Bölüm 2: Söz Dizimi Kuralları

JAVA ile Nesne Yönelimli Programlama

JAVA ile Yazılım Geliştirme

- Kodlarımızı yazdığımız yer düz metin dosyalarıdır. Bu dosyaların uzantısı genellikle **".java"** şeklindedir.
- ".java" uzantısı, Java programlama dilindeki kaynak kod dosyalarını belirtir. Bu uzantı sayesinde bilgisayarlarımız, bu dosyaları doğru bir şekilde işleyebilir.

Düz Metin Dosyalarının Avantajları

- **Okunaklıdır**: Düz metin dosyaları insanlar için okunaklıdır, bu nedenle kodları anlamak daha kolaydır.
- **Taşınabilir**: .java dosyalarını farklı geliştirme ortamlarında ve bilgisayarlarda kolayca kullanabilirsiniz.
- **Kolay Düzeltme**: Hataları düzeltmek veya kodu güncellemek düz metin dosyalarında basittir.

Programlama Araçları

- **Java** programlama dilini öğrenirken kullanabileceğiniz birçok programlama aracı vardır.
- Bunlar arasında **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 Nedir?

- Kaynak kodlarımızı yazdıktan sonra, bilgisayarlarımız bu kodları anlayabilmesi için derlenmesi gerekir.
- Derleme, kaynak kodunun makine diline çevrilmesi anlamına gelir.
- Java'da, kaynak kodlarımız ".java" uzantılı düz metin dosyalarıdır.
- Bu dosyaları çalıştırılabilir hale getirmek için "**javac**" adlı derleyici kullanılır.
- Kaynak kodlar "javac" ile derlendikten sonra, "**.class**" uzantılı dosyalara dönüşürler.
- .class dosyaları, Java sanal makinesi (**JVM**) tarafından yürütülmek üzere hazır hale getirilir.

Derleme Nedir?

- **Örnek**:
  - "MerhabaDunya.java" dosyası derlendikten sonra "MerhabaDunya .class" adlı bir dosya oluşturulur.
- Kaynak kodları derlemek, hataları kontrol etmek ve programın çalıştığından emin olmak için önemlidir.
- Derleme aşaması, programın daha hızlı çalışmasına da katkı sağlar.

Byte Kod Nedir?

- ".class" dosyaları, işlemcinizin doğrudan anlayabileceği naif bir kod içermez.
- Bu dosyalar, Java VM tarafından anlaşılabilen byte kodları içerir.
- Byte kod, Java programlarının Java VM tarafından yürütülmesi için kullanılan makine dilidir.
- Her komut, işlemcinin doğrudan anlayabileceği basit birer byte kod olarak saklanır.
- Byte kodlar, Java'nın anahtar özelliklerinden biri olan **bağımsızlık** ve **taşınabilirlik** sağlar.
- Aynı byte kodları **farklı** işlemciler ve işletim sistemleri üzerinde çalıştırabilirsiniz.

Byte Kod Nedir?

- İşlem Adımı
  - Java programı yazıldığında, kaynak kod ".java" dosyası olarak başlar.
  - Kaynak kod "javac" derleyicisi tarafından ".class" dosyasına dönüştürülür, içerisinde byte kodlar bulunur.
  - Java VM, ".class" dosyasındaki byte kodları çalıştırır.
- **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)

- JVM, Java uygulamalarının çalıştırılmasını sağlayan temel bir bileşendir. Byte kodları yorumlar ve uygulamayı işlemciye uygun şekilde yürütür.
- Java başlatıcı aracı (laucher tool), Java programlarını çalıştırmak için kullanılan bir araçtır. Bu araç, Java uygulamalarını başlatır ve JVM ile entegre çalışır.

Platform Bağımsızlık

- Platform bağımsızlığı, Java'nın temel özelliklerinden biridir.
- "Bir Kez Yaz, Her Yerde Çalıştır" ilkesi, aynı byte kodlarının farklı işletim sistemlerinde çalışabilmesini ifade eder.
- JVM, birçok farklı işletim sistemi için mevcuttur.
- Bu nedenle, aynı ".class" dosyaları Microsoft Windows, Solaris OS, Linux veya Mac OS gibi farklı işletim sistemlerinde çalışabilir.

Java Platform

- Bir platform, bir programın çalıştığı donanım veya yazılım ortamını ifade eder. Java Platformu, programların çalıştığı ortamı sağlayan bir yapıdır.
- Java Platformu, iki ana bileşeni içerir:
  - **Java Virtual Machine (JVM):** Java Platformunun temelini oluşturur ve farklı donanım tabanlı platformlara taşınabilir. Java uygulamalarını çalıştıran temel bileşendir. Farklı donanım tabanlı platformlara taşınabilirlik sağlar.
  - **Java Application Programming Interface (API):** Birçok hazır yazılım bileşenini içeren büyük bir koleksiyondur. Java API, kullanıcıların hazır bileşenleri kullanarak uygulamalarını hızlıca geliştirmelerini sağlar. Bu API, grafik arayüzleri, veritabanı işlemleri, ağ iletişimi ve diğer birçok işlevi içerir.

Java İle Temel İşlem Adımları

- **Adım 1: Derleme**
  - İlk adım, Java kaynak kodunun derlenmesidir. Derleme işlemi, kaynak kodun ".java" dosyasından ".class" dosyasına dönüştürülmesini içerir.
  - **Örnek**: `javac HelloWorld.java`
- **Adım 2: Çalıştırma**
  - Derleme işlemi tamamlandıktan sonra, Java programınızı çalıştırabilirsiniz. Çalıştırma işlemi için "java" komutu kullanılır.
  - **Örnek**: `java HelloWorld`
- **Çıktı**
  - Program çalıştığında, ekrana "Hello world!" yazar.

Temel Programlama Unsurları

- **Değişkenler, Türler ve İfadeler**
  - Programlamada verileri saklamak ve işlemek için değişkenlere ihtiyaç vardır. Değişkenler belirli türlerle ilişkilendirilir ve ifadeler aracılığıyla işlenir.
- **Kontrol Akışı**
  - Programlar, belirli bir sırayla çalışmazlar. Kontrol akışı, programın nasıl çalıştığını 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 tekrarlayarak programın verimliliğini artırır. "for" ve "while" gibi döngü yapıları, tekrarlı işlemleri gerçekleştirmek için kullanılır.

Basit Bir 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

- Programlarda, verileri saklamak ve işlemek için değişkenlere ihtiyaç vardır. Bu değişkenler, tıpkı bir tür konteyner gibi düşünülebilir.
- Değişkenleri, açıklayıcı ve anlamlı isimlerle tanımlamak önemlidir.
- Her değişkenin kullanılmadan önce bildirilmesi/tanımlanması gerekir.
- Bir değişkenin tanımlanması, **tipi** ve **adı** şeklinde olur. Tanımlama işlemi noktalı virgülle sona erer.

```
int numara, çekSayısı, mevcutSayısı;
double miktar, faizOranı;
char cevap;
```

İlkel Veri Türleri

- İlkel veri türleri, farklı veri türlerini temsil etmek için kullanılır.
- Her türün kendi **bellek kullanımı** ve **değer aralığı** vardır.
- İlkel veri türleri, programlamada veri işleme ve depolamada temel yapı taşlarıdır.
  - 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 arası

İlkel Veri Türleri - float

- Tür: Ondalıklı Sayı (Floating-point)
- Bellek Kullanımı: 4 bayt
- Değer Aralığı: ±3.40282347 × 10^38 ile ±1.40239846 × 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 programlama dilinde, bir değişkenin veya diğer öğelerin adına "tanımlayıcı" denir.
- Tanımlayıcılar, programlama dilinin bir parçasıdır ve 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 uzunluğuna sınırlama yoktur.
- Java, harf büyüklüğüne duyarlı bir dildir. Bu, «kisiAdi" ile «kisiadi" gibi farklı değişken adlarının iki farklı değişkeni temsil ettiği anlamına gelir.
- **Örnek Tanımlayıcılar**
  - «kullaniciAdi», «toplamPuan», «ogrenci_adi», «veri1»

Java Özel Anahtar Kelimeler (Reserved Words)

- Özel anahtar kelimeler, bir programın yapı taşlarını oluşturan ve belirli görevleri yerine getiren kelimelerdir.
- Java'da bir dizi özel anahtar kelime bulunur ve bu kelimelerin dikkatli bir şekilde kullanılması gerekir.
- Özel anahtar kelimeler, değişken, sınıf veya metot adları olarak kullanılamaz.

Java Özel Anahtar Kelimeler (Reserved Words)

- **abstract** kelimesi, bir sınıf veya metot tanımının soyut (abstract) olduğunu belirtmek için kullanılır. Soyut sınıflar, somut sınıfların temelini oluşturan ve somut sınıflar tarafından uygulanması gereken metotları içerir.
- **assert**, programın belirli bir koşulu kontrol etmesini ve belirtilen bir şartın sağlanıp sağlanmadığını denetlemesini sağlar. Hata ayıklama ve doğrulama işlemlerinde kullanılır.
- **boolean**, yalnızca iki değeri temsil eden bir veri türüdür: true (doğru) ve false (yanlış). Koşullu ifadelerde ve mantıksal işlemlerde kullanılır.
- **break**, döngülerden veya anahtar kelimeleri içeren bir yapıdan çıkmak için kullanılır. Özellikle switch ve for döngülerinde sıkça kullanılır.
- **byte**, 8-bit (1 byte) bir tamsayıyı temsil eder. Genellikle küçük tamsayılar için kullanılır.

Java Özel Anahtar Kelimeler (Reserved Words)

- **case**, **switch** ifadesi içinde kullanılan bir anahtar kelimedir. Belirli bir duruma veya değere karşılık gelen işlemleri yönlendirmek için kullanılır.
- **catch**, hata yakalama işlemi sırasında kullanılan bir anahtar kelimedir. Hata nesnelerini yakalar ve işler.
- **char**, karakter verilerini temsil eden bir veri türüdür. Tek bir karakteri (genellikle Unicode) depolar.
- **class**, bir sınıf tanımlamak için kullanılan bir anahtar kelimedir. Java programlarının temel yapı taşlarından biri olan sınıflar, nesnelerin şablonlarını oluşturur.
- **continue**, döngü içinde kullanılan bir anahtar kelimedir. Belirli bir koşulu karşılayan işlemleri atlar ve döngünün bir sonraki adımına geçer.

Java Özel Anahtar Kelimeler (Reserved Words)

- **default**, **switch** ifadesi içinde kullanılan bir anahtar kelimedir. Herhangi bir durumun eşleşmediği varsayılan işlemi belirler.
- **do**, döngülerde kullanılan bir anahtar kelimedir. Belirli bir işlemi en az bir kez gerçekleştirmek için kullanılır. Do-While döngüsünün bir parçasıdır.
- **double**, ondalıklı sayıları (çift hassaslıkta) temsil eden bir veri türüdür. Genellikle daha 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 ve sabitlerin bir koleksiyonunu temsil eder. Genellikle belirli bir türdeki seçenekleri temsil etmek için kullanılır.

Java Özel Anahtar Kelimeler (Reserved Words)

- **extends**, sınıflar arasında kalıtım (inheritance) ilişkisini tanımlamak için kullanılan bir anahtar kelimedir. Bir sınıfın diğer bir sınıfın özelliklerini ve yöntemlerini miras almasını sağlar.
- **final**, değişkenlere, metotlara veya sınıflara uygulanan bir anahtar kelimedir. Bir değişkenin değerinin değiştirilemez olduğunu, bir metodun yeniden yazılamaz olduğunu veya bir sınıfın alt sınıflara miras verilemeyeceğini belirtir.
- **finally**, **try-catch** bloklarının bir parçası olarak kullanılır. Bir işlem bloğunun sonunda her durumda çalıştırılmasını gerektiren kodu içerir.
- **float**, tek hassaslıkta ondalıklı sayıları temsil eden bir veri türüdür. Genellikle ondalıklı sayılar için kullanılır, ancak double türünden daha az hassastır.
- **for**, döngülerde kullanılan bir anahtar kelimedir. Belirli bir işlemi belirli bir koşul altında tekrarlayarak kullanılır.

Java Özel Anahtar Kelimeler (Reserved Words)

- **if**, koşullu ifadeler oluşturmak için kullanılan bir anahtar kelimedir. Belirli bir koşulu kontrol eder ve işlem akışını bu koşula göre yönlendirir.
- **implements**, bir sınıfın bir arayüzü uygulamak için kullanıldığı bir anahtar kelimedir. Bir sınıf, belirli bir arayüzü uygulayarak arayüzün belirlediği metotları sağlar.
- **import**, Java programlarında başka bir paketten veya sınıftan kullanılacak sınıfları içe aktarmak için kullanılır. Bu, farklı sınıfları projenize dahil etmenizi sağlar.
- **instanceof**, bir nesnenin belirli bir sınıf veya arayüz tarafından oluşturulup oluşturulmadığını kontrol etmek için kullanılır. Özellikle tür denetimi ve tür dönüşümü durumlarında kullanışlıdır.
- **int**, tam sayıları temsil eden bir veri türüdür. Genellikle matematiksel hesaplamalar ve sayısal değerler için kullanılır.

Java Özel Anahtar Kelimeler (Reserved Words)

- **interface**, Java'da kullanılan bir anahtar kelimedir ve arayüzleri tanımlamak için kullanılır. Arayüzler, belirli metotların imzasını ve davranışlarını tanımlayan sözleşmelerdir.
- **long**, uzun tam sayıları temsil eden bir veri türüdür. Genellikle büyük tam sayılar için kullanılır.
- **native**, Java dilinde yazılmamış olan ve Java Sanal Makinesi (JVM) için özgün kod içeren metotları tanımlamak için kullanılır. Genellikle platforma özgü işlevlerin kullanılmasında kullanılır.
- **new**, yeni bir nesne oluşturmak için kullanılan bir anahtar kelimedir. Yeni bir nesnenin bellekte ayrılmasını ve ilklendirilmesini sağlar.
- **package**, Java programlarını düzenlemek ve paketlemek için kullanılan bir anahtar kelimedir. Sınıfları ve paketleri gruplandırmak ve düzenlemek için kullanılır.

Java Özel Anahtar Kelimeler (Reserved Words)

- **private**, sınıf içinde kullanılan bir erişim belirleyici (access modifier) anahtar kelimesidir. Bir üyenin sadece aynı sınıf içinde erişilebilir olduğunu belirtir.
- **protected**, sınıfın alt sınıfları ve aynı paket içindeki sınıflar tarafından erişilebilen bir erişim belirleyici anahtar kelimesidir. Üye, alt sınıflar veya aynı paket içindeki sınıflar tarafından kullanılabilir.
- **public**, herkes tarafından erişilebilen bir erişim belirleyici anahtar kelimesidir. Bir üyenin her yerden erişilebilir olduğunu belirtir.
- **return**, bir metodu tamamlamak ve sonuç döndürmek için kullanılan bir anahtar kelimedir. Metottan bir değer döndürmek için kullanılır.
- **short**, kısa tam sayıları temsil eden bir veri türüdür. Genellikle daha küçük tam sayılar için kullanılır ve daha az bellek kullanır.

Java Özel Anahtar Kelimeler (Reserved Words)

- **static**, bir sınıfa veya metoda ait olduğunu belirten bir anahtar kelimedir. Sınıfa ait bir üye, tüm sınıf örnekleri arasında paylaşılır.
- **strictfp**, ondalıklı sayı işlemlerinin taşınabilirliğini ve hassasiyetini sağlayan bir anahtar kelimedir. Belli bir ondalıklı sayı işlemi sonucunun farklı platformlarda aynı sonucu üretmesini garanti eder.
- **super**, bir sınıfın üst sınıfına veya süper sınıfına erişim sağlayan bir anahtar kelimedir. Alt sınıfın üst sınıfın metotlarına veya üyelerine erişmesini sağlar.
- **switch**, çoklu seçim yapmak için kullanılan bir kontrol yapısıdır. Belirli bir ifadeyi değerlendirir ve farklı durumlara göre işlemler yapmanıza olanak tanır.
- **synchronized**, çoklu iş parçacığı (multithreading) ortamlarında senkronizasyonu sağlamak için kullanılır. İş parçacıklarının güvenli bir şekilde paylaşılan kaynaklara erişmesini sağlar.

Java Özel Anahtar Kelimeler (Reserved Words)

- **this**, bir sınıfın içindeki bir üye veya metot tarafından o sınıfın kendisini referans verebilmesi için kullanılan bir anahtar kelimedir. Sınıfın içinde aynı isimli yerel değişkenlerle sınıf üyelerini ayırmak için kullanılır.
- **throw**, bir istisna (**exception**) oluşturmak ve fırlatmak için kullanılan bir anahtar kelimedir. Programın normal akışını keserek, hata durumlarını işlemek için kullanılır.
- **throws**, bir metodun belirli türde istisnaları fırlatabileceğini belirtmek için kullanılan bir anahtar kelimedir. Metodun başlık kısmında yer alır ve istisna türlerini listeler.
- **transient**, bir nesnenin belirli verilerinin serileştirilirken dikkate alınmamasını sağlayan bir anahtar kelimedir. Özellikle nesnelerin durumlarını kaydederken kullanışlıdır.

Java Özel Anahtar Kelimeler (Reserved Words)

- **try**, istisna (**exception**) yakalama blokları oluşturmak için kullanılan bir anahtar kelimedir. Potansiyel olarak hata verebilecek kodları çevreleyerek istisnaları yakalamak ve işlemek için kullanılır.
- **void**, bir metodu belirli bir değer döndürmeyen bir geri dönüş türünü temsil eden bir anahtar kelimedir. Bu, metotların sadece işlemleri gerçekleştirdiği ve bir değer döndürmediği durumlar için kullanılır.
- **volatile**, çoklu iş parçacığı (multithreading) programlamasında kullanılan bir anahtar kelimedir. Bir değişkenin değerinin her iş parçacığı tarafından güncel ve görünür olmasını sağlar.
- **while**, döngüler oluşturmak için kullanılan bir anahtar kelimedir. Belirli bir koşul sağlandığı sürece belirli bir işlemi tekrarlamak için kullanılır.

İsimlendirme Kuralları

- Programlama dünyasında, uygun isimlendirme kuralları ve standartları önemlidir. Java'da da belirli isimlendirme kuralları vardır.
- Sınıf türleri, büyük harfle başlar. Örneğin: String. İlk harf büyük olmalıdır ve takip eden harf büyük olabilir.
- İlkel veri tipleri, küçük harfle başlar. Örneğin: float. İlk harf küçük olmalıdır.
- Hem sınıf türleri hem de ilkel veri türleri için değişkenler, küçük harfle başlar. Örneğin: firstName, classAverage. İlk harf küçük olmalıdır ve takip eden kelimeler büyük harfle başlayabilir.
- Çok kelimeli isimler, "camelCase" veya "PascalCase" kullanılarak oluşturulabilir.

Atama İfadeleri

- Bir atama ifadesi, bir değişkene bir değer atamak için kullanılır. «=» atama operatörü olarak bilinir.
- `değişken = ifade;` Burada "değişken," değeri alacak değişkeni temsil eder. "ifade," atanan değeri ifade eder.
- Atama ifadesi içinde "ifade," başka bir değişken, bir sabit veya bir matematiksel işlemin sonucu olabilir. Örnekler:
  - `sayi = 10; (sabit bir değer atama)`
  - `sonuc = sayi1 + sayi2; (iki değişkenin toplamını atama)`
- Sağ taraftaki ifade hesaplanır ve sonuç, sol taraftaki değişkene atanır.
- Örnek bir atama ifadesi: `isim = "Ahmet";` Bu ifade, "isim" adlı bir değişkene "Ahmet" değerini atar.

Değişkenlere İlk Değer Atama

- Bir değişken, tanımlanmış ancak henüz bir değer atanmamışsa, «ilklendirilmemiş» olarak adlandırılır.
- Bir sınıf değişkeninin, nesne oluşturulmadan önce değeri "null" olarak kabul edilir.
- İlklendirilmemiş ilkel değişkenler, varsayılan bir değere sahip olabilir.
- Varsayılan değerlere güvenmek, programların hatalı davranışlara neden olabilir.
- Örnek bir ilklendirme ifadesi: `int yas = 25;`

Sabitler (Constants)

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

Ondalık Sayıların Hassaslığı

- Ondalık sayılar, genellikle sınırlı sayıda bit ile saklandıkları için yaklaşımdır.
- Örnek olarak, 1.0/3.0 ifadesi tam olarak 1/3 değildir. Bunun nedeni, ondalık sayıların sınırlı kesirli ifadelerle temsil edilmesidir.
- Örneğin, 1.0/3.0 + 1.0/3.0 + 1.0/3.0 ifadesi tam olarak 1 sonucunu vermez.
- Bu tür yaklaşımların matematiksel hesaplamalarda potansiyel hata kaynakları olabileceği unutulmamalıdır.
- Kesirli ifadelerle çalışırken, sonuçlar tam olarak beklenen gibi olmayabilir.

İsimlendirilmiş Sabitler (Named Constant)

- Java, bir değişkeni tanımlamanıza, başlangıç değeri atamanıza ve bu değişkenin değerini sonradan değiştirmemenizi sağlayan bir mekanizma sunar. İsimlendirilmiş sabitler, değişmeyen ve anlaşılır sabit değerler tanımlamanıza yardımcı olur.
- İsimlendirilmiş sabitler, genellikle şu şekilde tanımlanır:
  - `public static final VeriTipi SabitAdı = Değer;`
- İsimlendirilmiş sabitler genellikle tüm büyük harfle yazılır ve kelimeler arasında alt tire ("_") kullanılır. Örnekler:
  - `public static final double PI = 3.14159;`
  - `public static final int GUNLER_HAFTADA = 7;`
- Örnek bir kullanım: `float alan = PI * r * r;`

Atama Uyumluluğu

- Java, farklı tipte değişkenleri destekleyen bir programlama dilidir, bu nedenle veri türlerinin uyumluluğu önemlidir.
- Atama sırası şu şekildedir:
  - **byte → short → int → long → float → double**
- Bir türün değeri, daha sağda bulunan (daha büyük kapasiteli) bir türün değişkenine atanabilir. Ancak tam tersi geçerli değildir.
  - Bir char türündeki bir değer, bir int türündeki bir değişkene atanabilir.

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

Veri Türü Dönüşümü (Casting)

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

Operatörler ve Öncelik Sırası

- Java'da operatörlerin işlem sırası, öncelik sırasına göre belirlenir.
- **İlk öncelik**, tekil operatörleredir:
  - Artı (+), Eksi (-), Değil (!), Artırma (++), Azaltma (--)
- **İkinci öncelik**, ikili aritmetik operatörlere ait:
  - Çarpma (*), Bölme (/), Modül (%)
- **Üçüncü öncelik** yine ikili aritmetik operatörlere ait:
  - Toplama (+), Çıkarma (-)
- Eğer ikili operatörlerin öncelikleri eşitse, sol taraftaki operatör(ler), sağ taraftaki operatörden önce işlem yapar.
- Eğer tekil operatörlerin öncelikleri eşitse, sağ taraftaki operatör, sol taraftaki operatörden önce işlem yapar.
- Öncelik sırasını değiştirmek için **parantezler** kullanılabilir.

Java Karşılaştırma Operatörleri

- Karşılaştırma operatörleri, değerler arasındaki ilişkiyi değerlendirmek için kullanılır.
- **Eşitlik Karşılaştırma (==)** iki değeri karşılaştırır ve eşitse true, değilse false döndürür.
- **Eşitsizlik Karşılaştırma (!=)** iki değeri karşılaştırır ve eşit değilse true, eşitse false döndürür.
- **Büyüklük Karşılaştırma (> ve <)**
  - > operatörü, bir değerin diğerinden büyük olup olmadığını kontrol eder.
  - < operatörü ise bir değerin diğerinden küçük olup olmadığını kontrol eder.
- **Büyük Eşit ve Küçük Eşit Karşılaştırma (>= ve <=)**
  - >= operatörü, bir değerin diğerinden büyük veya eşit olup olmadığını kontrol eder.
  - <= operatörü ise bir değerin diğerinden küçük veya eşit olup olmadığını kontrol eder.

Mantıksal Operatörler

- Koşulları ve ifadeleri değerlendirmek ve karşılaştırmak için kullanılır.
- **VE (AND) Operatörü (&&)**
  - iki koşulun da true olduğu durumda sonucu true yapar.
  - Örnek: A && B sadece A ve B true olduğunda sonuç true olur.
- **VEYA (OR) Operatörü (||)**
  - iki koşuldan en az birinin true olduğu durumda sonucu true yapar.
  - Örnek: X || Y X veya Y true olduğunda sonuç true olur.
- **DEĞİL (NOT) Operatörü (!)**
  - bir koşulun tersini alır. Yani true ise false, false ise true yapar.
  - Örnek: !A A true ise sonuç false, A false ise sonuç true olur.

"if" İfadesi

- "if" ifadesi, bir koşulu değerlendirir. Eğer koşul doğruysa, içindeki kod bloğunu çalıştırır.
- "if" ifadesinin temel yapısı şu şekildedir:

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

"if-else" İfadesi

- "if-else" ifadesi, bir koşulu değerlendirir. Eğer koşul doğruysa "if" bloğu çalışır, aksi takdirde "else" bloğu çalışır.
- "if-else" ifadesinin temel yapısı şu şekildedir:

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

"switch" İfadesi

- "switch" ifadesi, bir değerin durumlarına (case) göre farklı kod bloklarını çalıştırır.
- "switch" ifadesinin temel yapısı şu şekildedir:

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

Java Üçlü Operatör (Ternary Operator)

- Üçlü operatör, koşullu ifadeleri kısa ve okunaklı bir şekilde yazmamıza olanak tanır. Koşul doğruysa bir değer, değilse başka bir değer kullanılır.
- Üçlü operatörün temel yapısı şu şekildedir:

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

- Örnek: Bir sayının pozitif veya negatif olduğunu belirlemek için üçlü operatör kullanımı.

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

İç İçe Üçlü Operatörler

- Üçlü operatörler iç içe kullanılabilir. Bu, daha karmaşık koşulların kısa bir şekilde yazılmasına olanak tanır.
- Örnek: Bir sayının sıfır, pozitif veya negatif olduğunu belirlemek için iç içe üçlü operatörler kullanımı.

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

"for" Döngüsü

- "for" döngüsü, belirli bir işlemi veya koşulu belirli bir aralıkta tekrarlamak için kullanılır.
- Başlangıç değeri, koşul ve artırma/azaltma işlemi döngünün nasıl çalışacağını kontrol eder.
- Döngünün sonsuz bir şekilde çalışmasını önlemek için koşulu dikkatli bir şekilde belirlemek önemlidir.
- "for" döngüsünün temel yapısı şu şekildedir:

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

"while" Döngüsü

- "while" döngüsü, belirli bir koşulu sağladığı sürece bir işlemi tekrarlayan bir döngü türüdür.
- Koşulun nasıl kontrol edileceği, döngünün çalışma süresini belirler.
- Döngünün kaç kez çalışacağı önceden bilinmez.
- "while" döngüsünün temel yapısı şu şekildedir:

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

"do-while" Döngüsü

- "do-while" döngüsü, bir işlemi en az bir kez gerçekleştirmenizi sağlar ve ardından bir koşulu kontrol eder.
- Koşul sonda kontrol edildiği için kullanıcı girişi gibi durumlar için uygundur.
- "do-while" döngüsünün temel yapısı şu şekildedir:

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

"break" İfadesi

- "break" ifadesi, bir döngüden veya "switch" ifadesinden çıkmak için kullanılır.
- **Örnek**: Bir döngü içinde belirli bir koşulu kontrol ederek döngüyü sonlandırmak için "break" kullanımı.

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

"break" İfadesi

- Örnek: "switch" ifadesinde belirli bir durumu işledikten sonra "break" kullanarak "switch" bloğunu terk etmek.

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

"continue" İfadesi

- "continue" ifadesi, bir döngünün içinde belirli bir koşulu sağladığınızda, döngünün geri kalan kısmını atlayarak döngüyü devam ettirmenizi sağlar.
- Örnek: Bir döngü içinde belirli bir koşulu sağlayan sayıları atlayarak sadece belirli sayıları yazdırmak için "continue" kullanımı.

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

SON