



Bölüm 4: Nesne ve Sınıflar

JAVA ile Nesne Yönelimli Programlama



Nesne Yönelimli Programlama

- Nesnelerin birbiriyle etkileşimde bulunduğu bir programlama yaklaşımıdır.
- Nesne, gerçek dünyada net bir şekilde tanımlanabilen **varlığı** temsil eder.
- **Örnekler:**
 - Öğrenci
 - Masa
 - Çember
 - Düğme
 - Hatta bir Kredi



Nesneler

- *A class has members.*
- *Members can be fields, methods, or constructors.*
- Nesneler, nitelik (alan) ve davranış (metot)'a sahiptir.
- Daha düzenli, verimli ve sezgisel kod oluşturulmasını sağlar.
- Nesneler, **veri** saklayabilir.
 - Bu veri parçalarına «nitelik» (fields) denir.
- Nesneler, **işlemler** gerçekleştirebilir.
 - Bu işlemlere «davranış» (methods) denir.



Örnek Sınıflar

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



Avantajlar

- **Sarmalama:** Nesneler, nitelik ve davranışlarını sarmalar. Modülerlik ve bilgi gizleme sağlar. Dışarıya karşı kapalı bir kutu gibi gösterir.
- **Kalıtım:** Nesneler, diğer nesnelerden özelliklerini miras alabilir. Kodların yeniden kullanımını ve sınıf hiyerarşisi sağlar.
- **Çok Biçimlilik:** Nesneler, birden fazla biçim alabilir. Esneklik ve dinamik davranış sağlar.



Sınıf

- Bir sınıf, bir nesnenin **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 nitelikleri) ve yapabileceği **işlemleri** (nesnenin metotları) belirtir.
- *An object is an instance of a class.* Bir nesne, bir sınıfın **örneğidir**.
- Nesne sınıfın varlığıdır, somutlanmış halidir.
- Sınıflar, gerçek dünyadaki nesneleri **modellememizi** sağlar.
- **Örnek:** Bir "Araba" sınıfı, her türlü otomobili temsil edebilir ve her marka model otomobil bu sınıftan türetilir.



Örnek: Araba Sınıfı

- **Nitelikler (Veriler):**

- Marka
- Model
- Yıl
- Renk

- **Metotlar (İşlemler):**

- Hareket Et
- Dur
- Sinyal Ver



Örnek: Sınıfın Kullanımı

- Araba sınıfını kullanarak oluşturulan birkaç nesne:
- **Araba1:**
 - Marka=Toyota, Model=Corolla, Yıl=2022, Renk=Gri
- **Araba2:**
 - Marka=Ford, Model=Mustang, Yıl=2023, Renk=Mavi



Yapıcı Metotlar (Constructors)

- Nesneleri oluşturmak için kullanılan özel bir metottur.
- Nesne oluşturulurken, niteliklere ilk değerlerini atar.



Nesne Nasıl Oluşturulur?

- Bir nesne, **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 çağırır.
- Yapıcı metodun adı sınıf adıyla aynıdır ve geri dönüş türü içermez.
- Örnek bir nesne, yapıcı metot tarafından ilk değerleri verilerek oluşturulur.
- Bazı sınıfların nesnelerini oluşturmak için kısayollar vardır.
- Örnek:
 - `String baslik; // String sınıfından bir nesne oluşturur.`
 - `String yazar;`



Sınıf Tanımlama

- Sınıf, nesnelerin taslağını ve yapısal özelliklerini tanımlar.

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



Erişim Düzenleyiciler

- Bir sınıf ve öğelerine, sınıf dışından nasıl erişilebileceğini belirtir.
- Java'da dört temel erişim düzenleyici vardır:
 - **public** (Herkes 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.



Sınıfın Üyeleri

- Sınıf içinde tanımlanan nitelik (**fields**) ve davranış (**methods**) öğeleridir.
- Nitelikler, 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 nesneye ait (**instance**) olabilir.



Nitelikler ve Türleri

- Sınıfın verilerini ve özelliklerini temsil eder.
- Sınıf içinde tanımlanan veri değişkenleridir ve iki temel türe sahiptir:
 - **Örneğe ait alanlar**, her nesne ayrı bir kopyaya sahiptir,
 - **Sınıf alanları**, tüm nesneler tarafından paylaşılan niteliklerdir.
- Sabit (final) alanlar, değerleri değiştirilemeyen özel alanlardır.
- *Static initializers run only once, when the class is loaded.*
- Statik ilklendiriciler, sınıf yüklenirken bir kereye mahsus çalıştırılır.
- *Instance initializers, are run every time an instance is created.*
- Örneğe ait ilklendiriciler, her bir örnek yaratıldığında çalıştırılır.



Nitelikler ve Türleri

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



Örnek Kopek Sınıfı

```
public class Kopek {  
    String adi;           // nitelik  
    Kopek(String adi) {   // yapıcı  
        this.adi = adi;  
        System.out.println(adi);  
    }  
    String havla() {      // metot  
        return adi + " haw haw haw!";  
    }  
    static {              // sınıfa ait ilklendirici  
        System.out.println("Köpekler!");  
    }  
    {                     // örneğe ait ilklendirici  
        System.out.println("Köpek oluşturuluyor.");  
    }  
}
```



Örnek Kopek Sınıfı

```
public static void main(String...strings) {  
    var d1 = new Kopek("Kara");  
    var d2 = new Kopek("Gece");  
    d2.havla();  
}
```

Köpekler!
Köpek oluşturuluyor.
Kara
Köpek oluşturuluyor.
Gece



Nitelikler ve Türleri

```
public class Kitap {  
    // Nitelikler  
    private String baslik;  
    private String yazar;  
    private int sayfaSayisi;  
  
    // Sınıf ait alan  
    private static int toplamKitapSayisi;  
  
    // final tanımlanmış 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  
public static int y; // Statik, genel bir tamsayı alanı.  
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.
- Programın temel yapı taşlarından biridir.
- Bir sınıfın davranışını tanımlar ve programın işlevselliğini sağlar.
- Metotlar iki temel türe sahiptir:
 - **Örneğe ait metotlar** bir nesne ile ilişkilidir,
 - **Sınıfa ait metotlar** sınıfın kendisi ile ilişkilidir, tüm nesnelerde ortaktır.
- Sınıf metotları, tüm sınıf örnekleri tarafından paylaşılır ve nesnelere özgü verilere erişemezler.



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 bloklarıdır.
- Kodun tekrar kullanılabilirliğini ve düzenini sağlar.
- Bir metot tanımlarken kullanılan **temel öğeler**:

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



Sınıfa Ait Nitelikler ve Metotlar

- Sınıf içerisinde "static" anahtar kelimesi olmadan tanımlanırlar.
- Her nesne, nitelik ve metotların kendi kopyalarına sahiptir.
- Nitelikler, bir nesnenin **özelliklerini** temsil eder ve nesneler arasında **farklı verilere** sahip olabilirler.
- Metotlar, nesnenin **davranışını** tanımlar ve o **nesneyle ilişkilidir**.
- Nitelik ve metotları kullanabilmek için **nesne oluşturmak** gereklidir.



Yapıcı Metotlar (Constructor)

- Bir nesne oluşturulduğunda otomatik olarak çağrılırlar.
- Nesne oluşturulurken belirli işlemleri gerçekleştirmek için kullanılır.
- Genellikle nitelikleri ilklendirir ve varsa diğer görevleri gerçekleştirir.
- Sınıf ile aynı adı taşır.
- Geri dönüş türü yoktur.
- Herhangi bir değer döndüremez.
- Genel erişim düzenleyici ile tanımlanır (public).



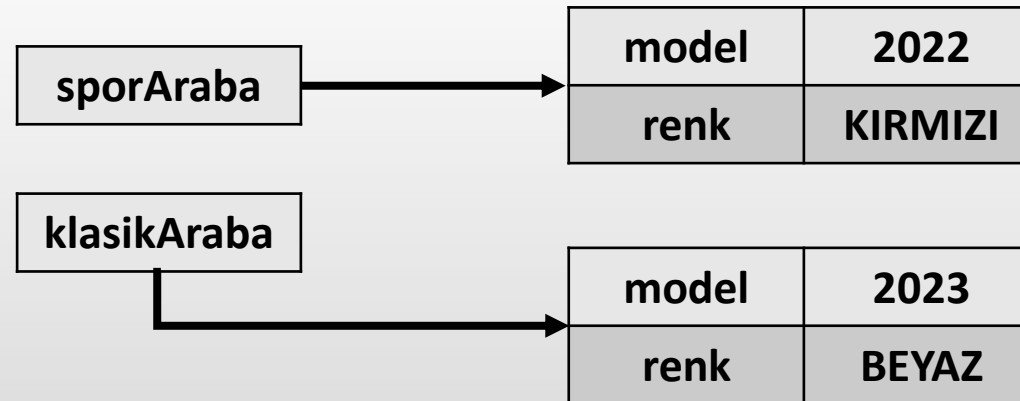
Yapıcı Metotlar (constructors)

```
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...  
}
```



Yapıcı Metotlar (constructors)

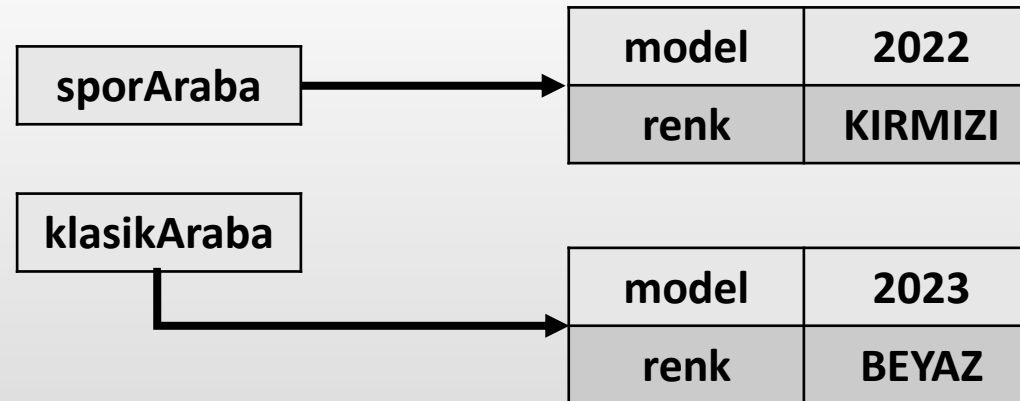
```
public static void main(String[] args) {  
    Araba sporAraba = new Araba("2022", "KIRMIZI");  
    Araba klasikAraba = new Araba("2023", "BEYAZ");  
}
```





Referanslar

- You can never use an object directly, only references to objects are stored.
- Bir nesne doğrudan kullanılamaz, referans ile erişilebilir.





Nokta (.) Operatörü

- Bir sınıfın nesnesi oluşturulduktan sonra, bu nesneye ait nitelik ve metotlara "nokta operatörü" kullanılarak erişilir.
- `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)



- Sınıf içindeki nitelikler (fields), genellikle dışarıdan direkt erişilememesi ve değiştirilememesi için «private» yapılır.
- **Erişimciler (Accessors)**
 - Sınıfın dışından niteliklere erişmek için kullanılan metotlardır.
 - Niteliklerin değerlerini okumak için kullanılırlar.
- **Değiştiriciler (Mutators)**
 - Sınıfın dışından niteliklere değer atamak için kullanılırlar.
 - Niteliklerin değerlerini değiştirirler.

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 dışarısından, nesnenin iç verilerine erişmek veya değiştirmek için sınıfın genel (**public**) metotları kullanılmalıdır.



Varsayılan Yapıcı Metot (default constructor)

- Bir nesne oluşturulduğunda, ilgili sınıfın yapıcı metodu **her zaman** çağrılır.
- Eğer bir yapıcı metot yazılmamışsa, Java bir tane sağlar.
- Bir sınıf için yapıcı metot **yazılmadığında** kullanılır.
- Parametre almayan bir yapıcı metottur.
- Nesnenin sayısal niteliklerine 0, boolean niteliklerine false ve referans değişkenlerine "null" değeri atar.



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 yoktur.
- Statik nitelik ve metotlara erişmek için nokta (.) operatörü kullanılır.

`SınıfAdı.AlanAdı`

`SınıfAdı.MetotAdı (GerçekParametreler)`

- Statik üyelere erişmek için nesneleri kullanmak, tercih **edilmez**.

`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 işlemi, nesneler arasında ilişki kurar ve bir nesnenin başka bir nesneyi işaret etmesini sağlar.
- Kütüphanede kitabın rafta konumunu gösteren **etiket**, ilgili kitabın referansını içerir.
- Bir arabanın **anahtarı**, o arabayı 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

- Metodun çağrıldığı nesneyi işaret eder.
- Sınıfa ait (static) metotlar içinde kullanılamaz.
- Yapıcı metotlarda sık kullanılır. Sınıfın nitelik adı ile metodun parametre adı genellikle aynı olur.
- İlgili nesneyi bir metoda parametre olarak iletmek için kullanılır.



"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 türü veya sayısıyla, 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;  
}
```



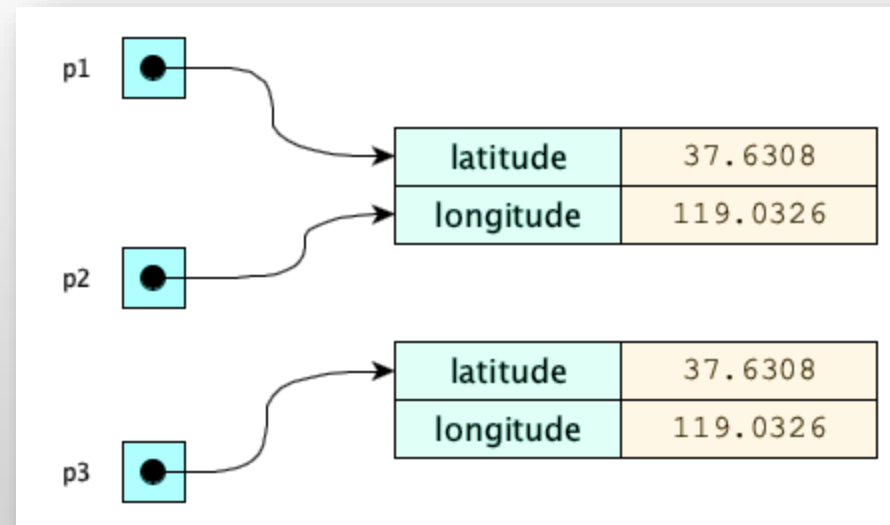
Nesneleri Karşılaştırma

- **Bellek Adresleri ile Karşılaştırma (==):**
 - İki nesne == ile karşılaştırıldığında, bellek adresleri karşılaştırılır.
 - Bellek adresi karşılaştırması, içerik karşılaştırmasına göre sınırlıdır.
 - **Örnek:** nesne1 == nesne2 sadece bellek adreslerini kontrol eder.
- **Eşitlik İçin equals() Metodu:**
 - İki nesnenin eşitliği için, tüm nesne özellikleri karşılaştırılır.
 - Bu amaçla, equals() metodunu özelleştirmek önemlidir.
 - equals(), nesnelerin içeriğine dayalı doğru ve güvenilir sonuçlar sağlar.
 - equals() içinde null kontrolü yapmak, hata olasılığını azaltır.



Nesneleri Karşılaştırma

- Variables can only store references to objects.
- Değişkenler sadece nesnelere ait referansları saklar.
- == operator only checks if the references refer to the same object.
- == işleci nesnelerin nitelikleri yerine bellek adreslerini karşılaştırır.





Örnek equals Metodu

```
public boolean equals(Object obj) {  
    // Aynı referansa sahipse, eşittir.  
    if (this == obj)  
        return true;  
    // Null kontrolü yapılır, ve nesne türü karşılaştırılır.  
    if (obj == null || getClass() != obj.getClass())  
        return false;  
    // Kisi nesnelerinin alanları karşılaştırılır.  
    Kisi kisi = (Kisi) obj;  
    return yas == kisi.yas && ad.equals(kisi.ad);  
}
```




getClass() ve instanceof

- **getClass()**

- Object sınıfı içinde tanımlıdır.
- Çalışma zamanında nesnenin ait olduğu sınıfı döndürür.
- Çalışma zamanında tür kontrolü yapılırken kullanılır.

- **instanceof**

- Nesnenin bir sınıfın örneği olup olmadığını kontrol etmek için kullanılır.
- Nesne belirtilen tipte ise true, değilse false döner.
- Tür dönüşümü (cast) yapmadan önce, nesnenin türünün uyumlu olup olmadığını kontrol etmek için kullanılır.
- Üst ve alt sınıflar karşılaştırıldığında true döner.



toString() Metodu

- Bir nesnenin temsilini döndüren bir Java Object sınıfı metodudur.
- Sınıflar, kendi özelliklerini içeren bu metodunun içeriğini özelleştirebilir.
- Nesnenin içeriğini anlamayı kolaylaştırır.
- Eğer sınıf toString()'i özelleştirmezse, Object sınıfındaki varsayılan toString() kullanılır.
- Eğer nesne null ise, null referansını dönmelidir.

```
public String toString() {  
    return "Ogrenci [ad=" + ad + ", numara=" + numara + "];"  
}
```



Wrapper Sınıflar

- Temel veri tiplerini nesne olarak temsil etmek için kullanılan sınıflardır.
- **Temel Veri Tipleri:** byte, short, int, long, float, double, char, Boolean
- **Wrapper Sınıflar Listesi:** Byte, Short, Integer, Long, Float, Double, Character, Boolean
- Bazı durumlarda, temel veri tiplerini nesne olarak kullanmak gerekebilir.
- Java'da generics ile çalışırken, nesne türleri kullanılması tercih edilir.



Wrapper Sınıflar

- Temel veri tiplerinde null değer atanamaz, ancak wrapper sınıflarda atanabilir.
- Wrapper sınıflar, temel veri tipleri üzerinde yardımcı metotlar sağlar.
- Doğrudan temel veri tiplerini kullanmak, bazı durumlarda daha performanslı olabilir.
- Otomatik ambalajlama (autoboxing) ve ambalajı açma (unboxing) durumlarına dikkat edilmelidir.



Autoboxing ve Unboxing

- **Autoboxing:** Temel veri tiplerini onların karşılık gelen wrapper sınıflarına otomatik olarak dönüştürme işlemidir.
- **Unboxing:** Wrapper sınıflarındaki değerleri otomatik olarak temel veri tiplerine dönüştürme işlemidir.

// Autoboxing: int tipindeki değer otomatik Integer sınıfına dönüş

```
int sayi = 42;
```

```
Integer wrapperSayi = sayi;
```

// Unboxing: Integer sınıfındaki değer otomatik int'e dönüştürülür.

```
Integer wrapperSayi = 42;
```

```
int sayi = wrapperSayi;
```



Wrapper Sınıflar

```
// int tipinde bir değişken
int sayi = 42;
// Integer wrapper sınıfı kullanımı
Integer wrapperSayi = Integer.valueOf(sayi);
// Autoboxing (Otomatik ambalajlama)
Integer wrapperSayi2 = sayi;
// Unboxing
int sayi2 = wrapperSayi2.intValue();
```



Kayıtlar (Records)

- Temiz kod yazma ihtiyaçlarını karşılamak üzere tasarlanmış sınıf türüdür.
- Değişmez (immutable) nesneleri temsil eder.
- Temel metotları otomatik oluşturur.
- Tüm nitelikler özeldir. (private)
- Erişim metotları otomatik tanımlanır. (get/set)
- equals, hashCode ve toString metodları otomatik yeniden tanımlanır.

```
record Nokta(double x, double y) {}
```



Kayıtlar (Records)

```
record Nokta(double x, double y) {}  
Nokta nokta1 = new Nokta(1.0, 2.5);  
Nokta nokta2 = new Nokta(1.0, 2.5);  
boolean esitMi = nokta1.equals(nokta2); // true
```




enum (Numaralandırma Türü)

- Özel bir veri türüdür.
- Sabit değerleri temsil etmek için kullanılır.
- Belirli değerlerin sabit ve sınırlı olmasını sağlar.
- Enum değerleri üzerinde döngü yapılabilir.

```
enum HaftaGunleri {  
    PAZARTESI, SALI, CARSAMBA, PERSEMBE, CUMA, CUMARTESI, PAZAR  
}  
HaftaGunleri[] tumGunler = HaftaGunleri.values();  
HaftaGunleri persembe = HaftaGunleri.valueOf("PERSEMBE");
```



İç Sınıflar (Inner Class)

- Bir sınıfın içinde tanımlanan sınıflardır.
- Dış sınıfın nitelik ve metotlarına doğrudan erişim sağlar.
- Genellikle Listener sınıflarını tanımlamak için kullanılır.
- **Statik İç Sınıf:** static anahtar kelimesi ile tanımlanır, dış sınıfın nesnesi olmadan da erişilebilir.
- **Üye İç Sınıf:** Dış sınıfın bir üyesi olarak tanımlanır, dış sınıfın örneği olmadan erişilemez.
- **Yerel İç Sınıf:** Bir metodun içinde tanımlanır, metodun içinden erişilebilir.
- **Anonim İç Sınıf:** İsimsiz sınıftır ve arayüz veya soyut sınıfları gerçekler.



Statik İç Sınıf

```
public class Outer {  
    // Dış sınıfın içinde statik bir iç sınıf  
    static class StaticInner {  
        void display() {  
            System.out.println("Statik İç Sınıf");  
        }  
    }  
    public static void main(String[] args) {  
        // dış sınıf örneği oluşturmadan erişim  
        Outer.StaticInner staticInner = new Outer.StaticInner();  
        staticInner.display();  
    }  
}
```



Üye İç Sınıf

```
public class Outer {  
    class MemberInner { // Dış sınıfın içinde üye bir iç sınıf  
        void display() {  
            System.out.println("Üye İç Sınıf");  
        }  
    }  
    public static void main(String[] args) {  
        // Dış sınıf örneği üzerinden üye iç sınıfa erişim  
        Outer outer = new Outer();  
        Outer.MemberInner memberInner = outer.new MemberInner();  
        memberInner.display();  
    }  
}
```



Yerel İç Sınıf

```
public class Outer {  
    void outerMethod() { // Bir metodun içinde tanımlanır  
        class LocalInner {  
            void display() {  
                System.out.println("Yerel İç Sınıf");  
            }  
        }  
        // Yerel iç sınıfa sadece bu metodun içinden erişim  
        LocalInner localInner = new LocalInner();  
        localInner.display();  
    }  
    public static void main(String[] args) {  
        Outer outer = new Outer();  
        outer.outerMethod();  
    }  
}
```



Anonim İç Sınıf

```
public class Outer {  
    public static void main(String[] args) {  
        // Genellikle arayüzleri veya soyut sınıfları uygular  
        MyInterface myInterface = new MyInterface() {  
            @Override  
            public void myMethod() {  
                System.out.println("Anonim İç Sınıf");  
            }  
        };  
        myInterface.myMethod();  
    }  
}
```



SON