



Bölüm 4: Nesne ve Sınıflar

JAVA ile Nesne Yönelimli Programlama



Nesne Yönelimli Programlama

- Nesneleri kullanarak programlama anlamına gelir.
- Nesne, gerçek dünyada net bir şekilde tanımlanabilen varlığı temsil eder.
- Örnekler:
 - Öğrenci
 - Masa
 - Çember
 - Düğme
 - Hatta bir Kredi



Nesneler

- Java'da Nesne Yönelimli Programlama, gerçek dünya varlıklarını modellemede güçlü bir çerçeve sunar.
- Veri (alanlar) ve davranış (metodlar) içeren nesneler, daha düzenli, verimli ve sezgisel kod oluşturmamıza olanak tanır.
- Nesneler, **veri** saklayabilir.
 - Bu veri parçalarına «alanlar» (fields) denir.
- Nesneler, **işlemler** gerçekleştirebilir.
 - Bu işlemlere «metodlar» (methods) denir.



Örnek Sınıflar

- Öğrenci Nesnesi
 - Alanlar: İsim, Yaş, Numara
 - Metodlar: Derslere Kaydol, Sınavlara Gir
- Masa Nesnesi
 - Alanlar: Malzeme, Boyut, Renk
 - Metodlar: Eşyaları Sakla, Çalışma Alanı Sağla
- Çember Nesnesi
 - Alanlar: Yarıçap, Merkez Koordinatlar
 - Metodlar: Alan Hesapla, Çevre Hesapla
- Kredi Nesnesi
 - Alanlar: Ana Para, Faiz Oranı, Vade
 - Metodlar: Aylık Taksitleri Hesapla, Onayla/Reddet



Avantajlar

- **Kapsülleme:** Nesneler, veriyi ve davranışı kapsüller, bu modülerlik ve veri gizleme sağlar.
- **Kalıtım:** Nesneler, diğer nesnelerden özellikleri devralabilir, bu kod yeniden kullanımı ve hiyerarşi sağlar.
- **Çok Biçimlilik:** Nesneler, birden fazla biçim alabilir, bu esneklik ve dinamik davranış sağlar.



Sınıf Nedir?

- Bir sınıf, nesnelerin **taslağıdır**. Bir sınıftan **birden çok** nesne oluşturabilir ve bu nesneler **farklı verileri** temsil edebilir.
- Sınıflar, nesneleri **tanımlamak** ve **oluşturmak** için temel bir yapı taşıdır.
- Bir sınıf, nesnenin taşıyabileceği **verileri** (nesnenin alanları) ve yapabileceği **işlemleri** (nesnenin metotları) belirtir.
- Bir nesne, belirli bir sınıfın bir **örneğidir**.
- Sınıflar, gerçek dünyadaki nesneleri **modellememizi** sağlar.
- **Örnek:** Bir "Araba" sınıfı, her türlü otomobili temsil edebilir ve her otomobil bu sınıftan türetilir.



Örnek: Araba Sınıfı

- Alanlar (Veriler):
 - Marka
 - Model
 - Yıl
 - Renk
- Metotlar (İşlemler):
 - Hareket Et
 - Dur
 - Sinyal Ver



Örnek: Sınıfın Kullanımı

- Sınıfımızı kullanarak birkaç araba nesnesi oluşturalım:
- Araba1:
 - Marka=Toyota, Model=Corolla, Yıl=2022, Renk=Gri
- Araba2:
 - Marka=Ford, Model=Mustang, Yıl=2023, Renk=Mavi



Sınıfın Özel Metotları: Yapıcılar (Constructors)

- Nesneleri oluşturmak için kullanılan özel bir metot türüdür.
- Yapıcılar, nesnelerin oluşturulmasında önemli bir rol oynar.
- Yapıcılar, nesneleri sınıftan oluştururken ilk değerlerini vermek için kullanılır.



Nesne Nasıl Oluşturulur?

- Bir nesne, genellikle **new** operatörü kullanılarak bir sınıftan oluşturulur.
- **Örnek:**
 - `SınıfAdı nesneAdı = new SınıfAdı(parametreler);`
 - `Araba corolla = new Araba("Toyota", "Corolla", 2022);`
 - `Scanner klavye = new Scanner(System.in);`
 - `Kitap kitap = new Kitap();`



new Operatörü ve Yapıcı Metodu

- new operatörü, bir sınıfın yapıcı metodunu çağırarak nesnelerin oluşturulmasını sağlar.
- Yapıcı metodun adı sınıf adıyla aynıdır ve geri dönüş türü içermez.
- Örnek nesne, yapıcı metot tarafından başlatılır ve inşa edilir.
- Java'da bazı önceden tanımlanmış sınıfların nesnelerini oluşturmak için kısayollar vardır.
- Örnek:
 - `String baslik; // String sınıfından bir nesne oluşturur.`
 - `String yazar;`



Sınıf Tanımlama

- Bir sınıf, nesnelerin (objelerin) taslağını ve yapısal özelliklerini tanımlayan bir yapıdır.

```
public class Araba {  
    // Alanlar (Fields)  
    String marka; String model;  
    int yıl; String renk;  
    // Metotlar (Methods)  
    void ilerle() {    // İlerleme işlemleri  
    }  
    void dur() {      // Durdurma işlemleri  
    }  
    void sinyalVer() { // Sinyal verme işlemleri  
    }  
}
```



Sınıf Erişilebilirlik Değiştiricileri

- Erişilebilirlik değiştiricileri, bir sınıfın diğer sınıflar tarafından nasıl erişilebileceğini belirler.
- Java'da dört temel erişilebilirlik değiştiricisi vardır:
 - **public** (Herkese Açık): Her yerden erişilebilir.
 - **private** (Özel): Sadece sınıfın içinden erişilebilir.
 - **protected** (Korumalı): Aynı paketten veya alt sınıflardan erişilebilir.
 - **default** (Belirtilmeyen): Yalnızca aynı paketten (package) erişilebilir.



Sınıf Üye Tanımlamaları

- Sınıf içinde tanımlanan veri alanları (**fields**) ve işlevsel metotlar (**methods**) gibi öğelerdir.
- Sınıf üyeleri, nesnelerin (**objelerin**) davranışını ve verilerini tanımlar.
- Veri alanları, nesnelerin taşıyabileceği verileri temsil eder.
 - **Örnek:** Bir "Öğrenci" sınıfında "Adı" ve "Soyadı" veri alanları.
- Metotlar, nesnelerin gerçekleştirebileceği işlemleri tanımlar.
 - **Örnek:** Bir "Köpek" sınıfında "Havla" ve "Koş" metotları.
- Sınıf üyeleri özel (**private**), korumalı (**protected**), varsayılan (**default**) ve genel (**public**) olabilir.
- Sınıf üyeleri ayrıca statik (**static**) veya örneksel (**instance**) olabilir.



Alanlar (Nitelikler) ve Türleri

- Alanlar (Nitelikler), sınıfların verilerini ve özelliklerini temsil eder.
- Alanlar, sınıf içinde tanımlanan veri değişkenleridir ve iki temel türe sahiptir:
 - **Örnek alanlar**, her nesne için ayrı bir kopyaya sahiptir,
 - **Sınıf alanları** ise sınıfın tüm örnekleri tarafından paylaşılır.
- Sabit (final) alanlar, değerleri değiştirilemeyen özel alanlardır.



Alanlar (Nitelikler) ve Türleri

- Örnek Alanlar (Instance Fields)
 - Bir sınıfın örneği (nesnesi) içinde bulunan ve her örnek için ayrı bir kopyası olan verileri temsil eder.
 - **Örnek:** Bir "Öğrenci" sınıfının örnek alanları "Adı" ve "Soyadı" olabilir.
- Sınıf Alanları (Static Fields)
 - Sınıfın kendisine ait olan ve sınıfın her örneği tarafından paylaşılan verileri temsil eder.
 - **Örnek:** Bir "BankaHesabı" sınıfının sınıf alanı, bankadaki toplam hesap sayısını saklayabilir.



Alanlar (Nitelikler) ve Türleri

```
public class Kitap {  
    // Örnek Alanlar  
    private String başlık;  
    private String yazar;  
    private int sayfaSayısı;  
  
    // Sınıf Alanı  
    private static int toplamKitapSayısı;  
  
    // final Modifikatörlü Bir Alan (Sabit)  
    public static final double PI = 3.14159265359;  
}
```



Nitelik Tanımlamaları

```
public int a; // Genel tamsayı alanı (public access modifier).  
int b = 1, c = 2; // Tamsayı alanları, varsayılan erişim düzeyi  
private double x; // Özel ondalık sayı alanı (private Access).  
private static int xx; // Statik, özel bir tamsayı alanı (private  
ve static access modifier).  
public static int y; // Statik, genel bir tamsayı alanı (public  
ve static access modifier).  
public final int CONST1 = 5; // Genel ve değiştirilemeyen (final)  
private static final int CONST2 = 6; // Statik, özel ve  
değiştirilemeyen bir tamsayı sabiti.
```



Metot Türleri ve Kullanımları

- Metotlar, sınıf içinde tanımlanan işlevsel bloklardır ve programın temel yapı taşlarından biridir.
- Metotlar, bir sınıfın davranışını tanımlar ve programın işlevselliğini sağlar.
- Metotlar iki temel türe sahiptir:
 - **Örnek metotlar** bir nesne ile ilişkilidir,
 - **Sınıf metotları** ise sınıfın kendisine aittir.
- Sınıf metotları, tüm sınıf örnekleri tarafından paylaşılır ve nesnelere özgü verilere erişemezler.



Metot Türleri ve Kullanımları

- Örnek Metotlar
 - Örnek metotlar, bir nesneye aittir ve o nesneyle çalışırlar.
 - **Örnek:** Bir "Öğrenci" sınıfındaki "Öğrenci Bilgisini Göster" metodu.
- Sınıf Metotları
 - Sınıf metotları, sınıfın kendisine aittir ve tüm sınıf örnekleri tarafından paylaşılır.
 - **Örnek:** Bir "Matematik İşlemleri" sınıfındaki "Toplama" metodu.



Metot Türleri ve Kullanımları

```
public class CepTelefonu {  
    // Örnek Metot  
    public void aramaYap(String numara) {  
        // Arama işlemleri  
    }  
    // Örnek Metot  
    public void mesajGönder(String alıcı, String mesaj) {  
        // Mesaj gönderme işlemleri  
    }  
    // Sınıf Metodu (static)  
    public static void modeliGüncelle(String yeniModel) {  
        // Model güncelleme işlemleri  
    }  
}
```



Metot Tanımlama

- Metotlar, sınıf içinde belirli bir görevi gerçekleştiren kod bloklarını temsil eder.
- Kodun tekrar kullanılabilirliğini ve düzenini sağlar.
- Bir metot tanımlarken kullanılan temel öğeler:

```
[erişim düzeyi] [geri dönüş türü] [metot adı]([parametreler])  
{  
    // Metotun içeriği  
}
```



Metot Tanımlama

- **Erişim Düzeyi** (Access Modifier)
 - Metotun başka sınıflar tarafından nasıl erişilebileceğini belirler.
 - **Örnek:** public, private, protected, default
- **Geri Dönüş Türü** (Return Type)
 - Metotun geriye döndüreceği değer türünü belirler.
 - **Örnek:** int, String, void, kullanıcı tanımlı sınıf türleri
- **Metot Adı** (Method Name)
 - Metotun çağrılabilmesi için kullanılan ismi belirler.
 - **Örnek:** hesapla, verileriGöster, kameraAç
- **Parametreler** (Parameters)
 - Metota giriş parametreleri olarak aktarılan değerleri tanımlar.
 - **Örnek:** int sayi1, int sayi2, String ad, double en



Metot Tanımlama Örneği

- Bir "E-posta" sınıfı:
 - **Metot Adı:** gonder
 - **Parametreler:** String alici, String konu, String icerik
 - **Geri Dönüş Türü:** boolean (gönderim başarılı mı değil mi)

```
public class Eposta {  
    public boolean gonder(String alici, String konu, String icerik){  
        // E-posta gönderme işlemleri  
        // Başarılı ise true, aksi takdirde false döndürülür.  
    }  
}
```




Örnek (instance) Alanlar ve Metotlar

- "static" anahtar kelimesi olmadan bir sınıf içinde tanımlanırlar.
- Her nesne, örnek alanlar ve örnek metotların kendi kopyalarına sahiptir.
- Örnek alanlar, bir nesnenin özelliklerini temsil eder ve nesneler arasında farklı verilere sahip olabilirler.
- Örnek metotlar, bir nesnenin davranışını tanımlar ve o nesneyle ilişkilidir.
- Örnek alanlar ve örnek metotlar kullanabilmek için bir nesne oluşturmak gereklidir.



Constructor (Yapıcı Metotlar)

- Yapıcı metotlar, sınıfların özel metotlarıdır ve bir nesne oluşturulduğunda otomatik olarak çağrılırlar.
- Yapıcı metotlar, nesne oluşturulurken belirli işlemleri gerçekleştirmek için kullanılır.
- Yapıcı metotlar genellikle örnek alanları başlatır ve diğer nesne başlatma görevlerini gerçekleştirir.
- Metot, sınıfın adını taşır.
- Metotun bir geri dönüş türü yoktur (void bile yoktur).
- Metot herhangi bir değer döndüremez.
- Genellikle yapıcı metotlar geneldir (public).



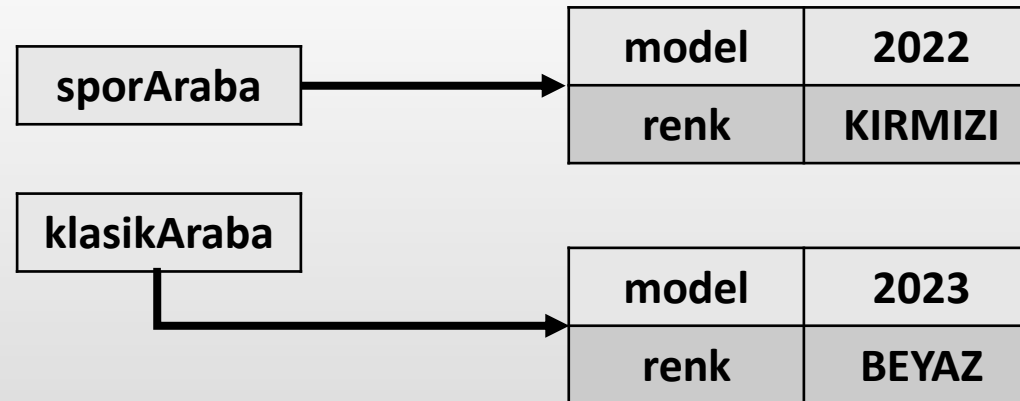
Constructor (Yapıcı Metotlar)

```
public class Araba {  
  
    private String model;  
    private String renk;  
  
    // Yapıcı Metot  
    public Araba(String model, String renk) {  
        this.model = model;  
        this.renk = renk;  
    }  
    // Diğer metotlar...  
}
```



Constructor (Yapıcı Metotlar)

```
public static void main(String[] args) {  
    Araba sporAraba = new Araba("2022", "KIRMIZI");  
    Araba klasikAraba = new Araba("2023", "BEYAZ");  
}
```





Nokta (.) Operatörü

- Bir sınıfın nesnesi oluşturulduktan sonra, bu nesneye ait örnek metotlar "nokta operatörü" kullanılarak çağrılabilir.
- `nesne.metodAdı(gerçek-parametreler);`

```
Telefon telefon1 = new Telefon("5551234567");
```

```
Telefon telefon2 = new Telefon("5051234567");
```

```
telefon1.aramaYap(telefon2);
```

```
telefon2.mesajGonder(telefon1, "Merhaba!");
```

```
telefon1.kameraAc();
```



Dikdörtgen Sınıfı

```
public class Dikdortgen {  
    private double uzunluk;        // Dikdörtgenin uzunluk özelliği  
    private double genislik;       // Dikdörtgenin genişlik özelliği  
    public Dikdortgen(double uzunluk, double genislik) {  
        this.uzunluk = uzunluk;    // Yapıcı metot  
        this.genislik = genislik;  
    }  
    public double alanHesapla() {  
        return uzunluk * genislik; // Alanı hesaplayan metot  
    }  
    public double cevreHesapla() {  
        return 2 * (uzunluk + genislik); // Çevreyi hesaplayan metot  
    }  
}
```



Dikdörtgen Sınıfı

```
public static void main(String[] args) {  
  
    Dikdortgen dikdortgen = new Dikdortgen(5.0, 3.0); // Nesne oluşturuluyor  
    Dikdortgen kare = new Dikdortgen(5.0, 5.0); // Nesne oluşturuluyor  
  
    double alan = dikdortgen.alanHesapla(); // Dikdörtgenin alanı hesaplanıyor  
    double cevre = dikdortgen.cevreHesapla(); // Çevresi hesaplanıyor  
    System.out.println("Dikdörtgen Alanı: " + alan + " Çevresi: " + cevre);  
  
    alan = kare.alanHesapla(); // Dikdörtgenin alanı hesaplanıyor  
    cevre = kare.cevreHesapla(); // Dikdörtgenin çevresi hesaplanıyor  
    System.out.println("Dikdörtgen Alanı: " + alan + " Çevresi: " + cevre);  
}
```

Erişimciler (Accessors) ve Değiştiriciler (Mutators)



- Veri Gizleme Kavramı
 - Bir sınıf içindeki alanlar (fields), genellikle dışarıdan direkt erişilememesi ve değiştirilememesi için «private» yapılır.
- Erişimciler (Accessors) Nedir?
 - Sınıfın dışından alanlara erişmek için kullanılan metotlardır.
 - Alanların değerlerini okumak için kullanılırlar.
- Değiştiriciler (Mutators) Nedir?
 - Sınıfın dışından alanlara değer atamak için kullanılan metotlardır.
 - Alanların değerlerini değiştirmek için kullanılırlar.

Erişimciler (Accessors) ve Değiştiriciler (Mutators)



```
public class BankaHesabi {  
    private double hesapBakiyesi;  
    private String hesapSahibi;  
    public double getHesapBakiyesi() { // Hesap Bakiyesi Erişimcisi  
        return hesapBakiyesi;  
    }  
    public String getHesapSahibi() { // Hesap Sahibi Erişimcisi  
        return hesapSahibi;  
    }  
    public void setHesapBakiyesi(double yeniBakiye) { // Bakiye Değiştirici  
        hesapBakiyesi = yeniBakiye;  
    }  
}
```



Veri Gizleme

- Sınıflar genellikle büyük yazılım sistemlerinde birer bileşen olarak kullanılır ve birçok yazılımcı tarafından geliştirilir.
- Veri gizleme, bir nesnenin iç verilerini korumaya ve güvenli bir şekilde erişmeye yardımcı olur.
- Sınıfların dış dünyayla etkileşimini düzenler ve bütünlüğünü korur.
- Bir nesnenin iç verilerine sadece kendi sınıfının metotları doğrudan erişebilir ve değişiklik yapabilir.
- Sınıfın dışındaki kodlar, nesnenin iç verilerine erişmek veya değiştirmek için sınıfın genel metotlarını kullanmalıdır.



Varsayılan Yapıcı Metot

- Bir nesne oluşturulduğunda, ilgili sınıfın yapıcı metodu her zaman çağrılır.
- Varsayılan yapıcı metot, bir sınıf için yapıcı metot yazılmadığında kullanılır.
- Eğer bir yapıcı metot yazmazsanız, Java sınıfı derlendiğinde bir tane sağlar.
- Varsayılan yapıcı metot, nesnenin sayısal alanlarını 0 olarak, boolean alanlarını false olarak ve referans değişkenlerini "null" olarak ayarlar.
- Varsayılan yapıcı metot, parametre almayan bir yapıcı metottur.
- Eğer bir sınıf için zaten bir yapıcı metot yazdıysanız, Java varsayılan bir yapıcı metot sağlamaz.



Varsayılan Yapıcı Metot

```
public class Arac {  
    private String marka;  
    private String model;  
    private int yıl;  
  
    // Varsayılan yapıcı metot  
    public Arac() {  
        marka = "Bilinmiyor";  
        model = "Bilinmiyor";  
        yıl = 0;  
    }  
}
```



Statik Değişkenler ve Metotlar

- "static" anahtar kelimesi kullanılarak tanımlanırlar.
- Statik üyelere erişmek için, sınıfın bir nesnesini oluşturmaya gerek yok.
- Statik alan ve metotlara erişmek için nokta operatörü kullanılır.

`SınıfAdı.AlanAdı`

`SınıfAdı.MetotAdı (GerçekParametreler)`

- Statik üyelere erişmek için nesneleri kullanabiliriz, ancak bu genellikle tercih edilen bir yöntem değildir.

`Nesne.Ad`

`Nesne.MetotAdı (GerçekParametreler)`



Statik Değişkenler ve Metotlar

```
public class Matematik {  
    // Statik toplama metodu  
    public static int toplama(int sayi1, int sayi2) {  
        return sayi1 + sayi2;  
    }  
  
    // Ana metod (main method) içinde kullanımı  
    public static void main(String[] args) {  
        int sonuc1 = Matematik.toplama(5, 3); // Statik toplama metodu  
        System.out.println("Toplama Sonucu: " + sonuc1);  
    }  
}
```



Referans Atama

- Java'da, nesneler genellikle referanslar (adresler) aracılığıyla işlenir.
- Referans atama, bir nesneye başka bir nesnenin referansını atama işlemidir.
- Referans atama işlemleri, nesneler arasında ilişki kurar ve bir nesnenin başka bir nesneyi işaret etmesini sağlar.
- Gerçek dünyada referans atamanın pek çok örneği vardır:
 - Kütüphanede bir kitabın raflardaki konumunu gösteren etiket, kitabın referansını içerir.
 - Bir aracın anahtarı, o aracı işaret eder ve kullanmamıza olanak tanır.



Referans Atama

```
public static void main(String[] args) {  
    Araba araba1 = new Araba("Toyota"); // araba1 oluşturuluyor  
    Araba araba2 = araba1; // araba2, araba1'i işaret ediyor  
  
    System.out.println("Araba 1 Markası:" + araba1.getMarka());  
    System.out.println("Araba 2 Markası:" + araba2.getMarka());  
}
```




"this" Anahtar Kelimesi

- "this" anahtar kelimesi, bir metotun çağrıldığı nesneyi işaret eder ve nesne içindeki işlemleri gerçekleştirmek için kullanılır.
- "this" anahtar kelimesi, statik metotlar içinde kullanılamaz, çünkü statik metotlar bir nesneye bağlı değildir.
- "this" anahtar kelimesinin iki yaygın kullanımı vardır:
 - Alıcı Nesneyi Parametre Olarak Aktarmak: Metot, alıcı nesneyi başka bir metoda parametre olarak iletebilir.
 - Yerel Değişkenler Tarafından Gölgeleyen Alanlara Erişmek: Metot içinde yerel bir değişken ile aynı ada sahip bir alanı (field) erişmek için "this" kullanılabilir.



"this" Anahtar Kelimesi

```
public class Ogrenci {  
    private String ad;  
    public void selamVer() {  
        System.out.println("Merhaba, ben " + this.ad);  
    }  
    public static void main(String[] args) {  
        Ogrenci ogrenci1 = new Ogrenci("Ali");  
        Ogrenci ogrenci2 = new Ogrenci("Ayşe");  
        ogrenci1.selamVer();  
        ogrenci2.selamVer();  
    }  
}
```



Metot Aşırı Yükleme (Method Overloading)

- Farklı veri türleri veya sayılarıyla, aynı işlevi yürüten metotlar tanımlanabilir.

```
// İki tamsayıyı topla
```

```
public int topla(int sayi1, int sayi2) {  
    return sayi1 + sayi2;  
}
```

```
// İki ondalıklı sayıyı topla
```

```
public double topla(double sayi1, double sayi2) {  
    return sayi1 + sayi2;  
}
```

```
// Üç tamsayıyı topla
```

```
public int topla(int sayi1, int sayi2, int sayi3) {  
    return sayi1 + sayi2 + sayi3;  
}
```



SON