Java, Sun Microsystems tarafından 1995 yılında geliştirilen, nesne yönelimli, platform bağımsız ve yüksek performanslı bir programlama dilidir. Günümüzde Oracle tarafından desteklenmektedir. Java, güvenliği, taşınabilirliği ve geniş kütüphane desteği sayesinde web uygulamaları, mobil uygulamalar, masaüstü programları ve büyük ölçekli sistemler geliştirmek için yaygın olarak kullanılır.

**🔹 Java’nın Temel Özellikleri**

**1️⃣ Platform Bağımsızlık**

Java, "Write Once, Run Anywhere" (Bir kere yaz, her yerde çalıştır) felsefesine sahiptir. Yazılan kod, **Java Virtual Machine (JVM)** sayesinde farklı işletim sistemlerinde çalıştırılabilir.

🔹 **Derleme süreci:**

* Java kodu **.java** uzantılı dosyada yazılır.
* javac derleyicisi tarafından **bytecode**'a (**.class** dosyası) dönüştürülür.
* JVM bu bytecode'u işletim sistemine uygun hale getirerek çalıştırır.

**2️⃣ Nesne Yönelimli Programlama (OOP)**

Java, **OOP (Object-Oriented Programming)** prensiplerine dayanır:  
✔ **Encapsulation (Kapsülleme)**  
✔ **Inheritance (Miras Alma)**  
✔ **Polymorphism (Çok Biçimlilik)**  
✔ **Abstraction (Soyutlama)**

Java'da her şey bir nesnedir ve veri ile davranışları birlikte barındırır.

**3️⃣ Otomatik Bellek Yönetimi (Garbage Collection)**

Java, **Garbage Collector (Çöp Toplayıcı)** mekanizması sayesinde kullanılmayan nesneleri otomatik olarak temizler, bellek yönetimini geliştirir.

**4️⃣ Güvenlik (Security)**

Java, bellek erişimini sınırlandıran ve kötü amaçlı yazılımlara karşı koruma sağlayan güvenlik mekanizmaları sunar. Örneğin, Java’da **Pointer (işaretçi)** kullanılmaz, böylece bellek taşması gibi güvenlik açıkları engellenir.

**5️⃣ Çoklu İş Parçacığı (Multithreading)**

Java, **Thread** yapısı ile aynı anda birden fazla işlemi yürütebilir, böylece uygulamaların performansını artırır.

Java’da **yazım (syntax) ve isimlendirme kuralları**, kodun okunabilirliği, bakım kolaylığı ve hata oranını düşürmek için oldukça önemlidir. Java, belirli standartlara ve en iyi uygulamalara sahiptir. İşte detaylı bir açıklama:

**📌 1. Genel Yazım Kuralları (Syntax Rules)**

Java’da kod yazarken **uyulması gereken temel yazım kuralları** şunlardır:

**🔹 a) Java Büyük/Küçük Harfe Duyarlıdır**

Java, büyük ve küçük harf arasında ayrım yapar. Yani Sayi ve sayi farklı değişkenlerdir.  
📌 **Yanlış Kullanım:**

int sayi = 5;

System.out.println(Sayi); // HATA: 'Sayi' tanımlı değil!

📌 **Doğru Kullanım:**

int sayi = 5;

System.out.println(sayi); // Çıktı: 5

**🔹 b) Her Java Programı Bir Sınıf İçinde Olmalıdır**

Java’da tüm kodlar bir sınıfın içinde bulunmalıdır.  
📌 **Doğru Kullanım:**

public class MerhabaDunya {

public static void main(String[] args) {

System.out.println("Merhaba, Dünya!");

}

}

**🔹 c) main Metodu Zorunludur**

Java programlarının başlangıç noktası main metodudur.

public static void main(String[] args) {

// Programın başlangıç noktası burasıdır.

}

**🔹 d) Kod Satırları Noktalı Virgül (;) ile Biter**

Noktalı virgül (;), bir ifadenin bittiğini belirtir.

int yas = 25;

System.out.println(yas); // Noktalı virgül kullanımı zorunludur.

**🔹 e) Süslü Parantez {} Kullanımı**

Kod blokları {} içine yazılır.

if (yas > 18) {

System.out.println("Ehliyet alabilirsiniz.");

}

**📌 2. İsimlendirme Kuralları (Naming Conventions)**

Java’da **değişken, sınıf, metot, sabitler** gibi bileşenleri adlandırırken belli kurallara uyulmalıdır.

**✅ a) Sınıf (Class) İsimlendirme Kuralları**

* Sınıf isimleri **büyük harfle başlamalıdır (PascalCase).**
* Kelimeler bitişik yazılmalı ve her kelimenin baş harfi büyük olmalıdır.

📌 **Doğru Kullanım:**

public class OgrenciBilgileri { }

public class ArabaModeli { }

📌 **Yanlış Kullanım:**

public class ogrenciBilgileri { } // Küçük harfle başlamamalı

public class araba\_modeli { } // Alt çizgi kullanılmamalı

**✅ b) Değişken (Variable) İsimlendirme Kuralları**

* **Değişkenler küçük harfle başlamalıdır (camelCase).**
* Eğer birden fazla kelime içeriyorsa, ikinci kelime ve sonrası büyük harfle başlamalıdır.
* **Türü hakkında bilgi veren adlar seçilmelidir.**
* **Özel karakterler kullanılmamalıdır.**

📌 **Doğru Kullanım:**

int ogrenciSayisi = 25;

double urunFiyati = 99.99;

boolean kullaniciAktif = true;

📌 **Yanlış Kullanım:**

int OgrenciSayisi = 25; // Büyük harfle başlamamalı

double Urun\_Fiyati = 99.99; // Alt çizgi kullanılmamalı

boolean 2kullanici = true; // Rakam ile başlamamalı

**✅ c) Metot (Fonksiyon) İsimlendirme Kuralları**

* **Metot isimleri küçük harfle başlamalıdır (camelCase).**
* **Fiil (eylem) içeren isimler kullanılmalıdır.**
* **Kelimeler bitişik yazılmalı ve ikinci kelimenin ilk harfi büyük olmalıdır.**

📌 **Doğru Kullanım:**

public void notHesapla() { }

public int kullaniciSayisiGetir() { return 100; }

📌 **Yanlış Kullanım:**

public void Nothesapla() { } // Büyük harfle başlamamalı

public void kullanici\_sayisi\_getir() { } // Alt çizgi kullanılmamalı

**✅ d) Sabitler (Constants) İsimlendirme Kuralları**

* **Tüm harfler büyük olmalıdır (UPPER\_CASE).**
* **Kelimeler \_ (alt çizgi) ile ayrılmalıdır.**
* final anahtar kelimesi ile sabit olarak tanımlanır.

📌 **Doğru Kullanım:**

public static final double PI = 3.14159;

public static final int MAKS\_HIZ = 180;

📌 **Yanlış Kullanım:**

public static final int MaksHiz = 180; // PascalCase kullanılmamalı

public static final int maks\_hiz = 180; // Küçük harfli olmamalı

**✅ e) Paket (Package) İsimlendirme Kuralları**

* **Paket isimleri tamamen küçük harfle yazılmalıdır.**
* **Genellikle ters domain adı kullanılır.**

📌 **Doğru Kullanım:**

package com.chefai.uygulama;

package org.kutuphane.sistem;

📌 **Yanlış Kullanım:**

package OgrenciSistemi; // Büyük harf kullanılmamalı

package Kutuphane\_Sistemi; // Alt çizgi kullanılmamalı

**✅ f) JavaDosyaAdı.java Formatı**

* **Dosya adı, içinde tanımlı olan sınıf ile aynı olmalıdır.**
* **İlk harf büyük olmalıdır.**

📌 **Doğru Kullanım:**

public class KullaniciBilgileri { } // Dosya adı: KullaniciBilgileri.java

📌 **Yanlış Kullanım:**

public class kullaniciBilgileri { } // Dosya adı büyük harf ile başlamalıdır

**📌 3. Yorum Satırları Kullanımı**

Kodun okunabilir olması için yorum satırları eklenmelidir.

📌 **Tek Satır Yorum:**

// Bu bir yorum satırıdır

📌 **Çok Satırlı Yorum:**

/\*

Bu birden fazla

satır içeren yorumdur.

\*/

📌 **Javadoc Yorumları:**

/\*\*

\* Bu metod, öğrencinin notunu hesaplar.

\* @param not1 Birinci not

\* @param not2 İkinci not

\* @return Ortalama

\*/

public double notHesapla(int not1, int not2) {

return (not1 + not2) / 2.0;

}

**📌 Özet:**

✔ **Sınıf Adları:** Büyük harfle başlar → PascalCase  
✔ **Değişken ve Metotlar:** Küçük harfle başlar → camelCase  
✔ **Sabitler:** Büyük harfle yazılır → UPPER\_CASE  
✔ **Paketler:** Küçük harf ve ters domain → com.ornek.proje  
✔ **Yorum Satırları:** Kodun açıklayıcı olmasını sağlar

Java'da **doğru isimlendirme ve yazım kurallarına uymak**, kodun **temiz, okunabilir ve yönetilebilir** olmasını sağlar. 🚀

**Java'da Değişken İsimlendirme Kuralları ve En İyi Uygulamalar**

Java'da **değişken isimlendirme** kuralları, kodun okunabilirliği ve sürdürülebilirliği açısından önemlidir. Değişken isimleri, anlamlı ve açık olmalı, kodun işlevini net bir şekilde yansıtmalıdır.

**📌 1. Java'da Değişken Nedir?**

Değişken (**variable**), programın çalışması sırasında **değer saklamak için kullanılan bellek alanıdır**. Java'da her değişken bir **veri tipine** sahiptir ve belirli bir isme göre çağrılır.

📌 **Örnek:**

int yas = 25; // 'yas' adında bir değişken tanımlandı ve 25 değeri atandı.

**📌 2. Değişken Tanımlama Kuralları**

1️⃣ **Değişken isimleri harf (a-z, A-Z), rakam (0-9) ve \_ (alt çizgi) içerebilir, ancak rakamla başlayamaz.**  
📌 **Doğru Kullanım:**

int yas = 20;

double urunFiyati = 99.99;

📌 **Yanlış Kullanım:**

int 1ogrenci = 30; // HATA: Değişken rakam ile başlayamaz!

double urun-fiyat = 99.99; // HATA: '-' karakteri kullanılamaz!

2️⃣ **Boşluk ve özel karakterler (@, #, $, %, &, \*) kullanılamaz.**  
📌 **Yanlış Kullanım:**

int ogrenci sayisi = 25; // HATA: Boşluk içeremez!

double fiyat#indirimli = 50.0; // HATA: '#' karakteri kullanılamaz!

3️⃣ **Java büyük/küçük harf duyarlıdır (Case Sensitive).**  
📌 **Örnek:**

int yas = 25;

int YAS = 30; // Farklı değişkenlerdir!

4️⃣ **Java'da ayrılmış kelimeler (reserved words) değişken ismi olarak kullanılamaz.**  
📌 **Yanlış Kullanım:**

int class = 10; // HATA: 'class' Java'da ayrılmış bir kelimedir!

boolean static = true; // HATA: 'static' kullanılamaz!

5️⃣ **Değişken isimleri anlamlı olmalı ve kodun amacını yansıtmalıdır.**  
📌 **Yanlış Kullanım:**

int a = 100; // Anlamsız bir değişken ismi!

double x = 55.5; // 'x' yerine daha açıklayıcı bir isim kullanılmalı!

📌 **Doğru Kullanım:**

int ogrenciSayisi = 100;

double indirimOrani = 55.5;

**📌 3. Java'da Değişken İsimlendirme Konvansiyonları**

**🔹 Camel Case Kullanımı (Önerilen)**

* Değişken isimleri **küçük harfle başlar** ve sonraki kelimelerin ilk harfi büyük olur.
* **camelCase** kuralı kullanılır.

📌 **Örnek:**

int ogrenciSayisi = 30;

double urunFiyati = 149.99;

boolean kullaniciAktif = true;

**🔹 Büyük Harf ve Alt Çizgi Kullanımı (Sabitler İçin)**

* **Sabitler (constants) tamamen büyük harfle yazılır** ve kelimeler \_ ile ayrılır.

📌 **Örnek:**

public static final double PI = 3.14159;

public static final int MAKS\_HIZ = 120;

**🔹 Pascal Case Kullanımı (Sınıf İsimleri İçin)**

* **Sınıf isimleri büyük harfle başlar** ve her kelimenin ilk harfi büyük olur.  
  📌 **Örnek:**

public class OgrenciBilgileri { }

**📌 4. Farklı Değişken Türleri İçin İsimlendirme Örnekleri**

**🔹 Tamsayı Değişkenleri (int, long, byte, short)**

📌 **Doğru Kullanım:**

int yas = 25;

long telefonNumarasi = 5551234567L;

📌 **Yanlış Kullanım:**

int YAS = 25; // Büyük harfle değişken ismi yazılmamalı

long Telefon\_Numarasi = 5551234567L; // Alt çizgi kullanılmamalı

**🔹 Ondalıklı Sayılar (float, double)**

📌 **Doğru Kullanım:**

double urunFiyati = 49.99;

float indirimOrani = 10.5f;

📌 **Yanlış Kullanım:**

float indirim\_orani = 10.5f; // Alt çizgi kullanılmamalı

**🔹 Boolean Değişkenler (boolean)**

* **Boolean değişkenler genellikle is, has, can, should gibi kelimelerle başlar.**

📌 **Doğru Kullanım:**

boolean isAktif = true;

boolean hasPermission = false;

📌 **Yanlış Kullanım:**

boolean aktif = true; // Anlamı belirsiz

boolean KullaniciAktifMi = true; // PascalCase kullanılmamalı

**🔹 Metin Değişkenleri (String, char)**

📌 **Doğru Kullanım:**

String kullaniciAdi = "Emre";

char ilkHarf = 'E';

📌 **Yanlış Kullanım:**

String KullaniciAdi = "Emre"; // Büyük harfle başlamamalı

char IlkHarf = 'E'; // PascalCase kullanılmamalı

**📌 5. En İyi Uygulamalar (Best Practices)**

✔ **1. Anlamlı ve açıklayıcı isimler kullanın:**

int x = 25; // Kötü

int ogrenciSayisi = 25; // İyi

✔ **2. İngilizce kelimeler tercih edin:**

int toplamUcret = 500; // Kötü

int totalPrice = 500; // İyi

✔ **3. Küçük harf ile başlayın ve camelCase kullanın:**

double OgrenciNotu = 85.5; // Kötü

double ogrenciNotu = 85.5; // İyi

✔ **4. Boolean değişkenler için is, has, can, should kullanın:**

boolean adminMi = true; // Kötü

boolean isAdmin = true; // İyi

✔ **5. Değişken adında birim bilgisi ekleyin:**

int sure = 60; // Kötü

int sureSaniye = 60; // İyi

✔ **6. Sabit değişkenler için UPPER\_CASE kullanın:**

public static final double pi = 3.14; // Kötü

public static final double PI = 3.14; // İyi

**📌 Özet:**

* **Küçük harfle başlayın, camelCase kullanın.**
* **Anlamlı ve açıklayıcı isimler seçin.**
* **Boolean değişkenler için is, has, can gibi ön ekler kullanın.**
* **Sabitler için UPPER\_CASE kullanın.**
* **Java'nın ayrılmış kelimelerini değişken ismi olarak kullanmayın.**

Bu kurallara uyarak **temiz, okunabilir ve anlaşılır kodlar** yazabilirsiniz! 🚀

**Java'da Değişkenler ve Veri Tipleri**

Java, **statik olarak türlendirilmiş (statically-typed)** bir programlama dili olduğu için her değişkenin bir **veri tipine (data type)** sahip olması gerekir. Veri tipleri, değişkenlerin bellekte ne kadar yer kaplayacağını ve hangi türde veriler saklayabileceğini belirler.

Bu konu anlatımında **değişkenler**, **veri tipleri** ve **Java'daki ilkel (primitive) ve ilkel olmayan (non-primitive) veri tipleri** hakkında detaylı bilgi vereceğiz.

**📌 1. Java'da Değişkenler (Variables)**

Değişken, program çalıştığı sürece **değer saklayan bir bellek alanıdır**. Java'da bir değişkenin **veri tipi, ismi, değeri ve bellekte bir adresi** vardır.

**🔹 Değişken Tanımlama Kuralları**

Bir değişken şu şekilde tanımlanır:

<veri\_tipi> <değişken\_adı> = <değer>;

📌 **Örnek:**

int yas = 25; // int veri tipinde yas isminde bir değişken tanımlandı ve 25 değeri atandı.

📌 **Değişkenler ile İlgili Kurallar:**

1. Değişken isimleri **harf, rakam, \_ (alt çizgi) veya $ işareti içerebilir**, ancak **rakamla başlayamaz.**
2. Java **büyük-küçük harf duyarlıdır (case-sensitive)**, yani yas ve YAS farklı değişkenlerdir.
3. **Ayrılmış kelimeler (reserved words)** değişken ismi olarak kullanılamaz (örneğin, int class = 10; hatalıdır).
4. **Değişken ismi anlamlı olmalıdır**, x, y gibi anlamsız isimler yerine ogrenciSayisi, urunFiyati gibi açıklayıcı isimler kullanılır.
5. Boolean değişkenler için is, has, can gibi önekler tercih edilmelidir (isAvailable, hasPermission gibi).

**📌 2. Java'da Veri Tipleri (Data Types)**

Java'da veri tipleri **ilkel (primitive)** ve **ilkel olmayan (non-primitive)** olmak üzere ikiye ayrılır:

| **Veri Tipi** | **Açıklama** | **Bellek Kullanımı** |
| --- | --- | --- |
| **İlkel (Primitive) Veri Tipleri** | Basit veri tipleridir (int, double, boolean vb.) | **Sabit bellek kullanır** |
| **İlkel Olmayan (Non-Primitive) Veri Tipleri** | Nesne tabanlıdır (String, Array, Class vb.) | **Dinamik bellek kullanır** |

**📌 3. İlkel (Primitive) Veri Tipleri**

Java'da toplam **8 ilkel veri tipi** vardır.

**🔹 3.1. Tam Sayı Veri Tipleri**

Tam sayı veri tipleri **byte, short, int ve long** şeklinde sıralanır.

| **Veri Tipi** | **Bellek Kullanımı** | **Değer Aralığı** |
| --- | --- | --- |
| byte | 1 byte (8 bit) | -128 ile 127 |
| short | 2 byte (16 bit) | -32,768 ile 32,767 |
| int | 4 byte (32 bit) | -2,147,483,648 ile 2,147,483,647 |
| long | 8 byte (64 bit) | -9,223,372,036,854,775,808 ile 9,223,372,036,854,775,807 |

📌 **Örnek Kullanım:**

byte yas = 25;

short kisaSayi = 1500;

int ogrenciSayisi = 100000;

long evreninYasi = 13800000000L; // 'L' eklemek zorunludur

**🔹 3.2. Ondalıklı Sayılar**

Ondalıklı sayılar için **float** ve **double** veri tipleri kullanılır.

| **Veri Tipi** | **Bellek Kullanımı** | **Hassasiyet** |
| --- | --- | --- |
| float | 4 byte (32 bit) | 7 basamak |
| double | 8 byte (64 bit) | 16 basamak |

📌 **Örnek Kullanım:**

float oran = 3.14f; // 'f' eklemek zorunludur

double hassasSayi = 3.141592653589793;

**🔹 3.3. Karakter Veri Tipi (char)**

char veri tipi, **tek bir Unicode karakterini** saklar.

| **Veri Tipi** | **Bellek Kullanımı** | **Değer Aralığı** |
| --- | --- | --- |
| char | 2 byte (16 bit) | 0 - 65,535 (Unicode) |

📌 **Örnek Kullanım:**

char harf = 'A';

char sembol = '@';

**🔹 3.4. Mantıksal Veri Tipi (boolean)**

boolean veri tipi yalnızca **true veya false** değerlerini saklar.

| **Veri Tipi** | **Bellek Kullanımı** | **Olası Değerler** |
| --- | --- | --- |
| boolean | 1 bit | true, false |

📌 **Örnek Kullanım:**

boolean isStudent = true;

boolean hasPermission = false;

**📌 4. İlkel Olmayan (Non-Primitive) Veri Tipleri**

İlkel olmayan veri tipleri **referans (reference) veri tipleri** olarak da adlandırılır.

**🔹 4.1. String Veri Tipi**

String, metinsel verileri saklamak için kullanılan bir sınıftır.

📌 **Örnek Kullanım:**

String isim = "Emre";

String mesaj = "Merhaba, Java!";

**🔹 4.2. Dizi (Array)**

Diziler aynı türden birden fazla değeri saklamak için kullanılır.

📌 **Örnek Kullanım:**

int[] sayilar = {1, 2, 3, 4, 5};

String[] isimler = {"Ali", "Veli", "Ayşe"};

**🔹 4.3. Sınıflar ve Nesneler**

Kendi veri tiplerinizi oluşturmak için **sınıflar (class)** kullanabilirsiniz.

📌 **Örnek Kullanım:**

class Ogrenci {

String isim;

int yas;

}

Ogrenci ogr1 = new Ogrenci();

ogr1.isim = "Mehmet";

ogr1.yas = 21;

**📌 5. Veri Tipi Dönüşümleri (Type Casting)**

Java'da veri tipleri arasında dönüşüm yapılabilir.

**📌 Küçükten büyüğe otomatik dönüşüm:**

int sayi = 10;

double yeniSayi = sayi; // int → double (otomatik dönüşüm)

**📌 Büyükten küçüğe dönüşüm (manuel dönüşüm gereklidir):**

double pi = 3.14;

int tamSayi = (int) pi; // double → int (kesme işlemi yapılır)

**📌 Sonuç**

* **Değişkenler bellek üzerinde veri saklar ve bir isimle çağrılır.**
* **Java'da 8 ilkel veri tipi bulunur.**
* **İlkel olmayan veri tipleri String, Array, Class gibi nesnelerdir.**
* **Veri tipi dönüşümleri ile bir veri başka bir türe çevrilebilir.**

Bu bilgiler sayesinde **Java'da değişkenleri ve veri tiplerini etkin şekilde kullanabilirsiniz!** 🚀

Değişkenler programlamada geçici bilgileri sakladığımız ve programcı tarafından belirlenen yapılardır. Değişkenler programlamanın temel yapısını oluşturmaktadır ve tüm programlama dillerinde kullanılan bir yapıdır. Değişkenler sayesinde program içinde yaptığımız işlemleri hafıza da tutar ve gerektiği yerlerde kullanırız. Değişkenlerin 4 özelliği bulunur ; Veri Tipi, İsim, Değer ve Adres. Değişkenlerde Veri Tipi, adından da anlaşılacağı üzere verinin saklanacağı türünü belirtmektedir bunlar sayılar, sözcükler ve programatik alanlar olabilir. Her değişkenin program içinde kullanıldığı ve çağrıldığı benzersiz bir ismi vardır. Bu değişkenlere birde değer atarız , işletim sistemimizde bu değişkeni hafızada tutar ve bir adres belirler. Java'da değişkenlerin veri tipleri vardır. Bu tipler Java'da varsayılan olarak tanımlı gelen ilkel tipler (primitive) de olabilir yahut yazılımcıların kendi tanımladığı ilkel olmayan tipler (non-primitive) de olabilir. İlkel tipler her zaman bir değere sahiptir. İlkel olmayan tipler ise 'null' (boş) olabilir. İlkel olmayan türlerin tümü aynı boyuta sahipken ilkel veri tiplerin boyutu alacağı veri tipine bağlıdır. Java Değişken Tanımlama <veri tipi> <değişken ismi> = veri (değer) İlk önce değişkenin veri tipini ve değişkenin ismini yazarız ve istenirse aynı matematikteki gibi "=" eşittir ile değerini atarız. int number; // number isminde, int veri tipinde bir değişken tanımlanmış Veri tipleri aynı olan değişkenleri aynı satırda tanımlayabiliriz int a, b, c; // int veri tipinde 3 tane değişken tanımlanmış Değişkeni tanımladıktan sonra, atama operatörü (=) kullanarak değişkene atayabiliriz. double pi; // ilk başta double türünde bir değişken tanımladık pi = 3.14; // Daha sonra bu değişkene bir değer atadık Eğer bir değişkene hemen değer atayacaksanız, bunu iki satırda yapmak yerine tek bir satırda halledebilirsiniz. double pi = 3.14; Aynı satırda aynı türden birden fazla değişken tanımlıyorsak : int a = 1 , b = 2; // Aynı satırda int türünde 2 farklı değişken tanımlanmış ve ikisine de değer verilmiş Değişkene verilen değer sonrasında değiştirilebilir, ama aynı isimde ikinci bir değişken oluşturulamaz ve hata alırız. Hatalı Kullanım : int a = 5; // a isminde bir değişken tanımlanmış int a = 1 ; // Bu satır hataya neden olur, a değişkeni zaten var Doğru Kullanım : boolean a = true; // a isminde bir değişken tanımlanmış ve varsayılan bir değer verilmiş a = false; // a değişkeninin değeri değiştirilmiş.

**Byte, Short, Int ve Long Veri Tipleri (Tam Sayılar) - Detaylı Konu Anlatımı**

Java'da **tam sayı veri tipleri**, **pozitif ve negatif** tam sayıları saklamak için kullanılır. Bellekte kapladıkları alan, **sınır değerleri** ve kullanım amaçları farklıdır.

Bu ders anlatımında **byte, short, int ve long** veri tiplerini detaylı bir şekilde inceleyeceğiz.

**📌 1. Byte Veri Tipi (byte)**

**📌 Genel Özellikleri:**

* **8 bit (1 byte) uzunluğundadır.**
* **Değer aralığı:** **-128 ile 127** arasındadır.
* **Bellekte küçük yer kaplar**, bu yüzden büyük ölçekli verilerde kullanışlıdır.
* **Sayısal işlemlerde fazla tercih edilmez**, çünkü çok küçük bir aralığa sahiptir.

**📌 Kullanım Senaryoları:**

* Bellek kısıtlı yerlerde (örneğin gömülü sistemler, mobil cihazlar).
* **Çok fazla veri içeren dizilerde bellek tasarrufu sağlamak için** (örneğin, bir resmin piksellerini saklarken).
* **RGB renk kodlarını saklamak için**, çünkü her renk 0-255 arasındadır ama işaretsiz olmadığı için sadece 0-127 arasını saklayabiliriz.

**📌 Örnek Kullanım:**

public class ByteExample {

public static void main(String[] args) {

byte minDeger = -128;

byte maxDeger = 127;

byte yas = 25;

System.out.println("Min Byte Değeri: " + minDeger);

System.out.println("Max Byte Değeri: " + maxDeger);

System.out.println("Yaş: " + yas);

}

}

📌 **Hata Alabileceğiniz Durum:**

byte sayi = 130; // Hata! Byte veri tipi -128 ile 127 arasında olmalı.

**📌 2. Short Veri Tipi (short)**

**📌 Genel Özellikleri:**

* **16 bit (2 byte) uzunluğundadır.**
* **Değer aralığı:** **-32,768 ile 32,767** arasındadır.
* Bellekte **byte'tan daha fazla yer kaplar ama int'ten daha küçük**.
* **Genellikle küçük ölçekli tam sayı hesaplamalarında kullanılır.**

**📌 Kullanım Senaryoları:**

* **Eski sistemlerde ve bellek optimizasyonu gerektiren durumlarda kullanılır.**
* **Sensör verilerinde** (örneğin, sıcaklık, basınç ölçümleri).
* **Küçük sayılar içeren büyük veri setlerinde bellek tasarrufu yapmak için.**

**📌 Örnek Kullanım:**

public class ShortExample {

public static void main(String[] args) {

short minDeger = -32768;

short maxDeger = 32767;

short ogrenciSayisi = 1500;

System.out.println("Min Short Değeri: " + minDeger);

System.out.println("Max Short Değeri: " + maxDeger);

System.out.println("Öğrenci Sayısı: " + ogrenciSayisi);

}

}

📌 **Hata Alabileceğiniz Durum:**

short sayi = 40000; // Hata! Short -32,768 ile 32,767 arasında olmalı.

**📌 3. Integer Veri Tipi (int)**

**📌 Genel Özellikleri:**

* **32 bit (4 byte) uzunluğundadır.**
* **Değer aralığı:** **-2,147,483,648 ile 2,147,483,647** arasındadır.
* **Java'da en çok kullanılan tam sayı veri tipidir!**
* **Bellek ve performans açısından en ideal veri tipidir.**

**📌 Kullanım Senaryoları:**

* **Genellikle sayaç, id ve indeksleme işlemlerinde kullanılır.**
* **Büyük veri kümelerindeki sayısal hesaplamalar için tercih edilir.**
* **Dosya boyutlarını saklamak için (örneğin, bir dosyanın byte cinsinden büyüklüğü).**

**📌 Örnek Kullanım:**

public class IntExample {

public static void main(String[] args) {

int minDeger = -2147483648;

int maxDeger = 2147483647;

int maas = 50000;

System.out.println("Min Int Değeri: " + minDeger);

System.out.println("Max Int Değeri: " + maxDeger);

System.out.println("Maaş: " + maas);

}

}

📌 **Hata Alabileceğiniz Durum:**

int sayi = 3000000000; // Hata! int -2,147,483,648 ile 2,147,483,647 arasında olmalı.

**📌 4. Long Veri Tipi (long)**

**📌 Genel Özellikleri:**

* **64 bit (8 byte) uzunluğundadır.**
* **Değer aralığı:** **-9,223,372,036,854,775,808 ile 9,223,372,036,854,775,807** arasındadır.
* **Çok büyük tam sayıları saklamak için kullanılır.**
* **Uzun işlemler için "L" harfi ile belirtilmelidir.**

**📌 Kullanım Senaryoları:**

* **Çok büyük sayıları saklamak için (örneğin, gezegenler arası mesafeler, nüfus sayıları, astronomik hesaplamalar).**
* **Timestamp'leri saklamak için (örneğin, bir dosyanın oluşturulma tarihi).**
* **Büyük finansal işlemleri hesaplamak için (örneğin, borsa, kripto para işlemleri).**

**📌 Örnek Kullanım:**

public class LongExample {

public static void main(String[] args) {

long minDeger = -9223372036854775808L;

long maxDeger = 9223372036854775807L;

long evreninYasi = 13800000000L; // 13.8 milyar yıl

System.out.println("Min Long Değeri: " + minDeger);

System.out.println("Max Long Değeri: " + maxDeger);

System.out.println("Evrenin Yaşı: " + evreninYasi);

}

}

📌 **Hata Alabileceğiniz Durum:**

long buyukSayi = 9223372036854775808; // Hata! Long değerler için 'L' harfi kullanılmalı.

**📌 5. Özet Tablo**

| **Veri Tipi** | **Boyut (Bit)** | **Değer Aralığı** | **Bellek Kullanımı** | **Kullanım Alanı** |
| --- | --- | --- | --- | --- |
| **byte** | 8 bit (1 byte) | -128 ile 127 | **Çok düşük** | Küçük veri kümeleri, hafıza optimizasyonu |
| **short** | 16 bit (2 byte) | -32,768 ile 32,767 | **Düşük** | Küçük ölçekli hesaplamalar |
| **int** | 32 bit (4 byte) | -2,147,483,648 ile 2,147,483,647 | **Orta (en çok kullanılan)** | Genel amaçlı tam sayılar |
| **long** | 64 bit (8 byte) | -9,223,372,036,854,775,808 ile 9,223,372,036,854,775,807 | **Yüksek** | Büyük tam sayılar, finansal işlemler |

**📌 Sonuç**

* **Küçük sayılar için byte, orta ölçekli sayılar için short, en çok tercih edilen veri tipi int, büyük sayılar için long kullanılır.**
* **Bellek optimizasyonu yaparken dikkatli seçimler yapmak önemlidir!** 🚀

**Float ve Double Veri Tipleri (Ondalıklı Sayılar) - Detaylı Konu Anlatımı**

Java'da **ondalıklı (küsuratlı) sayılar**, yani **virgülden (.) sonra basamak içeren sayılar** için **Float ve Double** veri tipleri kullanılır.

Bu veri tipleri, **tam sayı tiplerinden (byte, short, int, long) farklı olarak** ondalık değerleri saklayabilir.

**📌 1. Float Veri Tipi (float)**

**📌 Genel Özellikleri:**

* **32 bit (4 byte) uzunluğundadır.**
* **Değer aralığı:** **1.4 × 10⁻⁴⁵ ile 3.4 × 10³⁸** arasındadır.
* **Ondalıklı sayıların yaklaşık değerini tutar (hassasiyeti düşüktür).**
* **Bellekte daha az yer kaplar ve daha hızlıdır.**
* **Hassas hesaplamalarda kayıplar yaşanabilir.**
* **Java'da float değerler varsayılan olarak double kabul edilir, bu yüzden float değerlerin sonuna f veya F koyulmalıdır.**

**📌 Kullanım Senaryoları:**

* **Hassasiyetin önemli olmadığı grafik işlemlerinde**
* **Büyük ölçekli veri kümeleriyle çalışırken bellek tasarrufu yapmak için**
* **GPS koordinatları gibi yaklaşık değerlerin yeterli olduğu hesaplamalarda**

**📌 Örnek Kullanım:**

public class FloatExample {

public static void main(String[] args) {

float pi = 3.14f; // "f" eklenmezse hata alınır

float sayi = 123.456F;

float negatifSayi = -78.91f;

System.out.println("Pi değeri: " + pi);

System.out.println("Sayı: " + sayi);

System.out.println("Negatif Sayı: " + negatifSayi);

}

}

📌 **Hata Alabileceğiniz Durum:**

float sayi = 3.14; // Hata! Float değerler için "f" eklenmeli.

**📌 2. Double Veri Tipi (double)**

**📌 Genel Özellikleri:**

* **64 bit (8 byte) uzunluğundadır.**
* **Değer aralığı:** **4.9 × 10⁻³²⁴ ile 1.8 × 10³⁰⁸** arasındadır.
* **Float'a göre çok daha hassastır (ondalık basamak hassasiyeti daha fazladır).**
* **Java'daki tüm ondalıklı sayılar varsayılan olarak double kabul edilir.**
* **Üst düzey matematiksel işlemler için tercih edilir.**

**📌 Kullanım Senaryoları:**

* **Bilimsel hesaplamalar, istatistik, mühendislik uygulamaları**
* **Bankacılık ve finans sektöründe hassas hesaplamalar**
* **Sinüs, kosinüs gibi trigonometrik işlemler**
* **Fiziksel hesaplamalar (örneğin, gezegenler arası mesafeler, ışık hızı hesaplamaları)**

**📌 Örnek Kullanım:**

public class DoubleExample {

public static void main(String[] args) {

double pi = 3.141592653589793; // Çok hassas bir değer saklayabilir

double e = 2.718281828459045;

double bilimselSayi = 1.5e6; // 1.5 x 10^6 (1.500.000)

System.out.println("Pi değeri: " + pi);

System.out.println("Euler Sayısı: " + e);

System.out.println("Bilimsel Notasyon: " + bilimselSayi);

}

}

📌 **Hata Alabileceğiniz Durum:**

double sayi = 1.7976931348623157E309; // Hata! Double maksimum değeri aşamaz.

**📌 3. Float ve Double Arasındaki Farklar**

| **Özellik** | **Float (float)** | **Double (double)** |
| --- | --- | --- |
| **Bellek Kullanımı** | 32 bit (4 byte) | 64 bit (8 byte) |
| **Değer Aralığı** | 1.4 × 10⁻⁴⁵ ile 3.4 × 10³⁸ | 4.9 × 10⁻³²⁴ ile 1.8 × 10³⁰⁸ |
| **Hassasiyet** | ~7 ondalık basamak | ~15-16 ondalık basamak |
| **Hız** | Daha hızlı | Daha yavaş |
| **Kullanım Alanları** | Grafik işlemleri, bellek tasarrufu gereken yerler | Matematik, mühendislik, finans hesaplamaları |

**📌 4. Float ve Double Kullanım Tercihleri**

* **Eğer çok hassas hesaplamalara ihtiyacınız varsa (double kullanmalısınız).**
* **Eğer bellek önemliyse ve yüksek hassasiyet gerekmiyorsa (float tercih edilebilir).**
* **Java'da birçok matematiksel fonksiyon double veri tipiyle çalışır, bu yüzden genellikle double tercih edilir.**
* **Örneğin, Math.sqrt() veya Math.sin() gibi fonksiyonlar double döndürür.**

📌 **Örnek:**

double sonuc = Math.sqrt(25); // Sonuç 5.0 (double türünde)

System.out.println(sonuc);

**📌 5. Özet Sonuç**

* **Eğer hassas hesaplamalar yapıyorsanız ve bellek önemsizse double kullanın.**
* **Eğer bellek tasarrufu yapmak istiyorsanız ve aşırı hassasiyet gerekmiyorsa float kullanın.**
* **Varsayılan olarak Java'daki tüm ondalıklı sayılar double kabul edilir. float kullanmak istiyorsanız f veya F eklemelisiniz.**

🚀 **Sonuç olarak double veri tipi, float'tan daha güvenlidir ve daha sık kullanılır!**

**Java'da Char ve Boolean Veri Tipleri - Detaylı Konu Anlatımı**

Java'da karakterleri ve mantıksal değerleri saklamak için kullanılan iki temel veri tipi vardır:

* **char (karakter veri tipi)**
* **boolean (mantıksal veri tipi)**

Bu veri tipleri, Java'nın temel yapı taşlarından biridir ve programlamada sıkça kullanılır.

**📌 1. Char Veri Tipi (char)**

**📌 Genel Özellikleri:**

* **char değişkenleri, tek bir karakter saklamak için kullanılır.**
* **Java'daki char veri tipi, 16 bit (2 byte) uzunluğundadır.**
* **Diğer dillerde char genellikle 8 bit (1 byte) olarak tutulur ama Java'da Unicode karakter seti kullanıldığı için 16 bit'tir.**
* **Unicode karakter kümesi sayesinde, char veri tipi tüm dünya dillerindeki karakterleri destekler.**
* **ASCII karakterleri de destekler.**
* **char veri tipi, karakterleri tek tırnak (' ') içinde saklar.**
* **Sayı olarak da saklanabilir çünkü her karakterin Unicode veya ASCII karşılığı vardır.**

**📌 Kullanım Senaryoları:**

* **Tek bir harf veya sembol saklamak için**
* **Klavye girdileri ve metin işleme işlemleri için**
* **ASCII ve Unicode karakterleriyle çalışmak için**

**📌 Örnek Kullanım:**

public class CharExample {

public static void main(String[] args) {

char letter = 'A'; // Harf saklama

char numberChar = '7'; // Sayı karakteri

char symbol = '$'; // Sembol karakteri

char unicodeChar = '\u20AC'; // Unicode ile "€" sembolü

System.out.println("Harf: " + letter);

System.out.println("Sayı karakteri: " + numberChar);

System.out.println("Sembol: " + symbol);

System.out.println("Unicode karakter: " + unicodeChar);

}

}

📌 **Unicode Kullanımı:**

* **Java, karakterleri Unicode kullanarak saklar.**
* **Örneğin:** '\u20AC' Unicode karakter kodudur ve "€" sembolünü temsil eder.

📌 **Sayı Olarak Kullanım:**

char ch = 65; // 65, ASCII tablosunda 'A' harfine karşılık gelir

System.out.println(ch); // Çıktı: A

**📌 2. Boolean Veri Tipi (boolean)**

**📌 Genel Özellikleri:**

* **Mantıksal (doğru-yanlış) değerleri saklamak için kullanılır.**
* **Sadece iki değer alabilir:** true (doğru) veya false (yanlış).
* **Java'da boolean değişkenleri true veya false değerine sahiptir, 0 veya 1 gibi sayısal değerler kullanılmaz!**
* **Koşul ifadelerinde, döngülerde ve mantıksal işlemlerde kullanılır.**
* **Boolean değerler genellikle if, while, for gibi kontrol yapılarında kullanılır.**

**📌 Kullanım Senaryoları:**

* **Koşul ifadelerinde (if, while, for gibi)**
* **Mantıksal işlemlerde (&&, ||, ! gibi operatörlerle)**
* **Bayrak (flag) değişkeni olarak**

**📌 Örnek Kullanım:**

public class BooleanExample {

public static void main(String[] args) {

boolean isJavaFun = true;

boolean isFishTasty = false;

System.out.println("Java eğlenceli mi? " + isJavaFun);

System.out.println("Balık lezzetli mi? " + isFishTasty);

}

}

📌 **Koşul İfadelerinde Kullanım:**

int age = 20;

boolean isAdult = age >= 18; // Koşul sonucu true olacak

if (isAdult) {

System.out.println("Yetişkinsiniz.");

} else {

System.out.println("Henüz yetişkin değilsiniz.");

}

📌 **Mantıksal Operatörlerle Kullanım:**

boolean a = true;

boolean b = false;

System.out.println(a && b); // false (ve işlemi)

System.out.println(a || b); // true (veya işlemi)

System.out.println(!a); // false (değil işlemi)

📌 **Döngülerde Kullanım:**

boolean isRunning = true;

int count = 0;

while (isRunning) {

System.out.println("Program Çalışıyor...");

count++;

if (count == 5) {

isRunning = false; // Döngüden çıkmak için değişkeni false yapıyoruz

}

}

**📌 3. Char ve Boolean Karşılaştırması**

| **Özellik** | **Char (char)** | **Boolean (boolean)** |
| --- | --- | --- |
| **Veri Saklama** | Tek bir karakter (harf, rakam, sembol) saklar | true veya false saklar |
| **Bellek Kullanımı** | 16 bit (2 byte) | 1 bit (ama genellikle 1 byte olarak saklanır) |
| **Kullanım Alanları** | Metin işleme, karakter tanımlama | Koşul ifadeleri, mantıksal işlemler |
| **Varsayılan Değer** | '\u0000' (boş karakter) | false |

**📌 4. Char ve Boolean Kullanım Tercihleri**

* **Tek bir harf, sayı veya sembol saklayacaksanız char kullanın.**
* **Mantıksal koşulları kontrol etmek için boolean kullanın.**
* **char, Unicode desteği sayesinde tüm dillerde kullanılabilir.**
* **boolean sadece true veya false değerini alabilir, bu yüzden bellekte çok az yer kaplar.**

🚀 **Sonuç olarak:**

* **Metin işlemlerinde char kullanılır.**
* **Mantıksal karar mekanizmalarında boolean kullanılır.**

**Java'da String Sınıfı - Detaylı Konu Anlatımı**

Java'da **String**, metinsel verileri saklamak ve işlemek için kullanılan özel bir sınıftır. **java.lang.String** sınıfının bir parçasıdır ve Java'da metinlerle çalışırken en yaygın kullanılan yapıdır.

**📌 1. String Nedir?**

* **Birden fazla karakteri saklayan özel bir veri tipidir.**
* **char veri tipinin birleşimiyle oluşur.**
* **Nesne yönelimli programlamada String bir nesne olarak tanımlanır.**
* **String değişkenleri "çift tırnak" içinde yazılır.**
* **Değiştirilemezdir (Immutable)** – Bir kere oluşturulduğunda içeriği değiştirilemez.

**📌 String Tanımlama Örnekleri:**

public class StringExample {

public static void main(String[] args) {

// String değişkeni tanımlama

String message = "Merhaba, Java!";

System.out.println(message);

// Boş bir String oluşturma

String emptyString = "";

// String nesnesi oluşturarak tanımlama

String newString = new String("Hello World");

System.out.println(newString);

}

}

**📌 2. String Özellikleri ve Davranışı**

**📌 a) String Immutable (Değiştirilemez) Olması**

* **Bir String nesnesi oluşturulduktan sonra, içeriği değiştirilemez.**
* **Her değişiklik yeni bir String nesnesi oluşturur.**

📌 **Immutable String Örneği:**

public class ImmutableString {

public static void main(String[] args) {

String str1 = "Java";

str1 = str1 + " Programlama"; // Yeni bir String nesnesi oluşturuluyor

System.out.println(str1); // "Java Programlama"

}

}

**Not:** str1 değişkeninin içeriği değişmiş gibi görünse de, aslında yeni bir String nesnesi oluşturulmuş ve str1 değişkenine atanmıştır.

**📌 b) String Havuzu (String Pool)**

* **Java, belleği verimli kullanmak için "String Pool" adlı özel bir bellek bölgesi kullanır.**
* **Aynı metne sahip iki String nesnesi, bellek israfını önlemek için ortak bir havuzda tutulur.**
* **Eğer "Java" gibi bir String oluşturulursa, Java önce String havuzunu kontrol eder.**
* **Eğer "Java" zaten havuzda varsa, yeni bir nesne oluşturulmaz ve mevcut nesneye referans verilir.**

📌 **String Havuzu Örneği:**

public class StringPoolExample {

public static void main(String[] args) {

String str1 = "Java"; // Havuzda tutulur

String str2 = "Java"; // Aynı referansa işaret eder

System.out.println(str1 == str2); // true, çünkü aynı nesneyi gösteriyorlar

String str3 = new String("Java"); // Heap belleğinde yeni bir nesne oluşturur

System.out.println(str1 == str3); // false, çünkü farklı nesneler

}

}

**Not:** == operatörü, String nesnelerinin bellekte aynı nesne olup olmadığını kontrol eder. Eğer içerikleri karşılaştırmak istiyorsak .equals() kullanmalıyız.

**📌 3. String Metotları**

Java'da **String sınıfı**, metinler üzerinde işlem yapmak için birçok hazır metoda sahiptir.

**📌 a) Temel Metotlar**

| **Metot** | **Açıklama** |
| --- | --- |
| length() | String'in uzunluğunu döndürür. |
| charAt(index) | Belirtilen indisteki karakteri döndürür. |
| toUpperCase() | Tüm harfleri büyük harfe çevirir. |
| toLowerCase() | Tüm harfleri küçük harfe çevirir. |
| trim() | Başındaki ve sonundaki boşlukları siler. |
| substring(start, end) | Belirtilen aralıktaki alt String'i döndürür. |
| replace(old, new) | Belirtilen karakter veya alt String'i değiştirir. |
| contains(text) | String belirli bir alt String içeriyor mu? |
| indexOf(text) | String içinde belirli bir metnin ilk indeksini döndürür. |
| equals(otherString) | İki String'in içerik açısından eşit olup olmadığını kontrol eder. |
| split(delimiter) | String'i belirli bir ayırıcıya göre böler. |

📌 **Örnek Kullanımlar:**

public class StringMethods {

public static void main(String[] args) {

String str = " Java Programlama Dili ";

System.out.println("Uzunluk: " + str.length()); // 25

System.out.println("1. karakter: " + str.charAt(0)); // ' '

System.out.println("Büyük Harf: " + str.toUpperCase()); // " JAVA PROGRAMLAMA DİLİ "

System.out.println("Küçük Harf: " + str.toLowerCase()); // " java programlama dili "

System.out.println("Boşlukları sil: " + str.trim()); // "Java Programlama Dili"

System.out.println("Alt string (5-15): " + str.substring(5, 15)); // "Programlam"

System.out.println("Replace: " + str.replace("Java", "Python")); // " Python Programlama Dili "

System.out.println("Contains 'Dili': " + str.contains("Dili")); // true

System.out.println("indexOf 'Programlama': " + str.indexOf("Programlama")); // 6

// String'i bölme

String[] words = str.trim().split(" ");

for (String word : words) {

System.out.println(word);

}

}

}

**📌 4. String Karşılaştırma (equals() ve ==)**

Java'da String karşılaştırırken dikkat edilmesi gereken nokta, **== operatörünün referansları karşılaştırması, equals() metodunun ise içeriği karşılaştırmasıdır.**

📌 **Örnek:**

public class StringComparison {

public static void main(String[] args) {

String s1 = "Java";

String s2 = "Java";

String s3 = new String("Java");

System.out.println(s1 == s2); // true (Aynı nesne)

System.out.println(s1 == s3); // false (Farklı nesneler)

System.out.println(s1.equals(s3)); // true (İçerikler aynı)

}

}

**Not:** equalsIgnoreCase() metodu büyük-küçük harf duyarlılığını göz ardı ederek karşılaştırma yapar.

**📌 5. StringBuilder ve StringBuffer ile Farkı**

* **String Immutable olduğu için her değişiklik yeni bir nesne oluşturur ve bu performans kaybına neden olabilir.**
* **Bu sorunu aşmak için StringBuilder ve StringBuffer kullanılır.**
* **StringBuilder daha hızlıdır, ama StringBuffer thread-safe'dir.**

📌 **Örnek:**

StringBuilder sb = new StringBuilder("Java");

sb.append(" Programlama");

System.out.println(sb); // "Java Programlama"

**📌 6. Sonuç**

✅ **String, Java'da metinleri saklamak ve işlemek için kullanılan en temel veri tipidir.**  
✅ **Immutable (değiştirilemez) olduğu için dikkatli kullanılmalıdır.**  
✅ **String havuzu (String Pool) sayesinde belleği verimli kullanır.**  
✅ **Metin işlemleri için birçok hazır metot sağlar.**  
✅ **Performans için StringBuilder veya StringBuffer kullanılabilir.**

**Java'da Aritmetik Operatörler**

Java'da **aritmetik operatörler**, sayılar üzerinde matematiksel işlemler gerçekleştirmek için kullanılır. Aşağıda, Java'daki temel aritmetik operatörler ve kullanım örnekleri detaylı bir şekilde açıklanmıştır.

**📌 1. Aritmetik Operatörler Tablosu**

| **Operatör** | **Açıklama** | **Örnek Kullanım** | **Sonuç** |
| --- | --- | --- | --- |
| + | Toplama | 5 + 3 | 8 |
| - | Çıkarma | 10 - 4 | 6 |
| \* | Çarpma | 6 \* 7 | 42 |
| / | Bölme | 10 / 2 | 5 |
| % | Mod (Kalan) | 10 % 3 | 1 |

**📌 2. Aritmetik Operatör Kullanımı**

📌 **Örnek: Aritmetik Operatörlerin Kullanımı**

public class ArithmeticOperators {

public static void main(String[] args) {

int a = 15;

int b = 4;

System.out.println("Toplama: " + (a + b)); // 15 + 4 = 19

System.out.println("Çıkarma: " + (a - b)); // 15 - 4 = 11

System.out.println("Çarpma: " + (a \* b)); // 15 \* 4 = 60

System.out.println("Bölme: " + (a / b)); // 15 / 4 = 3 (Tam sayı bölme)

System.out.println("Mod (Kalan): " + (a % b)); // 15 % 4 = 3

}

}

📌 **Not:**

* int tipinde iki sayı bölündüğünde sonuç yine int olur. Örneğin, 15 / 4 işlemi 3.75 yerine 3 sonucunu verir.
* Eğer ondalıklı bir sonuç almak isterseniz **double veya float** veri tipi kullanmalısınız.

📌 **Örnek: Ondalıklı Bölme İşlemi**

public class FloatingPointDivision {

public static void main(String[] args) {

double x = 15;

double y = 4;

System.out.println("Ondalıklı Bölme: " + (x / y)); // 15.0 / 4.0 = 3.75

}

}

**📌 3. Mod (Kalan) Operatörü %**

* Bir sayının başka bir sayıya **bölümünden kalanını** bulmak için kullanılır.
* Genellikle **tek/çift sayı kontrolü, belirli bir aralıkta sayı üretme ve döngü işlemlerinde** kullanılır.

📌 **Örnek: Tek / Çift Sayı Kontrolü**

public class EvenOddCheck {

public static void main(String[] args) {

int num = 7;

if (num % 2 == 0) {

System.out.println(num + " çift bir sayıdır.");

} else {

System.out.println(num + " tek bir sayıdır.");

}

}

}

**📌 4. Aritmetik Operatörler ve Atama Operatörleri**

Java'da **aritmetik işlemleri** daha kısa yazmak için **bileşik atama operatörleri** kullanılabilir.

| **Operatör** | **Açıklama** | **Uzun Kullanım** | **Kısa Kullanım** |
| --- | --- | --- | --- |
| += | Toplama ve atama | x = x + 5; | x += 5; |
| -= | Çıkarma ve atama | x = x - 3; | x -= 3; |
| \*= | Çarpma ve atama | x = x \* 2; | x \*= 2; |
| /= | Bölme ve atama | x = x / 4; | x /= 4; |
| %= | Mod ve atama | x = x % 2; | x %= 2; |

📌 **Örnek: Bileşik Atama Operatörleri Kullanımı**

public class CompoundOperators {

public static void main(String[] args) {

int num = 10;

num += 5; // num = num + 5; → 10 + 5 = 15

System.out.println("Toplama: " + num);

num -= 3; // num = num - 3; → 15 - 3 = 12

System.out.println("Çıkarma: " + num);

num \*= 2; // num = num \* 2; → 12 \* 2 = 24

System.out.println("Çarpma: " + num);

num /= 4; // num = num / 4; → 24 / 4 = 6

System.out.println("Bölme: " + num);

num %= 5; // num = num % 5; → 6 % 5 = 1

System.out.println("Mod: " + num);

}

}

**📌 5. Artırma ve Azaltma Operatörleri (++, --)**

* ++ **(Artırma operatörü)**: Değişkenin değerini **1 artırır**.
* -- **(Azaltma operatörü)**: Değişkenin değerini **1 azaltır**.

| **Operatör** | **Kullanım** | **Açıklama** |
| --- | --- | --- |
| x++ | int x = 5; x++; | Sonrasında artırır (**Post-Increment**) |
| ++x | int x = 5; ++x; | Öncesinde artırır (**Pre-Increment**) |
| x-- | int x = 5; x--; | Sonrasında azaltır (**Post-Decrement**) |
| --x | int x = 5; --x; | Öncesinde azaltır (**Pre-Decrement**) |

📌 **Örnek: ++ ve -- Operatörleri**

public class IncrementDecrement {

public static void main(String[] args) {

int num = 5;

System.out.println("Önceki değer: " + num); // 5

System.out.println("Post-Increment: " + (num++)); // Önce yazdır, sonra artır → 5

System.out.println("Sonraki değer: " + num); // 6

System.out.println("Pre-Increment: " + (++num)); // Önce artır, sonra yazdır → 7

System.out.println("Post-Decrement: " + (num--)); // Önce yazdır, sonra azalt → 7

System.out.println("Sonraki değer: " + num); // 6

System.out.println("Pre-Decrement: " + (--num)); // Önce azalt, sonra yazdır → 5

}

}

**📌 6. Sonuç**

✅ \**Java'daki temel aritmetik operatörler (+, -, , /, %) matematiksel işlemleri yapar.*  
✅ **Tam sayı bölme işlemlerinde ondalıklı sonuç almak için double veya float kullanılmalıdır.**  
✅ **% operatörü kalan hesaplamak için önemlidir (Tek/Çift sayı kontrolü gibi).**  
✅ **Bileşik atama operatörleri (+=, -=, \*=, vb.) işlemleri daha kısa yazmamızı sağlar.**  
✅ **++ ve -- operatörleri bir değişkenin değerini artırıp azaltmak için kullanılır.  
Java'da Atama Operatörleri**

Java'da **atama operatörleri**, değişkenlere değer atamak veya mevcut değerleri belirli bir işlemden geçirerek güncellemek için kullanılır.

**📌 1. Atama Operatörü (=)**

Atama operatörü (=), bir değişkene değer atamak için kullanılır.

📌 **Örnek: = Operatörü Kullanımı**

public class AssignmentOperators {

public static void main(String[] args) {

int a = 10; // 'a' değişkenine 10 atanıyor

System.out.println("a: " + a); // 10

double b = 5.5; // 'b' değişkenine 5.5 atanıyor

System.out.println("b: " + b); // 5.5

}

}

**📌 2. Bileşik Atama Operatörleri**

Bileşik atama operatörleri, bir değişkenin mevcut değerini değiştiren kısa yollar sağlar.

**📌 2.1 Bileşik Atama Operatörleri Tablosu**

| **Operatör** | **Açıklama** | **Uzun Kullanım** | **Kısa Kullanım** |
| --- | --- | --- | --- |
| += | Toplama ve atama | x = x + 5; | x += 5; |
| -= | Çıkarma ve atama | x = x - 3; | x -= 3; |
| \*= | Çarpma ve atama | x = x \* 2; | x \*= 2; |
| /= | Bölme ve atama | x = x / 4; | x /= 4; |
| %= | Mod (Kalan) ve atama | x = x % 2; | x %= 2; |

**📌 2.2 Bileşik Atama Operatörleri Kullanımı**

📌 **Örnek: Bileşik Atama Operatörleri**

public class CompoundAssignment {

public static void main(String[] args) {

int x = 10;

x += 5; // x = x + 5; → 10 + 5 = 15

System.out.println("x += 5: " + x);

x -= 3; // x = x - 3; → 15 - 3 = 12

System.out.println("x -= 3: " + x);

x \*= 2; // x = x \* 2; → 12 \* 2 = 24

System.out.println("x \*= 2: " + x);

x /= 4; // x = x / 4; → 24 / 4 = 6

System.out.println("x /= 4: " + x);

x %= 5; // x = x % 5; → 6 % 5 = 1

System.out.println("x %= 5: " + x);

}

}

**📌 3. Bit Düzeyinde (Bitwise) Atama Operatörleri**

Java'da **bitwise operatörler**, bit seviyesinde işlemler yapmak için kullanılır.

| **Operatör** | **Açıklama** | **Uzun Kullanım** | **Kısa Kullanım** |
| --- | --- | --- | --- |
| &= | Bitwise AND ve atama | x = x & 2; | x &= 2; |
| ` | =` | Bitwise OR ve atama | `x = x |
| ^= | Bitwise XOR ve atama | x = x ^ 2; | x ^= 2; |
| <<= | Bitwise sola kaydırma ve atama | x = x << 2; | x <<= 2; |
| >>= | Bitwise sağa kaydırma ve atama | x = x >> 2; | x >>= 2; |

📌 **Örnek: Bitwise Operatörleri Kullanımı**

public class BitwiseAssignment {

public static void main(String[] args) {

int x = 5; // 5 = 0101 (binary)

x &= 3; // x = x & 3; → 0101 & 0011 = 0001 → x = 1

System.out.println("x &= 3: " + x);

x |= 2; // x = x | 2; → 0001 | 0010 = 0011 → x = 3

System.out.println("x |= 2: " + x);

x ^= 2; // x = x ^ 2; → 0011 ^ 0010 = 0001 → x = 1

System.out.println("x ^= 2: " + x);

x <<= 2; // x = x << 2; → 0001 << 2 = 0100 → x = 4

System.out.println("x <<= 2: " + x);

x >>= 1; // x = x >> 1; → 0100 >> 1 = 0010 → x = 2

System.out.println("x >>= 1: " + x);

}

}

**📌 4. Özet**

✅ = operatörü, bir değişkene değer atamak için kullanılır.  
✅ Bileşik atama operatörleri (+=, -=, \*=, /=, %=), kısa yazımla matematiksel işlemler yapar.  
✅ &=, |=, ^=, <<=, >>= gibi operatörler, **bitwise (bit düzeyinde) işlemler yapar**.  
✅ **Bileşik operatörler**, hem kodu kısaltır hem de daha okunaklı hale getirir.

**Java'da Karşılaştırma Operatörleri**

Java'da **karşılaştırma operatörleri**, iki değeri karşılaştırarak **mantıksal (boolean) sonuç döndüren** operatörlerdir. Sonuç **true** (doğru) veya **false** (yanlış) olur. Genellikle **koşul ifadelerinde (if-else, while, for)** kullanılır.

**📌 Karşılaştırma Operatörleri Tablosu**

| **Operatör** | **Açıklama** | **Kullanım Örneği** | **Sonuç** |
| --- | --- | --- | --- |
| == | Eşittir | x == y | true veya false |
| != | Eşit değildir | x != y | true veya false |
| > | Büyüktür | x > y | true veya false |
| < | Küçüktür | x < y | true veya false |
| >= | Büyük eşittir | x >= y | true veya false |
| <= | Küçük eşittir | x <= y | true veya false |

**📌 1. == (Eşittir) Operatörü**

İki değerin **eşit olup olmadığını** kontrol eder. Eğer eşitse **true**, değilse **false** döner.

📌 **Örnek: == Kullanımı**

public class ComparisonOperators {

public static void main(String[] args) {

int a = 5, b = 5, c = 10;

System.out.println("a == b: " + (a == b)); // true

System.out.println("a == c: " + (a == c)); // false

}

}

**📌 2. != (Eşit Değildir) Operatörü**

İki değerin **eşit olmadığını** kontrol eder. Eğer **farklıysa true**, eşitse **false** döner.

📌 **Örnek: != Kullanımı**

public class NotEqualOperator {

public static void main(String[] args) {

int x = 5, y = 10;

System.out.println("x != y: " + (x != y)); // true

System.out.println("x != 5: " + (x != 5)); // false

}

}

**📌 3. > (Büyüktür) Operatörü**

Bir değerin diğerinden **büyük olup olmadığını** kontrol eder.

📌 **Örnek: > Kullanımı**

public class GreaterThanOperator {

public static void main(String[] args) {

int num1 = 20, num2 = 15;

System.out.println("num1 > num2: " + (num1 > num2)); // true

System.out.println("num2 > num1: " + (num2 > num1)); // false

}

}

**📌 4. < (Küçüktür) Operatörü**

Bir değerin diğerinden **küçük olup olmadığını** kontrol eder.

📌 **Örnek: < Kullanımı**

public class LessThanOperator {

public static void main(String[] args) {

int a = 8, b = 12;

System.out.println("a < b: " + (a < b)); // true

System.out.println("b < a: " + (b < a)); // false

}

}

**📌 5. >= (Büyük Eşittir) Operatörü**

Bir değerin diğerine **eşit veya büyük olup olmadığını** kontrol eder.

📌 **Örnek: >= Kullanımı**

public class GreaterOrEqualOperator {

public static void main(String[] args) {

int x = 10, y = 10, z = 5;

System.out.println("x >= y: " + (x >= y)); // true

System.out.println("x >= z: " + (x >= z)); // true

System.out.println("z >= x: " + (z >= x)); // false

}

}

**📌 6. <= (Küçük Eşittir) Operatörü**

Bir değerin diğerine **eşit veya küçük olup olmadığını** kontrol eder.

📌 **Örnek: <= Kullanımı**

public class LessOrEqualOperator {

public static void main(String[] args) {

int a = 5, b = 10, c = 5;

System.out.println("a <= b: " + (a <= b)); // true

System.out.println("a <= c: " + (a <= c)); // true

System.out.println("b <= a: " + (b <= a)); // false

}

}

**📌 7. Karşılaştırma Operatörlerinin Kullanım Alanları**

✅ **Koşul ifadelerinde (if-else)**  
✅ **Döngülerde (while, for, do-while)**  
✅ **Mantıksal işlemlerde (boolean kontrolleri)**

📌 **Örnek: if Koşulu ile Kullanımı**

public class ComparisonExample {

public static void main(String[] args) {

int age = 18;

if (age >= 18) {

System.out.println("Ehliyet alabilirsiniz.");

} else {

System.out.println("Ehliyet almak için yaşınız yetmiyor.");

}

}

}

🔹 **Çıktı:**

Ehliyet alabilirsiniz.

**📌 8. Özet**

✔ **==** → Eşit mi?  
✔ **!=** → Eşit değil mi?  
✔ **>** → Büyük mü?  
✔ **<** → Küçük mü?  
✔ **>=** → Büyük veya eşit mi?  
✔ **<=** → Küçük veya eşit mi?

✅ **Koşul ifadelerinde (if-else)** ve **döngülerde (while, for)** sıkça kullanılır.

**Java'da Mantıksal Operatörler**

Mantıksal operatörler, **birden fazla koşulu birleştirmek** ve **mantıksal işlemler yapmak** için kullanılır. **Boolean (true/false)** değerleri ile çalışırlar ve genellikle **koşul ifadelerinde (if-else, while, for)** kullanılırlar.

**📌 Mantıksal Operatörler Tablosu**

| **Operatör** | **Açıklama** | **Kullanım Örneği** | **Sonuç** |
| --- | --- | --- | --- |
| && | **Mantıksal VE (AND)** | (a > 5) && (b < 10) | Her iki koşul da **true** ise **true**, aksi halde **false** |
| ` |  | ` | **Mantıksal VEYA (OR)** |
| ! | **Mantıksal DEĞİL (NOT)** | !(a > 5) | Koşul **true** ise **false**, **false** ise **true** |

**📌 1. && (Mantıksal VE - AND) Operatörü**

* **İki koşulun da doğru olması gerekir.**
* **Bir tane bile false varsa sonuç false olur.**

📌 **Örnek: && Kullanımı**

public class LogicalAndOperator {

public static void main(String[] args) {

int x = 10, y = 20;

boolean sonuc1 = (x > 5) && (y < 30); // true && true → true

boolean sonuc2 = (x > 15) && (y < 30); // false && true → false

System.out.println("Sonuç 1: " + sonuc1); // true

System.out.println("Sonuç 2: " + sonuc2); // false

}

}

🔹 **Çıktı:**

Sonuç 1: true

Sonuç 2: false

**📌 2. || (Mantıksal VEYA - OR) Operatörü**

* **Koşullardan en az biri doğruysa sonuç true olur.**
* **Tüm koşullar false ise sonuç false olur.**

📌 **Örnek: || Kullanımı**

public class LogicalOrOperator {

public static void main(String[] args) {

int x = 10, y = 5;

boolean sonuc1 = (x > 5) || (y > 10); // true || false → true

boolean sonuc2 = (x < 5) || (y > 10); // false || false → false

System.out.println("Sonuç 1: " + sonuc1); // true

System.out.println("Sonuç 2: " + sonuc2); // false

}

}

🔹 **Çıktı:**

Sonuç 1: true

Sonuç 2: false

**📌 3. ! (Mantıksal DEĞİL - NOT) Operatörü**

* **Koşulun sonucunu tersine çevirir.**
* **true olanı false, false olanı true yapar.**

📌 **Örnek: ! Kullanımı**

public class LogicalNotOperator {

public static void main(String[] args) {

boolean durum = true;

System.out.println("Orijinal: " + durum); // true

System.out.println("Tersi: " + !durum); // false

}

}

🔹 **Çıktı:**

Orijinal: true

Tersi: false

**📌 4. Mantıksal Operatörlerin Kullanım Alanları**

✅ **Koşul ifadelerinde (if-else)**  
✅ **Döngülerde (while, for, do-while)**  
✅ **Karmaşık mantıksal işlemlerde**

📌 **Örnek: if ile Kullanımı**

public class LogicalExample {

public static void main(String[] args) {

int yas = 20;

boolean ehliyetVar = true;

if (yas >= 18 && ehliyetVar) {

System.out.println("Araç kullanabilirsiniz.");

} else {

System.out.println("Araç kullanamazsınız.");

}

}

}

🔹 **Çıktı:**

Araç kullanabilirsiniz.

**📌 5. Kısa Devre (Short-Circuit) Mantığı**

* **Java, && ve || operatörlerinde kısa devre mantığını kullanır.**
* **&& (AND) operatöründe ilk koşul false olursa, ikinci koşul kontrol edilmez.**
* **|| (OR) operatöründe ilk koşul true olursa, ikinci koşul kontrol edilmez.**

📌 **Örnek: Kısa Devre Kullanımı**

public class ShortCircuitExample {

public static void main(String[] args) {

int x = 5;

boolean sonuc = (x > 10) && (++x > 0); // İlk koşul false olduğu için ++x çalışmaz!

System.out.println("Sonuç: " + sonuc); // false

System.out.println("x: " + x); // 5 (x değişmedi)

}

}

🔹 **Çıktı:**

Sonuç: false

x: 5

**📌 6. Özet**

✔ **&& (VE - AND):** Her iki koşul da **true** olursa sonuç **true**, aksi halde **false**  
✔ **|| (VEYA - OR):** En az bir koşul **true** olursa sonuç **true**, aksi halde **false**  
✔ **! (DEĞİL - NOT):** Koşulun tersini döndürür (true ↔ false)

✅ **Mantıksal operatörler, koşul ifadeleri ve döngülerde yaygın olarak kullanılır.**

**Java'da Koşul (Ternary) Operatörü**

Java'da **koşul operatörü (?:)**, **if-else yapısının** daha kısa ve pratik bir alternatifi olarak kullanılır. **Ternary (üçlü) operatör** olarak da adlandırılır çünkü üç bileşenden oluşur.

**📌 Koşul Operatörünün Genel Yapısı**

şart ? doğruysa\_bu\_değer : yanlışsa\_bu\_değer;

* **şart** → Değerlendirilecek koşul (boolean sonuç üretmeli)
* **doğruysa\_bu\_değer** → Koşul true ise çalışacak ifade
* **yanlışsa\_bu\_değer** → Koşul false ise çalışacak ifade

**📌 Örnek 1: Koşul Operatörünün Kullanımı**

public class TernaryExample {

public static void main(String[] args) {

int a = 10, b = 20;

// a > b kontrolü

String sonuc = (a > b) ? "a büyüktür" : "b büyüktür veya eşittir";

System.out.println(sonuc); // Çıktı: "b büyüktür veya eşittir"

}

}

**Açıklama:**

* **a > b** kontrol ediliyor.
* a, b'den küçük olduğu için **false** dönüyor.
* **Sonuç:** "b büyüktür veya eşittir" ekrana yazdırılır.

**📌 Örnek 2: Sayının Pozitif, Negatif veya Sıfır Olduğunu Bulma**

public class CheckNumber {

public static void main(String[] args) {

int num = -5;

String result = (num > 0) ? "Pozitif" : (num < 0) ? "Negatif" : "Sıfır";

System.out.println("Sayı: " + result);

}

}

🔹 **Çıktı:**

Sayı: Negatif

**Nasıl Çalışıyor?**

1. num > 0 kontrol edilir. **False** (çünkü -5 küçüktür 0).
2. O zaman (num < 0) kontrol edilir. **True**, sonuç "Negatif" olur.

**📌 Örnek 3: En Büyük Sayıyı Bulma**

public class FindMax {

public static void main(String[] args) {

int x = 15, y = 25;

int max = (x > y) ? x : y;

System.out.println("En büyük sayı: " + max);

}

}

🔹 **Çıktı:**

En büyük sayı: 25

**📌 Örnek 4: Kullanıcıya Geçti veya Kaldı Mesajı Gösterme**

public class PassOrFail {

public static void main(String[] args) {

int not = 50;

String sonuc = (not >= 50) ? "Geçti" : "Kaldı";

System.out.println("Sonuç: " + sonuc);

}

}

🔹 **Çıktı:**

Sonuç: Geçti

**📌 Koşul Operatörü mü, if-else mi Kullanmalıyım?**

✅ **Ternary (?:) operatörü**, **basit koşullarda** tercih edilir.  
✅ **if-else**, **daha karmaşık işlemlerde** kullanılmalıdır.

📌 **Örnek: Ternary operatör ile if-else karşılaştırması**

// if-else kullanımı

if (a > b) {

max = a;

} else {

max = b;

}

// Ternary operatör ile

max = (a > b) ? a : b;

✔ **Ternary operatör**, daha kısa ve okunaklıdır.

**📌 Sonuç**

✔ **Koşul operatörü (?:)**, kısa ve hızlı karar vermek için kullanılır.  
✔ **Üç bileşenden oluşur:** (koşul) ? (true değeri) : (false değeri);  
✔ **Basit koşullarda if-else yerine kullanılabilir.**  
✔ **Daha karmaşık işlemlerde if-else tercih edilmelidir.**

**Java'da b++ ve ++b Farkı**

Java'da **b++ (post-increment)** ve **++b (pre-increment)** operatörleri, bir değişkenin değerini **1 artırmak** için kullanılır. Ancak, **artışın ne zaman gerçekleştiği** konusunda önemli bir fark vardır.

**📌 b++ (Post-Increment - Sonra Artırma)**

* **Önce mevcut değeri kullanır**, sonra artırır.
* İşlem tamamlandıktan sonra **değişkenin değeri 1 artar**.

**Örnek:**

public class PostIncrementExample {

public static void main(String[] args) {

int b = 5;

int sonuc = b++; // önce b'nin mevcut değeri atanır, sonra artırılır

System.out.println("sonuc: " + sonuc); // Çıktı: 5

System.out.println("b: " + b); // Çıktı: 6

}

}

🔹 **Açıklama:**

* b++ işlemi yapıldığında, önce b'nin **eski değeri (5)** sonuc değişkenine atanır.
* Daha sonra b, **1 artarak 6 olur**.

**📌 ++b (Pre-Increment - Önce Artırma)**

* **Önce değişkenin değerini artırır**, sonra kullanır.
* İşlem öncesinde **değişkenin değeri 1 artar** ve yeni değer atanır.

**Örnek:**

public class PreIncrementExample {

public static void main(String[] args) {

int b = 5;

int sonuc = ++b; // önce b artırılır, sonra atanır

System.out.println("sonuc: " + sonuc); // Çıktı: 6

System.out.println("b: " + b); // Çıktı: 6

}

}

🔹 **Açıklama:**

* ++b işlemi yapıldığında, **önce b artırılır (6 olur)**.
* Sonra, yeni değeri sonuc değişkenine atanır.

**📌 b++ ve ++b Karşılaştırması**

| **Operatör** | **Açıklama** | **Kullanımda Etkisi** |
| --- | --- | --- |
| **b++ (Post-Increment)** | Önce eski değeri kullanır, sonra artırır. | Değer **sonraki satırda** artar. |
| **++b (Pre-Increment)** | Önce artırır, sonra yeni değeri kullanır. | Değer **hemen artar**. |

**📌 b++ ve ++b Farkını Gösteren Örnek**

public class IncrementComparison {

public static void main(String[] args) {

int x = 10;

int y = 10;

int sonuc1 = x++; // Önce x'in eski değeri atanır, sonra artırılır

int sonuc2 = ++y; // Önce artırılır, sonra yeni değeri atanır

System.out.println("sonuc1 (x++): " + sonuc1); // Çıktı: 10

System.out.println("x (x++ sonrası): " + x); // Çıktı: 11

System.out.println("sonuc2 (++y): " + sonuc2); // Çıktı: 11

System.out.println("y (++y sonrası): " + y); // Çıktı: 11

}

}

🔹 **Çıktı:**

sonuc1 (x++): 10

x (x++ sonrası): 11

sonuc2 (++y): 11

y (++y sonrası): 11

**📌 Ne Zaman b++ veya ++b Kullanılır?**

✅ **b++ kullanılır:**

* Eski değerin işlemde kullanılmasını istiyorsanız.
* Örneğin, **döngülerde (for, while)** genellikle i++ kullanılır.

✅ **++b kullanılır:**

* Değerin **hemen artmasını** istiyorsanız.
* Hesaplamalar ve karşılaştırmalarda daha sık kullanılır.

**📌 Döngüde Kullanım Örneği**

public class LoopExample {

public static void main(String[] args) {

for (int i = 0; i < 5; i++) { // i++ post-increment

System.out.println(i);

}

}

}

🔹 **Çıktı:**

0

1

2

3

4

* i++ kullanıldığı için önce i değeri kontrol edilir, sonra artırılır.
* ++i kullanılsaydı, döngü **1’den başlayıp 5’te bitmiş olurdu**.

**📌 Sonuç**

✔ **b++ (Post-Increment):** Önce eski değer kullanılır, sonra artırılır.  
✔ **++b (Pre-Increment):** Önce artırılır, sonra yeni değer kullanılır.  
✔ **Karmaşık işlemler ve hesaplamalar için ++b daha güvenlidir.**  
✔ **Döngülerde genellikle b++ tercih edilir.**

Java'da kullanıcıdan veri almak için birkaç farklı yöntem vardır. En yaygın kullanılanları:

1. **Scanner Sınıfı**
2. **BufferedReader Sınıfı**
3. **JOptionPane (GUI için)**

Şimdi bu yöntemleri detaylıca inceleyelim.

**1. Scanner Sınıfı ile Kullanıcıdan Veri Alma**

Scanner sınıfı, **kullanıcıdan konsol aracılığıyla giriş almak** için kullanılır.  
java.util.Scanner paketinden çağrılır.

**Örnek: Kullanıcıdan String ve Sayı Alma**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in); // Scanner nesnesi oluşturduk

// Kullanıcıdan isim alma

System.out.print("Adınızı girin: ");

String isim = scanner.nextLine();

// Kullanıcıdan yaş alma

System.out.print("Yaşınızı girin: ");

int yas = scanner.nextInt();

// Kullanıcıdan maaş alma

System.out.print("Maaşınızı girin: ");

double maas = scanner.nextDouble();

// Çıktıyı ekrana yazdırma

System.out.println("Adınız: " + isim);

System.out.println("Yaşınız: " + yas);

System.out.println("Maaşınız: " + maas);

scanner.close(); // Scanner'ı kapatma

}

}

**Scanner Metotları**

| **Metot** | **Açıklama** |
| --- | --- |
| next() | Kullanıcının girdiği ilk kelimeyi alır (boşluk varsa keser). |
| nextLine() | Kullanıcının tüm satırını alır. |
| nextInt() | Kullanıcının girdiği tam sayıyı alır. |
| nextDouble() | Kullanıcının girdiği ondalıklı sayıyı alır. |
| nextBoolean() | true veya false değerlerini alır. |

**2. BufferedReader ile Kullanıcıdan Veri Alma**

BufferedReader, daha büyük veri girişleri için **daha hızlı** çalışır.  
InputStreamReader ile birlikte kullanılır.

**Örnek: Kullanıcıdan Veri Alma**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main {

public static void main(String[] args) throws IOException {

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

// Kullanıcıdan isim alma

System.out.print("Adınızı girin: ");

String isim = reader.readLine();

// Kullanıcıdan yaş alma

System.out.print("Yaşınızı girin: ");

int yas = Integer.parseInt(reader.readLine());

// Kullanıcıdan maaş alma

System.out.print("Maaşınızı girin: ");

double maas = Double.parseDouble(reader.readLine());

// Çıktıyı ekrana yazdırma

System.out.println("Adınız: " + isim);

System.out.println("Yaşınız: " + yas);

System.out.println("Maaşınız: " + maas);

}

}

**BufferedReader Avantajları**

* **Daha hızlıdır** çünkü veriyi tampon bellek kullanarak işler.
* **Büyük veriler** okuma gereksinimlerinde kullanılır.

**Dezavantajı**: readLine() metodu **her şeyi String olarak alır**, bu yüzden sayıya çevirmek için Integer.parseInt() veya Double.parseDouble() kullanmalıyız.

**3. JOptionPane ile Kullanıcıdan Veri Alma (Grafik Arayüz)**

JOptionPane, **GUI (Grafiksel Kullanıcı Arayüzü)** ile veri almak için kullanılır.  
javax.swing.JOptionPane paketinden çağrılır.

**Örnek: Kullanıcıdan Veri Alma (Popup Penceresi)**

import javax.swing.JOptionPane;

public class Main {

public static void main(String[] args) {

// Kullanıcıdan isim alma

String isim = JOptionPane.showInputDialog("Adınızı girin:");

// Kullanıcıdan yaş alma (String alınıp int'e çevriliyor)

int yas = Integer.parseInt(JOptionPane.showInputDialog("Yaşınızı girin:"));

// Kullanıcıdan maaş alma (String alınıp double'a çevriliyor)

double maas = Double.parseDouble(JOptionPane.showInputDialog("Maaşınızı girin:"));

// Mesaj kutusu ile sonucu gösterme

JOptionPane.showMessageDialog(null,

"Adınız: " + isim + "\nYaşınız: " + yas + "\nMaaşınız: " + maas);

}

}

**Avantajları**

* Konsol kullanmadan **pencere (popup) ile giriş alır**.
* Kullanıcı dostu bir arayüz sunar.

**Dezavantajı**: Daha basit projelerde gereksiz olabilir.

**Hangi Yöntemi Seçmeliyim?**

| **Yöntem** | **Kullanım Alanı** |
| --- | --- |
| **Scanner** | Küçük ve orta ölçekli projeler, kullanıcıdan veri almak için ideal. |
| **BufferedReader** | Büyük veri okuma işlemleri, dosya işlemleri için daha uygun. |
| **JOptionPane** | Grafiksel arayüzlü (GUI) programlar için önerilir. |

Bunları öğrendikten sonra, **Scanner en yaygın kullanılan** yöntemdir. Ancak ihtiyaca göre diğerlerini de tercih edebilirsin.  
Bir proje yapıyorsan **hangisi daha pratik ve hızlı olacaksa** onu kullanmalısın. 🚀

**📌 Java'da Karşılaştırma Operatörleri (Comparison Operators)**

Java'da **karşılaştırma operatörleri**, iki değeri karşılaştırmak için kullanılır ve **boolean (true/false)** sonuç döndürür. Genellikle **koşullu ifadeler (if-else)** ve **döngülerde (while, for)** kullanılır.

**🔹 Java Karşılaştırma Operatörleri ve Kullanımları**

| **Operatör** | **Anlamı** | **Kullanımı** | **Açıklama** |
| --- | --- | --- | --- |
| == | Eşittir | a == b | a ve b eşitse **true**, değilse **false** döner. |
| != | Eşit Değildir | a != b | a ve b eşit değilse **true**, eşitse **false** döner. |
| > | Büyüktür | a > b | a, b'den büyükse **true**, değilse **false** döner. |
| < | Küçüktür | a < b | a, b'den küçükse **true**, değilse **false** döner. |
| >= | Büyük Eşittir | a >= b | a, b'den büyük veya eşitse **true**, değilse **false** döner. |
| <= | Küçük Eşittir | a <= b | a, b'den küçük veya eşitse **true**, değilse **false** döner. |

**🔹 Örnek Kullanım**

public class Main {

public static void main(String[] args) {

int A = 10, B = 20, C = 10, D = 40;

System.out.println("== Operatörü:");

System.out.println(A == B); // false

System.out.println(A == C); // true

System.out.println("\n!= Operatörü:");

System.out.println(A != D); // true

System.out.println(A != C); // false

System.out.println("\n> Operatörü:");

System.out.println(D > C); // true

System.out.println("\n< Operatörü:");

System.out.println(A < D); // true

System.out.println("\n>= Operatörü:");

System.out.println(A >= C); // true

System.out.println("\n<= Operatörü:");

System.out.println(C <= B); // true

}

}

**Çıktı:**

== Operatörü:

false

true

!= Operatörü:

true

false

> Operatörü:

true

< Operatörü:

true

>= Operatörü:

true

<= Operatörü:

true

**📌 Dikkat Edilmesi Gerekenler**

**✅ 1. == ile equals() Arasındaki Fark (String Karşılaştırması)**

Java’da ==, **referansları** karşılaştırırken, .equals() **içeriği** karşılaştırır.

String str1 = "Java";

String str2 = new String("Java");

System.out.println(str1 == str2); // false (farklı nesneler)

System.out.println(str1.equals(str2)); // true (içerik aynı)

**✅ 2. char Değerlerini Karşılaştırma**

char değerleri ASCII tablosuna göre karşılaştırılır.

char x = 'A';

char y = 'B';

System.out.println(x < y); // true (çünkü 'A' ASCII'de 65, 'B' ise 66)

**🔥 Karşılaştırma Operatörleri Nerelerde Kullanılır?**

✔ **Koşullu İfadeler (if-else)**

int yas = 18;

if (yas >= 18) {

System.out.println("Ehliyet alabilirsin.");

} else {

System.out.println("Yaşın yetmiyor!");

}

✔ **Döngüler (while, for)**

int i = 0;

while (i < 5) {

System.out.println(i);

i++;

}

✔ **Ternary Operatörü (? :)**

int a = 10, b = 20;

String sonuc = (a > b) ? "A büyüktür" : "B büyüktür";

System.out.println(sonuc);

**🎯 Özet**

* **Karşılaştırma operatörleri**, iki değeri karşılaştırarak **true/false** döndürür.
* ==, !=, >, <, >=, <= gibi operatörler kullanılır.
* **String karşılaştırmalarında** equals() kullanılır.
* **Koşullu ifadelerde ve döngülerde** yaygın olarak kullanılır.

Bu konuyu tamamen öğrendiğini düşünüyor musun, yoksa daha fazla örnek ister misin? 🚀😊

**📌 Java'da Mantıksal Operatörler (Logical Operators)**

Java'da **mantıksal operatörler**, boolean (true/false) değerleri üzerinde işlem yaparak **karar verme mekanizmalarında** kullanılır.

Genellikle **if-else blokları** ve **döngülerde** koşulları birleştirmek veya tersine çevirmek için kullanılırlar.

**🔹 Java Mantıksal Operatörleri ve Kullanımları**

| **Operatör** | **Anlamı** | **Kullanımı** | **Açıklama** |
| --- | --- | --- | --- |
| && | VE (AND) | a && b | **Her iki ifade** de **true** ise sonuç **true**, aksi halde **false**. |
| ` |  | ` | VEYA (OR) |
| ! | DEĞİL (NOT) | !a | a **true** ise **false**, a **false** ise **true** yapar. |
| ? : | Ternary Operatörü | (koşul) ? doğruysa : yanlışsa | if-else yerine kullanılan **kısa bir koşul** operatörüdür. |

**🔥 1. && (AND) - VE Operatörü**

**Her iki koşul da true ise sonuç true döner.**

🔹 **Örnek:**

public class Main {

public static void main(String[] args) {

int x = 5;

System.out.println(x > 2 && x < 10); // true (5, hem 2'den büyük hem 10'dan küçük)

System.out.println(x > 2 && x > 10); // false (5, 10'dan büyük değil)

}

}

**Çıktı:**

true

false

✔ **Kullanım Alanları:**

* **İki veya daha fazla koşulu aynı anda kontrol etmek** için kullanılır.
* **Örneğin:** Kullanıcı giriş yaparken hem kullanıcı adı hem de şifre doğru mu kontrol edilebilir.

**🔥 2. || (OR) - VEYA Operatörü**

**Koşullardan en az biri true ise sonuç true döner.**

🔹 **Örnek:**

public class Main {

public static void main(String[] args) {

int x = 5;

System.out.println(x < 2 || x < 10); // true (x < 10 olduğu için)

System.out.println(x < 2 || x > 10); // false (hiçbiri sağlanmıyor)

}

}

**Çıktı:**

true

false

✔ **Kullanım Alanları:**

* **Biri bile doğruysa işlemi gerçekleştirmek için** kullanılır.
* **Örneğin:** Kullanıcı adı ya da e-posta ile giriş yapılmasını sağlamak.

**🔥 3. ! (NOT) - Değil Operatörü**

**Koşulu tersine çevirir.**

🔹 **Örnek:**

public class Main {

public static void main(String[] args) {

boolean durum = true;

System.out.println(!durum); // false (true'nun tersi)

int x = 5;

System.out.println(!(x > 2)); // false (x > 2 doğru olduğu için, ters çevrilince false oldu)

}

}

**Çıktı:**

false

false

✔ **Kullanım Alanları:**

* **Koşulları tersine çevirmek için kullanılır.**
* **Örneğin:** Bir kullanıcı oturum açtıysa, butonları gizlemek için !girisYapildi gibi bir kontrol kullanılabilir.

**🔥 4. ? : (Ternary Operatörü) - Tek Satırda if-else**

**Koşul sağlanıyorsa bir işlem yapar, sağlanmıyorsa başka bir işlem yapar.**

🔹 **Örnek:**

public class Main {

public static void main(String[] args) {

int a = 10;

int b = (a == 1) ? 20 : 30;

System.out.println("b'nin değeri: " + b); // b = 30 çünkü a == 1 false

b = (a == 10) ? 20 : 30;

System.out.println("b'nin değeri: " + b); // b = 20 çünkü a == 10 true

}

}

**Çıktı:**

b'nin değeri: 30

b'nin değeri: 20

✔ **Kullanım Alanları:**

* **Kısa if-else kontrollerinde kullanılır.**
* **Örneğin:** Kullanıcının yaşına göre yetişkin mi değil mi belirlemek.

🔹 **Örnek:**

int yas = 20;

String mesaj = (yas >= 18) ? "Ehliyet alabilir" : "Ehliyet alamaz";

System.out.println(mesaj); // Ehliyet alabilir

**📌 Mantıksal Operatörlerin Kullanım Alanları**

Mantıksal operatörler **karar yapılarında** ve **döngülerde** sıkça kullanılır.

✔ **Koşullu İfadelerde Kullanımı (if-else)**

int yas = 25;

boolean ehliyetVar = true;

if (yas >= 18 && ehliyetVar) {

System.out.println("Araba kullanabilirsin.");

} else {

System.out.println("Araba kullanamazsın.");

}

✔ **Döngülerde Kullanımı (while)**

int i = 0;

while (i < 10 && i % 2 == 0) {

System.out.println("i: " + i);

i += 2;

}

✔ **Form Kontrollerinde Kullanımı**

String kullaniciAdi = "admin";

String sifre = "1234";

if (kullaniciAdi.equals("admin") && sifre.equals("1234")) {

System.out.println("Giriş başarılı!");

} else {

System.out.println("Giriş başarısız!");

}

**🎯 Özet**

| **Operatör** | **Açıklama** | **Kullanım** |
| --- | --- | --- |
| && | **VE (AND)** – Her iki koşul da doğruysa **true** | x > 5 && x < 10 |
| ` |  | ` |
| ! | **DEĞİL (NOT)** – Koşulu tersine çevirir | !(x > 5) |
| ? : | **Ternary Operatörü** – Tek satırda if-else yapar | (x > 5) ? "Büyük" : "Küçük" |

Bu konuyu tamamen öğrendiğini düşünüyor musun, yoksa daha fazla örnek ister misin? 🚀😊

**📌 Java'da Kod Blokları (Scope)**

Java'da **kod blokları (scope)**, programın düzenli çalışmasını sağlamak ve değişkenlerin **yaşam süresini** belirlemek için kullanılır. **Kod blokları, {} süslü parantezler arasındaki alanı temsil eder** ve kodun yapısını belirleyen temel unsurlardan biridir.

**📌 Kod Blokları Nedir?**

Kod blokları, { (açılış süslü parantezi) ile başlar ve } (kapanış süslü parantezi) ile biter.

Kod blokları sayesinde:  
✔ **Kodların hiyerarşik bir düzende olması sağlanır.**  
✔ **Değişkenlerin nerede geçerli olduğu belirlenir.**  
✔ **Fonksiyonlar, döngüler, koşul ifadeleri gibi yapılar düzenlenir.**  
✔ **Kodun okunabilirliği artar.**

**🔥 1. Java'da Kod Blokları (Scope) Kullanımı**

Kod bloklarını **sınıflar**, **metotlar**, **döngüler**, **koşul ifadeleri** ve diğer yapılar içinde kullanırız.

**Örnek 1:**  
Aşağıdaki kodda, farklı kod bloklarının nasıl kullanıldığını görebiliriz.

public class JavaPatika { // Sınıf bloğu başlıyor

public static void main(String[] args) { // Main metot bloğu başlıyor

System.out.println("Burası main metota ait"); // main bloğu içinde

} // Main metot bloğu bitiyor

} // Sınıf bloğu bitiyor

**📌 Önemli Noktalar:**

✅ **JavaPatika sınıfı**: En dış kod bloğudur.  
✅ **main metodu**: Java programlarının başlangıç noktasıdır.  
✅ **System.out.println("Burası main metota ait");** satırı main bloğu içinde çalışır.

**🔥 2. Kod Bloklarında Değişkenlerin Kullanımı**

Bir değişken **tanımlandığı kod bloğu içinde geçerlidir** ve bu blok dışında **kullanılamaz**.

**📌 Örnek 2: Kod Bloğu Dışında Değişken Kullanımı**

public class ScopeTest {

public static void main(String[] args) {

int x = 10; // x değişkeni main bloğunda tanımlandı.

if (x > 5) {

int y = 20; // y değişkeni if bloğu içinde tanımlandı.

System.out.println("x: " + x); // Çalışır

System.out.println("y: " + y); // Çalışır

}

System.out.println("x: " + x); // Çalışır

// System.out.println("y: " + y); // HATA! y değişkeni if bloğu dışında erişilemez.

}

}

**Çıktı:**

x: 10

y: 20

x: 10

❌ **Hata!** y değişkenine if bloğunun dışında erişilemez çünkü **sadece if bloğunda tanımlıdır**.

**🔥 3. Kod Blokları ile İç İçe Kullanım (Nested Scope)**

Kod blokları iç içe olabilir ve **içteki blok, dışarıdaki bloktan değişkenlere erişebilir**. Ancak, dıştaki blok **içteki bloktaki değişkenleri göremez**.

**📌 Örnek 3: İç İçe Kod Blokları**

public class NestedScope {

public static void main(String[] args) {

int a = 5; // a değişkeni main bloğunda tanımlandı.

{

int b = 10; // b değişkeni bu blok içinde tanımlandı.

System.out.println("a: " + a); // Çalışır

System.out.println("b: " + b); // Çalışır

}

System.out.println("a: " + a); // Çalışır

// System.out.println("b: " + b); // HATA! b değişkeni dışarıdan erişilemez.

}

}

**Çıktı:**

a: 5

b: 10

a: 5

❌ **Hata!** b değişkeni sadece {} içinde tanımlandığı için dışarıdan erişilemez.

**🔥 4. Döngülerde Kod Blokları Kullanımı**

Kod blokları **döngüler içinde** kullanılarak değişkenlerin erişim alanı belirlenebilir.

**📌 Örnek 4: Döngü İçinde Değişken Kullanımı**

public class LoopScope {

public static void main(String[] args) {

for (int i = 0; i < 3; i++) {

int square = i \* i;

System.out.println("i: " + i + ", Square: " + square);

}

// System.out.println("Square: " + square); // HATA! square değişkeni for bloğu içinde tanımlandı.

}

}

**Çıktı:**

i: 0, Square: 0

i: 1, Square: 1

i: 2, Square: 4

❌ **Hata!** square değişkeni sadece for döngüsünün içinde kullanılabilir.

**🔥 5. Metotlarda Kod Blokları Kullanımı**

Kod blokları **metotlar içinde** de kullanılır.

**📌 Örnek 5: Metot Kullanımı**

public class MethodScope {

public static void main(String[] args) {

printMessage(); // Metodu çağırıyoruz

}

public static void printMessage() {

String mesaj = "Merhaba Java!";

System.out.println(mesaj);

}

}

**Çıktı:**

Merhaba Java!

❌ **Hata yok!** Ancak mesaj değişkeni sadece printMessage() metodu içinde tanımlı olduğundan, main metodundan erişilemez.

**📌 Kod Bloklarında İyi ve Kötü Kod Örnekleri**

✔ **İyi Kod Örneği:** (Temiz ve düzenli)

public class JavaPatika {

public static void main(String[] args) {

int a = 1;

int b = 2;

int c = 3;

System.out.println("Burası main metota ait");

}

}

❌ **Kötü Kod Örneği:** (Okunması zor ve düzensiz)

public class JavaPatika {

public static void main(String[] args) {int a=1,b=2,c=3;System.out.println("Burası main metota ait");}

}

🚀 **Düzgün yazılmış kod, okunabilirliği artırır ve hata yapmayı önler!**

**🎯 Özet: Java'da Kod Blokları (Scope)**

| **Özellik** | **Açıklama** |
| --- | --- |
| **Kod Bloğu** | {} süslü parantezlerle belirlenir. |
| **Değişkenlerin Geçerliliği** | Bir değişken sadece tanımlandığı kod bloğunda geçerlidir. |
| **İç İçe Kod Blokları** | İç blok dış bloğa erişebilir, ama dış blok iç bloğa erişemez. |
| **Döngülerde Kullanım** | Döngüler içinde tanımlanan değişkenler sadece o döngü içinde geçerlidir. |
| **Metotlarda Kullanım** | Bir metotta tanımlanan değişken, sadece o metotta kullanılabilir. |

Kod bloklarını artık tamamen anladığını düşünüyor musun? 😃 Daha fazla örnek ister misin? 🚀

**Java'da If ve Else Blokları**

Java'da mantıksal ve koşullu ifadeleri kontrol etmek için **if** ve **else** blokları kullanılır. Bu bloklar, programın belirli durumlara göre farklı işlemler yapmasını sağlar.

**1. If ve Else Bloklarının Temelleri**

**If** kelimesi Türkçede "eğer" anlamına gelir. Bir koşul belirterek programın belirli bir kod bloğunu çalıştırmasını sağlar.

**Else** ise "değilse" anlamına gelir ve **if** koşulu sağlanmadığında çalıştırılacak kod bloğunu belirtir.

**Temel If Kullanımı**

if (koşul) {

// Koşul doğruysa (true) çalışacak kodlar

}

**If-Else Kullanımı**

if (koşul) {

// Koşul doğruysa (true) çalışacak kodlar

} else {

// Koşul yanlışsa (false) çalışacak kodlar

}

**If-Else If-Else Kullanımı**

if (koşul1) {

// Eğer koşul1 doğruysa çalışacak kodlar

} else if (koşul2) {

// Eğer koşul1 yanlış, koşul2 doğruysa çalışacak kodlar

} else {

// Eğer koşul1 ve koşul2 yanlışsa çalışacak kodlar

}

**2. If-Else Kullanımı ile Örnekler**

**Örnek 1: Basit If Kullanımı**

public class IfExample {

public static void main(String[] args) {

int age = 20;

if (age >= 18) {

System.out.println("Ehliyet alabilirsiniz.");

}

}

}

**Çıktı:**

Ehliyet alabilirsiniz.

**Örnek 2: If-Else Kullanımı**

public class IfElseExample {

public static void main(String[] args) {

int age = 16;

if (age >= 18) {

System.out.println("Ehliyet alabilirsiniz.");

} else {

System.out.println("Ehliyet almak için yaşınız yeterli değil.");

}

}

}

**Çıktı:**

Ehliyet almak için yaşınız yeterli değil.

**Örnek 3: If-Else If-Else Kullanımı**

public class IfElseIfExample {

public static void main(String[] args) {

int grade = 85;

if (grade >= 90) {

System.out.println("Harf notunuz: A");

} else if (grade >= 80) {

System.out.println("Harf notunuz: B");

} else if (grade >= 70) {

System.out.println("Harf notunuz: C");

} else {

System.out.println("Dersi geçemediniz.");

}

}

}

**Çıktı:**

Harf notunuz: B

**3. Mantıksal Operatörler ile If Kullanımı**

**Mantıksal operatörler**, birden fazla koşulu aynı anda kontrol etmemizi sağlar.

| **Operatör** | **Açıklama** |
| --- | --- |
| && | Ve (And) Operatörü: Her iki koşul da doğruysa (true) sonuç true olur. |
| ` |  |
| ! | Değil (Not) Operatörü: Koşulun tersini alır. true ise false, false ise true döner. |

**Örnek 4: && (Ve) Operatörü Kullanımı**

public class LogicalAndExample {

public static void main(String[] args) {

int age = 25;

int weight = 60;

if (age >= 18 && weight >= 50) {

System.out.println("Kan verebilirsiniz.");

} else {

System.out.println("Kan veremezsiniz.");

}

}

}

**Çıktı:**

Kan verebilirsiniz.

**Örnek 5: || (Veya) Operatörü Kullanımı**

public class LogicalOrExample {

public static void main(String[] args) {

int age = 16;

int parentalConsent = 1; // 1: Var, 0: Yok

if (age >= 18 || parentalConsent == 1) {

System.out.println("Ehliyet sınavına girebilirsiniz.");

} else {

System.out.println("Ehliyet sınavına giremezsiniz.");

}

}

}

**Çıktı:**

Ehliyet sınavına girebilirsiniz.

**Örnek 6: ! (Değil) Operatörü Kullanımı**

public class LogicalNotExample {

public static void main(String[] args) {

boolean isWeekend = false;

if (!isWeekend) {

System.out.println("Bugün hafta içi, çalışmalısınız.");

} else {

System.out.println("Bugün hafta sonu, dinlenebilirsiniz.");

}

}

}

**Çıktı:**

Bugün hafta içi, çalışmalısınız.

**4. İç İçe If Blokları (Nested If)**

Bir if bloğu içinde başka bir if bloğu kullanılabilir.

**Örnek 7: İç İçe If Kullanımı**

public class NestedIfExample {

public static void main(String[] args) {

int age = 20;

int weight = 55;

if (age >= 18) {

if (weight >= 50) {

System.out.println("Kan verebilirsiniz.");

} else {

System.out.println("Kilonuz yeterli değil, kan veremezsiniz.");

}

} else {

System.out.println("Yaşınız 18'den küçük, kan veremezsiniz.");

}

}

}

**Çıktı:**

Kan verebilirsiniz.

**5. If-Else ve Switch Karşılaştırması**

* **If-Else**: Mantıksal ifadeler kullanarak koşulları kontrol eder.
* **Switch**: Belirli bir değere göre çalışır ve genellikle daha okunaklıdır.

| **Kriter** | **If-Else** | **Switch** |
| --- | --- | --- |
| Kullanım | Koşullu ifadeler için uygundur | Sabit değerlere göre çalışır |
| Performans | Daha esnektir ama uzun kodlarda yavaş olabilir | Daha hızlıdır |
| Karmaşıklık | Karmaşık koşulları kontrol edebilir | Sadece eşitlik kontrolü yapabilir |

Bu rehber ile **if-else blokları** hakkında her şeyi öğrendiniz! 🚀

**Java'da Switch-Case Yapısı: Detaylı Konu Anlatımı**

**1. Switch-Case Nedir?**

Java’da **switch-case**, bir değişkenin farklı değerlere sahip olmasına bağlı olarak belirli kod bloklarının çalıştırılmasını sağlayan bir **koşul yapısıdır**. **if-else-if** yapısına alternatif olarak kullanılır ve özellikle **çok fazla koşul içeren durumlarda kodun daha okunaklı ve düzenli** olmasını sağlar.

**2. Switch-Case Sözdizimi (Syntax)**

Aşağıdaki genel yapı, Java'da switch-case kullanımını göstermektedir:

switch(değişken) {

case değer1:

// Eğer değişken değer1’e eşitse bu kod çalışır

break;

case değer2:

// Eğer değişken değer2’ye eşitse bu kod çalışır

break;

case değer3:

// Eğer değişken değer3’e eşitse bu kod çalışır

break;

default:

// Eğer değişken hiçbir case değeriyle eşleşmezse bu kod çalışır

}

**Açıklamalar:**

* **switch(değişken):** Switch ifadesinin içine kontrol etmek istediğimiz değişken yazılır.
* **case değer:** Eğer değişkenin değeri burada yazılan değere eşitse, ilgili kod bloğu çalıştırılır.
* **break;** Switch-case yapısının bir case bloğunda eşleşme bulduğunda diğer case'lere geçmesini engeller.
* **default:** Eğer değişkenin değeri yukarıdaki case ifadeleriyle eşleşmiyorsa, default bloğu çalıştırılır (zorunlu değildir).

**3. Switch-Case Örneği**

Aşağıdaki örnekte, kullanıcıdan bir gün numarası girmesi isteniyor ve switch-case yapısı ile hangi güne karşılık geldiği belirleniyor.

import java.util.Scanner;

public class GunBulma {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("Bir gün numarası girin (1-7): ");

int gun = input.nextInt();

switch (gun) {

case 1:

System.out.println("Pazartesi");

break;

case 2:

System.out.println("Salı");

break;

case 3:

System.out.println("Çarşamba");

break;

case 4:

System.out.println("Perşembe");

break;

case 5:

System.out.println("Cuma");

break;

case 6:

System.out.println("Cumartesi");

break;

case 7:

System.out.println("Pazar");

break;

default:

System.out.println("Geçersiz gün numarası!");

}

}

}

**Çalışma Mantığı:**

* Kullanıcı **1-7** arasında bir sayı girer.
* Sayıya karşılık gelen gün ekrana yazdırılır.
* Eğer kullanıcı **1-7 dışında bir sayı** girerse, **default** bloğu çalışır ve "Geçersiz gün numarası!" mesajı görüntülenir.

**Örnek Çıktılar:**

Bir gün numarası girin (1-7): 3

Çarşamba

Bir gün numarası girin (1-7): 9

Geçersiz gün numarası!

**4. Switch-Case'de break Kullanımı**

**Break Kullanımı**

Switch-case yapısında **break** komutu kullanılmazsa, bir case çalıştıktan sonra sonraki case'ler de **otomatik olarak çalışmaya devam eder** (fall-through denilen durum oluşur).

Aşağıdaki örnekte **break kullanılmamıştır:**

public class BreakOlmadanSwitch {

public static void main(String[] args) {

int sayi = 2;

switch (sayi) {

case 1:

System.out.println("Sayı 1");

case 2:

System.out.println("Sayı 2");

case 3:

System.out.println("Sayı 3");

default:

System.out.println("Geçersiz sayı");

}

}

}

**Çıktı:**

Sayı 2

Sayı 3

Geçersiz sayı

Çünkü **break olmadığı için**, case 2 çalıştıktan sonra case 3 ve default da çalışmaya devam eder.

Eğer **break** kullanılsaydı, sadece Sayı 2 yazdırılacaktı.

**5. Default Kullanımı**

**default** case’i, switch bloğundaki hiçbir case’in eşleşmediği durumda devreye girer.

Örnek:

public class DefaultOrnegi {

public static void main(String[] args) {

int gun = 10;

switch (gun) {

case 1:

System.out.println("Pazartesi");

break;

case 2:

System.out.println("Salı");

break;

case 3:

System.out.println("Çarşamba");

break;

default:

System.out.println("Geçersiz giriş!");

}

}

}

**Çıktı:**

Geçersiz giriş!

Burada **10** değeri hiçbir case ile eşleşmediği için **default** bloğu çalışır.

**6. Switch-Case ile String Kullanımı**

Java 7 ve sonraki sürümlerde **String** veri tipi de switch-case içinde kullanılabilir.

Örnek:

public class StringSwitch {

public static void main(String[] args) {

String meyve = "Elma";

switch (meyve) {

case "Elma":

System.out.println("Elma seçildi.");

break;

case "Armut":

System.out.println("Armut seçildi.");

break;

case "Muz":

System.out.println("Muz seçildi.");

break;

default:

System.out.println("Geçersiz meyve!");

}

}

}

**Çıktı:**

Elma seçildi.

Burada değişken bir **String** olduğu için case "Elma" eşleşir ve ilgili kod çalışır.

**7. Switch-Case ile Karakter (char) Kullanımı**

Switch-case yapısı **char** veri tipiyle de kullanılabilir.

Örnek:

public class CharSwitch {

public static void main(String[] args) {

char harf = 'B';

switch (harf) {

case 'A':

System.out.println("A harfi seçildi.");

break;

case 'B':

System.out.println("B harfi seçildi.");

break;

case 'C':

System.out.println("C harfi seçildi.");

break;

default:

System.out.println("Bilinmeyen harf!");

}

}

}

**Çıktı:**

B harfi seçildi.

**8. Switch-Case vs If-Else**

| **Özellik** | **Switch-Case** | **If-Else** |
| --- | --- | --- |
| **Performans** | Daha hızlıdır (derleme zamanında sabit değerleri kullanır). | Daha yavaştır (koşullar çalışma zamanında değerlendirilir). |
| **Okunabilirlik** | Daha okunaklıdır, özellikle çok fazla seçenek varsa. | Daha esnektir ama uzun kodlarda karmaşık hale gelebilir. |
| **Kullanım Alanı** | Sabit değerler için idealdir (int, char, String). | Mantıksal ifadeler, karşılaştırmalar gerektiren durumlar için uygundur. |

**9. Sonuç**

* **Switch-case**, bir değişkenin belirli sabit değerlere göre hangi kod bloğunun çalışacağını belirlemek için kullanılır.
* **break** kullanmazsak, eşleşen case’den sonra gelen tüm case’ler çalışır.
* **default** case’i, hiçbir eşleşme olmazsa çalışır.
* **String ve char** veri tipleri de switch-case içinde kullanılabilir.
* Çok fazla koşul içeren durumlarda switch-case, if-else'e göre **daha okunaklı ve hızlı** bir çözümdür.

Bu bilgileri kullanarak switch-case yapısını projelerinde daha etkin bir şekilde kullanabilirsin! 🚀

Tabii, sana Java’daki döngü yapılarını detaylıca ve örneklerle açıklayayım! 🚀

**🔥 Java’da Döngü Yapıları**

Java’da 3 temel döngü yapısı vardır:

| **Döngü Yapısı** | **Kullanım Amacı** | **Koşul Kontrolü** |
| --- | --- | --- |
| for | Belirli bir sayıda tekrarlama | Döngü başında |
| while | Koşul doğru olduğu sürece tekrarlama | Döngü başında |
| do-while | En az bir kez çalıştırır, sonra kontrol eder | Döngü sonunda |

**1️⃣ for Döngüsü**

Genelde **başlangıç değeri, bitiş şartı ve artış/azalış** belli ise kullanılır.

**Söz dizimi:**

for (başlangıç; koşul; arttır/azalt) {

// tekrar edilecek kod bloğu

}

💡 **Örnek:** 1’den 5’e kadar sayıları yazdıran döngü

for (int i = 1; i <= 5; i++) {

System.out.println("i: " + i);

}

➡️ i = 1 başlar, i <= 5 olduğu sürece System.out.println çalışır ve i her döngüde 1 artar.  
Koşul sağlanmazsa döngü biter.

**2️⃣ while Döngüsü**

Eğer tekrar sayısı önceden bilinmiyorsa ve **koşula göre döngüyü sürdürmek** istiyorsan while kullanılır.

**Söz dizimi:**

while (koşul) {

// tekrar edilecek kod bloğu

}

💡 **Örnek:** 5’ten 1’e kadar geriye sayan döngü

int i = 5;

while (i > 0) {

System.out.println("i: " + i);

i--;

}

➡️ i sıfırdan büyük olduğu sürece kod bloğu tekrar eder.

**3️⃣ do-while Döngüsü**

Bu döngüde **önce işlem yapılır, sonra koşul kontrol edilir.** Yani en az 1 defa çalışır.

**Söz dizimi:**

do {

// en az bir kez çalışacak kod bloğu

} while (koşul);

💡 **Örnek:** Kullanıcı 0 girene kadar sayı isteme

import java.util.Scanner;

int number;

Scanner input = new Scanner(System.in);

do {

System.out.print("Bir sayı girin (Çıkış için 0): ");

number = input.nextInt();

} while (number != 0);

➡️ Kullanıcı 0 girene kadar döngü devam eder. İlk giriş her zaman istenir!

**💡 Ekstra: Döngü Kontrol İfadeleri**

* break: Döngüyü anında bitirir.
* continue: O anki adımı atlayıp sonraki adıma geçer.

💡 **Örnek: break**

for (int i = 1; i <= 10; i++) {

if (i == 5) break;

System.out.println(i);

}

➡️ i == 5 olduğunda döngü sonlanır.

💡 **Örnek: continue**

for (int i = 1; i <= 5; i++) {

if (i == 3) continue;

System.out.println(i);

}

➡️ i == 3 olduğunda o adımı atlar, diğerlerini yazdırır.

**📌 Özet**

* Döngüler **tekrar eden işlemler** için çok kullanışlıdır.
* for → Sabit sayıda tekrar.
* while → Belirsiz sayıda, koşula bağlı tekrar.
* do-while → En az bir defa çalışır, sonra koşul kontrol edilir.

Eğer istersen, algoritma örnekleri veya nested loop (iç içe döngü) mantığını da örneklerle açıklayayım!  
İstersen Java’da döngülerle küçük bir mini-proje bile yazabiliriz. 🎯  
Ne dersin?

Tabii, çok güzel bir konu seçmişsin! 🎯  
Şimdi sana **Java'da while döngüsünü** en sade, detaylı ve örnekli bir şekilde anlatıyorum.

**💡 Java while Döngüsü Nedir?**

while döngüsü, **koşul sağlandığı sürece aynı kod bloğunu tekrar tekrar çalıştırmak** için kullanılır.

**🔥 Yapısı:**

while (koşul) {

// tekrar edecek kod bloğu

}

**🧠 Çalışma Mantığı:**

1. Java, önce **koşulu kontrol eder**.
2. Eğer true ise, **kod bloğunu çalıştırır**.
3. Kod bloğunun sonunda **başa döner**, koşulu tekrar kontrol eder.
4. Koşul false olursa, döngü **biter**.

**⚡️ Basit Örnek:**

int i = 1;

while (i <= 5) {

System.out.println("i: " + i);

i++;

}

💡 **Açıklama:**

* i değişkeni 1 olarak başlar.
* i <= 5 olduğu sürece System.out.println() satırı çalışır.
* Her seferinde i bir artırılır.
* i değeri 6 olduğunda koşul false olur ve döngü biter.

**💡 Tek Satırlık Döngü:**

Eğer döngü içinde sadece 1 satırlık kod varsa, {} süslü parantez yazmaya gerek yok.

int i = 1;

while (i <= 3)

System.out.println(i++);

**⚠️ Sonsuz Döngü:**

Eğer koşul hiçbir zaman false olmazsa, döngü sonsuza kadar çalışır!

while (true) {

System.out.println("Bu döngü asla bitmez!");

}

Dikkatli kullanılmalı, programı kilitler.

**💡 Uygulamalı Gerçek Hayat Örneği: Şifre Kontrolü**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String password = "Java123";

boolean correct = false;

while (!correct) {

System.out.print("Şifrenizi giriniz: ");

String input = scanner.nextLine();

if (input.equals(password)) {

correct = true;

System.out.println("✅ Şifre doğru, giriş başarılı!");

} else {

System.out.println("❌ Yanlış şifre, tekrar deneyin.");

}

}

}

}

💡 **Ne oldu burada?**

* Kullanıcı doğru şifreyi girene kadar döngü tekrar eder.
* Doğru şifre girildiğinde correct = true olur, döngü biter.

**🧾 Kod Bloğu Olmadan Döngü (İlginç Kullanım)**

Bazen döngünün **içinde kod olmaz**, tüm iş sadece koşul içinde yapılır.

int left = 100, right = 200;

while (++left < --right);

System.out.println("Ortadaki sayı: " + left);

Bu kod left ve right değerlerini ortada buluşturur, sonuç:

Ortadaki sayı: 150

**📌 While Döngüsü Kullanım Senaryoları:**

| **Senaryo** | **Açıklama** |
| --- | --- |
| Kullanıcıdan doğru şifre beklemek | Kaç kez yanlış gireceğini bilemeyiz. |
| Sayaç kullanmadan belirsiz tekrarlama | Şart sağlanana kadar otomatik döner. |
| Veritabanı, network, IO kontrolleri | Bağlantı sağlanana kadar tekrar dener. |
| Oyunlarda sonsuz döngüler | Oyuncu çıkış yapana kadar oyun devam eder. |

**✅ Kısaca Özet:**

* while döngüsü koşul doğru olduğu sürece kod bloğunu çalıştırır.
* En çok **kaç kere çalışacağı belli olmayan durumlarda** kullanılır.
* Döngü başlamadan önce koşul kontrol edilir, false ise hiç çalışmayabilir.

💡 İstersen sana do-while döngüsüyle while döngüsünün farklarını da örnekli anlatırım!  
İstersen bir mini oyun mantığında bir while döngüsü de yazabiliriz. İlgilenir misin? 🚀

Tabii! Hadi sana basit, eğlenceli ve öğretici bir **mini oyun** yazayım!  
Java’da while döngüsünü kullanarak **"Sayı Tahmin Oyunu"** yapalım. 🎲🎮

**💡 Java - Sayı Tahmin Oyunu (while döngüsü ile)**

import java.util.Scanner;

import java.util.Random;

public class SayiTahminOyunu {

public static void main(String[] args) {

Random random = new Random();

Scanner scanner = new Scanner(System.in);

int rastgeleSayi = random.nextInt(100) + 1; // 1-100 arası sayı üretir

int tahmin = 0;

int denemeSayisi = 0;

System.out.println("🎮 Sayı Tahmin Oyununa Hoş Geldin!");

System.out.println("1 ile 100 arasında bir sayı tuttum, bakalım bulabilecek misin?");

while (tahmin != rastgeleSayi) {

System.out.print("Tahminin: ");

tahmin = scanner.nextInt();

denemeSayisi++;

if (tahmin < rastgeleSayi) {

System.out.println("Daha büyük bir sayı dene! 🔼");

} else if (tahmin > rastgeleSayi) {

System.out.println("Daha küçük bir sayı dene! 🔽");

} else {

System.out.println("🎉 TEBRİKLER! " + denemeSayisi + " denemede doğru tahmin ettin!");

}

}

scanner.close();

}

}

**💡 Nasıl Çalışır?**

* Program 1-100 arası bir sayı tutar (Random sınıfı ile).
* while döngüsü sayesinde kullanıcı doğru tahmin edene kadar sayı istemeye devam eder.
* Doğru tahmin yapılınca döngü biter ve tebrik mesajı basar.

**🧠 Burada while döngüsü nasıl kullanıldı?**

while (tahmin != rastgeleSayi) {

// Kullanıcı doğru bilene kadar dönüyor.

}

Yani, ne zaman biteceği belli olmayan (doğru tahmin) bir durum için mükemmel bir kullanım!

💪 İstersen:

* hak sayısı sınırlı versiyonunu,
* puanlı bir versiyonunu,
* do-while döngüsüyle yapılmış halini

de yazabilirim! İster misin? 🚀

Tabii, çok güzel bir konu! Hadi sana **Java'da for döngüsü** konusunu detaylı ve anlaşılır şekilde anlatayım. 🚀

**💡 Java — For Döngüsü Nedir?**

for döngüsü, **belli bir başlangıçtan itibaren, koşul sağlandığı sürece, adım adım artan veya azalan bir sayaç** ile belirli işlemleri tekrar etmek için kullanılır.

Yani, **kaç kez çalışacağını önceden bildiğimiz** durumlarda en ideal döngü tipidir.

**⚙️ For Döngüsünün Söz Dizimi:**

for (başlangıç; koşul; arttır/azalt) {

// döngü çalışacak kod bloğu

}

**🧠 For Döngüsünün Adımları:**

| **Adım** | **Açıklama** |
| --- | --- |
| 1️⃣ | başlangıç: Döngü değişkeni tanımlanır ve başlangıç değeri atanır. (ör: int i = 0;) |
| 2️⃣ | koşul: Bu şart sağlandığı sürece döngü devam eder. Şart sağlanmazsa döngü biter. |
| 3️⃣ | arttır/azalt: Her döngü sonunda sayaç güncellenir (ör: i++). |

**💡 Örnek 1: Basit For Döngüsü**

for (int i = 1; i <= 5; i++) {

System.out.println("i değeri: " + i);

}

**Çıktı:**

i değeri: 1

i değeri: 2

i değeri: 3

i değeri: 4

i değeri: 5

**💡 Örnek 2: Geriye Doğru Sayım**

for (int i = 5; i > 0; i--) {

System.out.println(i);

}

**Çıktı:**

5

4

3

2

1

**💡 Örnek 3: Çift Sayıları Yazdırma**

for (int i = 2; i <= 20; i += 2) {

System.out.println(i);

}

**Çıktı:**

2

4

6

8

10

12

14

16

18

20

**💡 For Döngüsünde Süslü Parantez Olmadan Yazım**

Eğer döngü içinde **tek satırlık kod** varsa {} kullanmak zorunda değilsin:

for (int i = 0; i < 5; i++)

System.out.println(i);

**💡 For Döngüsü Kullanım Alanları**

| **Kullanım Alanı** | **Açıklama** |
| --- | --- |
| Liste/Array elemanlarıyla işlem | Dizi, liste tarama |
| Belirli aralıkta tekrar eden işlem | Sayaçlı adım işlemleri |
| Matematiksel işlemler | Faktöriyel, toplam, çarpım gibi |
| Geriye sayım | Zamanlayıcı, geri sayım |

**🚀 For Döngüsüne Örnek Mini Uygulama: Faktöriyel Hesaplama**

int sayi = 5;

int faktoriyel = 1;

for (int i = 1; i <= sayi; i++) {

faktoriyel \*= i;

}

System.out.println("Sonuç: " + faktoriyel);

**Çıktı:**

Sonuç: 120

✅ **Kısaca Özet:**

* for döngüsü, **tekrar sayısı bilinen durumlarda** tercih edilir.
* Başlangıç, koşul ve adım kısmı döngü başında net tanımlanır.
* Okunaklı, düzenli ve hata riski düşük bir döngü yapısı sağlar.

İstersen sana for-each döngüsünü de anlatayım (Java koleksiyonlarında çok sık kullanılır)!  
İstersen for ile bir **yıldızlı üçgen çizimi** gibi görsel bir örnek hazırlayayım mı? 🌟

Süper soru! Hadi sana **Java’daki for, while ve do-while döngülerinin farklarını** detaylıca, sade bir anlatımla ve örneklerle açıklayayım.

**💡 For, While ve Do-While Döngülerinin Farkları**

| **Özellik** | **for Döngüsü** | **while Döngüsü** | **do-while Döngüsü** |
| --- | --- | --- | --- |
| **Kullanım amacı** | Döngü kaç kez dönecek **belli** ise | Döngü kaç kez dönecek **belli değilse** | Döngü en az **1 kere çalışacaksa** |
| **Koşul kontrolü** | Döngü başlamadan **önce** kontrol edilir | Döngü başlamadan **önce** kontrol edilir | Döngü başladığında, koşul **sonunda** kontrol edilir |
| **Genel yapı** | Başlangıç, koşul ve artış aynı satırda | Sadece koşul var, sayaç dışarıda tanımlanır | Önce çalışır, sonra koşul kontrol edilir |
| **Kullanım tipi** | Sabit, sayılabilir işlemler (örn: 1’den 10’a kadar) | Koşula bağlı belirsiz durumlar (örn: şifre girişi) | Kullanıcı en az 1 kez işlem yapmalıysa (örn: menü gösterme) |

**🔥 For Döngüsü — Ne Zaman Kullanılır?**

**→ Tekrar sayısı baştan belliyse for döngüsü kullanılır.**

Örnek: 10 kere bir mesaj yazdırmak.

for (int i = 1; i <= 10; i++) {

System.out.println("Merhaba, bu bir döngüdür! (" + i + ")");

}

💡 **Ne zaman kullanılır?**

* Sabit sayıda işlem tekrarlanacaksa
* Diziler, listeler gibi elemanlara sırayla ulaşılacaksa
* Sayaç kullanımı gerekiyorsa (örn: 0’dan 100’e kadar artış)

**🔥 While Döngüsü — Ne Zaman Kullanılır?**

**→ Kaç kez çalışacağı baştan belli değilse while döngüsü kullanılır.**

Örnek: Kullanıcı şifreyi doğru girene kadar deneme yapması.

Scanner scanner = new Scanner(System.in);

String sifre = "12345";

String girilen;

do {

System.out.print("Şifreyi girin: ");

girilen = scanner.nextLine();

} while (!girilen.equals(sifre));

System.out.println("Tebrikler, giriş başarılı!");

💡 **Ne zaman kullanılır?**

* Kullanıcı girdisine göre işlem tekrarı gerekiyorsa.
* Program belirli bir duruma ulaşana kadar çalışacaksa.
* Döngü koşulu **önceden bilinemeyen** durumlarda.

**🔥 Do-While Döngüsü — Ne Zaman Kullanılır?**

**→ Döngü en az 1 kere çalışmalıysa do-while döngüsü kullanılır.**  
(çünkü önce kod çalışır, sonra koşula bakılır.)

Örnek: Kullanıcıdan sayı al, 100'den küçükse tekrar iste.

int sayi;

Scanner scanner = new Scanner(System.in);

do {

System.out.print("100'den büyük bir sayı giriniz: ");

sayi = scanner.nextInt();

} while (sayi < 100);

System.out.println("Tebrikler! Geçerli sayı girdiniz.");

💡 **Ne zaman kullanılır?**

* Menülerde: Kullanıcı seçim yapmadan program kapanmasın.
* En az bir defa işlem yapma zorunluluğu varsa.
* Kullanıcının girdiği değere göre döngünün devam edip etmeyeceğine karar verilecekse.

**🎯 Genel Özet:**

| **Senaryo** | **Kullanılacak Döngü** |
| --- | --- |
| 1-10 arası sayıları yazdır | for döngüsü |
| Kullanıcı şifre girene kadar bekleme | while döngüsü |
| Kullanıcı en az 1 kere işlem yapmalı | do-while döngüsü |

**🧠 Bonus: Aynı İş, 3 Döngüyle**

Mesela: 1'den 5'e kadar yazdırma.

**For Döngüsü:**

for (int i = 1; i <= 5; i++) {

System.out.println(i);

}

**While Döngüsü:**

int i = 1;

while (i <= 5) {

System.out.println(i);

i++;

}

**Do-While Döngüsü:**

int i = 1;

do {

System.out.println(i);

i++;

} while (i <= 5);

💡 **SONUÇ:**

* for: Tekrar sayısı belli ise.
* while: Koşul sağlandıkça devam, sayı önceden belli değilse.
* do-while: En az 1 kez çalıştır, sonra kontrol et.

✅ Eğer istersen bu 3 döngüyle mini oyunlar, menülü uygulamalar veya dizi işlemleri için örnekler hazırlayayım!  
Bir senaryo seç, birlikte kodlayalım! ⚡💻