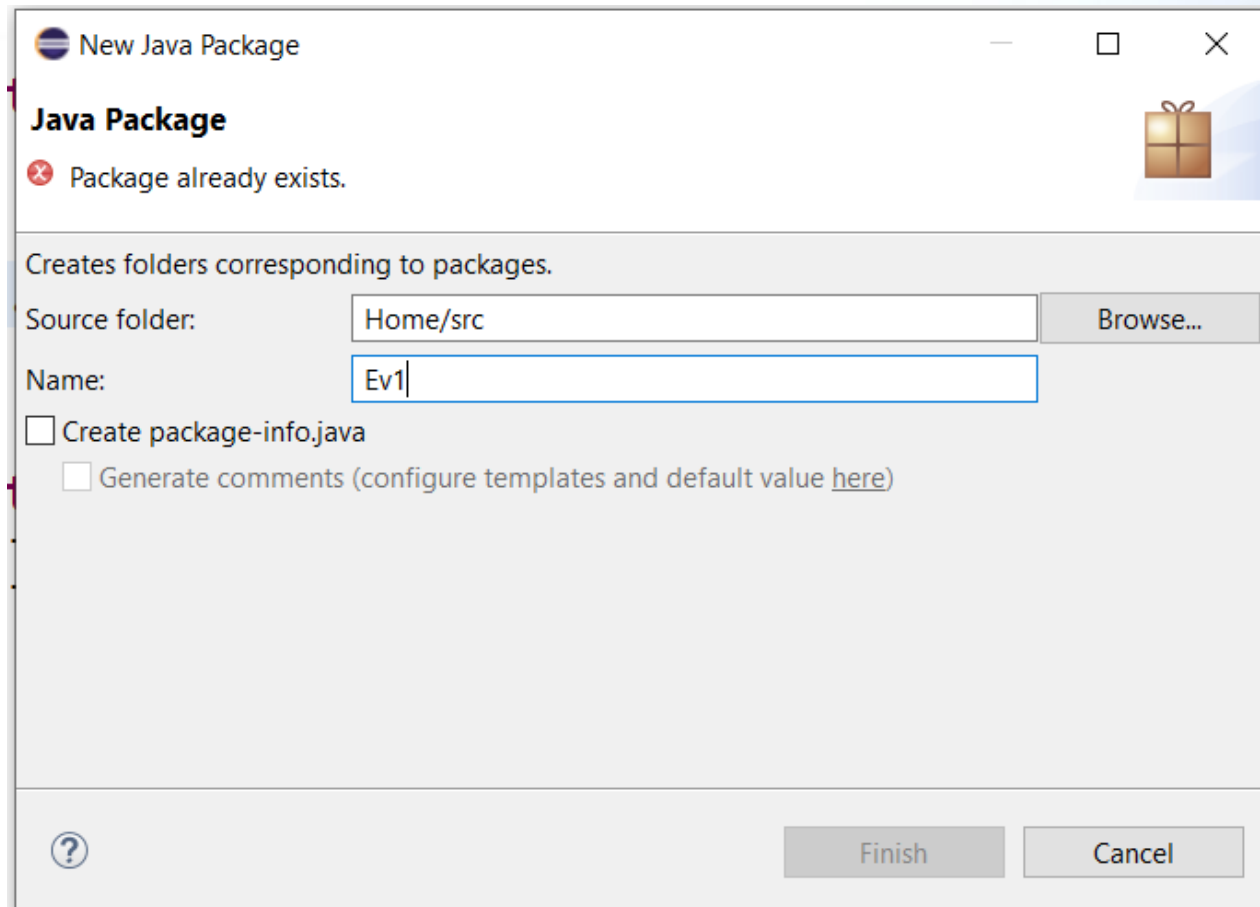
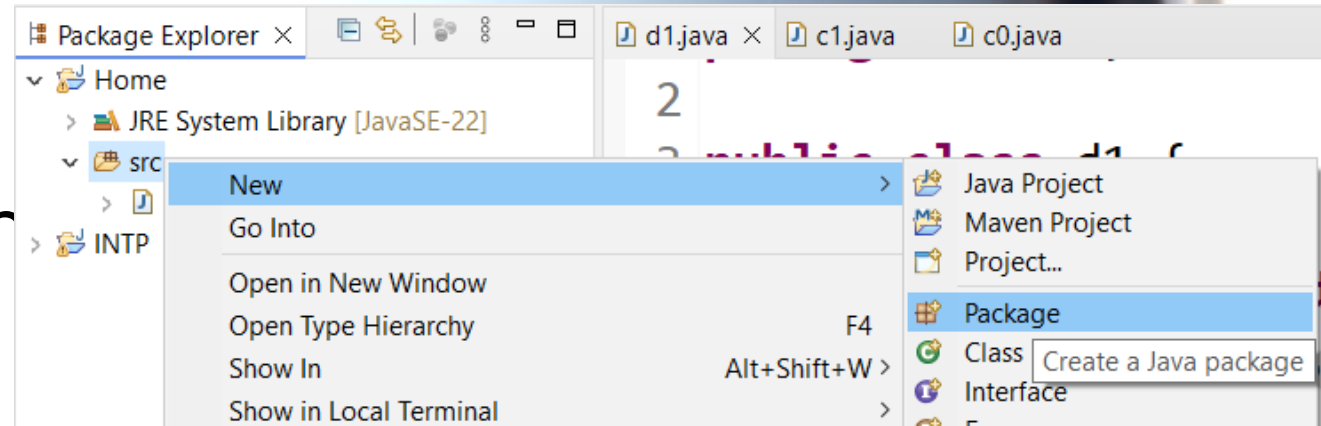
# SINIFLAR

Yeni bir proje oluřturalım  
Adını "ev" koyalım.



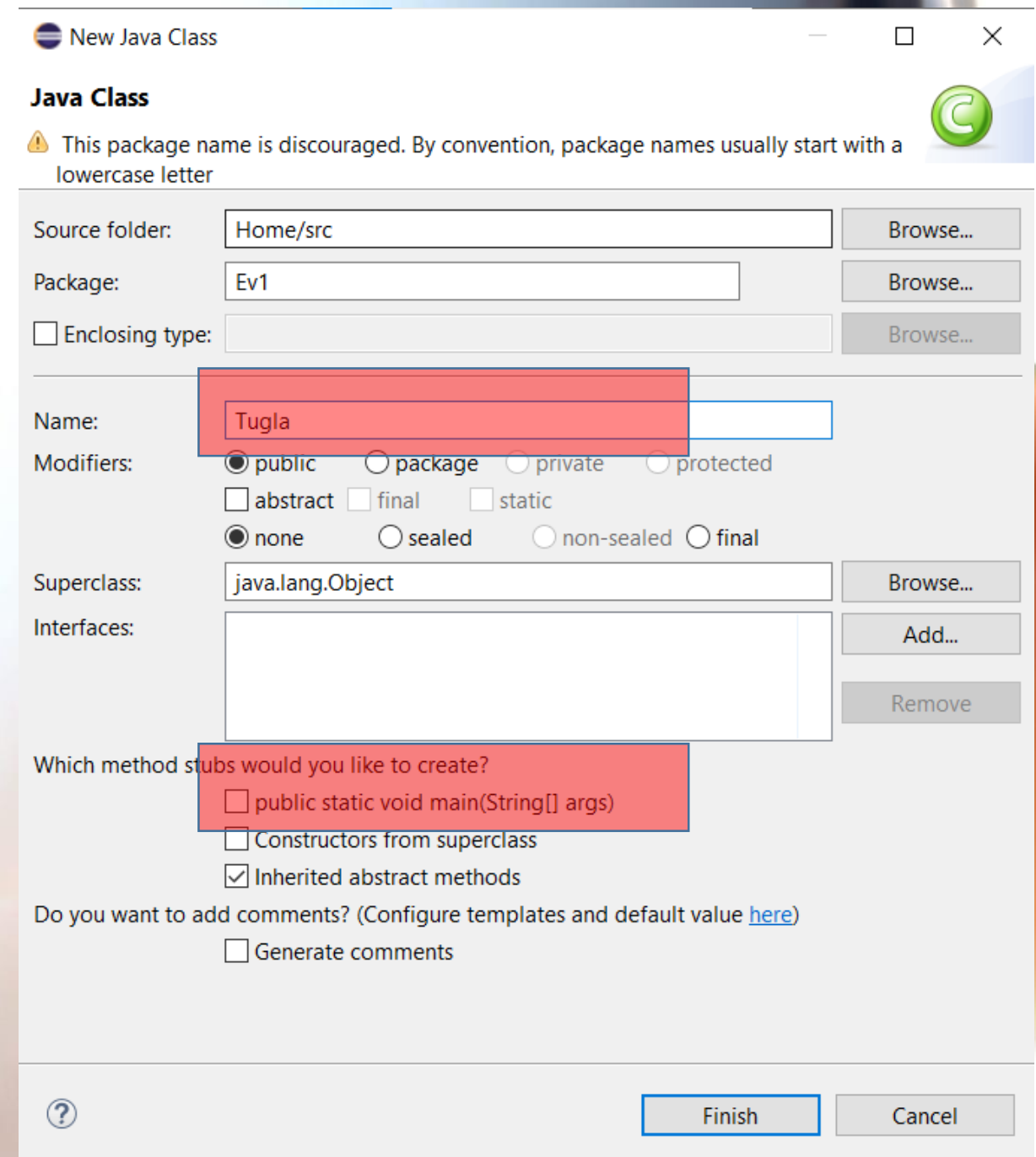
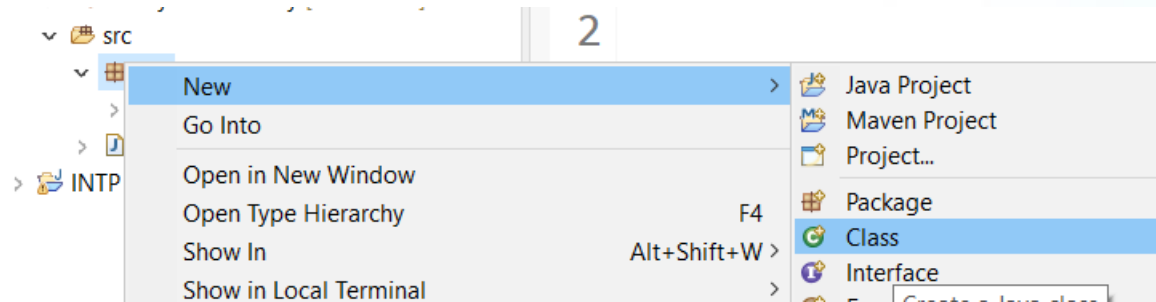
# SINIFLAR

Yeni bir paket oluşturalım  
Adını "Ev1" koyalım.




# SINIFLAR

Ev1 paketi içerisinde  
bir class oluşturalım  
Adını "Tugla" koyalım.



# SINIFLAR

A close-up, artistic photograph of a hand holding a pen, with the pen tip pointing towards the code editor window. The background is a soft, out-of-focus gradient of light blue and orange.

```
Tugla.java ×  
1 package Ev1;  
2  
3 public class Tugla {  
4  
5 }  
6
```

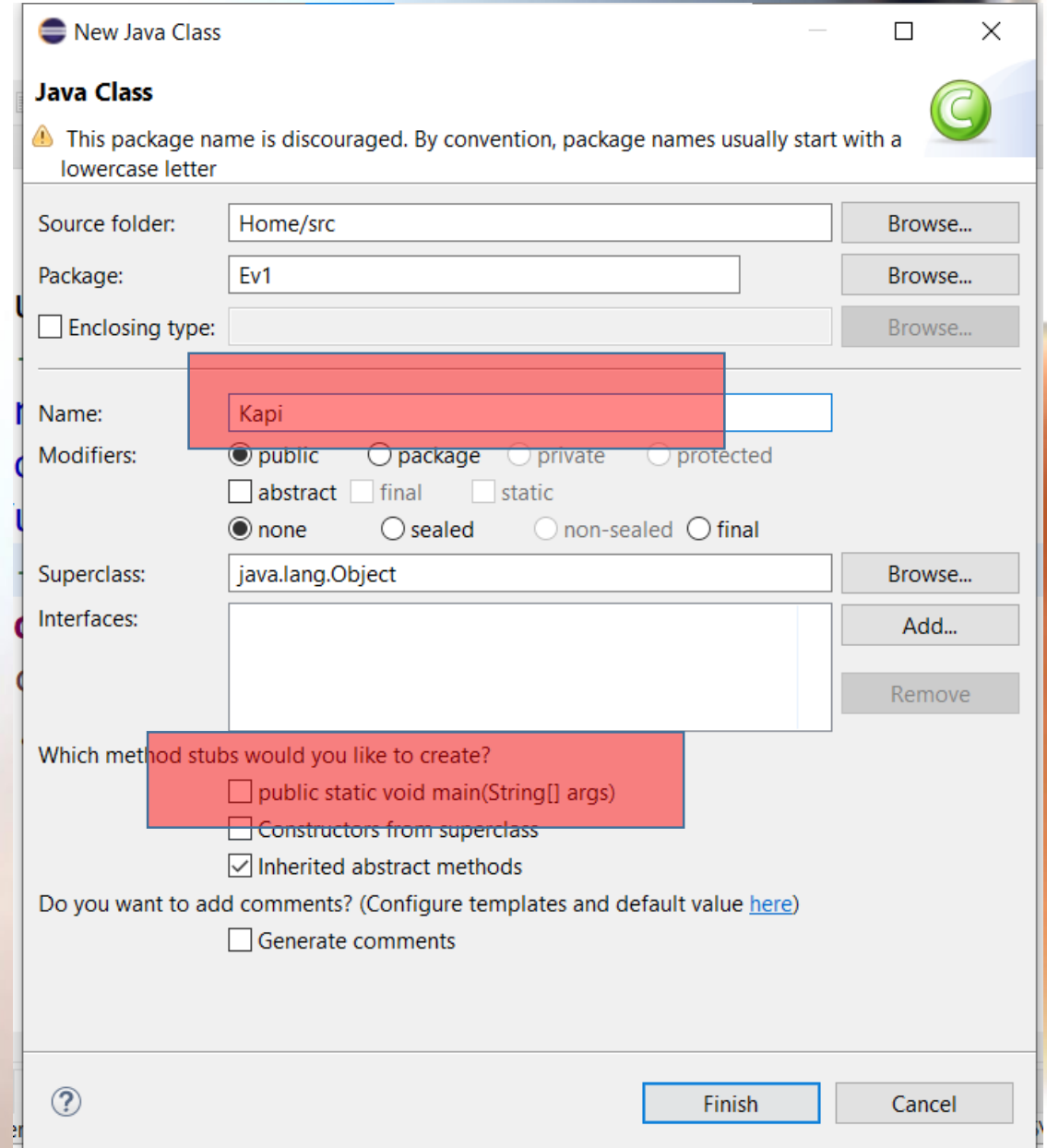
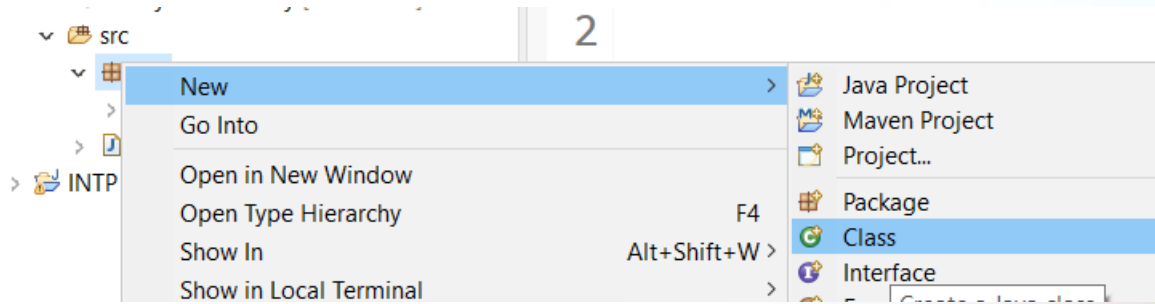
# SINIFLAR

Tugla.java ×

```
1 package Ev1;
2
3 public class Tugla {
4     //variable-değişkenler-Sınıfın pasif özellikleri
5     int tuglaEn=10;
6     int tuglaBoy=5;
7     int tuglaYukseklık=4;
8     //metotlar-Sınıfın eylemlerini gerçekleştiren öğeler
9     public void hacim () {
10         int hacim=tuglaBoy*tuglaEn*tuglaYukseklık;
11         System.out.println("Tuğlanın hacmi:"+hacim);
12     }
13 }
14
```

# SINIFLAR

Ev1 paketi içerisinde  
bir class oluşturalım  
Adını "Kapi" koyalım.



```
1 package Ev1;
2
3 public class Kapi {
4     int kapiYukseklik=190;
5     int kapiGenislik=90;
6     String kapiAcilisYonu="Sağ";
7     String kapiRengi="Kahverengi";
8
9     public void kapiAc() {
10         if(kapiAcilisYonu.equals("Sağ")) {
11             System.out.println("Kapı sağa açılan kapıdır");
12         }
13         else
14         {
15             System.out.println("Kapı sola açılan kapıdır");
16         }
17     }
18 }
```

# SINIFLAR

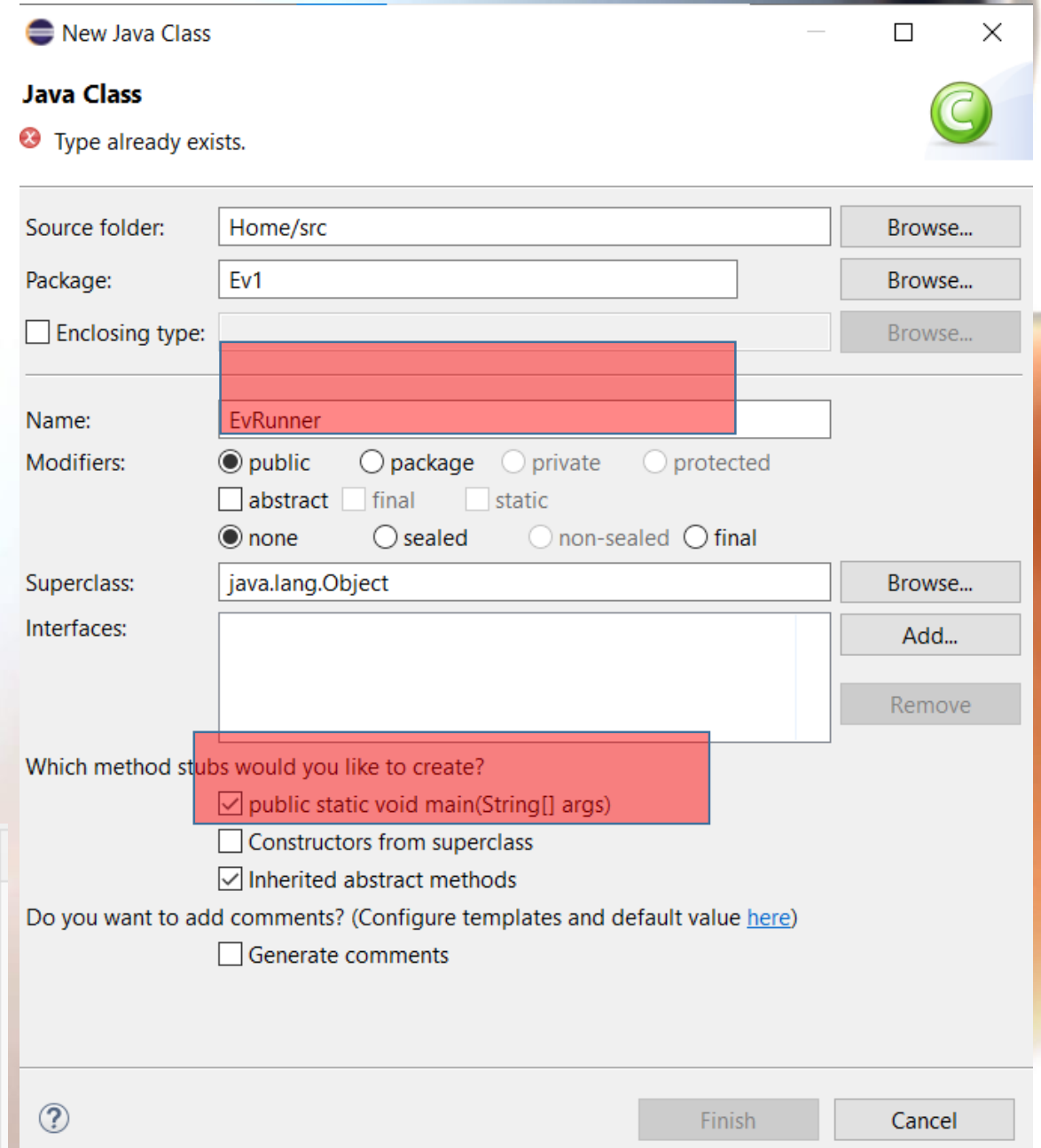
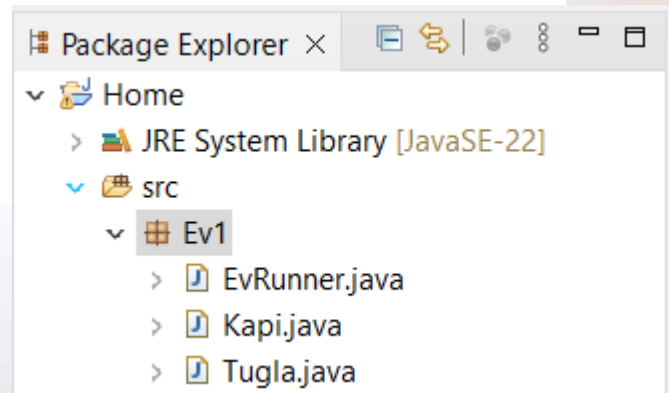
1- Bir veri yapısı veya yardımcı bir class oluştururken, genellikle main methoduna ihtiyaç duymazsınız.

2- Ancak bir uygulama geliştirirken veya bağımsız bir program çalıştırırken, main metodu oluşturmanız gerekebilir.

3- Gerçek projelerde genelde birçok class olur ve sadece 1 tane main metotlu class olur. Adına da "runner" eklenir.

# SINIFLAR

Gerçek projelerde genelde birçok class olur ve sadece 1 tane main method'lu class olur. Adına da "runner" eklenir.



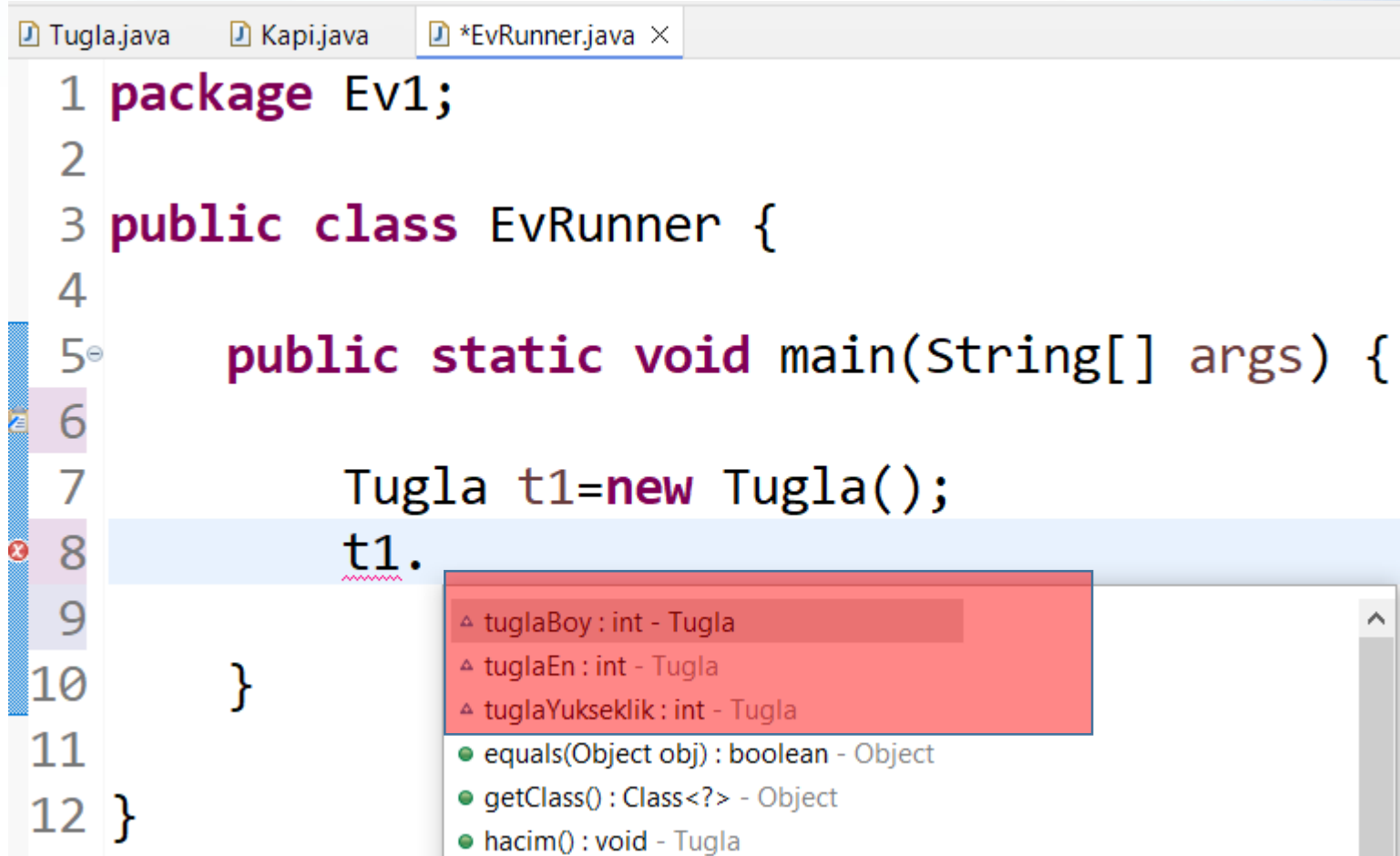
# SINIFLAR

Tugla.java Kapi.java \*EvRunner.java ×

```
1 package Ev1;
2
3 public class EvRunner {
4
5     public static void main(String[] args) {
6
7         Tugla t1=new Tugla();|
8
9     }
10
11 }
12
```

# SINIFLAR

Özelliklerine . Koyarak ulaşırız.



```
1 package Ev1;
2
3 public class EvRunner {
4
5     public static void main(String[] args) {
6
7         Tugla t1=new Tugla();
8         t1.
9
10    }
11
12 }
```

- △ tuglaBoy : int - Tugla
- △ tuglaEn : int - Tugla
- △ tuglaYukseklik : int - Tugla
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hacim() : void - Tugla

Tugla.java Kapi.java \*EvRunner.java ×

```
1 package Ev1;
2
3 public class EvRunner {
4
5     public static void main(String[] args) {
6
7         Tugla t1=new Tugla();
8         t1.hacim(); //eylemini çağırdık
9         System.out.println(t1.tuglaBoy);//özellik çağırdık
10        System.out.println(t1.tuglaYukseklık);
11        System.out.println(t1.tuglaEn);
12
13    }
14
15 }
```

Problems Javadoc Declaration Console ×

<terminated> EvRunner [Java Application] C:\Users\nmacit\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_23.0.0.v20240919-1

Tuğlanın hacmi:200

5

4

10

Tugla.java ×

```
1 package Ev1;
2
3 public class Tugla {
4     //variable-değişkenler-Sınıfın pasif özellikleri
5     int tuglaEn=10;
6     int tuglaBoy=5;
7     int tuglaYukseklık=4;
8     //metotlar-Sınıfın eylemlerini gerçekleştiren öğeler
9     public void hacim () {
10         int hacim=tuglaBoy*tuglaEn*tuglaYukseklık;
11         System.out.println("Tuğlanın hacmi:"+hacim);
12     }
13 }
14
```

```
Tugla.java Kapi.java EvRunner.java x
1 package Ev1;
2
3 public class EvRunner {
4
5     public static void main(String[] args) {
6
7         Tugla t1=new Tugla();
8         t1.hacim(); //eylemine çağırıldık
9         System.out.println(t1.tuglaBoy);
10        Tugla t2=new Tugla();
11        t2.hacim();|
12        System.out.println(t2.tuglaBoy);|
13    }
14
15 }
16
```

Problems @ Javadoc Declaration Console x

<terminated> EvRunner [Java Application] C:\Users\nmacit\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.v

Tuğlanın hacmi:200  
5  
Tuğlanın hacmi:200  
5

```
Tugla.java x
1 package Ev1;
2
3 public class Tugla {
4     //variable-değişkenler-Sınıfın pasif özellikleri
5     int tuglaEn=10;
6     int tuglaBoy=5;
7     int tuglaYukseklık=4;
8     //metotlar-Sınıfın eylemlerini gerçekleştiren öğeler
9     public void hacim () {
10         int hacim=tuglaBoy*tuglaEn*tuglaYukseklık;
11         System.out.println("Tuğlanın hacmi:"+hacim);
12     }
13 }
14
```

Özellikler; en- boy-  
yükseklik  
İşlevler; hacim

t1

t2

en: 10  
boy: 5  
yükseklik:4

en: 10  
boy: 5  
yükseklik:4

# Constructor Nedir?

- Class'tan object oluşturmanın bölümüdür.
- Class oluşturduğumuzda Java'da default olarak bir constructor verir.
- Bu constructor'lara "default constructor" denir.

İNTP

```
1 package Ev1;
2
3 public class Tugla {
4     //variable-değişkenler-Sınıfın pasif özellikleri
5     int tuglaEn=10;
6     int tuglaBoy=5;
7     int tuglaYukseklik=4;
8     //metotlar-Sınıfın eylemlerini gerçekleştiren öğeler
9     public void hacim () {
10         int hacim=tuglaBoy*tuglaEn*tuglaYukseklik;
11         System.out.println("Tuğlanın hacmi:"+hacim);
12     }
13 }
14
```

```
Tugla.java  Kapi.java  *EvRunner.java x
1 package Ev1;
2
3 public class EvRunner {
4
5     public static void main(String[] args) {
6
7         Tugla t1=new Tugla();
8     }
```

# Constructor Nedir?

- Her defasında sabit özelliklerde nesne üretmek istemezsek?
- Yeni constructorlar oluştururuz.
- Kendiniz constructor'inizi oluşturduğunuz zaman Java default constructor'i siler
- Nasıl oluşturulur  
Access Modifier + **Class ismi** + () +  
{  
  
}

```
1 package hafta8.ders1;
2
3 public class Tugla {
4     //özellikleri
5     int tuglaEn=10;
6     int tuglaBoy=5;
7     int tuglaYukseklık=10;
8     //CONSTRUCTOR
9     public Tugla(int en, int boy, int yukseklik) {
10         this.tuglaEn=en;
11         this.tuglaBoy=boy;
12         this.tuglaYukseklık=yukseklık;
13     }
14
15     //metodları
16     public void hacim() {
17         int hacim=tuglaEn*tuglaBoy*tuglaYukseklık;
18         System.out.println("Tugla hacim: " + hacim);
19     }
20
21 }
```

```
1 package hafta8.ders1;
2
3 public class EvRunner {
4
5     public static void main(String[] args) {
6         Tugla d1= new Tugla();
7         //default constructor silindiği için hata veriyor
8     }
9
10 }
```

# Constructor Tanımlama

```
EvRunner.java  Tugla.java  Kapi.java
1 package hafta8.ders1;
2
3 public class Tugla {
4     //özellikleri
5     int tuglaEn=10;
6     int tuglaBoy=5;
7     int tuglaYukseklik=10;
8     //CONSTRUCTOR
9     public Tugla(int en, int boy, int yukseklik) {
10         this.tuglaEn=en;
11         this.tuglaBoy=boy;
12         this.tuglaYukseklik=yukseklik;
13     }
14
15     //metodları
16     public void hacim() {
17         int hacim=tuglaEn*tuglaBoy*tuglaYukseklik;
18         System.out.println("tuğlanın hacim: " + hacim);
19     }
20
21 }
```

- Artık tanımlama yaparken yeni constructor metodumuzu görüyoruz

```
*EvRunner.java  Tugla.java  Kapi.java
1 package hafta8.ders1;
2
3 public class EvRunner {
4
5     public static void main(String[] args) {
6         //Tugla d1= new Tugla();
7         //default constructor silindiği için hata veriyor
8         Tugla d1=new Tugla(10,5,10);
9     }
10
11 }
12 }
```

- Tugla(int en, int boy, int yukseklik) - hafta8.ders1.Tugla
- Tugla() - hafta8.ders2.Tugla

```
1 package Ev1;
2
3 public class EvRunner {
4
5     public static void main(String[] args) {
6
7         Tugla t1=new Tugla(8,6,9);
8         t1.hacim();
9         System.out.println(t1.tuglaBoy);
10        Tugla t2=new Tugla(5,4,2);
11        t2.hacim();
12        System.out.println(t2.tuglaBoy);
13    }
14
15 }
```

Problems Javadoc Declaration Console ×

<terminated> EvRunner [Java Application] C:\Users\nmacit\.p2\pool\plugins\or

Tuğlanın hacmi:432  
6  
Tuğlanın hacmi:40  
4

```
Tugla.java × Kapi.java × *EvRunner.java
1 package EV1;
2
3 public class Tugla {
4     //variable-değişkenler-Sınıfın pasif özellikleri
5     int tuglaEn=10;
6     int tuglaBoy=5;
7     int tuglaYukseklık=4;
8     //constructor
9     public Tugla(int tuglaEn, int tuglaBoy, int tuglaYukseklık) {
10         this.tuglaEn = tuglaEn;
11         this.tuglaBoy = tuglaBoy;
12         this.tuglaYukseklık = tuglaYukseklık;
13     }
14     //metotlar-Sınıfın eylemlerini gerçekleştiren öğeler
15     public void hacim () {
16         int hacim=tuglaBoy*tuglaEn*tuglaYukseklık;
17         System.out.println("Tuğlanın hacmi:"+hacim);
18     }
19 }
```

# Tüm Özellikleri göndermeye gerek yok

```
*Tugla.java x Kapi.java EvRunner.java
1 package Ev1;
2 public class Tugla {
3     //variable-değişkenler-Sınıfın pasif özellikleri
4     int tuglaEn=10;
5     int tuglaBoy=5;
6     int tuglaYukseklık=4;
7     //constructor1
8     public Tugla(int tuglaEn, int tuglaBoy, int tuglaYukseklık) {
9         this.tuglaEn = tuglaEn;
10        this.tuglaBoy = tuglaBoy;
11        this.tuglaYukseklık = tuglaYukseklık;
12    }
13    //constructor1
14    public Tugla(int tuglaEn, int tuglaBoy) {
15        this.tuglaEn = tuglaEn;
16        this.tuglaBoy = tuglaBoy;
17    }
```

# Default özelliklerde Constructor?

```
//CONSTRUCTOR 3
//Yeni bir constructor oluşturulduğunda default contrucotr silinir
//bu default constructor ı yeniden aktif etmek için onu da yeniden tanımlamanız gerekir.
/*bu şekilde tanımlandığında
 * default olarak tanımlanan tuglaEn değeri 10
 * tuglaBoy değeri 5
 * tuglaYukseklık değeri 10 olacaktır.
 *
 * yani başlangıçta verilen özellikler kullanılacaktır. */
public Tugla() {
}
```

```

1 package hafta8.ders2;
2
3 public class Tugla {
4     //özellikleri
5     int tuglaEn=10;
6     int tuglaBoy=5;
7     int tuglaYukseklik=10;
8     //CONSTRUCTOR 1
9     public Tugla(int en, int boy, int yukseklik) {
10         this.tuglaEn=en;
11         this.tuglaBoy=boy;
12         this.tuglaYukseklik=yukseklik;
13     }
14     //CONSTRUCTOR 2
15     //Constructor lar aslında aşırı yüklenmiş fonksiyonlara örnektir
16     //parametre sayıları ya da parametre türleri birbirinden farklı
17     //birden fazla constructor oluşturulabilir.
18     public Tugla(int en, int boy) {
19         this.tuglaEn=en;
20         this.tuglaBoy=boy;
21     }
22     //CONSTRUCTOR 3
23     //Yeni bir constructor oluşturulduğunda default constructor silinir
24     //bu default constructor ı yeniden aktif etmek için onu da yeniden
25     /*bu şekilde tanımlandığında
26     * default olarak tanımlanan tuglaEn değeri 10
27     * tuglaBoy değeri 5
28     * tuglaYukseklik değeri 10 olacaktır.
29     *
30     * yani başlangıçta verilen özellikler kullanılacaktır. */
31     public Tugla() {
32
33     }
34

```

```

Tugla.java EvRunner.java
1 package hafta8.ders2;
2
3 public class EvRunner {
4
5     public static void main(String[] args) {
6         //Tugla d1= new Tugla();
7         //default constructor silindiği için hata veriyor
8         Tugla t1=new Tugla(8,9,5);
9         t1.hacim();
10        System.out.println(t1.tuglaYukseklik);
11        Tugla t2=new Tugla(5, 4);
12        t2.hacim();
13        System.out.println(t2.tuglaYukseklik);
14        Tugla t3=new Tugla();
15        t3.hacim();
16        System.out.println(t3.tuglaYukseklik);
17    }
18
19 }
20

```

Console

```

<terminated> EvRunner (1) [Java Application] C:\Users\BIDB\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v202
tuğlanın hacmi:360
5
tuğlanın hacmi:200
10
tuğlanın hacmi:500
10

```

# Soru

## 1.Öğrenci Sınıfı (Ogrenci):

•Öğrenciye ait şu bilgileri içeren özelliklere sahip olmalıdır:

- adSoyad (String)
- sinif (int)
- tcNo (String)
- ders (String)
- dersNotu (double)

## Constructor:

•**1. Constructor:** Tüm özellikleri parametre olarak alan bir constructor tanımlayın.

•**2. Constructor:** Sadece adSoyad, sınıf ve tcNo alan bir constructor tanımlayın. Bu constructor'da ders ve dersNotu için varsayılan değerler atayın (örneğin, ders = "Henüz atanmadı" ve dersNotu = 0).

# Soru

## Metotlar:

- notDurumu()**: Ders notu 50 ve üzerindeyse "Geçti", aksi halde "Kaldı" mesajı döndüren metot.
- harfNotu()**: Ders notuna göre harf notu döndüren metot:
  - 90-100: "AA"
  - 80-89: "BA"
  - 70-79: "BB"
  - 60-69: "CB"
  - 50-59: "CC"
  - 0-49: "FF"
- bilgileriGoster()**: Öğrenci bilgilerini ekrana yazdıran metot.

## 1.Main Sınıfı:

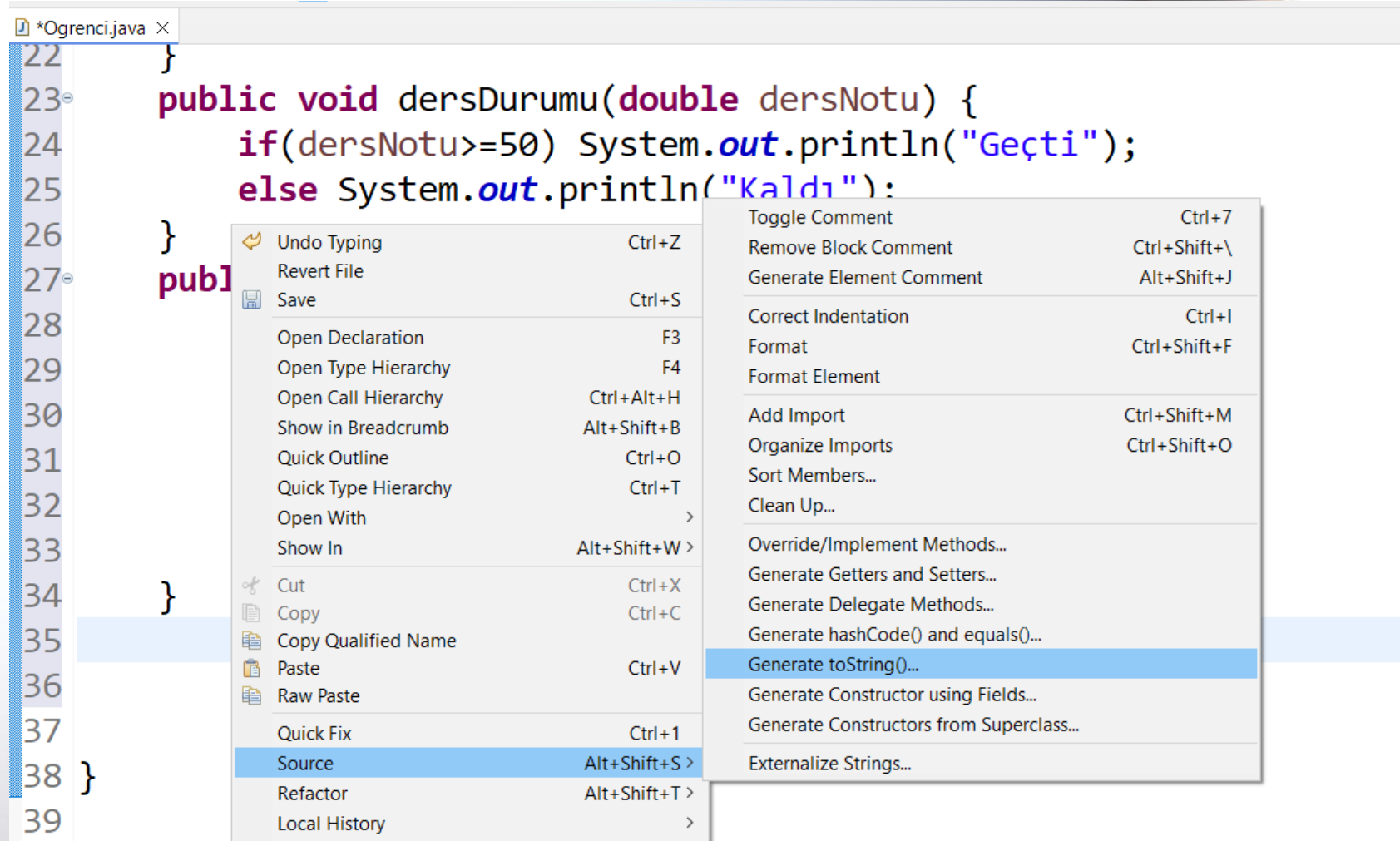
- main metodunda, her iki constructor'ı da kullanarak en az iki Öğrenci nesnesi oluşturun ve bu nesneler için bilgileriGoster() ve notDurumu() metodlarını çağırarak sonuçları ekrana yazdırın.

## Örnek Çıktı:

Ad Soyad: Ahmet Yılmaz Sınıf: 10 TC No: 12345678901 Ders: Matematik Ders Notu: 75.5 Durum: Geçti Harf Notu: BB Ad Soyad: Elif Demir Sınıf: 9 TC No: 10987654321 Ders: Henüz atanmadı Ders Notu: 0.0 Durum: Kaldı Harf Notu: FF

# ToString Metodu

- Java'da toString metodu genellikle bir nesnenin içeriğini okunabilir bir String olarak döndürmek için kullanılır.



# ToString Metodu

```
@Override
public String toString() {
    return "Ogrenci [adSoyad=" + adSoyad +
        ", sinif=" + sinif +
        ", tcNo=" + tcNo +
        ", ders=" + ders +
        ", dersNotu="+ dersNotu + "]";
}
```

# ToString Metodu- Runner Da Kullanımı

```
1 package hafta3_;
2
3 public class OgrenciRunner {
4
5     public static void main(String[] args) {
6         Ogrenci o1= new Ogrenci("Elif Demir", 9, "10987654321");
7         Ogrenci o2 = new Ogrenci("Ahmet Yılmaz", 10,
8 "10987654321", "Matematik", 75.5);
9         System.out.println(o2.toString());
10        System.out.println(o1.toString());
11
12
13    }
14
15 }
```

# PASS BY VALUE

**Referans türleri** (nesneler) de pass by value ile aktarılır, ancak burada dikkat edilmesi gereken nokta, referansın kendisinin bir kopyasının aktarılmasıdır. Yani metot içinde bu referans üzerinden nesnenin içeriğine erişilebilir ve bu içerik değiştirilebilir, fakat referansın kendisi değiştirildiğinde orijinal nesne etkilenmez.

# PASS BY VALUE

NewClass.java × \*PassByValue3.java

```
1 package hafta3_;  
2  
3 public class NewClass {  
4     int value;  
5 }  
6
```

# PASS BY VALUE

## Adım adım ne oluyor?

1.obj adında bir referansımız var ve bu referans, bellekte bir NewClass nesnesini işaret ediyor. Bu nesnenin value adlı bir alanı var ve başlangıçta 20 olarak ayarlanmış.

2.degistirNesne(obj) metodu çağrıldığında:

- obj referansının bir kopyası metota geçiyor. Bu, aynı nesneyi işaret eden ikinci bir referans gibi davranır.
- Bu referans üzerinden object.value = 20; satırında nesnenin içindeki value alanı değiştiriliyor. Dolayısıyla, orijinal nesne etkilenir ve değeri 20 olur.

```
NewClass.java  *PassByValue3.java x
1 package hafta3_;
2
3 public class PassByValue3 {
4
5     public static void main(String[] args) {
6         NewClass obj=new NewClass();
7         obj.value=20;
8         degistirNesne(obj);
9         System.out.println(obj.value);
10
11     }
12     private static void degistirNesne(NewClass obj) {
13         obj.value=30;
14     }
15
16 }
17
```

# PASS BY VALUE

degistirReferans(obj) metodu çağrıldığında:

- Yine obj referansının bir kopyası metota geçiyor. Ancak bu metotta obj = new NewClass(); diyerek kopya referansı yeni bir nesneye yönlendiriyoruz. Yani, object artık farklı bir MyClass nesnesini işaret ediyor.
- Ama bu yeni nesne, sadece object referansının kopyasını etkiler. Orijinal obj referansı hala ilk nesneyi işaret etmeye devam eder.
- Bu nedenle, object.value = 50; satırında yeni nesnenin value değeri değiştirilir ama obj referansı hala ilk nesneyi gösterir. İlk nesnenin value değeri hala 20 olarak kalır.

```
NewClass.java  PassByValue3.java x
4
5  public static void main(String[] args) {
6      NewClass obj=new NewClass();
7      obj.value=20;
8      degistirNesne(obj);
9      System.out.println(obj.value);
10     degistirReferans(obj);
11     System.out.println(obj.value);
12
13 }
14 private static void degistirNesne(NewClass obj) {
15     obj.value=30;
16 }
17 private static void degistirReferans(NewClass obj) {
18     obj=new NewClass();
19     obj.value=50;
20 }
21
```

# ACCESS MODIFIER-Erişim Belirleyici

- 1)public
- 2)protected
- 3)default
- 4)private

public > protected > default > private

# ACCESS MODIFIER-Erişim Belirleyici

==> **public** olanlar her class'dan kullanılabilir

==> **protected** olanlar başka package'lardan kullanılamaz, ancak başka package'larda child classlar içinden kullanılabilir

**Access Modifiers**

Modifier	Class	Package	SubClass	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Default	✓	✓	✓	✗
private	✓	✗	✗	✗

# ACCESS MODIFIER-Erişim Belirleyici

- Public:**

- Örnek: Bir üniversite kampüsündeki kütüphane gibi düşünün. Kütüphane, kampüsteki herkesin erişimine açık. İsteyen herkes içeri girebilir ve kitaplara ulaşabilir. Yani, sınıf veya metotlar "public" olarak tanımlandığında, uygulamanın her yerinden erişilebilirler.

- Private:**

- Örnek: Evinizin içindeki özel bir kasa gibi düşünün. Bu kasa, sadece anahtarı olan kişi tarafından açılabilir. Dışarıdan kimse kasaya doğrudan erişemez. "Private" erişim belirleyici de benzer şekilde, bir sınıf içindeki private metot veya değişkenlere sadece o sınıfın içinde erişilebilir.

**Access Modifiers**

Modifier	Class	Package	SubClass	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Default	✓	✓	✓	✗
private	✓	✗	✗	✗

# ACCESS MODIFIER-Erişim Belirleyici

==> **default** olanlar başka package'lerden kullanılamazlar

==> **private** olanlar sadece oluşturuldukları class içinden kullanılabilirler. (**Package-Private**)

**Access Modifiers**

Modifier	Class	Package	SubClass	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Default	✓	✓	✓	✗
private	✓	✗	✗	✗

# ACCESS MODIFIER-Erişim Belirleyici

- **Protected:**
- Örnek: Bir ailedeki aile albümü gibi düşünebiliriz. Bu albümü yalnızca aile üyeleri görebilir. Aile dışından kimse albüme erişemez, ama aile içinde miras kalan kişiler (örneğin çocuklar) bu albüme erişebilir. "Protected", bir sınıftaki üye değişken ya da metodun sadece aynı paketdeki diğer sınıflar ve o sınıftan türetilen alt sınıflar tarafından erişilebilir olduğunu ifade eder.

**Access Modifiers**

Modifier	Class	Package	SubClass	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Default	✓	✓	✓	✗
private	✓	✗	✗	✗

# ACCESS MODIFIER-Erişim Belirleyici

- **Default (Package-Private):**
- Örnek: Apartman dairesinin ortak kullanım alanları gibi düşünün. Bu alanlara sadece apartmanda yaşayanlar erişebilir, ama dışarıdan biri erişemez. Java'da bir sınıf veya üye herhangi bir access modifier ile tanımlanmadığında, varsayılan olarak "package-private" olur ve sadece aynı paket içindeki sınıflar erişebilir.

**Access Modifiers**

Modifier	Class	Package	SubClass	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Default	✓	✓	✓	✗
private	✓	✗	✗	✗

```

1 package hafta3_;
2
3 public class AccessModifiers1 {
4     public String publicDegisken="Kütüphane gibi";
5     private String privateDegisken="kasa gibi";
6     protected String protectedDegisken="albüm";
7     String defaultDegisken="apartman gibi";
8 }

```

Aynı Proje  
içindeler.

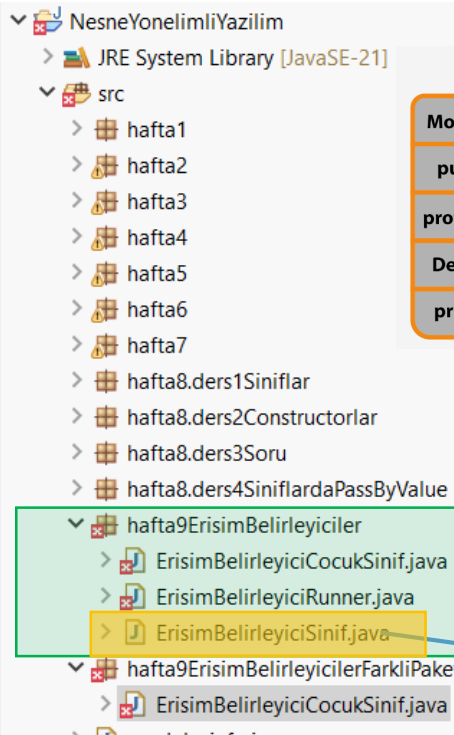
### Access Modifiers

Modifier	Class	Package	SubClass	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
private	✓	✗	✗	✗

```

1 package hafta2_PassByValue_Classes_Const;
2
3 import hafta3_.AccessModifiers1;
4
5 public class AccessModifierGlobal {
6     public static void main(String[] args) {
7         AccessModifiers1 a1=new AccessModifiers1();
8         System.out.println(a1.publicDegisken);
9         System.out.println(a1.privateDegisken);
10        System.out.println(a1.protectedDegisken);
11        System.out.println(a1.defaultDegisken);
12    }
13

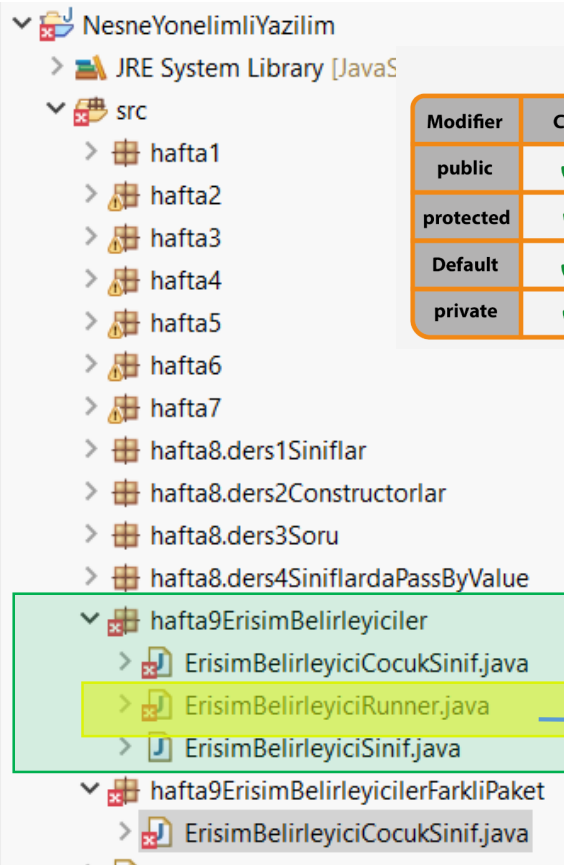
```



Access Modifiers				
Modifier	Class	Package	SubClass	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Default	✓	✓	✓	✗
private	✓	✗	✗	✗

Projenin tamamı NesneYoneliimliYazilim  
hafta9ErisimBerlileyiciler paketi altında  
bir sınıf oluşturunuz.  
ErisimBelirleyiciSinif

```
*ErisimBelirleyiciSinif.java  ErisimBelirleyiciRunner.java  ErisimBelirleyiciCocukSinif.java  ErisimBelirleyiciCocukSinif.java
1 package hafta9ErisimBelirleyiciler;
2
3 public class ErisimBelirleyiciSinif {
4     public String publicDegisken="kütüphane gibi her yerden ulaşım var";//her yerden erişim
5     private String privateDegisken="kasa gibi "; //"Sadece tanımlandığı yerden erişim var"
6     protected String protectedDegisken="aile albümü"; //aileye dahil olan childlar görecekt
7     String defaultDegisken="apartman gibi";
8
9     /*AYNI SINIFIN İÇERİSİNDE YUKARIDAKİ
10    * BÜTÜN ERIŞİM BELİRLEYİCİ DEĞİŞKENLERE ULAŞABİLİRİM.
11    * */
12     public void sinifIciFonksiyon() {
13         System.out.println(publicDegisken); //her yerden ulaşılabilir olduğu için ✓
14         System.out.println(protectedDegisken); //AYNI SINIF İÇİNDE olduğu için ✓
15         System.out.println(defaultDegisken); //AYNI SINIF İÇİNDE olduğu için ✓
16         System.out.println(privateDegisken); //AYNI SINIF İÇİNDE olduğu için ✓
17     }
```

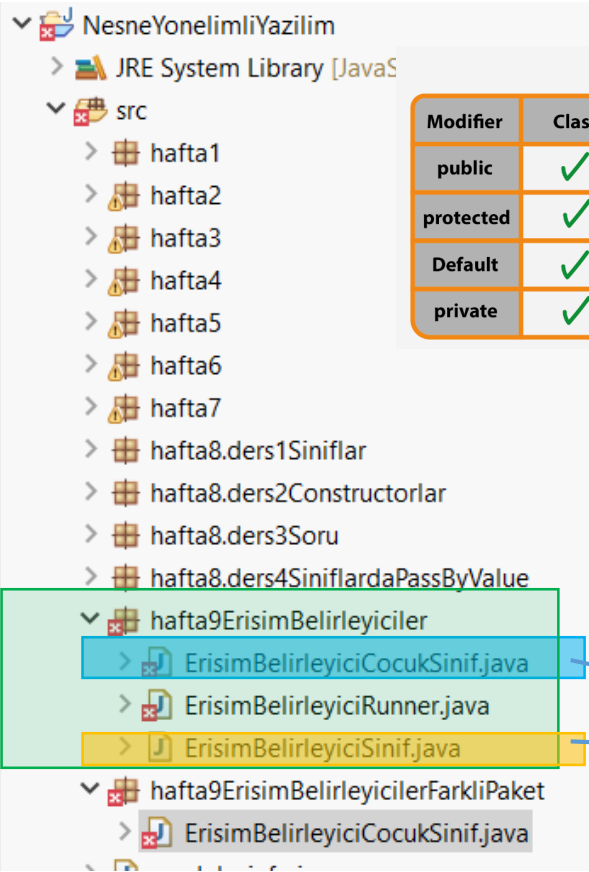


Access Modifiers

Modifier	Class	Package	SubClass	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Default	✓	✓	✓	✗
private	✓	✗	✗	✗

Projenin tamamı **NesneYonelimliYazilim**  
**hafta9ErisimBerlileyiciler** paketi altında  
başka bir sınıf ya da runner oluşturuyoruz.  
ErisimBelirleyiciRunner

```
ErisimBelirleyiciSinif.java  *ErisimBelirleyiciRunner.java  ErisimBelirleyiciCocukSinif.java  ErisimBelirleyiciCocukSinif.java
1 package hafta9ErisimBelirleyiciler;
2
3 public class ErisimBelirleyiciRunner {
4     public static void main(String[] args) {
5         ErisimBelirleyiciSinif e1=new ErisimBelirleyiciSinif();
6         System.out.println(e1.publicDegisken); //her yerden ulaşılabilir olduğu için ✓
7         System.out.println(e1.protectedDegisken); //AYNI PAKET İÇİNDE olduğu için ✓
8         System.out.println(e1.defaultDegisken); //AYNI SINIF İÇİNDE olduğu için ✓
9         System.out.println(e1.privateDegisken);
10        //AYNI PAKET İÇİNDE olsa BİLE private içinde bulunduğu sınıf a ait olduğu için ✗
11
12    }
```



Access Modifiers

Modifier	Class	Package	SubClass	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Default	✓	✓	✓	✗
private	✓	✗	✗	✗

Projenin tamamı **NesneYoneliYazilim**  
**hafta9ErisimBelirleyiciler** paketi altında  
başka bir subclass=çocuksınıf  
oluşturuyoruz.  
**ErisimBelirleyiciCocukSinif**

```
ErisimBelirleyiciSinif.java  ErisimBelirleyiciRunner.java  ErisimBelirleyiciCocukSinif.java  ErisimBelirleyiciCocukSinif.java
1 package hafta9ErisimBelirleyiciler;
2
3 public class ErisimBelirleyiciCocukSinif extends ErisimBelirleyiciSinif {
4 //extends ErisimBelirleyiciSinif İFADESİ
5 //extends ErisimBelirleyiciSinif SINIFININ ÇOCUĞU OLDUĞU
6 // YANI SUBCLASS I OLDUĞUNU BELİRTİR.
7 public void subClassFonksiyon() {
8     ErisimBelirleyiciSinif e1=new ErisimBelirleyiciSinif();
9     System.out.println(e1.publicDegisken); //her yerden ulaşılabilir olduğu için ✓
10    System.out.println(e1.protectedDegisken); //AYNI PAKET İÇİNDE olduğu için ✓
11    System.out.println(e1.defaultDegisken); //AYNI PAKET İÇİNDE olduğu için ✓
12    System.out.println(e1.privateDegisken);
13    //AYNI PAKET İÇİNDE olsa BİLE private içinde bulunduğu sınıf a ait olduğu için ✗
14    //SUBCLASS I olsa BİLE private içinde bulunduğu sınıf a ait olduğu için ✗
15 }
```

NesneYoneliYazilim

JRE System Library [JavaS

src

hafta1  
hafta2  
hafta3  
hafta4  
hafta5  
hafta6  
hafta7  
hafta8.ders1Siniflar  
hafta8.ders2Constructorlar  
hafta8.ders3Soru  
hafta8.ders4SiniflardaPassByValue

hafta9ErisimBelirleyiciler  
ErisimBelirleyiciCocukSinif.java  
ErisimBelirleyiciRunner.java  
ErisimBelirleyiciSinif.java

hafta9ErisimBelirleyicilerFarkliPaket  
ErisimBelirleyiciCocukSinif.java

Access Modifiers

Modifier	Class	Package	SubClass	Global
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
Default	✓	✓	✓	✗
private	✓	✗	✗	✗

Projenin tamamı NesneYoneliYazilim  
hafta9ErisimBelirleyicilerFarkliPaket  
paketi altında başka bir  
subclass=çocuksınıf oluşturuyoruz.  
ErisimBelirleyiciCocukSinif

```
ErisimBelirleyiciSinif.java ErisimBelirleyiciRunner.java ErisimBelirleyiciCocukSinif.java *ErisimBelirleyiciCocukSinif.java X
1 package hafta9ErisimBelirleyicilerFarkliPaket;
2
3 import hafta9ErisimBelirleyiciler.ErisimBelirleyiciSinif;
4 //FARKLI PAKETLERDE OLDUKLARI İÇİN
5 //YUKARIDAKİ KODU EKLEMEDİKÇE EXTENDS ÇALIŞMAYACAKTI
6
7 public class ErisimBelirleyiciCocukSinif extends ErisimBelirleyiciSinif {
8 //extends ErisimBelirleyiciSinif İFADESİ
9 //extends ErisimBelirleyiciSinif SINIFININ ÇOCUĞU OLDUĞU
10 // YANİ SUBCLASS I OLDUĞUNU BELİRTİR.
11 public void subClassFonksiyon() {
12 ErisimBelirleyiciSinif e1=new ErisimBelirleyiciSinif();
13 System.out.println(e1.publicDegisken); //her yerden ulaşılabilir olduğu için ✓
14 System.out.println(e1.protectedDegisken); //FARKLI PAKET İÇİNDE olduğu için ✗
15 System.out.println(e1.defaultDegisken); //FARKLI PAKET İÇİNDE olduğu için ✗
16 System.out.println(e1.privateDegisken);
17 //AYNI PAKET İÇİNDE olsa BİLE private içinde bulunduğu sınıf a ait olduğu için ✗
18 //SUBCLASS I olsa BİLE private içinde bulunduğu sınıf a ait olduğu için ✗
19
```

# STATIC

◆ **static class member'lara** (değişken veya metotlara) ulaşmak için **obje oluşturmak gerekmez.**

◆ Buna karşılık, **non-static (örnek seviyesindeki)** class member'lara erişmek için **obje oluşturmak zorunludur.**

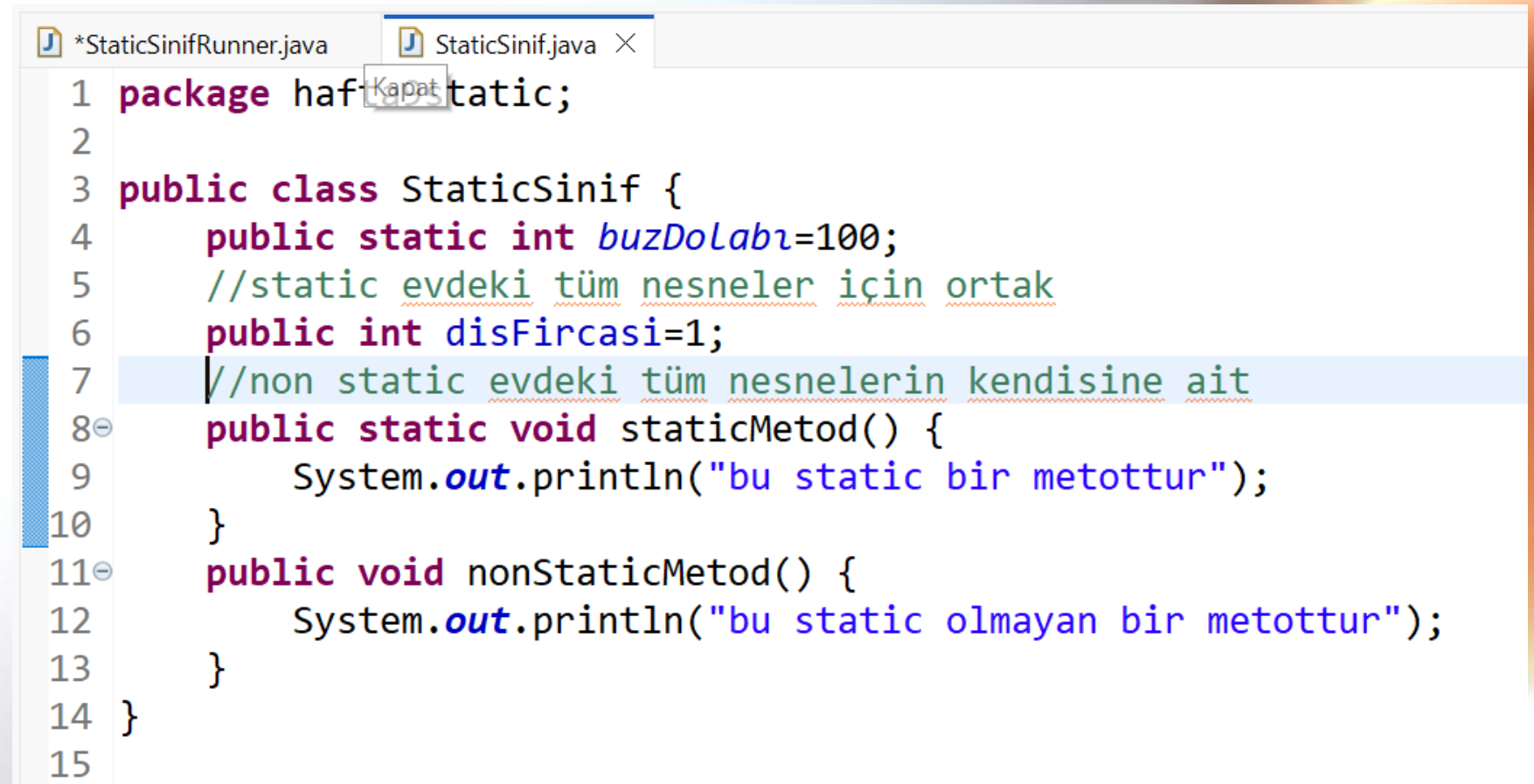
# STATIC METOD- VARIABLE

Bir sınıf oluşturalım

Bu sınıfta iki tane metod var ve 2 tane değişken var

Biri static

Biri non static



```
1 package hafıza;
2
3 public class StaticSinif {
4     public static int buzDolabi=100;
5     //static evdeki tüm nesneler için ortak
6     public int disFircasi=1;
7     //non static evdeki tüm nesnelerin kendisine ait
8     public static void staticMetod() {
9         System.out.println("bu static bir metottur");
10    }
11    public void nonStaticMetod() {
12        System.out.println("bu static olmayan bir metottur");
13    }
14 }
15
```

# STATIC

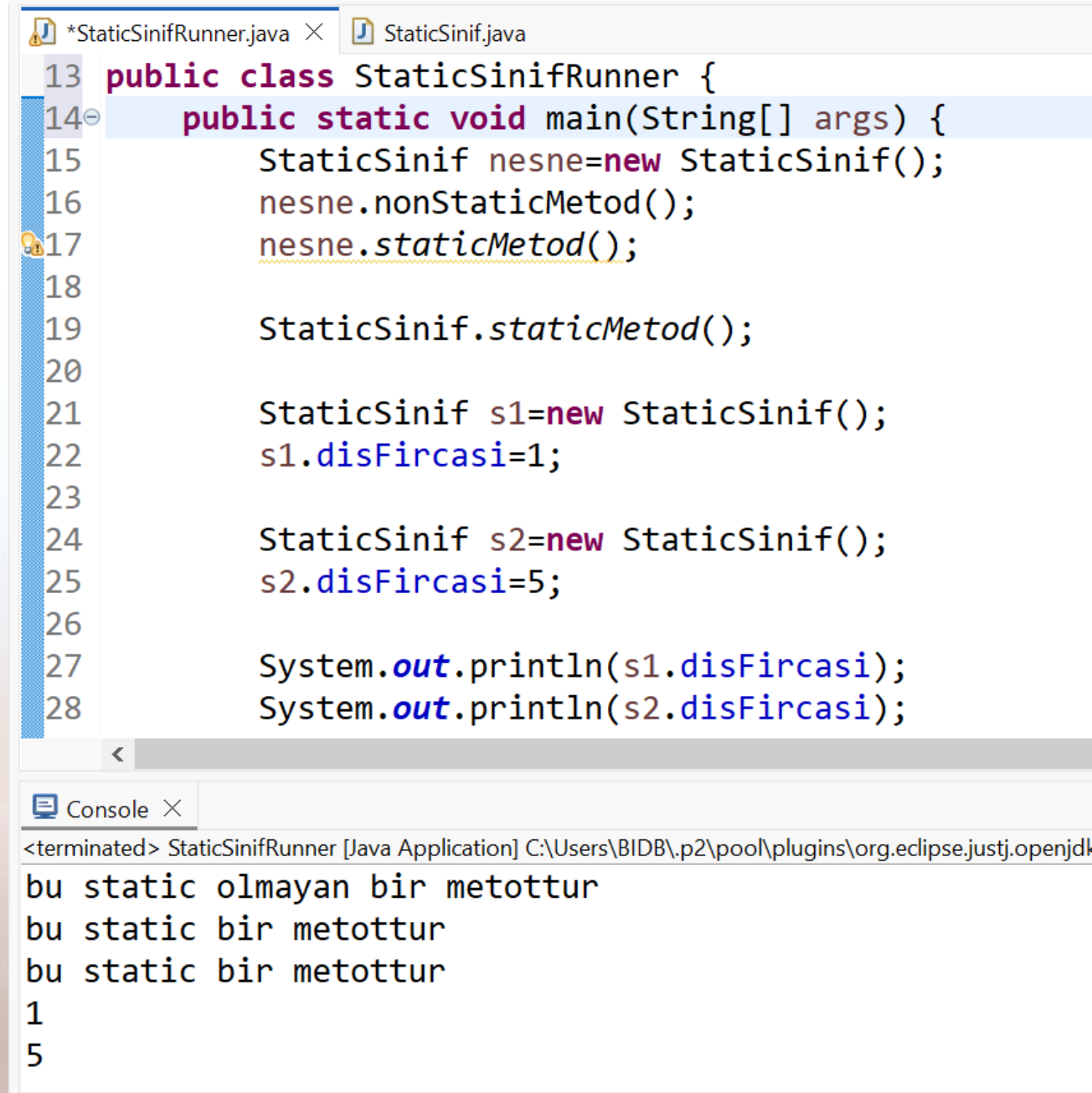
- $\blacklozenge$  static anahtar kelimesi, bir değişkenin veya metodun
- doğrudan sınıfa ait olduğunu belirtir.
- $\blacklozenge$  static olarak tanımlanan öğelere sınıfın bir örneği
- (nesnesi) oluşturulmadan erişilebilir.
- $\blacklozenge$  Yani static class member'lara (değişken veya metotlara)
- ulaşmak için obje oluşturmak gerekmez.
- $\blacklozenge$  Buna karşılık, non-static (örnek seviyesindeki) class
- member'lara erişmek için obje oluşturmak zorunludur.

```
*StaticSinifRunner.java × StaticSinif.java
1 package hafta9static;
2 public class StaticSinifRunner {
3     public static void main(String[] args) {
4         ////STATIC
5         /*
6          * static anahtar kelimesi, bir değişkenin veya metodun
7          * doğrudan sınıfa ait olduğunu belirtir.
8          * static olarak tanımlanan öğelere sınıfın bir örneği
9          * (nesnesi) oluşturulmadan erişilebilir.
10         * Yani static class member'lara (değişken veya metotlara)
11         * ulaşmak için obje oluşturmak gerekmez.
12         * Buna karşılık, non-static (örnek seviyesindeki) class
13         * member'lara erişmek için obje oluşturmak zorunludur.
14         */
15         StaticSinif nesne=new StaticSinif();
16         nesne.nonStaticMetod();
17         nesne.staticMetod();
18
19         StaticSinif.staticMetod();
20         StaticSinif.
21     }
22
23 }
```

$\bullet$  class : Class<hafta9static.StaticSinif>  
 $\bullet$  staticMetod() : void - StaticSinif

# STATIC

- 1) static variable veya static methodlar (class member) tüm object'ler için ortak elemandır
- 2) static class member'lar üzerinde yapılan değişiklikler tüm objectleri etkiler.



The screenshot shows the Eclipse IDE with two tabs: `*StaticSinifRunner.java` and `StaticSinif.java`. The `StaticSinifRunner.java` tab is active, displaying the following Java code:

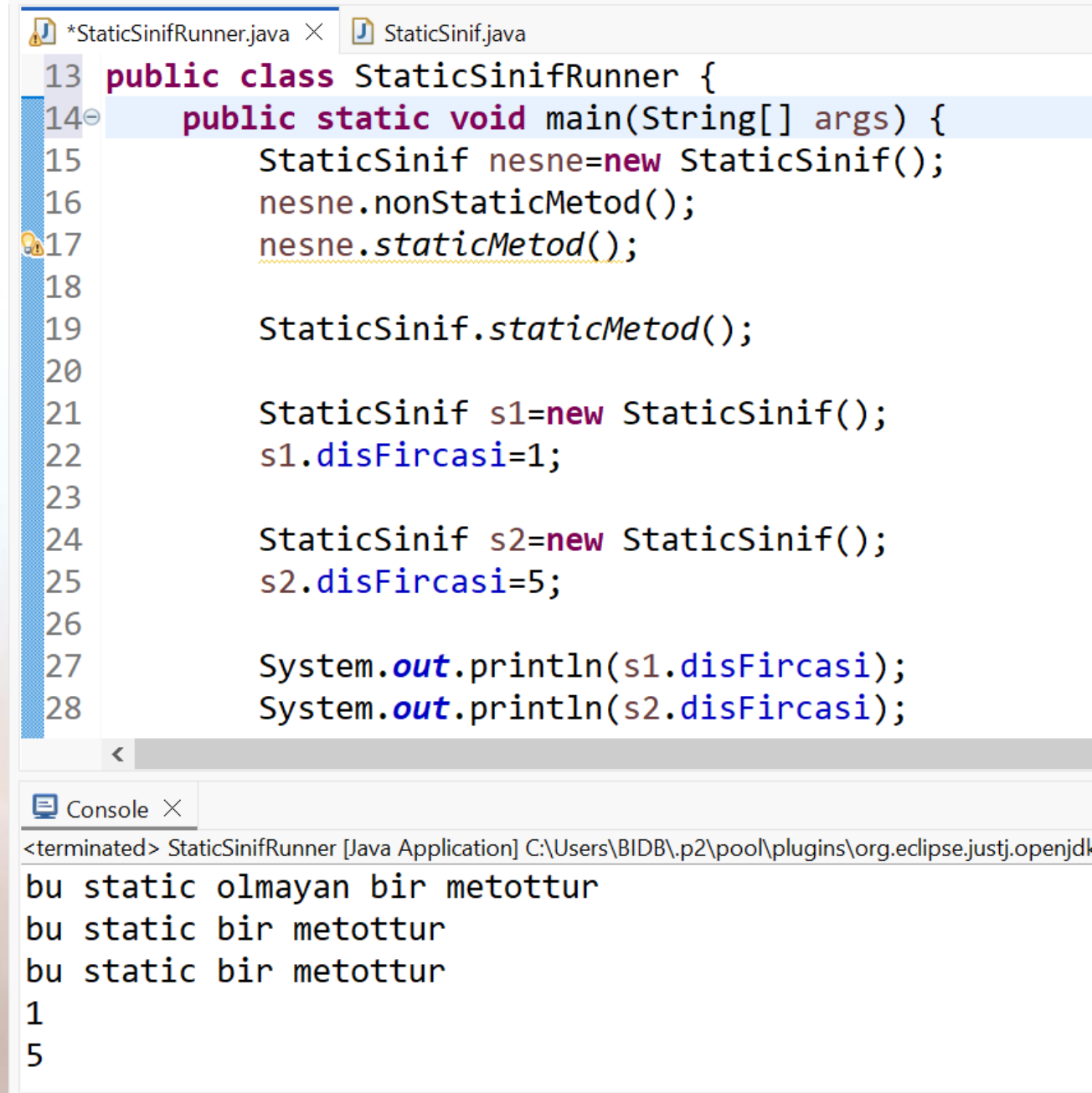
```
13 public class StaticSinifRunner {
14     public static void main(String[] args) {
15         StaticSinif nesne=new StaticSinif();
16         nesne.nonStaticMetod();
17         nesne.staticMetod();
18
19         StaticSinif.staticMetod();
20
21         StaticSinif s1=new StaticSinif();
22         s1.disFircasi=1;
23
24         StaticSinif s2=new StaticSinif();
25         s2.disFircasi=5;
26
27         System.out.println(s1.disFircasi);
28         System.out.println(s2.disFircasi);
```

Below the code editor, the `Console` tab is visible, showing the output of the program:

```
<terminated> StaticSinifRunner [Java Application] C:\Users\BIDB\p2\pool\plugins\org.eclipse.justj.openjdk
bu static olmayan bir metottur
bu static bir metottur
bu static bir metottur
1
5
```

# STATIC

- 1) Non static verilerin her birini nesnenin kendine özeldir.
- 2) S1 nesnesinin disFircasi ile S2 nesnesinin disFircasi birbirinden farklıdır.



The screenshot shows an Eclipse IDE with two tabs: `*StaticSinifRunner.java` and `StaticSinif.java`. The `StaticSinifRunner.java` tab is active, displaying the following code:

```
13 public class StaticSinifRunner {
14     public static void main(String[] args) {
15         StaticSinif nesne=new StaticSinif();
16         nesne.nonStaticMetod();
17         nesne.staticMetod();
18
19         StaticSinif.staticMetod();
20
21         StaticSinif s1=new StaticSinif();
22         s1.disFircasi=1;
23
24         StaticSinif s2=new StaticSinif();
25         s2.disFircasi=5;
26
27         System.out.println(s1.disFircasi);
28         System.out.println(s2.disFircasi);
```

Below the code editor, the `Console` tab is visible, showing the output of the program:

```
<terminated> StaticSinifRunner [Java Application] C:\Users\BIDB\p2\pool\plugins\org.eclipse.justj.openjdk
bu static olmayan bir metottur
bu static bir metottur
bu static bir metottur
1
5
```

# STATIC

- static veriler her nesne için ortaktır. Birinde yapılan değişiklik diğerinde de yapılmış sayılır. Bu örnekte static(ortak) olan buzDolabi değişkenine
- S4 nesnesi 1 demişti
- S3 nesnesi de son olarak 5 dedi
- Son değişikliği s3 nesnesi yaptığı için s1-s2-s3-s4 tüm nesnelerin buzdolabi nesnesi 5 olarak değişti.

```
34
35 StaticSinif s4=new StaticSinif();
36 s4.buzDoLabı=1;
37
38 StaticSinif s3=new StaticSinif();
39 s3.buzDoLabı=5;|
40
41 System.out.println(s4.buzDoLabı);
42 System.out.println(s3.buzDoLabı);
43 System.out.println(s2.buzDoLabı);
44 System.out.println(s1.buzDoLabı);
45
46
47
```

<

Console X

<terminated> StaticSinifRunner [Java Application] C:\Users\BIDB\.p2\pool\plugins\org.eclipse

5

5

5

5

# STATIC

1. static değişkenler veya metodlar (class member), tüm nesneler (object'ler) için ortaktır.
2. Static class member'larda yapılan bir değişiklik, tüm nesneleri etkiler.

## ✦ Özetle:

- **Static class member'lar** sınıfa (**class'a**) aittir.
- **Non-static class member'lar** ise nesnelere (**object'lere**) aittir.

## 💡 Terimler:

- static variable → **class variable**
- non-static variable → **instance variable**

# Soru

Aşağıdaki gereksinimlere uygun bir Person sınıfı yazınız ve belirtilen işlemleri yaparak main metodunda test ediniz:

1. Person sınıfı içinde:

- name isminde **public** bir String alan tanımlayın.
- age isminde **public** bir int alan tanımlayın.
- **public static** bir int değişkeni olan personCount tanımlayın. Bu değişken, oluşturulan Person nesnelerinin sayısını tutmalıdır.
- Sınıfa bir **constructor** ekleyin ve bu constructor her çağrıldığında personCount değişkenini bir artırarak nesne sayısını takip edin.
- personCount değişkeninin değerini döndüren bir **public static** getPersonCount metodu ekleyin.
- personCount'u ekrana yazdıran bir **public static** printPersonCount metodunu ekleyin.
- Başka bir Person nesnesinin yaşını değiştiren **public** bir changeAge metodunu ekleyin. Bu metod, bir Person nesnesi ve yeni bir yaş değeri almalıdır.

# Soru

main metodunda:

- Üç farklı Person nesnesi oluşturun: p1, p2, ve p3. p1 için ismi "Ali" ve yaşı 25 olarak başlatın. p2 için ismi "Veli" ve yaşı 30 olarak başlatın. p3 için ismi "Ayşe" ve yaşı 28 olarak başlatın.
- Person nesnelerinin oluşturulmasından sonra, printPersonCount metodunu kullanarak kaç Person nesnesi oluşturulduğunu ekrana yazdırın.
- p1 nesnesini kullanarak p2 nesnesinin yaşını 35 olarak değiştirin.
- p2 ve p3 nesnelerinin yaşlarını ekrana yazdırın.
- Değişikliklerden sonra printPersonCount metodunu tekrar kullanarak kaç nesne olduğunu ekrana yazdırın.

# OOP FAYDALARI

- 1) Object oluşturma bir Class içerisinde toplanır ve tüm projelerde kullanılabilirliğe olanak sağlar.(String gibi)
- 2) Class'ların 1 kez oluşturulması sayesinde uzun kodları tekrardan yazmak yerine kısa kodlamalar ile çalıştırılabilir.
- 3) Uzun kodların tekrar yazılmasının engellenmesi sayesinde geliştirme süreci kısalmıştır.
- 4) Object'ler birbirinden bağımsız olduğundan bilgi gizliliği konusunda avantaj sağlar.
- 5) Class'lar sayesinde tüm projelerde değişiklik yapmak yerine tek bir class' ta değişiklik yapıp tüm projelerde çalışması sağlanır. Bu zaman kaybını büyük ölçüde azaltır.

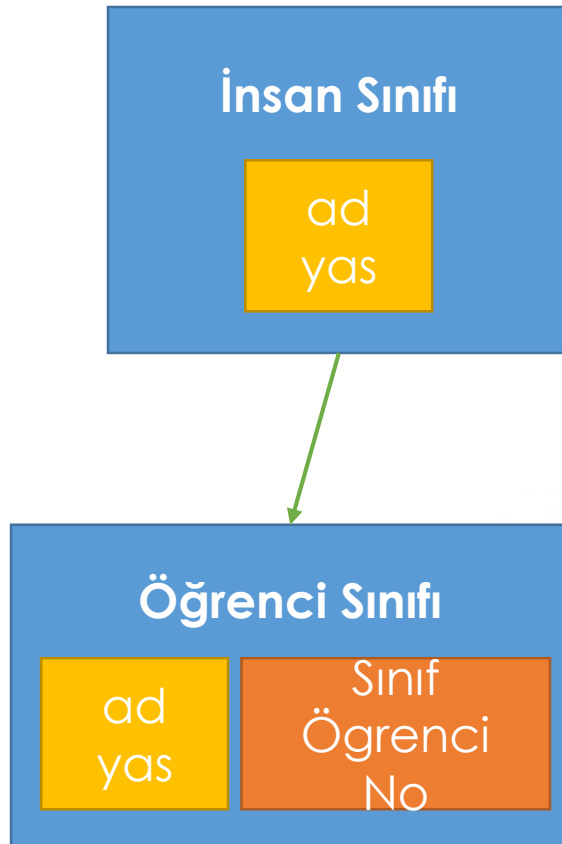
# OOP 4 TEMEL ÖZELLİĞİ

1. Inheritance - Miras Alma
2. Polymorphism - Çok Biçimlilik
3. Encapsulation - Kapsülleme
4. Abstraction - Soyutlama



# INHERITANCE-MİRAS ALMA

Özellikleri bir class'tan diğer class'lara aktaran bir kavramdır.



# INHERITANCE FAYDALARI

## 1. Kodun Yeniden Kullanılabilirliği (Reusability)

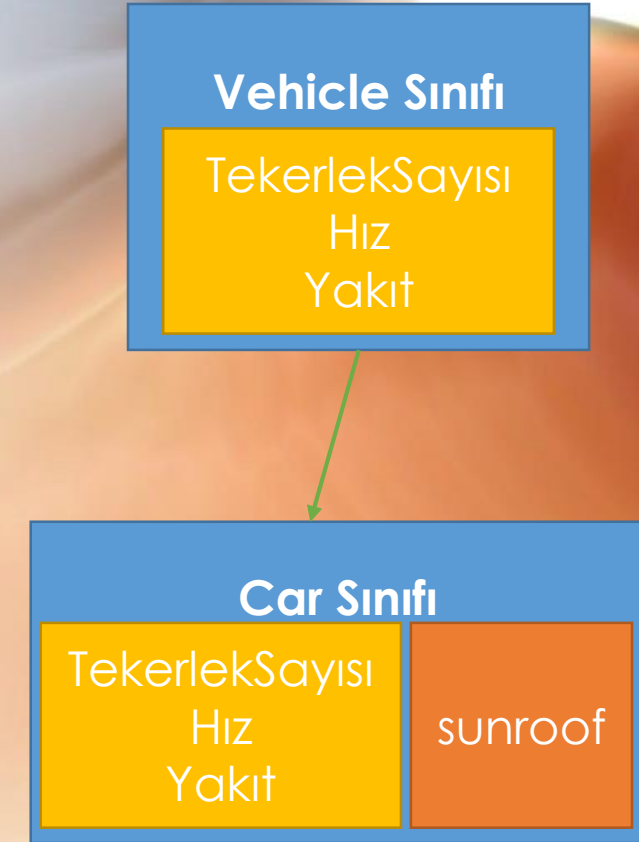
•Miras sayesinde, bir sınıf içinde tanımlanan özellikler ve yöntemler (metodlar), başka sınıflar tarafından tekrar kullanılabilir. Yeni bir sınıf yazarken sıfırdan başlamak yerine mevcut bir sınıfın özelliklerini ve davranışlarını miras alabilirsiniz.

•Örneğin, **Vehicle** sınıfından **Car** ve **Motorcycle** sınıfları türetilirse, **Vehicle** sınıfındaki temel özellikler (tekerlek sayısı, hız, yakıt türü gibi) tekrar tanımlanmadan bu alt sınıflar tarafından kullanılabilir.

## 2. Kod Bakımı ve Geliştirmeyi Kolaylaştırır (Maintainability)

•Kodda değişiklik yapmak gerektiğinde, temel sınıf üzerinde yapılan değişiklikler otomatik olarak alt sınıflara yansır. Bu, kodun bakımını kolaylaştırır ve yazılım geliştirme sürecinde tutarlılığı sağlar.

•Örneğin, **Person** sınıfında bir değişiklik yapıldığında, **Student** veya **Teacher** gibi bu sınıftan türeyen sınıflarda da otomatik olarak bu değişiklikten yararlanılır.



# INHERITANCE FAYDALARI

## 3. Genel Özelliklerin Merkezileştirilmesi (Centralization)

•Ortak özelliklerin ve metodların bir üst sınıfta toplanması, kodun merkezi bir yerde yönetilmesini sağlar. Bu da sınıflar arasında kod tekrarını azaltır.

•Örneğin, bir şirketin çalışanlarını temsil eden bir yazılımda **Employee** adında bir sınıf oluşturup bu sınıftan **Manager**, **Developer**, **Tester** gibi sınıflar türetmek, tüm çalışanlar için ortak olan **salary** veya **workingHours** gibi özelliklerin merkezi bir yerde tanımlanmasını sağlar.

## 4. Daha Anlaşılır Kod ve Organizasyon (Clearer Code Structure)

•Miras, sınıflar arasında doğal bir hiyerarşi oluşturur ve kodun yapısını daha anlaşılır hale getirir. Bu, karmaşık sistemlerin daha düzenli ve anlaması kolay bir şekilde modellenmesine yardımcı olur.

•Örneğin, hayvanlar dünyasını modellemek istiyorsanız, **Animal** adlı bir üst sınıf oluşturup **Mammal**, **Bird**, **Fish** gibi alt sınıflar tanımlamak daha anlaşılır bir yapı sağlar.

## 5. Polimorfizmi Destekler (Supports Polymorphism)

•Miras ilişkisi, polimorfizmi desteklediği için farklı sınıfların ortak bir arayüz üzerinden ele alınmasına imkan tanır. Bir üst sınıftan türeyen farklı sınıflar, aynı metodları farklı şekillerde uygulayabilir.

•Örneğin, **Animal** sınıfında bir **makeSound()** metodu tanımlayıp, **Dog**, **Cat** gibi alt sınıflarda bu metodu kendilerine has bir şekilde yeniden tanımlayabilirsiniz. Bu sayede bir **Animal** nesnesi olarak ele alınan **Dog** veya **Cat**, kendi seslerini çıkarabilir.



# INHERITANCE

Animal adında bir sınıf oluşturalım.

Animal Sınıfı

eat()  
drink()

```
*Animal.java ×
1 package hafta4;
2
3 public class Animal {
4     public void eat(){
5         System.out.println("hayvanlar yer");
6     }
7     public void drink(){
8         System.out.println("hayvanlar içer");
9     }
10    //static sınıflar miras olarak aktarılamaz.
11 }
```

# INHERITANCE

AnimalRunner adında bir sınıf oluşturalım.

```
Animal.java  Mammal.java  *AnimalRunner.java X
1 package hafta4;
2
3 public class AnimalRunner {
4
5     public static void main(String[] args) {
6         Animal a1=new Animal();
7         a1.|
8
9     }
10
11 }
12
```

- drink() : void - Animal
- eat() : void - Animal
- equals(Object obj) : boolean - Object
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object

Animal Sınıfı

eat()  
drink()

# INHERITANCE

Mammal adında bir sınıf oluşturalım.

Animal Sınıfı

eat()  
drink()

\*Animal.java

Mammal.java ×

```
1 package hafta4;
2
3 public class Mammal {
4     public void feedWithMilk(){
5         System.out.println("Mammals feed their babies with milk");
6     }
7 }
8
```

Mammal Sınıfı

feedWithMilk()

# INHERITANCE

M

```
Animal.java  Mammal.java  *AnimalRunner.java x
1 package hafta4;
2
3 public class AnimalRunner {
4
5     public static void main(String[] args) {
6         Animal a1=new Animal();
7         a1.drink();
8         Mammal m1=new Mammal();
9         m1.
10
11     }
12
13 }
14
```

- equals(Object obj) : boolean - Object
- feedWithMilk() : void - Mammal
- getClass() : Class<?> - Object
- hashCode() : int - Object
- notify() : void - Object
- notifyAll() : void - Object
- toString() : String - Object

## Animal Sınıfı

eat()  
drink()

## Mammal Sınıfı

feedWithMilk()

# INHERITANCE

Extends ile parent ini belirliyoruz.

```
*Animal.java  *Mammal.java x
1 package hafta4;
2
3 public class Mammal extends Animal{
4     public void feedWithMilk(){
5         System.out.println("Mammals feed their babies with milk");
6     }
7 }
8
```

```
*Animal.java  *Mammal.java x
1 package hafta4;
2
3 public class Mammal extends Animal{
4     public void feedWithMilk(){
5         System.out.println("Mammals feed their babies with milk");
6     }
7 }
```

Animal Sınıfı

eat()  
drink()

Mammal Sınıfı

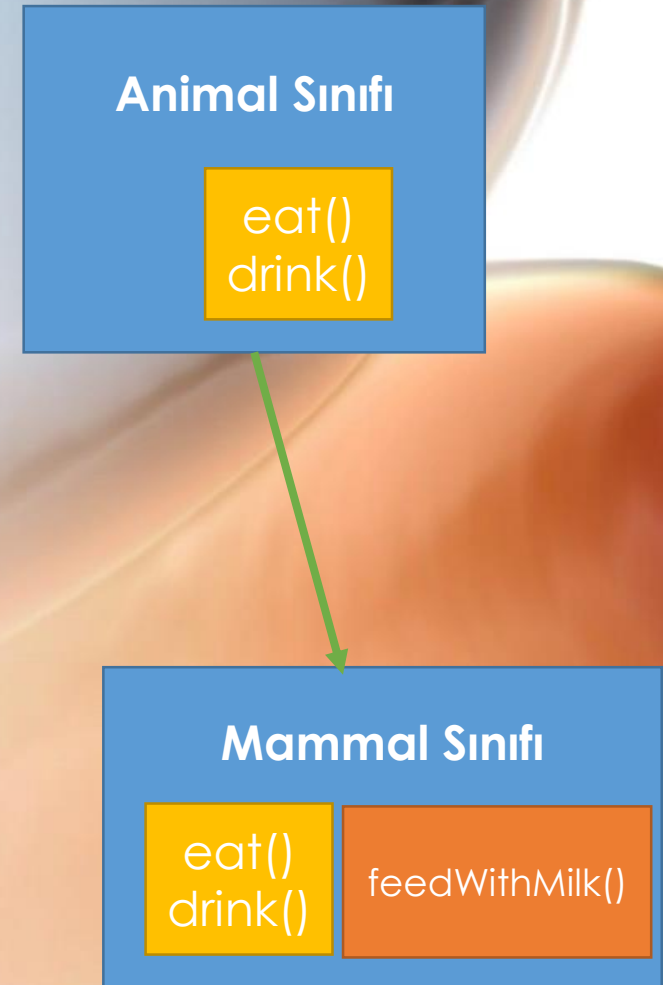
eat()  
drink()

feedWithMilk()

# INHERITANCE

```
Animal.java  Mammal.java  *AnimalRunner.java x
1 package hafta4;
2
3 public class AnimalRunner {
4
5     public static void main(String[] args) {
6         Animal a1=new Animal();
7         a1.drink();
8         Mammal m1=new Mammal();
9         m1.
10
11     }
12
13 }
```

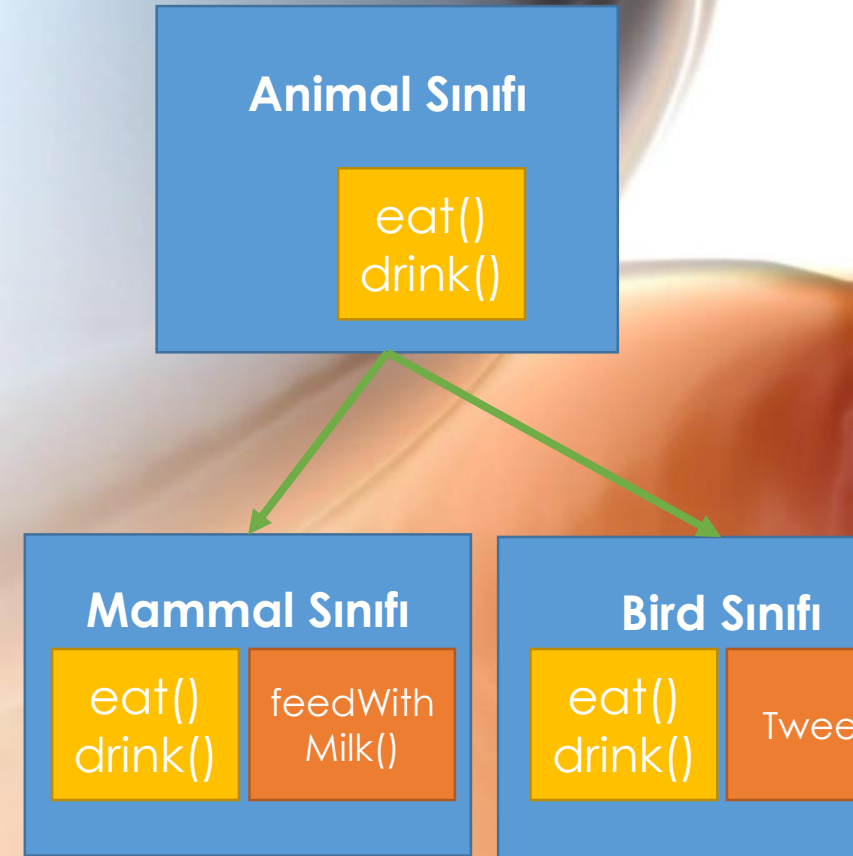
- drink() : void - Animal
- eat() : void - Animal
- equals(Object obj) : boolean - Object
- feedWithMilk() : void - Mammal
- getClass() : Class<?> - Object
- hashCode() : int - Object



# INHERITANCE

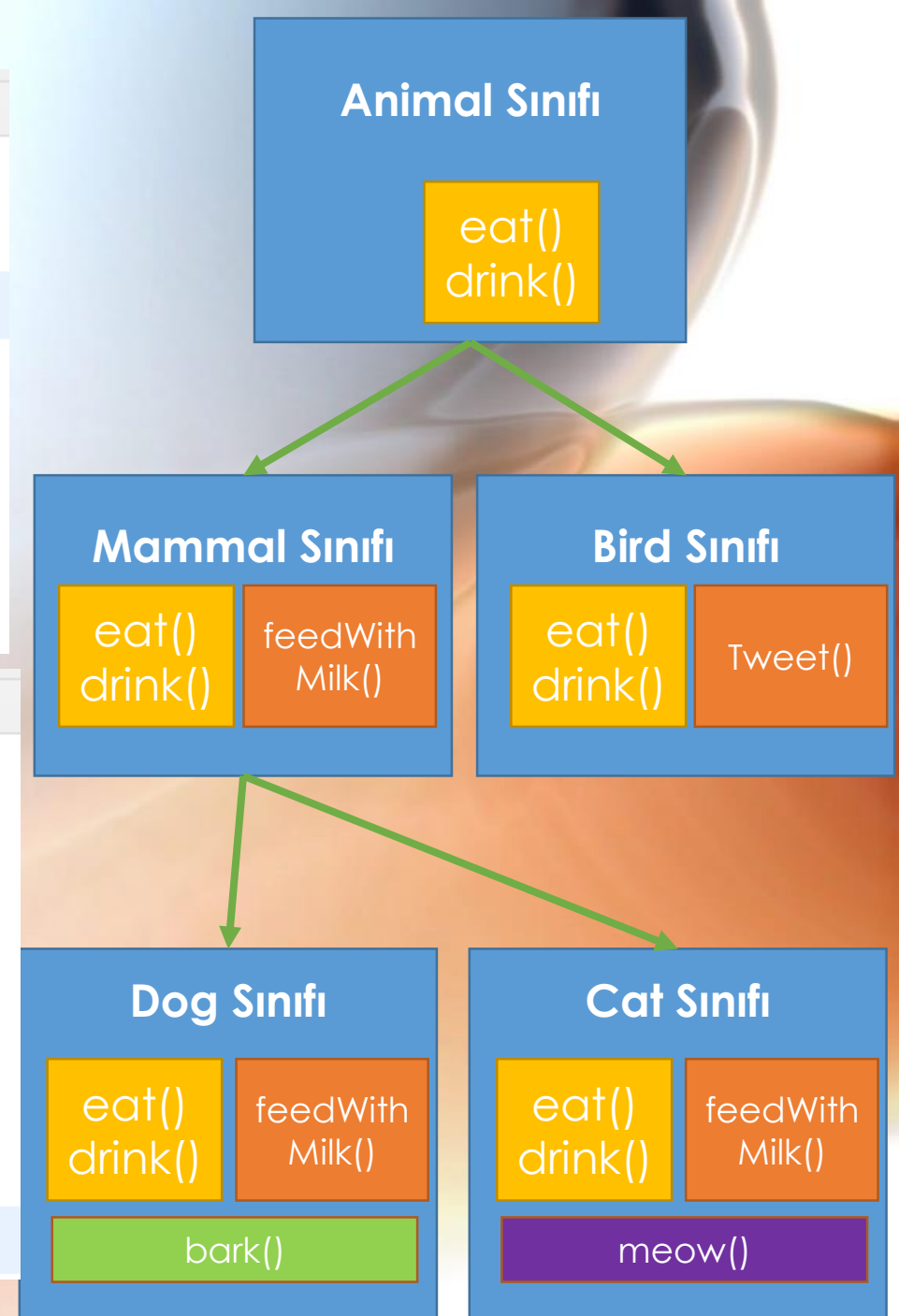
Bir sınıf birden fazla sınıfa parent olabilir.

```
Animal.java  Mammal.java  AnimalRunner.java  Bird.java x
1 package hafta4;
2
3 public class Bird extends Animal{
4     public void tweet(){
5         System.out.println("birds tweet");
6     }
7 }
8
```



```
Animal.java Mammal.java *AnimalRunner.java Bird.java Cat.java x *Dog.java
1 package hafta4;
2
3 public class Cat extends Mammal {
4     public void meow(){
5         System.out.println("cats meow...");
6     }
7 }
8
```

```
Animal.java Mammal.java *AnimalRunner.java Next Annotation (Ctrl+.) Cat.java Dog.java x
1 package hafta4;
2
3 public class Dog extends Mammal {
4     public void bark(){
5         System.out.println("dogs bark..");
6     }
7 }
8
```

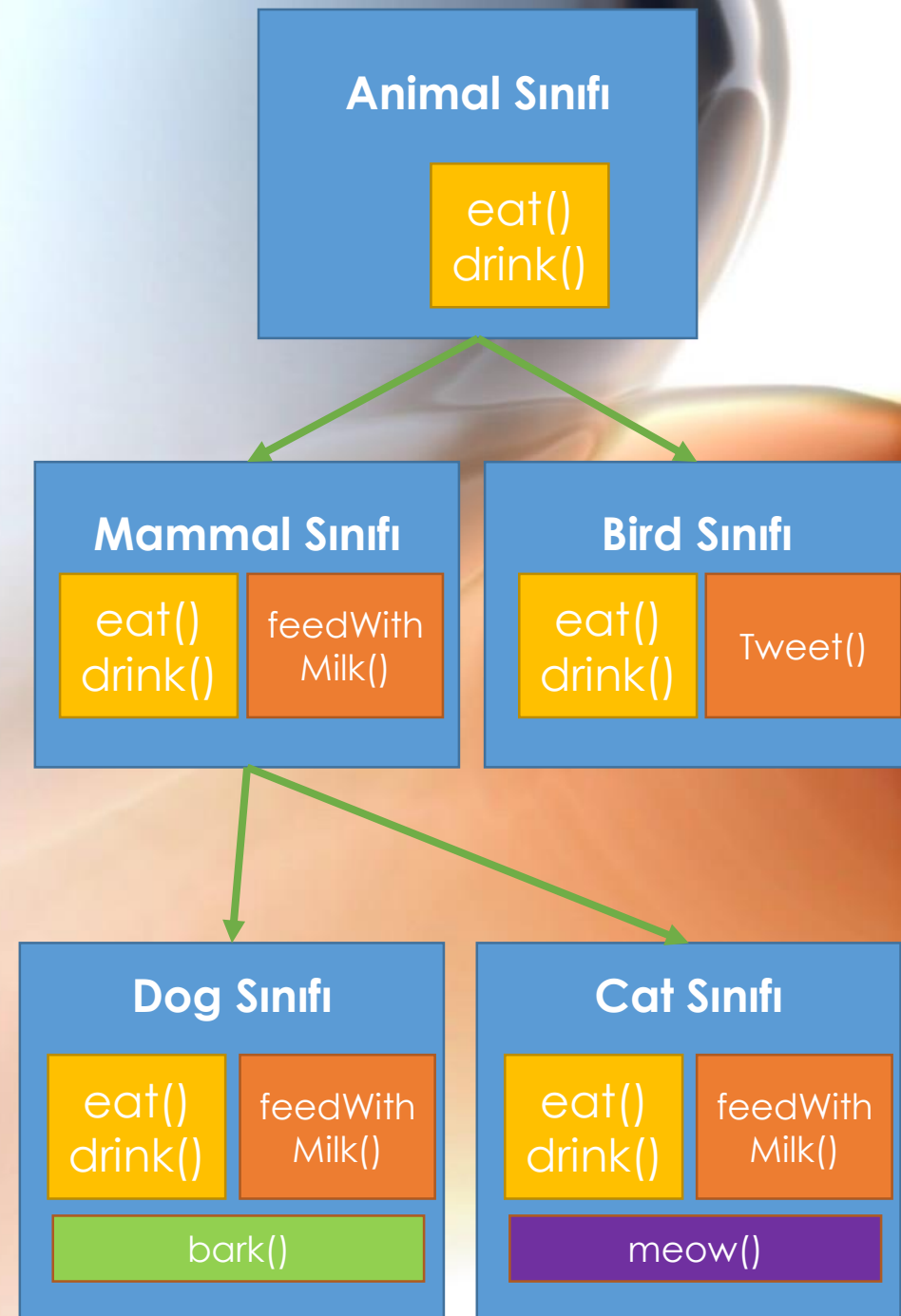


```
Animal.java  Mammal.java  *AnimalRunner.java  Bird.java

1 package hafta4;
2
3 public class AnimalRunner {
4
5     public static void main(St
6     Animal a1=new Animal();
7     a1.drink();
8     Mammal m1=new Mammal();
9     m1.eat();
10    Cat c1=new Cat();
11    c1.
12    }
13
14 }
15
```

- drink() : void - Animal
- eat() : void - Animal
- equals(Object obj) : boolean - Object
- feedWithMilk() : void - Mammal
- getClass() : Class<?> - Object
- hashCode() : int - Object
- meow() : void - Cat
- notify() : void - Object

rt olabilir.

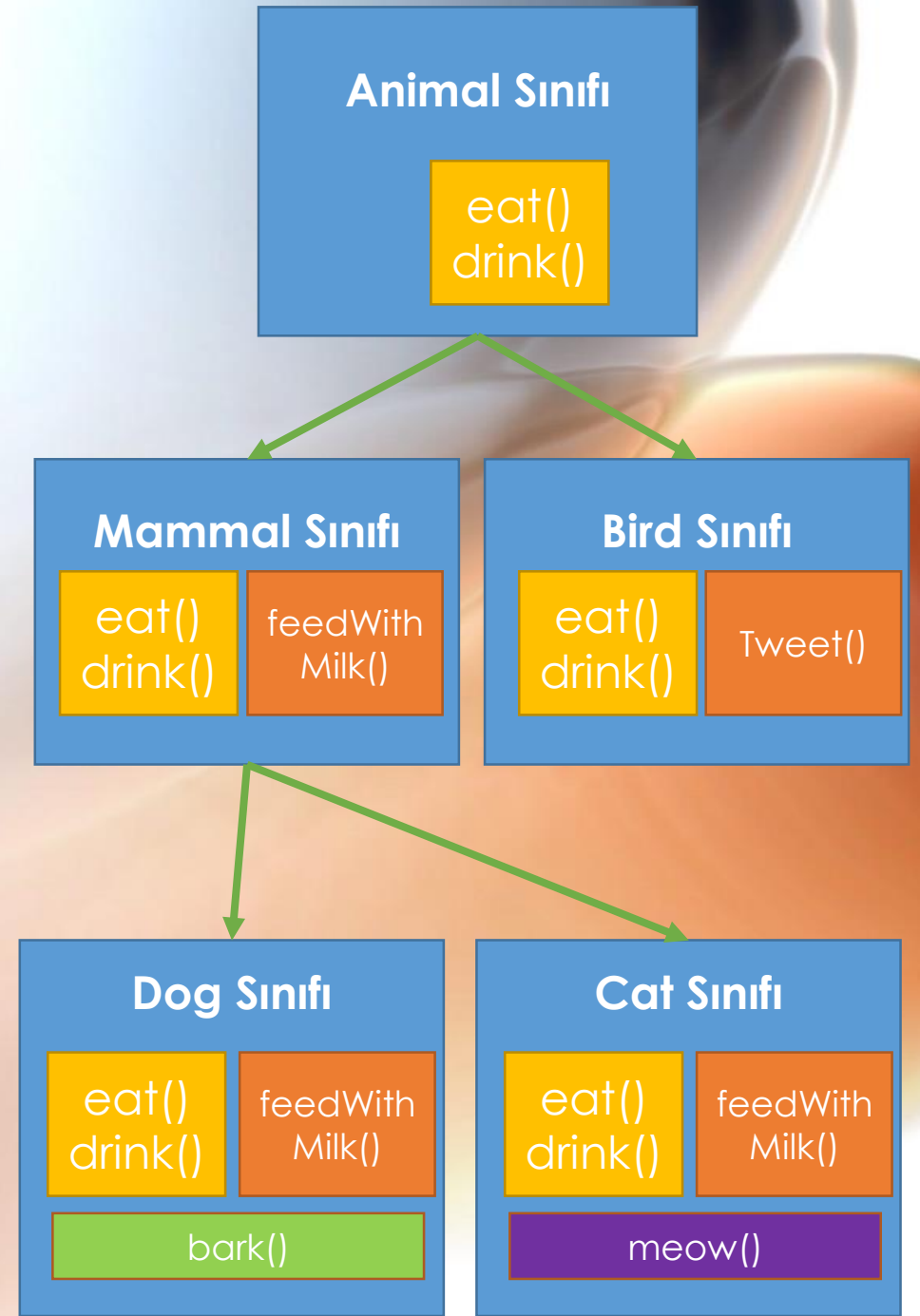


# INHERITANCE

Java multiple inheritance desteklemez.

Java **SINGLE INHERITANCE** i destekler .

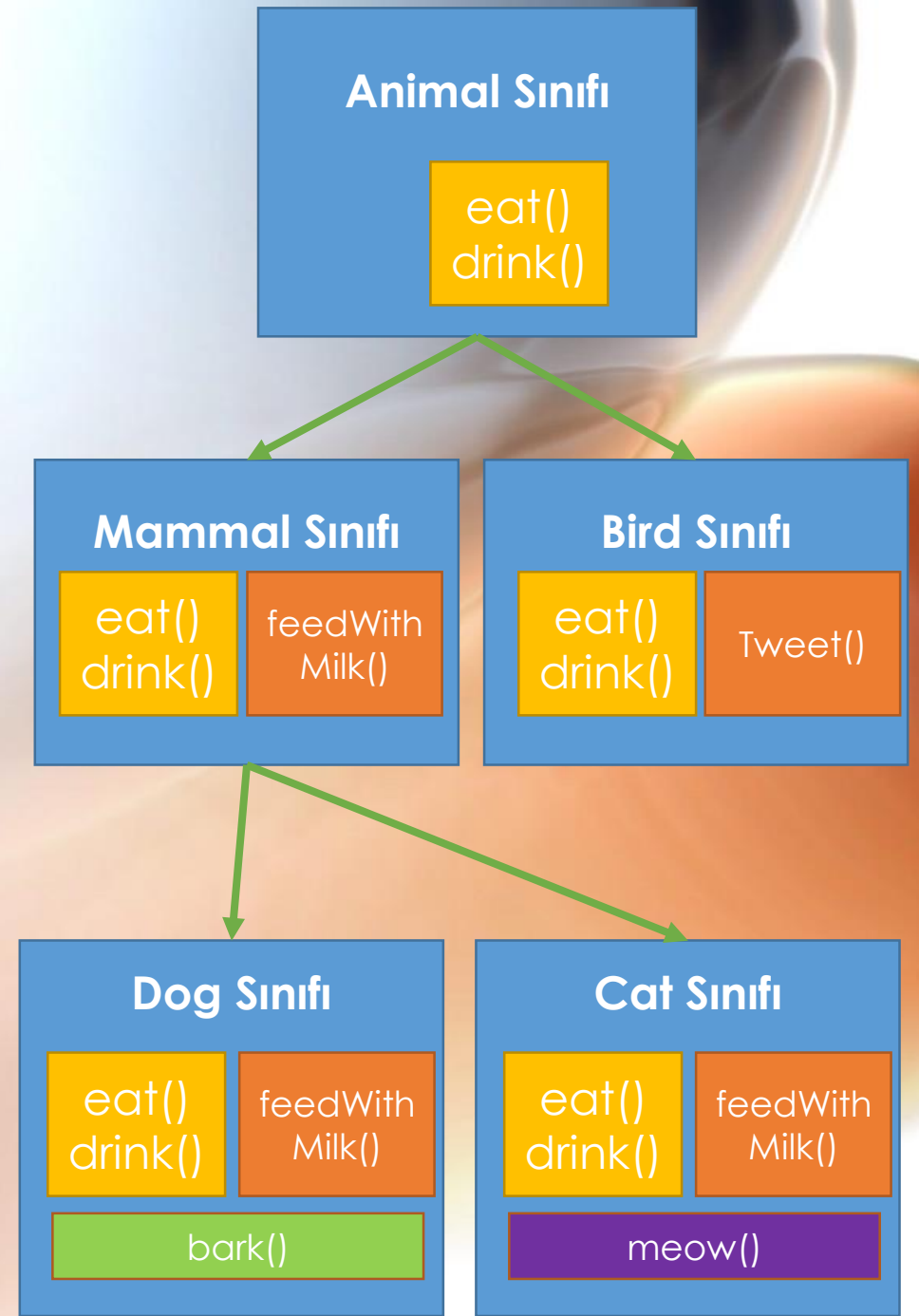
Bir Child in Bir Parent ı olur.



# INHERITANCE

Java, Bir parent ve birden fazla child in olduğu yapıyı destekler.

## HIERARCHICAL INHERITANCE

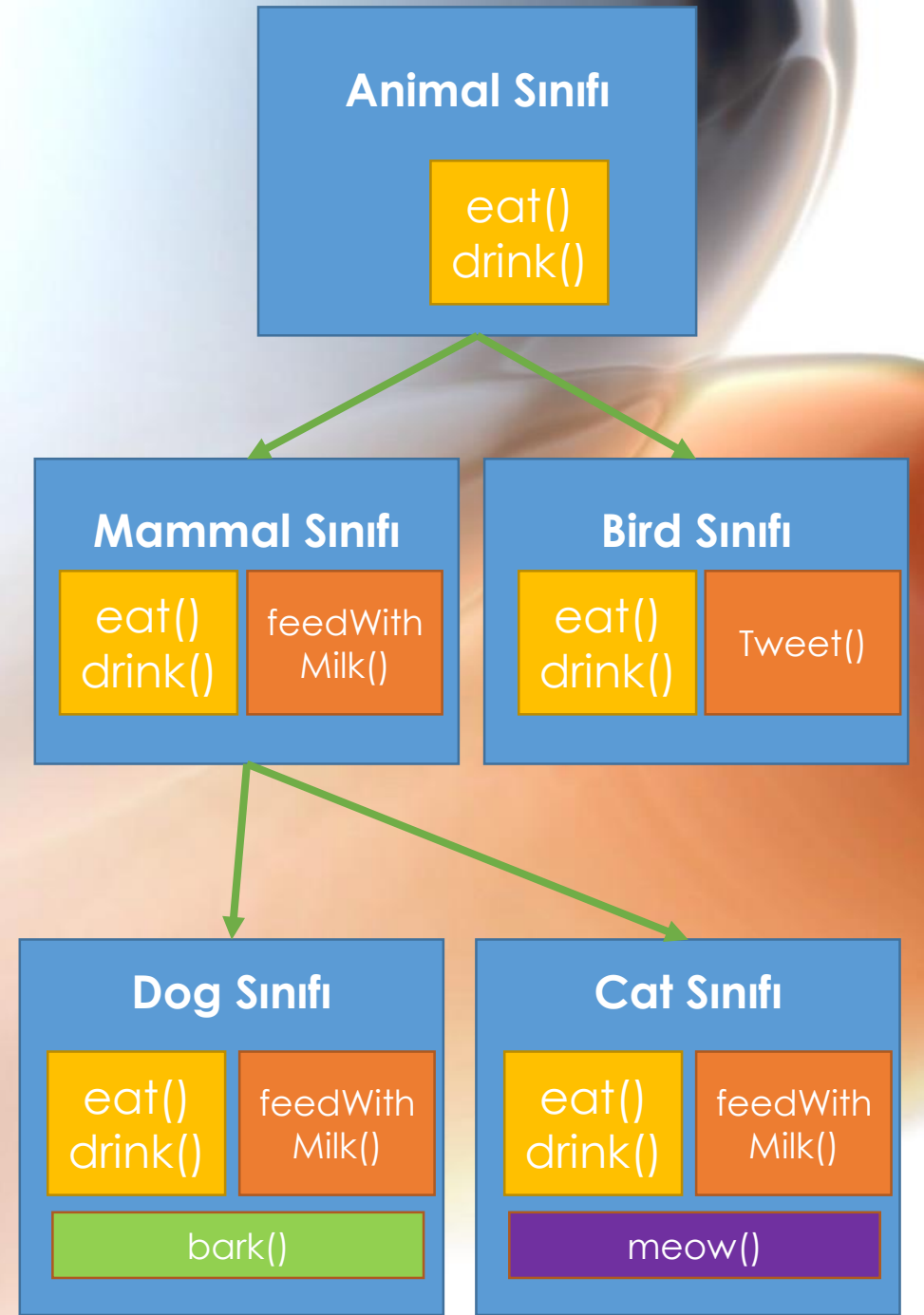


# INHERITANCE

Bir child a birden fazla sınıfın özelliklerini çekmek istersem, single inheritance dan dolayı yapamıyorum 😞

Buna çözüm olarak apartman tarzi alt alta yapılan inheritance bize çözüm olacak.

## MULTI LEVEL INHERITANCE

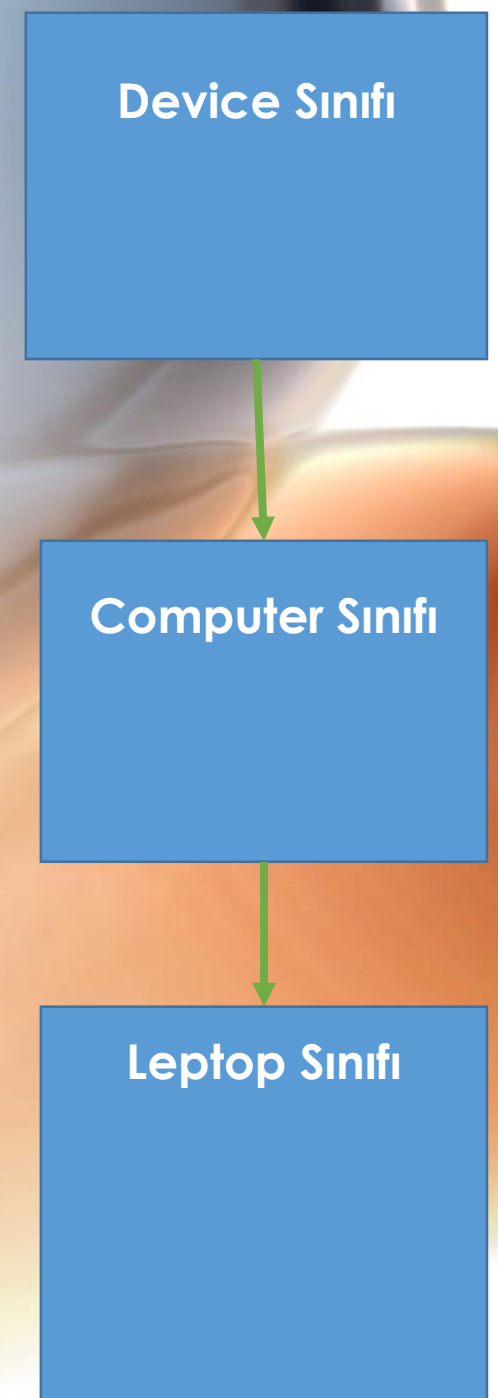


# CONSTRUCTOR HIYERARŞİSİ

Child class'tan bir object oluşturduğunuzda constructor'lar en üstteki parent class'tan başlatılarak alta doğru çalıştırılır.

//çağrı aşağıdan yukarı doğru olurken

//constructorlar yukarıdan aşağı iner



# CONSTRUCTOR HIYERARŞİSİ

```
*Device.java × Computer.java Laptop.java DeviceRunner.java
NesneYonelimliYazilim ▶ src ▶ haftaaa10ConstructorHiyerarsisi ▶ Device ▶
9
10
11 public class Device {
12     public Device() {
13         System.out.println("Elektronik cihaz başlatılıyor...");
14     }
15 }
16
17
18
```

Device Sınıfı

Computer Sınıfı

Leptop Sınıfı



# CONSTRUCTOR HIYERARŞİSİ

```
*Device.java  Computer.java  Laptop.java  DeviceRunner.java
NesneYonelimliYazilim  src  haftaaa10ConstructorHiyerarsisi  Computer
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Computer extends Device{
4     public Computer() {
5         System.out.println("Bilgisayar sistemi hazır.");
6     }
7 }
8
```

Device Sınıfı

Computer Sınıfı

Leptop Sınıfı

# CONSTRUCTOR HIYERARŞİSİ

```
*Device.java  Computer.java  *Laptop.java  DeviceRunner.java
NesneYonelimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Laptop > Laptop()
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Laptop extends Computer {
4     public Laptop() {
5         System.out.println("Dizüstü bilgisayar: "
6             + "16 GB RAM, i7 işlemci.");
7     }
8 }
9
```

Device Sınıfı

Computer Sınıfı

Leptop Sınıfı

# CONSTRUCTOR HIYERARŞİSİ

Child class'tan bir object oluşturduğunuzda constructor'lar en üstteki parent class'tan başlatılarak alta doğru çalıştırılır.

//çağrı aşağıdan yukarı doğru olurken

//constructorlar yukarıdan aşağı iner

//çağrı aşağıdan yukarı doğru olurken

//constructorlar yukarıdan aşağı iner

Device Sınıfı

```
9
10
11 public class Device {
12     public Device() {
13         System.out.println("Elektronik cihaz başlatılıyor...");
14     }
15 }
16
17
18
```

Computer Sınıfı

```
1 package hafta10ConstructorHiyerarsisi;
2
3 public class Computer extends Device{
4     public Computer() {
5         System.out.println("Bilgisayar sistemi hazır.");
6     }
7 }
8
```

Laptop Sınıfı

```
1 package hafta10ConstructorHiyerarsisi;
2
3 public class Laptop extends Computer {
4     public Laptop() {
5         System.out.println("Dizüstü bilgisayar: "
6             + "16 GB RAM, i7 işlemci.");
7     }
8 }
9
```

# CONSTRUCTOR HIYERARŞİSİ

```
*Device.java  Computer.java  *Laptop.java  *DeviceRunner.java ×
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > DeviceRunner > main(String[]) : void
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class DeviceRunner {
4
5     public static void main(String[] args) {
6
7         Laptop h1= new Laptop();
8     }
9
10 }
11
```

//çağrı aşağı  
//constructor

```
Console ×
<terminated> DeviceRunner [Java Application] C:\Users\BIDB\p2\pool\plugins\org.eclipse.justj.openjdk.hotsp
Elektronik cihaz başlatılıyor...
Bilgisayar sistemi hazır.
Dizüstü bilgisayar: 16 GB RAM, i7 işlemci.
```

Device Sınıfı

```
*Device.java  Computer.java  Laptop.java  DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Device
9
10
11 public class Device {
12     public Device() {
13         System.out.println("Elektronik cihaz başlatılıyor...");
14     }
15 }
16
17
18
```

Computer Sınıfı

```
*Device.java  Computer.java  Laptop.java  DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Computer
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Computer extends Device{
4     public Computer() {
5         System.out.println("Bilgisayar sistemi hazır.");
6     }
7 }
8
```

Laptop Sınıfı

```
*Device.java  Computer.java  *Laptop.java  DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Laptop > Laptop()
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Laptop extends Computer {
4     public Laptop() {
5         System.out.println("Dizüstü bilgisayar: "
6             + "16 GB RAM, i7 işlemci.");
7     }
8 }
9
```

# INSTANCE BLOK

Bir Class'in her Object'i (nesnesi) oluşturulduğunda çalışan bir kod bloğudur.

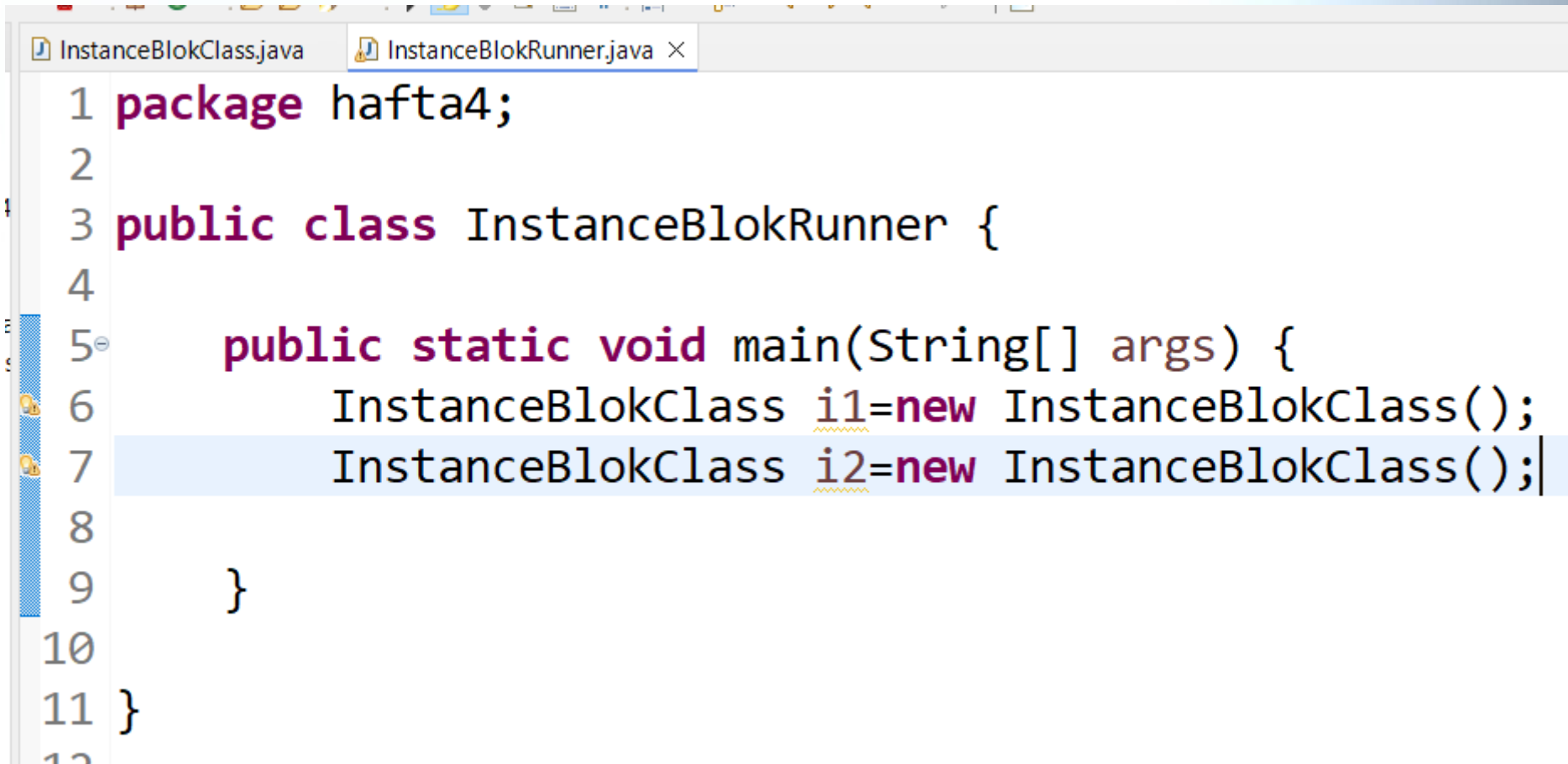
Instance bloklarının temel özellikleri şunlardır:

- 1- Her Örnek İçin Çalışır: Instance blok, bir class'in her örneği oluşturulduğunda çalışır. Yani, her yeni object için bu blok tekrar çalışır.
- 2- Kod Tekrarını Önler: Instance blokları kullanarak, her Object'in başlatma işlemlerini veya belirli kod parçalarını tekrar tekrar yazmaktan kaçınırsınız.

# INSTANCE BLOK

```
*InstanceBlokClass.java × InstanceBlokRunner.java
1 package hafta4;  ileriNesneTabanlıProgramlama_Guz_24_25/src/hafta4/InstanceBlokRunner.java
2
3 public class InstanceBlokClass {
4     {
5         System.out.println("instance blok çalıştı");
6     }
7
8     public InstanceBlokClass() {
9         //constructor
10    }
11
12 }
```

# INSTANCE BLOK



The screenshot shows a Java IDE with two tabs: 'InstanceBlokClass.java' and 'InstanceBlokRunner.java'. The 'InstanceBlokRunner.java' tab is active, displaying the following code:

```
1 package hafta4;
2
3 public class InstanceBlokRunner {
4
5     public static void main(String[] args) {
6         InstanceBlokClass i1=new InstanceBlokClass();
7         InstanceBlokClass i2=new InstanceBlokClass();
8
9     }
10
11 }
```

The code defines a package 'hafta4' and a public class 'InstanceBlokRunner'. Inside the 'main' method, two instances of 'InstanceBlokClass' are created and assigned to variables 'i1' and 'i2'. The line 'InstanceBlokClass i2=new InstanceBlokClass();' is highlighted in blue.

# INSTANCE BLOK

Gerçek hayattan bir örnek: Bir otomobil Class'i düşünün. Her otomobilin örneği (nesnesi-object'i) oluşturulduğunda, otomobilin rengini ve yakıt türünü belirlemek için instance bloklarını kullanabilirsiniz.

Örneğin: instance blok, her otomobil Object'i oluşturulduğunda çalışır ve otomobilin rengini siyah ve yakıt türünü benzin olarak başlatır. Bu, her yeni otomobilin aynı başlangıç değerleriyle başlamasını sağlar. Diğer özellikler sonra ayarlanır ki tekrardan kurtuluruz.

# INSTANCE BLOK

Her constructorda tekrar eden bir kodunuz var.  
Her nesne oluşturulduğunda çalıştırılsın istiyorsunuz.

```
*InstanceBlokClass.java  InstanceBlokRunner.java  *InstanceBlokClass1.java ×
1 package hafta4;
2
3 public class InstanceBlokClass1 {
4     int year;
5     public InstanceBlokClass1() {
6         System.out.println("selam");
7     }
8     public InstanceBlokClass1(int year) {
9         System.out.println("selam");
10        this.year = year;
11    }
12 }
13
```

# INSTANCE BLOK

Her constructorda tekrar eden bir kodunuz var.  
Her nesne oluşturulduğunda çalıştırılsın istiyorsunuz.

```
InstanceBlokClass.java  InstanceBlokRunner.java  InstanceBlokClass1.java
1 package hafta4;
2
3 public class InstanceBlokRunner {
4
5     public static void main(String[] args) {
6         InstanceBlokClass1 i1=new InstanceBlokClass1();
7         InstanceBlokClass1 i2=new InstanceBlokClass1(4);
8     }
9
10 }
11 }
12
```

Problems Javadoc Declaration Console X

<terminated> InstanceBlokRunner [Java Application] C:\User

selam  
selam

```
*InstanceBlokClass.java  InstanceBlokRunner.java  *InstanceBlokClass1.java X
1 package hafta4;
2
3 public class InstanceBlokClass1 {
4
5     {
6         System.out.println("selam");
7     }
8     int year;
9     public InstanceBlokClass1() {
10         //constructor
11     }
12     public InstanceBlokClass1(int year) {
13         this.year = year;
14     }
15 }
```

# INSTANCE BLOK

Örnek 3: Bir araba fabrikasında her araba siyah renkli ve yakıt tipi benzin olsun.

Hız, sunroof

//(Bunlar ortak özellikler. Diğerleri sonra ayarlanсын)

//1. constructor hiç parametre almasın. Arabaya ait bilgileri ekrana yazdırsın

//2. constructor 2 parametre alsın. Bilgileri ekrana yazdırsın.

# STATIC BLOK

- "static" blok, bir class'in yüklenmesi sırasında otomatik olarak çalışır.
- Class'in başlatılmasını veya başlangıç ayarlarını yapmayı sağlayan bir özelliktir.

Statik blok, yalnızca bir kez çalıştırılır.

Bu bloklar örneğin, veritabanı bağlantıları için kullanılır.

Bu, uygulamanın herhangi bir bölümünde veritabanına erişim gerektiğinde bağlantının hızlı bir şekilde hazır olmasını sağlar.

# STATICK BLOK

//static bloklar  
içinde yerel  
değişken  
tanımlanmaz,  
ancak değer  
atanabilir.

//yerel  
değişkenler bu  
blok içinde  
tanımlanamaz

```
StaticBlokClass.java x
1 package hafta4;
2
3 public class StaticBlokClass {
4     static double pi;
5     static{
6         pi=3.147;
7         System.out.println("static blok 1");
8     }
9     public static void main(String[] args) {
10        System.out.println(pi);
11    }
12
13 }
```

# STATIC BLOK dezavantajları

//programın çalışma zamanı uzayabilir, ancak sonradan hızlı yol alınabilir.

//statik bloklarda yapılan hata programın içerisinde beklenmedik sonuçlara yol açabilir.

# STATIC BLOK

Örnek: Şubat ayında fiyatlarda indirim olacak. Class çağırıldığında otomatik price'i alacak ve class'in her yerinde bu değeri kullanacak.

# STATIC BLOK

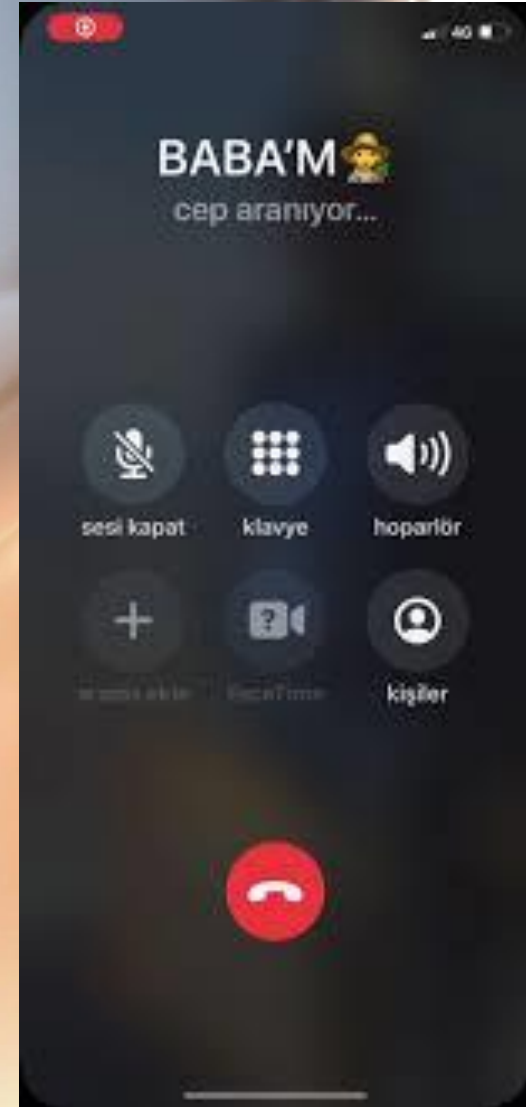
Örnek: Şubat ayında fiyatlarda indirim olacak. Class çağırıldığında otomatik price'i alacak ve class'in her yerinde bu değeri kullanacak.

```
StaticBlokClass.java  *StaticBlokOrn.java ×
1 package hafta4;
2 import java.time.LocalDate;
3 public class StaticBlokOrn {
4     static int price;
5     static{
6         System.out.println("static blok örnek");
7         LocalDate currentDate=LocalDate.now();
8         if(currentDate.getMonthValue()==2)
9         {
10             price=1000;
11         }
12         else{
13             price=2000;
14         }
15     }
16     public static void main(String[] args) {
17         System.out.println(price);
18     }
19 }
```

# Super(); Çağrısı

## **SUPER CALL**

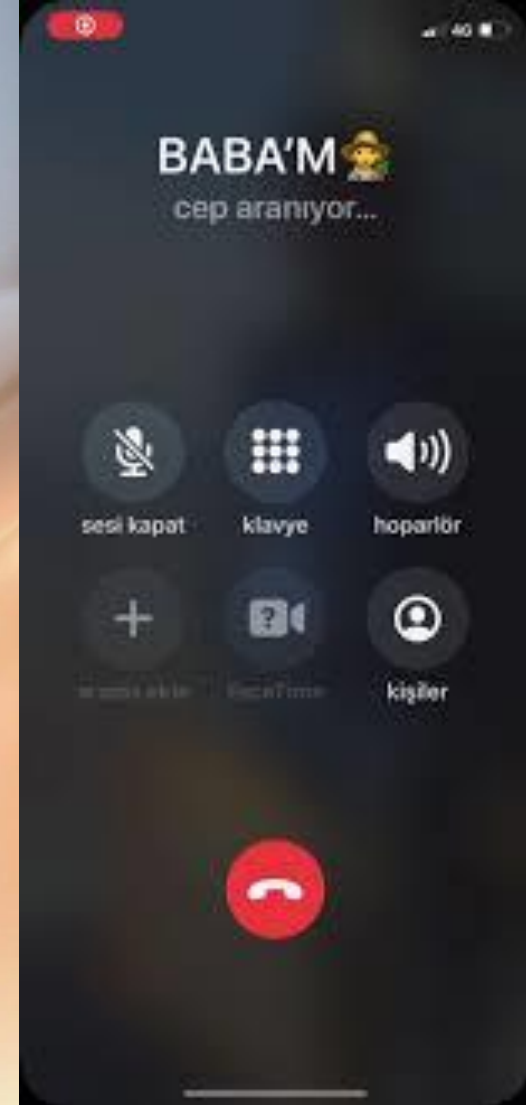
bir alt sınıfın (subclass-child),  
üst sınıfının (superclass-parent)  
yapıcı (constructor) metodunu veya diğer  
yöntemlerini çağırmasına olanak tanır.



# Super(); Çağrısı Kullanım Amacı

Super anahtar kelimesi Java'da iki ana amaçla kullanılır:

- Üst sınıfın yapıcı metodunu çağırmak.
- Üst sınıfın metotlarına veya değişkenlerine erişmek.



# CONSTRUCTOR HIYERARŞİSİ

Device Sınıfı

```
*Device.java  Computer.java  *Laptop.java  *DeviceRunner.java X
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > DeviceRunner > main(String[]) : void
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class DeviceRunner {
4
5     public static void main(String[] args) {
6
7         Laptop h1= new Laptop();
8     }
9
10 }
11
```

```
Console X
<terminated> DeviceRunner [Java Application] C:\Users\BIDB\p2\pool\plugins\org.eclipse.justj.openjdk.hotsp
Elektronik cihaz başlatılıyor...
Bilgisayar sistemi hazır.
Dizüstü bilgisayar: 16 GB RAM, i7 işlemci.
```

//çağrı aşağıdan yukarı doğru olurken  
//constructorlar yukarıdan aşağı iner

```
*Device.java X  Computer.java  Laptop.java  DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Device >
9
10
11 public class Device {
12     public Device() {
13         System.out.println("Elektronik cihaz başlatılıyor...");
14     }
15 }
16
17
18
```

Computer Sınıfı

```
*Device.java  Computer.java X  Laptop.java  DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Computer >
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Computer extends Device{
4     public Computer() {
5         System.out.println("Bilgisayar sistemi hazır.");
6     }
7 }
8
```

Laptop Sınıfı

```
*Device.java  Computer.java  *Laptop.java X  DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Laptop > Laptop()
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Laptop extends Computer {
4     public Laptop() {
5         System.out.println("Dizüstü bilgisayar: "
6             + "16 GB RAM, i7 işlemci.");
7     }
8 }
9
```

# SUPER()

```
*Device.java *Computer.java *Laptop.java *DeviceRunner.java X
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > DeviceRunner > main(String[]) : void
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class DeviceRunner {
4
5     public static void main(String[] args) {
6
7         Laptop h1= new Laptop();
8     }
9 }
```

```
Console X
<terminated> DeviceRunner [Java Application] C:\Users\BIDB\p2\pool\plugins\org.eclipse.justj.openjdk.hotsp
Elektronik cihaz başlatılıyor...
Bilgisayar sistemi hazır.
Dizüstü bilgisayar: 16 GB RAM, i7 işlemci.
```

Bir child class'ta bir object oluşturulduğunda, otomatik olarak çağrılır.  
Super çağrısı parentınızın constructorını çağırır.

## Device Sınıfı

```
*Device.java *Computer.java *Laptop.java *DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Device >
9
10
11 public class Device {
12     public Device() {
13         System.out.println("Elektronik cihaz başlatılıyor...");
14     }
15 }
16
17
18
```

## Computer Sınıfı

```
*Device.java *Computer.java *Laptop.java *DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Computer >
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Computer extends Device{
4     public Computer() {
5         System.out.println("Bilgisayar sistemi hazır.");
6     }
7 }
8
```

## Laptop Sınıfı

```
*Device.java *Computer.java *Laptop.java *DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Laptop > Laptop()
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Laptop extends Computer {
4     public Laptop() {
5         System.out.println("Dizüstü bilgisayar: "
6             + "16 GB RAM, i7 işlemci.");
7     }
8 }
9
```

# SUPER()

```
*Device.java  *Computer.java  *Laptop.java  *DeviceRunner.java X
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > DeviceRunner > main(String[]) : void
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class DeviceRunner {
4
5     public static void main(String[] args) {
6
7         Laptop h1= new Laptop();
8     }
9 }
```

```
Console X
<terminated> DeviceRunner [Java Application] C:\Users\BIDB\p2\pool\plugins\org.eclipse.justj.openjdk.hotsp
Elektronik cihaz başlatılıyor...
Bilgisayar sistemi hazır.
Dizüstü bilgisayar: 16 GB RAM, i7 işlemci.
```

Biz bunu constructor hiyerşisinde gördük.  
Constructor çağrıları parent a doğru gitti.  
Cevaplar da parenttan başladı.  
İşte burada süper çağrı yapıldı

## Device Sınıfı

```
*Device.java  *Computer.java  *Laptop.java  *DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Device >
9
10
11 public class Device {
12     public Device() {
13         System.out.println("Elektronik cihaz başlatılıyor...");
14     }
15 }
16
17
18
```

## Computer Sınıfı

```
*Device.java  *Computer.java  *Laptop.java  *DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Computer >
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Computer extends Device{
4     public Computer() {
5         System.out.println("Bilgisayar sistemi hazır.");
6     }
7 }
8
```

## Laptop Sınıfı

```
*Device.java  *Computer.java  *Laptop.java  *DeviceRunner.java
NesneYonemimliYazilim > src > haftaaa10ConstructorHiyerarsisi > Laptop > Laptop()
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Laptop extends Computer {
4     public Laptop() {
5         System.out.println("Dizüstü bilgisayar: "
6             + "16 GB RAM, i7 işlemci.");
7     }
8 }
9
```

# SUPER()

```
*Device.java  *Computer.java  *Laptop.java  *DeviceRunner.java X
NesneYonemliYazilim > src > haftaaa10ConstructorHiyerarsisi > DeviceRunner > main(String[]) : void
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class DeviceRunner {
4
5     public static void main(String[] args) {
6
7         Laptop h1= new Laptop();
8     }
```

Console X

<terminated> DeviceRunner [Java Application] C:\Users\BIDB\p2\pool\plugins\org.eclipse.justj.openjdk.hotsp

Elektronik cihaz başlatılıyor...

Bilgisayar sistemi hazır.

Dizüstü bilgisayar: 16 GB RAM, i7 işlemci.

Bu çağrı default olarak yapılan bu çağrıya Super() çağrı deriz.  
super() çağrısını manuel olarak da çağırabilirsiniz ve istediğimiz constructora yönlendirebiliriz.

Device Sınıfı

```
*Device.java  *Computer.java  *Laptop.java  *DeviceRunner.java
NesneYonemliYazilim > src > haftaaa10ConstructorHiyerarsisi > Device >
9
10
11 public class Device {
12     public Device() {
13         System.out.println("Elektronik cihaz başlatılıyor...");
14     }
15 }
16
17
18
```

Computer Sınıfı

```
*Device.java  *Computer.java  *Laptop.java  *DeviceRunner.java
NesneYonemliYazilim > src > haftaaa10ConstructorHiyerarsisi > Computer >
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Computer extends Device{
4     public Computer() {
5         System.out.println("Bilgisayar sistemi hazır.");
6     }
7 }
8
```

Laptop Sınıfı

```
*Device.java  *Computer.java  *Laptop.java  *DeviceRunner.java
NesneYonemliYazilim > src > haftaaa10ConstructorHiyerarsisi > Laptop > Laptop()
1 package haftaaa10ConstructorHiyerarsisi;
2
3 public class Laptop extends Computer {
4     public Laptop() {
5         System.out.println("Dizüstü bilgisayar: "
6             + "16 GB RAM, i7 işlemci.");
7     }
8 }
9
```

# Neden Super(); Çağrısı Yapalım?

1- super keyword'unu kullanırsanız, hem parent class'in methodunu çağırabilir hem de bu methodun davranışını child class'in ihtiyacına göre genişletebilirsiniz.

2 - extends ise sadece miras almayı sağlar, ancak parent class'in methodlarını değiştirmez.

```
package haftaaa11SuperCagrisi;
public class Runner {
    public static void main(String[] args) {
        Torun t1= new Torun("Araba");
    }
}
```

Yakışır torunuma! Ben destek olurum  
Daha erken önce para biriktir.  
ben biraz para biriktireyim. Sen ve dedem de katkı edersiniz değil mi?

```
package haftaaa11SuperCagrisi;
public class Dede {
    public Dede() {
        super();
        System.out.println("Torun büyüdü"
            + " ama araba felan istemiyor, ilginç");
    }
    public Dede(String soruIstek) {
        super();
        System.out.println("Yakışır torunuma!"
            + " Ben destek olurum");
    }
}
```

```
package haftaaa11SuperCagrisi;
public class Torun extends Baba {
    public Torun() {
        super();
        System.out.println("ne arabası ya oyuncak mı?");
    }
    public Torun(String cocukIstek) {
        super("baba ben araba istiyorum");
        System.out.println("ben biraz para biriktireyim"
            + ". Sen ve dedem de katkı edersiniz değil mi?");
    }
}
```

```
package haftaaa11SuperCagrisi;
public class Baba extends Dede {
    public Baba() {
        super();
        System.out.println("Sakin bizim "
            + "çocuk demekki zamanı var");
    }
    public Baba(String fikirIstek) {
        super("Dedesi sen ne diyorsun");
        System.out.println("Daha erken "
            + "önce para biriktir.");
    }
}
```

```
package haftaaa11SuperCagrisi;
public class Runner {
    public static void main(String[] args) {
        Torun t1= new Torun();
    }
}
```

Torun büyüdü ama araba felan istemiyor, ilginç  
Sakin bizim çocuk demekki zamanı var  
ne arabası ya oyuncak mı?

```
package haftaaa11SuperCagrisi;
public class Dedede {
    public Dedede() {
        super();
        System.out.println("Torun büyüdü"
            + " ama araba felan istemiyor, ilginç");
    }
    public Dedede(String soruIstek) {
        super();
        System.out.println("Yakışır torunuma!"
            + "Ben destek olurum");
    }
}
```

```
package haftaaa11SuperCagrisi;
public class Torun extends Baba {
    public Torun() {
        super();
        System.out.println("ne arabası ya oyuncak mı?");
    }
    public Torun(String cocukIstek) {
        super("baba ben araba istiyorum");
        System.out.println("ben biraz para biriktireyim"
            + ".Sen ve dedem de katkı edersiniz değil mi?");
    }
}
```

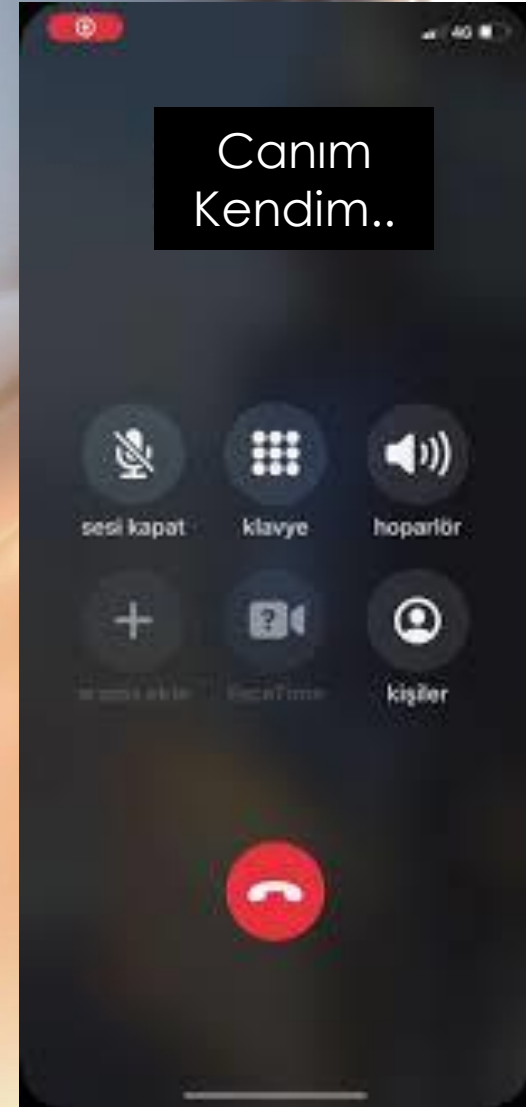
```
package haftaaa11SuperCagrisi;
public class Baba extends Dedede {
    public Baba() {
        super();
        System.out.println("Sakin bizim "
            + "çocuk demekki zamanı var");
    }
    public Baba(String fikirIstek) {
        super("Dedesi sen ne diyorsun");
        System.out.println("Daha erken "
            + "önce para biriktir.");
    }
}
```

# This(); Çağrısı

## THIS CALL

\*bir sınıfın kendi içinde başka bir constructor'ı çağırmak için kullanılır.

\*constructor'lar arasında kod tekrarını azaltmak ve bir constructor'ın diğer constructor ile aynı işlevleri gerçekleştirmesini sağlamak için faydalıdır.

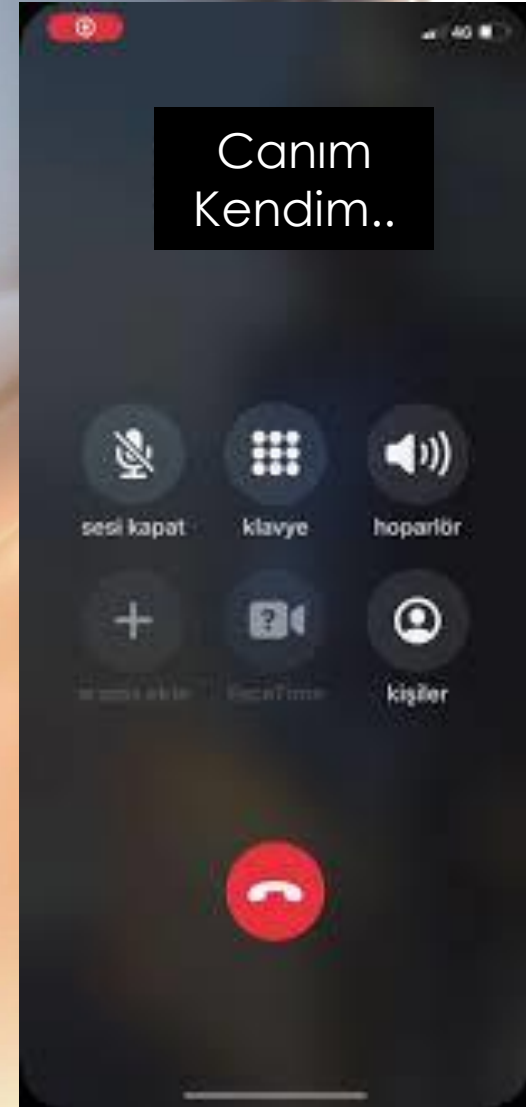


# This(); Çağrısı

## THIS CALL

\*this() ile bir constructor, aynı sınıftaki başka bir constructor'ı parametrelerle birlikte çağırabilir.

\*Önemli nokta, this() çağrısının constructor'ın ilk satırında yapılması gerektiridir.



# This(); Çağrısı ve Super Çağrısı

This kendine çağrı  
Super parentına çağrı

this() ve super() her zaman ilk satırda olmalıdır.

Bu yüzden bir constructor'da ikisi bir arada kullanılamazlar.

```
package haftaaa11SuperCagrisi;
public class Runner {
    public static void main(String[] args) {
        Torun t1= new Torun("Araba");
    }
}
```

gençlikte buraların tozunu attık  
 Yakışır torunuma! Ben destek olurum  
 Daha erken önce para biriktir.  
 ben biraz para biriktireyim. Sen ve dedem de katkı edersiniz değil mi?

```
package haftaaa11ThisCagrisi;
public class Dedede {
    public Dedede() {
        super();
        System.out.println("gençlikte buraların tozunu attık");
    }
    public Dedede(String soruIstek) {
        this();
        System.out.println("Yakışır torunuma!"
            + "Ben destek olurum");
    }
}
```

```
package haftaaa11SuperCagrisi;
public class Torun extends Baba {
    public Torun() {
        super();
        System.out.println("ne arabası ya oyuncak mı?");
    }
    public Torun(String cocukIstek) {
        super("baba ben araba istiyorum");
        System.out.println("ben biraz para biriktireyim"
            + ".Sen ve dedem de katkı edersiniz değil mi?");
    }
}
```

```
package haftaaa11SuperCagrisi;
public class Baba extends Dedede {
    public Baba() {
        super();
        System.out.println("Sakin bizim "
            + "çocuk demekki zamanı var");
    }
    public Baba(String fikirIstek) {
        super("Dedesi sen ne diyorsun");
        System.out.println("Daha erken "
            + "önce para biriktir.");
    }
}
```

# This(); Çağrısı ve Super Çağrısı

## 1. Görev;

- This kendine çağrı
- Super parentına çağrı

şimdiye kadar hep constructorlara çağrıda bulunduk metod ve değişkenlere ulaşabilir miyiz?

## 2. Görev;

- This kendinde bulunan bir metod ya da değişkeni çağırır.
- Super parentında bulunan metod ya da değişkeni çağırır.

```
package haftaaaaa11CagrilarDegiskenlerle;
public class Runner {
    public static void main(String[] args) {
        Torun t1= new Torun("Araba");
    }
}
```

Console X

<terminated> Runner (5) [Java Application] C:\Users\BIDBI\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_21.0.7.v20250502-0916\jre\bin\javaw.exe

gençlikte buraların tozunu attık  
Yakışır torunuma! Ben destek olurum  
Daha erken önce para biriktir.  
ben biraz para biriktireyim. Sen ve dedem de katkı edersiniz değil mi?  
Kendi cüzdanımda ne var 100  
Babamın cüzdanında ne var 1000  
Dedemin cüzdanında ne var 10000  
toplam paramız 11100

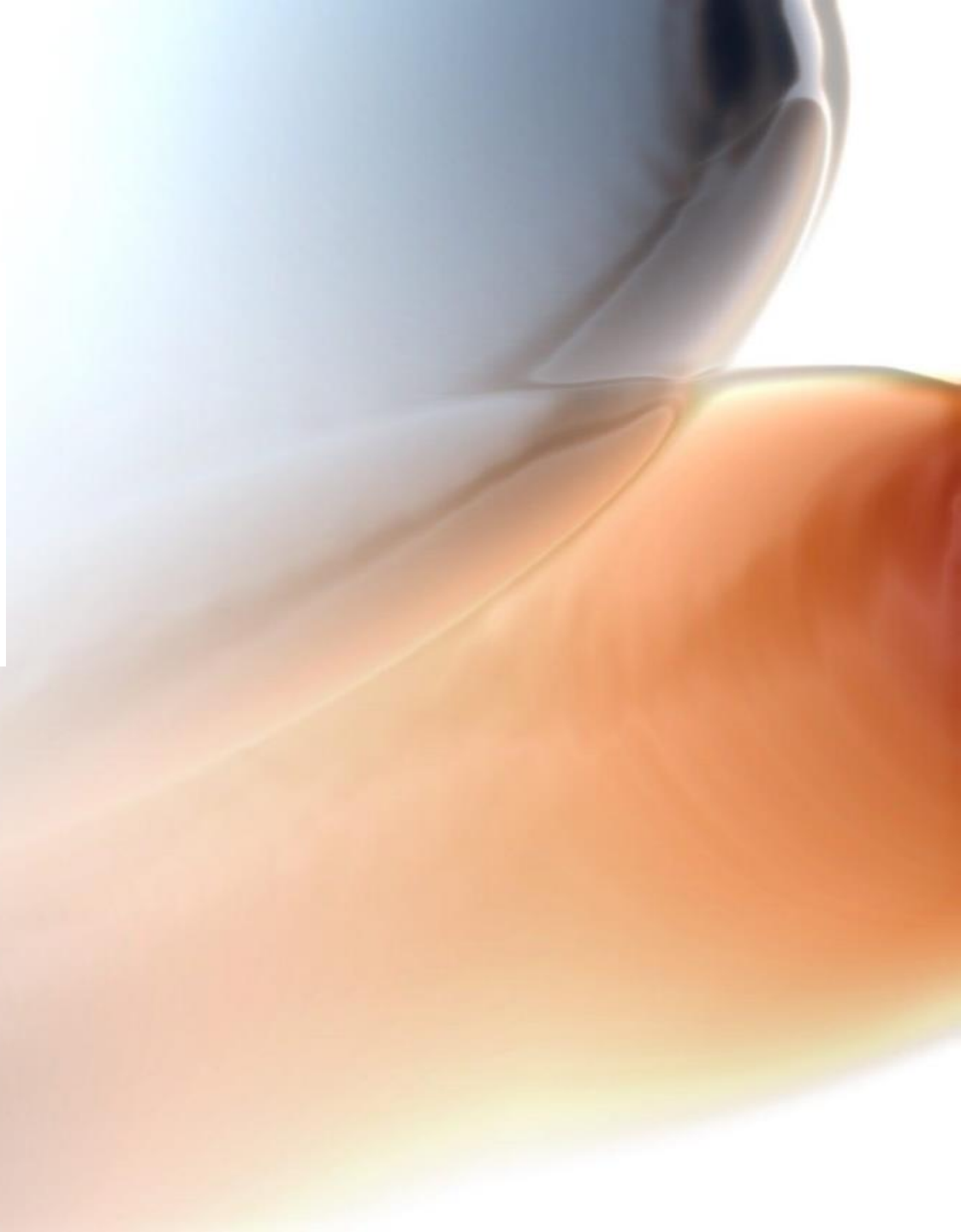
```
package haftaaaaa11CagrilarDegiskenlerle;
public class Dedede {
    int dedeCuzdan=10000;
    public Dedede() {
        super();
        System.out.println("gençlikte buraların tozunu attık");
    }
    public Dedede(String soruIstek) {
        this();
        System.out.println("Yakışır torunuma!"
            + "Ben destek olurum");
    }
}
```

```
package haftaaaaa11CagrilarDegiskenlerle;
public class Torun extends Baba {
    int torunCuzdan=100;
    public Torun() {
        super();
        System.out.println("ne arabası ya oyuncak mı?");
    }
    public Torun(String cocukIstek) {
        super("baba ben araba istiyorum");
        System.out.println("ben biraz para biriktireyim"
            + ".Sen ve dedem de katkı edersiniz değil mi?");
        System.out.println("Kendi cüzdanımda ne var"+this.torunCuzdan);
        System.out.println("Babamın cüzdanında ne var"+super.babaCuzdan);
        System.out.println("Dedemin cüzdanında ne var"+super.dedeCuzdan);
        int toplam=this.torunCuzdan+super.babaCuzdan+super.dedeCuzdan;
        System.out.println("toplam paramız"+toplam);
    }
}
```

```
package haftaaaaa11CagrilarDegiskenlerle;
public class Baba extends Dedede {
    int babaCuzdan=1000;
    public Baba() {
        super();
        System.out.println("Sakin bizim "
            + "çocuk demekki zamanı var");
    }
    public Baba(String fikirIstek) {
        super("Dedesi sen ne diyorsun");
        System.out.println("Daha erken "
            + "önce para biriktir.");
    }
}
```

# OOP 4 TEMEL ÖZELLİĞİ

1. Inheritance - Miras Alma
2. **Polymorphism** - Çok Biçimlilik
3. Encapsulation - Kapsülleme
4. Abstraction - Soyutlama



# POLYMORPHISM

Polymorphism (Çok Biçimlilik), nesne yönelimli programlamada, bir sınıfın farklı şekillerde davranabilmesi anlamına gelir.

Java'da polymorphism, aynı isimdeki bir metodu farklı sınıflarda veya aynı sınıf içinde farklı parametrelerle kullanabilme yeteneği sağlar.

Polymorphism iki ana şekilde uygulanabilir

- Method Overriding (Metod Geçersiz Kılma)**: Bir alt sınıfın, üst sınıfında tanımlı bir metodu kendine göre yeniden tanımlaması.

- Method Overloading (Metod Aşırı Yükleme)**: Aynı sınıf içinde aynı isimli metotların farklı parametrelerle tanımlanması.

# Overriding

```
Payment.java x CreditCardPayment.java DebitCardPayment.java CashPayment.java PaymentRunner.java
1 package hafta5_4Polymorphism;
2
3 public class Payment {
4     public void processPayment(double amount) {
5         System.out.println("Ödeme işleniyor: " + amount + " TL");
6     }
7     /*Ödeme işlemleri parent class*/
8 }
9
```

Payment Sınıfı

Process  
payment

# LARI

Payment.java CreditCardPayment.java X DebitCardPayment.java CashPayment.java PaymentRunner.java

```
1 package hafta5_4Polymorphism;
2
3 public class CreditCardPayment extends Payment {
4     /*Kredi kartı işlemleri için oluşturuldu
5     * parent class= Payment*/
6
7 }
8
```

Payment.java CreditCardPayment.java DebitCardPayment.java \*CashPayment.java X PaymentRunner.java

```
1 package hafta5_4Polymorphism;
2
3 public class CashPayment extends Payment{
4     /*Nakit işlemleri için oluşturuldu
5     * parent class= Payment*/
6 }
7
```

Payment.java CreditCardPayment.java DebitCardPayment.java X CashPayment.java PaymentRunner.java

```
1 package hafta5_4Polymorphism;
2
3 public class DebitCardPayment extends Payment {
4     /*Banka kartı işlemleri için oluşturuldu
5     * parent class 1 Payment*/
6 }
7
```

Payment Sınıfı

Process  
payment

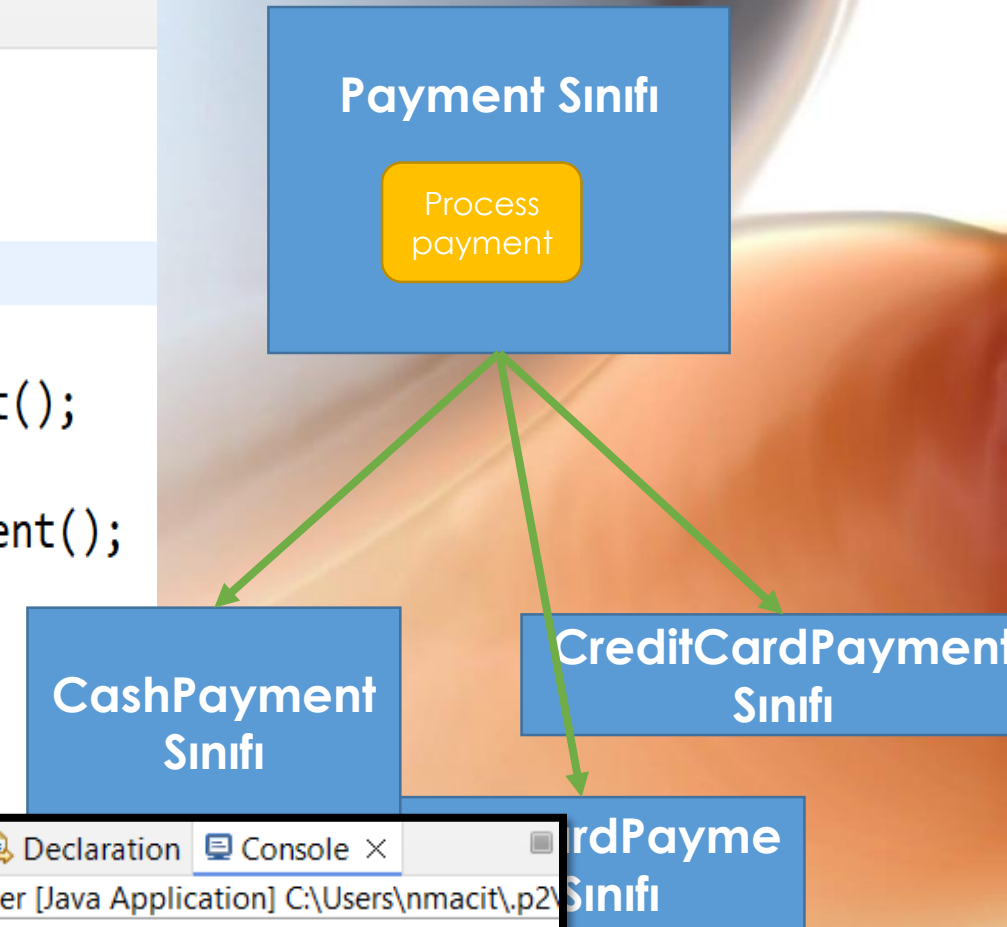
CashPayment  
Sınıfı

CreditCardPayment  
Sınıfı

DebitCardPayme  
nt Sınıfı

# Overriding

```
Payment.java CreditCardPayment.java DebitCardPayment.java CashPayment.java PaymentRunner.java X
1 package hafta5_4Polymorphism;
2
3 public class PaymentRunner {
4     public static void main(String[] args) {
5         Payment payment1 = new Payment();
6         DebitCardPayment payment2 = new DebitCardPayment();
7         CashPayment payment3 = new CashPayment();
8         CreditCardPayment payment4 = new CreditCardPayment();
9
10        payment1.processPayment(150.0);
11        payment2.processPayment(75.0);
12        payment3.processPayment(50.0);
13        payment4.processPayment(50.0);
14
15    }
16
17 }
```



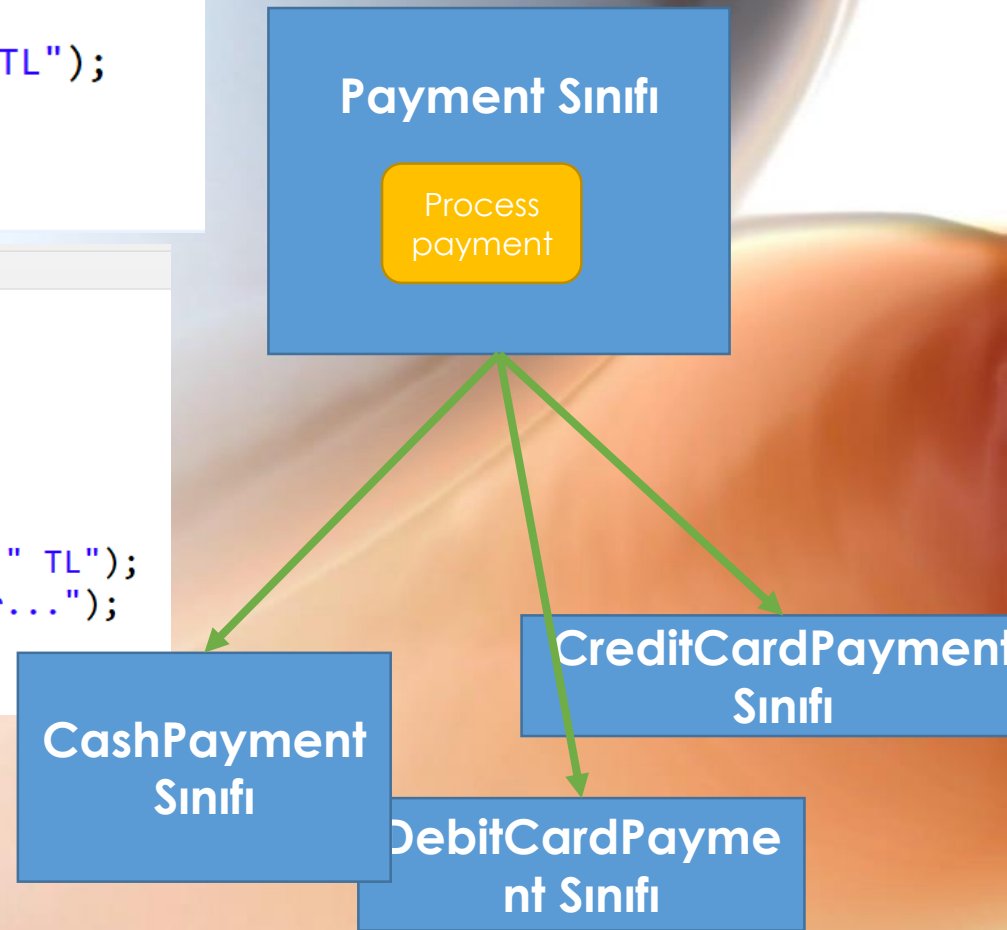
```
Problems Javadoc Declaration Console X
<terminated> PaymentRunner [Java Application] C:\Users\nmacit\p2V
Ödeme işleniyor: 150.0 TL
Ödeme işleniyor: 75.0 TL
Ödeme işleniyor: 50.0 TL
Ödeme işleniyor: 50.0 TL
```

Payment.java × CreditCardPayment.java DebitCardPayment.java \*CashPayment.java PaymentRunner.java

```
1 package hafta5_4Polymorphism;
2
3 public class Payment {
4     public void processPayment(double amount) {
5         System.out.println("Ödeme işleniyor: " + amount + " TL");
6     }
7     /*Ödeme işlemleri parent class*/
8 }
```

Payment.java \*CreditCardPayment.java × DebitCardPayment.java \*CashPayment.java PaymentRunner.java

```
1 package hafta5_4Polymorphism;
2 public class CreditCardPayment extends Payment {
3     /*Kredi kartı işlemleri için oluşturuldu
4     * parent class= Payment*/
5     @Override
6     public void processPayment(double amount) {
7         System.out.println("Kredi kartı ile ödeme yapılıyor: " + amount + " TL");
8         System.out.println("Kredi kartı doğrulama ve işlem onayı yapılıyor...");
9     }
10 }
```

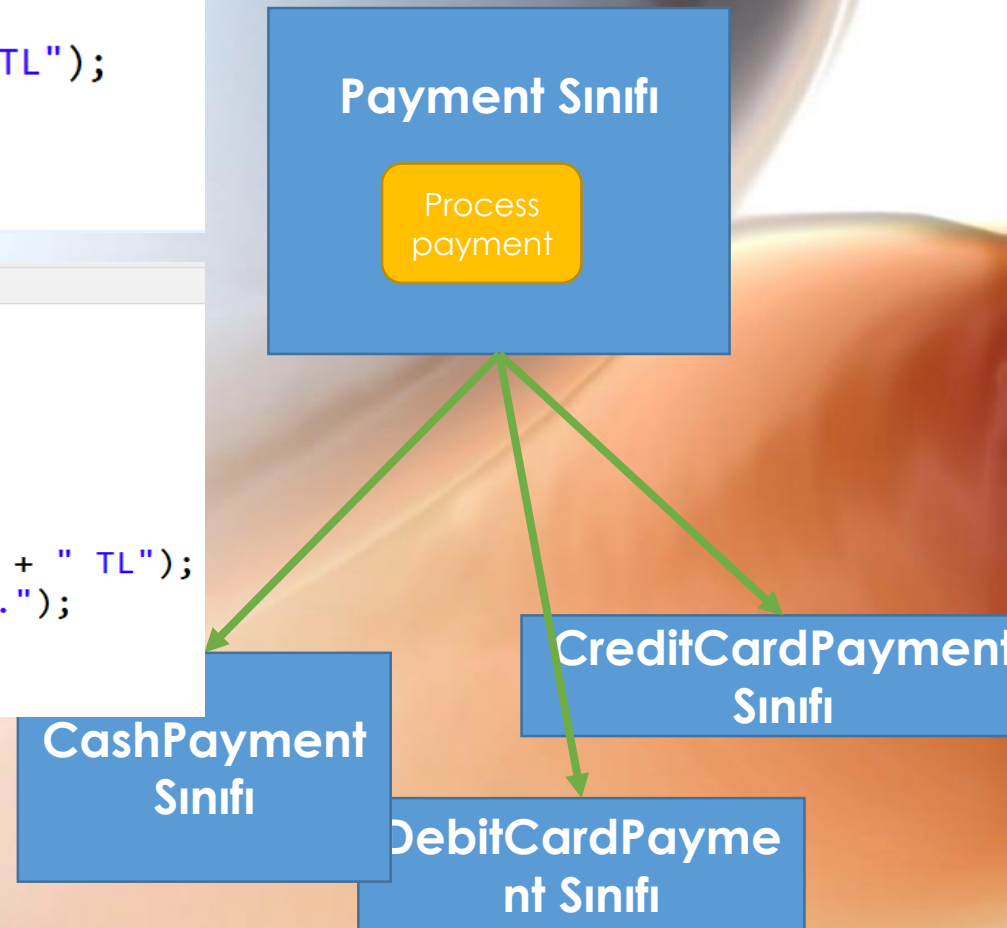


Payment.java × CreditCardPayment.java DebitCardPayment.java \*CashPayment.java PaymentRunner.java

```
1 package hafta5_4Polymorphism;
2
3 public class Payment {
4     public void processPayment(double amount) {
5         System.out.println("Ödeme işleniyor: " + amount + " TL");
6     }
7     /*Ödeme işlemleri parent class*/
8 }
```

Payment.java \*CreditCardPayment.java \*DebitCardPayment.java × \*CashPayment.java PaymentRunner.java

```
1 package hafta5_4Polymorphism;
2 public class DebitCardPayment extends Payment {
3     /*Banka kartı işlemleri için oluşturuldu
4     * parent class ı Payment*/
5     @Override
6     public void processPayment(double amount) {
7         System.out.println("Banka kartı ile ödeme yapılıyor: " + amount + " TL");
8         System.out.println("Banka hesabındaki bakiye kontrol ediliyor...");
9     }
10 }
```

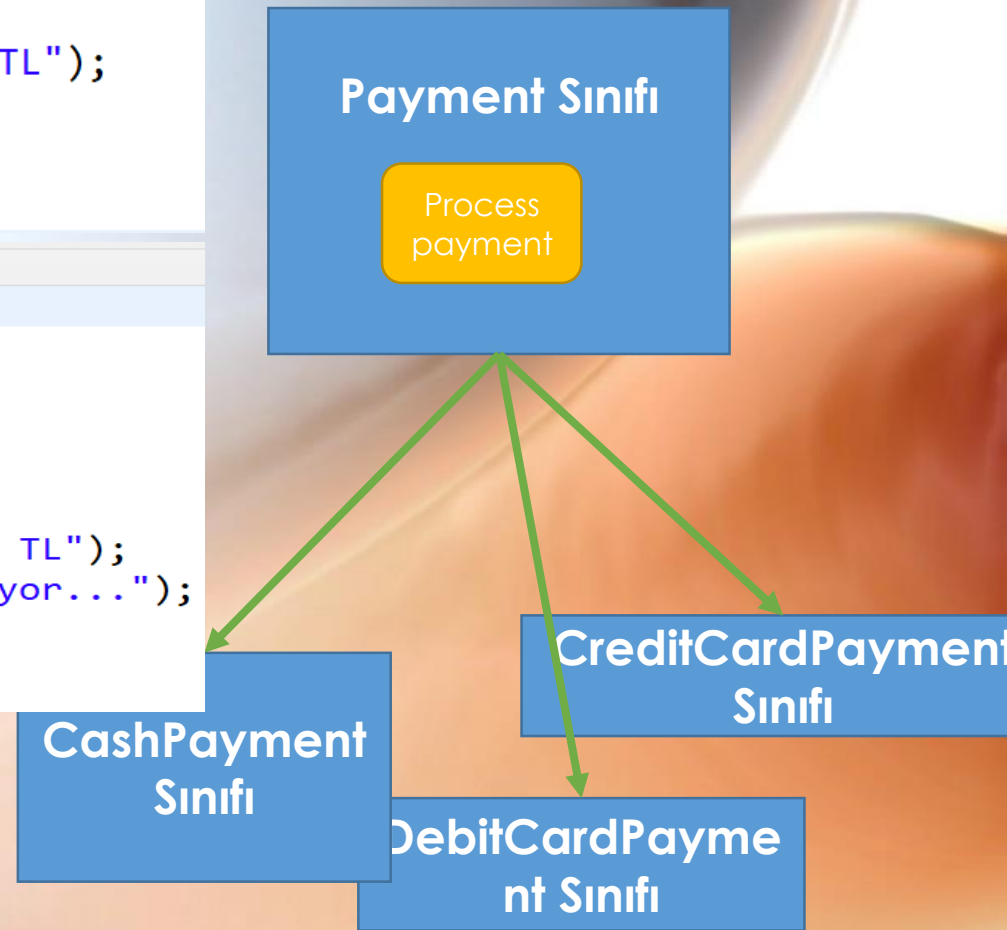


Payment.java × CreditCardPayment.java DebitCardPayment.java \*CashPayment.java PaymentRunner.java

```
1 package hafta5_4Polymorphism;
2
3 public class Payment {
4     public void processPayment(double amount) {
5         System.out.println("Ödeme işleniyor: " + amount + " TL");
6     }
7     /*Ödeme işlemleri parent class*/
8 }
```

Payment.java \*CreditCardPayment.java \*DebitCardPayment.java \*CashPayment.java × PaymentRunner.java

```
1 package hafta5_4Polymorphism;
2 public class CashPayment extends Payment{
3     /*Nakit işlemleri için oluşturuldu
4     * parent class= Payment*/
5     @Override
6     public void processPayment(double amount) {
7         System.out.println("Nakit ödeme yapılıyor: " + amount + " TL");
8         System.out.println("Kasiyere nakit ödeme alındı bildiriliyor...");
9     }
10 }
11
```



# Override

```
1 package hafta5_4Polymorphism;
2
3 public class PaymentRunner {
4     public static void main(String[] args) {
5         Payment payment1 = new Payment();
6         DebitCardPayment payment2 = new DebitCardPayment();
7         CashPayment payment3 = new CashPayment();
8         CreditCardPayment payment4 = new CreditCardPayment();
9
10        payment1.processPayment(150.0);
11        payment2.processPayment(75.0);
12        payment3.processPayment(50.0);
13        payment4.processPayment(50.0);
14    }
15 }
16 }
```

```
<terminated> PaymentRunner [Java Application] C:\Users\nmacit\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot
Ödeme işleniyor: 150.0 TL
Banka kartı ile ödeme yapılıyor: 75.0 TL
Banka hesabındaki bakiye kontrol ediliyor...
Nakit ödeme yapılıyor: 50.0 TL
Kasiyere nakit ödeme alındı bildiriliyor...
Kredi kartı ile ödeme yapılıyor: 50.0 TL
Kredi kartı doğrulama ve işlem onayı yapılıyor...
```

```
Payment.java x CreditCardPayment.java DebitCardPayment.java *CashPayment.java PaymentRunner.java
1 package hafta5_4Polymorphism;
2
3 public class Payment {
4     public void processPayment(double amount) {
5         System.out.println("Ödeme işleniyor: " + amount + " TL");
6     }
7     /*Ödeme işlemleri parent class*/
8 }
```

```
Payment.java CreditCardPayment.java DebitCardPayment.java *CashPayment.java PaymentRunner.java
1 package hafta5_4Polymorphism;
2 public class DebitCardPayment extends Payment {
3     /*Banka kartı işlemleri için oluşturuldu
4     * parent class ı Payment*/
5     @Override
6     public void processPayment(double amount) {
7         System.out.println("Banka kartı ile ödeme yapılıyor: " + amount + " TL");
8         System.out.println("Banka hesabındaki bakiye kontrol ediliyor...");
9     }
10 }
```

```
Payment.java CreditCardPayment.java DebitCardPayment.java *CashPayment.java PaymentRunner.java
1 package hafta5_4Polymorphism;
2 public class CashPayment extends Payment {
3     /*Nakit işlemleri için oluşturuldu
4     * parent class= Payment*/
5     @Override
6     public void processPayment(double amount) {
7         System.out.println("Nakit ödeme yapılıyor: " + amount + " TL");
8         System.out.println("Kasiyere nakit ödeme alındı bildiriliyor...");
9     }
10 }
```

## Sınıfı

```
Payment.java CreditCardPayment.java DebitCardPayment.java *CashPayment.java PaymentRunner.java
1 package hafta5_4Polymorphism;
2 public class CreditCardPayment extends Payment {
3     /*Kredi kartı işlemleri için oluşturuldu
4     * parent class= Payment*/
5     @Override
6     public void processPayment(double amount) {
7         System.out.println("Kredi kartı ile ödeme yapılıyor: " + amount + " TL");
8         System.out.println("Kredi kartı doğrulama ve işlem onayı yapılıyor...");
9     }
10 }
```

# Overriding

- Üst Sınıf Payment:** processPayment() adında genel bir ödeme işleme metodu içerir. Ancak, bu metot her ödeme türü için farklı bir işlem içerebilir.
- Alt Sınıflar (CreditCardPayment, DebitCardPayment, CashPayment):** Her bir alt sınıf Payment sınıfını genişletir ve processPayment() metodunu kendine özel olarak yeniden tanımlar. Bu sayede, her ödeme yöntemi için farklı bir işlem akışı sağlanır.
- Polymorphism:** PaymentTest sınıfında, Payment referansı kullanılarak farklı alt sınıfların processPayment() metodları çalıştırılır. Böylece, aynı metod farklı davranışlar sergileyerek polymorphism uygulanmış olur. Bu örnek sayesinde, bir ödeme sisteminde Method Overriding'in nasıl işlevsel ve esnek bir yapı sunduğunu görebiliriz.

# Overriding

**\*private method'lar override edilemezler** çünkü; *override edebilmek için o metoda başka bir class'dan ulaşabilmek gerekir. Ama private metotlar başka classlardan ulaşamaz metotlardır.*

**\*Static method'lar override edilemezler** çünkü; *static variable veya metotlar tüm objeler için ortaktır. Java static metotların override edilmesine müsaade etmez.*

**\*Final method'lar override edilemezler;** çünkü; *Method, final olarak işaretlenerek nihai ve değiştirilemez hale getirilir. Final olarak işaretlenmiş bir methodu child lar değiştiremez veya üzerine yazamaz. Final Variables, Final Classes, Final Parametreler'de olabilir. Sonuç: "Final" kelimesi, programcılara kodlarını daha güvenli ve öngörülebilir hale getirme, optimizasyon sağlama veya belirli davranışları zorlamada yardımcı olmak için kullanılır.*

# POLYMORPHISM AVANTAJLARI

- **Kod Esnekliği ve Genişletilebilirlik:** Yeni sınıflar eklenebilir ve polymorphism sayesinde kodun geri kalanında çok az veya hiç değişiklik yapmadan mevcut yapı genişletilebilir.
- **Bakımı Kolaylaştırır:** Polymorphic bir yapı ile nesnelerin davranışları kolayca değiştirilebilir ve güncellenebilir.
- **Kodun Tekrar Kullanılabilirliği:** Üst sınıfta tanımlanan ortak özellikler ve davranışlar alt sınıflar tarafından devralındığı için kod tekrarı azaltılır.

# Final

\*Java'da final anahtar kelimesi, değiştirilemez (sabit) olan yapıların tanımlanması için kullanılır.

\*final anahtar kelimesi,

**Değişkenler,**

**Metotlar,**

**Sınıflar**

üzerinde farklı işlevler sağlar.

# Final Değişkenler( Variables)

```
FinalVariableExample.java ×
1 package hafta5_5Final;
2
3 public class FinalVariableExample {
4     public static void main(String[] args) {
5         final int MAX_AGE = 100;
6         System.out.println("Max yaş: " + MAX_AGE);
7         // MAX_AGE = 120;
8         // Hata verir, çünkü final değişken tekrar atanamaz.
9     }
10 }
11
```

**Açıklama:** MAX\_AGE değeri bir kez atandıktan sonra bir daha değiştirilemez. Bu özellik, sabit değerlerin belirlenmesi için idealdir.

# Final Metotlar (Methods)

```
FinalVariableExample.java x Account.java x UserAccount.java
1 package hafta5_5Final;
2
3 public class Account {
4     public final void login() {
5         System.out.println("Kullanıcı sisteme giriş yapıyor...");
6     }
7
8     public void displayAccountType() {
9         System.out.println("Hesap türü: Genel Hesap");
10    }
11 }
12
```

```
FinalVariableExample.java Account.java *UserAccount.java x
1 package hafta5_5Final;
2
3 public class UserAccount extends Account{
4     @Override
5     public void displayAccountType() {
6         System.out.println("Hesap türü: Kullanıcı Hesabı");
7     }
8     // @Override
9     // public void login() {
10    //     System.out.println("Özel bir giriş işlemi.");
11    // } // Hata verir, çünkü login metodu final olarak tanımlandı.
12 }
13
```

final anahtar kelimesi bir metoda uygulandığında, o metodun alt sınıflar tarafından **override edilmesi engellenir**. Bu, özellikle önemli ve değişmez metotların koruma altına alınmasını sağlar.

# Final Sınıf (Class)

```
FinalVariableExample.java Account.java UserAccount.java Person.java X Doctor.java
1 package hafta5_5Final;
2
3 public final class Person {
4     public void think() {
5         System.out.println("İnsanlar düşünür");
6     }
7 }
8
```

```
FinalVariableExample.java Account.java UserAccount.java Person.java Doctor.java X
1 package hafta5_5Final;
2
3 public class Doctor extends Person{
4
5 }
6
```

Bir sınıf final olarak tanımlandığında, o sınıf **başka bir sınıf tarafından miras alınamaz**. Bu, sınıfın genişletilmesini ve değiştirilmesini engeller.

# Final Sınıf (Class)

- **Özet**
- **Final değişken:** Sabit olarak tanımlanır, bir kez atandıktan sonra değiştirilemez.
- **Final metod:** Alt sınıflarda override edilemez.
- **Final sınıf:** Başka sınıflar tarafından miras alınamaz.

# OOP 4 TEMEL ÖZELLİĞİ

1. Inheritance - Miras Alma
2. Polymorphism - Çok Biçimlilik
3. Encapsulation - Kapsülleme
4. Abstraction - Soyutlama

# Encapsulation-Kapsülleme

- Java'da **kapsülleme** (encapsulation), nesne yönelimli programlamanın (OOP) temel prensiplerinden biridir.
- Kapsülleme, bir sınıfın verilerini (değişkenlerini) ve bu verilere erişim sağlayan yöntemleri (metotları) bir arada tutma ve bu verilere doğrudan dışarıdan erişimi kısıtlama işlemidir.

# Kapsülleme

```
Person.java x PersonRunner.java
ileriNesneTabanlıProgramlama_Guz_24_25 > src > hafta6 > Person >
1 package hafta6;
2 public class Person {
3     private String name;    // Private değişken
4     private int age;
5     public String getName() {
6         return name;        // Getter metodu
7     }
8     public void setName(String name) {
9         this.name = name;    // Setter metodu
10    }
11    public int getAge() {
12        return age;          // Getter metodu
13    }
14    public void setAge(int age) { // Setter metodu (doğrulama ekledik)
15        if (age > 0) {
16            // Yaşın pozitif olması gerektiğini kontrol ediyoruz
17            this.age = age;
18        } else {
19            System.out.println("Yaş negatif olamaz!");
20        }
21    }
22 }
```

# Kapsülleme

```
Person.java x PersonRunner.java
ileriNesneTabanlıProgramlama_Guz_24_25 > src > hafta6 > Person >
1 package hafta6;
2 public class Person {
3     private String name;    // Private değişken
4     private int age;
5     public String getName() {
6         return name;        // Getter metodu
7     }
8     public void setName(String name) {
9         this.name = name;    // Setter metodu
10    }
11    public int getAge() {
12        return age;          // Getter metodu
13    }
14    public void setAge(int age) { // Setter metodu (doğrulama ekledik)
15        if (age > 0) {
16            // Yaşın pozitif olması gerektiğini kontrol ediyoruz
17            this.age = age;
18        } else {
19            System.out.println("Yaş negatif olamaz!");
20        }
21    }
22 }
```

# Kapsülleme

```
Person.java *PersonRunner.java x
ileriNesneTabanlıProgramlama_Guz_24_25 > src > hafta6 > PersonRunner > main(String[]) : void
1 package hafta6;
2
3 public class PersonRunner {
4     public static void main(String[] args) {
5         Person person = new Person();
6         // Setter metoduyla veri atama
7
8         person.setName("Naciye");
9         person.setAge(25);
10
11        // Getter metoduyla verileri alma
12        System.out.println("Ad: " + person.getName());
13        System.out.println("Yaş: " + person.getAge());
14
15        // Hatalı veri atama
16        person.setAge(-5);
17        // "Yaş negatif olamaz!" yazdırır.
18    }
19 }
```

# Kapsülleme Amaçları

- **Veri gizliliği:** Sınıfın içinde tanımlanan değişkenlere doğrudan erişimi engeller ve dış dünyadan gizler.
- **Kontrollü erişim:** Verilere erişimi getter ve setter metotlarıyla düzenler.
- **Kodun sürdürülebilirliği:** Verileri kontrol eden metotlar sayesinde kodun daha güvenli ve bakımı kolay olur.
- **Veri bütünlüğü:** Yanlış veya geçersiz verilerin sınıfa atanmasını engelleyebilir.

# Kapsülleme Verilerin Gizliliği

Sınıfın değişkenlerini private yaparak, başka bir sınıfın ya da dış dünyanın bu verilere doğrudan erişmesini engellersiniz. Bu, yanlışlıkla ya da kötü niyetli olarak verilere müdahale edilmesini önler.

## Örnek:

Bir bankanın müşteri bakiyesini tutan bir sistemde, bakiyeye doğrudan erişim verilirse şu durum oluşabilir:

```
account.balance = -500; // Negatif bakiye atanabilir, bu mantıksız!
```

Kapsülleme ile bu tür hataların önüne geçilir:

```
public void setBalance(double balance) {  
    if (balance >= 0) {  
        this.balance = balance;  
    } else {  
        System.out.println("Bakiye negatif olamaz!");  
    }  
}
```

# Kapsülleme Verilere Kontrollü Erişim

- Getter ve setter metotları ile değişkenlere erişim sağlanır. Böylece:
- Değişkenlere erişim ya da değiştirme işlemleri sınıfın kontrolü altında olur.
- Gerektiğinde ek kontroller veya işlem sırasında başka işlemler de yapılabilir.

**Örnek:** Bir kişinin yaşını negatif yapmamak için doğrulama eklenebilir:

```
public void setAge(int age) {  
    if (age > 0) {  
        this.age = age;  
    } else {  
        System.out.println("Yaş negatif olamaz!");  
    }  
}
```

# Kapsülleme Kodun Sürdürülebilirliğini Artırır

- Kapsülleme ile bir sınıfın iç yapısında değişiklik yapsanız bile, dışarıdan bu sınıfı kullanan kodları değiştirmek zorunda kalmazsınız.
- **Örnek:** Eğer bir sınıfın veri tutma mantığını değiştirmeniz gerekirse (örneğin, bir String yerine List<String> kullanmak gibi), getter ve setter metotlarının imzaları aynı kalırsa, bu sınıfı kullanan diğer kodlar etkilenmez.

# Kapsülleme Verinin Bütünlüğünü Korur

- Verilere doğrudan erişim engellendiği için, yanlış ya da geçersiz değerlerin atanması kontrol altına alınır. Bu, uygulamanızdaki hataları ve tutarsızlıkları azaltır.
- **Örnek:** Bir ürünün fiyatı hiçbir zaman negatif olamaz:

```
public void setPrice(double price) {  
    if (price >= 0) {  
        this.price = price;  
    } else {  
        System.out.println("Fiyat negatif olamaz!");  
    }  
}
```

# Kapsülleme

## 5. Bağımlılığı Azaltır

Kapsülleme sayesinde, bir sınıfın iç detayları gizlenir (abstraction). Dışarıdaki kod, bir sınıfın yalnızca ne yaptığını bilir, nasıl yaptığını bilmek zorunda kalmaz. Bu da sınıflar arasında düşük bağımlılık sağlar.

## 6. Daha Modüler ve Esnek Kodlama

Kapsülleme, sınıfın sorumluluklarını net bir şekilde sınırlandırır. Bu da kodun daha kolay test edilebilir, modifiye edilebilir ve yeniden kullanılabilir olmasını sağlar.

# Kapsülleme Avantajları

- **Güvenlik:** Veriler yetkisiz erişimden korunur.
- **Esneklik:** Verilere erişimi ve güncellemeyi kontrol etmek kolaylaşır.
- **Bakım Kolaylığı:** Kodda değişiklik yapılması gerektiğinde sadece setter ve getter metotlarını düzenlemek yeterlidir.
- Kapsülleme ile daha güvenli ve sürdürülebilir bir yapı elde edersiniz!