for Loop

Control Flow kategorisinin üçüncü topic'i: klasik üç bölümlü (init;condition;update) `for` sözdizimi, `break` ile erken çıkış, `continue` ile bir adımı atlama, kasıtlı/kasıtsız sonsuz döngüler, virgüllü init/update ile birden fazla değişken, ve klasik index tabanlı dizi gezinme.

Başlangıç 20 dk

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

---

for Loop

Kontrol Akışı kategorisinin üçüncü konusu for döngüsü -- bir kod bloğunu, önceden bilinen ya da sayılabilen bir sayıda TEKRARLAMAYI sağlayan kontrol yapısı. `if` / `else` (bkz. "if / else" dersi) ve `switch` (bkz. "switch" dersi) bir kod yolunu YALNIZCA BİR KEZ çalıştırıp çalıştırmayacağına karar verirken, `for` aynı bloğu BİRDEN FAZLA KEZ, bir sayaç değişkeninin ilerlemesine bağlı olarak çalıştırır. Bu ders, klasik `for` sözdizimini, `break` / `continue` ile döngü akışını kontrol etmeyi, ve bir dizi (array) üzerinde indeks tabanlı gezinmeyi ele alıyor.

for Döngüsü Nedir?

`for` döngüsü, tek bir satırda üç parçayı birleştirir: BAŞLANGIÇ (initialization -- yalnızca bir kez, döngü başlamadan önce çalışır), KOŞUL (condition -- her tekrardan önce kontrol edilir, `false` olduğunda döngü biter) ve GÜNCELLEME (update -- her tekrarın SONUNDA çalışır). `for (int i = 0; i < 5; i++)` biçimindeki bu üç parça noktalı virgülle ayrılır. Başlangıçta tanımlanan sayaç değişkeni (`i`), yalnızca döngünün kendi kapsamında (scope) vardır -- döngü bittikten sonra erişilemez.

Neden Var?

Aynı işlemi belirli bir sayıda (ya da bir koşul sağlanana kadar) tekrarlamak, programlamanın en temel ihtiyaçlarından biridir -- bir listenin her elemanını işlemek, bir işlemi N kez denemek, bir aralıktaki her sayıyı kontrol etmek. Bunu döngü OLMADAN yazmak, aynı kodu elle N kez kopyalamayı gerektirirdi -- hem N'in kendisi değişkense imkansız, hem de N sabit olsa bile bakımı zor ve hataya açık olurdu. `for`, sayaç tabanlı tekrarların BAŞLANGIÇ/KOŞUL /GÜNCELLEME mantığını tek bir yerde, kompakt bir şekilde toplar.

Tarihçe

`for` döngüsü de `if` / `switch` gibi C'den (1972) Java'ya (1995) neredeyse birebir geçti -- üç bölümlü (`init; condition; update`) sözdizimi C'nin doğrudan mirasıdır ve o günden beri değişmedi. Java 5 (2004), bu klasik `for`'un yanına dizi/koleksiyon elemanlarını indekssiz gezmek için "enhanced for" (`for-each`) sözdizimini EKLEDİ (bkz. "Enhanced for Loop" dersi) -- klasik `for`'u DEĞİŞTİRMEDİ, ikisi hâlâ yan yana, farklı amaçlar için var.

Temel for Döngüsü Sözdizimi

En yaygın kullanım, `0`'dan başlayıp bir sınıra kadar (`<` ya da `<=` ile) sayan, her adımda `1` artan (`i++`) bir sayaçtır -- ama üç bölüm de tamamen esnektir: `i--` ile geriye sayılabilir, farklı bir adım büyüklüğü (`i += 2`) kullanılabilir.

```
public class ForLoopBasicsExample {
    public static void main(String[] args) {
        // The three parts of a for loop, separated by semicolons:
        // initialization (runs once) ; condition (checked before each iteration) ; update
(runs after each iteration)
        for (int i = 1; i <= 5; i++) {
            System.out.println("i = " + i);
        }

        // The loop variable declared in the header is scoped to the loop --
        // it does not exist after the loop ends.
        int total = 0;
        for (int i = 1; i <= 5; i++) {
            total += i;
        }
        System.out.println("Sum 1..5 = " + total);

        // Counting downward works the same way, with a decrementing update.
        for (int i = 5; i >= 1; i--) {
            System.out.print(i + " ");
        }
        System.out.println();
    }
}
```

Beklenen Çıktı

```
i = 1
i = 2
i = 3
i = 4
i = 5
Sum 1..5 = 15
5 4 3 2 1
```

break ile Döngüden Çıkmak

`break`, bir döngüyü KOŞULUN hâlâ `true` olup olmadığına BAKMAKSIZIN anında sonlandırır -- kontrol akışı döngüden hemen sonraki satıra atlar. En yaygın kullanımı, bir ARAMA'da aranan şey bulunduğunda kalan elemanları kontrol etmeye devam etmenin gereksiz olduğu durumlardır.

```
public class BreakExample {
    public static void main(String[] args) {
        int[] numbers = {4, 9, 15, 23, 42, 7};
        int target = 23;
        int foundIndex = -1;

        for (int i = 0; i < numbers.length; i++) {
            if (numbers[i] == target) {
                foundIndex = i;
                // break immediately exits the loop -- no point checking the
                // remaining elements once the target is found.
                break;
            }
        }

        if (foundIndex != -1) {
            System.out.println("Found " + target + " at index " + foundIndex);
        } else {
            System.out.println(target + " not found");
        }

        // Without break, every iteration would run even after finding a
        // match -- wasted work, and the LAST match would silently win
        // instead of the first one.
        int lastMatchIndex = -1;
        for (int i = 0; i < numbers.length; i++) {
            if (numbers[i] > 10) {
                lastMatchIndex = i;
            }
        }
        System.out.println("Last index > 10 (no break): " + lastMatchIndex);
    }
}
```

Beklenen Çıktı

```
Found 23 at index 3
Last index > 10 (no break): 4
```

continue ile Bir Adımı Atlamak

`continue`, mevcut tekrarın GERİ KALANINI atlar ve doğrudan GÜNCELLEME adımına (ardından tekrar KOŞUL kontrolüne) geçer -- döngünün kendisi `break`'in aksine SONA ERMEZ, yalnızca o TEK adım kısa kesilir. Yaygın bir kullanım, bir koleksiyondaki geçersiz /istenmeyen elemanları atlayıp geri kalanı işlemeye devam etmektir.

```
public class ContinueExample {
    public static void main(String[] args) {
        System.out.println("Odd numbers from 1 to 10:");
        for (int i = 1; i <= 10; i++) {
            // continue skips the REST of this iteration's body and jumps
            // straight to the update step -- the loop keeps running, unlike
            // break which would exit it entirely.
            if (i % 2 == 0) {
                continue;
            }
            System.out.print(i + " ");
        }
        System.out.println();

        // A common real use: skipping invalid entries while summing valid ones.
        int[] scores = {85, -1, 92, -1, 78, 60};
        int sum = 0;
        int validCount = 0;

        for (int i = 0; i < scores.length; i++) {
            if (scores[i] < 0) {
                continue; // -1 marks a missing score -- skip it
            }
            sum += scores[i];
            validCount++;
        }

        System.out.println("Valid scores: " + validCount + ", average: " + (sum /
validCount));
    }
}
```

Beklenen Çıktı

```
Odd numbers from 1 to 10:
1 3 5 7 9
Valid scores: 4, average: 78
```

`break` "döngüyü tamamen bırak", `continue` ise "yalnızca bu adımı atla, döngüye devam et" anlamına gelir -- ikisini karıştırmak yaygın bir kavramsal hatadır (bkz. "Yaygın Hatalar").

Sonsuz Döngüler

Üç bölümün de (`init`, `condition`, `update`) TAMAMEN atlanabildiği `for (;;)` biçimi, kendiliğinden asla bitmeyen KASITLI bir sonsuz döngü oluşturur -- yalnızca gövde içindeki bir `break` (ya da bir `return`) onu durdurabilir. Bu, "ne zaman duracağını önceden bilmediğin" durumlarda (örn. kullanıcıdan geçerli bir girdi gelene kadar sor) kullanışlıdır.

```
public class InfiniteLoopExample {
    public static void main(String[] args) {
        // Leaving out all three parts (for (;;)) creates a loop with no
        // built-in stopping condition -- it is intentionally infinite unless
        // something INSIDE the body breaks out of it.
        int attempts = 0;

        for (;;) {
            attempts++;
            System.out.println("Attempt " + attempts);

            if (attempts >= 3) {
                System.out.println("Condition met -- breaking out.");
                break;
            }
        }

        System.out.println("Done after " + attempts + " attempts.");

        // A common ACCIDENTAL infinite loop: forgetting the update step, or
        // writing an update that never makes the condition false. This is
        // shown as a comment, not run, because it would hang forever:
        //
        // for (int i = 0; i < 5; ) {
        //     System.out.println(i);
        //     // missing i++ -- the condition (i < 5) is NEVER updated
        // }
    }
}
```

Beklenen Çıktı

```
Attempt 1
Attempt 2
Attempt 3
Condition met -- breaking out.
Done after 3 attempts.
```

> Kasıtsız (yani bir hatadan kaynaklanan) sonsuz döngüler gerçek bir programın ÇÖKMESİNE ya da DONMASINA yol açar -- en yaygın nedeni, güncelleme adımının koşulu ASLA `false` yapmayacak şekilde yazılmasıdır (örn. `i++` yazmayı unutmak, ya da koşulu yanlışlıkla her zaman `true` kalacak şekilde kurmak). Her `for` yazarken güncelleme adımının gerçekten koşulu bir noktada `false` yapacağından emin olunmalıdır.

Birden Fazla Değişkenle for

`for`'un başlangıç ve güncelleme bölümleri, virgülle ayrılmış BİRDEN FAZLA ifade içerebilir -- bu, iki değişkenin BİRLİKTE ilerlemesi gerektiğinde (örn. bir dizinin iki ucundan ortaya doğru tarama) kullanışlıdır.

```
public class MultipleVariablesForExample {
    public static void main(String[] args) {
        // A for loop's init and update sections can each hold more than one
        // statement, separated by commas -- useful for two variables that
        // move together, like scanning from both ends toward the middle.
        int[] numbers = {10, 20, 30, 40, 50, 60, 70};

        for (int left = 0, right = numbers.length - 1; left < right; left++, right--) {
            System.out.println("left=" + left + " (" + numbers[left] + "), right=" + right
+ " (" + numbers[right] + ")");
        }

        // Checking whether an array is a palindrome uses the exact same pattern.
        int[] palindrome = {1, 2, 3, 2, 1};
        boolean isPalindrome = true;

        for (int left = 0, right = palindrome.length - 1; left < right; left++, right--) {
            if (palindrome[left] != palindrome[right]) {
                isPalindrome = false;
                break;
            }
        }

        System.out.println("Is palindrome: " + isPalindrome);
    }
}
```

Beklenen Çıktı

```
left=0 (10), right=6 (70)
left=1 (20), right=5 (60)
left=2 (30), right=4 (50)
Is palindrome: true
```

for Döngüsüyle Dizi Üzerinde Gezinme

Klasik `for`, bir dizinin her adımda İNDEKSİNİ de sağlar -- bu, yalnızca değerlere değil (bkz. "Arrays" dersindeki "Temel Kullanım: Oluşturma, Erişim, Varsayılan Değerler" bölümü) o değerlerin KONUMUNA da ihtiyaç duyulduğunda (örn. bir diziyi YERİNDE değiştirmek) gereklidir. Bunun neden önemli olduğu -- ve indekse ihtiyaç duyulmadığında daha kısa bir alternatifin var olduğu -- gelecek "Enhanced for Loop" dersinde ele alınacak.

```
public class ArrayIterationForExample {
    public static void main(String[] args) {
        int[] scores = {70, 85, 92, 60, 78};

        // The classic index-based for loop gives you the INDEX at every
        // step, not just the value -- useful for printing positions, or
        // comparing neighboring elements.
        for (int i = 0; i < scores.length; i++) {
            System.out.println("scores[" + i + "] = " + scores[i]);
        }

        // Because you have the index, you can also MODIFY the array in
        // place -- something a value-only loop cannot do (see "Enhanced for
        // Loop" for why).
        for (int i = 0; i < scores.length; i++) {
            scores[i] = scores[i] + 5; // curve every score by +5
        }

        System.out.print("Curved scores: ");
        for (int i = 0; i < scores.length; i++) {
            System.out.print(scores[i] + " ");
        }
        System.out.println();
    }
}
```

Beklenen Çıktı

```
scores[0] = 70
scores[1] = 85
scores[2] = 92
scores[3] = 60
scores[4] = 78
Curved scores: 75 90 97 65 83
```

Best Practices

- **Döngü sayacının kapsamını (scope) mümkün olduğunca döngüyle sınırlı tutun** -- sayacı `for`'un kendi başlangıç bölümünde tanımlamak, döngü dışında yanlışlıkla kullanılmasını önler.
- **`break`'i yalnızca gerçekten erken çıkmak GEREKTİĞİNDE kullanın** -- gereksiz `break` kullanımı akışı takip etmeyi zorlaştırabilir.

- **`continue` yerine, mümkünse koşulu TERSİNE çevirip erken atlamayı düşünün** -- bazen okunabilirlik açısından daha net olur, ama her iki yaklaşım da geçerlidir.
- **Bir `for (;;)` yazıyorsanız, döngüyü durduracak `break` koşulunun MUTLAKA var olduğundan emin olun.**
- **Güncelleme adımının koşulu bir noktada `false` yapacağını her zaman doğrulayın** -- kasıtsız sonsuz döngülerin en sık nedeni budur.

Yaygın Hatalar

- **Güncelleme adımını (`i++` gibi) yazmayı unutup kasıtsız bir sonsuz döngü yaratmak.** Koşul asla `false` olmaz, program donar.
- **`break` ile `continue`'yu karıştırmak.** `break` döngüyü tamamen bitirir, `continue` yalnızca mevcut adımı atlar -- ikisi çok farklı davranışlardır.
- **Bir dizinin sınırını `<=` ile `.length` değerine kadar kontrol edip `ArrayIndexOutOfBoundsException` almak.** Geçerli indeksler `0`'dan `length - 1`'e kadardır, bu yüzden koşul `i < dizi.length` olmalıdır, `i <= dizi.length` DEĞİL.
- **Döngü içinde tanımlanan bir sayacı döngü dışında kullanmaya çalışmak.** Sayaç yalnızca `for`'un kendi kapsamında var olur, derleme hatası alınır.

Özet, Cheat Sheet ve Terimler Sözlüğü

`for` döngüsü, BAŞLANGIÇ/KOŞUL/GÜNCELLEME'yi tek bir satırda birleştiren, sayaç tabanlı bir tekrar yapısıdır. `break` döngüyü tamamen sonlandırır; `continue` yalnızca mevcut adımı atlar. Üç bölümü de atlayan `for (;;)` kasıtlı bir sonsuz döngü oluşturur, yalnızca içindeki bir `break` onu durdurabilir. Başlangıç/güncelleme bölümleri virgülle birden fazla ifade içerebilir. Klasik `for`, bir dizi üzerinde gezinirken hem DEĞERE hem İNDEKSE erişim sağlar.

Hızlı referans:

```
for (int i = 0; i < 5; i++) {
    // ...
}

for (int i = 0; ; i++) {
    if (kosul) break;
}

for (int i = 0; i < dizi.length; i++) {
    if (dizi[i] < 0) continue;
    // ...
}

for (int i = 0, j = dizi.length - 1; i < j; i++, j--) {
    // ...
}
```

Terimler Sözlüğü

Initialization (Başlangıç) — `for`'un yalnızca bir kez, döngü başlamadan önce çalışan ilk bölümü.

Condition (Koşul) — Her tekrardan önce kontrol edilen, `false` olduğunda döngüyü sonlandıran `boolean` ifade.

Update (Güncelleme) — Her tekrarın sonunda çalışan, genellikle sayacı ilerleten bölüm.

break — Bir döngüyü koşuldan bağımsız olarak anında sonlandıran anahtar kelime.

continue — Mevcut tekrarın geri kalanını atlayıp doğrudan güncelleme adımına geçen anahtar kelime.