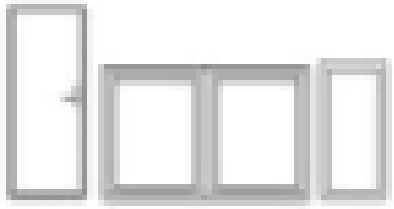
SINIFLAR

CLASS

Object Oriented Programming



SINIFLAR

Tuğla Sınıfı

Özellikler; en- boy-yükseklik
İşlevler; hacim

d1

d2

d3

en: 10
boy: 15
yükseklik:8

en: 20
boy: 30
yükseklik:10

en: 5
boy: 7
yükseklik:3

SINIFLAR

Erisim Belirleyici class sinifAdi()

{

..

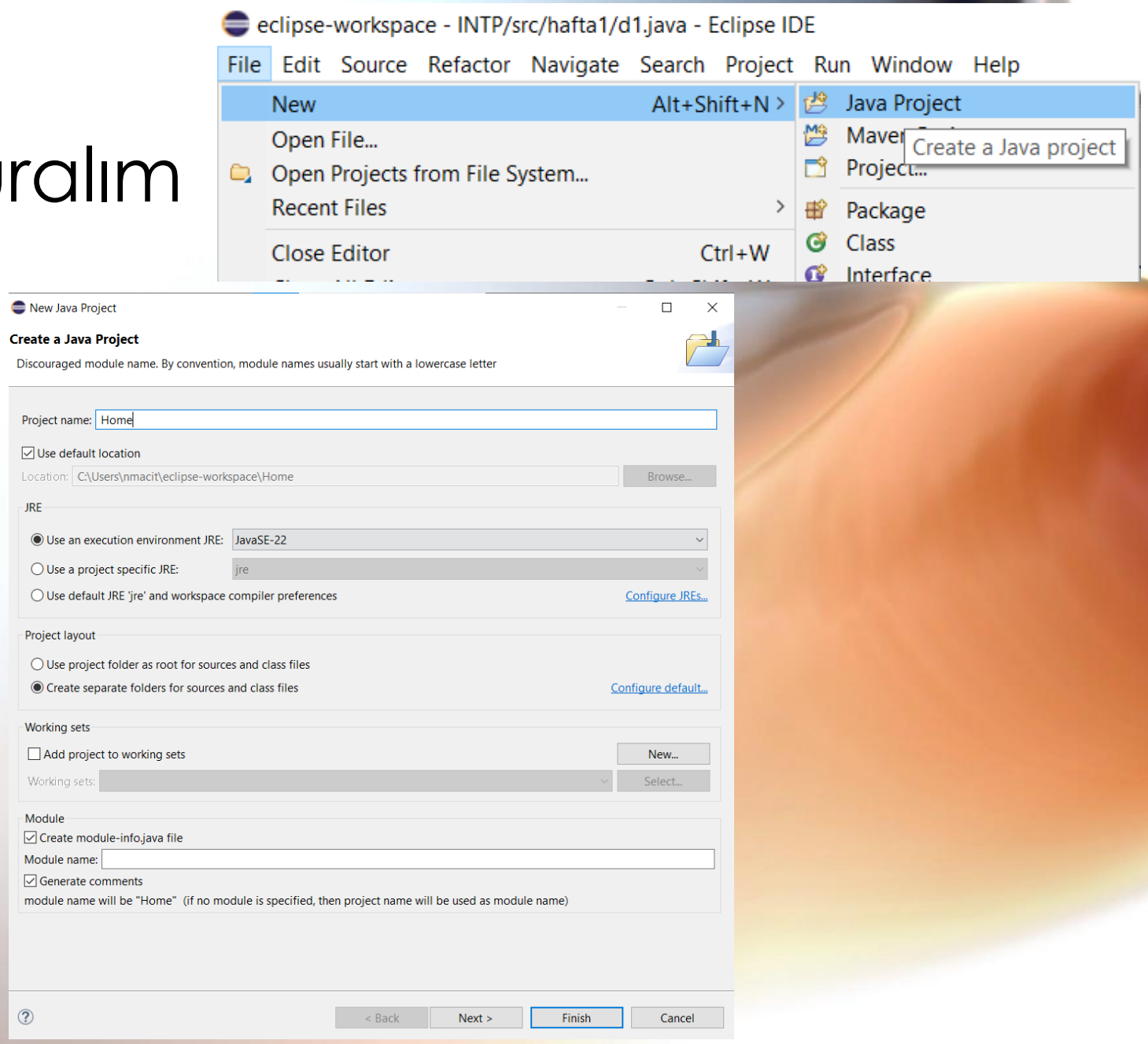
Sınıf ile ilgili parametreler

}

SINIFLAR

Yeni bir proje oluşturalım

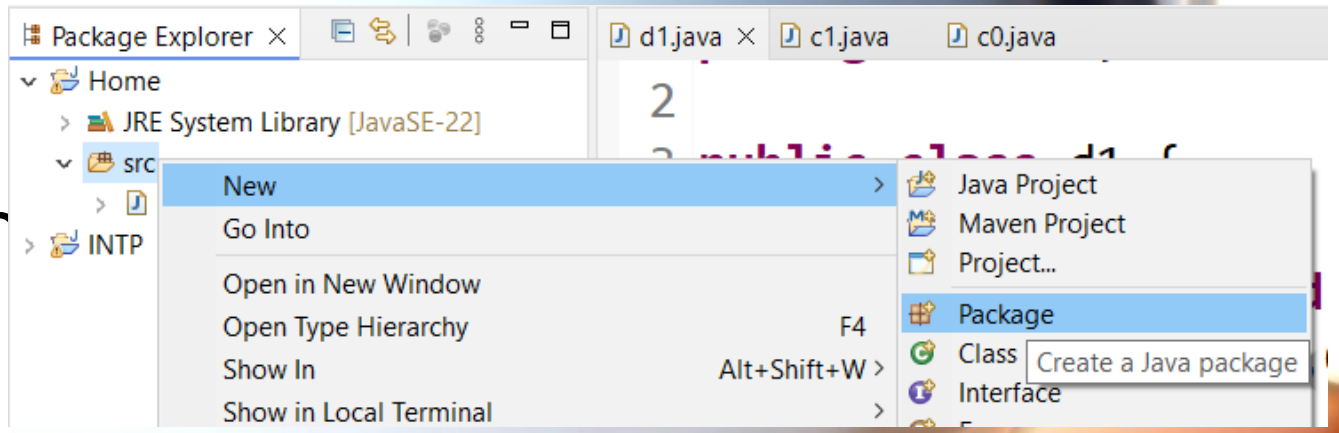
Adını "ev" koyalım.

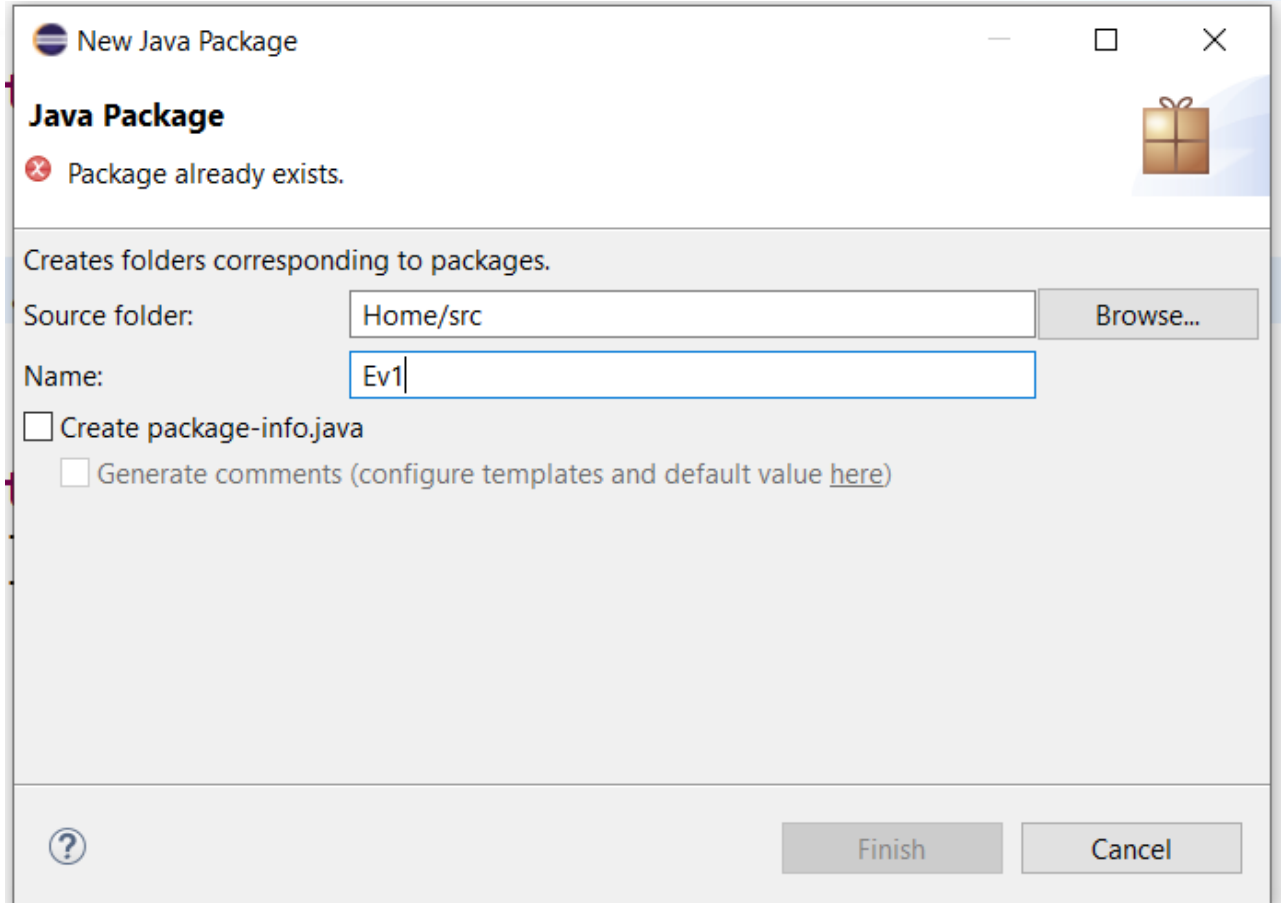


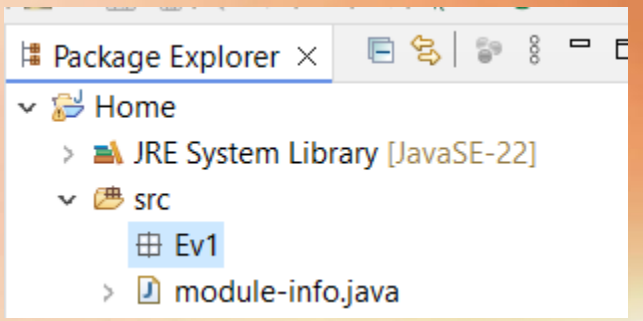
SINIFLAR

Yeni bir paket oluşturalım

Adını "Ev1" koyalım.



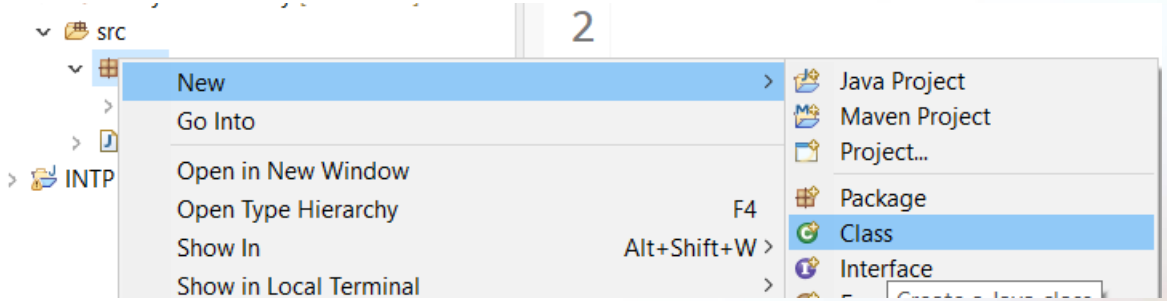


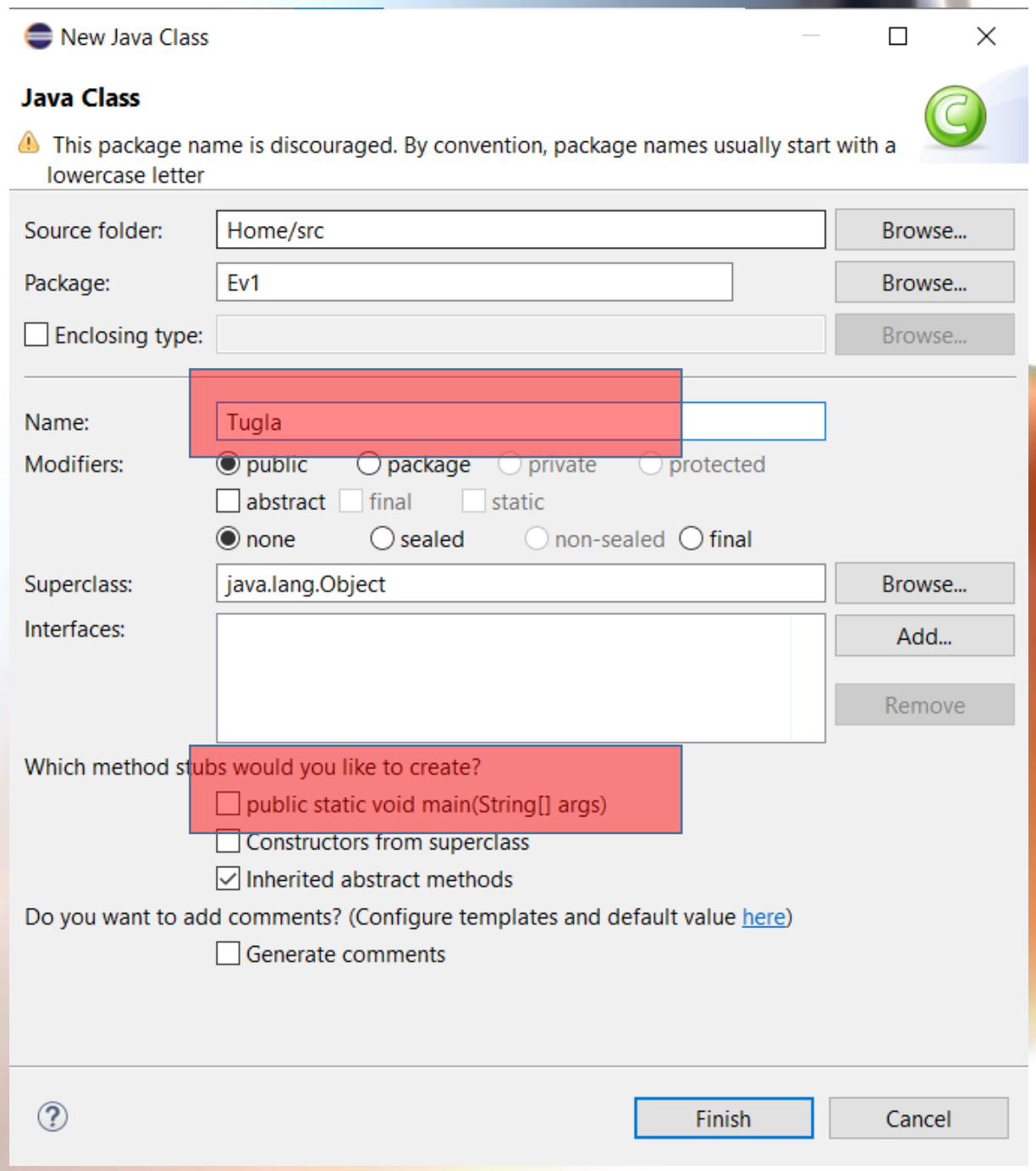


SINIFLAR

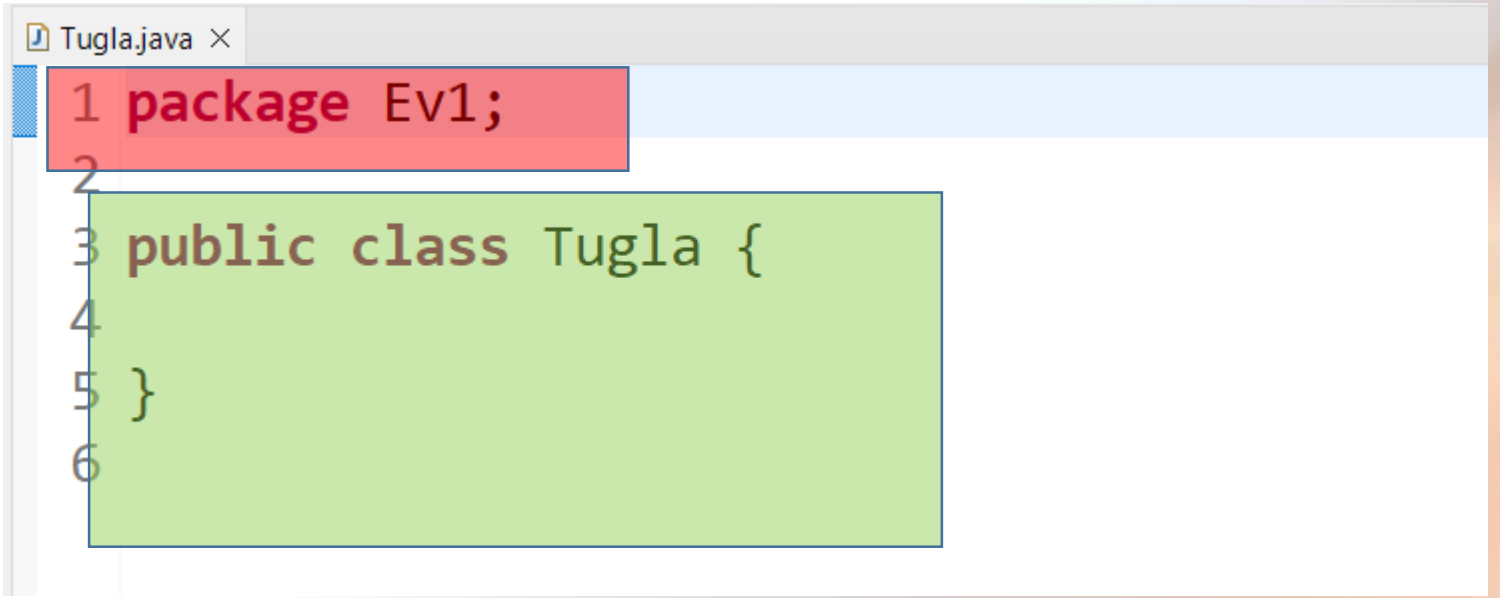
Ev1 paketi içerisine bir class oluşturalım

Adını "Tugla" koyalım.





SINIFLAR



SINIFLAR

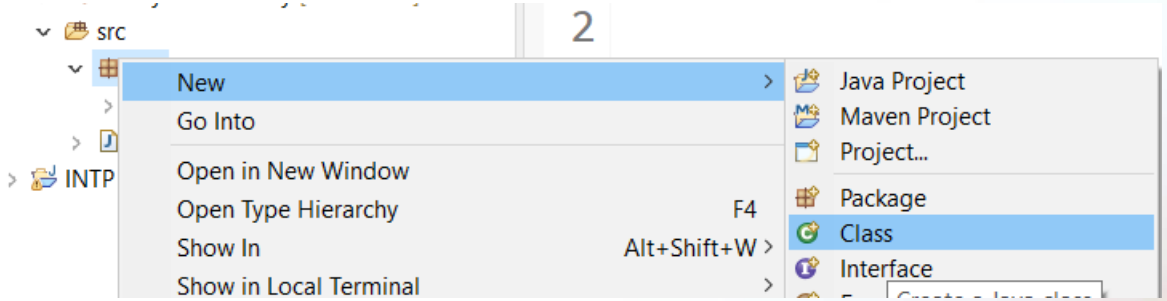
Tugla.java

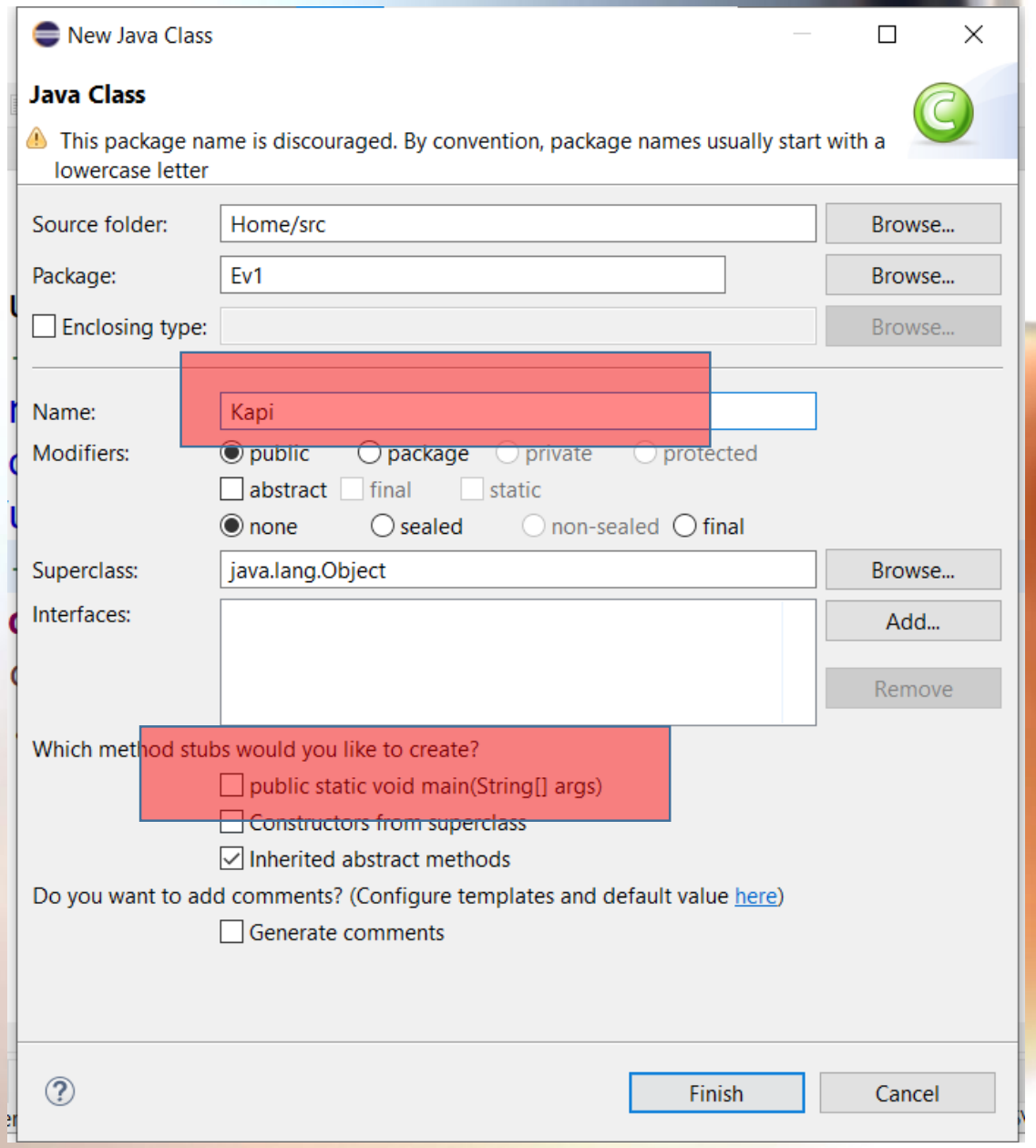
```
1  package Ev1;
2
3  public class Tugla {
4      //variable-değişkenler-Sınıfın pasif özellikleri
5      int tuglaEn=10;
6      int tuglaBoy=5;
7      int tuglaYukseklik=4;
8      //metotlar-Sınıfın eylemlerini gerçekleştiren öğeler
9      public void hacim () {
10         int hacim=tuglaBoy*tuglaEn*tuglaYukseklik;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

SINIFLAR

Ev1 paketi içerisine bir class oluşturalım

Adını "Kapi" koyalım.





Tugla.java | *Kapi.java

```java
package Ev1;

public class Kapi {
    int kapiYukseklik=190;
    int kapiGenislik=90;
    String kapiAcilisYonu="Sağ";
    String kapiRengi="Kahverengi";

    public void kapiAc() {
        if(kapiAcilisYonu.equals("Sağ")) {
            System.out.println("Kapı sağa açılan kapıdır");
        }
        else
        {
            System.out.println("Kapı sola açılan kapıdır");
        }
    }
}
```

SINIFLAR

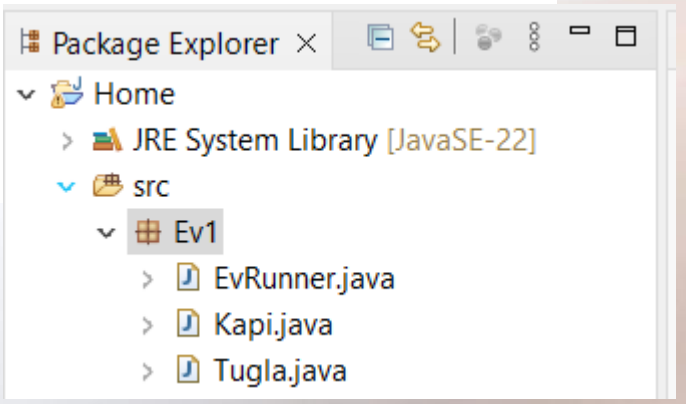
1- Bir veri yapısı veya yardımcı bir class oluştururken, genellikle main methoduna ihtiyaç duymazsınız.

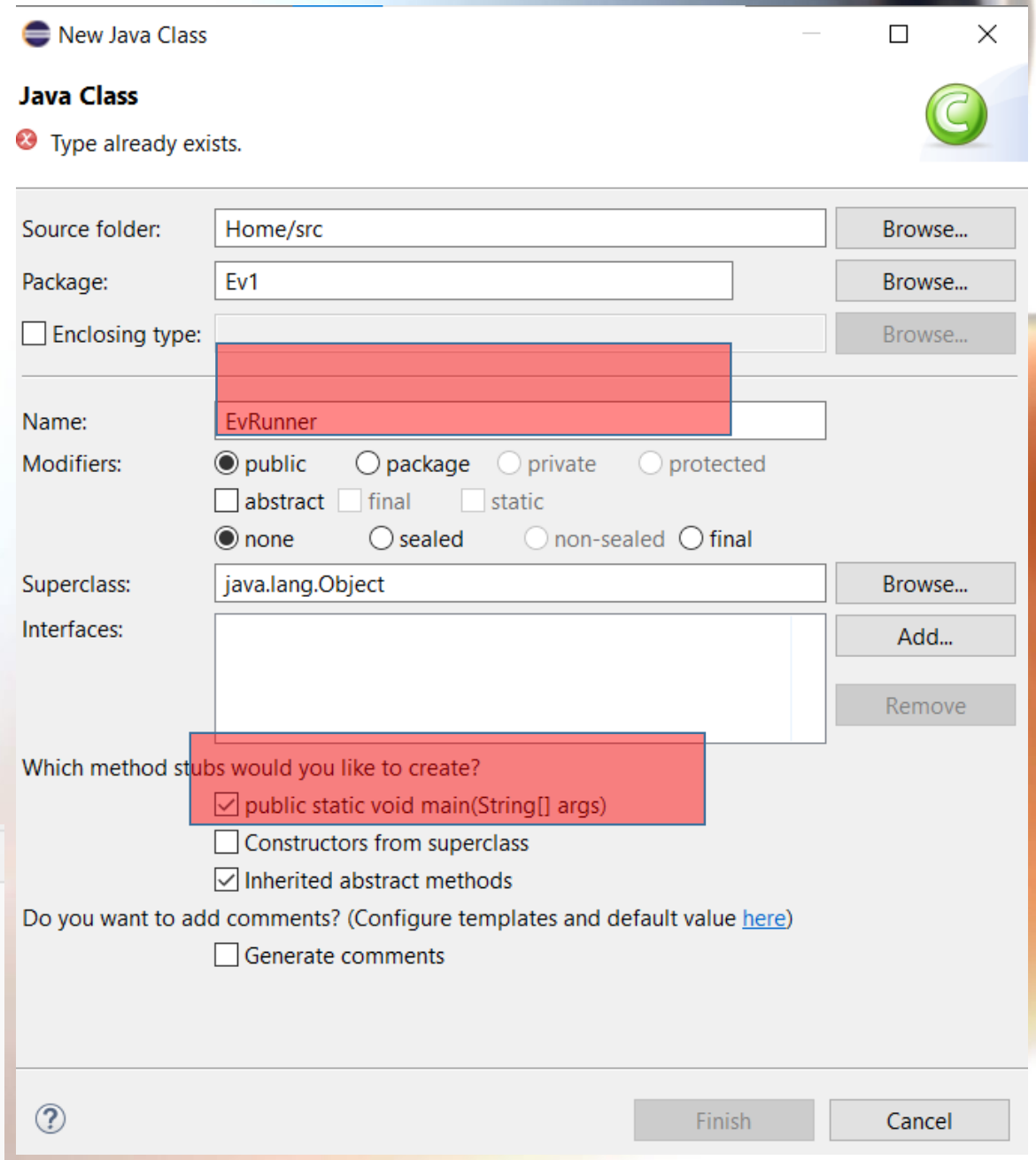
2- Ancak bir uygulama geliştirirken veya bağımsız bir program çalıştırırken,
main metodu oluşturmanız gerekebilir.

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.





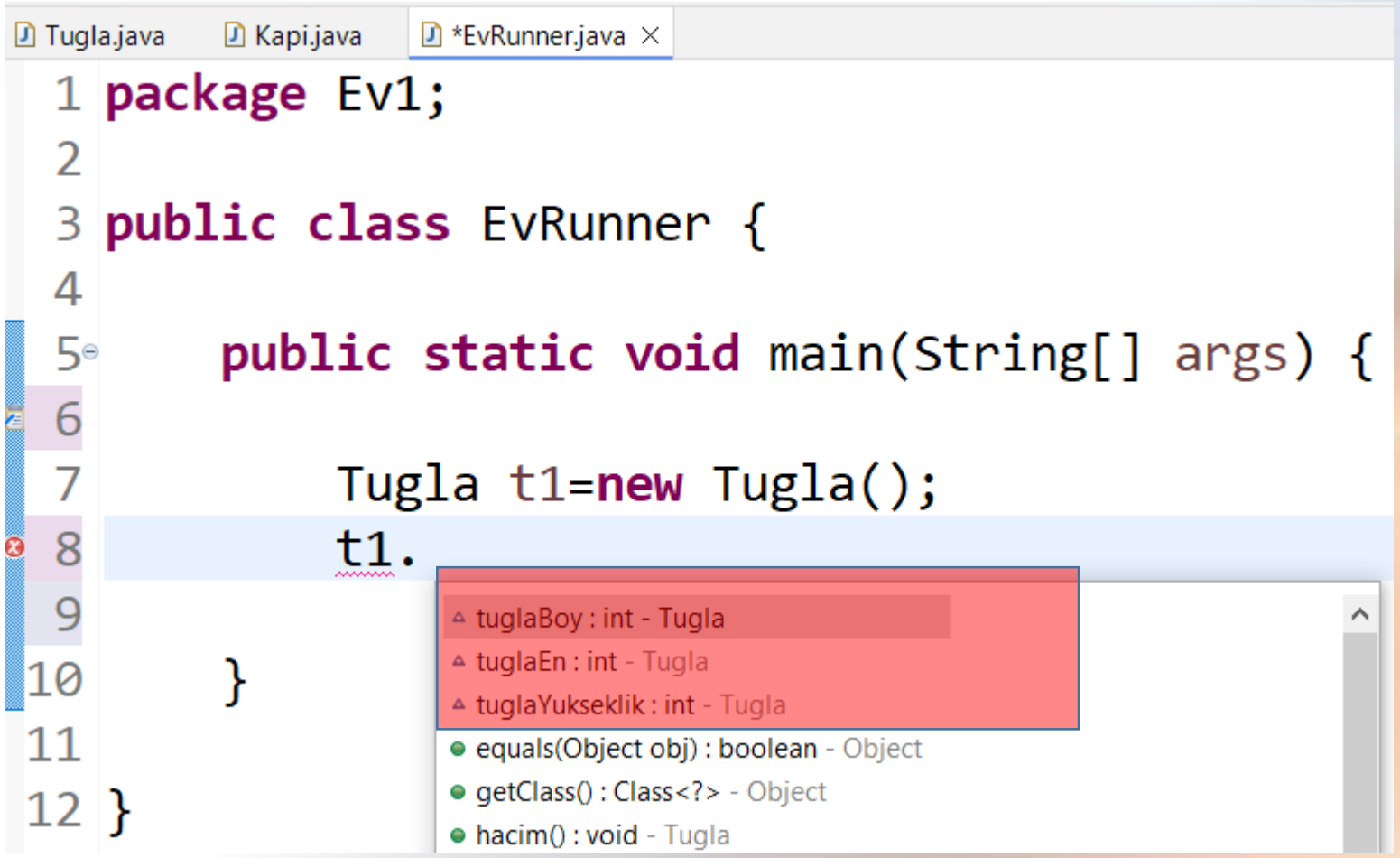
SINIFLAR

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.



Tugla.java | Kapi.java | *EvRunner.java

```java
package Ev1;

public class EvRunner {

    public static void main(String[] args) {

        Tugla t1=new Tugla();
        t1.hacim();  //eylemini çağırdık
        System.out.println(t1.tuglaBoy);//özellik çağırdık
        System.out.println(t1.tuglaYukseklik);
        System.out.println(t1.tuglaEn);

    }

}
```

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

```java
package Ev1;

public class Tugla {
    //variable-değişkenler-Sınıfın pasif özellikleri
    int tuglaEn=10;
    int tuglaBoy=5;
    int tuglaYukseklik=4;
    //metotlar-Sınıfın eylemlerini gerçekleştiren öğeler
    public void hacim () {
        int hacim=tuglaBoy*tuglaEn*tuglaYukseklik;
        System.out.println("Tuğlanın hacmi:"+hacim);
    }
}
```

Tugla.java | Kapi.java | EvRunner.java

```java
package Ev1;

public class EvRunner {

    public static void main(String[] args) {

        Tugla t1=new Tugla();
        t1.hacim();  //eylemini çağırdık
        System.out.println(t1.tuglaBoy);
        Tugla t2=new Tugla();
        t2.hacim();
        System.out.println(t2.tuglaBoy);
    }

}
```

Problems | Javadoc | Declaration | Console

<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

```java
package Ev1;

public class Tugla {
    //variable-değişkenler-Sınıfın pasif özellikleri
    int tuglaEn=10;
    int tuglaBoy=5;
    int tuglaYukseklik=4;
    //metotlar-Sınıfın eylemlerini gerçekleştiren öğeler
    public void hacim () {
        int hacim=tuglaBoy*tuglaEn*tuglaYukseklik;
        System.out.println("Tuğlanın hacmi:"+hacim);
    }
}
```

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 oluşturmamızı sağlayan bölümdür.
- Class oluşturduğumuzda Java bize otomatik olarak bir constructor verir.
- Bu constructor'lara "default constructor" denir. default constructor =

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 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

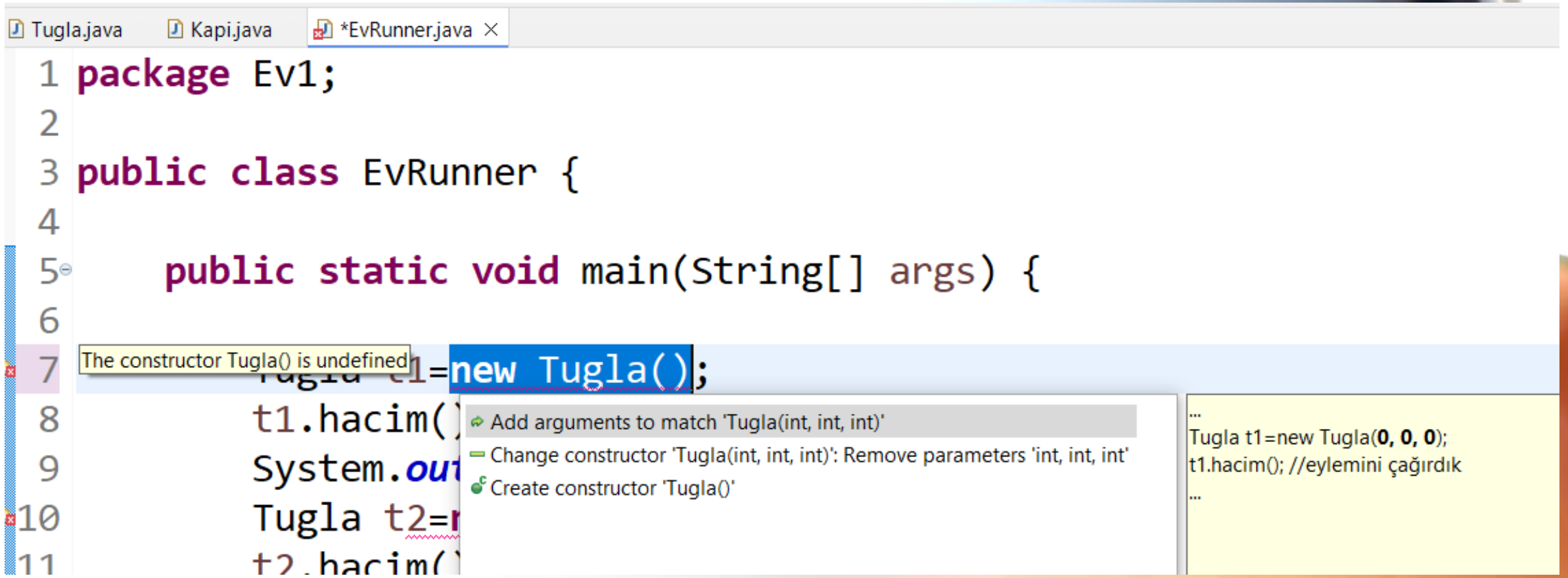
```
1  package Ev1;
2
3  public class EvRunner {
4
5      public static void main(String[] args) {
6
7          Tugla t1=new Tugla();
8
```

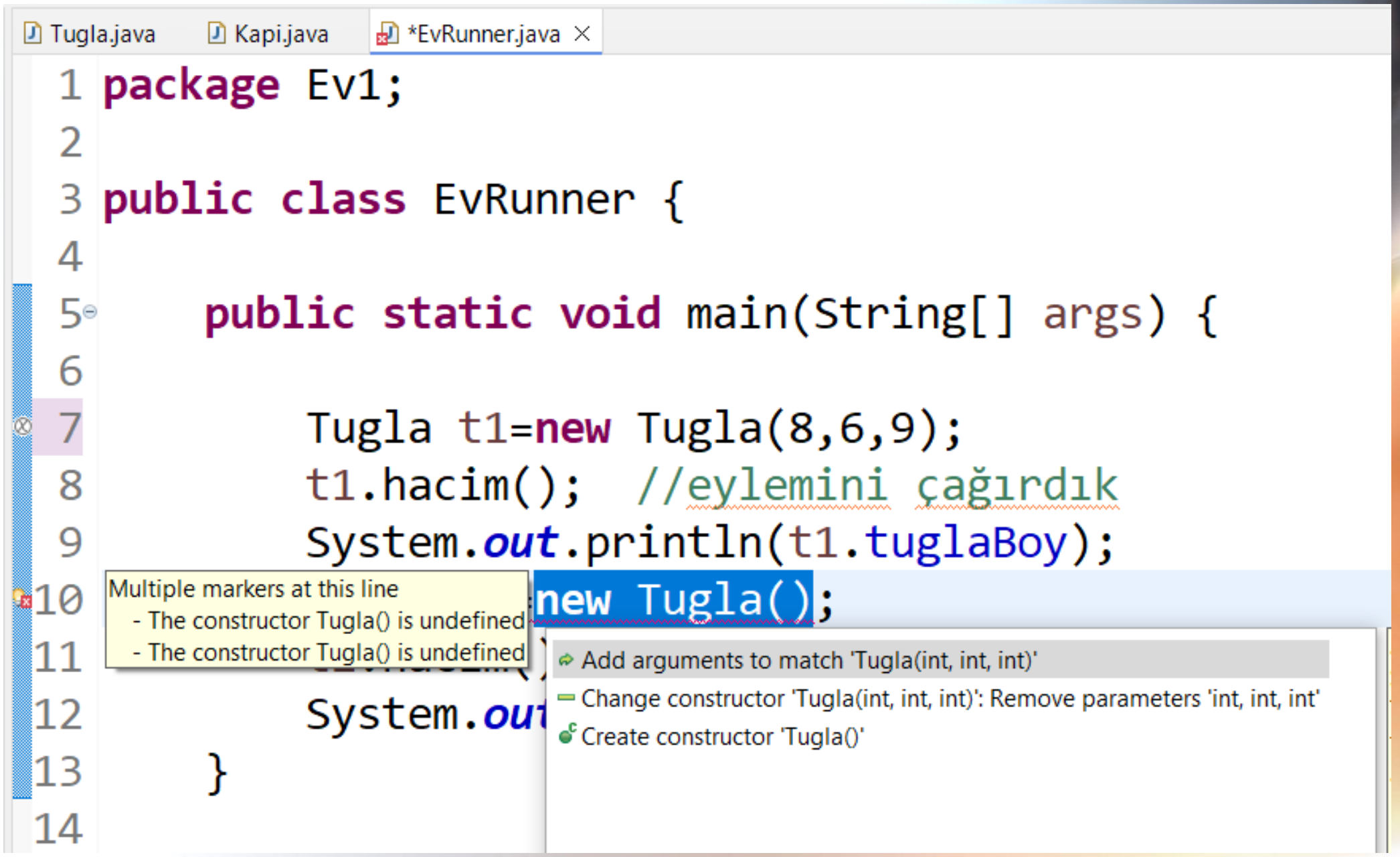
Constructor Nedir?

- 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** + () +

  {

  }


Tugla.java
Kapi.java
*EvRunner.java
1 package Ev1;
2
3 public class EvRunner {
4
5 public static void main(String[] args) {
6
7 The constructor Tugla() is undefined
new Tugla();
8 t1.hacim()
9 System.out
10 Tugla t2=n
Add arguments to match 'Tugla(int, int, int)'
Change constructor 'Tugla(int, int, int)': Remove parameters 'int, int, int'
Create constructor 'Tugla()'
...
Tugla t1=new Tugla(0, 0, 0);
t1.hacim(); //eylemini çağırdık
...


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(8,6,9);
8 t1.hacim(); //eylemini çağırdık
9 System.out.println(t1.tuglaBoy);
10 new Tugla();
11
12 System.out
13 }
14
Multiple markers at this line
- The constructor Tugla() is undefined
- The constructor Tugla() is undefined
Add arguments to match 'Tugla(int, int, int)'
Change constructor 'Tugla(int, int, int)': Remove parameters 'int, int, int'
Create constructor 'Tugla()'

Tugla.java | Kapi.java | EvRunner.java

```java
package Ev1;

public class EvRunner {

    public static void main(String[] args) {

        Tugla t1=new Tugla(8,6,9);
        t1.hacim();
        System.out.println(t1.uglaBoy);
        Tugla t2=new Tugla(5,4,2);
        t2.hacim();
        System.out.println(t2.uglaBoy);
    }

}
```

Tugla.java | Kapi.java | *EvRunner.java

```java
package Ev1;

public class Tugla {
    //variable-değişkenler-Sınıfın pasif özellikleri
    int uglaEn=10;
    int uglaBoy=5;
    int uglaYukseklik=4;
    //constructor
    public Tugla(int uglaEn, int uglaBoy, int uglaYukseklik) {
        this.uglaEn = uglaEn;
        this.uglaBoy = uglaBoy;
        this.uglaYukseklik = uglaYukseklik;
    }
    //metotlar-Sınıfın eylemlerini gerçekleştiren öğeler
        public void hacim () {
            int hacim=uglaBoy*uglaEn*uglaYukseklik;
            System.out.println("Tuğlanın hacmi:"+hacim);
        }
```

Problems | Javadoc | Declaration | Console

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

```
Tuğlanın hacmi:432
6
Tuğlanın hacmi:40
4
```

Tüm Özellikleri göndermeye gerek yok

*Tugla.java | 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 tuglaYukseklik=4;
7      //constructor1
8      public Tugla(int tuglaEn, int tuglaBoy, int tuglaYukseklik) {
9          this.tuglaEn = tuglaEn;
10         this.tuglaBoy = tuglaBoy;
11         this.tuglaYukseklik = tuglaYukseklik;
12     }
13     //constructor1
14     public Tugla(int tuglaEn, int tuglaBoy) {
15         this.tuglaEn = tuglaEn;
16         this.tuglaBoy = tuglaBoy;
17     }
```

Tüm Özellikleri gö

*Tugla.java × 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 tuglaYukseklik=4;
7      //constructor1
8      public Tugla(int tuglaEn, int tuglaBoy, int tuglaYukseklik
9          this.tuglaEn = tuglaEn;
10         this.tuglaBoy = tuglaBoy;
11         this.tuglaYukseklik = tuglaYukseklik;
12     }
13     //constructor1
14     public Tugla(int tuglaEn, int tuglaBoy) {
15         this.tuglaEn = tuglaEn;
16         this.tuglaBoy = tuglaBoy;
17     }
```

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(8,6,9);
8          t1.hacim();
9          System.out.println(t1.tuglaBoy);
10         Tugla t2=new Tugla(5,4);
11         t2.hacim();
12         System.out.println(t2.tuglaBoy);
13     }
14
15 }
```

Problems Javadoc Declaration Console ×

<terminated> EvRunner [Java Application] C:\Users\nmac

```
Tuğlanın hacmi:432
6
Tuğlanın hacmi:80
4
```

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, sinif 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.

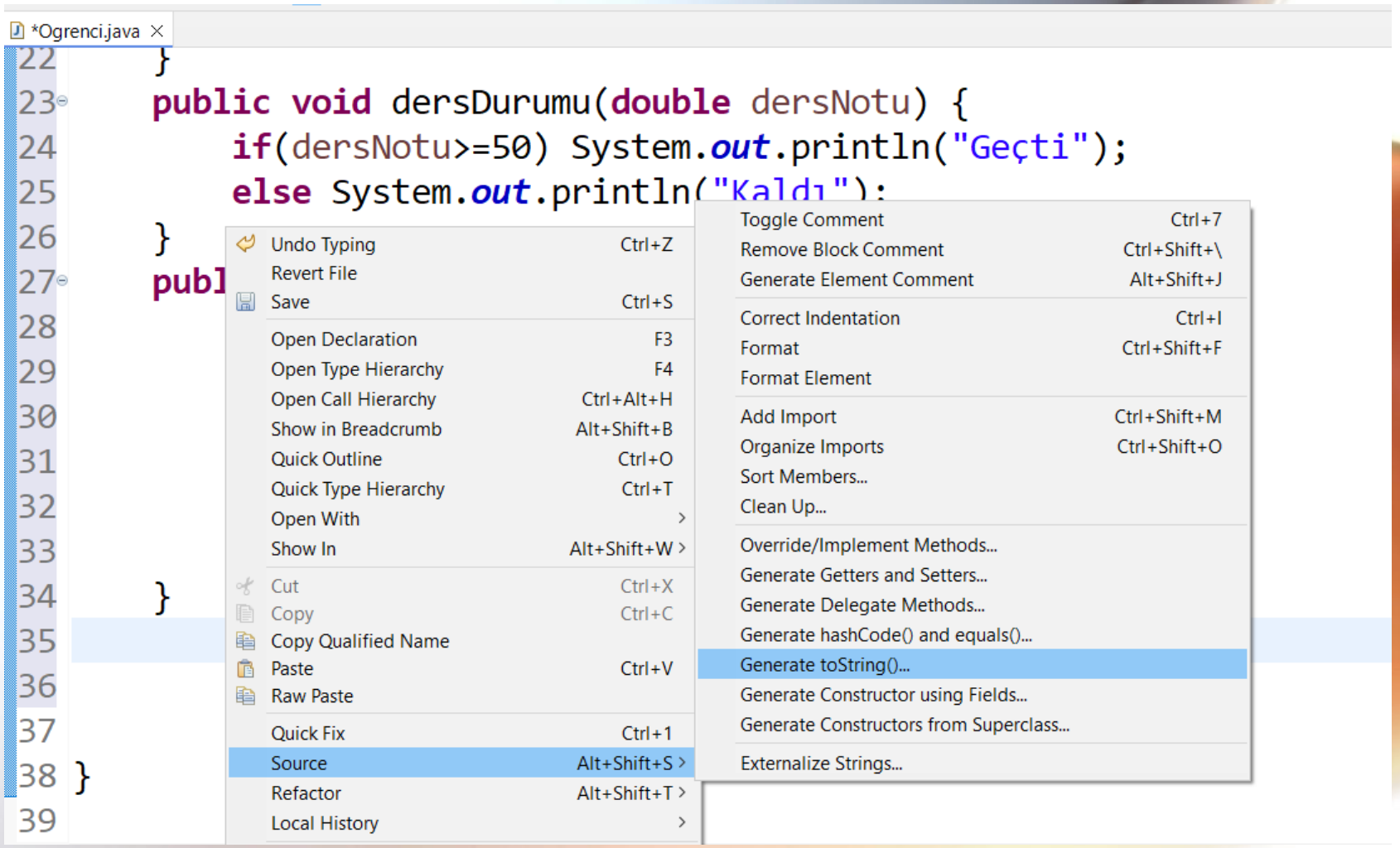
1.Main Sınıfı:

•main metodunda, her iki constructor'ı da kullanarak en az iki Ogrenci 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,
   "10987654321", "Matematik", 75.5);
8          System.out.println(o2.toString());
9          System.out.println(o1.toString());
10
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

```
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

```
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

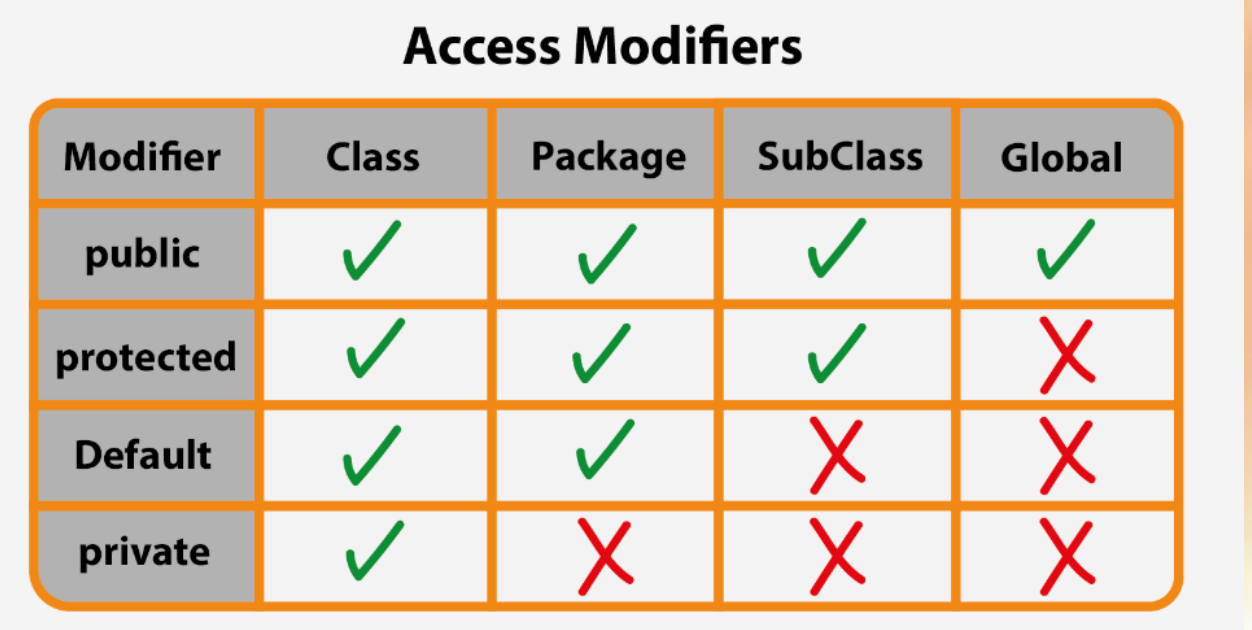
==> **default** olanlar başka package'lardan kullanılamazlar
==> **private** olanlar sadece oluşturuldukları class içinden kullanılabilirler. **(Package-Private)**

Access Modifiers

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

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 paketteki diğer sınıflar ve o sınıftan türetilen alt sınıflar tarafından erişilebilir olduğunu ifade eder.



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	✓	✗	✗	✗

AccessModifiers1.java

```java
package hafta3_;

public class AccessModifiers1 {
    public String publicDegisken="Kütüphane gibi";
    private String privateDegisken="kasa gibi";
    protected String protectedDegisken="albüm gibi";
    String defaultDegisken="apartman gibi";
}
```

AccessModifiers1.java | AccessModifierGlobal.java | AccessModifiersRunner.java | AccessModifierSubclass.java

```java
package hafta2_PassByValue_Classes_Constructors;

import hafta3_.AccessModifiers1;

public class AccessModifierGlobal {
    public static void main(String[] args) {
        AccessModifiers1 a1=new AccessModifiers1();
        System.out.println(a1.publicDegisken);
        System.out.println(a1.privateDegisken);
        System.out.println(a1.protectedDegisken);
        System.out.println(a1.defaultDegisken);
    }

}
```

Aynı Proje içindeler. GLOBAL

Access Modifiers

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

AccessModifiers1.java

```java
package hafta3_;

public class AccessModifiers1 {
    public String publicDegisken="Kütüphane gibi";
    private String privateDegisken="kasa gibi";
    protected String protectedDegisken="albüm gibi";
    String defaultDegisken="apartman gibi";
}
```

AccessModifiers1.java | AccessModifierGlobal.java | AccessModifiersRunner.java | AccessModifierSubclass.java

```java
package hafta3_;

public class AccessModifiersRunner {
    public static void main(String[] args) {
        AccessModifiers1 a1=new AccessModifiers1();
        System.out.println(a1.publicDegisken);
        System.out.println(a1.privateDegisken);
        System.out.println(a1.protectedDegisken);
        System.out.println(a1.defaultDegisken);
    }
}
```

Aynı Paket içindeler

Access Modifiers

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

AccessModifiers1.java

```java
package hafta3_;

public class AccessModifiers1 {
    public String publicDegisken="Kütüphane gibi";
    private String privateDegisken="kasa gibi";
    protected String protectedDegisken="albüm gibi";
    String defaultDegisken="apartman gibi";
}
```

AccessModifiers1.java | AccessModifierGlobal.java | AccessModifiersRunner.java | AccessModifierSubclass.java

```java
package hafta3_;

public class AccessModifierSubclass extends AccessModifiers1 {

    public static void main(String[] args) {
        AccessModifiers1 a1=new AccessModifiers1();
        System.out.println(a1.publicDegisken);
        System.out.println(a1.privateDegisken);
        System.out.println(a1.protectedDegisken);
        System.out.println(a1.defaultDegisken);
    }

```

Aynı Paket içindeler. SubClass ı

Access Modifiers

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

AccessModifiers1.java

```java
package hafta3_;

public class AccessModifiers1 {
    public String publicDegisken="Kütüphane gibi";
    private String privateDegisken="kasa gibi";
    protected String protectedDegisken="albüm gibi";
    String defaultDegisken="apartman gibi";
}
```

AccessModifiers1.java | *AccessModifierGlobal.java | AccessModifiersRunner.java | *AccessModifierSubclass.java

```java
package hafta2_PassByValue_Classes_Constructors;

import hafta3_.AccessModifiers1;

public class AccessModifierGlobal extends AccessModifiers1
    public static void main(String[] args) {
        AccessModifiers1 a1=new AccessModifiers1();
        System.out.println(a1.publicDegisken);
        System.out.println(a1.privateDegisken);
        System.out.println(a1.protectedDegisken);
        System.out.println(a1.
    }

}
```

The field AccessModifiers1.protectedDegisken is not visible

2 quick fixes available:

- Change visibility of 'protectedDegisken' to 'public'
- Create getter and setter for 'protectedDegisken'...

Press 'F2' for focus

Farklı paketteler Subclass

Access Modifiers

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

STATIC

-Bir değişkenin veya bir method'un Class'a ait olduğunu
-Class'in bir örneği (Object'i) oluşturulmadan kullanılabileceğini belirtmek için kullanılan bir anahtar kelimedir.

STATIC

-static class member'lara ulaşmak için object oluşturmaya gerek duyulmaz,
-non-static class member'lara ulaşmak için object oluşturmak şarttır.

```
1  package hafta3_;
2
3  public class StaticClassRunner {
4      public static void main(String[] args) {
5
6          System.out.println(StaticClass.dolap);
7          StaticClass.staticMethod(); //static metod
8
9          StaticClass s1=new StaticClass();
10         s1.nonStaticMethod(); //non-static metod
11     }
12 }
13
```

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.

```
1  package hafta3_;
2
3  public class StaticClassRunner {
4      public static void main(String[] args) {
5          //non static veriler için
6          StaticClass s1=new StaticClass();
7          s1.tencere=12;
8
9          StaticClass s2=new StaticClass();
10         s2.tencere=7;
11
12         System.out.println(s1.tencere);
13         System.out.println(s2.tencere);
14
15     }
16 }
```

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.

```
1  package hafta3_;
2
3  public class StaticClassRunner {
4      public static void main(String[] args) {
5          //static veriler için
6          StaticClass s1=new StaticClass();
7          s1.dolap=12;
8
9          StaticClass s2=new StaticClass();
10         s2.dolap=7;
11
12         System.out.println(s1.dolap);
13         System.out.println(s2.dolap);
14
15     }
16 }
17
```

STATIC

*static class member'lar class'a,
*non-static class member'lar object'lere aittir

static variable**= class variable**
non-static variable**= instance variable**
= object 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

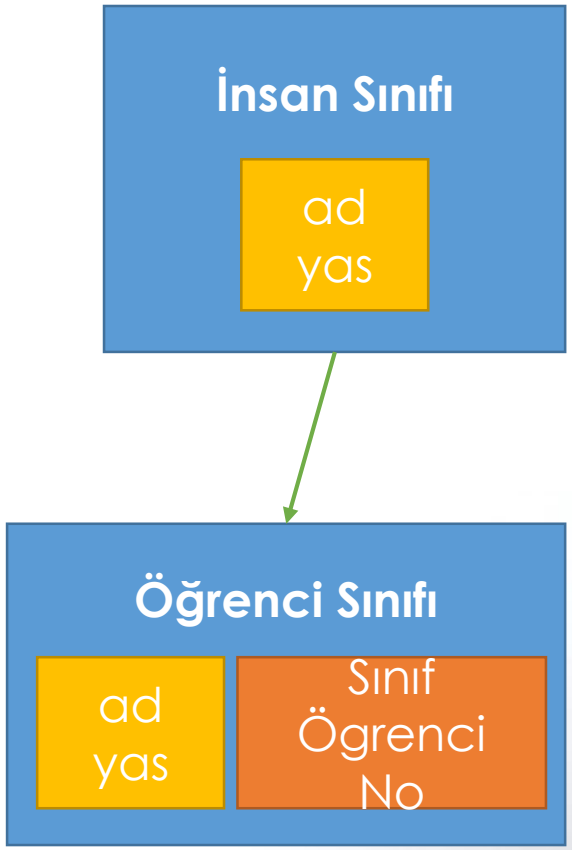
1) Object oluşturma bir Class içerisinde toplanır ve tüm projelerde kullanılabilirliğe olanak sağlar.(String gibi)
2) Class'larin 1 kez oluşturulması sayesinde uzun kodları tekrardan yazmak yerine kısa kodlamalar
ile çalıştırılabilir.
3) Uzun kodların tekrar yazılmasının engellenmesi sayesinde geliştirme süreci kısalır.
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 yapılıp 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)

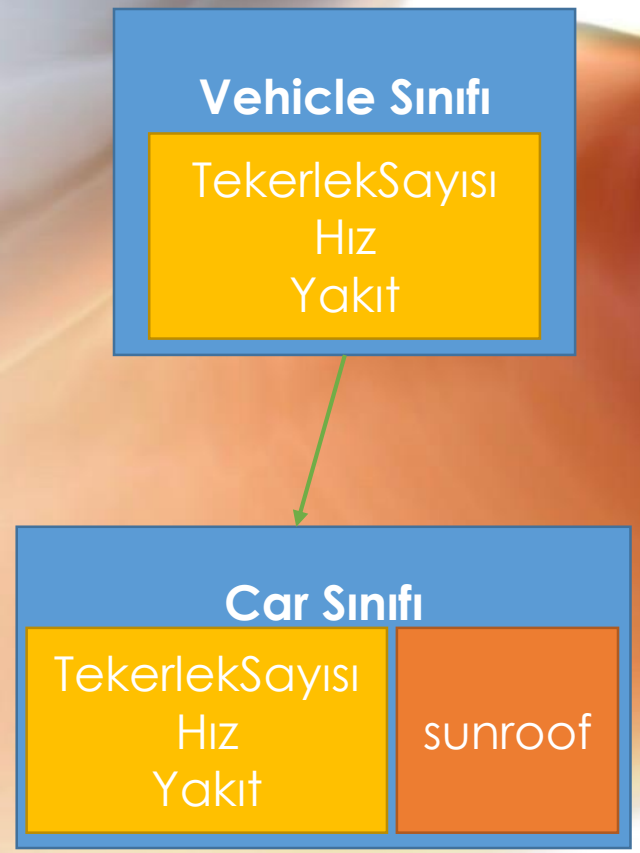
•Miras sayesinde, bir sınıf içinde tanımlanan özellikler ve yöntemler (metodlar), başka sınıflar tarafından tekrar kullanılabilir. Yeni bir sınıf yazarken sıfırdan başlamak yerine mevcut bir sınıfın özelliklerini ve davranışlarını miras alabilirsiniz.

•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

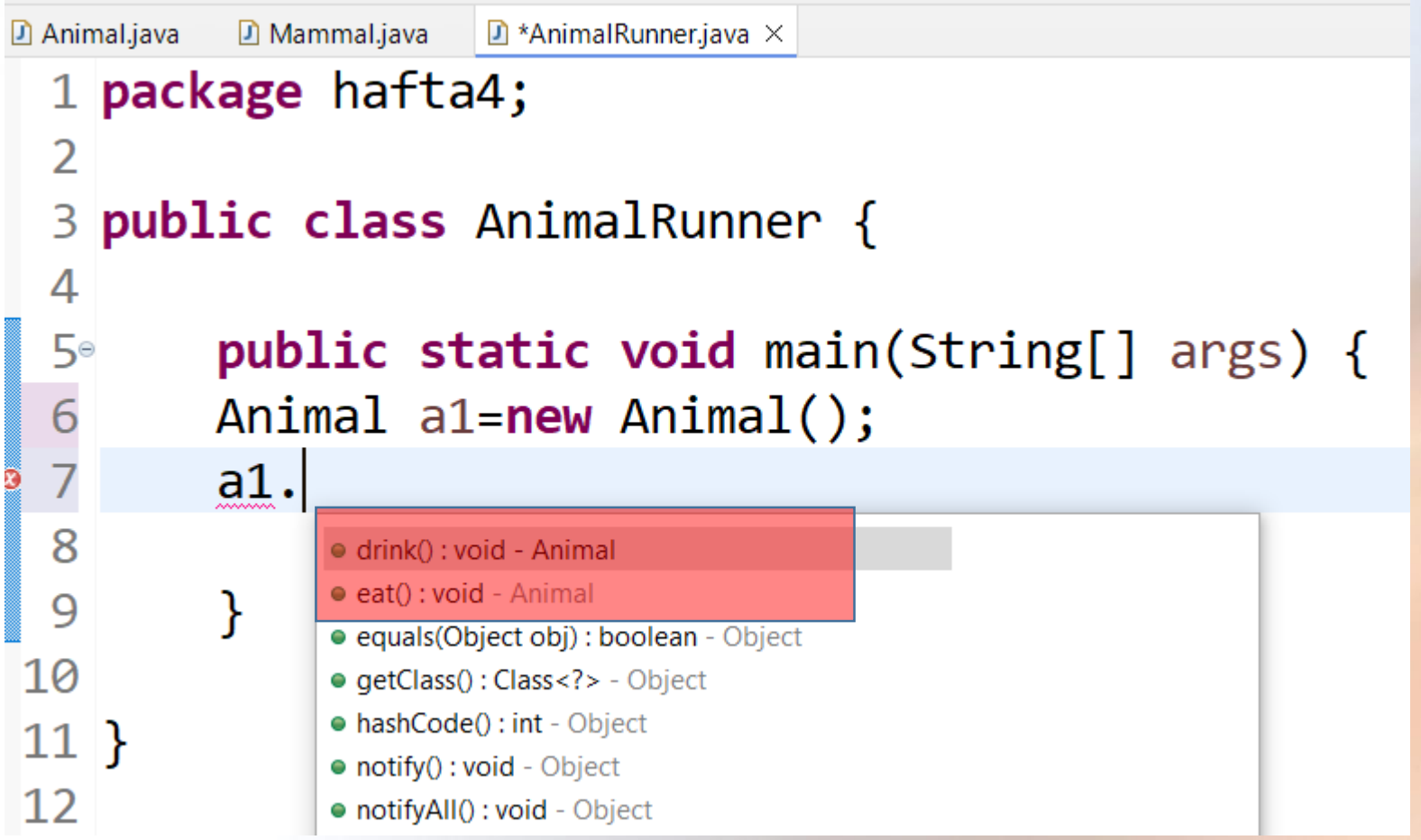
Animal adında bir sınıf oluşturalım.

*Animal.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 }
```

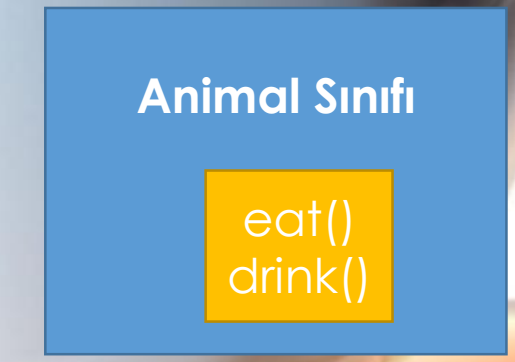
INHERITANCE

AnimalRunner adında bir sınıf oluşturalım.



INHERITANCE

Mammal adında bir sınıf oluşturalım.

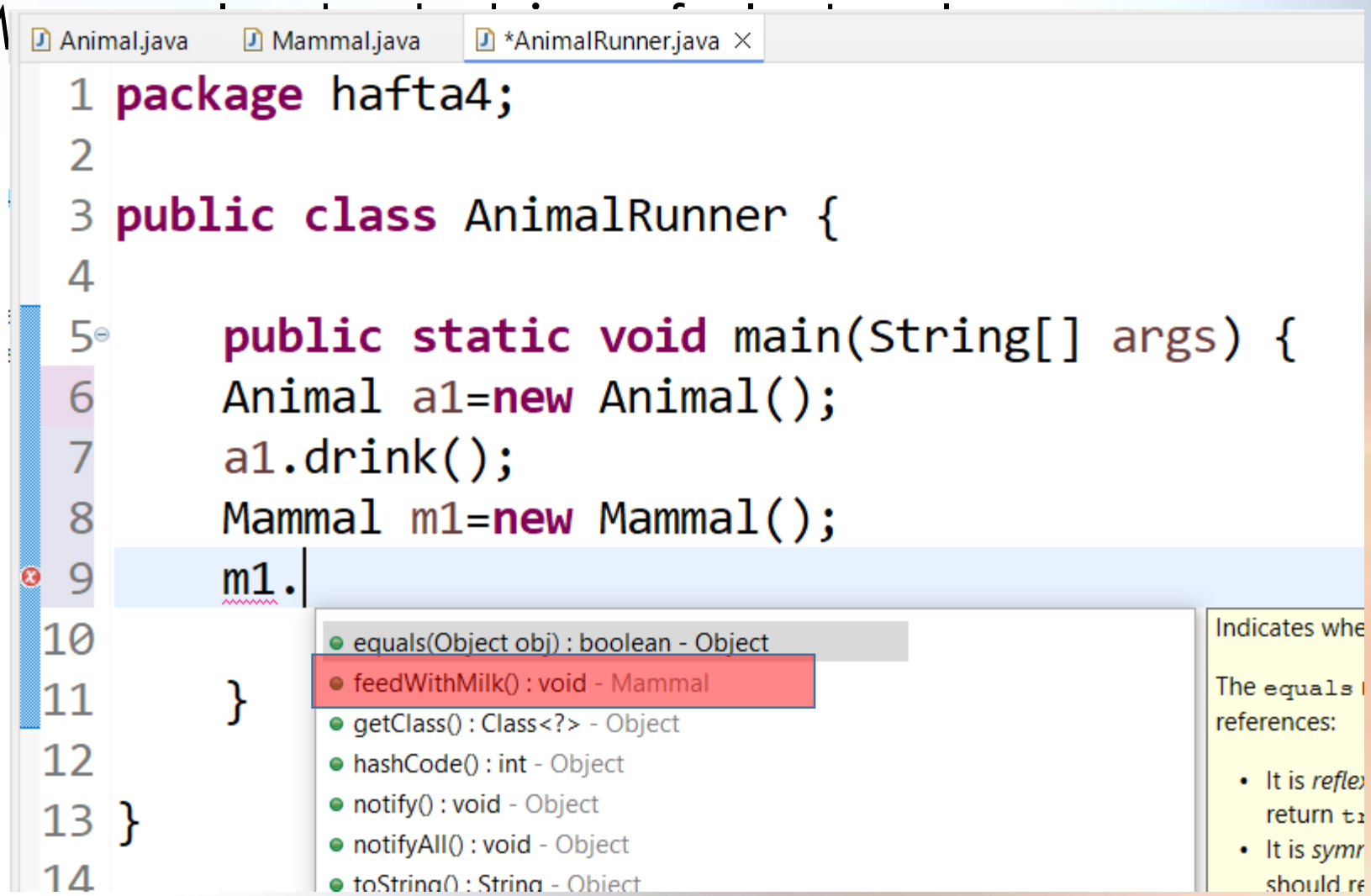


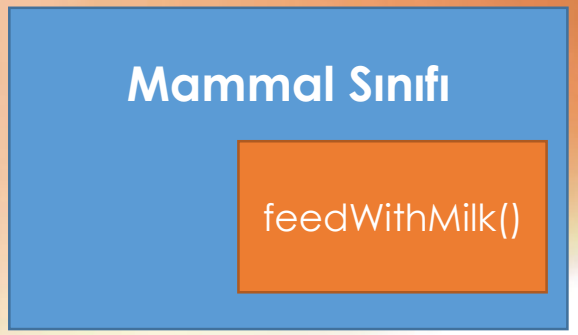
*Animal.java Mammal.java

```
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
```

INHERITANCE

M





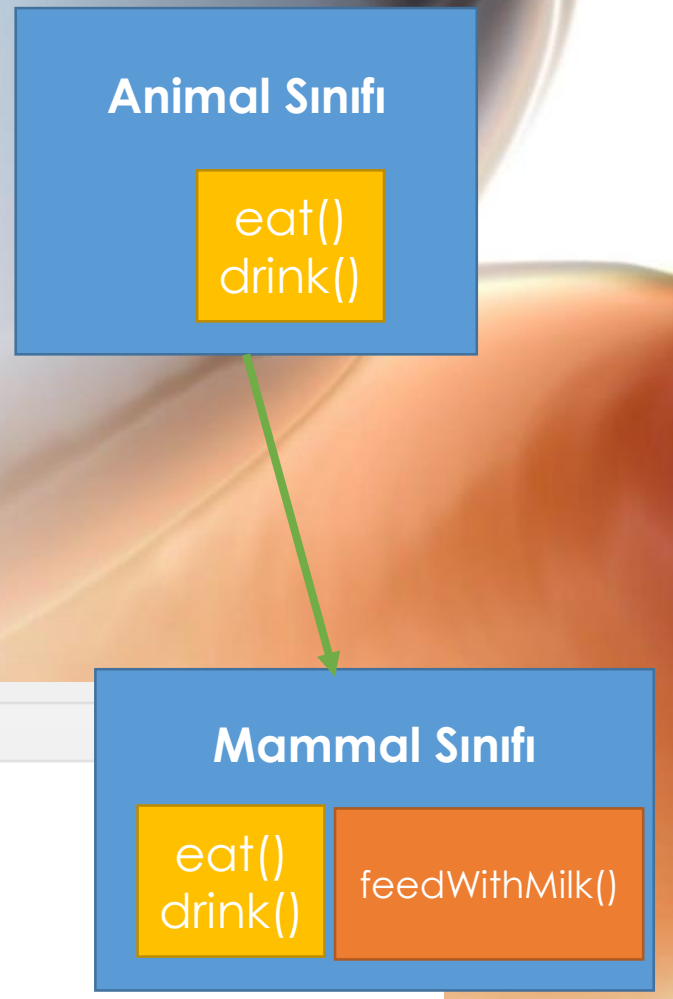
INHERITANCE

Extends ile parent ini belirliyoruz.

*Animal.java *Mammal.java

```
1 package hafta4;
2
3 public class Mammal extends An{
4     public void feedWithMilk(){
5         System.out.println("Mar
6     }
7 }
8
```

Animal - hafta4
Annotation - java.text
AnnotationFormatError - java
AnnotationTypeMismatchExc
AncestorEvent - javax.swing.e
EnumConstantNotPresentExce
IncompatibleClassChangeErro
InstantiationError - java.lang



*Animal.java *Mammal.java

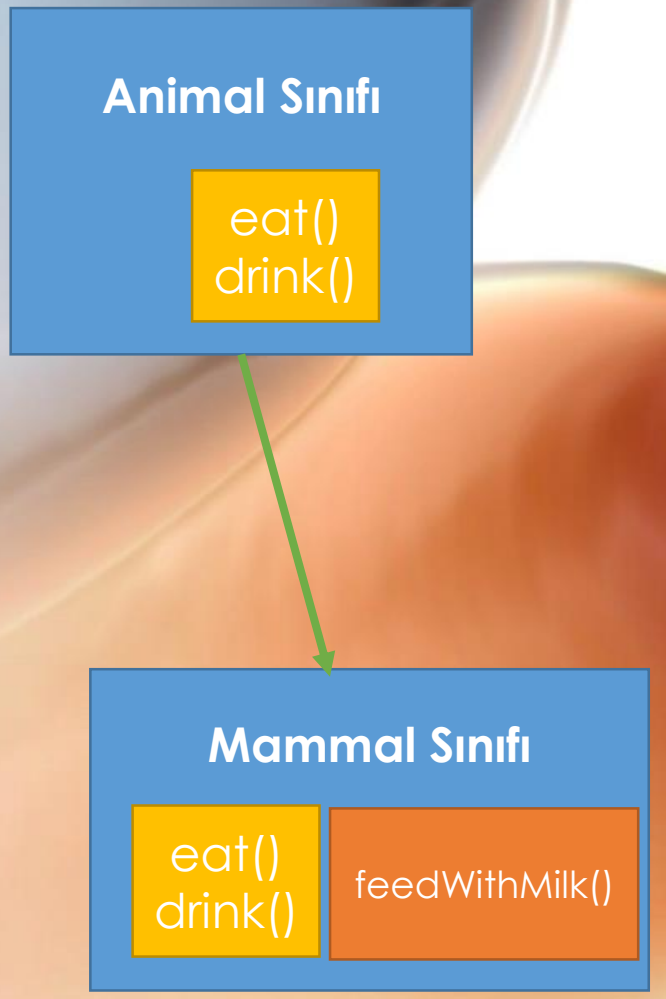
```
1 package hafta4;
2
3 public class Mammal extends Animal{
4     public void feedWithMilk(){
5     System.out.println("Mammals feed their babies with milk");
6     }
7 }
```

INHERITANCE

Animal.java | Mammal.java | *AnimalRunner.java

```
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

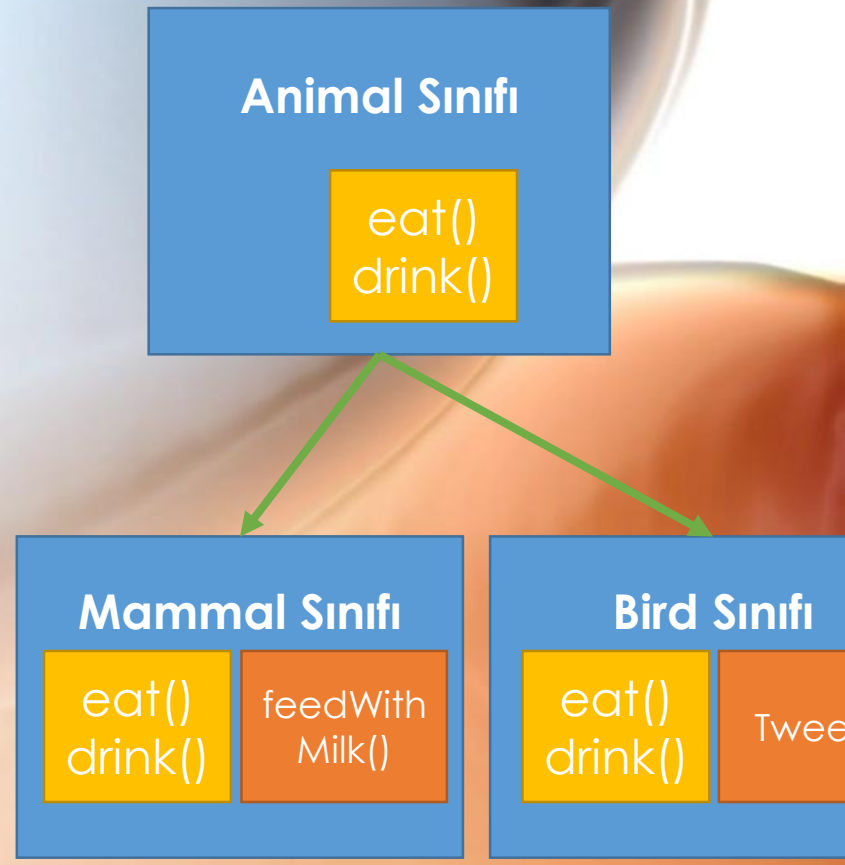


INHERITANCE

Bir sınıf birden fazla sınıfa parent olabilir.

Animal.java | Mammal.java | AnimalRunner.java | Bird.java

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

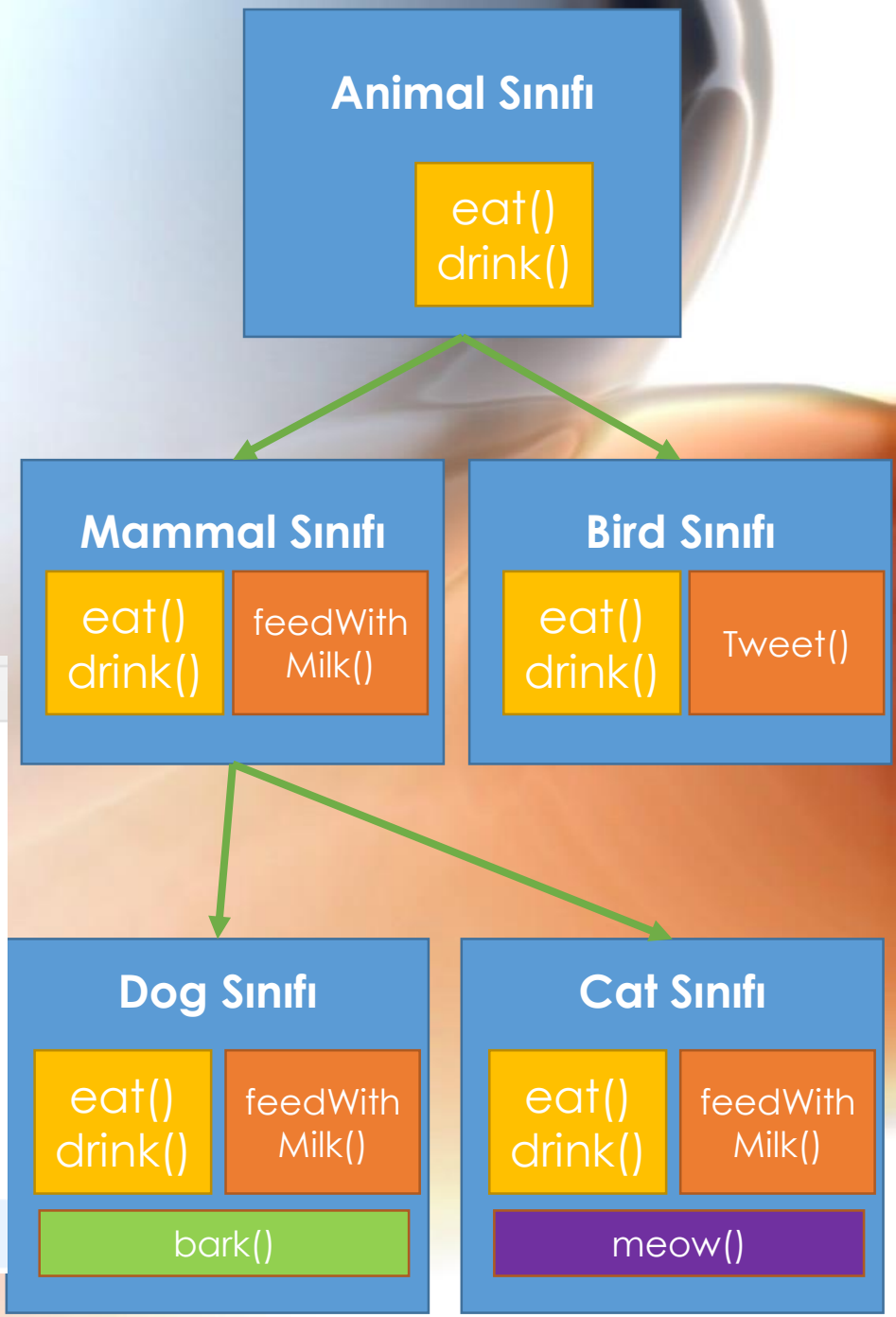


Animal.java | Mammal.java | *AnimalRunner.java | Bird.java | Cat.java | *Dog.java

```java
package hafta4;

public class Cat extends Mammal {
	public void meow(){
		System.out.println("cats meow...");
	}
}

```

Animal.java | Mammal.java | *AnimalRunne | Next Annotation (Ctrl+.) | Cat.java | Dog.java

```java
package hafta4;

public class Dog extends Mammal {
	public void bark(){
		System.out.println("dogs bark..");
	}
}

```



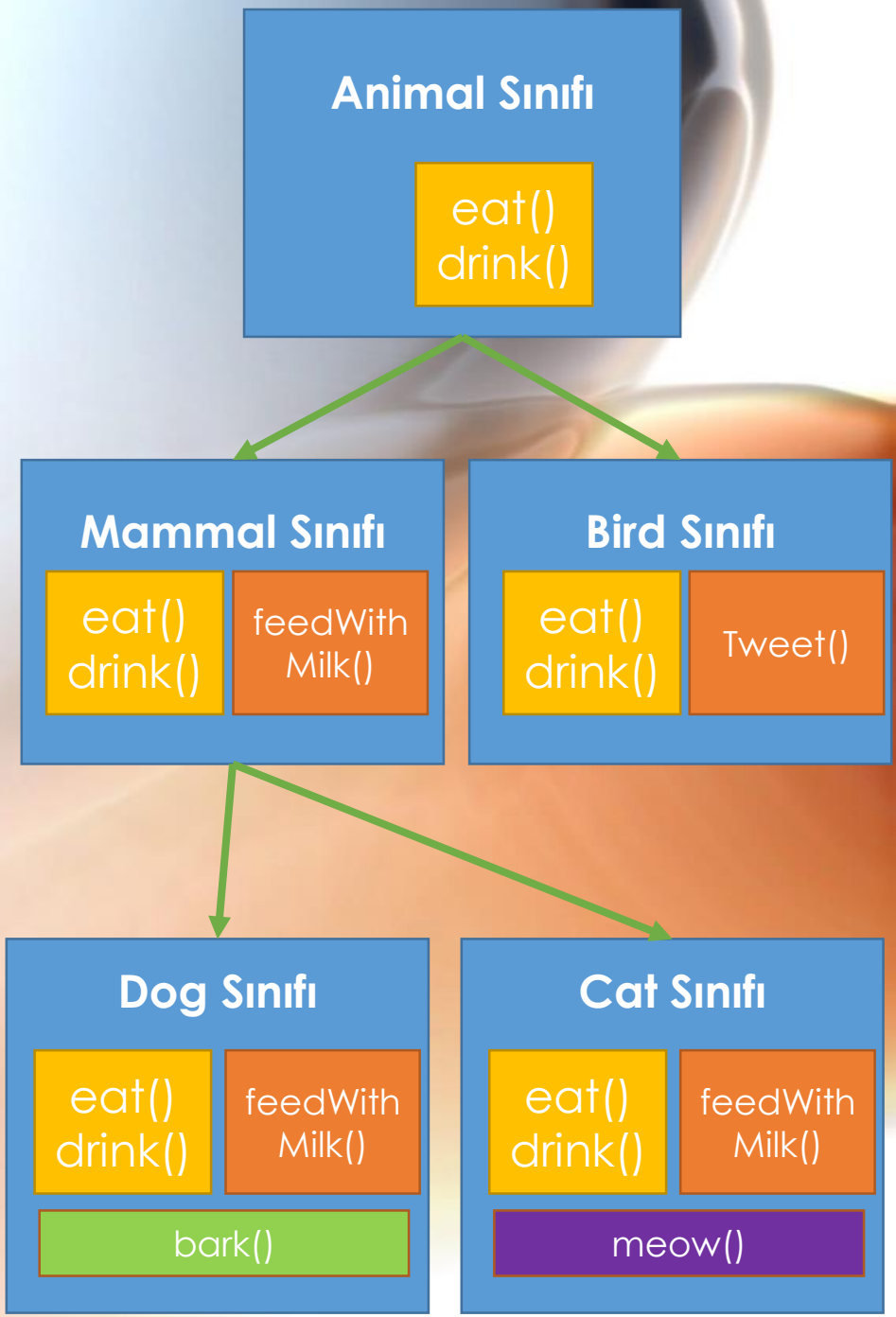
Animal.java | Mammal.java | *AnimalRunner.java | Bird.java

```
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
```

Problems | Javadoc

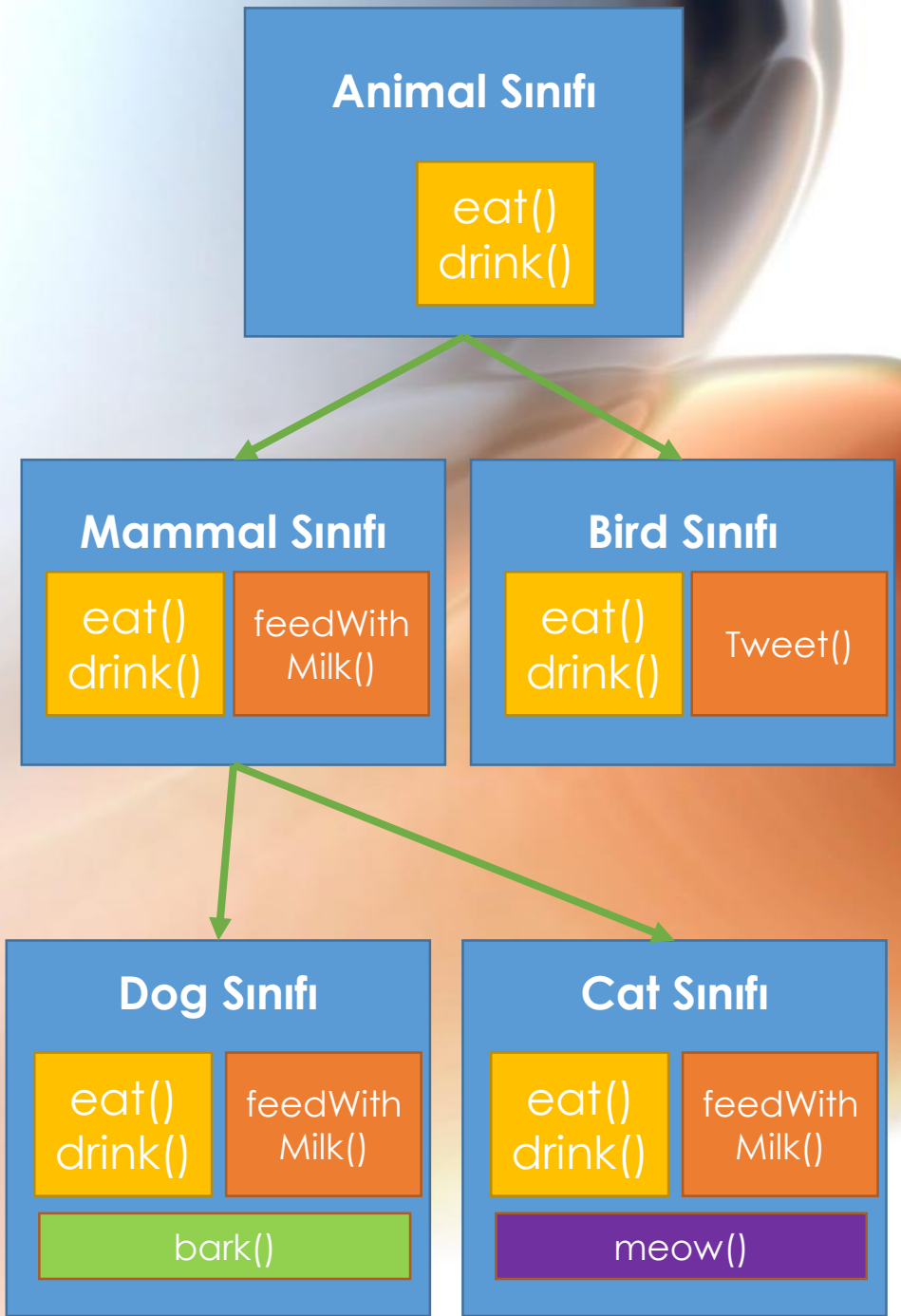
nt olabilir.



INHERITANCE

Java multiple inheritance desteklemez.
Java **SINGLE INHERITANCE** i destekler .

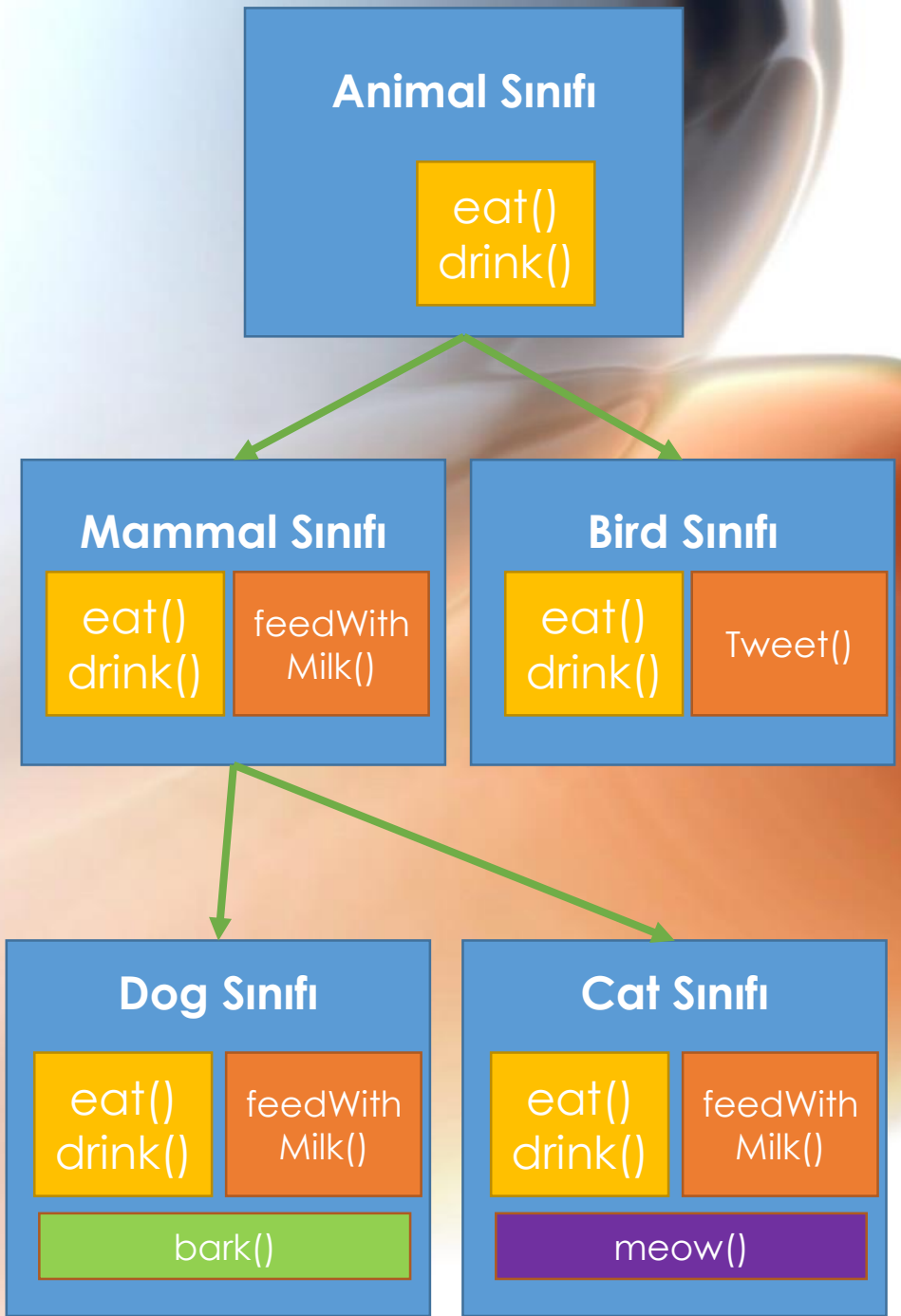
Bir Child in Bir Parent ı olur.



INHERITANCE

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

HIERARCHICAL INHERITANCE

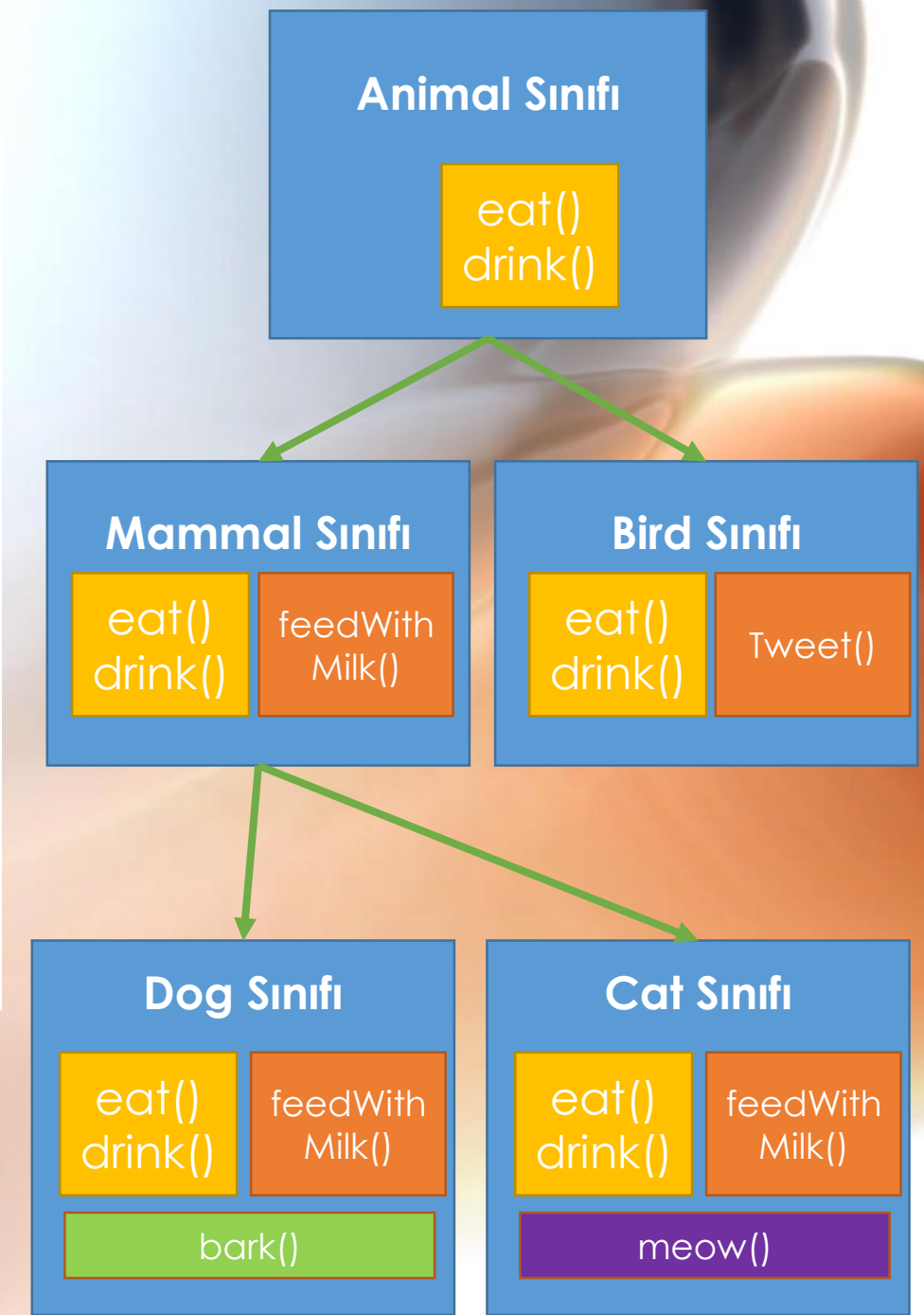


INHERITANCE

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

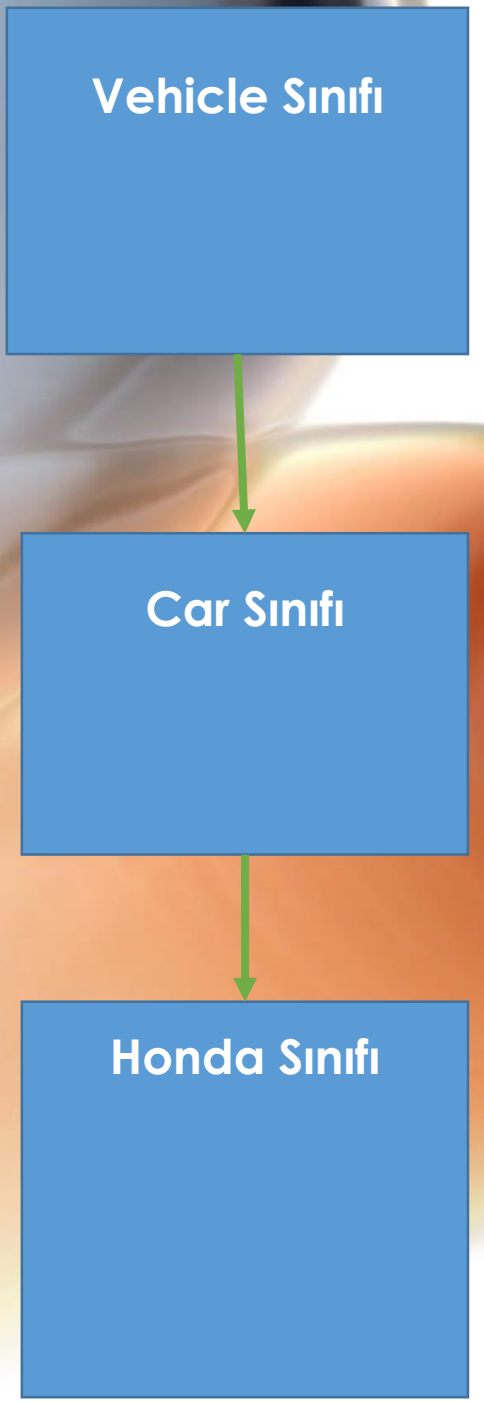
Buna çözüm olarak apartman tarzi alt alta yapılıan inheritance bize çözüm olacak.

MULTI LEVEL INHERITANCE



CONSTRUCTOR HIYERARŞİSİ

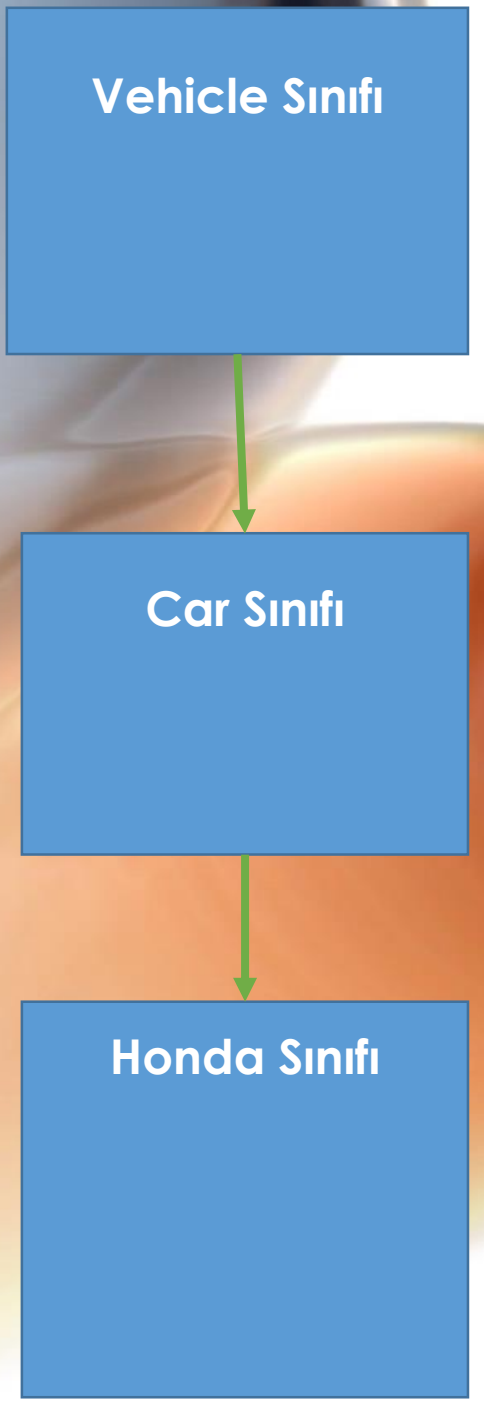
Child class'tan bir object oluşturduğunuzda constructor'lar en üstteki parent class'tan başlatılarak alta doğru çalıştırılır.
//çağrı aşağıdan yukarı doğru olurken
//contructorlar yukarıdan aşağı iner



CONSTRUCTOR HIYERARŞİSİ

Vehicle.java × Car.java ×

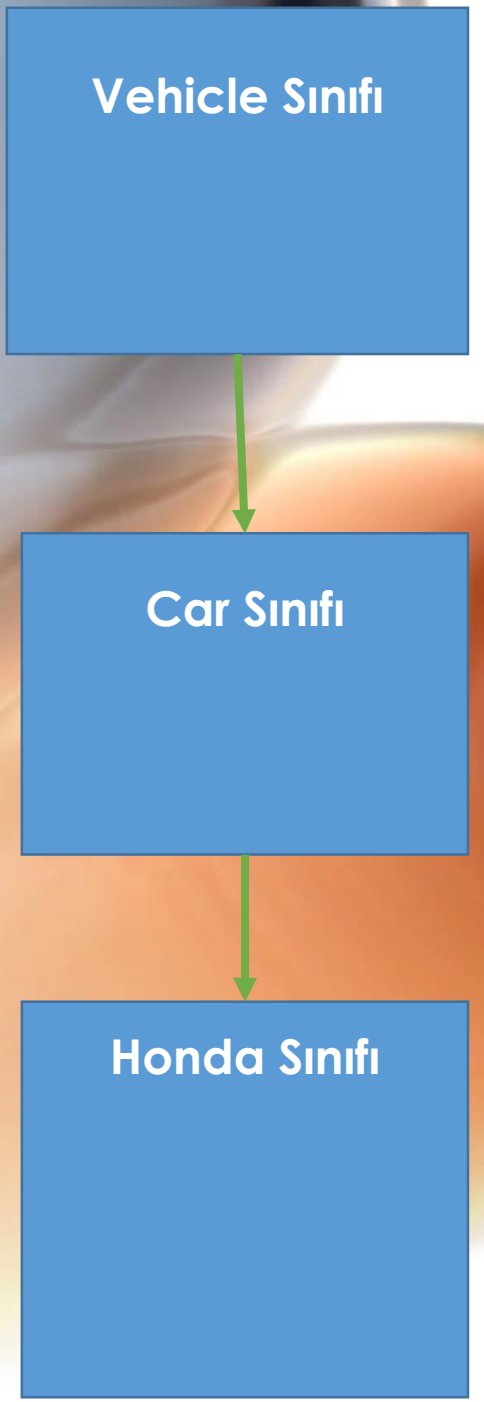
```
1 package hafta4;
2
3 public class Vehicle {
4         public Vehicle() {
5             System.out.println("Honda,2023,benzin");
6         }
7 }
8
```



CONSTRUCTOR HIYERARŞİSİ

Vehicle.java | Car.java | Honda.java | *HondaRunner.java

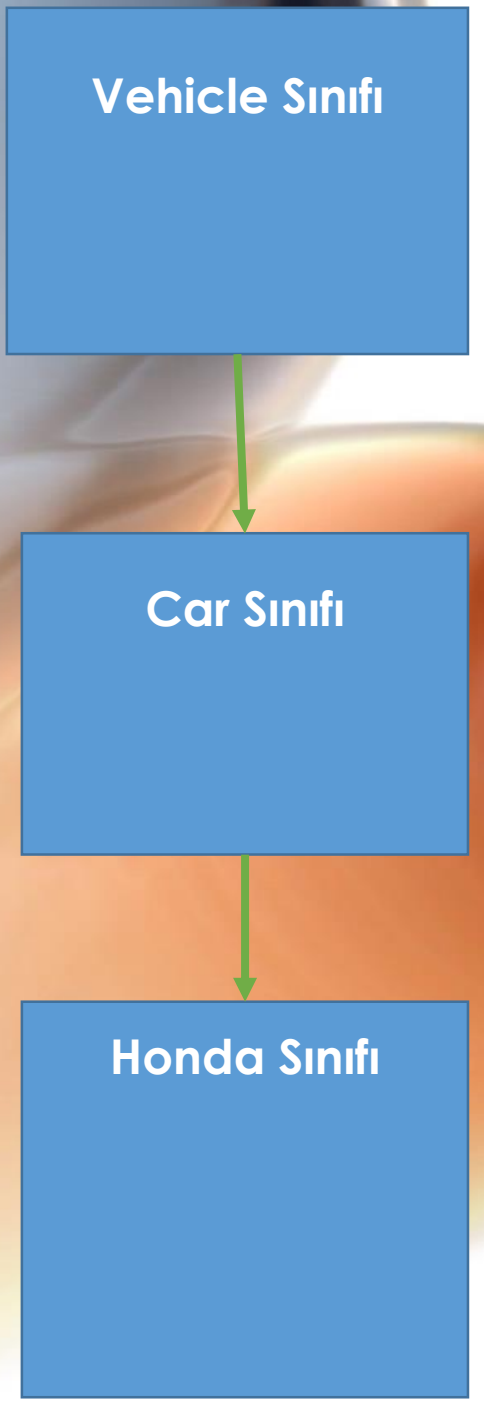
```
1 package hafta4;
2 public class Car extends Vehicle{
3     public Car() {
4         System.out.println("sedan");
5     }
6 }
```



CONSTRUCTOR HIYERARŞİSİ

Vehicle.java | Car.java | Honda.java | *HondaRunner.java

```
1 package hafta4;
2 public class Honda extends Car {
3       public Honda() {
4               System.out.println("civic");
5           }
6 }
7
```



CONSTRUCTOR HIYERARŞİSİ

Vehicle.java | Car.java | Honda.java | *HondaRunner.java

```java
package hafta4;

public class HondaRunner {

    public static void main(String[] args) {
        Honda h=new Honda();
        }
}
```

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

Vehicle Sınıfı

Vehicle.java | Car.java

```java
package hafta4;

public class Vehicle {
        public Vehicle() {
            System.out.println("Honda,2023,benzin");
        }
}

```

Car Sınıfı

Vehicle.java | Car.java | Honda.java | *HondaRunner.java

```java
package hafta4;
public class Car extends Vehicle{
    public Car() {
        System.out.println("sedan");
    }
}
```

Honda Sınıfı

Vehicle.java | Car.java | Honda.java | *HondaRunner.java

```java
package hafta4;
public class Honda extends Car {
    public Honda() {
        System.out.println("civic");
    }
}

```

CONSTRUCTOR HIYERARŞİSİ

Vehicle.java | Car.java | Honda.java | *HondaRunner.java

```
1 package hafta4;
2
3 public class HondaRunner {
4
5     public static void main(String[] args) {
6         Honda h=new Honda();
7         }
8 }
```

//çağrı aşağı
//contructorl

Problems | Javadoc | Declaration | Console | Terminal

<terminated> HondaRunner [Java Application] C:\Users\nmacit\.p2\pool\p

```
Honda,2023,benzin
sedan
civic
```

Vehicle Sınıfı

Vehicle.java | Car.java

```
1 package hafta4;
2
3 public class Vehicle {
4         public Vehicle() {
5             System.out.println("Honda,2023,benzin");
6         }
7 }
8
```

Car Sınıfı

Vehicle.java | Car.java | Honda.java | *HondaRunner.java

```
1 package hafta4;
2 public class Car extends Vehicle{
3     public Car() {
4         System.out.println("sedan");
5     }
6 }
```

Honda Sınıfı

cle.java | Car.java | Honda.java | *HondaRunner.java

```
package hafta4;
public class Honda extends Car {
    public Honda() {
        System.out.println("civic");
    }
}
```

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

ileriNesneTabanliProgramlama_Guz_24_25/src/hafta4/InstanceBlokRunner.java

```
1  package hafta4;
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

InstanceBlokClass.java | InstanceBlokRunner.java

```
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 }
```

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 }
```

Problems | Javadoc | Declaration | Console

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

```
selam
selam
```

*InstanceBlokClass.java | InstanceBlokRunner.java | *InstanceBlokClass1.java

```
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.
//(Bunlar ortak özellikler. Diğerleri sonra ayarlansın)
//1. contructor hiç parametre almasın. Arabaya ait bilgileri ekrana yazdırsın
//2. contructor 2 parametre alsın. Bilgileri ekrana yazdırsın.

STATİC 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.

INSTANCE BLOK

//static bloklar içinde yerel değişken tanımlanmaz, ancak değer atanabilir.
//yerel değişkenler bu blok içinde tanımlanamaz

StaticBlokClass.java

```
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 }
```

STATİC BLOK dezavantajları

//programın çalışma zamanı uzayabilir, ancak sonradan hızlı yol alınabilir.
//statik bloklarda yapılan hata programın içerisinde beklenmedik sonuçlara yol açabilir.

STATİC BLOK

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

STATİC 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

```
Problems  Javadoc  Declaration  Console  Terminal
<terminated> HondaRunner [Java Application] C:\Users\nmacit\.p2\pool\p
Honda,2023,benzin
sedan
civic
```

Vehicle.java | Car.java | Honda.java | *HondaRunn

```java
package hafta4;

public class HondaRunner {

    public static void main(String[] args) {
        Honda h=new Honda();
        }
}
```

Vehicle.java | Car.java

```java
package hafta4;

public class Vehicle {
        public Vehicle() {
            System.out.println("Honda,2023,benzin");
        }
}

```

Vehicle.java | Car.java | Honda.java | *HondaRunner.java

```java
package hafta4;
public class Car extends Vehicle{
    public Car() {
        System.out.println("sedan");
    }
}
```

Vehicle.java | Car.java | Honda.java | *HondaRunner.java

```java
package hafta4;
public class Honda extends Car {
    public Honda() {
        System.out.println("civic");
    }
}

```

bir child class'ta bir object oluşturulduğunda, otomatik olarak çağrılır.
Ancak, super() çağrısını manuel olarak da çağırabilirsiniz.

Super(); Çağrısı

Ancak, super() çağrısını manuel olarak da çağırabilirsiniz.

*super() ifadesini, child class'ın constructor'unun başında kullanabilirsiniz.

*Manuel olarak yazarsanız parametreli constructor'lari da çağırabilirsiniz.

Neden Super(); Çağrısı Yapalım?

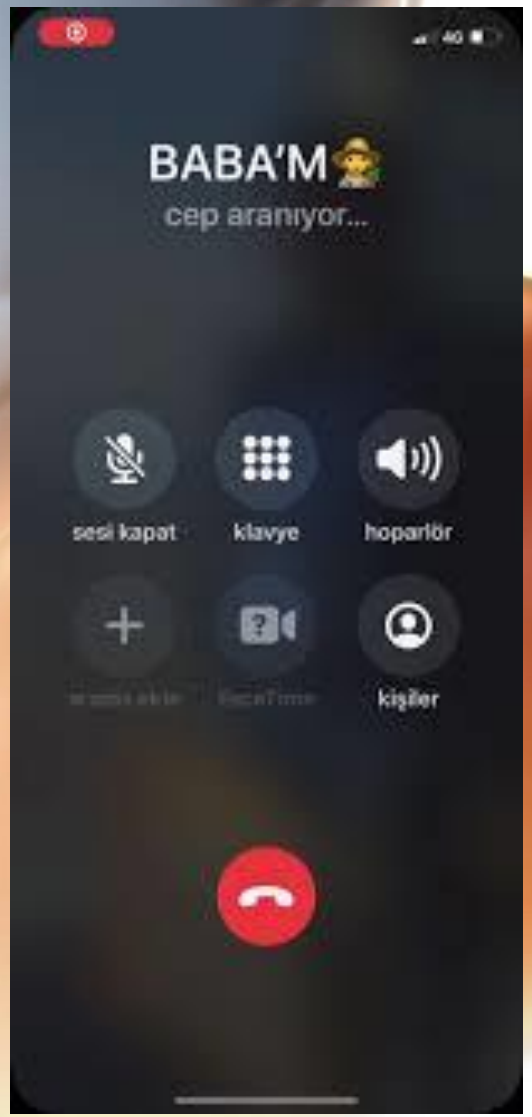
1- super keyword'unu kullanırsanız, hem parent class'in methodunu çağırabilir hem de bu methodun davranışını child class'in ihtiyacına göre genişletebilirsiniz.

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

Super(); Çağrısı

SUPER CALL

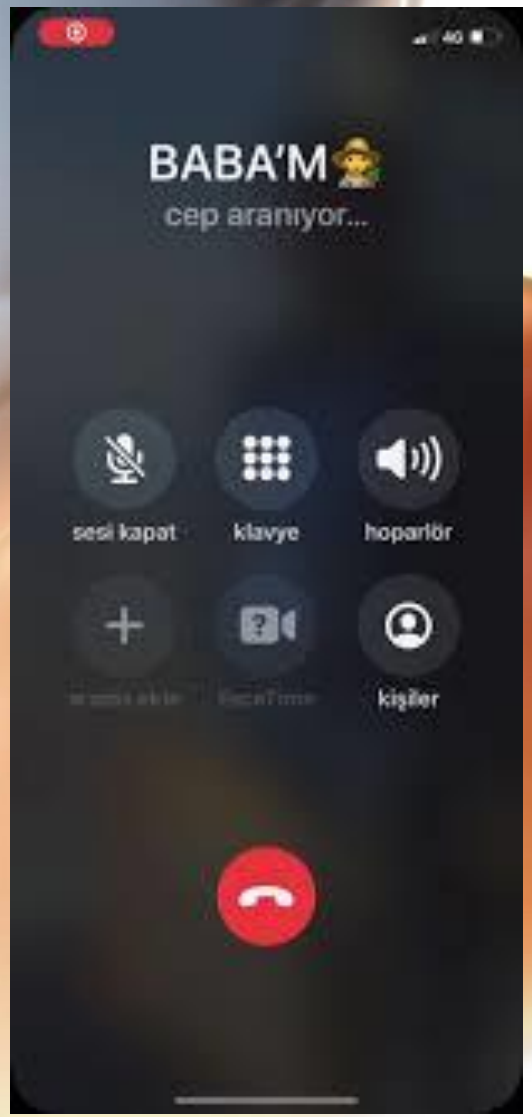
bir alt sınıfın (subclass-child),
üst sınıfının (superclass-parent)
yapıcı (constructor) metodunu veya diğer yöntemlerini çağırmasına olanak tanır.



Super(); Çağrısı Kullanım Amacı

Super anahtar kelimesi Java'da iki ana amaçla kullanılır:

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



Runner.java

```java
package hafta5_superCallsOrnek2;

public class Runner {

    public static void main(String[] args) {
        Torun t= new Torun("Araba");

    }

}
```

```
torun büyüdü ama araba falan istemiyor, ilginç!
Sakin çocuk bizimki, demek ki zamanı var.
Ne arabası ya, daha düşünmedim bile!
```

*Dede.java

```java
package hafta5_superCallsOrnek2;
public class Dede {
    public Dede() {
        super();
        System.out.println("torun büyüdü ama"
                + " araba falan istemiyor, ilginç!");
    }
    public Dede(String soruIstek) {
        super();
        System.out.println("Yakışır torunuma!"
                + "Ben destek olurum");
    }
```

*Torun.java

```java
package hafta5_superCallsOrnek2;
public class Torun extends Baba {
    public Torun() {
        super();
        System.out.println("Ne arabası ya,"
                + " daha düşünmedim bile!");
    }
    public Torun(String torunIstek) {
        super("Baba sen ne dersin");
        System.out.println("Ben biraz biriktireyim "
                + "dedem devamını tamamlar");
    }
```

*Baba.java

```java
package hafta5_superCallsOrnek2;
public class Baba extends Dede{
    public Baba() {
        super();
        System.out.println("Sakin çocuk bizimki,"
                + " demek ki zamanı var.");
    }
    public Baba(String fikirIstek) {
        super("Dedesi sen ne dersin");
        System.out.println("Daha erken!"
                + " Önce para biriktir");
    }
```

Runner.java

```java
package hafta5_superCallsOrnek2;

public class Runner {

    public static void main(String[] args) {
        Torun t= new Torun("Araba");

    }

}
```

*Dede.java

```java
package hafta5_superCallsOrnek2;
public class Dede {
    public Dede() {
        super();
        System.out.println("A");
    }
    public Dede(String soruIstek) {
        super();
        System.out.println("Yakışır torunuma!"
                + "Ben destek olurum");
```

<terminated> Runner [Java Application] C:\Users\nmacit\.p2\pool\plugins\org.eclipse.j

```
Yakışır torunuma!Ben destek olurum
Daha erken! Önce para biriktir
Ben biraz biriktireyim dedem devamını
tamamlar
```

*Torun.java

```java
package hafta5_superCallsOrnek2;

public class Torun extends Baba {

    public Torun() {
        super();
        System.out.println("E");
    }
    public Torun(String torunIstek) {
        super("Baba sen ne dersin");
        System.out.println("Ben biraz biriktireyim "
                + "dedem devamını tamamlar");
```

*Baba.java

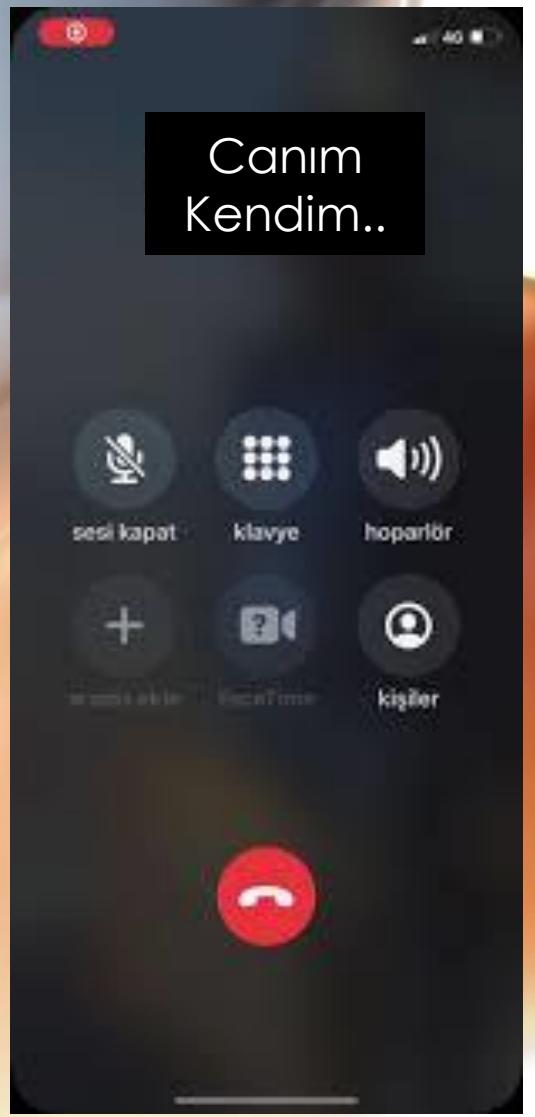
```java
package hafta5_superCallsOrnek2;
public class Baba extends Dede{
    public Baba() {
        super();
        System.out.println("C");
    }
    public Baba(String fikirIstek) {
        super("Dedesi sen ne dersin");
        System.out.println("Daha erken!"
                + " Önce para biriktir");
    }
}
```

This(); Çağrısı

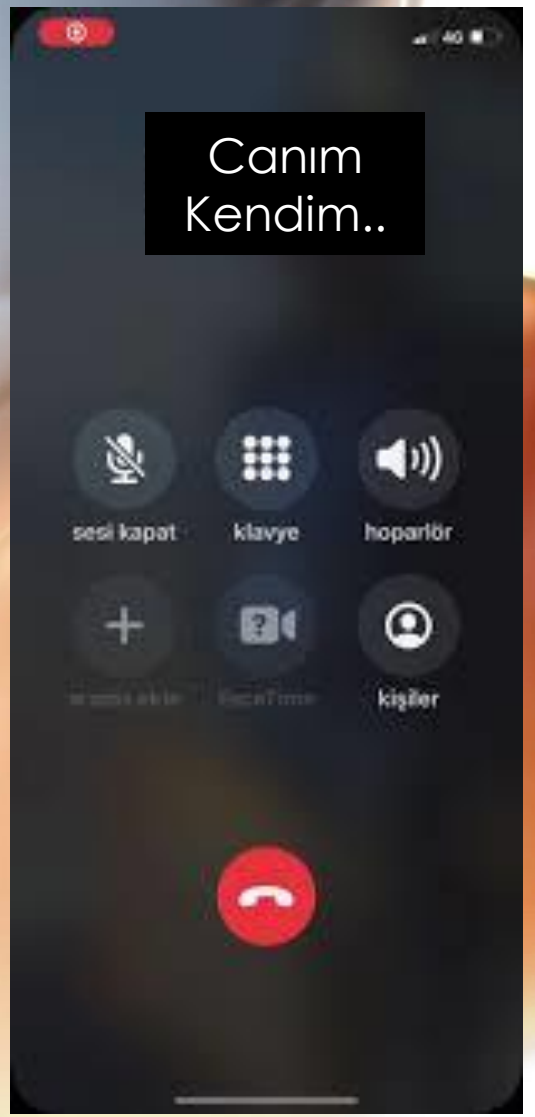
THİS CALL

*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



THİS CALL

*this() ile bir constructor, aynı sınıftaki başka bir constructor'ı parametrelerle birlikte çağırabilir.
*Önemli nokta, this() çağrısının constructor'ın ilk satırında yapılması gerektiğidir.

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.

Runner.java

```java
package hafta5_thisCallsOrnek1;

public class Runner {

    public static void main(String[] args) {
        Torun t= new Torun();

    }

}
```

Problems Javadoc Declaration Console

<terminated> Runner (1) [Java Application] C:\Users\nmacit\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.f

```
Yakışır torunuma!Ben destek olurum
torun büyüdü ama araba falan istemiyor, ilginç!
Sakin çocuk bizimki, demek ki zamanı var.
Ne arabası ya, daha düşünmedim bile!
```

Dede.java

```java
package hafta5_thisCallsOrnek1;
public class Dede {
    public Dede() {
        this("Ben gençliğimde napmıştım");
        System.out.println("torun büyüdü ama"
                + " araba falan istemiyor, ilginç!");
    }
    public Dede(String soruIstek) {
        super();
        System.out.println("Yakışır torunuma!"
                + "Ben destek olurum");
    }
}
```

Torun.java

```java
package hafta5_thisCallsOrnek1;
public class Torun extends Baba {
    public Torun() {
        super();
        System.out.println("Ne arabası ya,"
                + " daha düşünmedim bile!");
    }
    public Torun(String torunIstek) {
        super("Baba sen ne dersin");
        System.out.println("Ben biraz biriktireyim "
                + "dedem devamını tamamlar");
    }
```

Baba.java

```java
package hafta5_thisCallsOrnek1;
public class Baba extends Dede{
    public Baba() {
        super();
        System.out.println("Sakin çocuk bizimki,"
                + " demek ki zamanı var.");
    }
    public Baba(String fikirIstek) {
        super("Dedesi sen ne dersin");
        System.out.println("Daha erken!"
                + " Önce para biriktir");
    }
```

New r.java

```java
package hafta5_thisCallsOrnek1;

public class Runner {

    public static void main(String[] args) {
        Torun t= new Torun("Araba");

    }

}
```

Problems Javadoc Declaration Console

<terminated> Runner (1) [Java Application] C:\Users\nmacit\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.f

```
Yakışır torunuma!Ben destek olurum
Daha erken! Önce para biriktir
Ben biraz biriktireyim dedem devamını tamamlar
```

Dede.java

```java
package hafta5_thisCallsOrnek1;
public class Dede {
    public Dede() {
        this("Ben gençliğimde napmıştım");
        System.out.println("torun büyüdü ama"
                + " araba falan istemiyor, ilginç!");
    }
    public Dede(String soruIstek) {
        super();
        System.out.println("Yakışır torunuma!"
                + "Ben destek olurum");
    }
}
```

Torun.java

```java
package hafta5_thisCallsOrnek1;
public class Torun extends Baba {
    public Torun() {
        super();
        System.out.println("Ne arabası ya,"
                + " daha düşünmedim bile!");
    }
    public Torun(String torunIstek) {
        super("Baba sen ne dersin");
        System.out.println("Ben biraz biriktireyim "
                + "dedem devamını tamamlar");
    }
}
```

Baba.java

```java
package hafta5_thisCallsOrnek1;
public class Baba extends Dede{
    public Baba() {
        super();
        System.out.println("Sakin çocuk bizimki,"
                + " demek ki zamanı var.");
    }
    public Baba(String fikirIstek) {
        super("Dedesi sen ne dersin");
        System.out.println("Daha erken!"
                + " Önce para biriktir");
    }
}
```

This(); Çağrısı ve Super Çağrısı

1. **Görev;**

-This kendine çağrı

-Super parentına çağrı

2. **Görev;**

-This kendinde bulunan bir metod ya da değişkeni çağırır.

-Super parentında bulunan metod ya da değişkeni çağırır.

Runner.java

```java
package hafta5_3super_this_metodCagrisi;

public class Runner {

    public static void main(String[] args) {
        Torun t= new Torun();
    }
}
```

```
Yakışır torunuma!Ben destek olurum
torun büyüdü ama araba falan istemiyor, ilginç!
Sakin çocuk bizimki, demek ki zamanı var.
Kendi cüzdanıma bakayım100
Dedemin cüzdanına bakayım1200
Babamın cüzdanına bakayım2000
```

Torun.java

```java
package hafta5_3super_this_metodCagrisi;
public class Torun extends Baba {
    int torunCuzdan=100;
    public Torun() {
        super();
        System.out.println("Kendi cüzdanıma bakayım"
                +this.torunCuzdan);
        System.out.println("Dedemin cüzdanına bakayım"
        +super.dedeCuzdan);
        System.out.println("Babamın cüzdanına bakayım"
        +super.babaCuzdan);
    }
    public Torun(String torunIstek) {
        super("Baba sen ne dersin");
        System.out.println("Ben biraz biriktireyim "
                + "dedem devamını tamamlar");   }
```

Dede.java

```java
package hafta5_3super_this_metodCagrisi;
public class Dede {
    int dedeCuzdan=1200;
    public Dede() {
        this("Ben gençliğimde napmıştım");
        System.out.println("torun büyüdü ama"
                + " araba falan istemiyor, ilginç!");}
    public Dede(String soruIstek) {
        super();
        System.out.println("Yakışır torunuma!"
                + "Ben destek olurum");}
}
```

Baba.java

```java
package hafta5_3super_this_metodCagrisi;
public class Baba extends Dede{
    int babaCuzdan=2000;
    public Baba() {
        super();
        System.out.println("Sakin çocuk bizimki,"
                + " demek ki zamanı var.");
    }
    public Baba(String fikirIstek) {
        super("Dedesi sen ne dersin");
        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

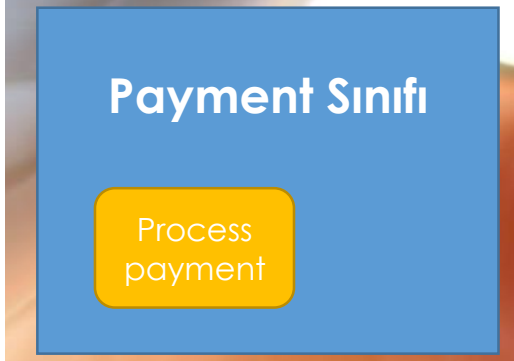
Polymorphism (Çok Biçimlilik), nesne yönelimli programlamada, bir sınıfın farklı şekillerde davranabilmesi anlamına gelir.
Java'da polymorphism, aynı isimdeki bir metodu farklı sınıflarda veya aynı sınıf içinde farklı parametrelerle kullanabilme yeteneği sağlar.
Polymorphism iki ana şekilde uygulanabilir

- **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 | CreditCardPayment.java | DebitCardPayment.java | CashPayment.java | PaymentRunner.java

```java
package hafta5_4Polymorphism;

public class Payment {
	public void processPayment(double amount) {
		System.out.println("Ödeme işleniyor: " + amount + " TL");
	  }
	/*Ödeme işlemleri parent class*/
}

```



LARI

Payment.java | CreditCardPayment.java | DebitCardPayment.java | CashPayment.java | PaymentRunner.java

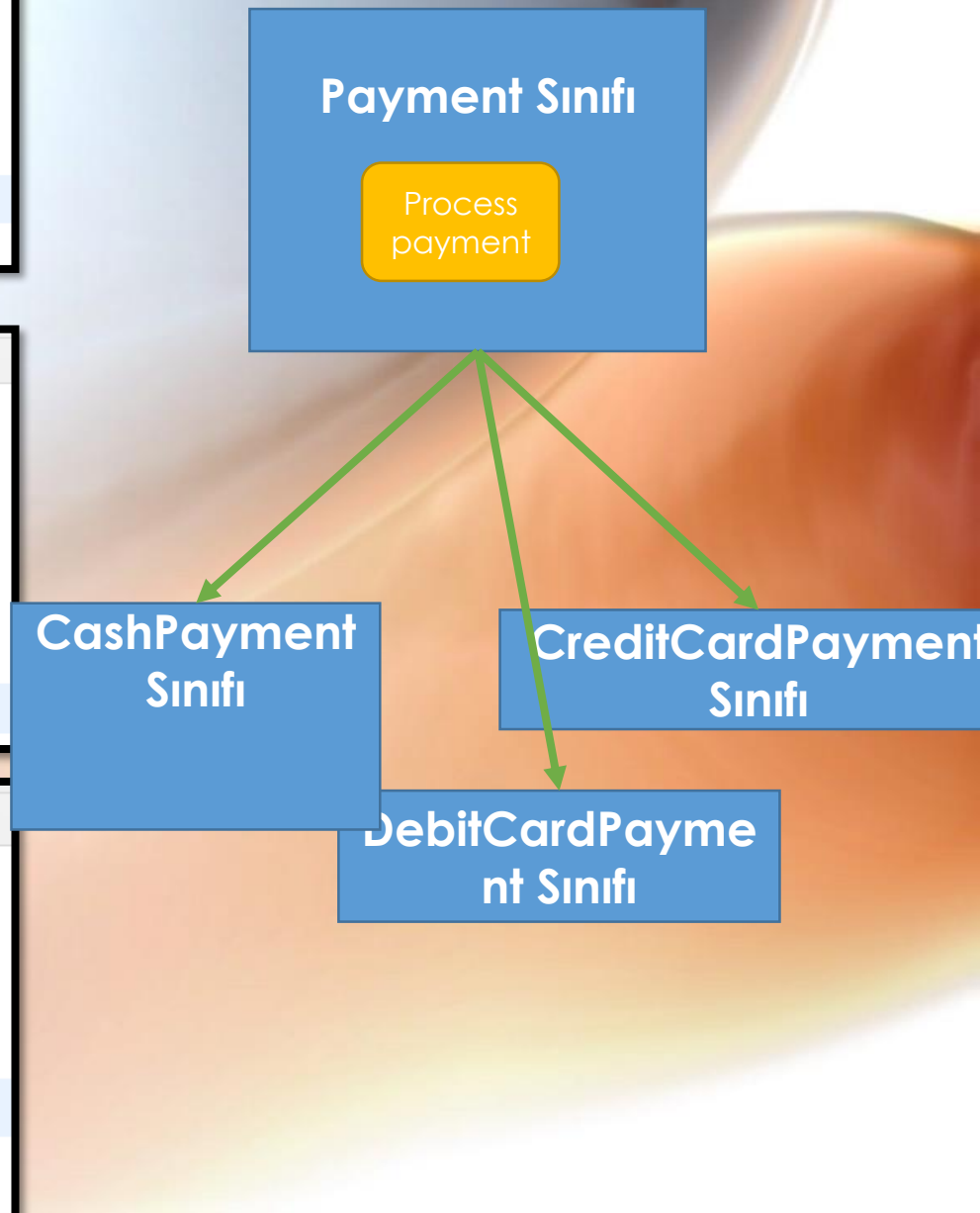
```java
package hafta5_4Polymorphism;

public class CreditCardPayment extends Payment {
    /*Kredi kartı işlemleri için oluşturuldu
     * parent class= Payment*/

}

```

Payment.java | CreditCardPayment.java | DebitCardPayment.java | *CashPayment.java | PaymentRunner.java

```java
package hafta5_4Polymorphism;

public class CashPayment extends Payment{
    /*Nakit işlemleri için oluşturuldu
     * parent class= Payment*/
}

```

Payment.java | CreditCardPayment.java | DebitCardPayment.java | CashPayment.java | PaymentRunner.java

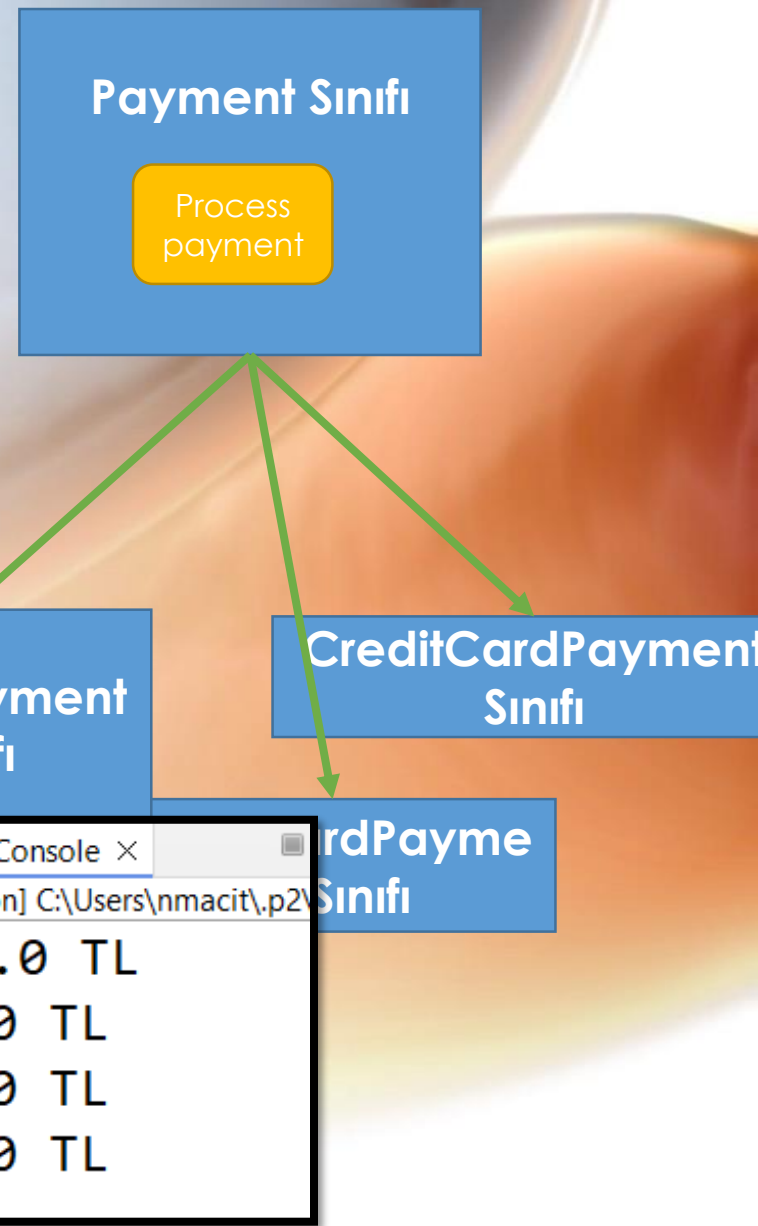
```java
package hafta5_4Polymorphism;

public class DebitCardPayment extends Payment {
    /*Banka kartı işlemleri için oluşturuldu
     * parent class ı Payment*/
}
```



Overriding

Payment.java | CreditCardPayment.java | DebitCardPayment.java | CashPayment.java | PaymentRunner.java

```java
package hafta5_4Polymorphism;

public class PaymentRunner {
    public static void main(String[] args) {
            Payment payment1 = new Payment();
            DebitCardPayment payment2 = new DebitCardPayment();
            CashPayment payment3 = new CashPayment();
            CreditCardPayment payment4 = new CreditCardPayment();

            payment1.processPayment(150.0);
            payment2.processPayment(75.0);
            payment3.processPayment(50.0);
            payment4.processPayment(50.0);

    }

}
```



Problems | Javadoc | Declaration | Console

<terminated> PaymentRunner [Java Application] C:\Users\nmacit\.p2

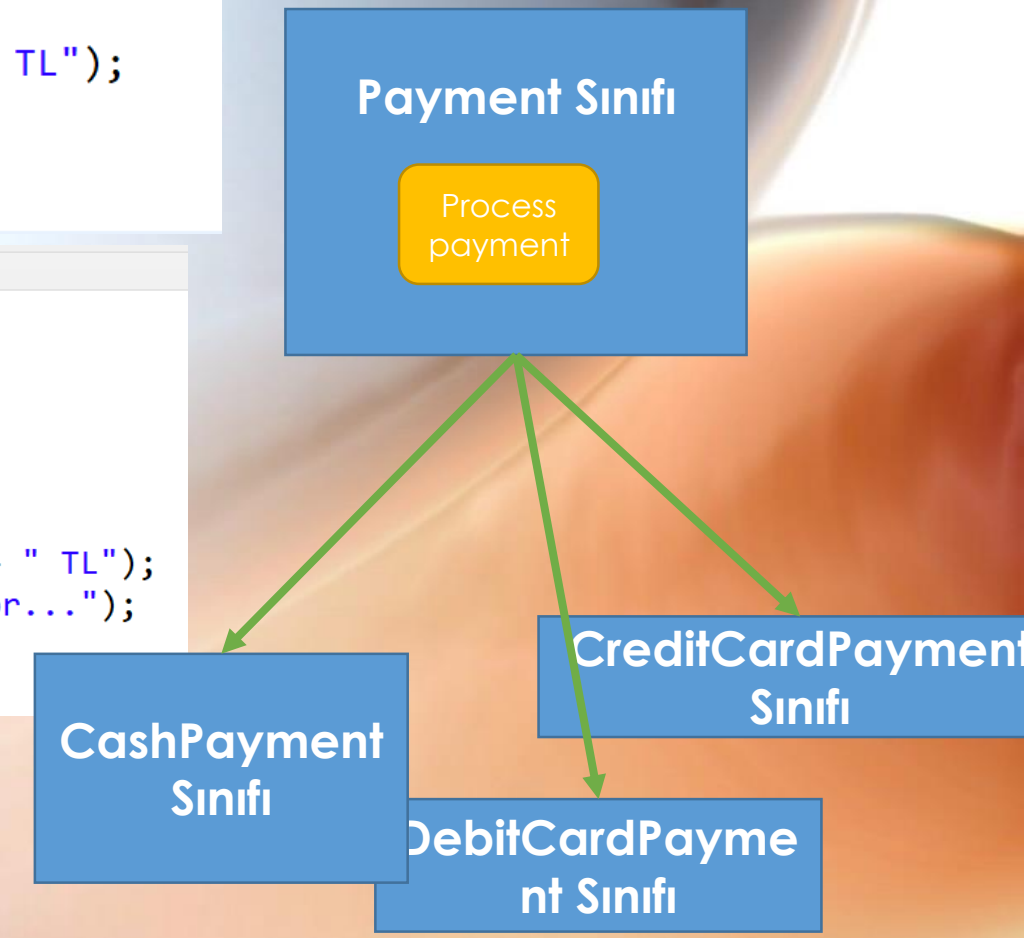
```
Ödeme işleniyor: 150.0 TL
Ödeme işleniyor: 75.0 TL
Ödeme işleniyor: 50.0 TL
Ödeme işleniyor: 50.0 TL
```

Payment.java | CreditCardPayment.java | DebitCardPayment.java | *CashPayment.java | PaymentRunner.java

```java
package hafta5_4Polymorphism;

public class Payment {
	public void processPayment(double amount) {
		System.out.println("Ödeme işleniyor: " + amount + " TL");
	}
	/*Ödeme işlemleri parent class*/
}
```

Payment.java | *CreditCardPayment.java | DebitCardPayment.java | *CashPayment.java | PaymentRunner.java

```java
package hafta5_4Polymorphism;
public class CreditCardPayment extends Payment {
	/*Kredi kartı işlemleri için oluşturuldu
	 * parent class= Payment*/
	@Override
	public void processPayment(double amount) {
		System.out.println("Kredi kartı ile ödeme yapılıyor: " + amount + " TL");
		System.out.println("Kredi kartı doğrulama ve işlem onayı yapılıyor...");
	}
}
```

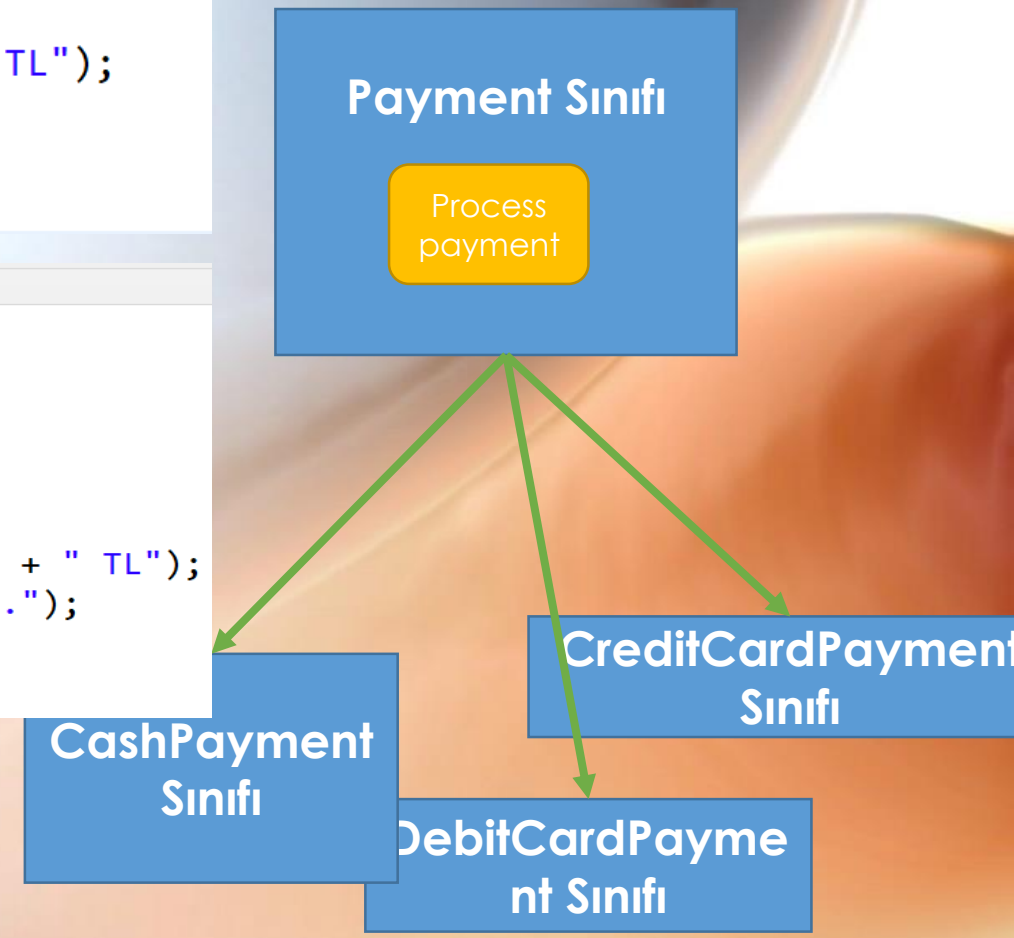


Payment.java | CreditCardPayment.java | DebitCardPayment.java | *CashPayment.java | PaymentRunner.java

```java
package hafta5_4Polymorphism;

public class Payment {
	public void processPayment(double amount) {
		System.out.println("Ödeme işleniyor: " + amount + " TL");
	}
	/*Ödeme işlemleri parent class*/
}
```

Payment.java | *CreditCardPayment.java | *DebitCardPayment.java | *CashPayment.java | PaymentRunner.java

```java
package hafta5_4Polymorphism;
public class DebitCardPayment extends Payment {
    /*Banka kartı işlemleri için oluşturuldu
     * parent class ı Payment*/
    @Override
    public void processPayment(double amount) {
        System.out.println("Banka kartı ile ödeme yapılıyor: " + amount + " TL");
        System.out.println("Banka hesabındaki bakiye kontrol ediliyor...");
    }
}
```

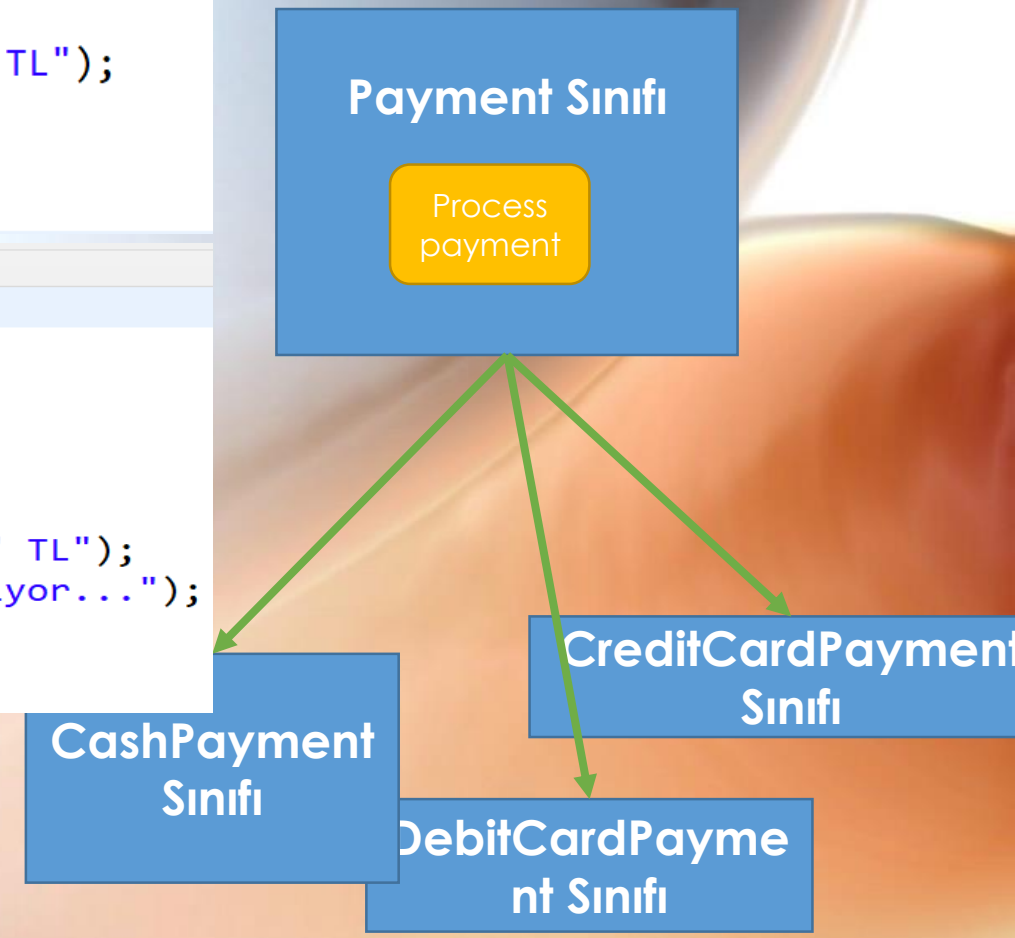


Payment.java | CreditCardPayment.java | DebitCardPayment.java | *CashPayment.java | PaymentRunner.java

```java
package hafta5_4Polymorphism;

public class Payment {
	public void processPayment(double amount) {
		System.out.println("Ödeme işleniyor: " + amount + " TL");
	}
	/*Ödeme işlemleri parent class*/
}
```

Payment.java | *CreditCardPayment.java | *DebitCardPayment.java | *CashPayment.java | PaymentRunner.java

```java
package hafta5_4Polymorphism;
public class CashPayment extends Payment{
	/*Nakit işlemleri için oluşturuldu
	 * parent class= Payment*/
	@Override
	public void processPayment(double amount) {
		System.out.println("Nakit ödeme yapılıyor: " + amount + " TL");
		System.out.println("Kasiyere nakit ödeme alındı bildiriliyor...");
	}
}
```



Override

```java
package hafta5_4Polymorphism;

public class PaymentRunner {
    public static void main(String[] args) {
    Payment payment1 = new Payment();
    DebitCardPayment payment2 = new DebitCardPayment();
    CashPayment payment3 = new CashPayment();
    CreditCardPayment payment4 = new CreditCardPayment();

    payment1.processPayment(150.0);
    payment2.processPayment(75.0);
    payment3.processPayment(50.0);
    payment4.processPayment(50.0);
    }
}
```

Payment.java

```java
package hafta5_4Polymorphism;

public class Payment {
      public void processPayment(double amount) {
            System.out.println("Ödeme işleniyor: " + amount + " TL");
        }
      /*Ödeme işlemleri parent class*/
}
```

*DebitCardPayment.java

```java
package hafta5_4Polymorphism;
public class DebitCardPayment extends Payment {
    /*Banka kartı işlemleri için oluşturuldu
     * parent class ı Payment*/
    @Override
    public void processPayment(double amount) {
        System.out.println("Banka kartı ile ödeme yapılıyor: " + amount + " T
        System.out.println("Banka hesabındaki bakiye kontrol ediliyor...");
    }
}
```

*CashPayment.java

```java
package hafta5_4Polymorphism;
public class CashPayment extends Payment{
     /*Nakit işlemleri için oluşturuldu
      * parent class= Payment*/
    @Override
    public void processPayment(double amount) {
        System.out.println("Nakit ödeme yapılıyor: " + amount + " TL");
        System.out.println("Kasiyere nakit ödeme alındı bildiriliyor...
    }
}
```

Sınıfı

*CreditCardPayment.java

```java
package hafta5_4Polymorphism;
public class CreditCardPayment extends Payment {
     /*Kredi kartı işlemleri için oluşturuldu
      * parent class= Payment*/
     @Override
     public void processPayment(double amount) {
         System.out.println("Kredi kartı ile ödeme yapılıyor: " + amount + " T
         System.out.println("Kredi kartı doğrulama ve işlem onayı yapılıyor...
     }
}
```

Problems | Javadoc | Declaration | Console

<terminated> PaymentRunner [Java Application] C:\Users\nmacit\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspo

```
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...
```

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 ulaşılamaz 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; *Methot, 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 | Account.java | UserAccount.java

```java
package hafta5_5Final;

public class Account {
    public final void login() {
        System.out.println("Kullanıcı sisteme giriş yapıyor...");
    }

    public void displayAccountType() {
        System.out.println("Hesap türü: Genel Hesap");
    }
}

```

FinalVariableExample.java | Account.java | *UserAccount.java

```java
package hafta5_5Final;

public class UserAccount extends Account{
    @Override
    public void displayAccountType() {
        System.out.println("Hesap türü: Kullanıcı Hesabı");
    }
    //@Override
    // public void login() {
    //     System.out.println("Özel bir giriş işlemi.");
    // }  // Hata verir, çünkü login metodu final olarak tanımlandı.
}

```

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 | 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

```
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

Encapsullation-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 × PersonRunner.java

ileriNesneTabanliProgramlama_Guz_24_25 ▸ src ▸ hafta6 ▸ Person ▸

```java
package hafta6;
public class Person {
    private String name;    // Private değişken
    private int age;
    public String getName() {
        return name;        // Getter metodu
    }
    public void setName(String name) {
        this.name = name;   // Setter metodu
    }
    public int getAge() {
        return age;         // Getter metodu
    }
    public void setAge(int age) { // Setter metodu (doğrulama ekledik)
        if (age > 0) {
            // Yaşın pozitif olması gerektiğini kontrol ediyoruz
            this.age = age;
        } else {
            System.out.println("Yaş negatif olamaz!");
        }
    }
}
```

Kapsülleme

Person.java × PersonRunner.java

ileriNesneTabanliProgramlama_Guz_24_25 ▸ src ▸ hafta6 ▸ Person ▸

```java
package hafta6;
public class Person {
    private String name;    // Private değişken
    private int age;
    public String getName() {
        return name;        // Getter metodu
    }
    public void setName(String name) {
        this.name = name;   // Setter metodu
    }
    public int getAge() {
        return age;         // Getter metodu
    }
    public void setAge(int age) { // Setter metodu (doğrulama ekledik)
        if (age > 0) {
            // Yaşın pozitif olması gerektiğini kontrol ediyoruz
            this.age = age;
        } else {
            System.out.println("Yaş negatif olamaz!");
        }
    }
}
```

Kapsülleme

Person.java | *PersonRunner.java

ileriNesneTabanliProgramlama_Guz_24_25 ▸ src ▸ hafta6 ▸ PersonRunner ▸ main(String[]) : void

```java
package hafta6;

public class PersonRunner {
    public static void main(String[] args) {
        Person person = new Person();
        // Setter metoduyla veri atama

        person.setName("Naciye");
        person.setAge(25);

        // Getter metoduyla verileri alma
        System.out.println("Ad: " + person.getName());
        System.out.println("Yaş: " + person.getAge());

        // Hatalı veri atama
        person.setAge(-5);
        // "Yaş negatif olamaz!" yazdırır.
    }
}
```

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!