if / else

Yeni Control Flow kategorisinin ilk topic'i: `if` / `else`'in temel kullanımı, `else if` zinciri, iç içe koşullar, altı karşılaştırma operatörü (`==` / `!=` / `<` / `>` / `<=` / `>=`) ve ondalıklı sayı/`String` karşılaştırma tuzakları, `&&` / `||` / `!` mantıksal operatörleri ve kısa devre değerlendirme, ve üçlü (ternary) operatör.

Başlangıç 20 dk

Son güncelleme: 19 Ekim 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/if-else

---

if / else

Java kursunun yeni "Kontrol Akışı" kategorisinin ilk konusu if / else -- bir programın farklı koşullara göre farklı kod yollarını çalıştırmasını sağlayan en temel karar verme (decision making) yapısı. `if`, bir koşul `true` olduğunda bir kod bloğunu çalıştırır; `else` ise o koşul `false` olduğunda çalışacak alternatif bir blok tanımlar. Bu ders, `String` / `Arrays` / `Scanner` gibi Java Temelleri konularının örneklerinde zaten örtük olarak karşılaşmış olabileceğin bu yapıyı ilk kez baştan sona, sistematik olarak ele alıyor.

if / else Nedir?

`if` ifadesi, parantez içindeki bir `boolean` koşulu değerlendirir -- koşul `true` ise hemen ardından gelen kod bloğu çalışır, `false` ise atlanır. `else` bloğu isteğe bağlıdır ve yalnızca `if` koşulu `false` olduğunda çalışır. `else if`, birden fazla koşulu sırayla kontrol etmek için `else` ile yeni bir `if`'i birleştirir (bkz. "else if Zinciri"). Koşul olarak yalnızca bir `boolean` ifade kabul edilir -- Java, C/C++'ın aksine `0` / `1` gibi sayısal değerleri otomatik olarak `boolean`'a çevirmez, `if (sayi)` gibi bir kod DERLENMEZ.

Neden Var?

Bir program her zaman aynı işlemleri yapsaydı, koşullu mantığa gerek kalmazdı -- ama gerçek dünyadaki her program dallanma (branching) gerektirir: bir kullanıcı girişi geçerli mi, bir stok miktarı yeterli mi, bir yaş bir eşiği geçiyor mu? `if` / `else`, bu tür ikili (ya da `else if` ile çoklu) kararları okunabilir, sıralı bir şekilde ifade etmenin en doğrudan yoludur. Neredeyse her algoritmanın temelinde -- doğrulama, filtreleme, hata yönetimi -- bir yerde `if` / `else` vardır.

Tarihçe

`if` / `else`, yapısal programlama (structured programming) hareketinin bir parçası olarak 1960'ların sonunda ALGOL ve Pascal gibi dillerde standartlaştı ve oradan C'ye (1972) geçti -- Java (1995), sözdizimini neredeyse birebir C'den miras aldı. Java'nın tek büyük farkı, koşulun kesinlikle bir `boolean` olması zorunluluğu: C'de yaygın olan `if (x)` (x sıfır değilse true) kalıbı, Java'da bilinçli olarak DERLEME HATASINA çevrildi -- bu, "atama (`=`) yerine karşılaştırma (`==`) yazma" gibi C'de sık görülen bir hata sınıfını baştan engelliyor.

Temel if / else Kullanımı

En basit hâliyle `if` tek başına, `else` ile birlikte, ya da tamamen atlanabilir. Her iki blok da tek bir ifadeden oluşuyorsa süslü parantez (`{}`) teknik olarak isteğe bağlıdır -- ama bu, gerçek bir hata kaynağıdır (bkz. "Yaygın Hatalar").

```
public class IfElseBasicsExample {
    public static void main(String[] args) {
        int number = 7;

        if (number % 2 == 0) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
        }

        int age = 16;

        if (age >= 18) {
            System.out.println("Eligible to vote.");
        } else {
            System.out.println("Not eligible to vote yet.");
        }

        // A single-statement if/else WITHOUT braces -- legal, but risky.
        // See "Common Mistakes" for why braces should be used anyway.
        int score = 42;
        if (score > 50)
            System.out.println("Passed.");
        else
            System.out.println("Failed.");
    }
}
```

Beklenen Çıktı

```
7 is odd.
Not eligible to vote yet.
Failed.
```

> Süslü parantez olmadan yazılan bir `if` / `else`, yalnızca HEMEN SONRAKİ tek ifadeyi kapsar -- ikinci bir satır eklemek istediğinde bu satır koşulun dışında kalır ama girinti (indentation) hâlâ içindeymiş gibi GÖRÜNÜR. Bu yüzden tek satırlık bloklarda bile süslü parantez kullanmak güçlü bir pratik olarak önerilir.

else if Zinciri

İkiden fazla dallanma gerektiğinde `else if`, `else` ile yeni bir `if`'i birleştirerek koşulları YUKARIDAN AŞAĞIYA sırayla kontrol eder -- ilk `true` olan koşulun bloğu çalışır, geri kalanlar (koşulları hâlâ `true` olsa bile) ATLANIR. Bu yüzden koşulların sırası önemlidir: daha DAR /ÖZEL koşullar genellikle daha GENİŞ koşullardan önce yazılmalıdır.

```
public class ElseIfChainExample {
    public static void main(String[] args) {
        printGrade(95);
        printGrade(82);
        printGrade(71);
        printGrade(65);
        printGrade(40);
    }

    // Note: repeating this for a list of scores with a loop is exactly what
    // you'll learn in the "for Loop" lesson -- this topic keeps it loop-free.
    private static void printGrade(int score) {
        String grade;

        if (score >= 90) {
            grade = "A";
        } else if (score >= 80) {
            grade = "B";
        } else if (score >= 70) {
            grade = "C";
        } else if (score >= 60) {
            grade = "D";
        } else {
            grade = "F";
        }

        System.out.println("Score " + score + " -> Grade " + grade);
    }
}
```

Beklenen Çıktı

```
Score 95 -> Grade A
Score 82 -> Grade B
Score 71 -> Grade C
Score 65 -> Grade D
Score 40 -> Grade F
```

İç İçe Koşullar (Nested Conditions)

Bir `if`'in gövdesi içine başka bir `if` yazmak -- nested condition -- iki koşulun BİRBİRİNE BAĞLI olarak (yani ikinci koşulun yalnızca birincisi sağlandığında anlamlı olduğu durumlarda) değerlendirilmesi gerektiğinde kullanılır. Bu, `&&` ile birleştirilmiş tek bir `if`'e çoğu zaman eşdeğerdir (bkz. "Mantıksal Operatörler ve Kısa Devre Değerlendirme") ama iç içe yapı, her koşul için AYRI bir `else` dalı (farklı bir hata mesajı gibi) gerektiğinde daha okunaklıdır.

```
public class NestedConditionsExample {
    public static void main(String[] args) {
        checkLogin("alice", "correct-password");
        checkLogin("unknown-user", "whatever");
        checkLogin("alice", "wrong-password");
    }

    private static void checkLogin(String username, String password) {
        String storedUsername = "alice";
        String storedPassword = "correct-password";

        // An if INSIDE another if's body -- a nested condition. The inner check
        // only runs when the outer one is already true.
        if (username.equals(storedUsername)) {
            if (password.equals(storedPassword)) {
                System.out.println("Login successful for " + username);
            } else {
                System.out.println("Wrong password for " + username);
            }
        } else {
            System.out.println("No such user: " + username);
        }
    }
}
```

Beklenen Çıktı

```
Login successful for alice
No such user: unknown-user
Wrong password for alice
```

> Üç dört seviyeden fazla iç içe geçen `if` bloğu genellikle okunabilirliği ciddi şekilde düşürür ("arrow anti-pattern" olarak bilinir) -- bu durumda erken dönüş (`return`) ile koşulları düzleştirmek ya da koşulları `&&` ile tek bir `if`'te birleştirmek tercih edilmelidir.

Karşılaştırma Operatörleri

Java altı karşılaştırma operatörü sunar: `==` (eşit mi), `!=` (eşit değil mi), `<` , `>` , `<=` , `>=` . Bunların hepsi bir `boolean` sonuç üretir ve doğrudan bir `if` koşulu olarak kullanılabilir. `==` ilkel (primitive) tipler için DEĞERİ karşılaştırırken, nesneler (örn. `String`) için REFERANSI karşılaştırır -- bkz. "String" dersindeki "String Pool ve == vs equals()" bölümü.

```
public class ComparisonOperatorsExample {
    public static void main(String[] args) {
        int a = 10;
        int b = 20;

        System.out.println("a == b: " + (a == b));
        System.out.println("a != b: " + (a != b));
        System.out.println("a < b: " + (a < b));
        System.out.println("a > b: " + (a > b));
        System.out.println("a <= b: " + (a <= b));
        System.out.println("a >= b: " + (a >= b));

        // Comparing floating-point numbers directly with == is risky -- binary
        // floating point cannot represent 0.1 or 0.2 exactly, so their sum is
        // not exactly 0.3.
        double sum = 0.1 + 0.2;
        System.out.println("0.1 + 0.2 == 0.3: " + (sum == 0.3));
        System.out.println("Actual value: " + sum);

        // Comparing String OBJECTS with == compares references, not content --
        // see the "String Pool ve == vs equals()" section in the String lesson.
        String literalOne = "hello";
        String literalTwo = "hello";
        String constructed = new String("hello");

        System.out.println("literalOne == literalTwo: " + (literalOne == literalTwo));
        System.out.println("literalOne == constructed: " + (literalOne == constructed));
        System.out.println("literalOne.equals(constructed): " + literalOne.equals
(constructed));
    }
}
```

Beklenen Çıktı

```
a == b: false
a != b: true
a < b: true
a > b: false
a <= b: true
a >= b: false
0.1 + 0.2 == 0.3: false
Actual value: 0.30000000000000004
literalOne == literalTwo: true
literalOne == constructed: false
literalOne.equals(constructed): true
```

Ondalıklı sayıları (`double` / `float`) `==` ile karşılaştırmak risklidir -- ikili (binary) kayan noktalı sayı temsili, `0.1` gibi ondalıkları TAM OLARAK saklayamaz, bu yüzden `0.1 + 0.2 == 0.3` `false` döner. Yaklaşık karşılaştırma için iki değer arasındaki farkın küçük bir eşik (epsilon) değerinden küçük olup olmadığına bakılmalıdır.

Mantıksal Operatörler ve Kısa Devre Değerlendirme

`&&` (VE), `||` (VEYA) ve `!` (DEĞİL), birden fazla `boolean` ifadeyi birleştirmeyi sağlar. İkisi de KISA DEVRE (short-circuit) değerlendirilir: `&&`'nin sol tarafı `false` ise sağ taraf HİÇ ÇALIŞTIRILMAZ (sonuç zaten `false` olacağından); `||`'nin sol tarafı `true` ise sağ taraf HİÇ ÇALIŞTIRILMAZ. Bu yalnızca bir performans optimizasyonu değildir -- sağ taraftaki bir metot çağrısının yan etkisi (örn. `null` kontrolü sonrası bir metot çağırma) buna GÜVENİLEREK yazılabilir.

```
public class LogicalOperatorsExample {
    public static void main(String[] args) {
        int age = 25;
        boolean hasLicense = true;

        System.out.println("Can drive: " + (age >= 18 && hasLicense));

        boolean isWeekend = false;
        boolean isHoliday = true;

        System.out.println("Day off: " + (isWeekend || isHoliday));
        System.out.println("Not a day off: " + !(isWeekend || isHoliday));

        // Short-circuit evaluation: && skips the right-hand side once the
        // left-hand side is already false; || skips it once the left-hand
        // side is already true.
        System.out.println("Short-circuit && demo:");
        System.out.println(checkFirst() && checkSecond());

        System.out.println("Short-circuit || demo:");
        System.out.println(checkThird() || checkFourth());
    }

    private static boolean checkFirst() {
        System.out.println("  checkFirst() called");
        return false;
    }

    private static boolean checkSecond() {
        System.out.println("  checkSecond() called -- should NOT print, short-circuited");
        return true;
    }

    private static boolean checkThird() {
        System.out.println("  checkThird() called");
        return true;
    }

    private static boolean checkFourth() {
        System.out.println("  checkFourth() called -- should NOT print, short-circuited");
        return false;
    }
}
```

Beklenen Çıktı

```
Can drive: true
Day off: true
Not a day off: false
Short-circuit && demo:
  checkFirst() called
false
Short-circuit || demo:
  checkThird() called
true
```

Üçlü (Ternary) Operatör

`koşul ? değer1 : değer2` biçimindeki üçlü operatör, basit bir `if` / `else`'in bir DEĞER üretmesi gerektiğinde (örn. bir değişkene atama) kısa bir alternatiftir -- `if` / `else` bir İFADE değil bir İFADE BLOĞUdur, üçlü operatör ise doğrudan bir değere değerlendirilen bir İFADEDİR.

```
public class TernaryOperatorExample {
    public static void main(String[] args) {
        int age = 20;
        String status = (age >= 18) ? "adult" : "minor";
        System.out.println("Status: " + status);

        int a = 15;
        int b = 42;
        int max = (a > b) ? a : b;
        System.out.println("Max: " + max);

        // Ternary expressions can be nested, but this quickly hurts readability --
        // prefer an if/else-if chain (see "else if Chains") once you need more
        // than two outcomes.
        int score = 72;
        String grade = (score >= 90) ? "A" : (score >= 80) ? "B" : (score >= 70) ? "C" :
"F";
        System.out.println("Nested ternary grade: " + grade);
    }
}
```

Beklenen Çıktı

```
Status: adult
Max: 42
Nested ternary grade: C
```

> Üçlü operatörü iç içe kullanmak (`a ? b : c ? d : e`) teknik olarak çalışır ama hızla okunmaz hâle gelir -- ikiden fazla sonuç gerektiğinde bir `else if` zinciri neredeyse her zaman daha nettir.

Best Practices

- **Tek satırlık `if`/`else` bloklarında bile süslü parantez kullanın** -- gelecekte eklenecek bir satırın yanlışlıkla koşulun dışında kalmasını önler.
- **`else if` zincirlerinde koşulları en dar/özelden en genişe doğru sıralayın** -- ilk eşleşen koşul çalışır, sıra yanlışsa daha geniş bir koşul daha özel olanı "gölgeleyebilir".
- **Üç dört seviyeden fazla iç içe `if` yerine erken dönüş (`return`) ya da `&&` ile birleştirilmiş tek bir koşul kullanın.**
- **Ondalıklı sayıları `==` ile değil, bir epsilon eşiğiyle karşılaştırın.**
- **Üçlü operatörü yalnızca basit, tek bir değer seçiminde kullanın** -- iç içe üçlü operatör okunabilirliği düşürür.

Yaygın Hatalar

- **Süslü parantezsiz bir `if`'in ardına yanlışlıkla ikinci bir satır eklemek.** Girinti bu satırın `if`'e ait olduğunu gösterir gibi görünür ama derleyici için değildir -- satır koşuldan BAĞIMSIZ her zaman çalışır.
- **`else if` sırasını yanlış kurup daha geniş bir koşulu daha özel olandan ÖNCE yazmak.** Bu durumda özel koşul asla çalışmaz çünkü ondan önceki geniş koşul zaten `true` olur.
- **Ondalıklı sayıları `==` ile karşılaştırıp beklenmedik `false` sonuçlar almak.** `0.1 + 0.2 == 0.3` klasik örnektir -- ikili kayan nokta temsilinden kaynaklanır.
- **İki `String`'i `==` ile karşılaştırmak.** Bu, İÇERİĞİ değil REFERANSI karşılaştırır -- her zaman `.equals()` kullanılmalıdır.

Özet, Cheat Sheet ve Terimler Sözlüğü

`if`/`else`, bir `boolean` koşula göre farklı kod yolları çalıştırmayı sağlayan en temel dallanma yapısıdır. `else if`, birden fazla koşulu sırayla (ilk `true` olan kazanır) kontrol eder. İç içe koşullar birbirine bağlı kontroller için kullanılır ama okunabilirliği hızla düşürebilir. Karşılaştırma operatörleri (`==`, `!=`, `<`, `>`, `<=`, `>=`) bir `boolean` üretir; `==` nesnelerde REFERANS karşılaştırır. Mantıksal operatörler (`&&`, `||`, `!`) kısa devre değerlendirilir. Üçlü operatör (`? :`), basit bir `if`/`else`'in bir DEĞER üreten kısa biçimidir.

Hızlı referans:

```
if (kosul) {
    // ...
} else if (baskaKosul) {
    // ...
} else {
    // ...
}

boolean sonuc = (a > b) && (c < d);                    // mantıksal VE, kısa devre
String etiket = (yas >= 18) ? "yetiskin" : "cocuk";   // üçlü operatör
```

Terimler Sözlüğü

Condition (Koşul) — Bir `if` / `else if`'in değerlendirdiği, `true` ya da `false` sonucu üreten `boolean` ifade.

Branching (Dallanma) — Bir programın koşula göre farklı kod yollarını çalıştırması.

Short-Circuit Evaluation (Kısa Devre Değerlendirme) — `&&` / `||`'de sonucun sol taraftan zaten kesinleştiği durumlarda sağ tarafın hiç çalıştırılmaması.

Ternary Operator (Üçlü Operatör) — `koşul ? değer1 : değer2` biçiminde, doğrudan bir değere değerlendirilen koşullu ifade.