Bölüm 2: Söz Dizimi Kuralları

JAVA ile Nesne Yönelimli Programlama

JAVA ile Yazılım Geliştirme

- Kod düz metin dosyalarına yazılır.
- Dosyaların uzantısı **".java"** şeklindedir.
- ".java" uzantısı, Java dilinde kaynak kod dosyalarını belirtir.
- Bu uzantı sayesinde, dosyalar doğru bir şekilde işlenir.

Düz Metin Dosyalarının Avantajları

- **Okunaklı**: Düz metin dosyaları insanlar için okunaklı, anlaması kolaydır.
- **Taşınabilir**: .java dosyaları farklı geliştirme ortamlarında da kullanılabilir.
- **Kolay Düzeltilebilir**: Hataları düzeltmek veya kodu güncellemek basittir.

Programlama Araçları

- **Java** programlama dili ile kullanılabilecek birçok araç vardır.
- IDE (*İntegrated Development Environment*)
  - **Eclipse**,
  - IntelliJ IDEA ve
  - NetBeans gibi entegre geliştirme ortamları bulunur.

Örnek .java Dosyası

```
public class MerhabaDunya {
    public static void main(String[] args) {
        System.out.println("Merhaba, Dünya!");
    }
}
```

Derleme

- Kaynak kodun, bilgisayar tarafından anlaşılması için derlenmesi gerekir.
- Derleme, kaynak kodun makine diline çevrilmesi anlamına gelir.
- Kaynak kodlar ".java" uzantılı düz metin dosyalarıdır.
- Derleme için "**javac**" komutu kullanılır.
- Kaynak kodlar derlendikten sonra, "**.class**" uzantılı dosyalara dönüşürler.
- .class dosyaları, Java sanal makinesi (**JVM**) tarafından yürütülür.

Derleme

- "MerhabaDunya.java" derlendikten sonra "MerhabaDunya .class" oluşur.
- Kaynak kodları derlemek,
  - Hataları kontrol etmek ve
  - Programın çalıştığından emin olmak için önemlidir.
- Derleme aşaması, derleyici tarafından yapılan iyileştirmelere bağlı olarak programın daha performanslı çalışmasına da katkı sağlar.

Byte Kod

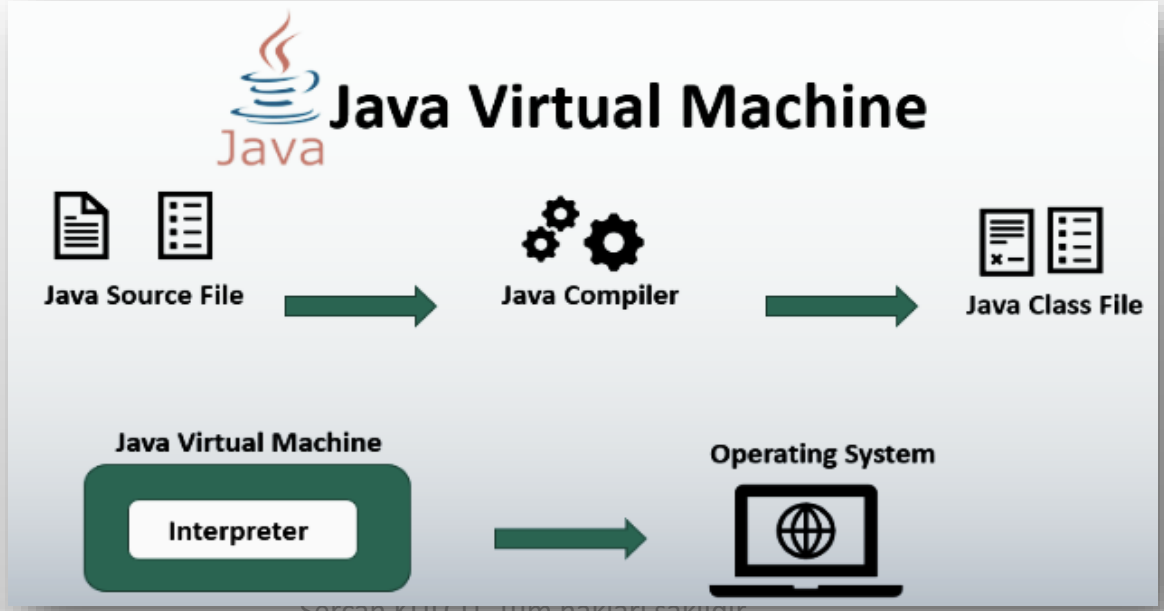
- ".class" dosyaları, işlemcinin doğrudan anlayabileceği bir kod içermez.
- Bu dosyalar, *Java VM* tarafından anlaşılabilen *byte kod*ları içerir.
- Byte kod, Java VM tarafından kullanılan makine dilidir.
- Kullanılan her bir talimat, basit birer byte kod olarak saklanır.
- Byte kodlar, **bağımsızlık** ve **taşınabilirlik** sağlar.
- Aynı byte kodlar, **farklı** işlemci ve işletim sistemleri üzerinde çalıştırılabilir.

Byte Kod Örneği

```
0: iconst_5
1: istore_1
2: getstatic     #16    // Field java/lang/System.out:Ljava/io/PrintStream;
5: iload_1
6: invokevirtual #22    // Method java/io/PrintStream.println:(I)V
9: return
```

Java Sanal Makinesi (JVM)

- Java uygulamalarının çalıştırılmasını sağlayan temel bileşendir.
- Byte kodları yorumlar ve uygulamayı işlemciye uyumlu şekilde yürütür.
- Java başlatıcı aracı (*laucher tool*), programları çalıştırmak için kullanılır.
- Bu araç, Java uygulamalarını başlatır ve JVM ile entegre çalışır.



Platform Bağımsızlık

- Java'nın temel özelliklerinden biridir.
- "*Bir Kez Yaz, Her Yerde Çalıştır*" ilkesi
- Aynı *byte kodlar* farklı işletim sistemlerinde çalıştırılabilir.
- JVM, birçok farklı işletim sistemi için mevcuttur.

Java Platformu

- Bir programın çalıştığı donanım veya yazılım ortamını ifade eder.
- Java Platformu, iki ana bileşeni içerir:
  - **Java Virtual Machine (JVM):**
    - Java uygulamalarını çalıştıran temel bileşendir.
    - Farklı donanım tabanlı platformlara taşınabilirlik sağlar.
  - **Application Programming Interface (API):**
    - Hazır yazılım bileşenleri kullanılarak hızlı geliştirim sağlar.
    - Grafik arayüzü, veritabanı, ağ iletişimi gibi birçok işlev içerir.

Java İle Temel İşlem Adımları

- **Adım 1: Derleme**
  - Kaynak kodun ".java" dosyasından ".class" dosyasına dönüştürülmesi.
  - **Örnek**: `javac HelloWorld.java`
- **Adım 2: Çalıştırma**
  - Derleme işleminden sonra çalıştırmak için "java" komutu kullanılır.
  - **Örnek**: `java HelloWorld`
- **Çıktı**
  - Program çalıştığında, ekrana "Hello world!" yazar.

Temel Programlama Unsurları

- **Değişkenler, Türler ve İfadeler**
  - Verileri saklamak ve işlemek için değişkenlere ihtiyaç vardır.
  - Değişkenler belirli türlerle ilişkilendirilir ve ifadeler aracılığıyla işlenir.
- **Kontrol Akışı**
  - Programın hangi sırada ve nasıl çalışacağını belirler.
    - **Dallanma (*Branching*):** Belirli koşullara bağlı olarak programın farklı kısımlarının çalıştırılmasını sağlar. Örneğin, "eğer-ise" (if-else) ifadeleri bir tür dallanma oluşturur.
    - **Döngüler (*Loops*):** Belirli bir işlemi tekrarlamayı sağlar. "for" ve "while" döngü yapıları, tekrarlı işlemleri gerçekleştirirler.

Örnek Kod Parçası

```
int sayi = 5;

if (sayi > 0) {
    System.out.println("Sayı pozitif.");
} else {
    System.out.println("Sayı negatif veya sıfır.");
}
```

Değişkenler: Veri Depolamanın Temel Taşları

- Verileri saklamak ve işlemek için değişkenlere ihtiyaç vardır.
- Değişkenler, bir tür konteyner gibi düşünülebilir.
- Değişkenler, açıklayıcı ve anlamlı isimlerle tanımlanmalıdır.
- Her değişkenin kullanılmadan önce bildirilmesi/tanımlanması gerekir.
- Bir değişkenin tanımlanması, **tipi** ve **adı** şeklinde olur.
- Tanımlama işlemi noktalı virgülle sona erer.

```
int numara, cekSayisi, mevcutSayisi;
double miktar, faizOrani;
char cevap;
```

İlkel (Primitive) Veri Türleri

- Farklı veri türlerini temsil etmek için kullanılır.
- Her türün kendine özgü **bellek kullanım miktarı** ve **değer aralığı** vardır.
- İlkel veri türleri,
  - **Tam sayı**: int, short, long, byte
  - **Kesirli sayı**: float, double
  - **Karakter**: char
  - **Mantıksal**: boolean

İlkel Veri Türleri - byte

- Tür: Tam Sayı (Integer)
- Bellek Kullanımı: 1 bayt
- Değer Aralığı: -128 ile 127 arası

İlkel Veri Türleri - short

- Tür: Tam Sayı (Integer)
- Bellek Kullanımı: 2 bayt
- Değer Aralığı: -32,768 ile 32,767 arası

İlkel Veri Türleri - int

- Tür: Tam Sayı (Integer)
- Bellek Kullanımı: 4 bayt
- Değer Aralığı: -2,147,483,648 ile 2,147,483,647 arası

İlkel Veri Türleri - long

- Tür: Tam Sayı (Integer)
- Bellek Kullanımı: 8 bayt
- Değer Aralığı: -9,223,372,036,854,775,808 ile 9,223,372,036,854,775,807

İlkel Veri Türleri - float

- Tür: Ondalıklı Sayı (Floating-point)
- Bellek Kullanımı: 4 bayt
- Değer Aralığı: $\pm 3.40282347 \times 10^{38}$ ile $\pm 1.40239846 \times 10^{-45}$ arası

İlkel Veri Türleri - double

- Tür: Ondalıklı Sayı (Floating-point)
- Bellek Kullanımı: 8 bayt
- Değer Aralığı: ±1.7976931348623157 × 10^{308} ile ±4.9406564584124654 × 10^{-324} arası

İlkel Veri Türleri - char

- Tür: Tek karakter (Unicode)
- Bellek Kullanımı: 2 bayt
- Değer Aralığı: 0 ile 65,535 arasındaki tüm Unicode değerleri

İlkel Veri Türleri - boolean

- Tür: 1 bitlik mantıksal değer (Doğru veya Yanlış)

Tanımlayıcılar (Identifiers)

- Bir değişkenin veya diğer öğelerin adına *tanımlayıcı* denir.
- Tanımlayıcılar, özel kurallara tabidir.
- Bir tanımlayıcı sadece harf, rakam (0-9) ve alt çizgi karakteri (_) içerebilir.
- İlk karakter bir harf veya alt çizgi (_) olmalıdır. İlk karakter rakam olamaz.
- Tanımlayıcıların uzunluk sınırlaması yoktur.
- Java, büyük küçük harfe duyarlı bir dildir.
- "kisiAdi" ile "kisiadi" farklı değişken adlarını temsil eder.
- kullaniciAdi, toplamPuan, ogrenci_adi, veri1…

Anahtar Kelimeler (Reserved Words)

- Belirli görevleri yerine getiren kelimelerdir.
- Java'da bir dizi özel anahtar kelime bulunur.
- Bu kelimeler dikkatli bir şekilde kullanılmalıdır.
- Anahtar kelimeler, değişken, sınıf veya metot adı olarak kullanılamaz.

Anahtar Kelimeler (Reserved Words)

- **abstract**, bir sınıf veya metot tanımının soyut (abstract) olduğunu belirtir. Somut sınıflar tarafından uygulanması gereken metotları içerir.
- **assert**, programın belirli bir koşulu kontrol etmesini sağlar. Hata ayıklama ve doğrulama işlemlerinde kullanılır.
- **boolean**, yalnızca iki değeri temsil eder: true (doğru) ve false (yanlış). Koşullu ifadelerde ve mantıksal işlemlerde kullanılır.
- **break**, döngülerden veya anahtar kelimeleri içeren bir yapıdan çıkmak için kullanılır. Özellikle switch-case ve for-while döngülerinde sıkça kullanılır.
- **byte**, 8-bit (1 byte) yer kaplayan bir tamsayıyı temsil eder. Küçük tamsayılar için kullanılır.

Anahtar Kelimeler (Reserved Words)

- **case**, **switch** ifadesi içinde kullanılır. Bir durumu veya değeri temsil eder.
- **catch**, hata yakalama işleminde kullanılır. Hata nesnesini yakalar ve işler.
- **char**, karakter verilerini temsil eder. Bir karakteri (Unicode) saklar.
- **class**, sınıf tanımlamak için kullanılır. Java programlarının temel yapı taşlarından sınıflar, nesnelerin şablonlarını oluşturur.
- **continue**, döngü içinde kullanılır. Belirli bir koşulu karşılayan işlemleri atlayarak, döngünün bir sonraki adımına geçer.

Anahtar Kelimeler (Reserved Words)

- **default**, **switch** ifadesi içinde kullanılır. Herhangi bir durum eşleşmediğinde, varsayılan olarak yapılacak işlemi belirtir.
- **do**, döngülerde kullanılır. Belirli bir işlemi en az bir kez gerçekleştirmek için kullanılır. do-while döngüsünün bir parçasıdır.
- **double**, ondalıklı sayıları (çift hassaslıkta) temsil eder. Büyük ve hassas ondalıklı sayılar için kullanılır.
- **else**, koşullu ifadelerin bir parçasıdır ve bir koşulun doğru olmadığı durumda gerçekleşen işlemi tanımlar.
- **enum**, bir veri türüdür. Sabit değerlerin bir koleksiyonunu temsil eder. Belirli bir türdeki seçenekleri temsil etmek için kullanılır.

Anahtar Kelimeler (Reserved Words)

- **extends**, sınıflar arasında kalıtım (inheritance) ilişkisini tanımlar. Bir sınıfın diğer bir sınıfın niteliklerini ve metotlarını miras almasını sağlar.
- **final**, değişkenlere, metotlara veya sınıflara uygulanır. Değişkenin değerinin değiştirilemez, metodun yeniden yazılamaz veya sınıfın kalıtım yoluyla türetilemez olduğunu belirtir.
- **finally**, **try-catch** bloklarının bir parçasıdır. Bir işlem bloğunun sonunda her durumda çalıştırılmasını gerektiren kodu içerir.
- **float**, tek hassaslıkta ondalıklı sayıları temsil eder. Ondalıklı sayılar için kullanılır, **double** türünden daha az hassastır.
- **for**, döngülerde kullanılır. Belirli bir işlemi belirli bir koşul altında tekrarlanmasını sağlar.

Anahtar Kelimeler (Reserved Words)

- **if**, koşullu ifadeler oluşturmak için kullanılır. Belirli bir koşulu kontrol eder ve işlem akışını bu koşula göre yönlendirir.
- **implements**, bir sınıf bir arayüzü uygulamak için kullanır. Sınıf, arayüzü uygulayarak arayüzün belirlediği metotları sağlar.
- **import**, başka bir paketten kullanılacak sınıfları içe aktarmak için kullanılır. Farklı sınıfları projeye dahil etmeye yarar.
- **instanceof**, bir nesnenin bir sınıf veya arayüz tarafından oluşturulup oluşturulmadığını kontrol eder. Tür denetimi ve tür dönüşümü için kullanılır.
- **int**, tam sayıları temsil eder. Genellikle matematiksel hesaplamalar ve sayısal değerler için kullanılır.

Anahtar Kelimeler (Reserved Words)

- **interface**, arayüzleri tanımlamak için kullanılır. Arayüzler, belirli metotların imzasını ve davranışlarını tanımlayan sözleşmelerdir.
- **long**, tam sayıları temsil eder. **int** veri tipinin yetersiz kaldığı büyük tam sayılar için kullanılır.
- **native**, Java dilinde yazılmamış olan ve Java Sanal Makinesi (JVM) kontrolü altındaki metotları tanımlar. Performansı artırmak, platforma özgü özellikleri kullanmak gibi amaçlarla kullanılır.
- **new**, nesne oluşturmak için kullanılır. Nesne için bellekte yer ayrılmasını ve ilk değerlerinin verilmesini sağlar.
- **package**, bir sınıfın veya arayüzün bulunduğu isim uzayını (namespace) belirlemek için kullanılır. İsim çakışmalarını önlemeye yardımcı olur.

Anahtar Kelimeler (Reserved Words)

- **private**, sınıf içinde kullanılan bir erişim düzenleyicidir (access modifier). Bir niteliğin sadece aynı sınıf içinden erişilebilir olduğunu belirtir.
- **protected**, sınıfın bir niteliğine alt sınıflardan erişilebilir olduğunu belirten erişim düzenleyicidir.
- **public**, sınıfın bir niteliğine diğer tüm sınıflar tarafından erişilebilir olduğunu belirten bir erişim düzenleyicidir.
- **return**, bir metodu tamamlamak ve bir sonuç döndürmek için kullanılır.
- **short**, tam sayıları temsil eder. **int** veri tipine kıyasla daha küçük tam sayılar için kullanılır ve daha az bellek alanı kullanır.

Anahtar Kelimeler (Reserved Words)

- **static**, bir niteliğin nesneye ait olmayıp sınıfa ait olduğunu belirtir. Sınıfa ait bir nitelik, tüm sınıf örnekleri arasında paylaşılır.
- **strictfp**, ondalıklı sayı işlemlerinin taşınabilirliğini ve hassasiyetini sağlar. İşlem sonucunun farklı platformlarda tutarlı olmasını garanti eder.
- **super**, bir sınıfın, üst sınıfın metotlarına ve niteliklerine erişmesini sağlar.
- **switch**, bir değişkenin farklı değerlerine göre farklı işlemleri gerçekleştirmek amacıyla kullanılır.
- **synchronized**, çoklu iş parçacığı (multithreading) uygulamalarında senkronizasyonu sağlar. İş parçacıklarının paylaşılan kaynaklara güvenli bir şekilde erişmesini sağlar.

Anahtar Kelimeler (Reserved Words)

- **this**, bir sınıfın içindeki metot tarafından, nesnenin kendisini referans verebilmesi için kullanılır. Sınıfın niteliği ile metot içinde aynı isimli yerel değişkenleri ayırt etmeyi sağlar.
- **throw**, bir hata durumu (**exception**) oluşturmak ve fırlatmak için kullanılır. Programın normal akışını keserek, hata durumlarını işlemeyi sağlar.
- **throws**, bir metodun belirli hata durumlarını verebileceğini belirtmek için kullanılır. Metodun başlık kısmında yer alır ve istisna türlerini listeler.
- **transient**, bir nesnenin serileştirilirken bazı verilerinin dikkate alınmamasını sağlar. Nesnelerin durumlarını kaydederken kullanışlıdır.

Anahtar Kelimeler (Reserved Words)

- **try**, potansiyel olarak hata verebilecek kod bloklarını çevreleyerek istisnaları (**exception**) yakalamak ve işlemek için kullanılır.
- **void**, herhangi bir değer döndürmeyen metotları tanımlamak için kullanılır.
- **volatile**, çoklu iş parçacığı (multithreading) uygulamalarında kullanılır. Bir değişkenin her iş parçacığı tarafından güncel olarak okunmasını sağlar.
- **while**, döngü oluşturmak için kullanılır. Belirli bir koşul sağlandığı sürece belirli bir işlemi tekrarlar.

İsimlendirme Kuralları

- Sınıf türleri, büyük harfle başlar.
  - String, Ogrenci.
- İlkel veri tipleri, küçük harfle başlar.
  - Örneğin: float.
- Sınıf nitelikleri, küçük harfle başlar.
  - Örneğin: firstName, classAverage.
- Çok kelimeli tanımlamalar,
  - *camelCase* veya *PascalCase* kullanılarak oluşturulabilir.

Atama İfadeleri

- Değişkene değer atamak için kullanılır.
- "=" atama operatörü olarak bilinir.
- `değişken = ifade;`
- Sağ taraftaki ifade hesaplanır ve sonuç, sol taraftaki değişkene atanır.
- "ifade" başka bir değişken, sabit veya matematiksel işlem olabilir.
- Örnekler:
  - `sayi = 10; (sabit bir değer atama)`
  - `sonuc = sayi1 + sayi2; (iki değişkenin toplamını atama)`
  - `isim = "Ahmet";`

Değişkenlere İlk Değer Atama

- Bir değişken değer atanmadan tanımlanabilir.
- Değer atanmamış ilkel değişkenler, varsayılan bir değere sahip olur.
- Sınıf niteliğinin, nesne oluşturulmadan önceki değeri *null*'dur.
- Varsayılan değerlere güvenmemek gerekir.
- Örnek bir ilk değer atama ifadesi:
  - `int yas = 25;`

Sabitler (Constants)

- Sabit değerler veya özel sayılar için kullanılır.
- Kodu daha anlaşılır ve bakımı daha kolay hale getirir.
- 2, 3.7 veya 'y' gibi doğrudan ifadeler, sabit olarak adlandırılır.
- Tam sayı sabitleri, artı (+) ve eksi (-) işareti ile başlayabilir, virgül içeremez.
  - **Örnekler**: +42, -17
- Ondalık sayı sabitleri, "e" gösterimi kullanılarak da ifade edilebilir.
  - **Örnekler**: 3.14159265, 2.5e3 (2.5×10^{3}), 7.68e-2 (7.68×10^{-2})
- Örnek bir sabit tanımı:
  - `final int PI = 314;`

Ondalık Sayıların Hassaslığı

- Ondalık sayılar, sınırlı sayıda bit ile saklanırlar.
- Örneğin, 1.0/3.0 ifadesi tam olarak 1/3 değildir.
- 1.0/3.0 + 1.0/3.0 + 1.0/3.0 ifadesi tam olarak 1 etmez.
- Bu tür yaklaşımlar matematiksel hesaplamalarda hata oluşturabilir.

İsimlendirilmiş Sabitler (Named Constant)

- Değişmeyen ve anlaşılır sabit değerler tanımlamayı sağlar.
  - `public static final VeriTipi SabitAdı = Değer;`
- İsimlendirilmiş sabitler genellikle büyük harfle yazılır.
- Kelimeler arasında alt tire ("_") kullanılır.
  - `public static final double PI = 3.14159;`
  - `public static final int GUNLER_HAFTADA = 7;`
- `float alan = PI * r * r;`

Atama Uyumluluğu

- Java, farklı tipte değişkenleri destekler,
- Ancak, veri türlerinin uyumluluğu önemlidir.
- Atama sırası şu şekildedir:
  - **byte → short → int → long → float → double**
- Bir değer, sadece daha sağda bulunan bir türün değişkenine atanabilir.
- char türündeki bir değer, int türündeki bir değişkene atanabilir.

```
char karakter;
int tamsayi;
tamsayi = karakter; // mümkün
karakter = tamsayi; // hata
```

Veri Türü Dönüşümü (Casting)

- Bir veri türünün başka bir türe dönüştürülmesi mümkündür.
- **Dolaylı Dönüşüm (*Implicit Conversion*)**
  - Java'da bazı veri türleri arasında otomatik dönüşüm gerçekleşir.
  - `double ondalikDegisken = 5; // 5.0`
  - Örnekte, tam sayı değer otomatik olarak ondalık değere dönüşür.
- **Açık Dönüşüm (*Explicit Conversion*)**
  - Bazı durumlarda, veri türü dönüşümünü açıkça belirtmek gerekir.
  - Parantez içinde dönüştürülen veri türü belirterek yapılır.
  - `double ondalikDegisken = 5.0;`
  - `int tamSayiDegisken = (int)ondalikDegisken; // Geçerli, 5`
  - Örnekte, ondalık değer açık dönüşüm ile tam sayıya dönüştürülür.

Operatörler ve Öncelik Sırası

- Java'da işleçlerin işlem sırası, öncelik sırasına göre belirlenir.
- **1. öncelik**, tekil işleçler:
  - Artı (+), Eksi (-), Değil (!), Artırma (++), Azaltma (--)
- **2. öncelik**, ikili aritmetik işleçler:
  - Çarpma (*), Bölme (/), Modül (%)
- **3. öncelik** ikili aritmetik işleçler:
  - Toplama (+), Çıkarma (-)
- İkili işleçlerin öncelikleri eşitse, soldaki, sağdakinden önce işlem yapar.
- Tekil işleçlerin öncelikleri eşitse, sağdaki, soldakinden önce işlem yapar.
- Öncelik sırasını değiştirmek için **parantezler** kullanılır.

Karşılaştırma Operatörleri

- Değerler arasındaki ilişkiyi değerlendirmek için kullanılır.
- **Eşitlik (==)** iki değer eşitse true, değilse false döndürür.
- **Eşitsizlik (!=)** iki değer eşit değilse true, eşitse false döndürür.
- **Büyüklük ve Küçüklük (> ve <)**
  - >, soldaki değer büyükse true, değilse false döndürür.
  - <, soldaki değer küçükse true, değilse false döndürür.
- **Büyük Eşit ve Küçük Eşit (>= ve <=)**
  - >=, bir değerin diğerinden büyük veya eşit olup olmadığını kontrol eder.
  - <=, bir değerin diğerinden küçük veya eşit olup olmadığını kontrol eder.

Mantıksal Operatörler

- Koşulları ve ifadeleri değerlendirmek ve karşılaştırmak için kullanılır.
- **VE (AND) işleci (&&)**
  - İki koşul da true ise sonucu true yapar.
  - A && B, sadece A ve B true olduğunda sonuç true olur.
- **VEYA (OR) işleci (||)**
  - İki koşuldan en az biri true ise sonucu true yapar.
  - X || Y, X veya Y true olduğunda sonuç true olur.
- **DEĞİL (NOT) işleci (!)**
  - Bir koşulun değerini true ise false, false ise true yapar.
  - !A, A true ise sonuç false, A false ise sonuç true olur.

"if" İfadesi

- Bir koşulu değerlendirir.
- Eğer koşul doğru ise, içindeki kod bloğunu çalıştırır.
- "if" ifadesinin temel yapısı:

```
if (koşul) {
    // Koşul doğruysa buradaki kod çalışır.
}
```

"if-else" İfadesi

- "if-else" ifadesi, bir koşulu değerlendirir.
- Koşul doğru ise "if" bloğu çalışır, aksi takdirde "else" bloğu çalışır.
- "if-else" ifadesinin temel yapısı:

```
if (koşul) {
    // Koşul doğruysa buradaki kod çalışır.
} else {
    // Koşul yanlışsa buradaki kod çalışır.
}
```

"switch" İfadesi

- Bir değerin durumlarına (case) göre farklı kod bloklarını çalıştırır.
- "switch" ifadesinin temel yapısı:

```
switch (değer) {
    case durum1:
        // Durum 1 için kod
        break;
    case durum2:
        // Durum 2 için kod
        break;
    default:
        // Hiçbir durum uymuyorsa buradaki kod
}
```

Üçlü Operatör (Ternary Operator)

- Koşullu ifadeleri kısa ve okunaklı bir şekilde yazmayı sağlar.
- Koşul doğru ise bir değer, değilse başka bir değer kullanılır.
- Üçlü operatörün temel yapısı:

```
sonuç = (koşul) ? değer1 : değer2;
```

- Bir sayının pozitif veya negatif olduğunu belirleme.

```
int sayi = -5;
String sonuc = (sayi > 0) ? "Pozitif" : "Negatif";
System.out.println("Sayı " + sonuc);
```

İç İçe Üçlü Operatörler

- Üçlü operatörler iç içe kullanılabilir.
- Daha karmaşık koşulların kısa bir şekilde yazılmasına olanak tanır.
- Bir sayının sıfır, pozitif veya negatif olduğunu belirleme.

```
int sayi = -5;
sonuc = (sayi == 0) ? "Sıfır" : (sayi > 0) ? "Pozitif" : "Negatif";
System.out.println("Sayı " + sonuc);
```

"for" Döngüsü

- Bir işlemi belirli bir aralıkta tekrarlamak için kullanılır.
- Başlangıç değeri, koşul ve artırma/azaltma işlemi döngünün çalışmasını kontrol eder.
- Sonsuz döngü hatası için koşul dikkatli bir şekilde belirlenmeli.
- "for" döngüsünün temel yapısı:

```
for (başlangıç değeri; koşul; artırma/azaltma) {
    // Döngü içinde yapılacak işlem
}
```

"while" Döngüsü

- Bir koşul sağlandığı sürece bir işlemi tekrarlar.
- Koşulun nasıl kontrol edileceği, döngünün çalışma süresini belirler.
- Döngünün kaç kez çalışacağı önceden bilinmez.
- "while" döngüsünün temel yapısı:

```
while (koşul) {
    // Döngü içinde yapılacak işlem
}
```

"do-while" Döngüsü

- Bir işlemin en az bir kez çalışmasını sağlar.
- Ardından koşulu kontrol eder.
- Koşul adım sonunda kontrol edilir.
- Kullanıcı girişi gibi senaryolar için uygundur.
- "do-while" döngüsünün temel yapısı:

```
do {
    // Döngü içinde yapılacak işlem
} while (koşul);
```

"break" İfadesi

- Döngüden veya "switch" ifadesinden çıkmak için kullanılır.
- Döngüyü sonlandırmak için "break" kullanımı.

```
for (int i = 1; i <= 10; i++) {
  if (i == 5) {
    break; // Döngüyü sonlandır
  }
  System.out.println(i);
}
```

"break" İfadesi

- "break" kullanarak "switch" bloğunundan çıkma.

```
int gun = 3;
switch (gun) {
    case 1:
        System.out.println("Pazartesi");
        break;
    case 2:
        System.out.println("Salı");
        break;
    // Diğer günler...
}
```

"continue" İfadesi

- Bir koşul sağlandığında, döngüyü bir sonraki değerden devam ettirir.
- Koşulu sağlayan sayıları atlayarak sadece belirli sayıları yazdırma.

```
for (int i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
        continue; // Çift sayıları atla
    }
    System.out.println(i);
}
```

SON