Generic Metotlar

Sınıfının generic olup olmamasından bağımsız olarak kendi tür parametresini bildiren generic metotlar -- metot tür parametreleri, birden fazla tür parametresi, tür çıkarımı (type inference) ve açık tür tanığı (type witness). Generics serisinin 2.'si.

Orta 20 dk

Oluşturulma tarihi: 2 Ekim 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/generic-methods

---

"Generics'e Giriş", tür parametrelerini bütün bir SINIFA bağladı — bir `Box<T>`'deki her metot aynı `T`'yi paylaşır. Ama pek çok yararlı generic kod, aslında hiçbir sınıfa ait değildir: hangi sınıfın içinde olduğundan bağımsız olarak herhangi bir türle çalışan, tek başına duran bir yardımcı metot. Bu ders, tek bir METODA kendi tür parametresini vermeyi ele alıyor.

Generic Metot Nedir?

Generic metot, dönüş türünden hemen önce açılı parantezler içinde yazılan kendi tür parametresini bildiren bir metottur — `static <T> T firstElement(...)`. Bu tür parametresi yalnızca metoda aittir: çevresindeki sınıfın generic olup olmamasıyla hiçbir ilgisi yoktur, ve her tek çağrıda yeni bir değer alır.

Neden Var?

Her yararlı generic davranış doğal olarak generic bir sınıfa ait değildir. "Herhangi bir listenin ilk elemanını ver" gibi bir yardımcı metot, aslında bir `Utils` sınıfının bir türle parametrelenmesiyle ilgili değildir — TAM OLARAK BU METODUN, sıradan, generic olmayan bir sınıftan çağrılsa bile herhangi bir tür için çalışması gerekmesiyle ilgilidir. Generic metotlar, bu esnekliğin tam olarak ihtiyaç duyulduğu yerde, metot seviyesinde yaşamasına izin verir.

Generic Metot Bildirmek

Tür parametresi bir kez, dönüş türünden hemen önce görünür, ve daha sonra o metodun parametre listesinde, gövdesinde ya da dönüş türünde herhangi bir yerde kullanılabilir.

```
import java.util.List;

public class GenericMethodBasicsExample {

    // A generic method: the <T> right before the return type is the
    // method's OWN type parameter -- declared here because this method
    // lives in an ordinary, non-generic class. Nothing about
    // GenericMethodBasicsExample itself is generic; only this one method is.
    static <T> T firstElement(List<T> list) {
        return list.get(0);
    }

    public static void main(String[] args) {
        List<String> names = List.of("Alice", "Bob");
        String first = firstElement(names); // T is deduced to be String here

        List<Integer> numbers = List.of(10, 20, 30);
        int firstNumber = firstElement(numbers); // same method, T is Integer this time

        System.out.println(first);
        System.out.println(firstNumber);
    }
}
```

`firstElement(...)`, tamamen sıradan, generic olmayan bir sınıf olan `GenericMethodBasicsExample`'ın içinde yaşıyor — ama metodun kendisi tamamen generic. Onu bir `List<String>` ile çağırmak `T`'yi `String` olarak çıkarır; bir `List<Integer>` ile çağırmak, aynı metotta `T`'yi `Integer` olarak çıkarır.

Birden Fazla Tür Parametresi

Bir metot, tıpkı generic bir sınıfın yapabildiği gibi, virgülle ayrılmış birden fazla tür parametresi bildirebilir.

```
public class MultipleTypeParametersMethodExample {

    // A method can declare as many type parameters as it needs, separated
    // by commas, exactly like a generic class can -- K and V here are
    // completely independent of each other and get deduced separately.
    static <K, V> String describeEntry(K key, V value) {
        return key + " -> " + value;
    }

    public static void main(String[] args) {
        System.out.println(describeEntry("age", 30));         // K=String, V=Integer
        System.out.println(describeEntry(101, "order-created")); // K=Integer, V=String
        System.out.println(describeEntry(true, 3.14));         // K=Boolean, V=Double
    }
}
```

`describeEntry(K key, V value)`, her çağrıda `K` ve `V`'yi birbirinden bağımsız olarak çıkarır — `describeEntry("age", 30)` ve `describeEntry(101, "order-created")`, ikisi de aynı metodun geçerli, birbiriyle ilgisiz kullanımları, her biri kendi çıkarılmış tür çiftiyle.

Tür Çıkarımı (Type Inference)

Şimdiye kadar gördüğün neredeyse her çağrıda, derleyici tür parametresini geçirdiğin argümanlardan tamamen kendi başına buldu — buna TÜR ÇIKARIMI denir. Tür parametresinin ne olduğunu neredeyse hiçbir zaman açıkça belirtmen gerekmez.

```
import java.util.List;

public class TypeInferenceExample {

    static <T> T firstElement(List<T> list) {
        return list.get(0);
    }

    public static void main(String[] args) {
        List<String> names = List.of("Alice", "Bob");

        // Normally, the compiler infers T entirely from the argument --
        // no need to spell it out.
        String inferred = firstElement(names);

        // The same call, but with an explicit TYPE WITNESS: the compiler
        // is told exactly what T is instead of figuring it out. This is
        // rarely necessary -- it exists for the (uncommon) cases where
        // there isn't enough information at the call site for inference
        // to work out T on its own.
        String explicit = TypeInferenceExample.<String>firstElement(names);

        System.out.println(inferred);
        System.out.println(explicit);
    }
}
```

Açık form olan `TypeInferenceExample.<String>firstElement(names)`'e TÜR TANIĞI (type witness) denir — derleyiciye `T`'nin ne olması gerektiğini çıkarmasına izin vermek yerine tam olarak söyler. Bu örnekteki iki çağrı da birebir aynı sonucu üretir; tanık formu, derleyicinin çağrı noktasında türü kendi başına çıkarmaya yetecek bilgiye sahip olmadığı (nadir) durumlar için var.

> Günlük kodda, derleyici gerçekten tanıksız şikayet etmedikçe asla bir tür tanığı yazma. `firstElement(names)` deyimseldir; `TypeInferenceExample.<String>firstElement(names)`, vakaların büyük çoğunluğunda derleyicinin ihtiyaç duymadığı, ayrıntılı bir gürültüdür.

Bir Metodun Tür Parametresi vs. Sınıfının Tür Parametresi

Generic bir metot generic bir sınıfın içinde yaşadığında, hangi tür parametresinin hangisi olduğu konusunda net olmakta fayda var — metot, sınıfınkinden tamamen ayrı, kendi tür parametresini bildirebilir.

```
public class GenericMethodInGenericClassExample {

    // Container<T> is a generic CLASS -- T belongs to the class itself,
    // fixed once for the whole instance (Container<String> means every T
    // in this instance is a String).
    static class Container<T> {
        private final T value;

        Container(T value) {
            this.value = value;
        }

        T getValue() {
            return value;
        }

        // combineWith declares its OWN type parameter, U -- completely
        // separate from the class's T. A single Container<String> instance
        // can call combineWith with an Integer, a Boolean, anything --
        // U is decided fresh on every call, independent of what T is.
        <U> String combineWith(U other) {
            return value + " + " + other;
        }
    }

    public static void main(String[] args) {
        Container<String> box = new Container<>("hello");

        System.out.println(box.combineWith(42));     // U = Integer
        System.out.println(box.combineWith(true));    // U = Boolean
        System.out.println(box.combineWith("world")); // U = String, unrelated to T being
String too
    }
}
```

`Container<T>`, `T`'yi bir kez, tüm instance için sabitler — bir `Container<String>` her zaman bir `String` tutar. Ama `combineWith`'in `U`'su her tek çağrıda yeni baştan belirlenir, `T`'den tamamen bağımsız olarak — aynı `Container<String>` instance'ı `combineWith`'i önce bir `Integer`, sonra bir `Boolean`, sonra bir `String` ile çağırır, ve her çağrı kendi `U`'sunu alır.

Pratik Bir Generic Metot

Generic metotlar, günlük yardımcı kodda yaygındır — tam olarak aynı mantığın herhangi bir türden bir array ya da koleksiyona uygulanması gereken her yerde.

```
public class PracticalArraySwapExample {

    // A single, practical generic method that works on an array of ANY
    // reference type -- no casting, no duplicating this method once per
    // element type.
    static <T> void swap(T[] array, int i, int j) {
        T temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    public static void main(String[] args) {
        String[] names = {"Alice", "Bob", "Charlie"};
        swap(names, 0, 2);
        System.out.println(String.join(", ", names));

        Integer[] numbers = {1, 2, 3};
        swap(numbers, 0, 1); // the exact same method, now working on Integer[]
        for (int n : numbers) {
            System.out.print(n + " ");
        }
    }
}
```

`swap(...)`, bir `String[]` ve bir `Integer[]` üzerinde birebir aynı şekilde çalışır — bir kez yazılmış tek bir metot, hiçbir cast olmadan ve uyuşmayan türden elemanları yanlışlıkla değiştirme riski olmadan.

Best Practices

- Generic davranış bir sınıfın tutacağı bütün bir durum ailesine değil, tek bir işleme aitse, generic bir sınıf yerine generic bir metodu tercih et.
- Tür çıkarımının işini yapmasına izin ver — derleyici türü gerçekten kendi başına çıkaramadığında açık bir tür tanığına başvur.
- Dil buna izin verse de, metot seviyesindeki bir tür parametresine, çevreleyen sınıfın aynı isimli bir tür parametresini gölgeleyen bir isim verme — iç içe bir kapsamda bir değişken adını yeniden kullanmak kadar kafa karıştırıcı okunur.
- Generic bir metodun tür parametresi listesini, işlemin gerçekten gerektirdiği kadar küçük tut.

Yaygın Hatalar

- Dönüş türünden önceki `<T>` bildirimini unutup `static T firstElement(...)` yazmak — bu derlenmez, çünkü `T` bildirilmemiş bir tür olurdu.
- Çıkarım zaten türü kendi başına doğru çözerken, alışkanlıktan her generic metot çağrısına bir tür tanığı eklemek.

- `combineWith`'in `U`'sunun gösterdiği gibi ikisi tamamen bağımsızken, generic bir metodun tür parametresinin bir şekilde çevreleyen sınıfının tür parametresine bağlı olduğunu varsaymak.
- Yalnızca metotlarından biri gerçekten bir tür parametresine ihtiyaç duyarken, bütün bir sınıfı generic yapmak — aksi hâlde sıradan bir sınıftaki generic bir metot genelde daha basit, daha doğru bir tasarımdır.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Generic bir metot, sınıfının generic olup olmamasından bağımsız olarak, dönüş türünden hemen önce açılı parantezler içinde kendi tür parametresini bildirir.
- Generic metotlar, bütün bir generic sınıfa ait olmayan mantık için, tür-parametreli davranışın metot seviyesinde yaşamasına izin verir.
- Bir metot, her çağrıda birbirinden bağımsız olarak çıkarılan, virgülle ayrılmış birden fazla tür parametresi bildirebilir.
- Tür çıkarımı, neredeyse her durumda generic bir metodun tür parametresini argümanlarından çözer; açık bir tür tanığına nadiren ihtiyaç duyulur.
- Bir metodun kendi tür parametresi (`combineWith`'teki `U` gibi), çevreleyen sınıfının tür parametresinden (`Container<T>`'deki `T` gibi) tamamen ayrıdır.

Cheat Sheet

```
// Sıradan bir sınıfta generic metot
class Utils {
    static <T> T firstElement(List<T> list) { return list.get(0); }
}
String first = Utils.firstElement(names); // T, String olarak çıkarılır

// Birden fazla tür parametresi
static <K, V> String describeEntry(K key, V value) { return key + " -> " + value; }

// Açık tür tanığı (nadiren gerekli)
String first = Utils.<String>firstElement(names);

// Sınıftan bağımsız metot tür parametresi
class Container<T> {
    <U> String combineWith(U other) { ... } // U != T
}
```

Terimler Sözlüğü

- **Generic metot**: çevreleyen sınıfının generic olup olmamasından bağımsız olarak kendi tür parametresini bildiren bir metot.
- **Tür çıkarımı (type inference)**: derleyicinin, generic bir metodun tür parametresini çağrı noktasında geçirilen argümanlardan çıkarması.

- **Tür tanığı (type witness)**: generic bir metodun çağrı noktasında sağlanan, çıkarımı geçersiz kılan açık bir tür argümanı.
- **Metot seviyesi tür parametresi**: bir metodun kendisinde bildirilen, çevreleyen sınıfının bildirdiği herhangi bir tür parametresinden ayrı bir tür parametresi.