# 1. Ders: JAVA ile Nesne Yönelimli Programlama Metotlar (Methods)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama (Ders: 1-10) video.

    - Ders 1: Nesne Yönelimli Programlamaya Giriş (izle)
    - Ders 2: Neden Nesne Yönelimli Programlama? (izle)
    - Ders 3: Nesne (Object) Kavramı (izle)
    - Ders 4: Sınıf (Class) Kavramı (izle)
    - Ders 5: Nesne ve Sınıf Kavramları Arasındaki İlişki (izle)
    - Ders 6: Kalıtım/Miras (Inheritance) Kavramı (izle)
    - Ders 7: Çok Biçimlilik (Polymorphism) (izle)
    - Ders 8: Soyutlama (Abstraction) (izle)
    - Ders 9: Kapsülleme (Encapsulation) (izle)
    - Ders 10: Prosedür Tabanlı Programlama ile OOP Karşılaştırması (izle)

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# Metot ihtiyacı

1’den 10’a kadar, 20’den 37’ye kadar ve 35’ten 49’a kadar tam sayıların toplamını bulmak isteyelim.

```
int top = 0;
for(int i = 1; i <= 10; i++)
   top += i;
System.out.println("1’den 10’a kadar toplam " + top);

top = 0;
for(int i = 20; i <= 37; i++)
   top += i;
System.out.println("20’den 37’ye kadar toplam " + top);

top = 0;
for(int i = 35; i <= 49; i++)
   top += i;
System.out.println("35’ten 49’a kadar toplam " + top);
```

# Metot kullanımı

1’den 10’a kadar, 20’den 37’ye kadar ve 35’ten 49’a kadar tam sayıların toplamını bulmak isteyelim.

```
public static int top(int i1, int i2){
    int sonuc = 0;
    for(int i = i1; i <= i2; i++)
        sonuc += i;
    return sonuc;
}
public static void main(String [] args){
    System.out.println("1’den 10’a kadar toplam " + top(1,10));
    System.out.println("20’den 37’ye kadar toplam " + top(20,37));
    System.out.println("35’ten 49’a kadar toplam " + top(35,49));
}
```

1’den 10’a kadar, 20’den 37’ye kadar ve 35’ten 49’a kadar tam sayıların toplamını bulmak isteyelim.

```java
import java.util.Scanner;
public class SiraliToplam1 {
    public static void main(String [] args) {
        int top = 0;
        for(int i = 1; i<= 10 ; i++)
            top += i;
        System.out.println("1'den 10'a kadar toplam: " + top);

        top = 0;
        for(int i = 20; i<= 37 ; i++)
            top += i;
        System.out.println("20'den 37'ye kadar toplam: " + top);

        top = 0;
        for(int i = 35; i<= 49 ; i++)
            top += i;
        System.out.println("35'ten 49'a kadar toplam: " + top);
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java SiraliToplam1
1'den 10'a kadar toplam: 55
20'den 37'ye kadar toplam: 513
35'ten 49'a kadar toplam: 630
```

1’den 10’a kadar, 20’den 37’ye kadar ve 35’ten 49’a kadar tam sayıların toplamını bulmak isteyelim.

```java
import java.util.Scanner;
public class SiraliToplam {
    public static int top(int i1, int i2) {
    int sonuc = 0;
    for(int i = i1; i<= i2 ; i++)
        sonuc += i;
    return sonuc;
    }

    public static void main(String [] args) {
        System.out.println("1'den 10'a kadar toplam: "   + top(1,10));
        System.out.println("20'den 37'ye kadar toplam: " + top(20,37));
        System.out.println("35'ten 49'a kadar toplam: "  + top(35,49));
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java SiraliToplam
1'den 10'a kadar toplam: 55
20'den 37'ye kadar toplam: 513
35'ten 49'a kadar toplam: 630
```

# Metotlar (Yöntemler)

- Kodun sadeleştirilmesi
- Kodun düzenlenmesi
- ve yeniden kullanılabilmesine yarar.

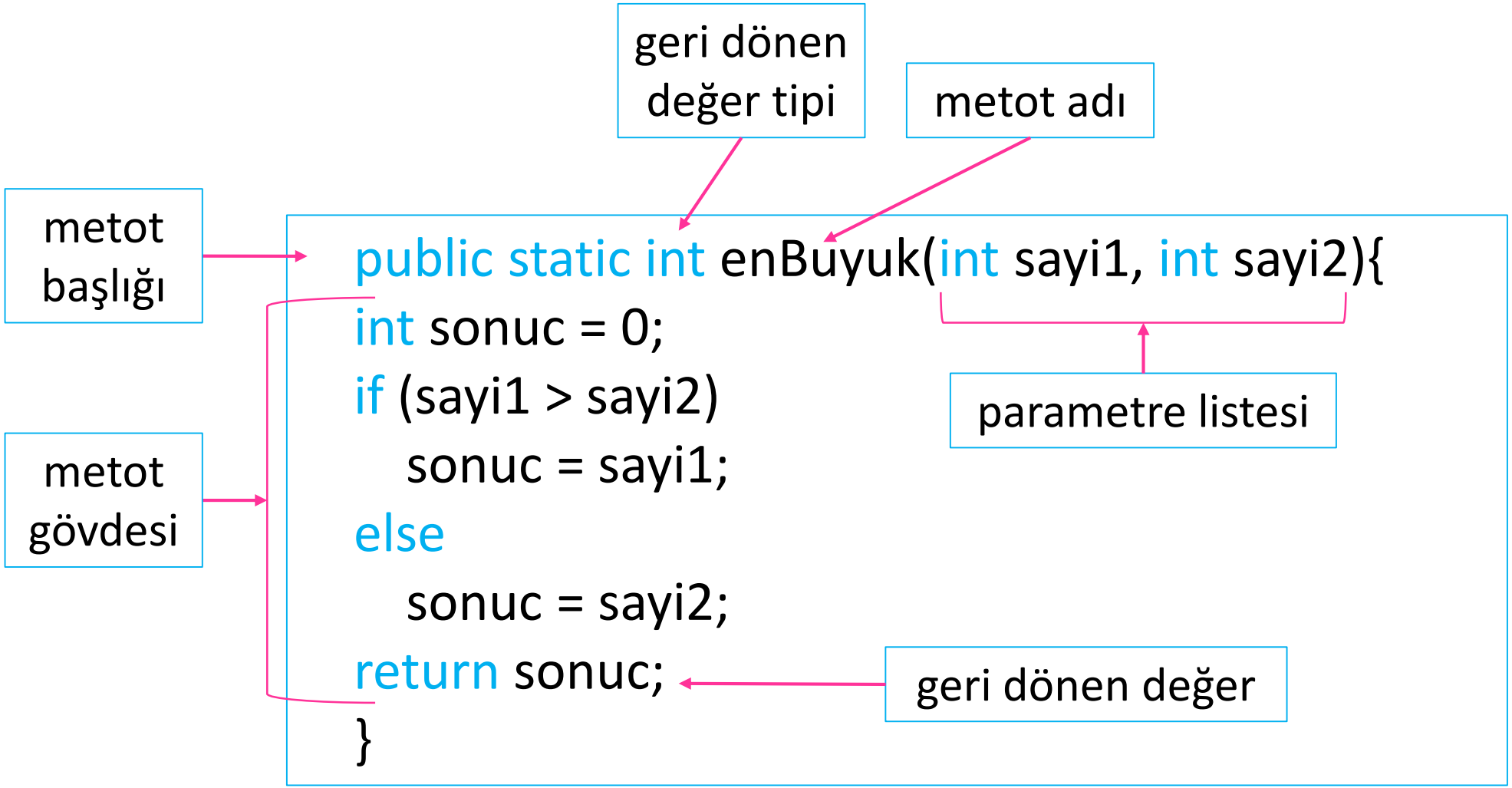
# Metot (Yöntem)

Metot,  bir işlemi gerçekleştirmek için bir araya getirilmiş ifadelerin koleksiyonudur.

Daha önceki derslerde ön tanımlı metotlar kullandık:
System.out.println
System.exit
Math.pow
Math.random
Bu metotlar Java kütüphanelerinde ön tanımlanmıştır.

Artık kendi metotlarınızı yazma zamanı geldi.

# Metot tanımlama



# Metot çağırma

Metot çağrılırken geri dönen veri tipi ve alması gereken parametreler girilir.

```
int z = enBuyuk(x, y)
```

Bazı dillerde metotlar fonksiyon ya da prosedür olarak isimlendirilirler. Bu dillerde geri değer döndürenlere fonksiyon ve değer döndürmeyenlere prosedür denir.

# Örnek 1:

```java
import java.util.Scanner;
public class TestEnBuyuk {
    public static void main(String [] args) {
        int i = 5;
        int j = 2;
        int k = enBuyuk(i, j);
        System.out.println(i + " ve " + j + " nin en buyugu "
        + k + " dir.");
    }

    public static int enBuyuk(int sayi1, int sayi2){
        int sonuc;

        if (sayi1 > sayi2)
            sonuc = sayi1;
        else
            sonuc = sayi2;

        return sonuc;
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestEnBuyuk
5 ve 2 nin en buyugu 5 dir.
```

# Örnek 1:

```
1  import java.util.Scanner;
2  public class TestEnBuyuk {
3      public static void main(String [] args) {
4          int i = 5;
5          int j = 2;
6          int k = enBuyuk(i, j);
7          System.out.println(i + " ve " + j + " nin en buyugu "
8          + k + " dir.");
9      }
10
11     public static int enBuyuk(int sayi1, int sayi2){
12         int sonuc;
13
14         if (sayi1 > sayi2)
15             sonuc = sayi1;
16         else
17             sonuc = sayi2;
18
19         return sonuc;
20     }
21 }
```

| satır | i | j | k | sayi1 | sayi2 | sonuc |
|---|---|---|---|---|---|---|
| 4 | 5 | | | | | |
| 5 | | 2 | | | | |
| 11 | | | | 5 | 2 | |
| 12 | | | | | | tanımlı değil |
| 15 | | | | | | 5 |
| 6 | | | 5 | | | |

# Bir sınıftan metot çağırma

Metotlar kod paylaşımına ve kodun yeniden kullanımına olanak sağlar.

Örnek 1’ deki enBuyuk metotu sadece TestEnBuyuk sınıfında değil, tüm sınıflardan çağrılabilir.

Bir metotu başka bir sınıftan çağırmak için SinifAdi.metotAdi şeklinde yazmamız yeterlidir. TestEnBuyuk.enBuyuk()

# Yığın (stack)

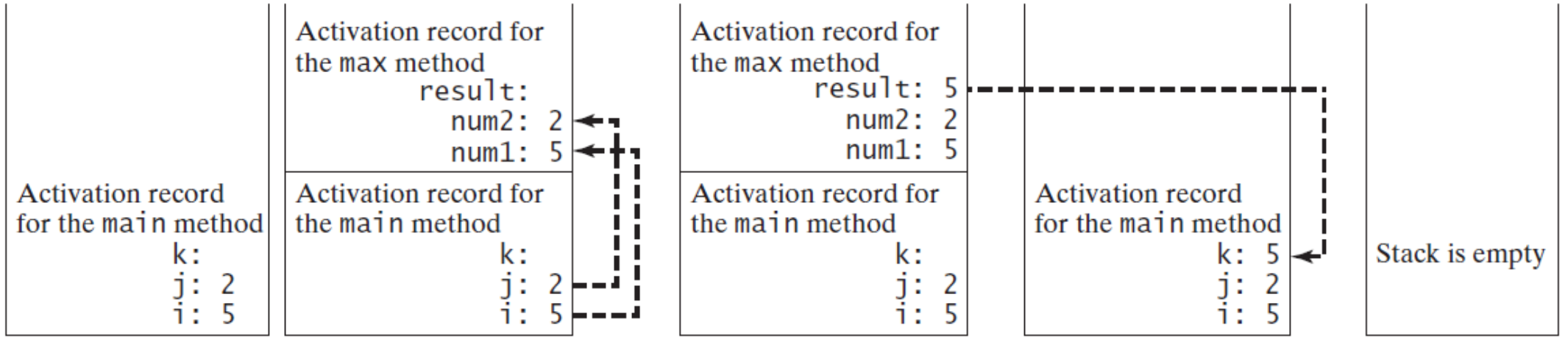



**FIGURE 6.3** When the `max` method is invoked, the flow of control transfers to the `max` method. Once the `max` method is finished, it returns control back to the caller.

# void Metot örneği notYazdir metodu

```java
import java.util.Scanner;
public class TestVoidMetot {
    public static void main(String [] args) {
        System.out.print("Not degeri: ");
        notYazdir(78.5);

        System.out.print("Not degeri: ");
        notYazdir(59.5);
    }

    public static void notYazdir(double dersnotu){
        if (dersnotu >= 90.0){
            System.out.println('A');
        }
        else if (dersnotu >= 80.0){
            System.out.println('B');
        }
        else if (dersnotu >= 70.0){
            System.out.println('C');
        }
        else if (dersnotu >= 60.0){
            System.out.println('D');
        }
        else{
            System.out.println('F');
        }
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestVoidMetot
Not degeri: C
Not degeri: F
```

```
11      public static void notYazdir(double dersnotu){
12          if (dersnotu<0 || dersnotu>100){
13              System.out.println("Gecersiz ders notu");
14              return;
15          }
16
17          if (dersnotu >= 90.0){
18              System.out.println('A');
19          }
20          else if (dersnotu >= 80.0){
21              System.out.println('B');
22          }
23          else if (dersnotu >= 70.0){
24              System.out.println('C');
25          }
26          else if (dersnotu >= 60.0){
27              System.out.println('D');
28          }
29          else{
30              System.out.println('F');
31          }
32      }
```

void metodunda return'e gerek yoktur.

Ancak metodu sonlandırmak istediğimiz durumlarda return kullanabiliriz.

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestVoidMetot
Not degeri: C
Not degeri: Gecersiz ders notu
```

# notGetir metodu

```
1  import java.util.Scanner;
2  public class TestNotGetir {
3      public static void main(String [] args) {
4          System.out.print("Not degeri: " + notGetir(78.5));
5          System.out.print("\nNot degeri: " + notGetir(59.5));
6      }
7
8      public static char notGetir(double dersnotu){
9          if (dersnotu >= 90.0)
10             return 'A';
11         else if (dersnotu >= 80.0)
12             return 'B';
13         else if (dersnotu >= 70.0)
14             return 'C';
15         else if (dersnotu >= 60.0)
16             return 'D';
17         else
18             return 'F';
19     }
20 }
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestNotGetir
Not degeri: C
Not degeri: F
```

# Parametre sırası

```java
import java.util.Scanner;
public class TestnPrintln {
    public static void main(String [] args) {
        nPrintln("Merhaba",5);
    }

    public static void nPrintln(String mesaj, int n){
        for(int i=0; i<n ;i++)
            System.out.println(mesaj);
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestnPrintln
Merhaba
Merhaba
Merhaba
Merhaba
Merhaba
```

# Argümanlar

```java
import java.util.Scanner;
public class TestArttir {
    public static void main(String [] args) {
        int x = 1;
        System.out.println("metodu cagirmadan once x: " + x);
        arttir(x);
        System.out.println("metodu cagirdiktan sonra x: " + x);
    }

    public static void arttir(int n){
        n++;
        System.out.println("metodun icinde n: " + n);
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestArttir
metodu cagirmadan once x: 1
metodun icinde n: 2
metodu cagirdiktan sonra x: 1
```

# Ödev 1:

# Ödev 2:

# Ödev 3:

# 2. Ders: JAVA ile Nesne Yönelimli Programlama Nesneler ve sınıflar (Objects and classes)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama (Ders: 1-10) video.

    - Ders 1: Nesne Yönelimli Programlamaya Giriş (izle)
    - Ders 2: Neden Nesne Yönelimli Programlama? (izle)
    - Ders 3: Nesne (Object) Kavramı (izle)
    - Ders 4: Sınıf (Class) Kavramı (izle)
    - Ders 5: Nesne ve Sınıf Kavramları Arasındaki İlişki (izle)
    - Ders 6: Kalıtım/Miras (Inheritance) Kavramı (izle)
    - Ders 7: Çok Biçimlilik (Polymorphism) (izle)
    - Ders 8: Soyutlama (Abstraction) (izle)
    - Ders 9: Kapsülleme (Encapsulation) (izle)
    - Ders 10: Prosedür Tabanlı Programlama ile OOP Karşılaştırması (izle)

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# Nesneler ve Sınıflar (Objects and Classes)

Nesne yönelimli programlama sayesinde büyük ölçekli yazılımlar ve grafik kullanıcı ara yüzleri tasarlayabilmek mümkündür.

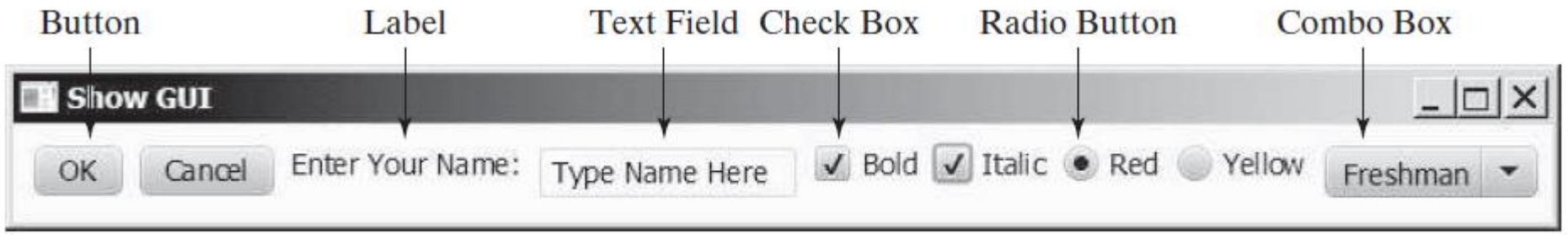



**FIGURE 9.1** The GUI objects are created from classes.

# Nesneler için Sınıf Tanımlama

Sınıf, bir nesnenin özelliklerini ve davranışlarını tanımlar.

Nesne yönelimli programlama nesneleri kullanarak programlamayı içerir.

Bir nesne, gerçek dünyada açıkça tanımlanabilen bir varlığı temsil eder.

Örnek olarak, bir öğrenci, bir masa, bir düğme, bir çember … bir nesnedir.

Bir nesnenin kendine has kimliği, durumu ve davranışı vardır.

# Durum (Özellik veya Nitelik)

Bir nesnenin durumu (özellikleri veya nitelikleri) (properties or attributes), veri alanları ve mevcut değerlerle temsil edilir.

Çember bir nesnedir ve bu çemberi karakterize eden yarıçap veri alanı özelliğine sahiptir.

Dikdörtgen bir nesnedir. Dikdörtgen kendisini karakterize eden en ve boy özelliklerine sahiptir. En ve boy veri alanı şeklindedir ve değerleri vardır.

# Davranış (Eylem)

Bir nesnenin davranışı (eylemleri) (actions), metotlarla temsil edilir.

Bir nesnenin metodunu çağırmak için nesneden eylem yapması istenir.
Örnek olarak bir çember için getArea() ve getPerimeter() metotları tanımlanmış olsun.
Çember nesnesi getArea() ve getPerimeter() metotlarını çağırarak kendi alanının ve çevresinin değerini geri döndürür.

Ayrıca setRadius(yaricap) şeklinde bir metot tanımlanabilir. Çember bu metodu kullanarak kendi yaricap özelliğini değiştirebilir.

# Nesne ve sınıf

Aynı türdeki nesneler, ortak bir sınıf kullanılarak tanımlanabilir.

Sınıf, bir nesnenin özelliklerinin (veri alanlarının) ve metotlarının (davranışının) ne olacağını tanımlayan bir şablon, plan veya sözleşmedir.

Bir nesne bir sınıfın örneğidir. Bir sınıfın birçok örneğini oluşturabilirsiniz. Örnek oluşturma, somutlaştırma olarak adlandırılır.

Nesne (object) ve örnek (instance) terimleri genellikle birbirinin yerine kullanılabilir.

Sınıflar ve nesneler arasındaki ilişki, bir kek tarifi ile kek arasındaki ilişkiye benzer: Tek bir tariften istediğiniz kadar kek yapabilirsiniz.

# Nesne ve sınıf

Bir Java sınıfı, veri alanlarını tanımlamak için değişkenleri ve eylemleri tanımlamak için metotları kullanır.

Bir yapıcı (constructor) herhangi bir eylemi gerçekleştirebilir, ancak yapıcılar başlangıç eylemlerini gerçekleştirmek için tasarlanmıştır.

Bir nesnenin veri alanlarının başlangıç değer ataması buna örnek olarak verilebilir.

# Circle sınıfı

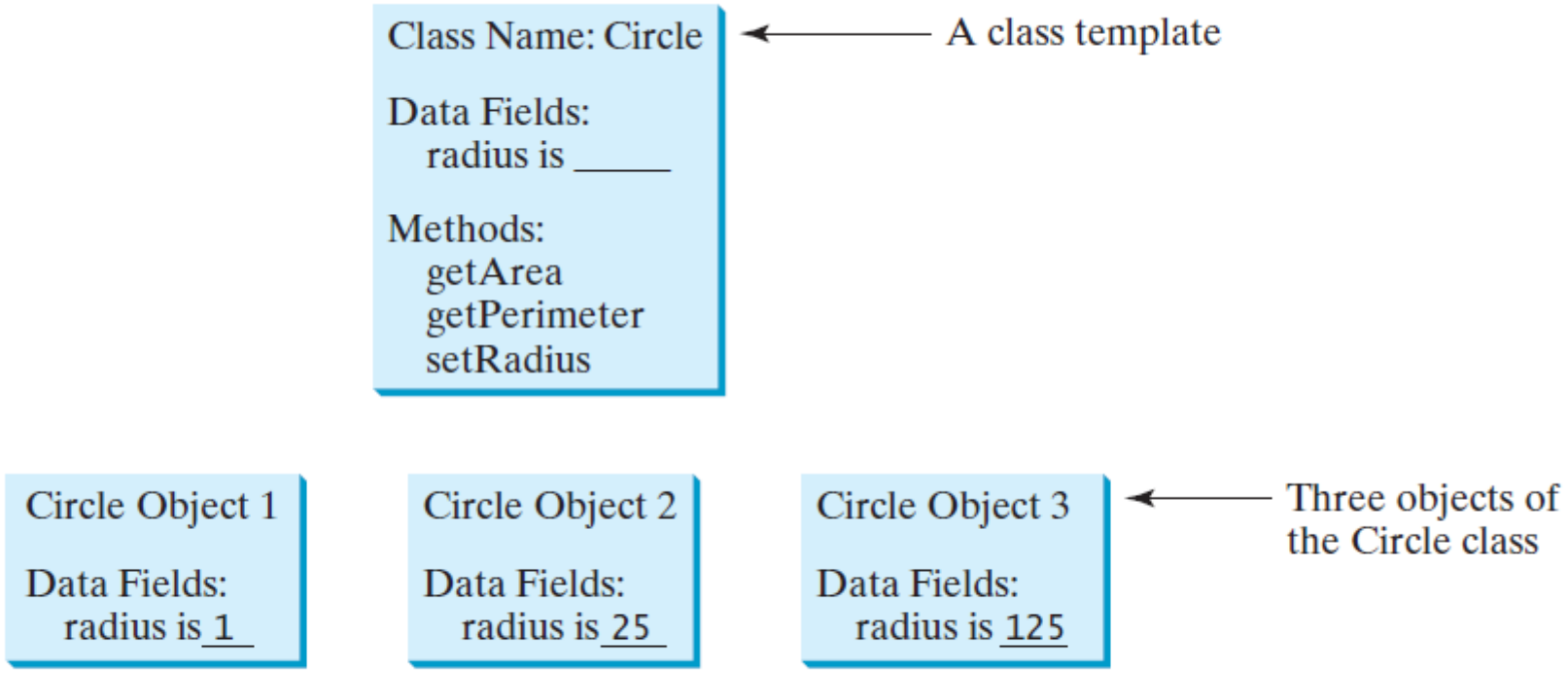



**FIGURE 9.2** A class is a template for creating objects.

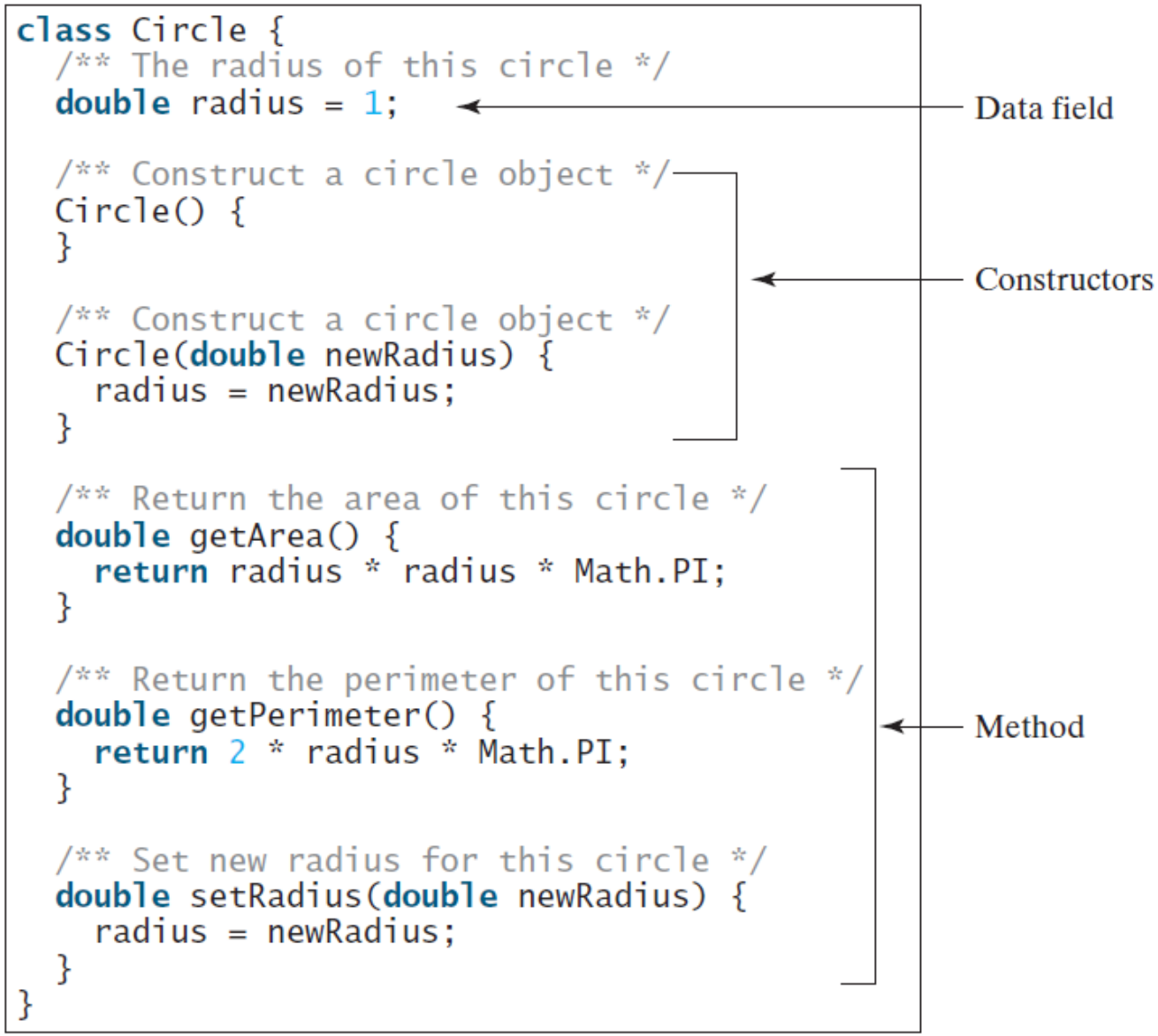



**FIGURE 9.3** A class is a construct that defines objects of the same type.

# UML Class Diagram

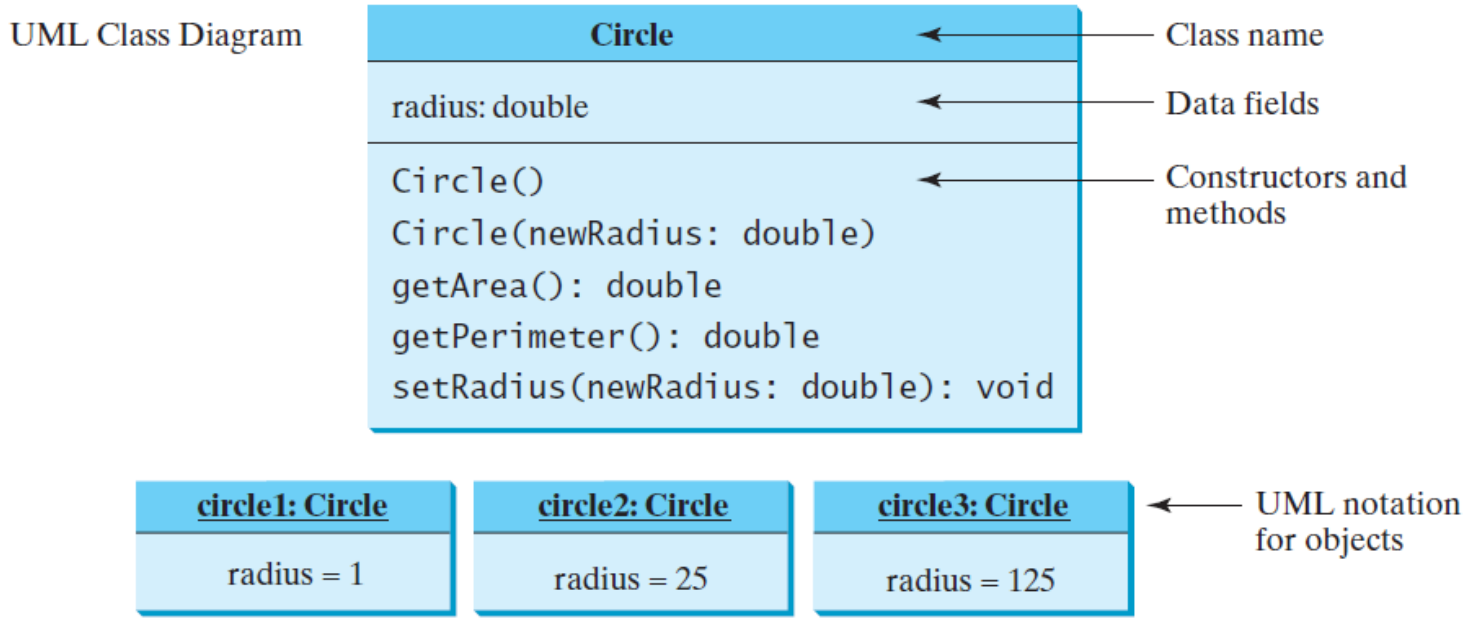



**FIGURE 9.4** Classes and objects can be represented using UML notation.

the data field is denoted as

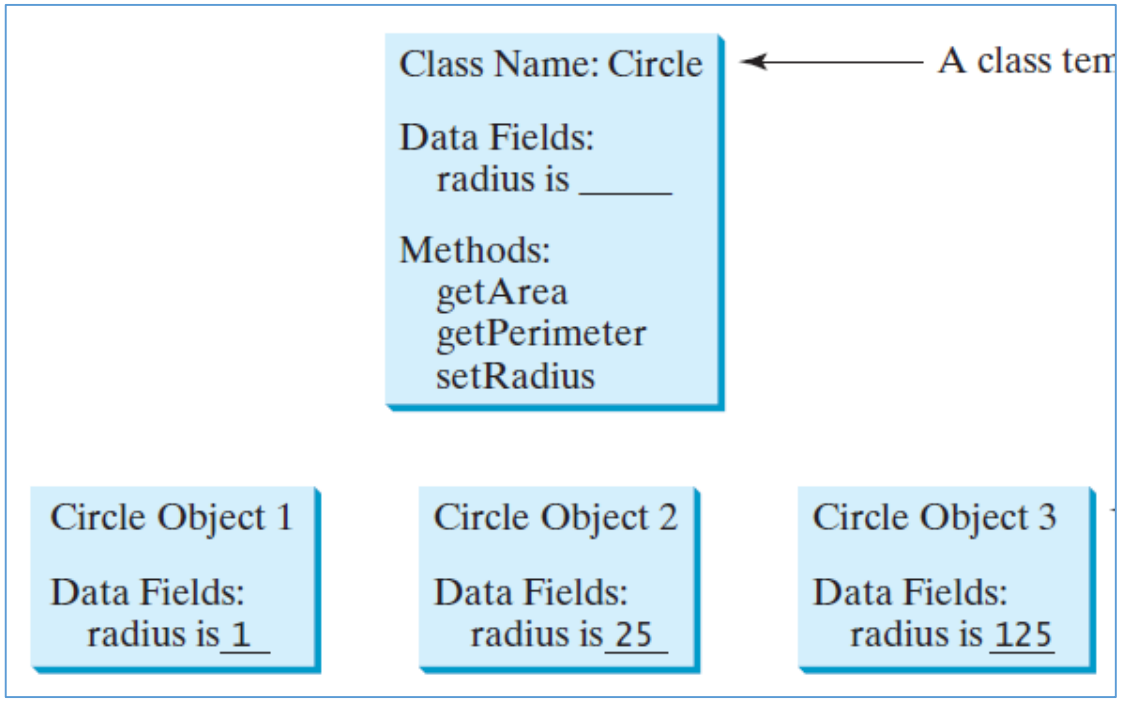
```
dataFieldName: dataFieldType
```

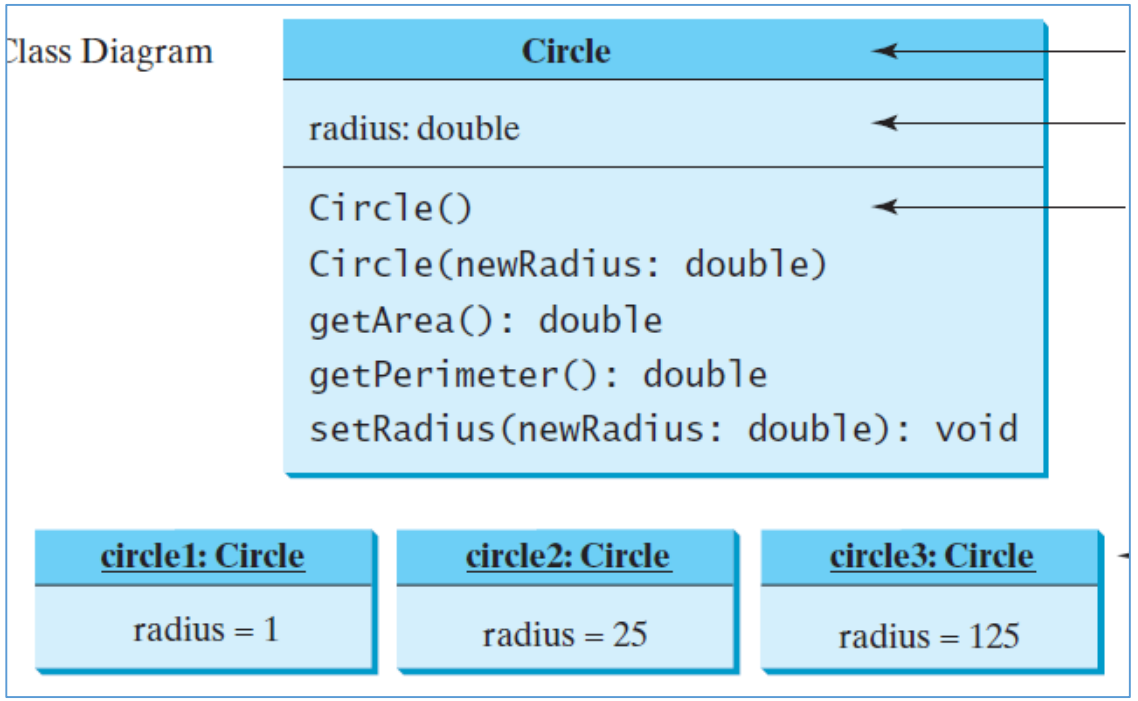
The constructor is denoted as

```
ClassName(parameterName: parameterType)
```

The method is denoted as

```
methodName(parameterName: parameterType): returnType
```




the data field is denoted as

```
dataFieldName: dataFieldType
```

The constructor is denoted as

```
ClassName(parameterName: parameterType)
```

The method is denoted as

```
methodName(parameterName: parameterType): returnType
```

# Örnek 1: Sınıf tanımlama, nesne oluşturma

```
1  public class TestBasitCember {
2      /** Main metot */
3      public static void main(String[] args) {
4      // Yaricapi 1 olan cember olusturma
5      BasitCember cember1 = new BasitCember();
6      System.out.println("Yaricapi " + cember1.yaricap
7      + " olan cemberin alani: " + cember1.getAlan());
8
9      // Yaricapi 25 olan cember olusturma
10     BasitCember cember2 = new BasitCember(25);
11     System.out.println("Yaricapi " + cember2.yaricap
12     + " olan cemberin alani: " + cember2.getAlan());
13
14     // Yaricapi 125 olan cember olusturma
15     BasitCember cember3 = new BasitCember(125);
16     System.out.println("Yaricapi " + cember3.yaricap
17     + " olan cemberin alani: " + cember3.getAlan());
18
19     // Cember yaricapi degistirme
20     cember2.yaricap = 100; // veya cember2.setRadius(100)
21     System.out.println("Yaricapi " + cember2.yaricap
22     + " olan cemberin alani: " + cember2.getAlan());
23     }
24 }
25
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestBasitCember
Yaricapi 1.0 olan cemberin alani: 3.141592653589793
Yaricapi 25.0 olan cemberin alani: 1963.4954084936207
Yaricapi 125.0 olan cemberin alani: 49087.385212340516
Yaricapi 100.0 olan cemberin alani: 31415.926535897932
```

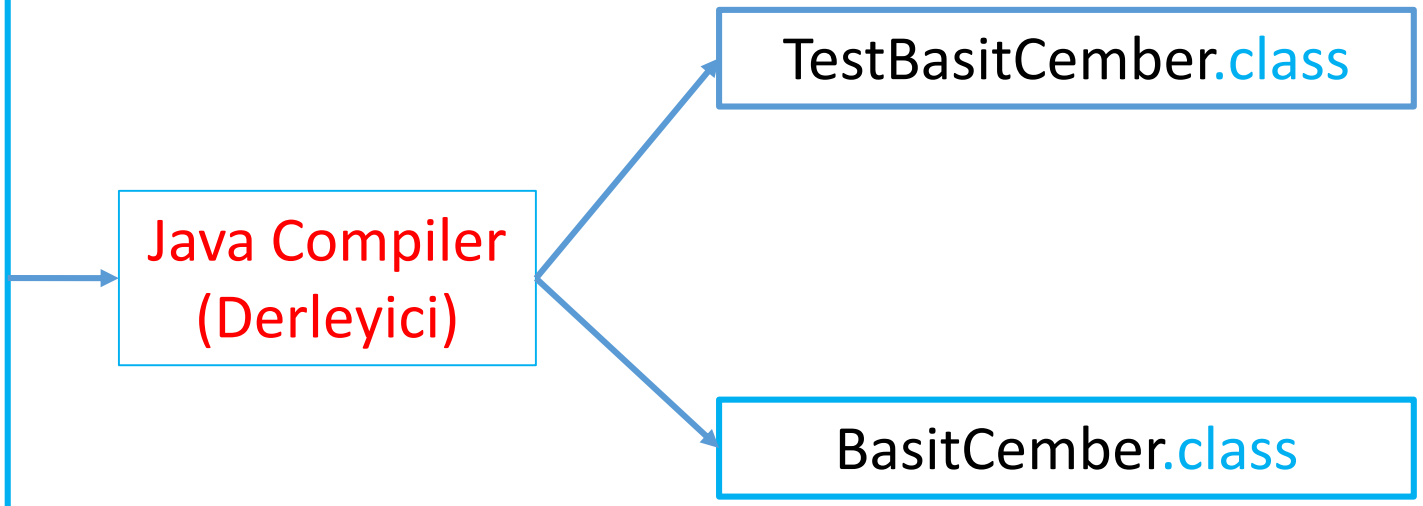
İki sınıfı tek bir dosyaya koyabiliriz, ancak dosyadaki yalnızca bir sınıf public class (genel sınıf) olabilir. Ayrıca, genel sınıf dosya adıyla aynı ada sahip olmalıdır.

# Örnek 1: Sınıf tanımlama, nesne oluşturma

İki sınıfı tek bir dosyaya koyabiliriz, ancak dosyadaki yalnızca bir sınıf public class (genel sınıf) olabilir. Ayrıca, genel sınıf dosya adıyla aynı ada sahip olmalıdır.

TestBasitCember.java dosyasını derlediğimiz zaman TestBasitCember.class ve BasitCember.class adlarında iki sınıf dosyası oluşturulur (üretilir).

```
// Dosya TestBasitCember.java
public class TestBasitCember {
    ...
}

class BasitCember {
    ...
}
```



# Örnek 1: Sınıf tanımlama, nesne oluşturma

```
26  // İki yapılandırıcı ile Cember sınıfı tanimalama
27  class BasitCember {
28      double yaricap;
29
30      /** Yaricapi 1 olan bir cember yapilandir */
31      BasitCember() {
32      yaricap = 1;
33      }
34
35      /** Yaricapi belirlenebilen bir cember yapilandir */
36      BasitCember(double yeniYaricap) {
37      yaricap = yeniYaricap;
38      }
39
40      /** Bu cemberin alanini geri döndür */
41      double getAlan() {
42      return yaricap * yaricap * Math.PI;
43      }
44
45      /** Bu cemberin cevresini geri döndür */
46      double getCevre() {
47      return 2 * yaricap * Math.PI;
48      }
49
50      /** Bu cember icin yeni bir yaricap ata */
51      void setYaricap(double yeniYaricap) {
52      yaricap = yeniYaricap;
53      }
54  }
```

Sınıflar, nesnelerin tanımlarıdır ve nesneler sınıflardan oluşturulur.

# Örnek 1: Sınıf tanımlama, nesne oluşturma

```java
public class TestBasitCember {
    /** Main metot */
    public static void main(String[] args) {
    // Yaricapi 1 olan cember olusturma
    BasitCember cember1 = new BasitCember();
    System.out.println("Yaricapi " + cember1.yaricap
    + " olan cemberin alani: " + cember1.getAlan());

    // Yaricapi 25 olan cember olusturma
    BasitCember cember2 = new BasitCember(25);
    System.out.println("Yaricapi " + cember2.yaricap
    + " olan cemberin alani: " + cember2.getAlan());

    // Yaricapi 125 olan cember olusturma
    BasitCember cember3 = new BasitCember(125);
    System.out.println("Yaricapi " + cember3.yaricap
    + " olan cemberin alani: " + cember3.getAlan());

    // Cember yaricapi degistirme
    cember2.yaricap = 100; // veya cember2.setRadius(100)
    System.out.println("Yaricapi " + cember2.yaricap
    + " olan cemberin alani: " + cember2.getAlan());
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestBasitCember
Yaricapi 1.0 olan cemberin alani: 3.141592653589793
Yaricapi 25.0 olan cemberin alani: 1963.4954084936207
Yaricapi 125.0 olan cemberin alani: 49087.385212340516
Yaricapi 100.0 olan cemberin alani: 31415.926535897932
```

```java
// İki yapılandırıcı ile Cember sınıfı tanimalama
class BasitCember {
    double yaricap;

    /** Yaricapi 1 olan bir cember yapilandir */
    BasitCember() {
    yaricap = 1;
    }

    /** Yaricapi belirlenebilen bir cember yapilandir */
    BasitCember(double yeniYaricap) {
    yaricap = yeniYaricap;
    }

    /** Bu cemberin alanini geri döndür */
    double getAlan() {
    return yaricap * yaricap * Math.PI;
    }

    /** Bu cemberin cevresini geri döndür */
    double getCevre() {
    return 2 * yaricap * Math.PI;
    }

    /** Bu cember icin yeni bir yaricap ata */
    void setYaricap(double yeniYaricap) {
    yaricap = yeniYaricap;
    }
}
```

# Örnek 2:

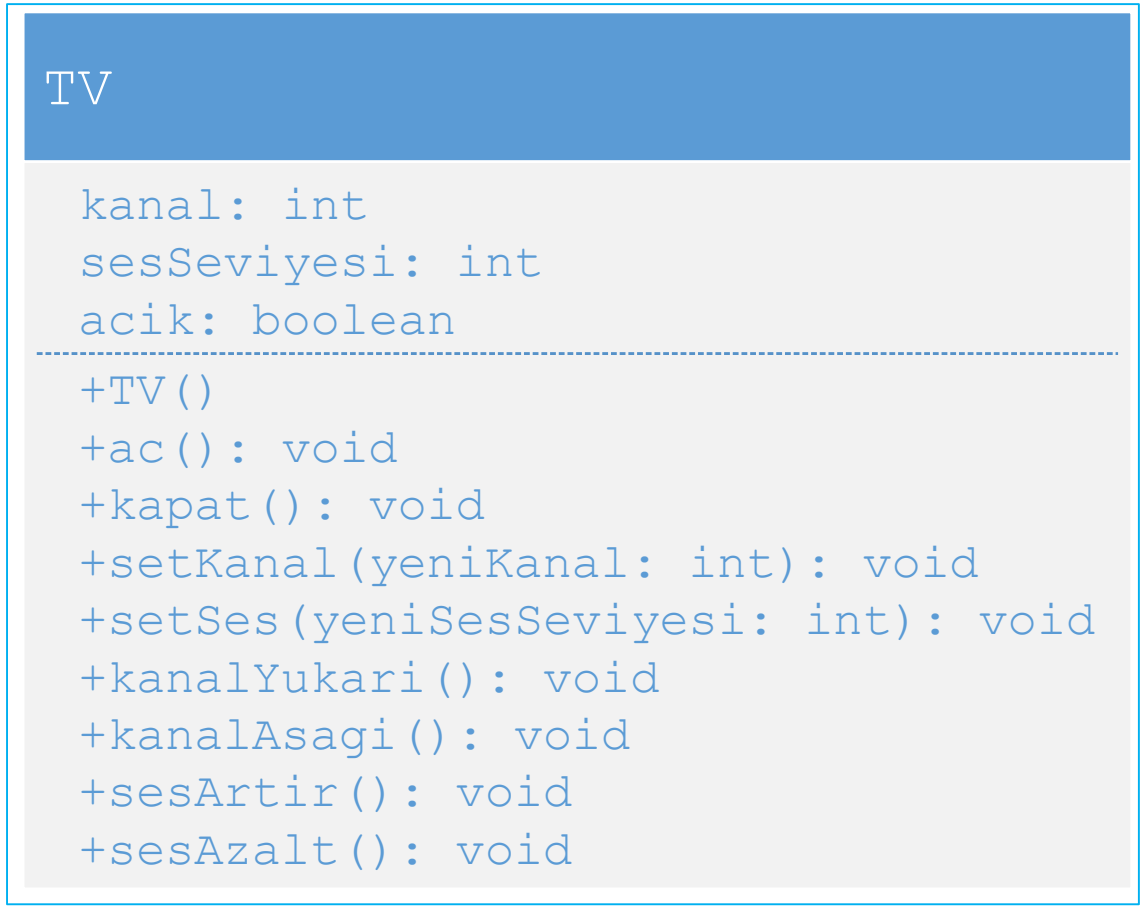
Java programları yazmanın birçok yolu vardır. Örnek 1'deki iki sınıfı aşağıda gösterildiği gibi tek bir sınıfta birleştirebiliriz.

```
1  public class BasitCemberr {
2      /** Main metot */
3      public static void main(String[] args) {
4      // Yaricapi 1 olan cember olusturma
5      BasitCemberr cember1 = new BasitCemberr();
6      System.out.println("Yaricapi " + cember1.yaricap
7      + " olan cemberin alani: " + cember1.getAlan());
8
9      // Yaricapi 25 olan cember olusturma
10     BasitCemberr cember2 = new BasitCemberr(25);
11     System.out.println("Yaricapi " + cember2.yaricap
12     + " olan cemberin alani: " + cember2.getAlan());
13
14     // Yaricapi 125 olan cember olusturma
15     BasitCemberr cember3 = new BasitCemberr(125);
16     System.out.println("Yaricapi " + cember3.yaricap
17     + " olan cemberin alani: " + cember3.getAlan());
18
19     // Cember yaricapi degistirme
20     cember2.yaricap = 100; // veya cember2.setRadius(100)
21     System.out.println("Yaricapi " + cember2.yaricap
22     + " olan cemberin alani: " + cember2.getAlan());
23     }
24
25     double yaricap;
```

```
26
27     /** Yaricapi 1 olan bir cember yapilandir */
28     BasitCemberr() {
29     yaricap = 1;
30     }
31
32     /** Yaricapi belirlenebilen bir cember yapilandir */
33     BasitCemberr(double yeniYaricap) {
34     yaricap = yeniYaricap;
35     }
36
37     /** Bu cemberin alanini geri döndür */
38     double getAlan() {
39     return yaricap * yaricap * Math.PI;
40     }
41
42     /** Bu cemberin cevresini geri döndür */
43     double getCevre() {
44     return 2 * yaricap * Math.PI;
45     }
46
47     /** Bu cember icin yeni bir yaricap ata */
48     void setYaricap(double yeniYaricap) {
49     yaricap = yeniYaricap;
50     }
51 }
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java BasitCemberr
Yaricapi 1.0 olan cemberin alani: 3.141592653589793
Yaricapi 25.0 olan cemberin alani: 1963.4954084936207
Yaricapi 125.0 olan cemberin alani: 49087.385212340516
Yaricapi 100.0 olan cemberin alani: 31415.926535897932
```

# Örnek 3: TV sınıfı TV kümesini modeller (UML)

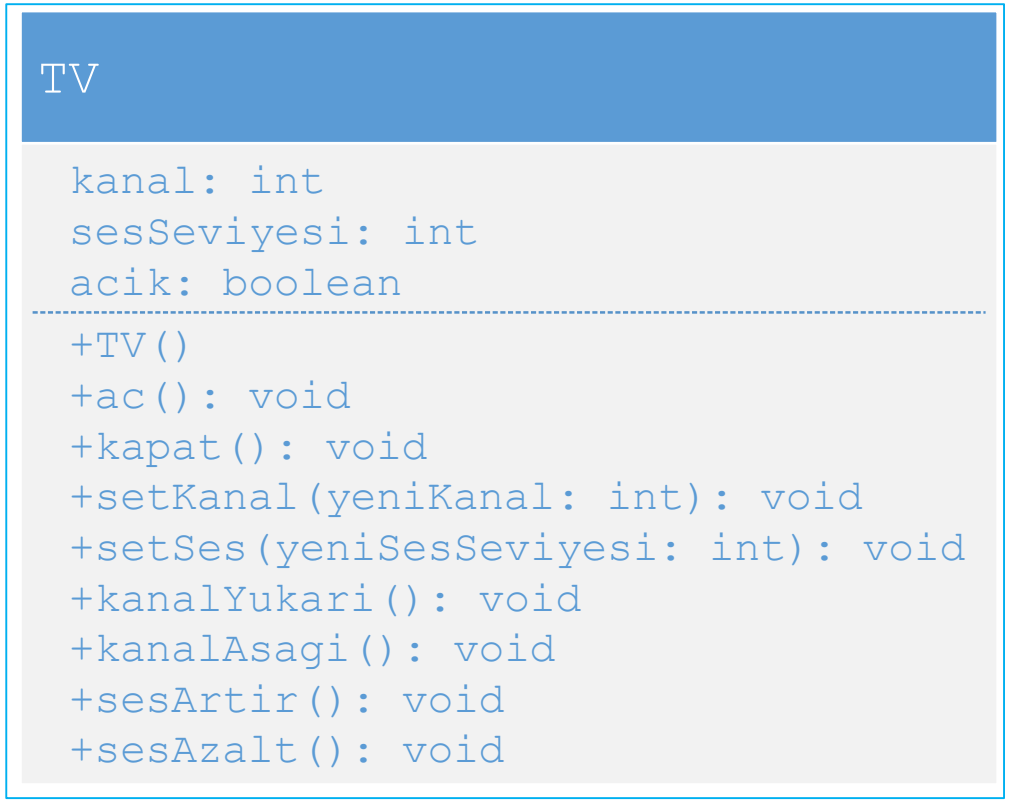



Bu TV'deki mevcut kanal (1-120 kanal).
Bu TV'deki mevcut ses seviyesi (1-7 seviye)
Bu TV'nin açık/kapalı olduğunu gösterir.

Varsayılan TV nesnesi oluştur.
Bu TV'yi aç.
Bu TV'yi kapat.
Bu TV için yeni kanal ata.
Bu TV için yeni ses seviyesi ata.
Kanal numarasını 1 artır.
Kanal numarasını 1 azalt.
Ses seviyesini 1 artır.
Ses seviyesini 1 azalt.

```java
public class TV {
    int kanal = 1; // Varsayılan kanal 1
    int sesSeviyesi = 1; // Varsayılan ses seviyesi 1
    boolean acik = false; // TV kapalı

    public TV() {
    }
    public void ac() {
    acik = true;
    }
    public void kapat() {
    acik = false;
    }
    public void setKanal(int yeniKanal) {
    if (acik && yeniKanal >= 1 && yeniKanal <= 120)
    kanal = yeniKanal;
    }
    public void setSes(int yeniSesSeviyesi) {
    if (acik && yeniSesSeviyesi >= 1 && yeniSesSeviyesi <= 7)
    sesSeviyesi = yeniSesSeviyesi;
    }
    public void kanalYukari() {
    if (acik && kanal < 120)
    kanal++;
    }
    public void kanalAsagi() {
    if (acik && kanal > 1)
    kanal--;
    }
    public void sesArtir() {
    if (acik && sesSeviyesi < 7)
    sesSeviyesi++;
    }
    public void sesAzalt() {
    if (acik && sesSeviyesi > 1)
    sesSeviyesi--;
    }
}
```

# Örnek 3: TV sınıfı TV kümesini modeller (UML)



# Örnek 3: TV sınıfını kullanan TestTV programı

```
1  public class TestTV {
2      public static void main(String[] args) {
3          TV tv1 = new TV();
4          tv1.ac();
5          tv1.setKanal(30);
6          tv1.setSes(3);
7
8          TV tv2 = new TV();
9          tv2.ac();
10         tv2.kanalYukari();
11         tv2.kanalYukari();
12         tv2.sesArtir();
13
14         System.out.println("tv1'nin kanali: " + tv1.kanal
15         + " ve ses seviyesi: " + tv1.sesSeviyesi);
16         System.out.println("tv2'nin kanali: " + tv2.kanal
17         + " ve ses seviyesi: " + tv2.sesSeviyesi);
18     }
19 }
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestTV
tv1'nin kanali: 30 ve ses seviyesi: 3
tv2'nin kanali: 3 ve ses seviyesi: 2
```

# Yapıcı kullanarak nesneleri oluşturma

new operatörü (işleci) kullanarak bir nesne oluşturmak için bir yapıcı çağrılır.

Yapıcılar özel bir metot türüdür. 3 özelliği vardır:

1. Bir yapıcı, sınıfın kendisiyle aynı isme sahip olmalıdır.

2. Yapıcının dönüş türü yoktur (void bile yok).

3. Yapıcılar, bir nesne oluşturulduğunda new operatörü kullanılarak çağrılır.
   Yapıcılar, nesneleri başlatma rolünü oynar.

# Yapıcı kullanarak nesneleri oluşturma

```
public void Cember() {
}
```

Yapıcı olabilmesi için void kullanmamalıyız.

```
new Cember()

new Cember(25)
```

# Referans değişkenleri, referans tipleri

```
SinifAdi   nesneRefDegiskeni;

Cember benimCember;

benimCember = new Cember();
```

```
SinifAdi   nesneRefDegiskeni = new SinifAdi();

Cember benimCember = new Cember();
```

# Nesnelere referans değişkenler aracılığıyla erişim

Bir nesnenin verilerine ve metotlarına, nesnenin referans değişkeni aracılığıyla nokta (.) operatörü ile erişilebilir.

```
nesneRefDeg.veriAlani;

nesneRefDeg.metot(argümanlar);
```

```
benimCember.yaricap;

benimCember.getAlan(20);

benimCember.getCevre(20);
```

# anonymous object (anonim nesne)

```
new Cember();

veya

System.out.println("Alan: " + new Cember(5).getAlan());
```

# `null` value (null değeri)

```
class Ogrenci {
        String isim; // isim varsayılan değeri null
        int yas; // yas varsayılan değeri 0
        boolean yetenekliMi; // yetenekliMi varsaılan değeri false
        char cinsiyet; // cinsiyet varsayılan değeri '\u0000'
}
```

# `null` value (null değeri)

```
class Ogrenci {
        String isim;
        int yas;
        boolean yetenekliMi;
        char cinsiyet;
}

class TestOgrenci {
        public static void main(String[] args) {
                Ogrenci ogrenci = new Ogrenci();
                System.out.println("İsmi? " + ogrenci.isim);
                System.out.println("Yasi? " + ogrenci.yas);
                System.out.println("Yetenekli mi? " + ogrenci.yetenekliMi);
                System.out.println("Cinsiyeti? " + ogrenci.cinsiyet);
        }
}
```

# Örnek 4:

```java
public class TestXY {
    public static void main(String[] args) {
        int x; // x'in varsayilan degeri yok
        String y; // y'nin varsayilan degeri yok
        System.out.println("x: " + x);
        System.out.println("y: " + y);
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac TestXY.java
TestXY.java:5: error: variable x might not have been initialized
                System.out.println("x: " + x);
                                           ^
TestXY.java:6: error: variable y might not have been initialized
                System.out.println("y: " + y);
                                           ^
2 errors
```

# İlkel tip ve referans tipleri değişkenleri arasındaki fark

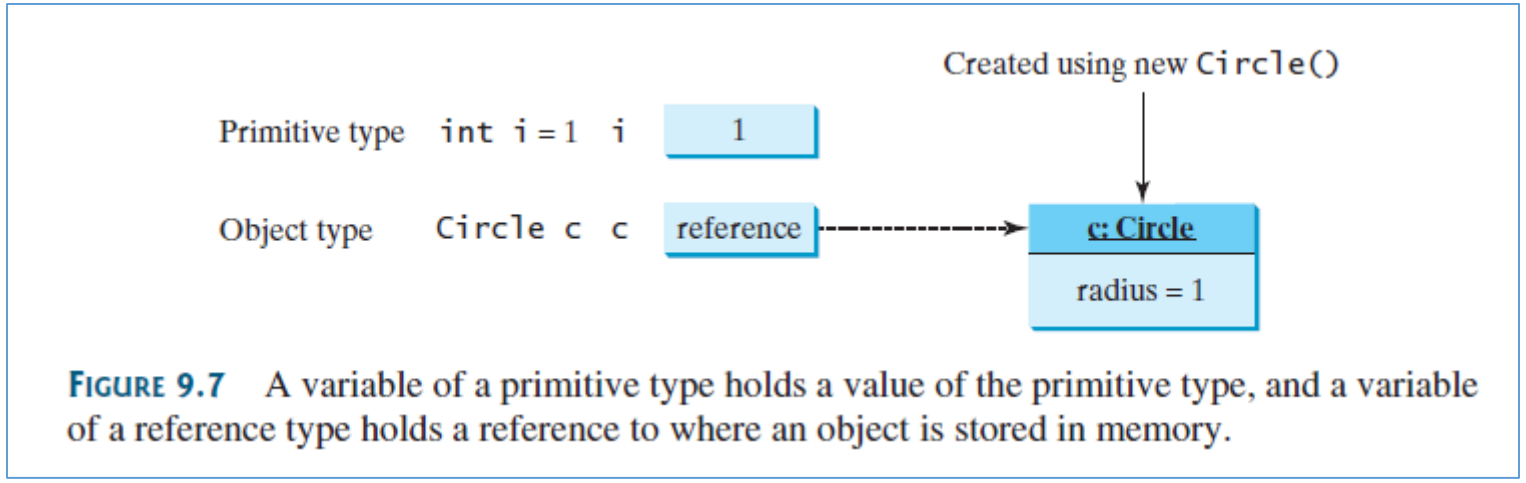



**FIGURE 9.7** A variable of a primitive type holds a value of the primitive type, and a variable of a reference type holds a reference to where an object is stored in memory.

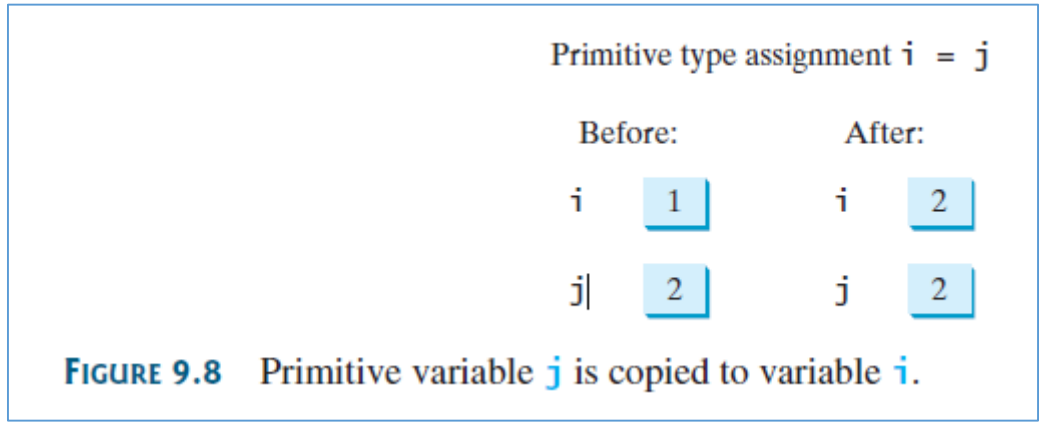



**FIGURE 9.8** Primitive variable `j` is copied to variable `i`.

# garbage collection

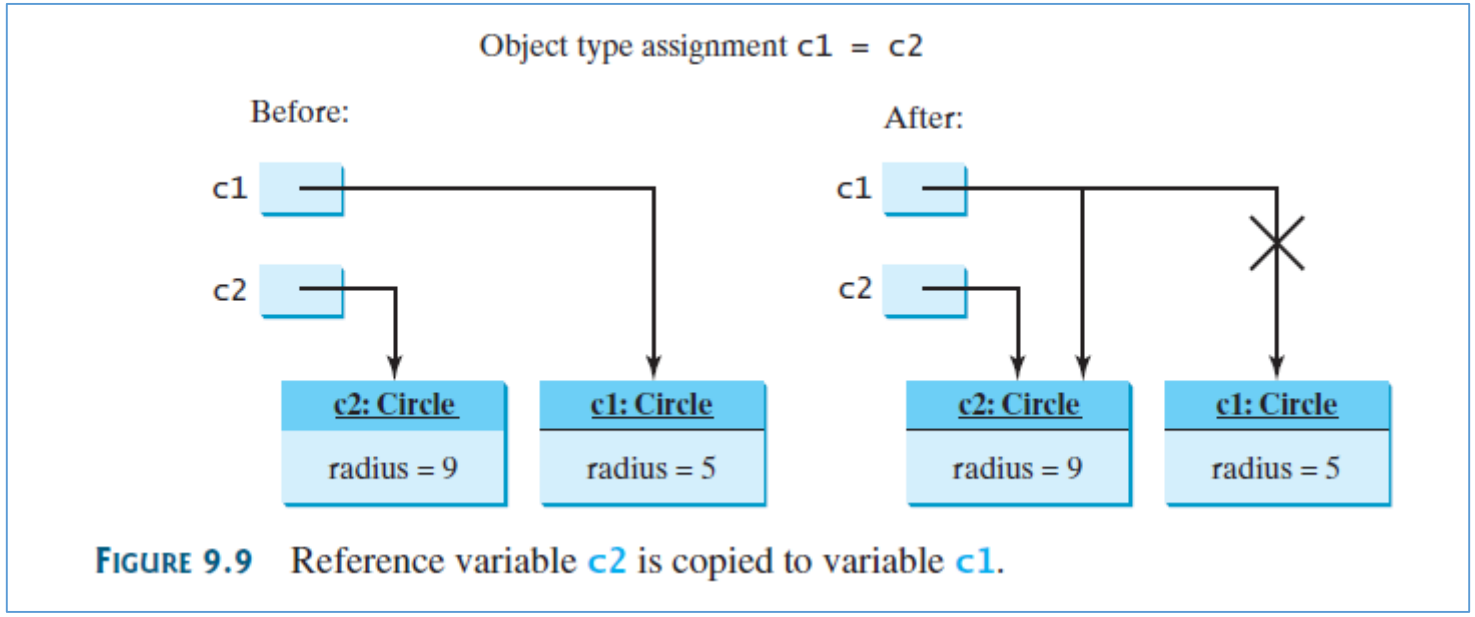



**FIGURE 9.9** Reference variable c2 is copied to variable c1.

# Ödev 1: Java kütüphanesi sınıfları: The Date class

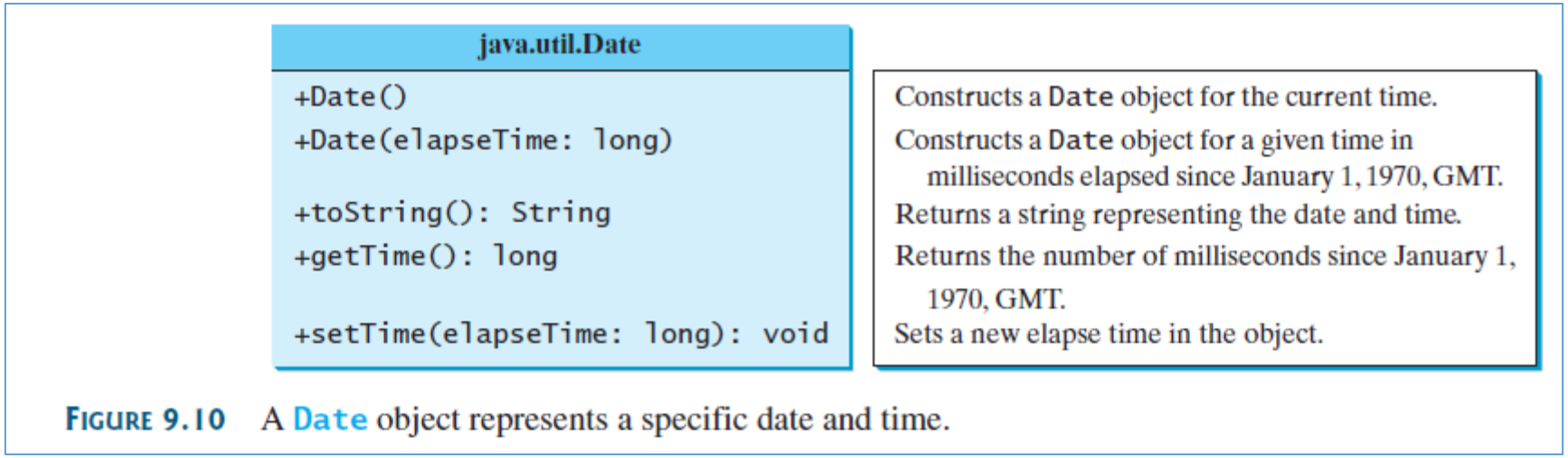



**FIGURE 9.10** A `Date` object represents a specific date and time.

# Ödev 1: Java kütüphanesi sınıfları: The Date class

```
java.util.Date date = new java.util.Date();
System.out.println("The elapsed time since Jan 1, 1970 is " +
  date.getTime() + " milliseconds");
System.out.println(date.toString());
```

```
The elapsed time since Jan 1, 1970 is 1324903419651 milliseconds
Mon Dec 26 07:43:39 EST 2011
```

# Ödev 2: Java kütüphanesi sınıfları: The Random class

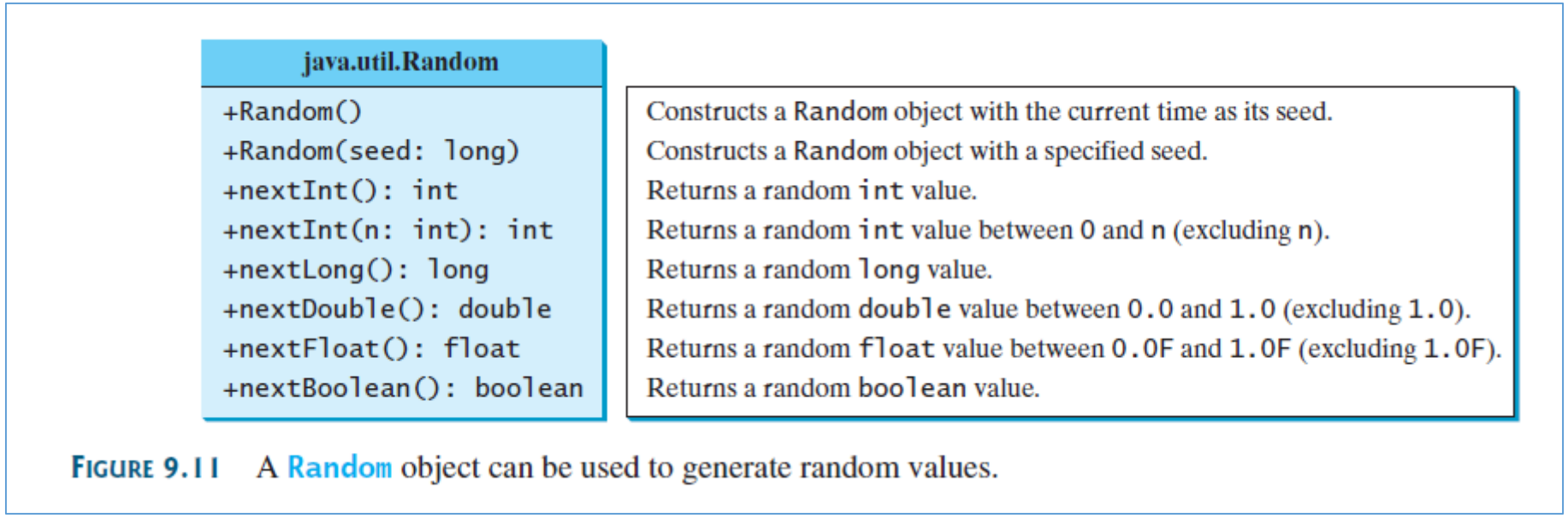



**FIGURE 9.11** A Random object can be used to generate random values.

# Ödev 2: Java kütüphanesi sınıfları: The Random class

```
Random random1 = new Random(3);
System.out.print("From random1: ");
for (int i = 0; i < 10; i++)
  System.out.print(random1.nextInt(1000) + " ");

Random random2 = new Random(3);
System.out.print("\nFrom random2: ");
for (int i = 0; i < 10; i++)
  System.out.print(random2.nextInt(1000) + " ");
```

| java.util.Random |
| --- |
| +Random() |
| +Random(seed: long) |
| +nextInt(): int |
| +nextInt(n: int): int |
| +nextLong(): long |
| +nextDouble(): double |
| +nextFloat(): float |
| +nextBoolean(): boolean |

The code generates the same sequence of random `int` values:

```
From random1: 734 660 210 581 128 202 549 564 459 961
From random2: 734 660 210 581 128 202 549 564 459 961
```

# Ödev 3: Java kütüphanesi sınıfları: The Point2D class

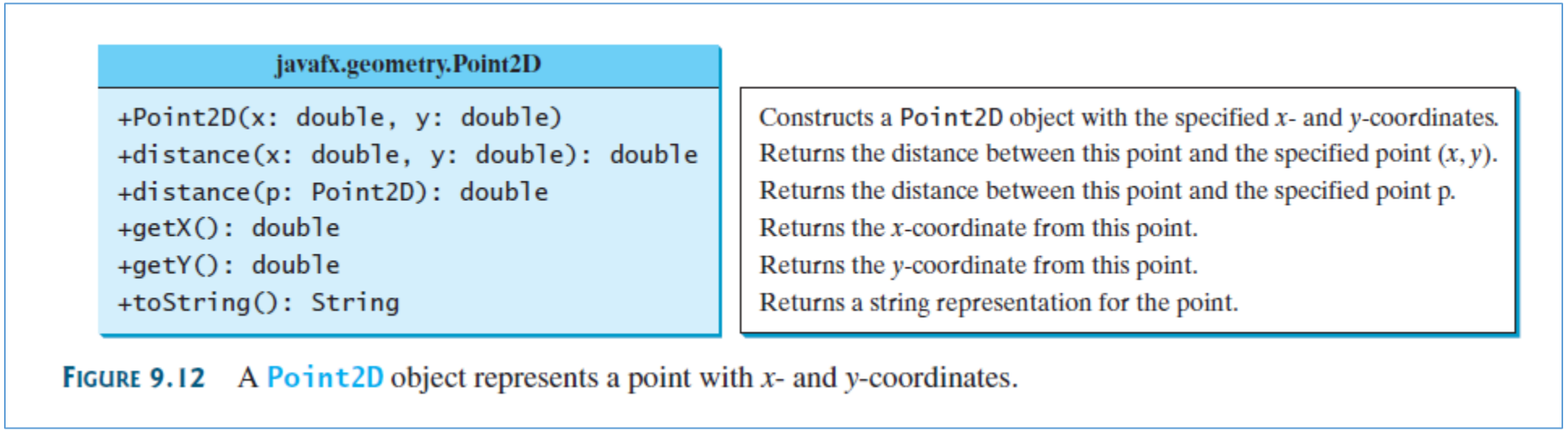



**FIGURE 9.12** A `Point2D` object represents a point with *x*- and *y*-coordinates.

# Ödev 3: Java kütüphanesi sınıfları: The Point2D class

```
1  import java.util.Scanner;
2  import javafx.geometry.Point2D;
3
4  public class TestPoint2D {
5    public static void main(String[] args) {
6      Scanner input = new Scanner(System.in);
7
8      System.out.print("Enter point1's x-, y-coordinates: ");
9      double x1 = input.nextDouble();
10     double y1 = input.nextDouble();
11     System.out.print("Enter point2's x-, y-coordinates: ");
12     double x2 = input.nextDouble();
13     double y2 = input.nextDouble();
14
15     Point2D p1 = new Point2D(x1, y1);
16     Point2D p2 = new Point2D(x2, y2);
17     System.out.println("p1 is " + p1.toString());
18     System.out.println("p2 is " + p2.toString());
19     System.out.println("The distance between p1 and p2 is " +
20       p1.distance(p2));
21   }
22 }
```

| javafx.geometry.Point2D |
| --- |
| +Point2D(x: double, y: double) |
| +distance(x: double, y: double): double |
| +distance(p: Point2D): double |
| +getX(): double |
| +getY(): double |
| +toString(): String |

```
Enter point1's x-, y-coordinates: 1.5 5.5 Enter
Enter point2's x-, y-coordinates: -5.3 -4.4 Enter
p1 is Point2D [x = 1.5, y = 5.5]
p2 is Point2D [x = -5.3, y = -4.4]
The distance between p1 and p2 is 12.010412149464313
```

# 3. Ders: JAVA ile Nesne Yönelimli Programlama
# Nesne Yönelimli Düşünme (Object-Oriented Thinking)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama (Ders: 11-20) video.

    - Ders 11: Java İsimlendirme Kuralları (izle)
    - Ders 12: Java Dilinde Sınıf ve Nesneler Nasıl Kodlanır? (izle)
    - Ders 13: Örnek Kodlar ile Nesne ve Sınıflar (izle)
    - Ders 14: Java Programları Yazarken Kullanılan Yaygın Yaklaşım (izle)
    - Ders 15: Nesneleri Hazırlamanın Farklı Yolları (izle)
    - Ders 16 Anonim Nesneler (Anonymous Object) (izle)
    - Ders 17: Örnek Bir Uygulama (izle)
    - Ders 18: Yapıcı Metotlar (Constructor Methods) (izle)
    - Ders 19: Java'da Constructor Aşırı Yüklemesi (izle)
    - Ders 20: Static Anahtar Sözcüğü (izle)

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# Sınıf Soyutlama ve Kapsülleme (Class Abstraction and Encapsulation)

Sınıf soyutlaması, bir sınıf uygulamasının (implement) sınıfın kullanımından ayrılmasıdır.

Uygulamanın (Implementation) ayrıntıları kapsüllenir ve kullanıcıdan gizlenir.

Bu, sınıf kapsülleme olarak bilinir.

# Sınıf Soyutlama ve Kapsülleme (Class Abstraction and Encapsulation)

Java, birçok soyutlama düzeyi sağlar ve sınıf soyutlaması, sınıf uygulamasını (implementation) sınıfın nasıl kullanıldığından ayırır.

Bir sınıfın yaratıcısı, sınıfın işlevlerini tanımlar ve kullanıcının sınıfın nasıl kullanılabileceğini bilmesini sağlar.

Sınıfın dışından erişilebilen metotların ve alanların koleksiyonu, bu üyelerin nasıl davranmasının beklendiğinin açıklamasıyla birlikte, sınıfın sözleşmesi olarak hizmet eder.

# Sınıf Soyutlama ve Kapsülleme (Class Abstraction and Encapsulation)

Şekil 10.1'de gösterildiği gibi, sınıfın kullanıcısının sınıfın nasıl uygulandığını bilmesine gerek yoktur.

Uygulamanın ayrıntıları kapsüllenir ve kullanıcıdan gizlenir. Buna sınıf kapsülleme denir.

Örneğin, bir Cember nesnesi oluşturabilir ve alanın nasıl hesaplandığını bilmeden dairenin alanını bulabilirsiniz.

Bu nedenle, bir sınıf aynı zamanda soyut veri türü (ADT) olarak da bilinir.

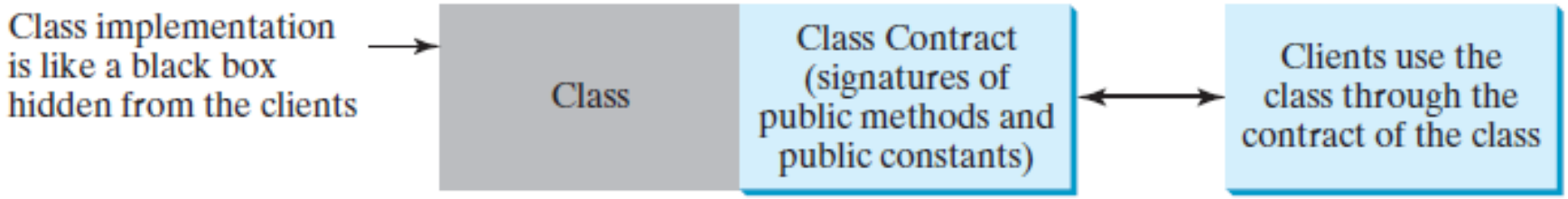



**FIGURE 10.1** Class abstraction separates class implementation from the use of the class.

# Sınıf Soyutlama ve Kapsülleme (Class Abstraction and Encapsulation)

Sınıf soyutlaması ve kapsülleme aynı madalyonun iki yüzüdür.

Pek çok gerçek hayat örneği, sınıf soyutlaması kavramını göstermektedir. Örneğin, bir bilgisayar sistemi oluşturmayı düşünelim. Kişisel bilgisayarınızda birçok bileşen vardır - bir CPU, bellek, disk, ana kart, fan vb. Her bileşen, özellikleri ve metotları olan bir nesne olarak görülebilir.

Bileşenlerin birlikte çalışmasını sağlamak için, yalnızca her bir bileşenin nasıl kullanıldığını ve diğerleriyle nasıl etkileşime girdiğini bilmeniz gerekir.

Bileşenlerin kendi içinde nasıl çalıştığını bilmenize gerek yoktur. İç uygulama (implementation) kapsüllenir ve sizden gizlenir.

Bir bileşenin nasıl uygulandığını (implement) bilmeden bir bilgisayar oluşturabilirsiniz.

# Sınıf Soyutlama ve Kapsülleme (Class Abstraction and Encapsulation)

Bilgisayar sistemi benzetmesi, nesne yönelimli yaklaşımı tam olarak yansıtır.

Her bileşen, bileşen için sınıfın bir nesnesi olarak görülebilir.

Örneğin, fan boyutu ve hızı gibi özellikler ve başlatma ve durdurma gibi metotlarla bir bilgisayarda kullanılmak üzere her türlü fanı modelleyen bir sınıfınız olabilir.

Belirlenmiş bir fan, bu sınıfın belirli özellik değerlerine sahip bir örneğidir.

# Nesnelerle Düşünme

Prosedüral programlama paradigması metotları tasarlamaya odaklanır.

Nesne yönelimli paradigma, verileri ve yöntemleri birlikte nesneler halinde birleştirir.

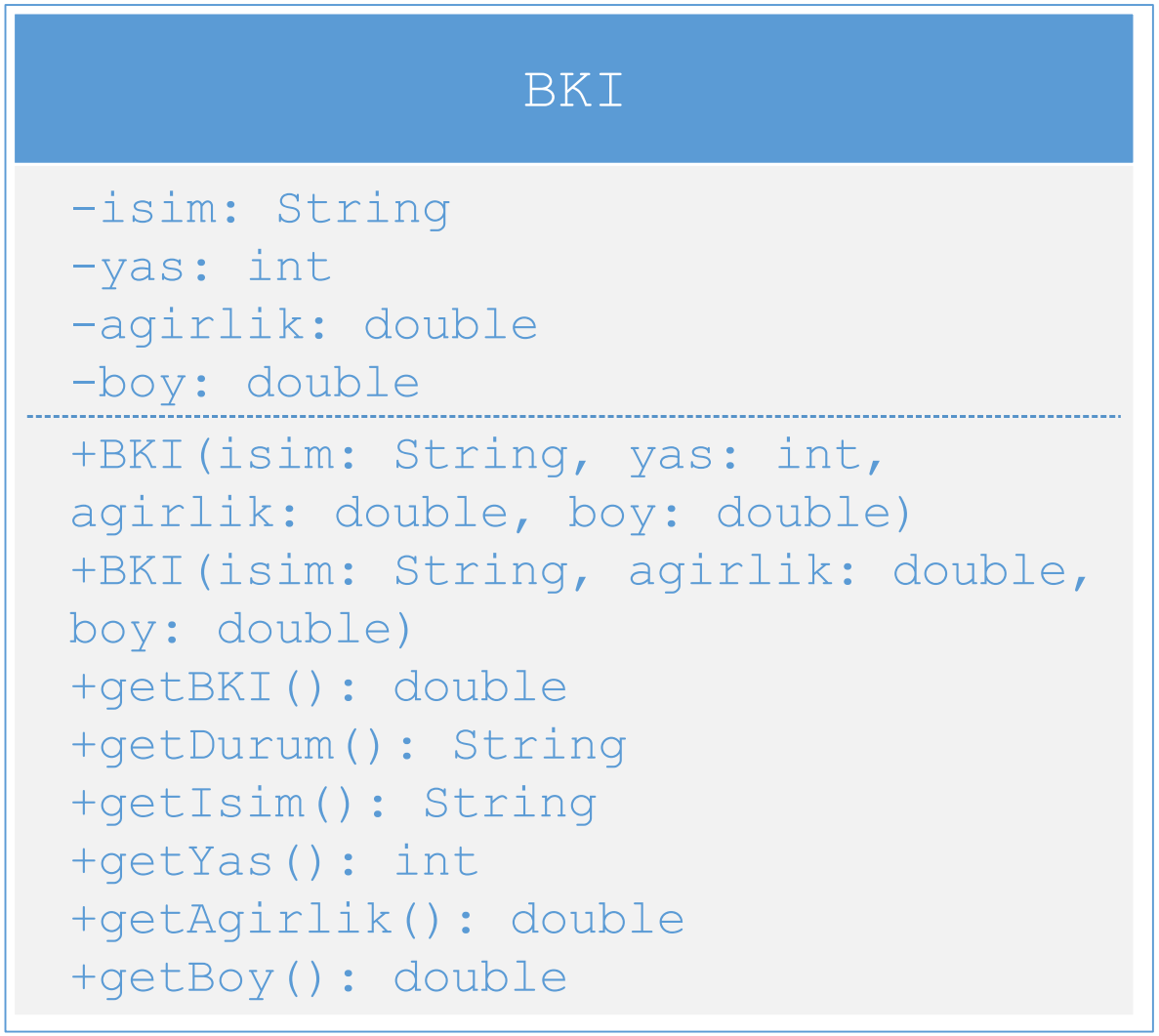
Nesneye yönelik paradigmayı kullanan yazılım tasarımı, nesnelere ve nesneler üzerindeki işlemlere odaklanır.

# BKI Hesapla ve Yorumla

```java
import java.util.Scanner;

public class BKIHesaplaveYorumla {
    public static void main(String[] args) {
        Scanner giris = new Scanner(System.in);

        System.out.print("Lutfen kilonuzu giriniz (kg): ");
        double agirlik = giris.nextDouble();
        System.out.print("Lutfen boyunuzu giriniz (m): ");
        double boy = giris.nextDouble();

        // BKI Hesapla
        double bki = agirlik /  (boy * boy);

        // Sonuçları göster
        System.out.println("BKI: " + bki);
        if (bki < 18.5)
            System.out.println("Zayif");
        else if (bki < 25)
            System.out.println("Normal");
        else if (bki < 30)
            System.out.println("Kilolu");
        else
            System.out.println("Obez");
    }
}
```

# BKI Sınıfı

BKI sınıfı, BKI bilgisini kapsüllüyor.




Kişinin ismi
Kişinin yaşı
Kişinin ağırlığı (kg)
Kişinin boyu (m)

BKI nesnesi oluştur (isim, yaş, ağırlık ve boy)
BKI nesnesi oluştur (isim, ağırlık ve boy)
BKI döndür.
Durum döndür.
İsim döndür.
Yaş döndür.
Ağırlık döndür.
Boy döndür.

# BKI Sınıfı kullanımı

```
1  public class BKISinifKullanimi {
2      public static void main(String[] args) {
3          BKI bki1 = new BKI("Yaman Akbulut", 43, 84, 1.78);
4          System.out.println(bki1.getIsim() + " icin BKI: "
5          + bki1.getBKI() + " " + bki1.getDurum());
6
7          BKI bki2 = new BKI("Diger kisi", 60, 1.70);
8          System.out.println(bki2.getIsim() + " icin BKI: "
9          + bki2.getBKI() + " " + bki2.getDurum());
10     }
11 }
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac BKISinifKullanimi.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java BKISinifKullanimi
Yaman Akbulut icin BKI: 26.51 Kilolu
Diger kisi icin BKI: 20.76 Normal
```

# BKI Sınıfı implement

```java
public class BKI {
    private String isim;
    private int yas;
    private double agirlik; // kg
    private double boy; // m

    public BKI(String isim, int yas, double agirlik, double boy) {
        this.isim = isim;
        this.yas = yas;
        this.agirlik = agirlik;
        this.boy = boy;
    }

    public BKI(String isim, double agirlik, double boy) {
        this(isim, 20, agirlik, boy);
    }

    public double getBKI() {
        double bki = agirlik / (boy * boy);
        return Math.round(bki * 100) / 100.0;
    }

    public String getDurum() {
        double bki = getBKI();
        if (bki < 18.5)
            return "Zayif";
        else if (bki < 25)
            return "Normal";
        else if (bki < 30)
            return "Kilolu";
        else
            return "Obez";
    }

    public String getIsim() {
        return isim;
    }

    public int getYas() {
        return yas;
    }

    public double getAgirlik() {
        return agirlik;
    }

    public double getBoy() {
        return boy;
    }
}
```

# Prosedür vs Nesne Yönelimli

Bu örnek, nesne yönelimli paradigmanın prosedüral paradigmaya göre avantajlarını göstermektedir.

Prosedüral paradigma, metotların tasarlanmasına odaklanır.

Nesne yönelimli paradigma, verileri ve metotları birlikte nesneler halinde birleştirir.

Nesneye yönelik paradigmayı kullanan yazılım tasarımı, nesnelere ve nesneler üzerindeki işlemlere odaklanır.

Nesne yönelimli yaklaşım, prosedüral paradigmanın gücünü, işlemlerle verileri nesnelere entegre eden ek bir boyutla birleştirir.

# Prosedür vs Nesne Yönelimli

Prosedürel programlamada, veriler üzerindeki veriler ve işlemler ayrıdır ve bu metodoloji verilerin metotlara aktarılmasını gerektirir.

Nesneye yönelik programlama, verileri ve bunlarla ilgili işlemleri bir nesneye yerleştirir.

Bu yaklaşım, prosedürel programlamanın doğasında bulunan birçok sorunu çözer.

Nesne yönelimli programlama yaklaşımı programları, tüm nesnelerin hem özniteliklerle hem de etkinliklerle ilişkilendirildiği gerçek dünyayı yansıtacak şekilde düzenler.

# Prosedür vs Nesne Yönelimli

Nesnelerin kullanılması, yazılımın yeniden kullanılabilirliğini artırır ve programların geliştirilmesini ve bakımını kolaylaştırır.

Java'da programlama nesneler açısından düşünmeyi içerir; bir Java programı, işbirliği yapan nesnelerin bir koleksiyonu olarak görülebilir.

# Örnek 1a:

```java
public class TestOgrenci {
    public static void main(String args[]){
        Ogrenci nesne1 = new Ogrenci(111,"Yaman Akbulut",5000f);
        Ogrenci nesne2 = new Ogrenci(222,"Fatih Ozkaynak",6000f);
        nesne1.bilgileriGoster();
        nesne2.bilgileriGoster();
    }
}
```

```java
public class Ogrenci {
    int ogrenciNo;
    String isim;
    float burs;

    Ogrenci(int ogrenciNo, String isim, float burs){
        ogrenciNo = ogrenciNo;
        isim = isim;
        burs = burs;
    }

    void bilgileriGoster(){
        System.out.println(ogrenciNo + " " + isim
        + " " + burs);
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac Ogrenci.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac TestOgrenci.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestOgrenci
0 null 0.0
0 null 0.0
```

# Örnek 1b:

```java
public class TestOgrenci {
    public static void main(String args[]){
        Ogrenci nesne1 = new Ogrenci(111,"Yaman Akbulut",5000f);
        Ogrenci nesne2 = new Ogrenci(222,"Fatih Ozkaynak",6000f);
        nesne1.bilgileriGoster();
        nesne2.bilgileriGoster();
    }
}
```

```java
public class Ogrenci {
    int ogrenciNo;
    String isim;
    float burs;

    Ogrenci(int ogrenciNo, String isim, float burs){
        this.ogrenciNo = ogrenciNo;
        this.isim = isim;
        this.burs = burs;
    }

    void bilgileriGoster(){
        System.out.println(ogrenciNo + " " + isim
        + " " + burs);
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac Ogrenci.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestOgrenci
111 Yaman Akbulut 5000.0
222 Fatih Ozkaynak 6000.0
```

# Örnek 1c:

```java
public class TestOgrenci {
    public static void main(String args[]){
        Ogrenci nesne1 = new Ogrenci(111,"Yaman Akbulut",5000f);
        Ogrenci nesne2 = new Ogrenci(222,"Fatih Ozkaynak",6000f);
        nesne1.bilgileriGoster();
        nesne2.bilgileriGoster();
    }
}
```

```java
public class Ogrenci {
    int ogrenciNo;
    String isim;
    float burs;

    Ogrenci(int o, String i, float b){
        ogrenciNo = o;
        isim = i;
        burs = b;
    }

    void bilgileriGoster(){
        System.out.println(ogrenciNo + " " + isim
        + " " + burs);
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac Ogrenci.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestOgrenci
111 Yaman Akbulut 5000.0
222 Fatih Ozkaynak 6000.0
```

# Ödev 1:

en, boy ve yukseklik özelliklerine sahip Kutu adında bir sınıf tasarlayınız.

Daha sonra TestKutu adında bir uygulama ile bir kutu örneğinin hacmini hesaplayınız.

# Ödev 2:

ogrenciNo, isim, burs ve ders özelliklerine sahip Ogrenci adında bir sınıf tasarlayınız. Sınıf içinde 2 tane yapıcı (kurucu) metot yazınız. Birisi 3 parametreli (ogrenciNo, isim, burs) diğeri 4 parametreli (ogrenciNo, isim, burs, ders) olsun. Ayrıca Ogrenci sınıfı bilgileriGoster() adında bir metota sahip olsun. Ogrenci sınıfına ait UML diyagramı çiziniz ve bu sınıfı java dilinde kodlayınız.

Daha sonra TestOgrenci adında bir sınıf ile (main metotu olan) Ogrenci sınıfını test ediniz. Ogrenci sınıfından 2 nesne oluşturunuz ve nesne1 ile 3 parametreli, nesne2 ile 4 parametreli yapıcıları kullanınız. Çalışan kod sonucunda ekran çıktılarını inceleyiniz.

# 4. Ders: JAVA ile Nesne Yönelimli Programlama
# Nesne Yönelimli Düşünme (Object-Oriented Thinking)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama (Ders: 14-25) video.

    - Ders 14: Java Programları Yazarken Kullanılan Yaygın Yaklaşım (izle)
    - Ders 15: Nesneleri Hazırlamanın Farklı Yolları (izle)
    - Ders 16: Anonim Nesneler (Anonymous Object) (izle)
    - Ders 17: Örnek Bir Uygulama (izle)
    - Ders 18: Yapıcı Metotlar (Constructor Methods) (izle)
    - Ders 19: Java'da Constructor Aşırı Yüklemesi (izle)
    - Ders 20: Static Anahtar Sözcüğü (izle)
    - Ders 21: Statik Değişkenler ile Program Sayacı (izle)
    - Ders 22: Statik Yöntemler (izle)
    - Ders 23: Statik Yöntemler Kullanılırken Dikkat Edilecek Unsurlar (izle)
    - Ders 24: this Anahtar Sözcüğünün Kullanımı I (izle)
    - Ders 25: this Anahtar Sözcüğünün Kullanımı II (izle)

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# Static anahtar sözcüğü

Bir sınıfın tüm örneklerinin (nesnelerinin) verileri paylaşmasını istiyorsanız, sınıf değişkenleri olarak da bilinen statik değişkenler kullanın.

Statik değişkenler, değişkenler için değerleri ortak bir bellek konumunda depolar. Bu ortak konum nedeniyle, bir nesne statik bir değişkenin değerini değiştirirse, aynı sınıftaki tüm nesneler etkilenir.

Java, statik değişkenlerin yanı sıra statik metotları da destekler. Statik metotlar, sınıfın bir örneğini oluşturmadan çağrılabilir.

# Static anahtar sözcüğü

Static (Statik) bir değişken sınıfın tüm nesneleri tarafından paylaşılır.

Statik bir metot, sınıfın örnek (nesne) üyelerine erişemez veya üyelerini kullanamaz.

BasitCember cember1 = new BasitCember();
BasitCember cember2 = new BasitCember(5);

cember1.yaricap
cember2.yaricap

```
26  // İki yapılandırıcı ile Cember sınıfı tanimalama
27  class BasitCember {
28      double yaricap;
29
30      /** Yaricapi 1 olan bir cember yapilandir */
31      BasitCember() {
32      yaricap = 1;
33      }
34
35      /** Yaricapi belirlenebilen bir cember yapilandir */
36      BasitCember(double yeniYaricap) {
37      yaricap = yeniYaricap;
38      }
39
40      /** Bu cemberin alanini geri döndür */
41      double getAlan() {
42      return yaricap * yaricap * Math.PI;
43      }
44
45      /** Bu cemberin cevresini geri döndür */
46      double getCevre() {
47      return 2 * yaricap * Math.PI;
48      }
49
50      /** Bu cember icin yeni bir yaricap ata */
51      void setYaricap(double yeniYaricap) {
52      yaricap = yeniYaricap;
53      }
54  }
```

# Örnek 1a: Static anahtar sözcüğü (yokken)

```java
public class Sayici {
    int sayac;

    Sayici(){
        sayac++;
        System.out.println(sayac);
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac Sayici.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac TestSayici.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestSayici
1
1
1
1
1
```

```java
public class TestSayici{
    public static void main(String arg[]){
        Sayici nesne1 = new Sayici();
        Sayici nesne2 = new Sayici();
        Sayici nesne3 = new Sayici();
        Sayici nesne4 = new Sayici();
        Sayici nesne5 = new Sayici();
    }
}
```

# Örnek 1b: Static anahtar sözcüğü (varken)

```java
public class Sayici {
    static int sayac;

    Sayici(){
        sayac++;
        System.out.println(sayac);
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac Sayici.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestSayici
1
2
3
4
5
```

```java
public class TestSayici{
    public static void main(String arg[]){
        Sayici nesne1 = new Sayici();
        Sayici nesne2 = new Sayici();
        Sayici nesne3 = new Sayici();
        Sayici nesne4 = new Sayici();
        Sayici nesne5 = new Sayici();
    }
}
```

# Örnek 1c: Static anahtar sözcüğü (nesne tanımlamadan)

```java
public class Sayici {
    static int sayac = 10;

    Sayici(){
        sayac++;
        System.out.println(sayac);
    }
}
```

```java
public class TestSayici{
    public static void main(String arg[]){
        System.out.println(Sayici.sayac);
        Sayici nesne1 = new Sayici();
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>
javac TestSayici.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>
java TestSayici
10
11
```

# Static anahtar sözcüğü

Bir sınıfın tüm örneklerinin (nesnelerinin) verileri paylaşmasını istiyorsanız, sınıf değişkenleri olarak da bilinen statik değişkenler kullanın.

Statik değişkenler, değişkenler için değerleri ortak bir bellek konumunda depolar. Bu ortak konum nedeniyle, bir nesne statik bir değişkenin değerini değiştirirse, aynı sınıftaki tüm nesneler etkilenir.

Java, statik değişkenlerin yanı sıra statik metotları da destekler. Statik metotlar, sınıfın bir örneğini oluşturmadan çağrılabilir.

# Örnek 2a: Static degisken, metot

```
1  public class A {
2      int i = 5;
3      static int k = 2;
4
5      public static void main(String[] args) {
6      int j = i; // hatali: i ornek degiskeni
7      m1(); // hatali: m1() metot
8      }
9
10     public void m1() {
11         // dogru: ornek, statik degisken ve metot
12         i = i + k + m2(i, k);
13     }
14
15     public static int m2(int i, int j) {
16     return (int)(Math.pow(i, j));
17     }
18 }
```

Math sınıfındaki bütün metotlar statiktir.

main metodu da statiktir.

# Örnek 2b: Static degisken, metot

```
1  public class A {
2      int i = 5;
3      static int k = 2;
4
5      public static void main(String[] args) {
6          A a = new A();
7          int j = a.i; // i ornek degiskeni
8          a.m1(); // m1() metot
9      }
10
11     public void m1() {
12         // dogru: ornek, statik degisken ve metot
13         i = i + k + m2(i, k);
14     }
15
16     public static int m2(int i, int j) {
17     return (int)(Math.pow(i, j));
18     }
19 }
```

Math sınıfındaki bütün metotlar statiktir.

main metodu da statiktir.

# Örnek 3:

```java
public class Hesapla{
    static int kupAl(int a){
        return a*a*a;
    }
    static int kareAl(int a){
        return a*a;
    }
}
```

Bir sınıftaki static metotları nesne oluşturmadan doğrudan kullanabiliriz.

```java
public class TestHesapla{
    public static void main(String arg[]){
        System.out.println(Hesapla.kupAl(5));
        System.out.println(Hesapla.kupAl(10));
        System.out.println(Hesapla.kareAl(5));
        System.out.println(Hesapla.kareAl(10));
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>
javac TestHesapla.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>
java TestHesapla
125
1000
25
100
```

# package, public, default, private

package, sınıfları organize etmek ve düzenlemek için kullanılır.
package paketAdi;
package (paket) en üstte olacak şekilde tanımlanır.

public, sınıflara, metotlara ve veri alanlarına herhangi bir sınıftan erişilebilmesini sağlar.

private, metotların ve veri alanlarının sadece kendi sınıfı içinde erişilebilmesini sağlar.

Bir şey belirtilmediyse (public, private, protected gibi) varsayılan (default) olarak aynı paketin içindeki tüm sınıflara, metotlara ve veri alanlarına herhangi bir sınıf tarafından erişilebilir.

# package, public, default, private

```
package p1;

public class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}
```

```
package p1;

public class C2 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    can access o.y;
    cannot access o.z;

    can invoke o.m1();
    can invoke o.m2();
    cannot invoke o.m3();
  }
}
```

```
package p2;

public class C3 {
  void aMethod() {
    C1 o = new C1();
    can access o.x;
    cannot access o.y;
    cannot access o.z;

    can invoke o.m1();
    cannot invoke o.m2();
    cannot invoke o.m3();
  }
}
```

**FIGURE 9.14** The private modifier restricts access to its defining class, the default modifier restricts access to a package, and the public modifier enables unrestricted access.

# public/public olmayan sınıf

```
package p1;

class C1 {
  ...
}
```

```
package p1;

public class C2 {
  can access C1
}
```

```
package p2;

public class C3 {
  cannot access C1;
  can access C2;
}
```

**FIGURE 9.15** A nonpublic class has package-access.

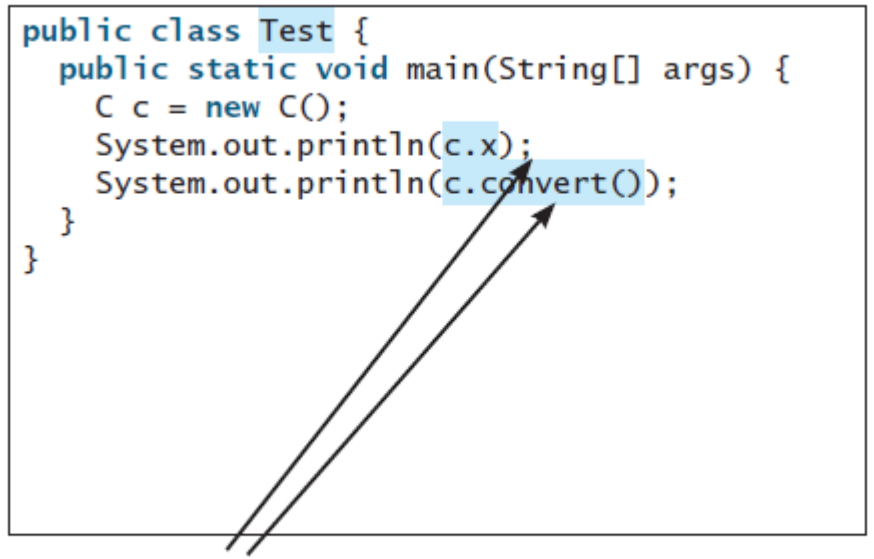
# package, public, default, private

private olduğunda metotlar ve veri alanları sadece kendi sınıfı içinde kullanılabilir.

```
public class C {
  private boolean x;

  public static void main(String[] args) {
    C c = new C();
    System.out.println(c.x);
    System.out.println(c.convert());
  }

  private int convert() {
    return x ? 1 : -1;
  }
}
```

(a) This is okay because object c is used inside the class C.

```
public class Test {
  public static void main(String[] args) {
    C c = new C();
    System.out.println(c.x);
    System.out.println(c.convert());
  }
}
```



(b) This is wrong because x and convert are private in class C.

**FIGURE 9.16** An object can access its private members if it is defined in its own class.

# Ödev 1a:

Aşağıdaki programların çıktısı nedir?

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = null;
    m1(date);
    System.out.println(date);
  }

  public static void m1(Date date) {
    date = new Date();
  }
}
```

(a)

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date = new Date(7654321);
  }
}
```

(b)

# Ödev 1b:

Aşağıdaki programların çıktısı nedir?

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date.setTime(7654321);
  }
}
```

(c)

```
import java.util.Date;

public class Test {
  public static void main(String[] args) {
    Date date = new Date(1234567);
    m1(date);
    System.out.println(date.getTime());
  }

  public static void m1(Date date) {
    date = null;
  }
}
```

(d)

# Ödev 2a:

Aşağıdaki programların çıktısı nedir?

```
public class Test {
  public static void main(String[] args) {
    int[] a = {1, 2};
    swap(a[0], a[1]);
    System.out.println("a[0] = " + a[0]
      + " a[1] = " + a[1]);
  }

  public static void swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
  }
}
```

(a)

```
public class Test {
  public static void main(String[] args) {
    int[] a = {1, 2};
    swap(a);
    System.out.println("a[0] = " + a[0]
      + " a[1] = " + a[1]);
  }

  public static void swap(int[] a) {
    int temp = a[0];
    a[0] = a[1];
    a[1] = temp;
  }
}
```

(b)

# Ödev 2b:

Aşağıdaki programların çıktısı nedir?

```
public class Test {
  public static void main(String[] args) {
    T t = new T();
    swap(t);
    System.out.println("e1 = " + t.e1
      + " e2 = " + t.e2);
  }

  public static void swap(T t) {
    int temp = t.e1;
    t.e1 = t.e2;
    t.e2 = temp;
  }
}

class T {
  int e1 = 1;
  int e2 = 2;
}
```

(c)

```
public class Test {
  public static void main(String[] args) {
    T t1 = new T();
    T t2 = new T();
    System.out.println("t1's i = " +
      t1.i + " and j = " + t1.j);
    System.out.println("t2's i = " +
      t2.i + " and j = " + t2.j);
  }
}

class T {
  static int i = 0;
  int j = 0;

  T() {
    i++;
    j = 1;
  }
}
```

(d)

# Ödev 3:

## Banka Örneği

- Bir banka müşterisinin hesap numarası, müşteri adı ve bankada ne kadar parası olduğu bilgilerini saklamaktadır.
- Müşteri bankaya para yatırabilir veya para çekebilir.
- Para yatırma ve çekme işlemleri sonucunda hesaptaki para miktarının tutarlı olduğunu gösteren kontrol isimli bir yöntem olmalıdır.
- Göster metodu müşteri no, adı ve tutar bilgileri yazdırmaktadır.

# 5. Ders: JAVA ile Nesne Yönelimli Programlama Kalıtım (Inheritance)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama (Ders: 20-30) video.

    - Ders 20: Static Anahtar Sözcüğü (izle)
    - Ders 21: Statik Değişkenler ile Program Sayacı (izle)
    - Ders 22: Statik Yöntemler (izle)
    - Ders 23: Statik Yöntemler Kullanılırken Dikkat Edilecek Unsurlar (izle)
    - Ders 24: this Anahtar Sözcüğünün Kullanımı I (izle)
    - Ders 25: this Anahtar Sözcüğünün Kullanımı II (izle)
    - Ders 26: Kalıtım/Miras (Inheritance) (izle)
    - Ders 27: Kalıtım Türleri (izle)
    - Ders 28: Java'da Neden Çoklu Kalıtım Desteklenmiyor? (izle)
    - Ders 29: Java Polimorfizmi / Yöntem Aşırı Yüklemesi (Method Overloading) (izle)
    - Ders 30: Java Polimorfizmi /Yöntem Geçersiz Kılma (Method Overriding) (izle)

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# Kalıtım (Inheritance)

Nesne yönelimli programlama, mevcut sınıflardan yeni sınıflar tanımlamamıza olanak tanır. Buna kalıtım ya da miras denir.

Önceki derslerde anlattığımız üzere, prosedürel paradigma metotların tasarlanmasına odaklanır ve nesne yönelimli paradigma, verileri ve metotları birlikte nesneler halinde birleştirir.

Nesne yönelimli paradigmayı kullanan yazılım tasarımı, nesnelere ve nesneler üzerindeki işlemlere odaklanır.

Nesne yönelimli yaklaşım, prosedürel paradigmanın gücünü, işlemlerle verileri nesnelere entegre eden ek bir boyutla birleştirir.

# Kalıtım (Inheritance)

Kalıtım, yazılımı yeniden kullanmak için önemli ve güçlü bir özelliktir.

Daireleri, dikdörtgenleri ve üçgenleri modellemek için sınıflar tanımlamamız gerektiğini varsayalım.

Bu sınıfların birçok ortak özelliği vardır.

Fazlalıktan kaçınmak ve sistemin anlaşılmasını ve bakımını kolaylaştırmak için bu sınıfları tasarlamanın en iyi yolu nedir?

Cevap, kalıtımı kullanmaktır.

# Üst sınıflar ve alt sınıflar (Superclasses and subclasses)

Kalıtım, genel bir sınıf (yani bir üst sınıf) tanımlamamıza ve daha sonra bunu daha özel sınıflara (yani alt sınıflara) genişletmemize olanak tanır.

Aynı türdeki nesneleri modellemek için bir sınıf kullanırız.

Farklı sınıflar, diğer sınıflar tarafından paylaşılabilen bir sınıfta genelleştirilebilen bazı ortak özelliklere ve davranışlara sahip olabilir.

Genelleştirilmiş sınıfı genişleten özel bir sınıf tanımlayabilirsiniz.

Özel sınıflar, özellikleri ve metotları genel sınıftan devralır.

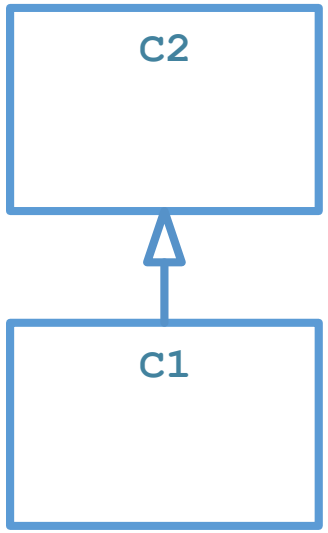
# Üst sınıflar ve alt sınıflar (Superclasses and subclasses)

Java terminolojisinde, başka bir C2 sınıfından genişletilmiş bir C1 sınıfı bir alt sınıf olarak adlandırılır ve C2 bir üst sınıf olarak adlandırılır.

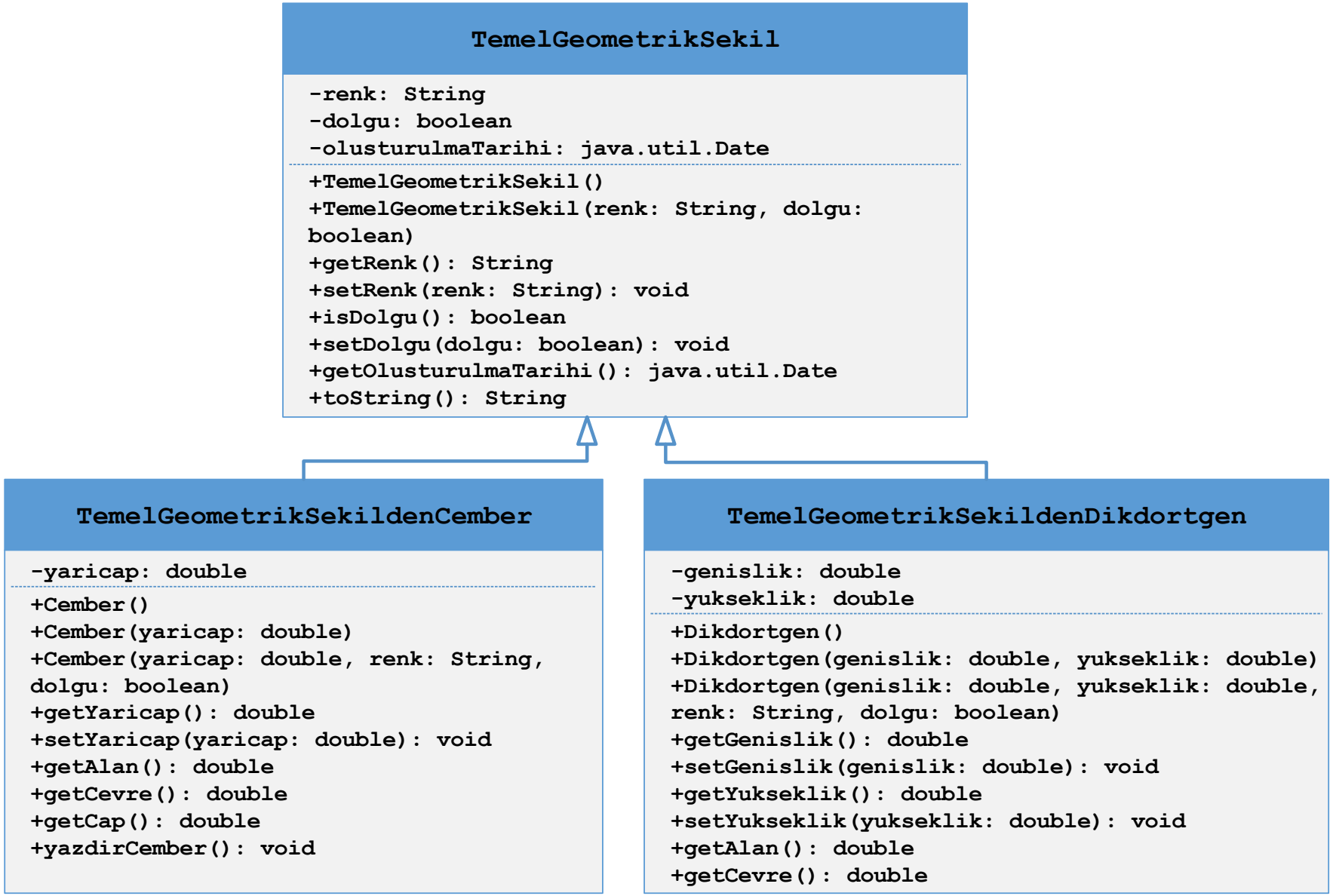
Bir üst sınıfa ayrıca bir ebeveyn (ana) sınıf veya bir temel sınıf

ve bir alt sınıfa bir çocuk sınıf, bir genişletilmiş sınıf veya bir türetilmiş sınıf olarak da başvurulur.

Bir alt sınıf, erişilebilir veri alanlarını ve metotlarını üst sınıfından devralır ve ayrıca yeni veri alanları ve metotlar ekleyebilir.



# Üst sınıflar ve alt sınıflar (Superclasses and subclasses)



# TemelGeometrikSekil.java

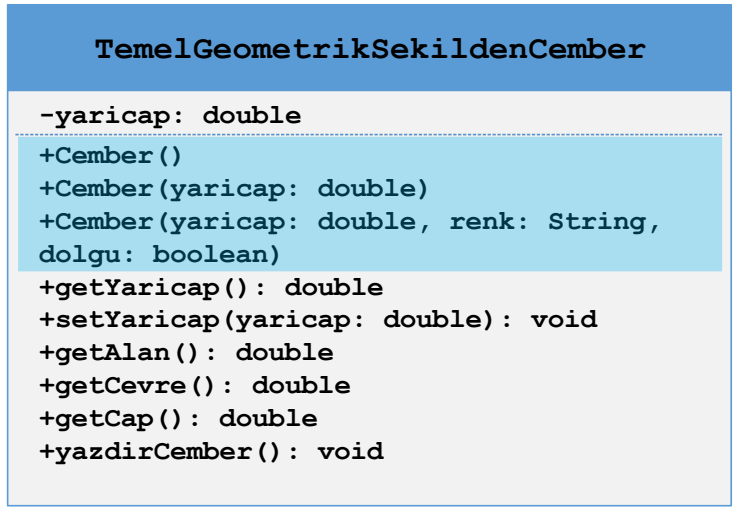
```
1  public class TemelGeometrikSekil {
2      private String renk = "beyaz";
3      private boolean dolgu;
4      private java.util.Date olusturulmaTarihi;
5
6      /** Varsayilan bir geometrik nesne olustur */
7      public TemelGeometrikSekil() {
8          olusturulmaTarihi = new java.util.Date();
9      }
10
11     /** Belirtilen renk ve dolgu degeri ile
12     * varsayilan bir geometrik nesne olustur */
13     public TemelGeometrikSekil(String renk,
14     boolean dolgu) {
15         olusturulmaTarihi = new java.util.Date();
16         this.renk = renk;
17         this.dolgu = dolgu;
18     }
19
20     /** Renk dondur */
21     public String getRenk() {
22         return renk;
23     }
24
25     /** Yeni renk ata */
26     public void setRenk(String color) {
27         this.renk = renk;
28     }
29
```

**TemelGeometrikSekil**

```
-renk: String
-dolgu: boolean
-olusturulmaTarihi: java.util.Date
+TemelGeometrikSekil()
+TemelGeometrikSekil(renk: String, dolgu:
boolean)
+getRenk(): String
+setRenk(renk: String): void
+isDolgu(): boolean
+setDolgu(dolgu: boolean): void
+getOlusturulmaTarihi(): java.util.Date
+toString(): String
```

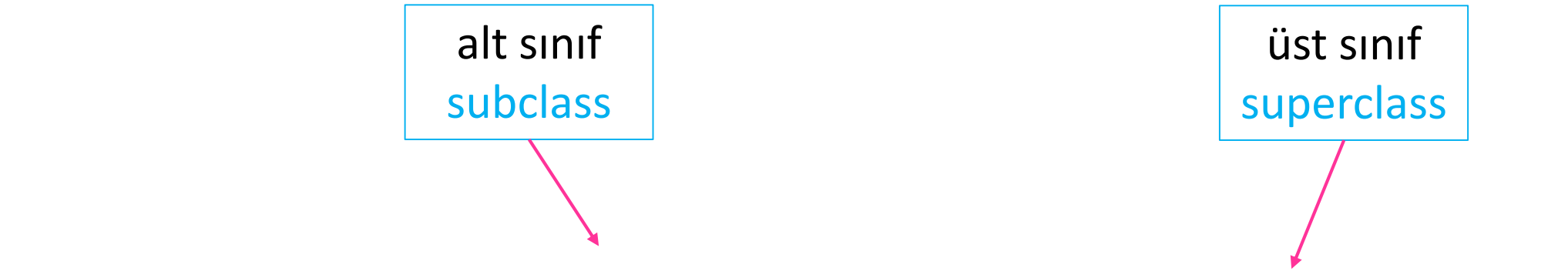
```
30     /** Dolgu dondur. dolgu boolean oldugundan,
31     get metodunu is..... seklinde adlandirildi */
32     public boolean isDolgu() {
33         return dolgu;
34     }
35
36     /** Dolgu ata */
37     public void setDolgu(boolean dolgu) {
38         this.dolgu = dolgu;
39     }
40
41     /** OlusturulmaTarihi dondur */
42     public java.util.Date getOlusturulmaTarihi() {
43         return olusturulmaTarihi;
44     }
45
46     /** Bu nesnenin string sunumunu dondur */
47     public String toString() {
48         return "Olusturulma Tarihi: " + olusturulmaTarihi
49         + "\nrenk: " + renk + " ve dolgu: " + dolgu;
50     }
51 }
```

# TemelGeometrikSekildenCember.java




```java
public class TemelGeometrikSekildenCember
        extends TemelGeometrikSekil{
    private double yaricap;

    public TemelGeometrikSekildenCember() {
    }

    public TemelGeometrikSekildenCember(double yaricap) {
        this.yaricap = yaricap;
    }

    public TemelGeometrikSekildenCember(double yaricap,
        String renk, boolean dolgu) {
        this.yaricap = yaricap;
        setRenk(renk);
        setDolgu(dolgu);
    }

    /** Yaricap dondur */
    public double getYaricap() {
        return yaricap;
    }

    /** Yeni bir yaricap ata */
    public void setYaricap(double yaricap) {
    this.yaricap = yaricap;
    }

    /** Alan dondur */
    public double getAlan() {
        return yaricap * yaricap * Math.PI;
    }

    /** Cap dondur */
    public double getCap() {
        return 2 * yaricap;
    }

    /** Cevre dondur */
    public double getCevre() {
        return 2 * yaricap * Math.PI;
    }

    /** Cember bilgisini yazdir */
    public void yazdirCember() {
        System.out.println("Cember, " + getOlusturulmaTarihi() +
        " tarihinde olusturuldu ve yaricapi " + yaricap + " dir.");
    }
}
```


alt sınıf
subclass
üst sınıf
superclass
public class TemelGeometrikSekildenCember extends TemelGeometrikSekil

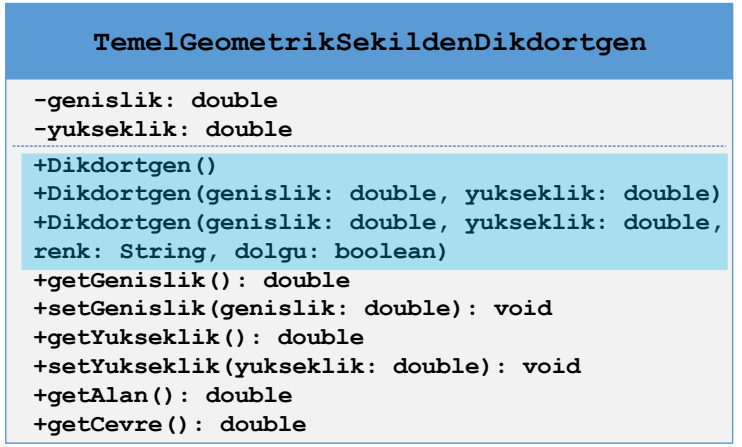

alt sınıf
subclass
üst sınıf
superclass
public class TemelGeometrikSekildenDikdortgen extends TemelGeometrikSekil

# private kullanımı

```java
public class TemelGeometrikSekil {
    private String renk = "beyaz";
    private boolean dolgu;
    private java.util.Date olusturulmaTarihi;

    /** Varsayilan bir geometrik nesne olustur */
    public TemelGeometrikSekil() {
        olusturulmaTarihi = new java.util.Date();
    }

    /** Belirtilen renk ve dolgu degeri ile
    * varsayilan bir geometrik nesne olustur */
    public TemelGeometrikSekil(String renk,
    boolean dolgu) {
        olusturulmaTarihi = new java.util.Date();
        this.renk = renk;
        this.dolgu = dolgu;
    }

    /** Renk dondur */
    public String getRenk() {
        return renk;
    }

    /** Yeni renk ata */
    public void setRenk(String color) {
        this.renk = renk;
    }

```

```java
public class TemelGeometrikSekildenCember
        extends TemelGeometrikSekil{
    private double yaricap;

    public TemelGeometrikSekildenCember() {
    }

    public TemelGeometrikSekildenCember(double yaricap) {
        this.yaricap = yaricap;
    }

    public TemelGeometrikSekildenCember(double yaricap,
        String renk, boolean dolgu) {
        this.yaricap = yaricap;
        this.renk;  //yanlış kullanım
        this.dolgu; //yanlış kullanım
    }
    /** Yaricap dondur */
    public double getYaricap() {
        return yaricap;
    }

    /** Yeni bir yaricap ata */
    public void setYaricap(double yaricap) {
    this.yaricap = yaricap;
    }

```

# TemelGeometrikSekildenDikdortgen.java




```java
public class TemelGeometrikSekildenDikdortgen
        extends TemelGeometrikSekil {
    private double genislik;
    private double yukseklik;

    public TemelGeometrikSekildenDikdortgen() {
    }

    public TemelGeometrikSekildenDikdortgen(double genislik,
            double yukseklik) {
        this.genislik = genislik;
        this.yukseklik = yukseklik;
    }

    public TemelGeometrikSekildenDikdortgen(double genislik,
            double yukseklik, String renk, boolean dolgu) {
        this.genislik = genislik;
        this.yukseklik = yukseklik;
        setRenk(renk);
        setDolgu(dolgu);
    }

    /** Genislik dondur */
    public double getGenislik() {
        return genislik;
    }
```

```java
    /** Yeni bir genislik ata */
    public void setGenislik(double genislik) {
        this.genislik = genislik;
    }

    /** Yukseklik dondur */
    public double getYukseklik() {
        return yukseklik;
    }

    /** Yeni bir yukseklik ata */
    public void setYukseklik(double yukseklik) {
        this.yukseklik = yukseklik;
    }

    /** Alan dondur */
    public double getAlan() {
        return genislik * yukseklik;
    }

    /** Cevre dondur */
    public double getCevre() {
        return 2 * (genislik + yukseklik);
    }
}
```

# TestCemberDikdortgen.java

```java
public class TestCemberDikdortgen {
    public static void main(String[] args) {
        TemelGeometrikSekildenCember cember = new TemelGeometrikSekildenCember(1);
        System.out.println("Bir cember: " + cember.toString());
        System.out.println("Renk: " + cember.getRenk());
        System.out.println("Yaricap: " + cember.getYaricap());
        System.out.println("Alan: " + cember.getAlan());
        System.out.println("Cap: " + cember.getCap());

        TemelGeometrikSekildenDikdortgen dikdortgen = new TemelGeometrikSekildenDikdortgen(2, 4);
        System.out.println("\nBir dikdortgen: " + dikdortgen.toString());
        System.out.println("Alan: " + dikdortgen.getAlan());
        System.out.println("Cevre: " + dikdortgen.getCevre());
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestCemberDikdortgen
Bir cember: Olusturulma Tarihi: Sun Apr 04 13:21:51 TRT 2021
renk: beyaz ve dolgu: false
Renk: beyaz
Yaricap: 1.0
Alan: 3.141592653589793
Cap: 2.0

Bir dikdortgen: Olusturulma Tarihi: Sun Apr 04 13:21:51 TRT 2021
renk: beyaz ve dolgu: false
Alan: 8.0
Cevre: 12.0
```

# super anahtar sözcüğü (keyword)

super anahtar sözcüğü üst sınıfa atıfta bulunur ve üst sınıf metotlarını ve yapıcılarını çağırmak için kullanılabilir.

İki şekilde kullanılabilir:

Bir üst sınıf yapıcısını çağırmak için.

Bir üst sınıf metodunu çağırmak için.

# üst sınıf (superclass) yapıcısı çağırma

super() ifadesi üst sınıfın argümansız yapıcısını çağırır.

super(argümanlar) ifadesi argümanlarla eşleşen üst sınıf yapıcısını çağırır.

super() veya super(argümanlar) ifadesi, alt sınıfın yapıcısının ilk ifadesi olmalıdır.

Bu, bir üst sınıf yapıcısını açıkça çağırmanın tek yoludur.

# üst sınıf (superclass) yapıcısı çağırma

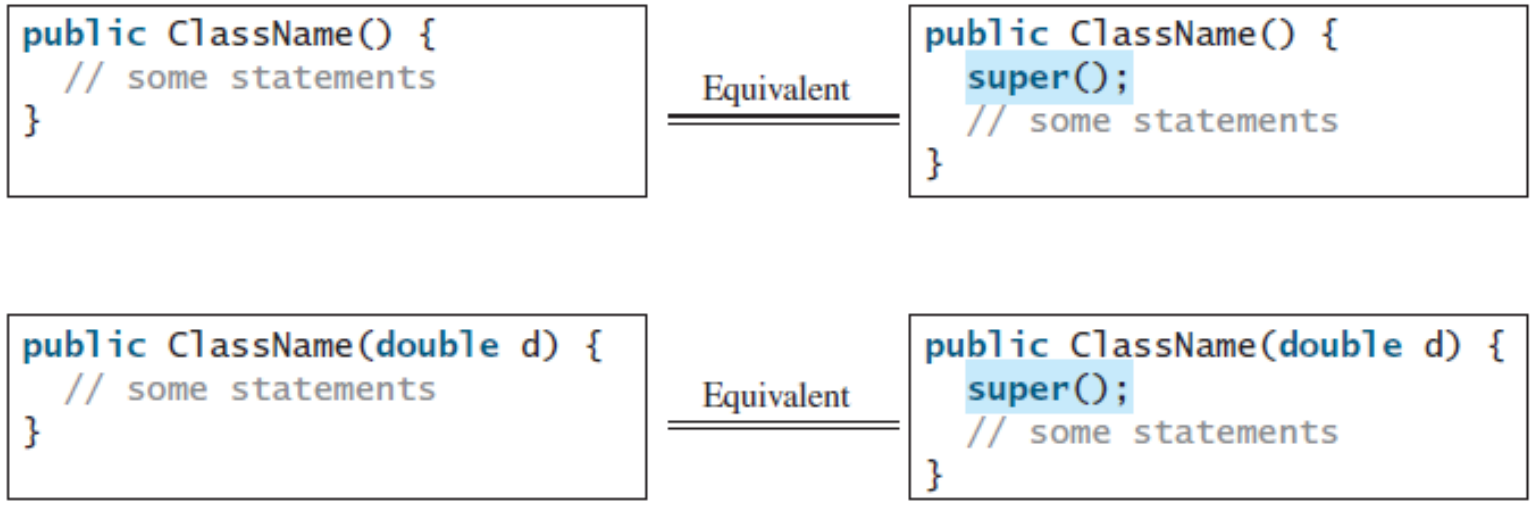
Üst sınıf kurucusunu çağırmak için super anahtar kelimesini kullanmamız gerekir ve çağrı, yapıcıdaki ilk ifade olmalıdır.

```
1  public class TemelGeometrikSekildenCember
2          extends TemelGeometrikSekil{
3      private double yaricap;
4
5      public TemelGeometrikSekildenCember() {
6      }
7
8      public TemelGeometrikSekildenCember(double yaricap) {
9          this.yaricap = yaricap;
10     }
11
12     public TemelGeometrikSekildenCember(double yaricap,
13         String renk, boolean dolgu) {
14         this.yaricap = yaricap;
15         setRenk(renk);
16         setDolgu(dolgu);
17     }
18
19     /** Yaricap dondur */
20     public double getYaricap() {
21         return yaricap;
22     }
23
24     /** Yeni bir yaricap ata */
25     public void setYaricap(double yaricap) {
26     this.yaricap = yaricap;
27     }
28
```

```
1  public class TemelGeometrikSekildenCember
2          extends TemelGeometrikSekil{
3      private double yaricap;
4
5      public TemelGeometrikSekildenCember() {
6      }
7
8      public TemelGeometrikSekildenCember(double yaricap) {
9          this.yaricap = yaricap;
10     }
11
12     public TemelGeometrikSekildenCember(double yaricap,
13         String renk, boolean dolgu) {
14         super(renk, dolgu);
15         this.yaricap = yaricap;
16     }
17
18     /** Yaricap dondur */
19     public double getYaricap() {
20         return yaricap;
21     }
22
23     /** Yeni bi
24     public void
25     this.yarica
26     }
27
```

Bir alt sınıfta bir üst sınıf kurucusunun adını çağırmak sözdizimi hatasına neden olur.

# yapıcı (constructor) zinciri




Bir yapıcı, aşırı yüklenmiş bir yapıcıyı veya üst sınıf yapıcısını çağırabilir.

Hiçbiri açıkça çağrılmazsa, derleyici otomatik olarak yapıcıya ilk ifade olarak super() koyar.

# yapıcı zincirleme (constructor chaining)

Her durumda, bir sınıfın bir örneğini oluşturmak, miras zinciri boyunca tüm üst sınıfların yapıcılarını çağırır.

Bir alt sınıfın bir nesnesini oluştururken, alt sınıf yapıcısı ilk olarak kendi görevlerini gerçekleştirmeden önce üst sınıf yapıcısını çağırır.

Üst sınıf başka bir sınıftan türetilmişse, üst sınıf yapıcısı kendi görevlerini gerçekleştirmeden önce üst sınıf yapıcısını çağırır.

Bu süreç, miras hiyerarşisi boyunca son kurucu çağrılana kadar devam eder. Buna yapıcı zincirleme denir.

# Örnek 1: yapıcı zincirleme (constructor chaining)

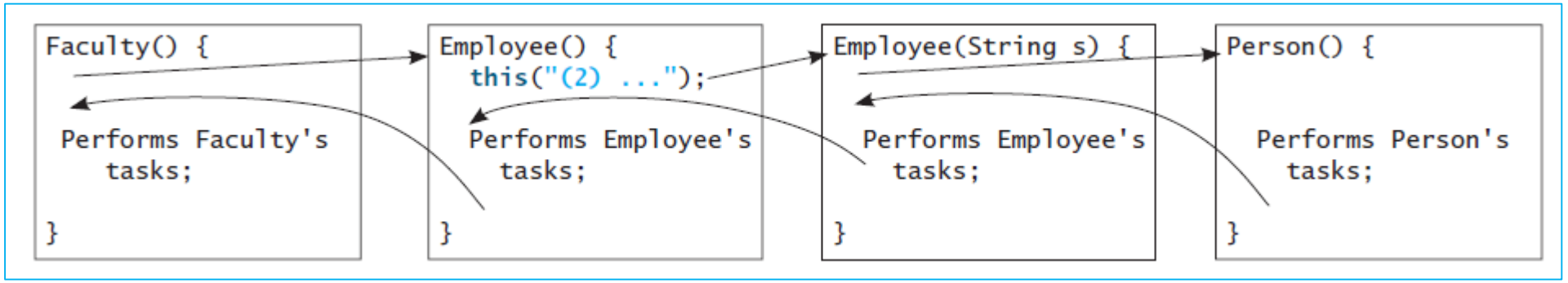
```
1  public class Faculty extends Employee {
2    public static void main(String[] args) {
3      new Faculty();
4    }
5
6    public Faculty() {
7      System.out.println("(4) Performs Faculty's tasks");
8    }
9  }
10
11 class Employee extends Person {
12   public Employee() {
13     this("(2) Invoke Employee's overloaded constructor");
14     System.out.println("(3) Performs Employee's tasks ");
15   }
16
17   public Employee(String s) {
18     System.out.println(s);
19   }
20 }
21
22 class Person {
23   public Person() {
24     System.out.println("(1) Performs Person's tasks");
25   }
26 }
```

```
(1) Performs Person's tasks
(2) Invoke Employee's overloaded constructor
(3) Performs Employee's tasks
(4) Performs Faculty's tasks
```

# Örnek 1: yapıcı zincirleme (constructor chaining)

```
1  public class Faculty extends Employee {
2    public static void main(String[] args) {
3      new Faculty();
4    }
5
6    public Faculty() {
7      System.out.println("(4) Performs Faculty's tasks");
8    }
9  }
10
11 class Employee extends Person {
12   public Employee() {
13     this("(2) Invoke Employee's overloaded constructor");
14     System.out.println("(3) Performs Employee's tasks ");
15   }
16
17   public Employee(String s) {
18     System.out.println(s);
19   }
20 }
21
22 class Person {
23   public Person() {
24     System.out.println("(1) Performs Person's tasks");
25   }
26 }
```

```
(1) Performs Person's tasks
(2) Invoke Employee's overloaded constructor
(3) Performs Employee's tasks
(4) Performs Faculty's tasks
```



# üst sınıf (superclass) metodu çağırma

super anahtar sözcüğü, üst sınıftaki yapıcıdan başka bir metoda başvurmak için de kullanılabilir.

Sözdizimi şöyledir:

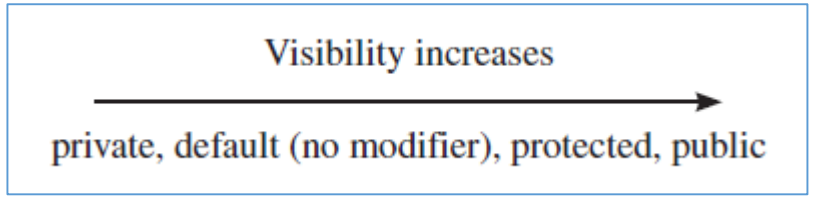
super.metot(parametre);

```
/** Cember bilgisini yazdir */
public void yazdirCember() {
    System.out.println("Cember, " + super.getOlusturulmaTarihi() +
    " tarihinde olusturuldu ve yaricapi " + yaricap + " dir.");
}
```

# public, protected, default, private

**TABLE 11.2** Data and Methods Visibility

| *Modifier on members in a class* | *Accessed from the same class* | *Accessed from the same package* | *Accessed from a subclass in a different package* | *Accessed from a different package* |
|---|---|---|---|---|
| public | ✓ | ✓ | ✓ | ✓ |
| protected | ✓ | ✓ | ✓ | – |
| default (no modifier) | ✓ | ✓ | – | – |
| private | ✓ | – | – | – |



# package, public, protected, default, private

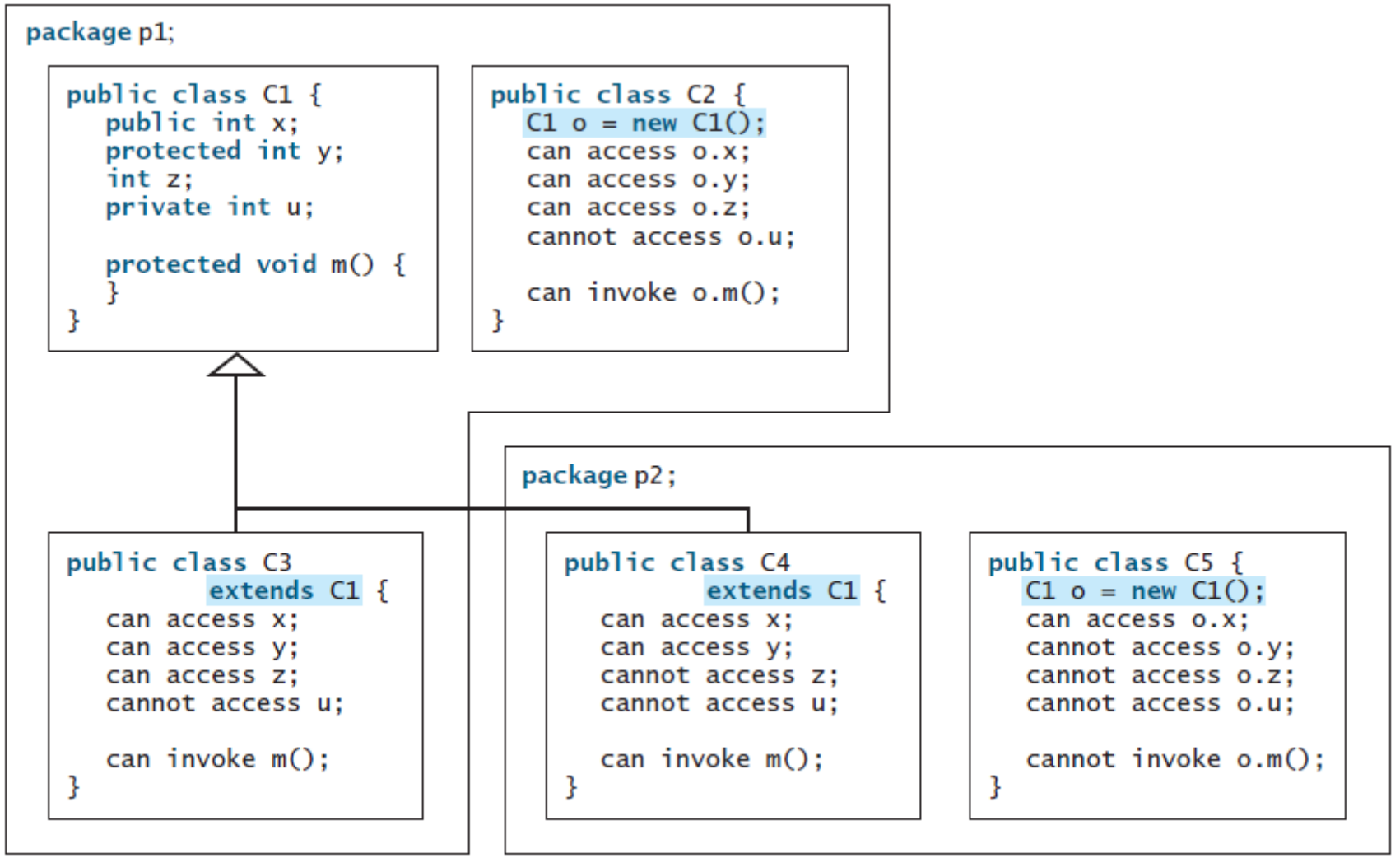



**FIGURE 11.5** Visibility modifiers are used to control how data and methods are accessed.

# Ödev 1:

Aşağıdaki programların çıktısı nedir?

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

(a)

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

(b)

# 6. Ders: JAVA ile Nesne Yönelimli Programlama
# Çok biçimlilik (Polymorphism)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama (Ders: 24-34) video.

    - Ders 24: this Anahtar Sözcüğünün Kullanımı I (izle)
    - Ders 25: this Anahtar Sözcüğünün Kullanımı II (izle)
    - Ders 26: Kalıtım/Miras (Inheritance) (izle)
    - Ders 27: Kalıtım Türleri (izle)
    - Ders 28: Java'da Neden Çoklu Kalıtım Desteklenmiyor? (izle)
    - Ders 29: Java Polimorfizmi / Yöntem Aşırı Yüklemesi (Method Overloading) (izle)
    - Ders 30: Java Polimorfizmi / Yöntem Geçersiz Kılma (Method Overriding) (izle)
    - Ders 31: super Anahtar Sözcüğünün Kullanımı (izle)
    - Ders 32: final Anahtar Sözcüğünün Kullanımı (izle)
    - Ders 33: Soyut Sınıf (Abstract Class) (izle)
    - Ders 34: Arayüz (Interface) (izle)

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# Metot Geçersiz Kılma (Method Overriding)

Bir metodu geçersiz kılmak için, o metodun alt sınıfta, üst sınıfındakiyle aynı imza ve aynı dönüş türü kullanılarak tanımlanması gerekir.

Bir alt sınıf, metotlarını bir üst sınıftan kalıtım yoluyla miras alır.

Bazen alt sınıfın, üst sınıfta tanımlanan bir metodun uygulamasını (implementation) değiştirmesi gerekir.

Bu, metodu geçersiz kılma olarak adlandırılır.

# Metot Geçersiz Kılma (Method Overriding)

```java
public class TemelGeometrikSekil {
    private String renk = "beyaz";
    private boolean dolgu;
    private java.util.Date olusturulmaTarihi;

    /** Bu nesnenin string sunumunu dondur */
    public String toString() {
        return "Olusturulma Tarihi: " + olusturulmaTarihi
        + "\nrenk: " + renk + " ve dolgu: " + dolgu;
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestCemberDikdortgen
Bir cember: Olusturulma Tarihi: Sun Apr 04 13:21:51 TRT 2021
renk: beyaz ve dolgu: false
Renk: beyaz
Yaricap: 1.0
Alan: 3.141592653589793
Cap: 2.0

Bir dikdortgen: Olusturulma Tarihi: Sun Apr 04 13:21:51 TRT 2021
renk: beyaz ve dolgu: false
Alan: 8.0
Cevre: 12.0
```

```java
public class TemelGeometrikSekildenCember
        extends TemelGeometrikSekil{
    private double yaricap;

    public TemelGeometrikSekildenCember() {
    }

    // Üst sınıfta tanımlanan toString metodunu gecersiz kılma
    public String toString() {
        return super.toString() + "\n yari cap: " + yaricap;
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestCemberDikdortgen
Bir cember: Olusturulma Tarihi: Sun Apr 11 00:02:23 TRT 2021
renk: beyaz ve dolgu: false
 yari cap: 1.0
Renk: beyaz
Yaricap: 1.0
Alan: 3.141592653589793
Cap: 2.0

Bir dikdortgen: Olusturulma Tarihi: Sun Apr 11 00:02:23 TRT 2021
renk: beyaz ve dolgu: false
Alan: 8.0
Cevre: 12.0
```

# Metot Geçersiz Kılma (Method Overriding)

```java
public class TestCemberDikdortgen {
    public static void main(String[] args) {
        TemelGeometrikSekildenCember cember = new TemelGeometrikSekildenCember(1);
        System.out.println("Bir cember: " + cember.toString());
        System.out.println("Renk: " + cember.getRenk());
        System.out.println("Yaricap: " + cember.getYaricap());
        System.out.println("Alan: " + cember.getAlan());
        System.out.println("Cap: " + cember.getCap());

        TemelGeometrikSekildenDikdortgen dikdortgen = new TemelGeometrikSekildenDikdortgen(2, 4);
        System.out.println("\nBir dikdortgen: " + dikdortgen.toString());
        System.out.println("Alan: " + dikdortgen.getAlan());
        System.out.println("Cevre: " + dikdortgen.getCevre());
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestCemberDikdortgen
Bir cember: Olusturulma Tarihi: Sun Apr 04 13:21:51 TRT 2021
renk: beyaz ve dolgu: false
Renk: beyaz
Yaricap: 1.0
Alan: 3.141592653589793
Cap: 2.0

Bir dikdortgen: Olusturulma Tarihi: Sun Apr 04 13:21:51 TRT 2021
renk: beyaz ve dolgu: false
Alan: 8.0
Cevre: 12.0
```

```java
    // Üst sınıfta tanımlanan toString metodunu gecersiz kılma
    public String toString() {
        return super.toString() + "\n yari cap: " + yaricap;
    }
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestCemberDikdortgen
Bir cember: Olusturulma Tarihi: Sun Apr 11 00:02:23 TRT 2021
renk: beyaz ve dolgu: false
 yari cap: 1.0
Renk: beyaz
Yaricap: 1.0
Alan: 3.141592653589793
Cap: 2.0

Bir dikdortgen: Olusturulma Tarihi: Sun Apr 11 00:02:23 TRT 2021
renk: beyaz ve dolgu: false
Alan: 8.0
Cevre: 12.0
```

# Geçersiz kılma vs. Aşırı yükleme (Overriding vs. Overloading)

Geçersiz kılma (overriding), alt sınıftaki bir metot için yeni bir uygulama (implement) sağlamak anlamına gelir.

Aşırı yükleme (overloading), aynı isme ve farklı imzalara sahip birden çok metodu tanımlamak anlamına gelir.

```java
public class Test1 {
    public static void main(String[] args) {
        A1 a = new A1();
        a.p(10);
        a.p(10.0);
    }
}
class B1 {
    public void p(double i) {
        System.out.println(i * 2);
    }
}
class A1 extends B1 {
    // Bu metot B1'deki metodu geçersiz kılar
    public void p(double i) {
        System.out.println(i * 3);
    }
}
```

```java
public class Test2 {
    public static void main(String[] args) {
        A2 a = new A2();
        a.p(10);
        a.p(10.0);
    }
}
class B2 {
    public void p(double i) {
        System.out.println(i * 2);
    }
}
class A2 extends B2 {
    // Bu metot B2'deki metodu aşırı yükler
    public void p(int i) {
        System.out.println(i * 3);
    }
}
```

# Geçersiz kılma vs. Aşırı yükleme (Overriding vs. Overloading)

```
C:\Program Files\Java\jdk-15.0.1\
bin\yeni2>java Test1
30.0
30.0
```

```
C:\Program Files\Java\jdk-15.0.1\
bin\yeni2>java Test2
30
20.0
```

```java
public class Test1 {
    public static void main(String[] args) {
        A1 a = new A1();
        a.p(10);
        a.p(10.0);
    }
}
class B1 {
    public void p(double i) {
        System.out.println(i * 2);
    }
}
class A1 extends B1 {
    // Bu metot B1'deki metodu geçersiz kılar
    public void p(double i) {
        System.out.println(i * 3);
    }
}
```

```java
public class Test2 {
    public static void main(String[] args) {
        A2 a = new A2();
        a.p(10);
        a.p(10.0);
    }
}
class B2 {
    public void p(double i) {
        System.out.println(i * 2);
    }
}
class A2 extends B2 {
    // Bu metot B2'deki metodu aşırı yükler
    public void p(int i) {
        System.out.println(i * 3);
    }
}
```

# Aşırı yükleme (Overloading)

```java
public class Test2 {
    public static void main(String[] args) {
        A2 a = new A2();
        a.p(10);
        a.p(10.0);
        a.p("Yazilim Muhendisligi");
    }
}
class B2 {
    public void p(double i) {
        System.out.println(i * 2);
    }
}
class A2 extends B2 {
    // Bu metot B2'deki metodu aşırı yükler
    public void p(int i) {
        System.out.println(i * 3);
    }
    // Bu metot B2'deki metodu aşırı yükler
    public void p(String yazi) {
        System.out.println("Merhaba " + yazi
        + "\nMerhaba " + yazi );
    }
}
```

Aşırı yükleme (overloading), aynı isme ve farklı imzalara sahip birden çok metodu tanımlamak anlamına gelir.

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>
javac Test2.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>
java Test2
30
20.0
Merhaba Yazilim Muhendisligi
Merhaba Yazilim Muhendisligi
```

# Çok biçimlilik (Polymorphism)

Çok biçimlilik, bir üst tip değişkeninin bir alt tip nesneye başvurabileceği anlamına gelir.

Nesne yönelimli programlamanın üç ayağı
kapsülleme (encapsulation),
kalıtım (inheritance)
ve çok biçimliliktir (polymorphism).

# Üst tip ve alt tip (supertype and subtype)

Bir sınıf, bir tipi tanımlar.

Bir alt sınıf (subclass) tarafından tanımlanan bir tipe alt tip (subtype) denir.

Bir üst sınıf (superclass) tarafından tanımlanan bir tipe üst tip (supertype) denir.

Bir önceki derste oluşturduğumuz *Cember* bir *GeometrikSekil*'in alt tipi veya *GeometrikSekil Cember*'in üst tipi olduğunu söyleyebiliriz.

Kalıtım ilişkisi, bir alt sınıfın ek yeni özelliklerle üst sınıfından özellikleri devralmasını sağlar.

# Çok biçimlilik (Polymorphism)

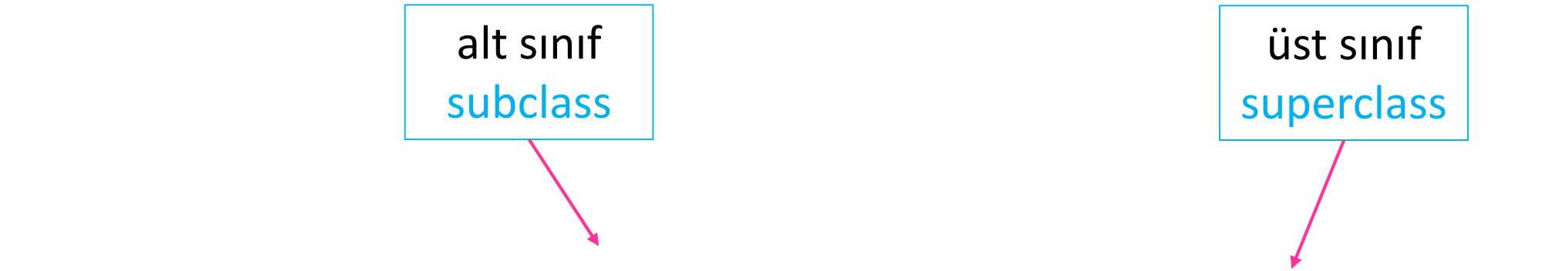
Bir alt sınıf, üst sınıfının bir özelleşmiş (uzmanlaşmış) halidir;

Bir alt sınıfın her örneği (nesnesi) aynı zamanda üst sınıfının bir örneğidir, ancak bunun tersi geçerli değildir.

Örneğin, her çember geometrik bir nesnedir, ancak her geometrik nesne bir çember değildir.

Bu nedenle, bir alt sınıfın bir örneğini her zaman üst sınıf türündeki bir parametreye geçirebilirsiniz.


alt sınıf
subclass
üst sınıf
superclass
public class TemelGeometrikSekildenCember extends TemelGeometrikSekil


alt sınıf
subclass
üst sınıf
superclass
public class TemelGeometrikSekildenDikdortgen extends TemelGeometrikSekil

# CokbicimlilikDemo.java

```java
public class CokbicimlilikDemo {
    /** Main metot */
    public static void main(String[] args) {
        // cember ve dikdortgen ozelliklerini goster
        nesneGoster(new TemelGeometrikSekildenCember(1, "kirmizi", false));
        nesneGoster(new TemelGeometrikSekildenDikdortgen(1, 1, "siyah", true));
    }

    /** Geometrik nesne ozelliklerini goster */
    public static void nesneGoster(TemelGeometrikSekil nesne) {
        System.out.println(nesne.getOlusturulmaTarihi() +
        " tarihinde nesne olusturuldu. Rengi " +
        nesne.getRenk() + " dir.");
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java CokbicimlilikDemo
Sat Apr 10 23:19:38 TRT 2021 tarihinde nesne olusturuldu. Rengi kirmizi dir.
Sat Apr 10 23:19:39 TRT 2021 tarihinde nesne olusturuldu. Rengi siyah dir.
```

Alt sınıfın bir nesnesi, üst sınıf nesnesinin kullanıldığı her yerde kullanılabilir. Bu genellikle çok biçimlilik (polimorfizm, polymorphism) olarak bilinir.

Basit bir ifadeyle, çok biçimlilik, bir süper tip değişkeninin bir alt tip nesneye atıfta bulunabileceği anlamına gelir.

# 7. Ders: JAVA ile Nesne Yönelimli Programlama İstisna İşleme (Exception Handling)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA Programlama Dili (Ders: 67-72) video.

    - Ders 67: İstisna İşleme (Exception Handling) (izle)
    - Ders 68: Exception Handling Sınıflarının Hiyerarşisi (izle)
    - Ders 69: Exception Handling Sınıfı Anahtar Kelimeleri (izle)
    - Ders 70: try-catch Bloğunun İç Yapısı (izle)
    - Ders 71: İç İçe (Yuvarlanmış) try Bloğu (izle)
    - Ders 72: finally Bloğu Kullanımı (izle)

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# İstisna İşleme (Exception Handling)

İstisna işleme (exception handling), bir programın istisnai durumlarla başa çıkmasına ve normal yürütülmesine devam etmesine imkan verir.

JVM, gerçekleştirilmesi imkansız bir işlem tespit ederse, bir program çalışırken çalışma zamanı hataları meydana gelir.

Örneğin, sınırların dışında bir indeks kullanarak bir diziye erişirseniz, ArrayIndexOutOfBoundsException ile bir çalışma zamanı hatası alırsınız.

Programınız bir tamsayı (int) beklediğinde double değer girerseniz, InputMismatchException ile bir çalışma zamanı hatası alırsınız.

# İstisna İşleme (Exception Handling)

Java'da çalışma zamanı hataları (runtime errors) istisna olarak atılır.

İstisna, yürütmenin normal şekilde ilerlemesini engelleyen bir hatayı veya koşulu temsil eden bir nesnedir.

İstisna ele alınmazsa, program anormal şekilde sona erecektir.

İstisnayı, programın çalışmaya devam etmesi veya düzgün bir şekilde sona erdirilmesi için nasıl ele almalıyız?

# İstisna İşleme (Exception Handling)

İstisnalar, bir metottan atılır. Metodun çağırıcısı istisnayı yakalayabilir ve işleyebilir.

Bolme.java

```
1  import java.util.Scanner;
2
3  public class Bolme {
4      public static void main(String[] args) {
5      Scanner giris = new Scanner(System.in);
6
7      System.out.print("Lutfen iki tam sayi giriniz: ");
8      int sayi1 = giris.nextInt();
9      int sayi2 = giris.nextInt();
10
11     System.out.println(sayi1 + " / " + sayi2 + " = " +
12     (sayi1 / sayi2) + " dir.");
13     }
14 }
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java Bolme
Lutfen iki tam sayi giriniz: 9 2
9 / 2 = 4 dir.

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java Bolme
Lutfen iki tam sayi giriniz: 9 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Bolme.main(Bolme.java:11)
```

# İstisna İşleme (Exception Handling)

```java
import java.util.Scanner;

public class BolmeIfIle {
    public static void main(String[] args) {
    Scanner giris = new Scanner(System.in);

    System.out.print("Lutfen iki tam sayi giriniz: ");
    int sayi1 = giris.nextInt();
    int sayi2 = giris.nextInt();

    if (sayi2 != 0)
        System.out.println(sayi1 + " / " + sayi2 + " = " +
        (sayi1 / sayi2) + " dir.");
    else
        System.out.println("Bolen sifir olamaz ");
    }
}
```

BolmeIfIle.java

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java BolmeIfIle
Lutfen iki tam sayi giriniz: 9 2
9 / 2 = 4 dir.

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java BolmeIfIle
Lutfen iki tam sayi giriniz: 9 0
Bolen sifir olamaz
```

# İstisna İşleme (Exception Handling)

```java
import java.util.Scanner;

public class BolmeMetotIle {
    public static int bolme(int sayi1, int sayi2) {
        if (sayi2 == 0) {
            System.out.println("Bolen sifir olamaz");
            System.exit(1);
        }

        return sayi1 / sayi2;
    }

    public static void main(String[] args) {
        Scanner giris = new Scanner(System.in);

        System.out.print("Lutfen iki tam sayi giriniz: ");
        int sayi1 = giris.nextInt();
        int sayi2 = giris.nextInt();

        int sonuc = bolme(sayi1, sayi2);
        System.out.println(sayi1 + " / " + sayi2 + " = " +
        sonuc + " dir.");
    }
}
```

BolmeMetotIle.java

```
C:\Program Files\Java\jdk-15.0.1\bin\
yeni2>java BolmeMetotIle
Lutfen iki tam sayi giriniz: 5 3
5 / 3 = 1 dir.

C:\Program Files\Java\jdk-15.0.1\bin\
yeni2>java BolmeMetotIle
Lutfen iki tam sayi giriniz: 5 0
Bolen sifir olamaz
```

# İstisna İşleme (Exception Handling)

```java
import java.util.Scanner;

public class BolmeIstisnaIle {
    public static int bolme(int sayi1, int sayi2) {
        if (sayi2 == 0)
            throw new ArithmeticException("Bolen sifir olamaz");

        return sayi1 / sayi2;
    }

    public static void main(String[] args) {
        Scanner giris = new Scanner(System.in);

        System.out.print("Lutfen iki tam sayi giriniz: ");
        int sayi1 = giris.nextInt();
        int sayi2 = giris.nextInt();

        try {
            int sonuc = bolme(sayi1, sayi2);
            System.out.println(sayi1 + " / " + sayi2 + " = " +
            sonuc + " dir.");
        }
        catch (ArithmeticException ex) {
            System.out.println("Istisna: bir tam sayi " +
            "sifira bolunemez ");
        }

        System.out.println("Yurutme devam ediyor ...");
    }
}
```

BolmeIstisnaIle.java

```
C:\Program Files\Java\jdk-15.0.1\bin\
yeni2>java BolmeIstisnaIle
Lutfen iki tam sayi giriniz: 5 3
5 / 3 = 1 dir.
Yurutme devam ediyor ...

C:\Program Files\Java\jdk-15.0.1\bin\
yeni2>java BolmeIstisnaIle
Lutfen iki tam sayi giriniz: 5 0
Istisna: bir tam sayi sifira boluneme
z
Yurutme devam ediyor ...
```

# try catch (dene yakala)

```
try {
    // Çalıştırılacak kod;
    // Bir istisna atabilecek bir ifade veya metot;
    // Çalıştırılacak daha fazla kod;
}
catch (tip ex) {
    // İstisnayı işlemek için kod;
}
```

# try catch (dene yakala)

```
try {
    // Bir istisna atabilecek ifadeler;
}
catch (Istisna1 exVar1) {
    // İstisna1'i işlemek için kod;
}
catch (Istisna2 exVar2) {
    // İstisna2'yi işlemek için kod;
}
catch (Istisna3 exVar3) {
    // İstisna3'ü işlemek için kod;
}

...

catch (IstisnaN exVarN) {
    // İstisnayı işlemek için kod;
}
```

# finally (sonunda)

finally bloğu, bir istisna olup olmadığına bakılmaksızın her zaman yürütülür.

```
try {
    // Bir istisna atabilecek ifadeler;
}
catch (Istisna ex) {
    // İstisna1'i işlemek için kod;
}
finally {
    // ifadeler;
}
```

# InputMismatchException

```java
import java.util.*;

public class InputMismatchExceptionDemo {
    public static void main(String[] args) {
        Scanner giris = new Scanner(System.in);
        boolean giriseDevamEt = true;

        do {
            try {
            System.out.print("Bir tam sayi giriniz: ");
            int number = giris.nextInt();

            // Sonucu göster
            System.out.println(
            "Girilen sayi " + number + " dir.");

            giriseDevamEt = false;
            }
            catch (InputMismatchException ex) {
            System.out.println("Tekrar deneyin. (" +
            "Gecersiz giris: bir tam sayi gerekiyor)");
            giris.nextLine(); // Girisi sil
            }
        } while (giriseDevamEt);
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>
java InputMismatchExceptionDemo
Bir tam sayi giriniz: 3
Girilen sayi 3 dir.

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>
java InputMismatchExceptionDemo
Bir tam sayi giriniz: 3.0
Tekrar deneyin. (Gecersiz giris: bir tam sa
yi gerekiyor)
Bir tam sayi giriniz: 5.0
Tekrar deneyin. (Gecersiz giris: bir tam sa
yi gerekiyor)
Bir tam sayi giriniz: 4
Girilen sayi 4 dir.
```

# İstisna sınıfları (Exception classes)

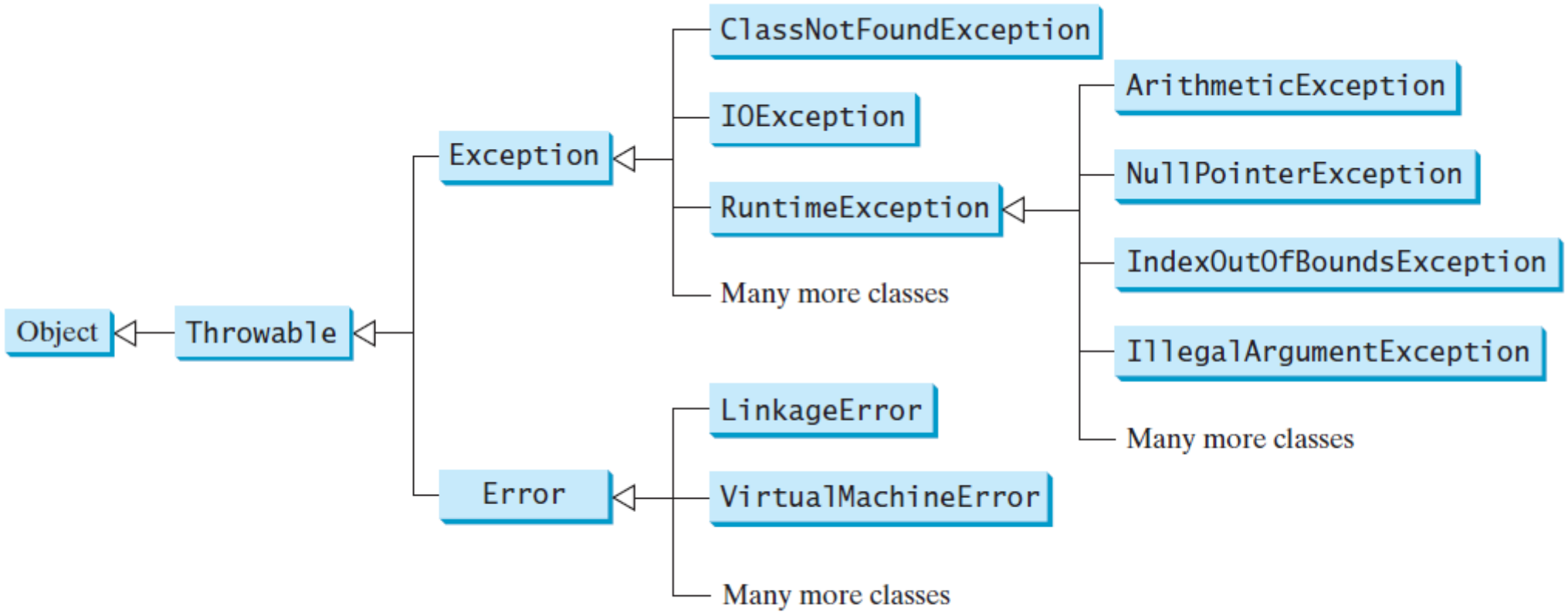



**FIGURE 12.1** Exceptions thrown are instances of the classes shown in this diagram, or of subclasses of one of these classes.

# Çalışma zamanı istisnasının alt sınıfları (Subclasses of RuntimeException)

**TABLE 12.3** Examples of Subclasses of `RuntimeException`

| *Class* | *Reasons for Exception* |
|---|---|
| `ArithmeticException` | Dividing an integer by zero. Note that floating-point arithmetic does not throw exceptions (see Appendix E, Special Floating-Point Values). |
| `NullPointerException` | Attempt to access an object through a `null` reference variable. |
| `IndexOutOfBoundsException` | Index to an array is out of range. |
| `IllegalArgumentException` | A method is passed an argument that is illegal or inappropriate. |

# Örnek: İstisna tanımlama, atma ve yakalama (IllegalArgumentException)

```java
public class CemberIstisnaIle {
    /** Cemberin yaricapi */
    private double yaricap;

    /** Olusturulan nesne sayisi */
    private static int nesneSayisi = 0;

    /** 1 yaricapli cember yapilandir */
    public CemberIstisnaIle() {
        this(1.0);
    }

    /** Belirtilen yaricap ile cember yapilandir */
    public CemberIstisnaIle(double yeniYaricap) {
        setYaricap(yeniYaricap);
        nesneSayisi++;
    }

    /** Yaricapi dondur */
    public double getYaricap() {
        return yaricap;
    }

    /** Yeni bir yaricap ata */
    public void setYaricap(double yeniYaricap)
            throws IllegalArgumentException {
        if (yeniYaricap >= 0)
            yaricap = yeniYaricap;
        else
            throw new IllegalArgumentException(
            "Yaricap negatif olamaz");
    }

    /** nesneSayisini dondur */
    public static int getNesneSayisi() {
        return nesneSayisi;
    }

    /** Cemberin alanini dondur */
    public double alanHesapla() {
        return yaricap * yaricap * 3.14159;
    }
}
```

# Örnek: İstisna tanımlama, atma ve yakalama (IllegalArgumentException)

```
1  public class TestCemberIstisnaIle {
2      public static void main(String[] args) {
3          try {
4              CemberIstisnaIle c1 = new CemberIstisnaIle(5);
5              CemberIstisnaIle c2 = new CemberIstisnaIle(-5);
6              CemberIstisnaIle c3 = new CemberIstisnaIle(0);
7          }
8          catch (IllegalArgumentException ex) {
9              System.out.println(ex);
10         }
11
12     System.out.println("Olusturulan nesne sayisi: " +
13     CemberIstisnaIle.getNesneSayisi());
14     }
15 }
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac CemberIstisnaIle.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>javac TestCemberIstisnaIle.java

C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestCemberIstisnaIle
java.lang.IllegalArgumentException: Yaricap negatif olamaz
Olusturulan nesne sayisi: 1
```

# İstisnaları yakalamak

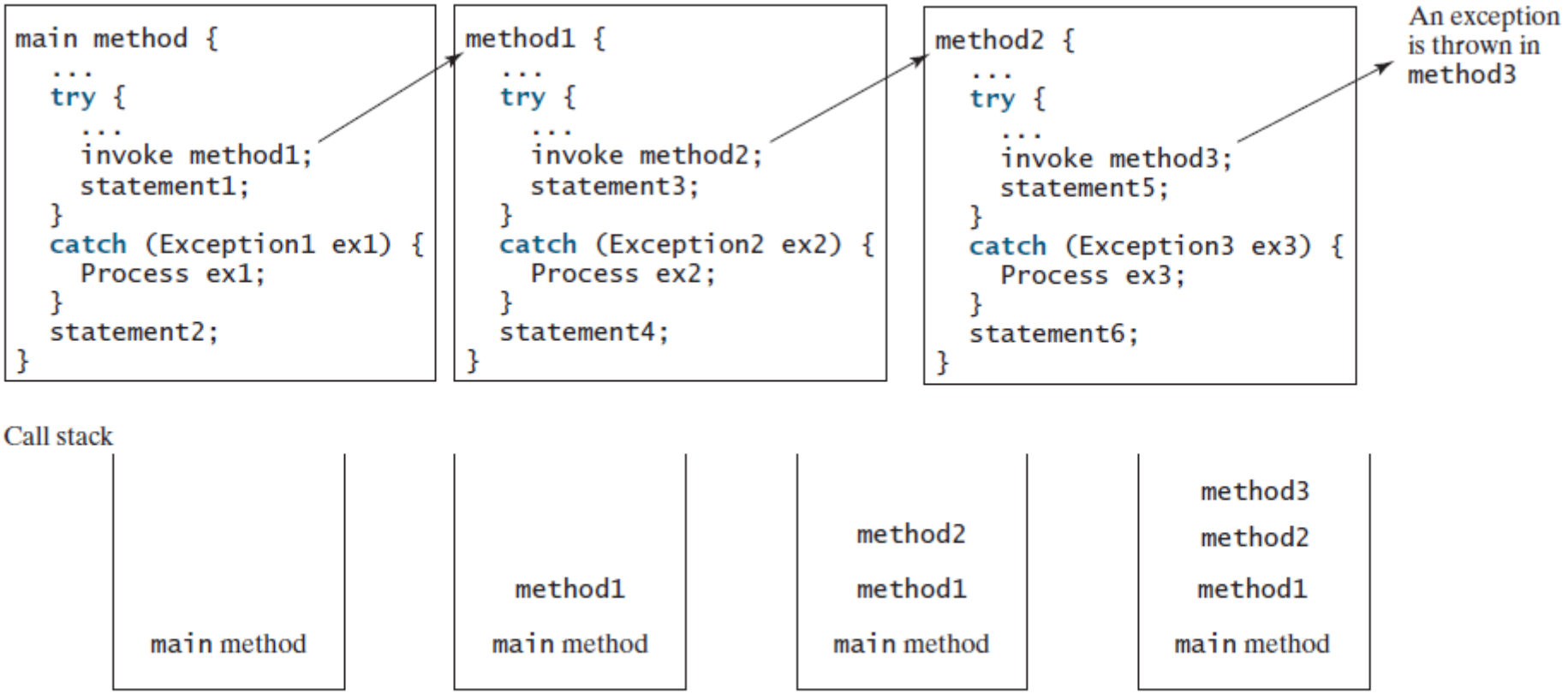



**FIGURE 12.3** If an exception is not caught in the current method, it is passed to its caller. The process is repeated until the exception is caught or passed to the `main` method.

# Ödev

```java
public class TestException {
    public static void main(String[] args) {
        try {
            System.out.println(sum(new int[] {1, 2, 3, 4, 5}));
        }
        catch (Exception ex) {
            ex.printStackTrace();
            System.out.println("\n" + ex.getMessage());
            System.out.println("\n" + ex.toString());

            System.out.println("\nTrace Info Obtained from getStackTrace");
            StackTraceElement[] traceElements = ex.getStackTrace();
                for (int i = 0; i < traceElements.length; i++) {
                System.out.print("method " + traceElements[i].getMethodName());
                System.out.print("(" + traceElements[i].getClassName() + ":");
                System.out.println(traceElements[i].getLineNumber() + ")");
                }
        }
    }

    private static int sum(int[] list) {
        int result = 0;
            for (int i = 0; i <= list.length; i++)
                result += list[i];
        return result;
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestException
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
        at TestException.sum(TestException.java:24)
        at TestException.main(TestException.java:4)

Index 5 out of bounds for length 5

java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5

Trace Info Obtained from getStackTrace
method sum(TestException:24)
method main(TestException:4)
```

# 8. Ders: JAVA ile Nesne Yönelimli Programlama
# File Class (Dosya Sınıfı)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# File Class (Dosya Sınıfı)

File sınıfı, bir dosya veya dizinin (klasör) özelliklerini elde etmek ve bir dosya veya dizini yeniden adlandırmak veya silmek için metotları içerir.

Programda depolanan veriler geçicidir; program sona erdiğinde kaybolurlar.

Bir programda oluşturulan verileri kalıcı olarak depolamak için, bunları bir diskteki veya başka bir kalıcı depolama aygıtındaki bir dosyaya kaydetmeniz gerekir.

Dosya daha sonra başka programlar tarafından taşınabilir ve okunabilir.

# File Class (Dosya Sınıfı)

Her dosya, dosya sistemindeki bir dizine yerleştirilir.

Mutlak bir dosya adı (veya tam adı), tam yolu ve sürücü harfiyle birlikte bir dosya adı içerir.

Örneğin, C:\Program Files\Java\jdk-15.0.1\bin\yeni2\Hesapla.java, Windows işletim sistemindeki Hesapla.java dosyasının mutlak dosya adıdır.

Burada C:\Program Files\Java\jdk-15.0.1\bin\yeni2, dosyanın dizin yolu olarak adlandırılır.

Mutlak dosya adları makineye bağlıdır. Windows işletim sisteminde "\", Unix veya Linux işletim sistemlerinde dizin ayırıcı "/" şeklindedir.

# File Class (Dosya Sınıfı)

Göreceli bir dosya adı, geçerli çalışma dizini ile ilişkilidir.

Göreceli bir dosya adı için tam dizin yolu atlanmıştır.

Örneğin, Hesapla.java göreceli bir dosya adıdır.

Mevcut çalışma dizini C:\Program Files\Java\jdk-15.0.1\bin\yeni2 ise,

mutlak dosya adı C:\Program Files\Java\jdk-15.0.1\bin\yeni2\Hesapla.java olacaktır.

# File Class (Dosya Sınıfı)

File sınıfının, dosyaların ve yol adlarının makineye bağlı karmaşıklıklarının çoğunu makineden bağımsız bir şekilde ele alan bir soyutlama sağlaması amaçlanmıştır.

File sınıfı, Şekil 12.6'da gösterildiği gibi, dosya ve dizin özelliklerini edinme ve dosya ve dizinleri yeniden adlandırma ve silme metotlarını içerir.

Ancak, File sınıfı, dosya içeriklerini okuma ve yazma metotlarını içermez.

# File Class (Dosya Sınıfı)

Dosya adı bir dizedir. File sınıfı, dosya adı ve dizin yolu için bir sarmalayıcı sınıfıdır.

Örneğin, new File(" C:\\Program Files\\Java\\jdk-15.0.1\\bin\\yeni2 ")
C:\Program Files\Java\jdk-15.0.1\bin\yeni2 dizini için bir File nesnesi oluşturur. (windows)

new File("C:\\Program Files\\Java\\jdk-15.0.1\\bin\\yeni2\\Hesapla.java")
C:\Program Files\Java\jdk-15.0.1\bin\yeni2\Hesapla.java dosyası için bir File nesnesi oluşturur. (windows)

Unix veya Linux işletim sistemlerinde …

# File Class (Dosya Sınıfı)

new File("C:\\Program Files\\Java\\jdk-15.0.1\\bin\\yeni2\\Hesapla.java")
Windows için dizin ayırıcı bir ters eğik çizgidir (\). (backslash)

Ters eğik çizgi, Java'da özel bir karakterdir ve bir dizin değişmezinde \\ olarak yazılmalıdır.

| Kaçış kodları | adı | unicode kodu | decimal değeri |
|---|---|---|---|
| \b | backspace | \u0008 | 8 |
| \t | tab | \u0009 | 9 |
| \n | linefeed | \u000A | 10 |
| \f | formfeed | \u000C | 12 |
| \r | carriage return | \u000D | 13 |
| \\ | backslash | \u005C | 92 |
| \" | double quote | \u0022 | 34 |

# File Class (Dosya Sınıfı)

Nesnenin bir dizini temsil edip etmediğini kontrol etmek için File sınıfının isDirectory() metodunu

ve nesnenin bir dosyayı temsil edip etmediğini kontrol etmek için isFile() metodunu kullanabilirsiniz.

Bir File örneğinin (nesne) oluşturulması, makinede bir dosya oluşturmaz.

Var olup olmadığına bakılmaksızın herhangi bir dosya adı için bir File örneği oluşturabilirsiniz.

Dosyanın var olup olmadığını kontrol etmek için bir File örneğinde exists() metodunu çağırabilirsiniz.

# java.io.File

| java.io.File | |
|---|---|
| `+File(pathname: String)` | Creates a `File` object for the specified path name. The path name may be a directory or a file. |
| `+File(parent: String, child: String)` | Creates a `File` object for the child under the directory parent. The child may be a file name or a subdirectory. |
| `+File(parent: File, child: String)` | Creates a `File` object for the child under the directory parent. The parent is a `File` object. In the preceding constructor, the parent is a string. |
| `+exists(): boolean` | Returns true if the file or the directory represented by the `File` object exists. |
| `+canRead(): boolean` | Returns true if the file represented by the `File` object exists and can be read. |
| `+canWrite(): boolean` | Returns true if the file represented by the `File` object exists and can be written. |
| `+isDirectory(): boolean` | Returns true if the `File` object represents a directory. |
| `+isFile(): boolean` | Returns true if the `File` object represents a file. |
| `+isAbsolute(): boolean` | Returns true if the `File` object is created using an absolute path name. |
| `+isHidden(): boolean` | Returns true if the file represented in the `File` object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period(.) character. |
| `+getAbsolutePath(): String` | Returns the complete absolute file or directory name represented by the `File` object. |
| `+getCanonicalPath(): String` | Returns the same as `getAbsolutePath()` except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows). |
| `+getName(): String` | Returns the last name of the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getName()` returns `test.dat`. |
| `+getPath(): String` | Returns the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getPath()` returns `c:\book\test.dat`. |
| `+getParent(): String` | Returns the complete parent directory of the current directory or the file represented by the `File` object. For example, new `File("c:\\book\\test.dat").getParent()` returns `c:\book`. |
| `+lastModified(): long` | Returns the time that the file was last modified. |
| `+length(): long` | Returns the size of the file, or 0 if it does not exist or if it is a directory. |
| `+listFile(): File[]` | Returns the files under the directory for a directory `File` object. |
| `+delete(): boolean` | Deletes the file or directory represented by this `File` object.The method returns true if the deletion succeeds. |
| `+renameTo(dest: File): boolean` | Renames the file or directory represented by this `File` object to the specified name represented in dest. The method returns true if the operation succeeds. |
| `+mkdir(): boolean` | Creates a directory represented in this `File` object. Returns true if the the directory is created successfully. |
| `+mkdirs(): boolean` | Same as `mkdir()` except that it creates directory along with its parent directories if the parent directories do not exist. |

**FIGURE 12.6** The `File` class can be used to obtain file and directory properties, to delete and rename files and directories, and to create directories.

# java.io.File

| java.io.File | |
|---|---|
| `+File(pathname: String)` | Creates a `File` object for the specified path name. The path name may be a directory or a file. |
| `+File(parent: String, child: String)` | Creates a `File` object for the child under the directory parent. The child may be a file name or a subdirectory. |
| `+File(parent: File, child: String)` | Creates a `File` object for the child under the directory parent. The parent is a `File` object. In the preceding constructor, the parent is a string. |
| `+exists(): boolean` | Returns true if the file or the directory represented by the `File` object exists. |
| `+canRead(): boolean` | Returns true if the file represented by the `File` object exists and can be read. |
| `+canWrite(): boolean` | Returns true if the file represented by the `File` object exists and can be written. |
| `+isDirectory(): boolean` | Returns true if the `File` object represents a directory. |
| `+isFile(): boolean` | Returns true if the `File` object represents a file. |
| `+isAbsolute(): boolean` | Returns true if the `File` object is created using an absolute path name. |
| `+isHidden(): boolean` | Returns true if the file represented in the `File` object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period(.) character. |
| `+getAbsolutePath(): String` | Returns the complete absolute file or directory name represented by the `File` object. |
| `+getCanonicalPath(): String` | Returns the same as `getAbsolutePath()` except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows). |
| `+getName(): String` | Returns the last name of the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getName()` returns `test.dat`. |

# java.io.File

| | |
|---|---|
| `+getPath(): String` | Returns the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getPath()` returns `c:\book\test.dat`. |
| `+getParent(): String` | Returns the complete parent directory of the current directory or the file represented by the `File` object. For example, new `File("c:\\book\\test.dat").getParent()` returns `c:\book`. |
| `+lastModified(): long` | Returns the time that the file was last modified. |
| `+length(): long` | Returns the size of the file, or 0 if it does not exist or if it is a directory. |
| `+listFile(): File[]` | Returns the files under the directory for a directory `File` object. |
| `+delete(): boolean` | Deletes the file or directory represented by this `File` object.The method returns true if the deletion succeeds. |
| `+renameTo(dest: File): boolean` | Renames the file or directory represented by this `File` object to the specified name represented in dest. The method returns true if the operation succeeds. |
| `+mkdir(): boolean` | Creates a directory represented in this `File` object. Returns true if the the directory is created successfully. |
| `+mkdirs(): boolean` | Same as `mkdir()` except that it creates directory along with its parent directories if the parent directories do not exist. |

**FIGURE 12.6** The `File` class can be used to obtain file and directory properties, to delete and rename files and directories, and to create directories.

# Örnek 1: java.io.File

```java
public class TestFileClass {
    public static void main(String[] args) {
        java.io.File dosya = new java.io.File("deneme.txt");
        System.out.println("Dosya var mi? " + dosya.exists());
        System.out.println("Dosyanin uzunlugu " + dosya.length() + " bytes");
        System.out.println("Dosya okunabilir mi? " + dosya.canRead());
        System.out.println("Dosya yazilabilir mi? " + dosya.canWrite());
        System.out.println("Bir dizin mi? " + dosya.isDirectory());
        System.out.println("Bir dosya mi? " + dosya.isFile());
        System.out.println("Mutlak mi? " + dosya.isAbsolute());
        System.out.println("Gizli mi? " + dosya.isHidden());
        System.out.println("Mutlak yolu " + dosya.getAbsolutePath());
        System.out.println("Degisiklik tarihi " + new java.util.Date(dosya.lastModified()));
    }
}
```

```
C:\Program Files\Java\jdk-16.0.1\bin\yeni2>java TestFileClass
Dosya var mi? true
Dosyanin uzunlugu 0 bytes
Dosya okunabilir mi? true
Dosya yazilabilir mi? true
Bir dizin mi? false
Bir dosya mi? true
Mutlak mi? false
Gizli mi? false
Mutlak yolu C:\Program Files\Java\jdk-16.0.1\bin\yeni2\deneme.txt
Degisiklik tarihi Mon May 10 00:44:30 TRT 2021
```

# Dosya Giriş ve Çıkış G/Ç (File Input and Output I/O)

Bir dosyadan metin verilerini okumak için Scanner sınıfını

ve bir dosyaya metin verilerini yazmak için PrintWriter sınıfını kullanılır.

Bir File nesnesi, bir dosyanın veya yolun özelliklerini kapsüller (içerir),

ancak bir dosya oluşturma veya bir dosyaya veri yazma veya dosyadan veri okuma metotlarını içermez (G/Ç (I/O)).

G/Ç gerçekleştirmek için, uygun Java G/Ç sınıflarını kullanarak nesneler oluşturmamız gerekir.

# Dosya Giriş ve Çıkış G/Ç (File Input and Output I/O)

Nesneler, bir dosyadan/dosyaya veri okuma/yazma yöntemlerini içerir.

İki tür dosya vardır: metin ve ikili.

Metin dosyaları esasen diskteki karakterlerdir.

Scanner ve PrintWriter sınıfları kullanılarak bir metin dosyasından stringlerin ve sayısal değerlerin nasıl okunacağı veya bir metin dosyasına stringlerin ve sayısal değerlerin nasıl yazılacağı gösterilecektir.

# PrintWriter Kullanarak Veri Yazma

java.io.PrintWriter sınıfı, bir dosya oluşturmak ve bir metin dosyasına veri yazmak için kullanılır.
Öncelikle, aşağıdaki gibi bir metin dosyası için bir PrintWriter nesnesi oluşturmanız gerekir:

PrintWriter cikis = new PrintWriter(dosya adı);

Ardından, bir dosyaya veri yazmak için PrintWriter nesnesindeki print, println ve printf metotlarını çağırabiliriz.

# java.io.PrintWriter

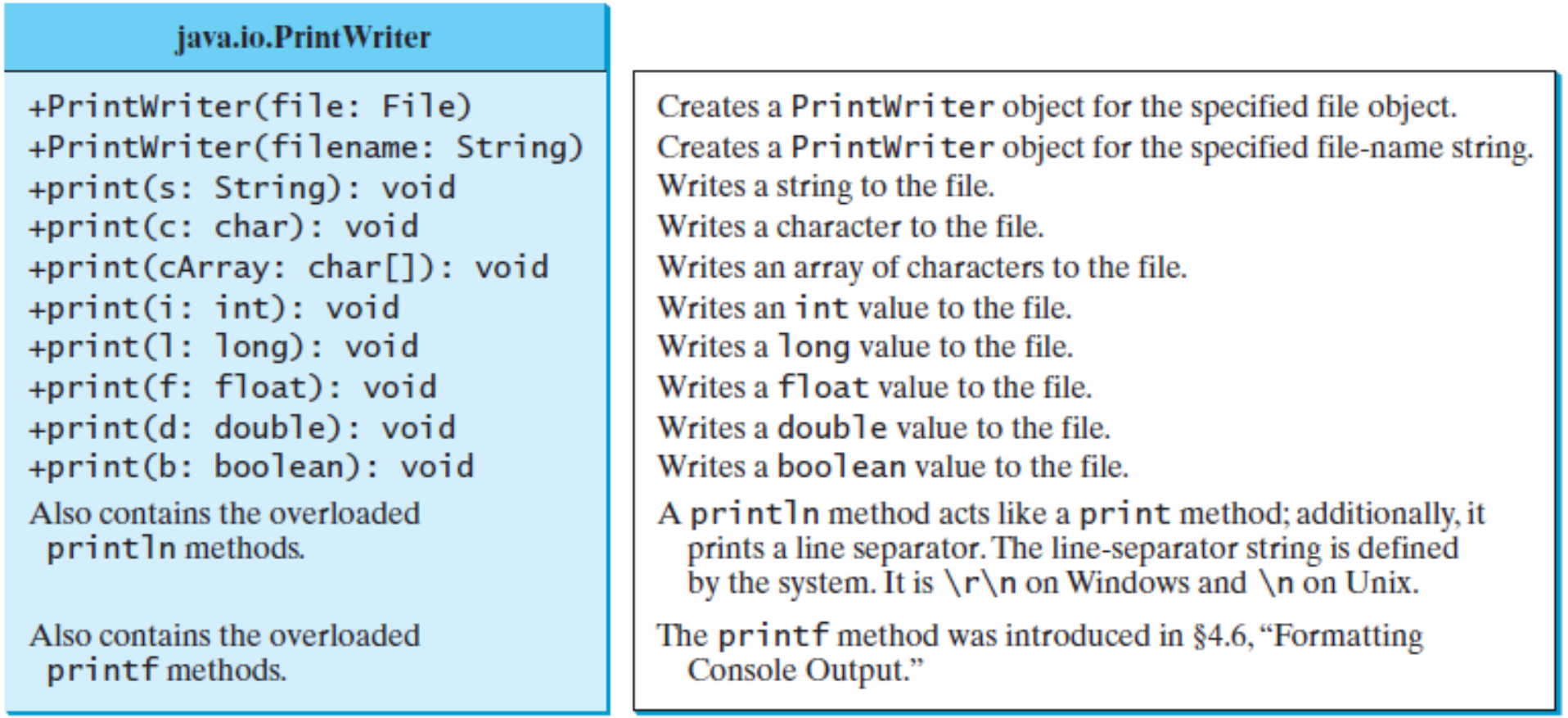
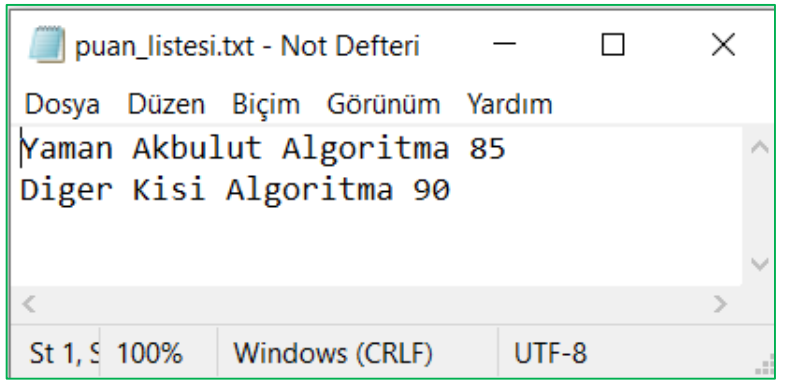


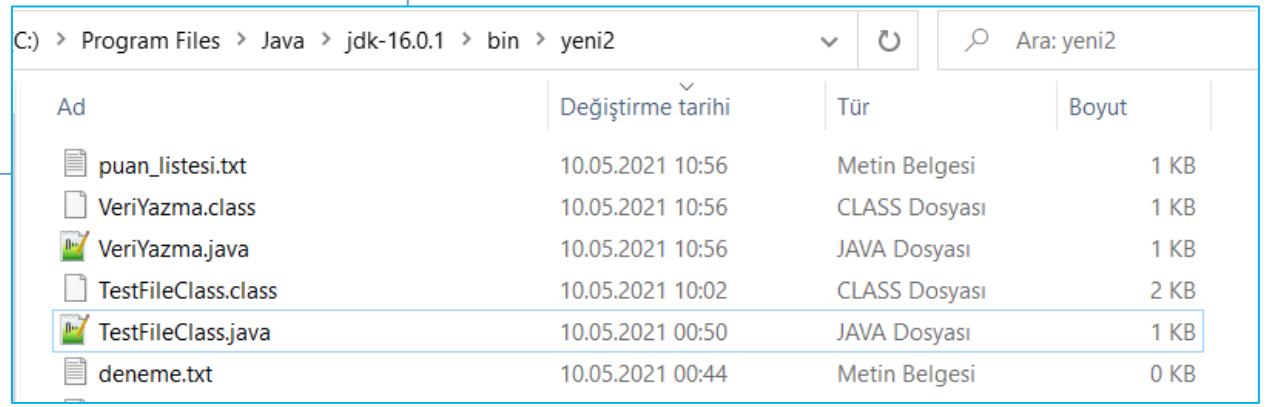

**FIGURE 12.8** The `PrintWriter` class contains the methods for writing data to a text file.

# Örnek 2: java.io.PrintWriter

```
1  import java.io.*;
2
3  public class VeriYazma {
4      public static void main(String[] args) throws IOException {
5          File dosya = new File("puan_listesi.txt");
6              if (dosya.exists()) {
7              System.out.println("Dosya mevcut");
8              System.exit(1);
9              }
10
11         // Dosya olusturma
12         PrintWriter cikis = new PrintWriter(dosya);
13
14         // Dosyaya yazma
15         cikis.print("Yaman Akbulut Algoritma ");
16         cikis.println(85);
17         cikis.print("Diger Kisi Algoritma ");
18         cikis.println(90);
19
20         // Dosyayı kapatma
21         cikis.close();
22     }
23 }
```

```
C:\Program Files\Java\jdk-16.0.1\bin\
yeni2>javac VeriYazma.java

C:\Program Files\Java\jdk-16.0.1\bin\
yeni2>java VeriYazma

C:\Program Files\Java\jdk-16.0.1\bin\
yeni2>
```



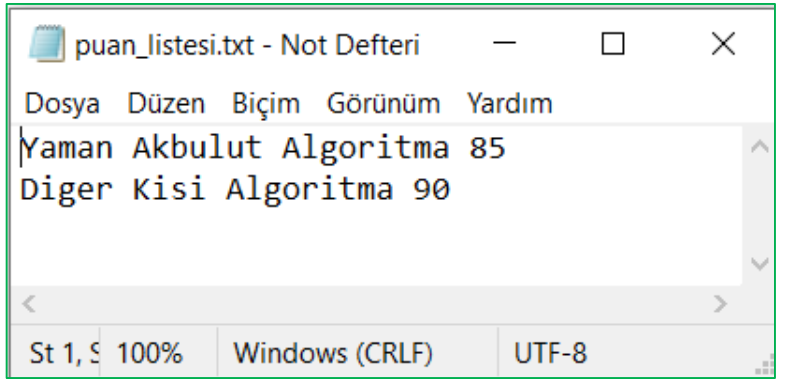



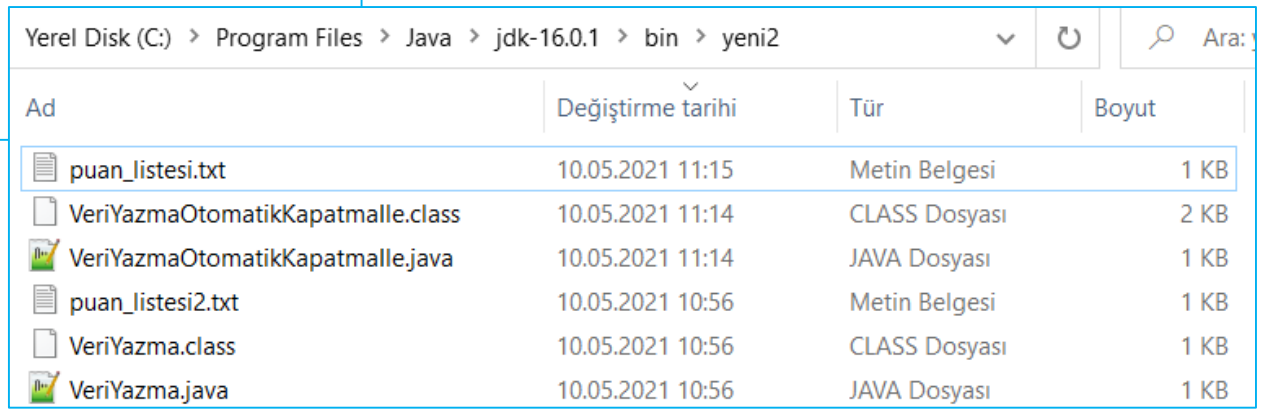

Dosyayı kapatmak için close() metodu kullanılmalıdır (satır 21). Bu metot kullanılmazsa, veriler dosyaya düzgün kaydedilemeyebilir.

# Örnek 3: java.io.PrintWriter ve try-with-resources

```java
import java.io.*;

public class VeriYazmaOtomatikKapatmaIle {
    public static void main(String[] args) throws Exception {
        File dosya = new File("puan_listesi.txt");
            if (dosya.exists()) {
            System.out.println("Dosya mevcut");
            System.exit(0);
            }

        try(
            // Dosya olusturma
            PrintWriter cikis = new PrintWriter(dosya);
        ){
            // Dosyaya yazma
            cikis.print("Yaman Akbulut Algoritma ");
            cikis.println(85);
            cikis.print("Diger Kisi Algoritma ");
            cikis.println(90);
        }
    }
}
```

```
C:\Program Files\Java\jdk-16.0.1\bin\yeni2>
javac VeriYazmaOtomatikKapatmaIle.java

C:\Program Files\Java\jdk-16.0.1\bin\yeni2>
java VeriYazmaOtomatikKapatmaIle
Dosya mevcut

C:\Program Files\Java\jdk-16.0.1\bin\yeni2>
java VeriYazmaOtomatikKapatmaIle
```




```
try(kaynakların tanımlanması ve oluşturulması){
   kaynakların kullanımı…
   //try-with-resources
  }
```

# Scanner Kullanarak Veri Okuma

java.util.Scanner sınıfı, konsoldan dizeleri (string) ve ilkel (primitive) değerleri okumak için kullandık.

Bir Scanner, girdisini boşluk karakterleriyle sınırlandırılmış belirteçlere (token) böler.

Klavyeden okumak için, aşağıdaki gibi System.in için bir Scanner oluşturulur:

```
Scanner giris = new Scanner(System.in);
```

Bir dosyadan okumak için aşağıdaki gibi bir Scanner oluşturun:

```
Scanner giris = new Scanner (new File(dosya adı));
```

# java.util.Scanner

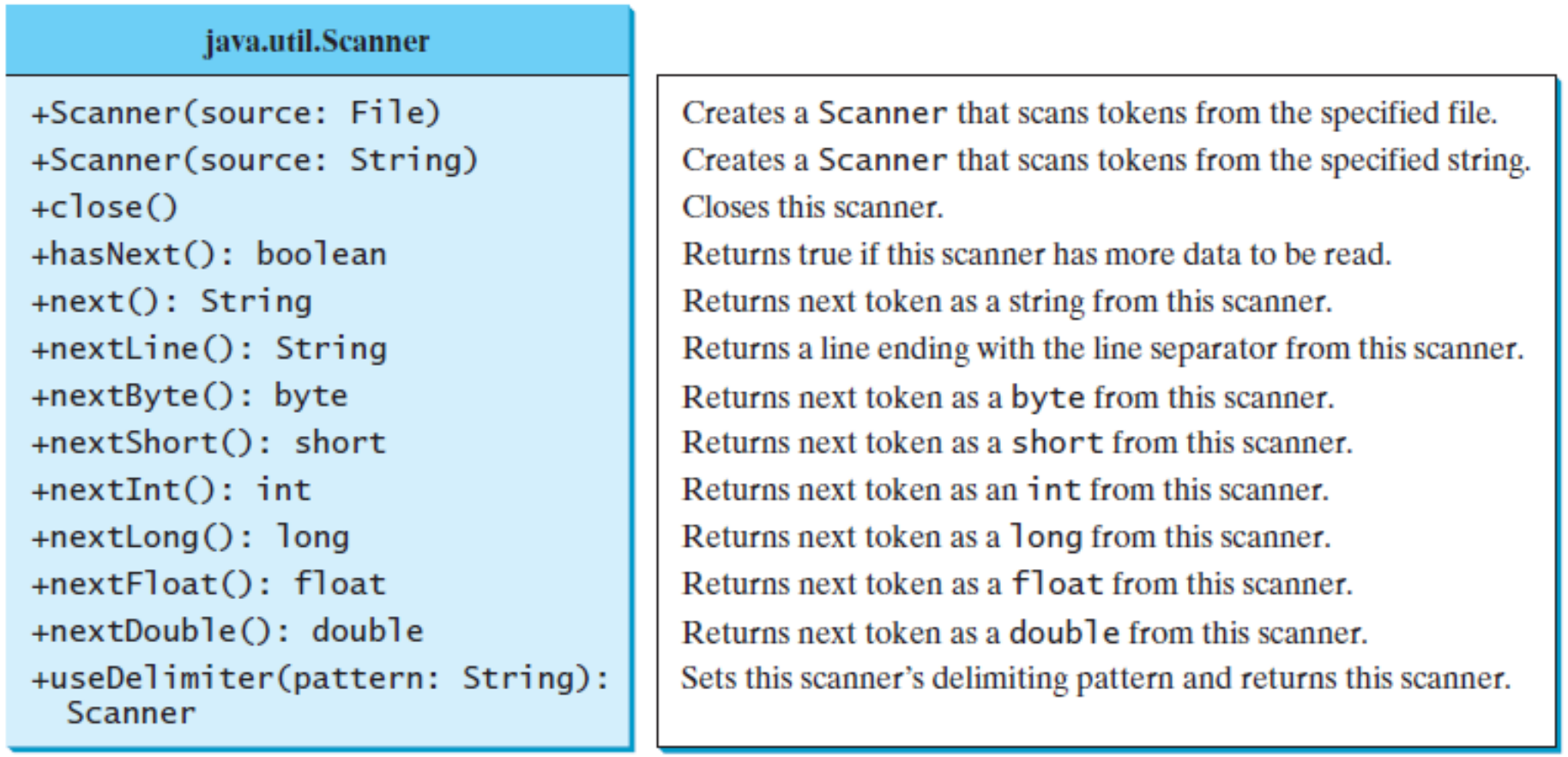



**FIGURE 12.9** The Scanner class contains the methods for scanning data.

# Örnek 4: java.util.Scanner

```java
import java.util.Scanner;

public class VeriOkuma {
    public static void main(String[] args) throws Exception {
        // File örneği oluşturma
        java.io.File dosya = new java.io.File("puan_listesi.txt");

        // Dosya için bir Scanner oluşturma
        Scanner giris = new Scanner(dosya);

        // Dosyadan veri okuma
        while (giris.hasNext()) {
        String isim = giris.next();
        String soyIsim = giris.next();
        String dersAdi = giris.next();
        int notDegeri = giris.nextInt();
        System.out.println(isim + " " + soyIsim + " " + dersAdi + " " + notDegeri);
        }

        // Dosyayı kapatma
        giris.close();
    }
}
```

```
C:\Program Files\Java\jdk-16.0.1\bin\yeni2>javac VeriOkuma.java

C:\Program Files\Java\jdk-16.0.1\bin\yeni2>java VeriOkuma
Yaman Akbulut Algoritma 85
Diger Kisi Algoritma 90
```

# Örnek 4: java.util.Scanner

```java
import java.util.Scanner;

public class VeriOkuma {
    public static void main(String[] args) throws Exception {
        // File örneği oluşturma
        java.io.File dosya = new java.io.File("puan_listesi.txt");

        // Dosya için bir Scanner oluşturma
        Scanner giris = new Scanner(dosya);

        // Dosyadan veri okuma
        while (giris.hasNext()) {
        String isim = giris.next();
        String soyIsim = giris.next();
        String dersAdi = giris.next();
        int notDegeri = giris.nextInt();
        System.out.println(isim + " " + soyIsim + " " + dersAdi + " " + notDegeri);
        }

        // Dosyayı kapatma
        giris.close();
    }
}
```

```
C:\Program Files\Java\jdk-16.0.1\bin\yeni2>java VeriOkuma
Exception in thread "main" java.io.FileNotFoundException: puan_listesi.txt (Sistem belirtilen dosyayı bulamıyor)
        at java.base/java.io.FileInputStream.open0(Native Method)
        at java.base/java.io.FileInputStream.open(FileInputStream.java:211)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:153)
        at java.base/java.util.Scanner.<init>(Scanner.java:639)
        at VeriOkuma.main(VeriOkuma.java:9)
```

# Scanner nasıl çalışır?

nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble() ve next() metotları, belirteç (token) okuma metotları olarak bilinir,

çünkü bunlar sınırlayıcılarla (delimiter) ayrılmış belirteçleri (token) okurlar.

Varsayılan olarak, sınırlayıcılar boşluk (whitespace) karakterleridir.

Sınırlayıcılar (delimiter) için yeni bir desen ayarlamak için useDelimiter(String regex) metodunu kullanabiliriz.

whitespace karakterler: ' ', \t, \f, \r, \n

# Örnek 5: Metin değişme

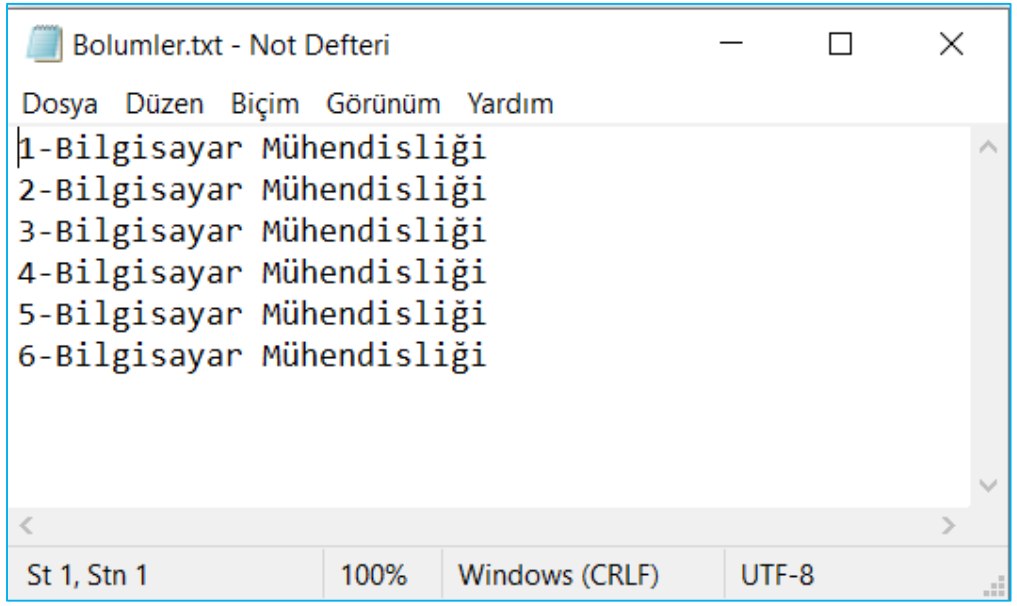
```java
import java.io.*;
import java.util.*;
public class MetinDegisme {
    public static void main(String[] args) throws Exception {
        // Komut satırı parametre kullanımını kontrol etme
        if (args.length != 4) {
            System.out.println(
            "Kullanim: java MetinDegisme kaynakDosya hedefDosya eskiMetin yeniMetin");
            System.exit(1);
        }
        // Kaynak dosya mevcut mu?
        File kaynakDosya = new File(args[0]);
        if (!kaynakDosya.exists()) {
            System.out.println("Kaynak dosya " + args[0] + " mevcut degil");
            System.exit(2);
        }
        // Hedef dosya mevcut mu?
        File hedefDosya = new File(args[1]);
        if (hedefDosya.exists()) {
            System.out.println("Hedef dosya " + args[1] + " zaten mevcut");
            System.exit(3);
        }
        try (
            // giris ve cikis dosyalarını oluşturma
            Scanner giris = new Scanner(kaynakDosya);
            PrintWriter cikis = new PrintWriter(hedefDosya);
        ) {
            while (giris.hasNext()) {
            String s1 = giris.nextLine();
            String s2 = s1.replaceAll(args[2], args[3]);
            cikis.println(s2);
            }
        }
    }
}
```

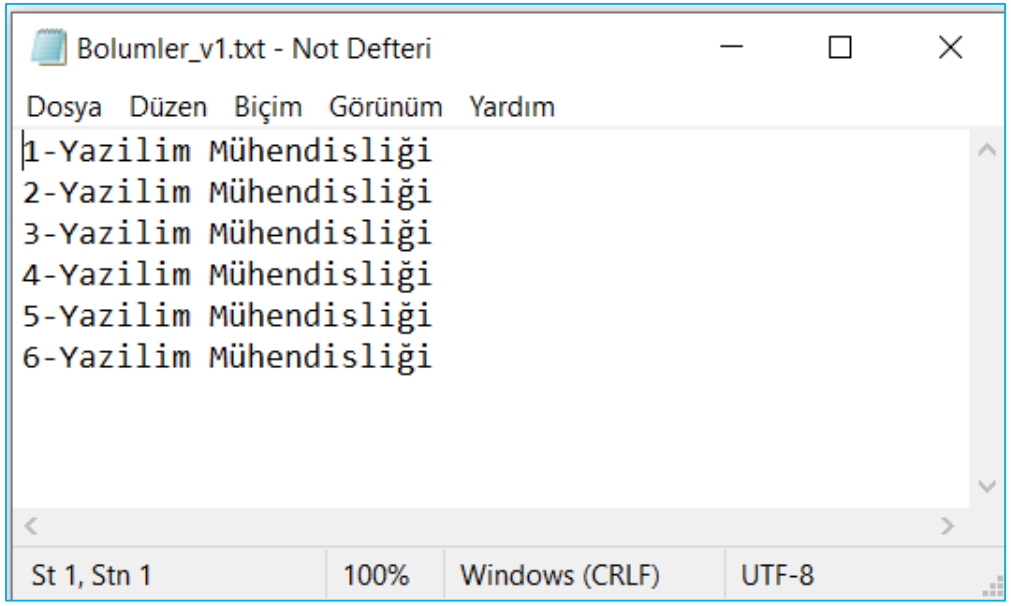
# Örnek 5: Metin değişme

```
java MetinDegisme kaynakDosya hedefDosya eskiMetin yeniMetin
```

```
C:\Program Files\Java\jdk-16.0.1\bin\yeni2>javac MetinDegisme.java

C:\Program Files\Java\jdk-16.0.1\bin\yeni2>java MetinDegisme Bolumler.txt Bolumler_v1.txt Bilgisayar Yazilim
```

```
C:\Program Files\Java\jdk-16.0.1\bin\yeni2>java MetinDegisme Bolumler.txt Bolumler_v1.txt Bilgisayar Yazilim
Kaynak dosya Bolumler.txt mevcut degil

C:\Program Files\Java\jdk-16.0.1\bin\yeni2>java MetinDegisme Bolumler.txt Bolumler_v1.txt Bilgisayar Yazilim
Hedef dosya Bolumler_v1.txt zaten mevcut
```



# 9. Ders: JAVA ile Nesne Yönelimli Programlama
# Abstract Classes (Soyut Sınıflar)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama

  - Ders 29: Java Polimorfizmi / Yöntem Aşırı Yüklemesi (Method Overloading) (izle)
  - Ders 30: Java Polimorfizmi / Yöntem Geçersiz Kılma (Method Overriding) (izle)
  - Ders 31: super Anahtar Sözcüğünün Kullanımı (izle)
  - Ders 32: final Anahtar Sözcüğünün Kullanımı (izle)
  - Ders 33: Soyut Sınıf (Abstract Class) (izle)
  - Ders 34: Arayüz (Interface) (izle)

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# Giriş

Bir üst sınıf, ilgili alt sınıflar için ortak davranışı tanımlar.

Sınıflar için ortak davranışı tanımlamak için bir arayüz (interface) kullanılabilir (ilgisiz sınıflar dahil).

Bir sayı dizisini veya dizeyi (string) sıralamak için java.util.Arrays.sort metodunu kullanabilirsiniz.

Bir geometrik nesne dizisini sıralamak için aynı sıralama (sort) metodunu uygulayabilir misiniz?

# Giriş

Bir geometrik nesne dizisini sıralamak için aynı sıralama (sort) metodunu uygulayabilir misiniz?

Bu tür bir kod yazabilmek için arayüzler hakkında bilgi sahibi olmamız gerekir.

Bir arayüz (interface), sınıflar için ortak davranışı tanımlamak içindir (ilgisiz sınıflar dahil).

Arayüzleri tartışmadan önce, yakından ilgili bir konuya bakalım: soyut sınıflar (abstract classes)

# Abstract Classes (Soyut Sınıflar)

Nesne oluşturmak için soyut bir sınıf (abstract class) kullanılamaz.

Soyut bir sınıf, somut alt sınıflarda uygulanan soyut metotlar içerebilir.

Kalıtım hiyerarşisinde, sınıflar her yeni alt sınıfla daha spesifik ve somut hale gelir.

Bir alt sınıftan bir üst sınıfa geçerseniz, sınıflar daha genel ve daha az spesifik hale gelir.

# Abstract Classes (Soyut Sınıflar)

Sınıf tasarımı, bir üst sınıfın alt sınıflarının ortak özelliklerini içermesini sağlamalıdır.

Bazen bir üst sınıf o kadar soyuttur ki herhangi bir özel örnek oluşturmak için kullanılamaz.

Böyle bir sınıfa soyut sınıf (abstract class) denir.

# Abstract Classes (Soyut Sınıflar)

Önceki bölümlerde GeometrikSekil, Cember ve Dikdortgen için süper sınıf olarak tanımlandı.

GeometrikSekil, geometrik nesnelerin ortak özelliklerini modeller.

Hem Cember hem de Dikdortgen, bir çemberin ve bir dikdörtgenin alanını ve çevresini hesaplamak için getAlan() ve getCevre() metotlarını içerir.

Tüm geometrik nesneler için alanları ve çevreleri hesaplayabildiğinizden, getAlan() ve getCevre() metotlarını GeometrikSekil sınıfında tanımlamak daha iyidir.

# Abstract Classes (Soyut Sınıflar)

Ancak, bu metotlar GeometrikSekil sınıfında uygulanamaz çünkü bunların uygulanması belirli geometrik nesne türüne bağlıdır.

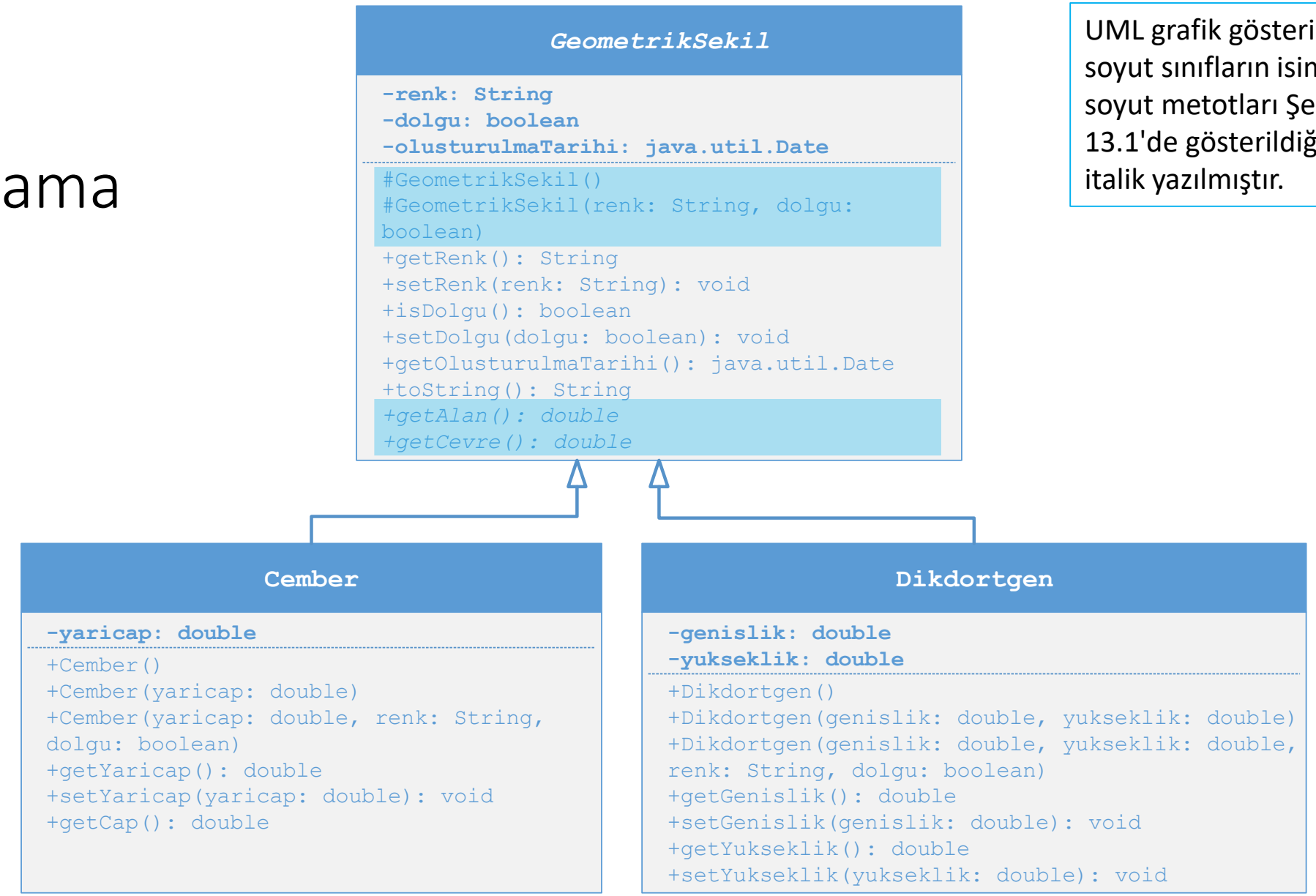
Bu tür metotlara soyut (abstract) metotlar olarak atıfta bulunulur ve metot başlığındaki abstract belirleyici kullanılarak belirtilir.

GeometrikSekil'de metotları tanımladıktan sonra, sınıf soyut bir sınıf haline gelir.

Soyut sınıflar, sınıf başlığındaki abstract belirleyici kullanılarak gösterilir.

UML grafik gösteriminde, soyut sınıfların isimleri ve soyut metotları Şekil 13.1'de gösterildiği gibi italik yazılmıştır.

# Soyut metot tanımlama




UML grafik gösteriminde, soyut sınıfların isimleri ve soyut metotları Şekil 13.1'de gösterildiği gibi italik yazılmıştır.

Şekil 13.1. Yeni GeometrikSekil sınıfı soyut metotlar içermektedir.

# GeometrikSekil sınıfı

```java
public abstract class GeometrikSekil {
    private String renk = "beyaz";
    private boolean dolgu;
    private java.util.Date olusturulmaTarihi;

    /** Varsayilan bir geometrik nesne olustur */
    protected GeometrikSekil() {
        olusturulmaTarihi = new java.util.Date();
    }

    /** Belirtilen renk ve dolgu degeri ile
    * varsayilan bir geometrik nesne olustur */
    protected GeometrikSekil(String renk, boolean dolgu) {
        olusturulmaTarihi = new java.util.Date();
        this.renk = renk;
        this.dolgu = dolgu;
    }

    /** Renk dondur */
    public String getRenk() {
        return renk;
    }

    /** Yeni renk ata */
    public void setRenk(String renk) {
        this.renk = renk;
    }

    /** Dolgu dondur. dolgu boolean oldugundan,
    get metodunu is..... seklinde adlandirildi */
    public boolean isDolgu() {
        return dolgu;
    }

    /** Dolgu ata */
    public void setDolgu(boolean dolgu) {
        this.dolgu = dolgu;
    }

    /** OlusturulmaTarihi dondur */
    public java.util.Date getOlusturulmaTarihi() {
        return olusturulmaTarihi;
    }

    /** Bu nesnenin string sunumunu dondur */
    @Override
    public String toString() {
        return "Olusturulma Tarihi: " + olusturulmaTarihi
        + "\nrenk: " + renk + " ve dolgu: " + dolgu;
    }

    /** Soyut metot getAlan */
    public abstract double getAlan();

    /** Soyut metot getCevre */
    public abstract double getCevre();
}
```

# Cember sınıfı

```java
public class Cember extends GeometrikSekil{
    private double yaricap;

    public Cember() {
    }

    public Cember(double yaricap) {
        this.yaricap = yaricap;
    }

    public Cember(double yaricap, String renk, boolean dolgu) {
        this.yaricap = yaricap;
        setRenk(renk);
        setDolgu(dolgu);
    }

    /** Yaricap dondur */
    public double getYaricap() {
        return yaricap;
    }

    /** Yeni bir yaricap ata */
    public void setYaricap(double yaricap) {
        this.yaricap = yaricap;
    }

    /** Alan dondur */
    public double getAlan() {
        return yaricap * yaricap * Math.PI;
    }

    /** Cap dondur */
    public double getCap() {
        return 2 * yaricap;
    }

    /** Cevre dondur */
    public double getCevre() {
        return 2 * yaricap * Math.PI;
    }

    /** Cember bilgisini yazdir */
    public void yazdirCember() {
        System.out.println("Cember, " + getOlusturulmaTarihi() +
        " tarihinde olusturuldu ve yaricapi " + yaricap + " dir.");
    }

    // Üst sınıfta tanımlanan toString metodunu gecersiz kılma
    public String toString() {
        return super.toString() + "\n yari cap: " + yaricap;
    }
}
```

# Dikdortgen sınıfı

```java
public class Dikdortgen extends GeometrikSekil {
    private double genislik;
    private double yukseklik;

    public Dikdortgen() {
    }

    public Dikdortgen(double genislik, double yukseklik) {
        this.genislik = genislik;
        this.yukseklik = yukseklik;
    }

    public Dikdortgen(double genislik, double yukseklik,
                      String renk, boolean dolgu) {
        this.genislik = genislik;
        this.yukseklik = yukseklik;
        setRenk(renk);
        setDolgu(dolgu);
    }

    /** Genislik dondur */
    public double getGenislik() {
        return genislik;
    }

    /** Yeni bir genislik ata */
    public void setGenislik(double genislik) {
        this.genislik = genislik;
    }

    /** Yukseklik dondur */
    public double getYukseklik() {
        return yukseklik;
    }

    /** Yeni bir yukseklik ata */
    public void setYukseklik(double yukseklik) {
        this.yukseklik = yukseklik;
    }

    /** Alan dondur */
    public double getAlan() {
        return genislik * yukseklik;
    }

    /** Cevre dondur */
    public double getCevre() {
        return 2 * (genislik + yukseklik);
    }
}
```

# Abstract Classes (Soyut Sınıflar)

Soyut sınıflar normal sınıflar gibidir,

ancak new operatörünü kullanarak soyut sınıfların örneklerini oluşturamazsınız.

Soyut bir metot, uygulama (implementation) olmadan tanımlanır.

Uygulanması (implementation) alt sınıflar tarafından sağlanır.

Soyut (abstract) metotlar içeren bir sınıf, soyut (abstract) olarak tanımlanmalıdır.

# Abstract Classes (Soyut Sınıflar)

Soyut sınıftaki yapıcı, yalnızca alt sınıflar tarafından kullanıldığından korumalı (protected) olarak tanımlanır.

Somut bir alt sınıfın bir örneğini oluşturduğunuzda, üst sınıfın yapıcısı, üst sınıfta tanımlanan veri alanlarını başlatmak için çağrılır.

# Abstract Classes (Soyut Sınıflar)

GeometrikSekil soyut sınıfı, geometrik nesneler için ortak özellikleri (veriler ve metotlar) tanımlar ve uygun yapıcılar sağlar.

Geometrik nesnelerin alanlarını ve çevresini nasıl hesaplanacağı o an için bilinmediğinden getAlan() ve getCevre() soyut metotlar olarak tanımlanır.

Bu metotlar alt sınıflarda uygulanmaktadır.

Cember ve Dikdortgen'in uygulaması, bu bölümde tanımlanan GeometrikSekil sınıfından türetilmeleri (genişletilmeleri) dışında önceki bölümlerdeki kod ile aynıdır.

# Abstract Classes (Soyut Sınıflar)

GeometrikSekil sınıfında getAlan() ve getCevre() metotlarını soyut (abstract) olarak tanımlayarak hangi avantajın elde edildiğini merak ediyor olabilirsiniz.

Bunları, GeometrikSekil sınıfında tanımlamanın faydaları Örnek 1'de gösterilmiştir.

Program, çember ve dikdörtgen olmak üzere iki geometrik nesne oluşturur, eşit alanlara sahip olup olmadıklarını kontrol etmek için esitAlan metodunu çağırır

ve bunları görüntülemek için gosterGeometrikSekil metodunu çağırır.

# Örnek 1:

```
C:\Program Files\Java\jdk-16.0.1\bin\yeni2>
java TestGeometrikSekil
Iki nesne esit alana mi sahip? false

Alan: 78.53981633974483
Cevre: 31.41592653589793

Alan: 15.0
Cevre: 16.0
```

```java
public class TestGeometrikSekil {
    /** Main metot */
    public static void main(String[] args) {
        // İki geometrik nesne oluştur
        GeometrikSekil geoNesne1 = new Cember(5);
        GeometrikSekil geoNesne2 = new Dikdortgen(5, 3);

        System.out.println("Iki nesne esit alana mi sahip? " + esitAlan(geoNesne1, geoNesne2));

        // Cemberi göster
        gosterGeometrikSekil(geoNesne1);

        // Dikdörtgeni göster
        gosterGeometrikSekil(geoNesne2);
    }

    /** İki geometrik nesnenin alanını karşılaştıran metot */
    public static boolean esitAlan(GeometrikSekil nesne1, GeometrikSekil nesne2) {
        return nesne1.getAlan() == nesne2.getAlan();
    }

    /** Geometrik nesneyi gösteren metot */
    public static void gosterGeometrikSekil(GeometrikSekil nesne) {
        System.out.println();
        System.out.println("Alan: " + nesne.getAlan());
        System.out.println("Cevre: " + nesne.getCevre());
    }
}
```

# Örnek 1:

GeometrikSekil sınıfında tanımlanan getAlan() ve getCevre() metotları, Cember sınıfında ve Dikdortgen sınıfında geçersiz (override) kılınır.

Örnek 1, ifadeler (5-6. Satırlar)

```
5        GeometrikSekil geoNesne1 = new Cember(5);
6        GeometrikSekil geoNesne2 = new Dikdortgen(5, 3);
```

yeni bir çember ve dikdörtgen oluşturun ve bunları geoNesne1 ve geoNesne2 değişkenlerine atayın.

Bu iki değişken GeometrikSekil tipindedir.

# Örnek 1:

esitAlan (geoNesne1, geoNesne2) (satır 8) çağrılırken, geoNesne1 bir cember olduğundan Cember sınıfında tanımlanan getAlan() metodu nesne1.getAlan() için kullanılır.

geoNesne2 bir dikdörtgen olduğundan Dikdortgen sınıfında tanımlanan getAlan() metodu nesne2.getAlan() için kullanılır.

Benzer şekilde, gosterGeometrikSekil(geoNesne1) (satır 11) çağrılırken, Cember sınıfında tanımlanan getAlan() ve getCevre() metotları kullanılır

ve gosterGeometrikSekil(geoNesne2) (satır 14) çağrılırken Dikdortgen sınıfındaki getAlan ve getCevre metotları kullanılır.

# Örnek 1:

JVM, metodu çağıran gerçek nesneye bağlı olarak bu metotlardan hangisinin çalışma zamanında çağrılacağını dinamik olarak belirler.

getAlan metodu GeometrikSekil'de tanımlanmasaydı, iki geometrik nesnenin aynı alana sahip olup olmadığını karşılaştırmak için esitAlan metodunu tanımlayamayacağımızı unutmayınız.

Artık GeometrikSekil'de soyut metotları tanımlamanın faydalarını görmüş olduk.

# 10. Ders: JAVA ile Nesne Yönelimli Programlama Interfaces (Arayüzler)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama

  - Ders 29: Java Polimorfizmi / Yöntem Aşırı Yüklemesi (Method Overloading) (izle)
  - Ders 30: Java Polimorfizmi / Yöntem Geçersiz Kılma (Method Overriding) (izle)
  - Ders 31: super Anahtar Sözcüğünün Kullanımı (izle)
  - Ders 32: final Anahtar Sözcüğünün Kullanımı (izle)
  - Ders 33: Soyut Sınıf (Abstract Class) (izle)
  - Ders 34: Arayüz (Interface) (izle)

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# Arayüz (Interface)

Arayüz (interface), yalnızca sabitler ve soyut metotlar içeren sınıf benzeri bir yapıdır.

Birçok yönden, bir arayüz soyut bir sınıfa benzer, ancak amacı ilgili sınıfların veya ilgisiz sınıfların nesneleri için ortak davranışı belirlemektir.

Örneğin, uygun arayüzleri kullanarak, nesnelerin karşılaştırılabilir olup olmadığını, yenilebilir olup olmadığını veya klonlanabilir olup olmadığını belirleyebiliriz.

# Arayüz (Interface)

Java, bir arayüzü (interface) bir sınıftan ayırmak için ve bir arayüzü tanımlamak için aşağıdaki sözdizimini kullanır:

```
erişim interface ArayuzAdi {
    /** Sabit tanımlamaları */
    /** Abstract metot imzaları */
}


public interface Yenilebilir {
    /** Nasıl yenileneceğini tanımla */
    public abstract String nasilYenir();
}
```

# Arayüz (Interface)

Java'da arayüz özel bir sınıf gibi ele alınır.

Her arayüz, normal bir sınıf gibi ayrı bir byte kodu dosyasında derlenir.

Bir arayüzü, soyut bir sınıfı kullandığınız gibi hemen hemen aynı şekilde kullanabiliriz.

Örneğin, bir referans değişkeni için veri türü olarak, çevrimin sonucu olarak bir arayüz vb. kullanabiliriz.

Soyut bir sınıfta olduğu gibi, new operatörünü kullanarak bir arayüzden örnek oluşturamayız.

# Arayüz (Interface)

Bir nesnenin yenilebilir olup olmadığını belirlemek için **Yenilebilir** arayüzünü kullanabiliriz.

Bu, nesne sınıfının **implements** anahtar sözcüğünü kullanarak bu arayüzü uygulamasına izin vererek gerçekleştirilir.

Örnek 1'deki Tavuk ve Meyve sınıfları (satır 20, 39), Yenilebilir arayüzünü uygular (implement eder).

Sınıf ve arayüz arasındaki ilişki, arayüz kalıtımı olarak bilinir. Arayüz kalıtımı ve sınıf kalıtımı temelde aynı olduğundan, her ikisine de kalıtım olarak değineceğiz.

# TestYenilebilir

```java
public class TestYenilebilir {
    public static void main(String[] args) {
        Object[] nesneler = {new Kaplan(), new Tavuk(), new Elma()};
        for (int i = 0; i < nesneler.length; i++) {
            if (nesneler[i] instanceof Yenilebilir)
                System.out.println(((Yenilebilir)nesneler[i]).nasilYenir());

            if (nesneler[i] instanceof Hayvan) {
                System.out.println(((Hayvan)nesneler[i]).ses());
            }
        }
    }
}

abstract class Hayvan {
    /** Return animal sound */
    public abstract String ses();
}

class Tavuk extends Hayvan implements Yenilebilir {
    @Override
    public String nasilYenir() {
        return "Tavuk: Kizartma yap";
    }

    @Override
    public String ses() {
        return "Tavuk: git-git-gidak";
    }
}

class Kaplan extends Hayvan {
    @Override
    public String ses() {
        return "Kaplan: RROOAARR";
    }
}
```

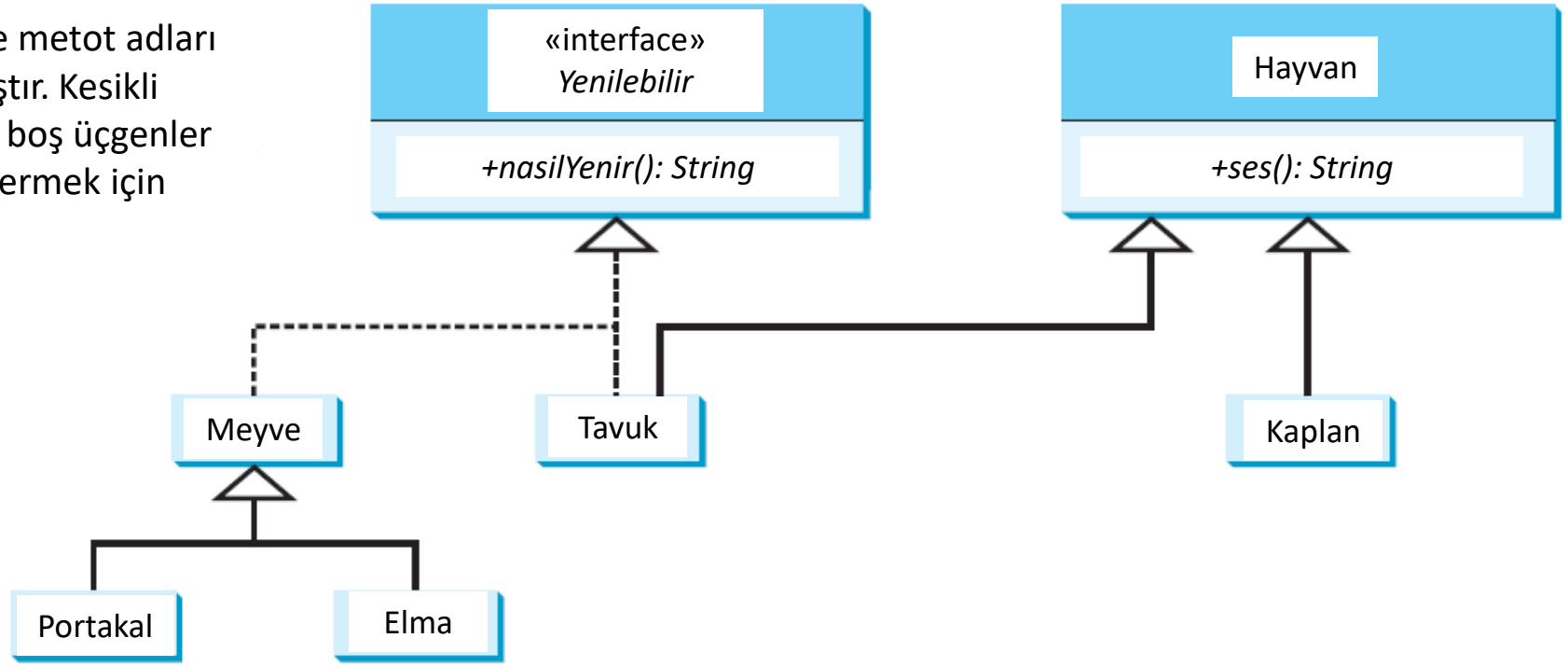
```java
public interface Yenilebilir {
    /** Nasıl yenileceğini tanımla */
    public abstract String nasilYenir();
}
```

```java
abstract class Meyve implements Yenilebilir {
    // Veri alanları, yapıcılar ve metotlar...
}

class Elma extends Meyve {
    @Override
    public String nasilYenir() {
        return "Elma: Elma suyu yapalim";
    }
}

class Portakal extends Meyve {
    @Override
    public String nasilYenir() {
        return "Portakal: Portakal suyu yapalim";
    }
}
```

```
C:\Program Files\Java\jdk-16.0.1\bin\yeni2>javac Yenilebilir.java

C:\Program Files\Java\jdk-16.0.1\bin\yeni2>javac TestYenilebilir.java

C:\Program Files\Java\jdk-16.0.1\bin\yeni2>java TestYenilebilir
Kaplan: RROOAARR
Tavuk: Kizartma yap
Tavuk: git-git-gidak
Elma: Elma suyu yapalim
```

# Yenilebilir Arayüz (Interface)

Gösterim:
Arayüz adı ve metot adları italik yazılmıştır. Kesikli çizgiler ve içi boş üçgenler arayüzü göstermek için kullanılır.




Yenilebilir, Tavuk ve Meyve için üst tiptir. Hayvan, Tavuk ve Kaplan için üst tiptir. Meyve, Portakal ve Elma için üst tiptir.

# Kalıtım

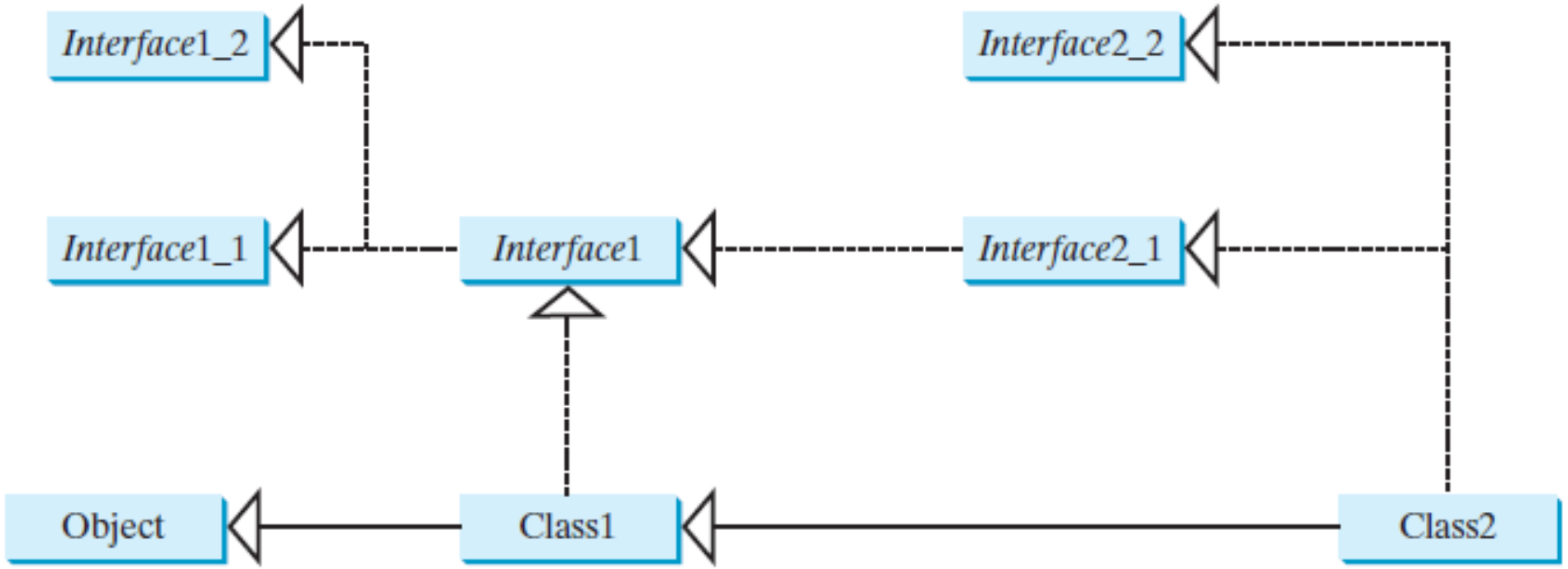



**FIGURE 13.7** `Class1` implements `Interface1`; `Interface1` extends `Interface1_1` and `Interface1_2`. `Class2` extends `Class1` and implements `Interface2_1` and `Interface2_2`.

# Comparable Interface (Karşılaştırılabilir Arayüz)

Comparable arayüzü, nesneleri karşılaştırmak için compareTo metodunu tanımlar.

İki öğrenci, iki tarih, iki çember, iki dikdörtgen veya iki kare gibi aynı türden iki nesneden daha büyük olanı bulmak için genel bir metot tasarlamak istediğimizi varsayalım.

Bunu başarmak için, iki nesnenin karşılaştırılabilir olması gerekir, bu nedenle nesneler için ortak davranış karşılaştırılabilir olmalıdır.

Java, bu amaç için Comparable arayüzü sağlar.

# Comparable Interface (Karşılaştırılabilir Arayüz)

Arayüz şu şekilde tanımlanır:

```
// Interface for comparing objects,
defined in java.lang
package java.lang;

public interface Comparable<E> {
        public int compareTo(E o);
}
```

compareTo metodu, belirtilen o nesnesiyle bu nesnenin sırasını belirler

ve bu nesne o değerinden küçük, ona eşit veya ondan büyükse

negatif bir tamsayı, sıfır veya pozitif bir tamsayı döndürür.

# Comparable Interface (Karşılaştırılabilir Arayüz)

Comparable arayüz genel bir arayüzdür.

Genel tip E, bu arayüz uygulanırken somut bir tiple değiştirilir.

Java kütüphanesindeki birçok sınıf, nesneler için doğal bir sıra tanımlamak üzere Comparable'ı uygular.

Byte, Short, Integer, Long, Float, Double, Character, BigInteger, BigDecimal, Calendar, String ve Date sınıflarının tümü Comparable arayüzünü uygular.

Örneğin, Integer, BigInteger, String ve Date sınıfları, Java API'sinde aşağıdaki gibi tanımlanır:

# Comparable Interface (Karşılaştırılabilir Arayüz)

Örneğin, Integer, BigInteger, String ve Date sınıfları, Java API'sinde aşağıdaki gibi tanımlanır:

```
public class Integer extends Number
    implements Comparable<Integer> {
  // class body omitted

  @Override
  public int compareTo(Integer o) {
    // Implementation omitted
  }
}
```

```
public class BigInteger extends Number
    implements Comparable<BigInteger> {
  // class body omitted

  @Override
  public int compareTo(BigInteger o) {
    // Implementation omitted
  }
}
```

```
public class String extends Object
    implements Comparable<String> {
  // class body omitted

  @Override
  public int compareTo(String o) {
    // Implementation omitted
  }
}
```

```
public class Date extends Object
    implements Comparable<Date> {
  // class body omitted

  @Override
  public int compareTo(Date o) {
    // Implementation omitted
  }
}
```

# SiralaKarsilastirilabilirNesne

```java
import java.math.*;

public class SiralaKarsilastirilabilirNesne {
    public static void main(String[] args) {
        String[] sehirler = {"Savannah", "Boston", "Atlanta", "Tampa"};
        java.util.Arrays.sort(sehirler);
            for (String sehir: sehirler)
                System.out.print(sehir + " ");
        System.out.println();

        BigInteger[] buyukSayilar = {new BigInteger("2323231092923992"),
                                     new BigInteger("432232323239292"),
                                     new BigInteger("54623239292")};
        java.util.Arrays.sort(buyukSayilar);
            for (BigInteger sayi: buyukSayilar)
                System.out.print(sayi + " ");
    }
}
```

```
C:\Program Files\Java\jdk-16.0.1\bin\yeni2>javac SiralaKarsilastirilabilirNesne.java

C:\Program Files\Java\jdk-16.0.1\bin\yeni2>java SiralaKarsilastirilabilirNesne
Atlanta Boston Savannah Tampa
54623239292 432232323239292 2323231092923992
```

# 11. Ders: JAVA ile Nesne Yönelimli Programlama Class Design Guidelines (Sınıf Tasarım Kılavuzları)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Doç. Dr. Yaman Akbulut

# JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama

  - Ders 29: Java Polimorfizmi / Yöntem Aşırı Yüklemesi (Method Overloading) (izle)
  - Ders 30: Java Polimorfizmi / Yöntem Geçersiz Kılma (Method Overriding) (izle)
  - Ders 31: super Anahtar Sözcüğünün Kullanımı (izle)
  - Ders 32: final Anahtar Sözcüğünün Kullanımı (izle)
  - Ders 33: Soyut Sınıf (Abstract Class) (izle)
  - Ders 34: Arayüz (Interface) (izle)

# Java Keywords

| | | | |
|---|---|---|---|
| abstract | double | int | super |
| assert | else | interface | switch |
| boolean | enum | long | synchronized |
| break | extends | native | this |
| byte | final | new | throw |
| case | finally | package | throws |
| catch | float | private | transient |
| char | for | protected | try |
| class | goto | public | void |
| const | if | return | volatile |
| continue | implements | short | while |
| default | import | static | |
| do | instanceof | strictfp | |

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

# 1. Cohesion (Uyum)

Bir sınıf tek bir varlığı tanımlamalı ve tüm sınıf işlemleri tutarlı bir amacı desteklemek için mantıksal olarak birbirine uymalıdır.

Örneğin, öğrenciler için bir sınıf kullanabilirsiniz,

ancak öğrenciler ve personel farklı varlıklar olduğundan, öğrencileri ve personeli aynı sınıfta birleştirmemelisiniz.

Birçok sorumluluğu olan tek bir varlık, sorumlulukları ayırmak için birkaç sınıfa ayrılabilir.

# 1. Cohesion (Uyum)

String, StringBuilder ve StringBuffer sınıflarının tümü, örneğin dizelerle (string) ilgilenir, ancak farklı sorumluluklara sahiptir.

String sınıfı değişmez dizelerle (string) ilgilenir,

StringBuilder sınıfı değiştirilebilir dizeler (string) oluşturmak içindir

ve StringBuffer sınıfı, StringBuffer'ın dizeleri (string) güncellemek için senkronize metotlar içermesi dışında StringBuilder'a benzer.

# 2. Consistency (Tutarlılık)

Standart Java programlama stilini ve adlandırma kurallarını izleyiniz.

Sınıflar, veri alanları ve metotlar için bilgilendirici adlar seçiniz.

Popüler bir stil, veri bildirimini yapıcıdan önce ve yapıcıları metotlardan önce yerleştirmektir.

İsimleri tutarlı hale getirin.  Benzer işlemler için farklı isimler seçmek iyi bir uygulama değildir.

Örneğin, length() metodu bir String, StringBuilder ve StringBuffer'ın boyutunu döndürür. Bu sınıflarda bu metot için farklı isimler kullanılsaydı tutarsızlık olurdu.

# 2. Consistency (Tutarlılık)

Genel olarak, varsayılan bir örnek oluşturmak için tutarlı bir şekilde genel argümanı olmayan (parametresiz) bir yapıcı sağlamalısınız.

Bir sınıf bağımsız argümanı olmayan (parametresiz) bir yapıcıyı desteklemiyorsa, nedenini belgeleyiniz.

Açık bir şekilde hiçbir yapıcı tanımlanmadıysa, boş bir gövdeye sahip genel bir varsayılan argümanı olmayan (parametresiz) yapıcı varsayılır.

Kullanıcıların bir sınıf için nesne oluşturmasını önlemek istiyorsanız, Math sınıfında olduğu gibi sınıfta özel bir yapıcı bildirebilirsiniz.

# 3. Encapsulation (Kapsülleme)

Bir sınıf, verilerini istemciler tarafından doğrudan erişimden gizlemek için private değiştiriciyi kullanmalıdır.

Bu, sınıfın bakımını kolaylaştırır.

Yalnızca veri alanının okunabilir olmasını istiyorsanız alıcı (get) metodu sağlayın ve yalnızca veri alanının güncellenebilir olmasını istiyorsanız bir ayarlayıcı (set) metodu sağlayınız.

Örneğin, Rational sınıfı, pay ve payda için bir alıcı (get) metodu sağlar, ancak bir Rational nesnesi değişmez olduğu için ayarlayıcı (set) bir metot sağlamaz.

# 4. Clarity (Açıklık, Netlik)

Uyum, tutarlılık ve kapsülleme, tasarım netliği elde etmek için iyi kılavuzlardır.

Ek olarak, bir sınıfın açıklanması ve anlaşılması kolay açık bir sözleşmesi olmalıdır.

Kullanıcılar sınıfları birçok farklı kombinasyonda, düzende ve ortamda birleştirebilir.

Bu nedenle, kullanıcının onu nasıl ve ne zaman kullanabileceği konusunda hiçbir kısıtlama getirmeyen bir sınıf tasarlamalısınız.

# 4. Clarity (Açıklık, Netlik)

Özellikleri, kullanıcının herhangi bir sırayla ve oluşum sıralarından bağımsız olarak işlev gören herhangi bir değer ve tasarım yöntemi kombinasyonu ile ayarlamasına izin verecek şekilde tasarlayınız.

Metotlar, kafa karışıklığına neden olmadan sezgisel olarak tanımlanmalıdır.

Örneğin, String sınıfındaki substring (int beginIndex, int endIndex) yöntemi biraz kafa karıştırıcıdır.

Metot, endIndex yerine beginIndex'ten endIndex – 1'e bir alt dize döndürür.

beginIndex'ten endIndex'e bir alt dizeyi döndürmek daha sezgisel olacaktır.

# 4. Clarity (Açıklık, Netlik)

Diğer veri alanlarından türetilebilecek bir veri alanı beyan etmemelisiniz.

Örneğin, aşağıdaki Kisi sınıfının iki veri alanı vardır: dogumTarihi ve yas.

yas, dogumTarihinden türetilebildiğinden, yas veri alanı olarak beyan edilmemelidir.

```
public class Kisi {
    private java.util.Date dogumTarihi;
    private int yas;
    ...
}
```

# 5. Completeness (Bütünlük, Tamlık)

Sınıflar, birçok farklı müşteri (istemci, kullanıcı) tarafından kullanılmak üzere tasarlanmıştır.

Çok çeşitli uygulamalarda yararlı olabilmesi için, bir sınıf, özellikler ve metotlar aracılığıyla özelleştirme için çeşitli yollar sağlamalıdır.

Örneğin, String sınıfı, çeşitli uygulamalar için yararlı olan 40'tan fazla metot içerir.

# 6. Instance vs. Static (Örnek vs. Statik)

Sınıfın belirli bir örneğine (instance, nesne) bağımlı olan bir değişken veya metot, bir örnek (instance, nesne) değişkeni veya metot olmalıdır.

Bir sınıfın tüm örnekleri tarafından paylaşılan bir değişken static olarak bildirilmelidir.

Sayici sınıfı içindeki sayac değişkeni,
Sayici sınıfının tüm nesneleri tarafından paylaşılır ve bu nedenle static olarak bildirilir.

```
1  public class Sayici {
2      static int sayac;
3
4      Sayici(){
5          sayac++;
6          System.out.println(sayac);
7      }
8  }
```

# 6. Instance vs. Static (Örnek vs. Statik)

Belirli bir örneğe (instance, nesne) bağlı olmayan bir metot, static bir metot olarak tanımlanmalıdır.

Hesapla'daki kupAl() ve kareAl() metotları belirli bir örneğe bağlı değildir.

ve bu nedenle static bir metotlar olarak tanımlanır.

```
1  public class Hesapla{
2      static int kupAl(int a){
3          return a*a*a;
4      }
5      static int kareAl(int a){
6          return a*a;
7      }
8  }
```

Okunabilirliği artırmak ve hataları önlemek için her zaman bir sınıf adından (bir referans değişkeni yerine) statik değişkenlere ve metotlara başvurunuz.

# 6. Instance vs. Static (Örnek vs. Statik)

Statik bir veri alanını başlatmak için yapıcıdan parametre iletmeyiniz.

Statik veri alanını değiştirmek için bir ayarlayıcı (set) metodu kullanmak daha iyidir.
Bu nedenle, aşağıda (a)'daki belirtilen sınıfın (b) ile değiştirilmesi daha iyidir.

```
public class SomeThing {
  private int t1;
  private static int t2;

  public SomeThing(int t1, int t2) {
    ...
  }
}
```

(a)

```
public class SomeThing {
  private int t1;
  private static int t2;

  public SomeThing(int t1) {
    ...
  }

  public static void setT2(int t2) {
    SomeThing.t2 = t2;
  }
}
```

(b)

# 6. Instance vs. Static (Örnek vs. Statik)

Örnek ve statik, nesne yönelimli programlamanın ayrılmaz parçalarıdır.

Bir veri alanı veya metot ya örnektir veya statiktir.

Statik veri alanlarını veya metotlarını yanlışlıkla gözden kaçırmayınız.

Statik olması gereken metodu bir örnek metodu olarak tanımlamak yaygın bir tasarım hatasıdır.

Örneğin, n'nin faktöriyelini hesaplamak için faktoriyel(int n) metodu, herhangi bir özel durumdan bağımsız olduğu için statik olarak tanımlanmalıdır.

# 6. Instance vs. Static (Örnek vs. Statik)

Bir yapıcı her zaman örnektir (instance),

çünkü belirli bir örnek (instance) oluşturmak için kullanılır.

Statik bir değişken veya metot, bir örnek metottan çağrılabilir,

ancak bir örnek (instance) değişkeni veya metot, statik bir metottan

çağrılamaz.

# 7. Inheritance vs. Aggregation
## (Kalıtım vs. Toplama)

Kalıtım ve toplama arasındaki fark, is-a ve has-a ilişkisi arasındaki farktır.

Örneğin, elma bir meyvedir; bu nedenle, Elma ve Meyve sınıfları arasındaki ilişkiyi modellemek için kalıtımı kullanırız.

Bir kişinin bir adı vardır; bu nedenle, Kisi ve Ad sınıfları arasındaki ilişkiyi modellemek için toplamayı kullanırız.

# 8. Interfaces vs. Abstract Classes
## (Arayüzler vs. Soyut Sınıflar)

Hem arayüzler hem de soyut sınıflar, nesneler için ortak davranışı belirtmek için kullanılabilir.

Arayüz mü yoksa sınıf mı kullanacağımıza nasıl karar veriyoruz?

Genel olarak, ebeveyn-çocuk (parent-child) ilişkisini açıkça tanımlayan güçlü bir (is-a) ilişki, sınıflar kullanılarak modellenmelidir.

Örneğin, portakal bir meyve olduğu için, ilişkileri sınıf kalıtımı kullanılarak modellenmelidir.

# 8. Interfaces vs. Abstract Classes
## (Arayüzler vs. Soyut Sınıflar)

Zayıf bir (is-a) ilişki, aynı zamanda bir tür (is-kind-of) ilişki olarak da bilinir, bir nesnenin belirli bir özelliğe sahip olduğunu gösterir.

Zayıf bir is-a ilişkisi, arayüzler kullanılarak modellenebilir.

Örneğin, tüm dizeler (string) karşılaştırılabilir, bu nedenle String sınıfı Comparable arayüzünü uygular (implements).

Cember veya dikdörtgen geometrik bir nesnedir, bu nedenle Cember, GeometrikSekil'in bir alt sınıfı olarak tasarlanabilir.

Cemberler, farklıdır ve yarıçaplarına göre karşılaştırılabilirdir, bu nedenle Cember, Comparable arayüzü uygulayabilir (implements).

# 8. Interfaces vs. Abstract Classes
## (Arayüzler vs. Soyut Sınıflar)

Arayüzler, soyut sınıflardan daha esnektir, çünkü bir alt sınıf yalnızca bir üst sınıfı genişletebilir, ancak herhangi bir sayıda arayüz uygulayabilir.

Ancak arayüzler somut yöntemler içeremez.

Arayüzlerin ve soyut sınıfların erdemleri, onu uygulayan soyut bir sınıfla bir arayüz oluşturularak birleştirilebilir.

Ardından, hangisi uygunsa, arayüzü veya soyut sınıfı kullanabilirsiniz.