while & do-while Loops

Control Flow kategorisinin beşinci topic'i: temel `while` sözdizimi, gövdeyi en az bir kez çalıştıran `do-while`, while vs do-while karşılaştırması, break/continue, Scanner ile girdi doğrulama döngüsü, ve uygulamalı örnek olarak Sayı Tahmin Oyunu (Number Guessing Game).

Başlangıç 20 dk

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

---

while & do-while Loops

`for` döngüsü "kaç kez çalışacağını önceden biliyorum" durumları için idealdi. `while` ve `do-while` ise tam tersi bir soruyu cevaplar: *"Ne zaman duracağımı biliyorum ama kaç adımda duracağımı bilmiyorum."* Kullanıcıdan geçerli bir giriş gelene kadar sormak, bir dosyanın sonuna gelene kadar okumak, bir bağlantı kurulana kadar denemek -- bunların hepsi "koşul sağlandığı sürece devam et" mantığıdır, sayaç mantığı değil.

while Döngüsü Nedir?

`while` döngüsü, bir koşul doğru olduğu sürece bir kod bloğunu tekrar tekrar çalıştırır:

```
while (koşul) {
    // koşul doğru olduğu sürece çalışır
}
```

Koşul her yinelemeden **önce** kontrol edilir. Koşul en baştan yanlışsa, döngü gövdesi hiç çalışmaz -- bu, `for` döngüsüyle aynı davranıştır, ama `while`'da sayaç/artırım gibi bir zorunluluk yoktur; koşul tamamen keyfi bir mantık olabilir.

Neden Var?

`for` döngüsü "başlangıç; koşul; artırım" üçlüsünü tek satırda toplar çünkü bu üçü genelde birlikte değişir (bir sayaç değişkeni). Ama gerçek dünyadaki birçok tekrar, bir sayaçla değil bir **durumla** ilgilidir: kullanıcı geçerli bir değer girene kadar, bir bağlantı başarılı olana kadar, bir kuyruk boşalana kadar. Bu durumları `for` ile ifade etmek zorlama olurdu (`for (; kosul; )` yazıp başlangıç/artırım kısımlarını boş bırakmak gerekirdi). `while`, bu "sadece bir koşulum var" durumunu doğrudan ve okunaklı şekilde ifade eder. `do-while` ise buna ek

olarak "en az bir kez çalışmalı" ihtiyacını karşılar -- örneğin bir menüyü en az bir kez göstermek, kullanıcı girdisini en az bir kez almak gibi.

Tarihçe

`while` ve `do-while`, Java'nın ilk gününden (JDK 1.0, 1996) beri dilde var -- C ve C++'tan doğrudan miras alınan, C-tarzı dillerin ortak temel yapı taşlarından ikisidir. `for` döngüsü gibi bunlar da Java 5'teki Enhanced for Loop'tan çok önce geldi ve o zamandan beri sözdizimleri hiç değişmedi.

Temel while Sözdizimi

En temel haliyle `while`, koşul doğru olduğu sürece gövdeyi çalıştırır:

```
public class WhileBasicsExample {
    public static void main(String[] args) {
        int count = 1;

        // The condition is checked BEFORE each iteration -- if it's false
        // from the start, the body never runs at all.
        while (count <= 5) {
            System.out.println("count = " + count);
            count++;
        }

        int sum = 0;
        int n = 1;
        while (sum < 20) {
            sum += n;
            n++;
        }
        System.out.println("Stopped once sum reached " + sum + " (n was " + n + ")");
    }
}
```

Beklenen Çıktı

```
count = 1
count = 2
count = 3
count = 4
count = 5
Stopped once sum reached 21 (n was 7)
```

Dikkat: gövde içinde koşulu etkileyen bir şey (yukarıdaki örnekte `count++` ya da `sum += n`) mutlaka olmalı -- aksi halde döngü hiç bitmez (bkz. "Yaygın Hatalar").

do-while: En Az Bir Kez Çalışan Döngü

`do-while`, koşulu gövdeden **sonra** kontrol eder:

```
do {
    // en az bir kez çalışır
} while (koşul);
```

Bu, `while`'ın aksine, koşul en baştan yanlış olsa bile gövdenin en az bir kez çalışacağı anlamına gelir:

```
public class DoWhileBasicsExample {
    public static void main(String[] args) {
        int count = 10;

        // do-while checks the condition AFTER the body -- so the body runs
        // at least ONCE even though "count <= 5" is already false when the
        // loop starts.
        do {
            System.out.println("count = " + count);
            count++;
        } while (count <= 5);

        System.out.println("Loop ran exactly once, count is now " + count);
    }
}
```

Beklenen Çıktı

```
count = 10
Loop ran exactly once, count is now 11
```

Bu davranış özellikle "önce bir şey yap, sonra devam edip etmeyeceğine karar ver" kalıbına uyar -- örneğin bir kullanıcıdan girdi almak: girdiyi almadan önce onun geçerli olup olmadığını kontrol edemezsiniz.

while vs do-while: Ne Zaman Hangisi

İkisi arasındaki tek fark, koşulun **ne zaman** kontrol edildiğidir -- ama bu fark, koşul en baştan yanlışsa gövdenin hiç mi yoksa bir kez mi çalışacağını belirler:

```
public class WhileVsDoWhileExample {
    public static void main(String[] args) {
        int x = 100;

        System.out.println("while (condition is false from the start):");
        int whileRuns = 0;
        while (x < 10) {
            whileRuns++;
        }
        System.out.println("Body ran " + whileRuns + " times.");

        System.out.println("do-while (same condition, false from the start):");
        int doWhileRuns = 0;
        do {
            doWhileRuns++;
        } while (x < 10);
        System.out.println("Body ran " + doWhileRuns + " times.");
    }
}
```

Beklenen Çıktı

```
while (condition is false from the start):
Body ran 0 times.
do-while (same condition, false from the start):
Body ran 1 times.
```

Genel kural: gövdenin en az bir kez çalışması gerektiğini biliyorsanız (bir menü göstermek, bir girdi istemi vermek) `do-while` kullanın; aksi halde, yani "belki hiç çalışmayabilir" durumundaysanız `while` kullanın.

break ve continue ile while

`break` ve `continue`, `for` döngüsündeki ("for Loop" dersinde işlenen "break ile Döngüden Çıkmak" ve "continue ile Bir Adımı Atlamak" bölümlerine bakabilirsiniz) ile birebir aynı şekilde çalışır -- tek fark döngünün kendi başlık sözdizimidir:

```
public class BreakContinueInWhileExample {
    public static void main(String[] args) {
        // break and continue work exactly the same way in while/do-while as
        // they do in for (see the "for Loop" lesson) -- only the loop's own
        // header differs.
        int i = 0;
        System.out.println("break in a while loop:");
        while (true) {
            i++;
            if (i > 3) {
                break;
            }
            System.out.println("i = " + i);
        }

        System.out.println("continue in a while loop:");
        int j = 0;
        while (j < 10) {
            j++;
            if (j % 2 == 0) {
                continue;
            }
            System.out.println("odd j = " + j);
        }
    }
}
```

Beklenen Çıktı

```
break in a while loop:
i = 1
i = 2
i = 3
continue in a while loop:
odd j = 1
odd j = 3
odd j = 5
odd j = 7
odd j = 9
```

`break` döngüyü tamamen sonlandırır, `continue` ise yalnızca o anki yinelemeyi atlayıp koşul kontrolüne geri döner.

Kullanıcı Girdisiyle Doğrulama Döngüsü (Scanner ile)

`do-while`'ın en doğal kullanım alanlarından biri, kullanıcıdan geçerli bir değer gelene kadar tekrar tekrar sormaktır ("Scanner" dersindeki "Temel Kullanım: Token Okumak" bölümüne bakabilirsiniz):

```
import java.util.Scanner;

public class InputValidationLoopExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int age;

        // do-while is the natural fit here: you must ask AT LEAST once,
        // and keep re-asking only if the input turns out to be invalid.
        do {
            System.out.print("Enter your age (1-120): ");
            age = scanner.nextInt();

            if (age < 1 || age > 120) {
                System.out.println("Invalid age, try again.");
            }
        } while (age < 1 || age > 120);

        System.out.println("Valid age received: " + age);
        scanner.close();
    }
}
```

Burada `do-while` seçilmesinin nedeni açık: kullanıcıya en az bir kez sormanız gerekir -- girdiyi almadan onun geçerli olup olmadığını bilemezsiniz.

Uygulamalı Örnek: Sayı Tahmin Oyunu (Number Guessing Game)

Şimdiye kadar gördüğümüz her şeyi (do-while, koşullu doğrulama, Scanner ile girdi okuma) tek bir küçük ama eksiksiz programda birleştirelim -- klasik "sayı tahmin oyunu":

```
import java.util.Scanner;

public class NumberGuessingGameExample {
    public static void main(String[] args) {
        // In a real game this would be random (new Random().nextInt(100) + 1) --
        // fixed here so the walkthrough below is reproducible.
        int target = 42;
        Scanner scanner = new Scanner(System.in);
        int guess;
        int attempts = 0;

        System.out.println("Guess a number between 1 and 100.");

        do {
            System.out.print("Your guess: ");
            guess = scanner.nextInt();
            attempts++;

            if (guess < target) {
                System.out.println("Too low!");
            } else if (guess > target) {
                System.out.println("Too high!");
            } else {
                System.out.println("Correct! You guessed it in " + attempts + " attempts.");
            }
        } while (guess != target);

        scanner.close();
    }
}
```

Program, kullanıcı doğru sayıyı tahmin edene kadar sormaya devam eder -- bu da yine "kaç adımda biteceğini bilmiyorum, ama ne zaman biteceğini biliyorum" durumunun tipik bir örneğidir, tam olarak `do-while`'ın var olma nedenidir.

Best Practices

Döngü gövdesinde koşulu etkileyen değişkeni mutlaka güncelleyin -- güncellemeyi unutmak, kazara sonsuz döngü yaratmanın en yaygın nedenidir. Koşulu olabildiğince basit ve okunaklı tutun; karmaşık bir koşul gerekiyorsa, koşulu anlamlı isimli bir `boolean` değişkende toplamayı düşünün (`while (kayıtVar)` gibi). Gövdenin en az bir kez çalışması gerektiği her durumda `while` yerine `do-while`'ı tercih edin -- bu, "en başta bir kere dene" niyetini kodda açıkça ifade eder.

Yaygın Hatalar

En sık yapılan hata, koşulu etkileyen değişkeni güncellemeyi unutup kazara sonsuz döngü oluşturmaktır. İkinci yaygın hata, `while` ile `do-while` arasında yanlış seçim yapmaktır -- özellikle girdi doğrulama gibi "en az bir kez çalışmalı" durumlarında `while` kullanıp ilk kontrolü döngü dışında elle tekrarlamaya çalışmaktır (gereksiz kod tekrarına yol açar).

Üçüncü hata, `do-while`'ın kapanışındaki noktalı virgülü unutmaktır -- `} while (koşul)` satırının sonunda `;` zorunludur, aksi halde derleme hatası alınır.

Özet, Cheat Sheet ve Terimler Sözlüğü

`while`, koşulu **önce** kontrol eder; gövde hiç çalışmayabilir. `do-while`, koşulu **sonra** kontrol eder; gövde en az bir kez çalışır. `break` döngüyü tamamen bitirir, `continue` yalnızca o yinelemeyi atlar -- ikisi de `for` döngüsündekiyle aynı şekilde davranır. Girdi doğrulama gibi "en az bir kez dene" senaryolarında `do-while` doğal seçimdir.

```
// while: koşul önce kontrol edilir
while (koşul) {
    // ...
}

// do-while: koşul sonra kontrol edilir, gövde en az bir kez çalışır
do {
    // ...
} while (koşul);
```

Terimler Sözlüğü

- **while döngüsü:** Koşulu her yinelemeden önce kontrol eden, koşul doğru olduğu sürece çalışan döngü yapısı.
- **do-while döngüsü:** Koşulu her yinelemeden sonra kontrol eden, bu yüzden gövdesi en az bir kez çalışan döngü yapısı.
- **Girdi doğrulama döngüsü (input validation loop):** Kullanıcıdan geçerli bir değer alınana kadar tekrar tekrar soran döngü kalıbı.
- **Sonsuz döngü (infinite loop):** Koşulun hiçbir zaman yanlış olmadığı, bu yüzden hiç bitmeyen döngü -- genelde koşulu etkileyen bir güncellemenin unutulmasından kaynaklanır.