Sınırlı Tür Parametreleri

Üst sınırlar (`<T extends ...>`), `&` ile birleştirilen birden fazla sınır, sınıflarla ve interface'lerle sınırlama, ve bir tür parametresini sınırlamanın tür parametresine hangi metotların çağrılabileceğini nasıl genişlettiği. Generics serisinin 3.'sü.

Orta 20 dk

Son güncelleme: 7 Kasım 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/bounded-type-parameters

---

Şimdiye kadar yazdığın her tür parametresi — `Box<T>`'deki `T`, `firstElement(...)`'teki `T` — SINIRSIZDI (unbounded): literal olarak herhangi bir türle doldurulabilirdi. Ama bu esnekliğin gerçek bir bedeli var, şimdi göreceğin gibi: sınırsız bir `T`, hakkında bilebileceğin EN AZ şeydir de aynı zamanda. Bu ders, bunu bilinçli olarak daraltmayı ele alıyor.

Sınırlı (Bounded) Tür Parametresi Nedir?

Sınırlı bir tür parametresi, hangi türlerin onu doldurmasına izin verildiğini kısıtlar — her şeyi kabul etmek yerine, yalnızca belirtilen bir gereksinimi karşılayan bir türü (ya da onun bir alt türünü) kabul eder. Bu kısıtlama karşılığında, derleyici artık o tür parametresinin bir değerinin gerçekte ne yapabileceği hakkında daha fazla şey bilir, ve sınırsız bir `T`'nin asla izin vermeyeceği metotları onun üzerinde çağırmana izin verir.

Neden Var?

Sınırsız bir `T`, kesinlikle her şey olabilir, bu yüzden derleyici yalnızca her `Object`'in sahip olduğu metotlara sahip olduğunu varsayabilir — `toString()`, `equals(...)`, ve daha spesifik hiçbir şey.

```
public class UnboundedMethodCallLimitationExample {

    // With an UNBOUNDED type parameter, the compiler only knows T could be
    // literally anything -- so the only methods it will let you call on a
    // T value are the ones every single Object has (toString, equals,
    // hashCode, ...). Nothing more specific is available.
    static <T> String describe(T value) {
        return value.toString(); // fine -- toString() belongs to Object

        // return value.doubleValue(); // would NOT compile -- the compiler
        //                                has no idea T even has a
        //                                doubleValue() method, because an
        //                                unbounded T might not.
    }

    public static void main(String[] args) {
        System.out.println(describe(42));
        System.out.println(describe("hello"));
        System.out.println(describe(3.14));
    }
}
```

Beklenen Çıktı

```
42
hello
3.14
```

`describe(...)`, her `Object`'in bir tanesi olduğu için `value.toString()`'i çağırabilir, ama bunun ötesinde hiçbir şey mevcut değil — sayılara özgü bir metodu, karşılaştırmaya özgü bir metodu ya da başka bir şeyi çağırmanın hiçbir yolu yok, çünkü sınırsız bir `T`, derleyiciye böyle bir garanti vermez. Sınırlı tür parametreleri, tam olarak bu garantiyi mümkün kılmak için var.

extends ile Üst Sınırlar (Upper Bounds)

`<T extends SomeType>` yazmak bir ÜST SINIR (upper bound) bildirir: `T`, `SomeType`'ın kendisi ya da onun alt türlerinden biri OLMAK ZORUNDADIR — bu ailenin dışındaki hiçbir şeye izin verilmez. Sınır bir interface olduğunda bile, yalnızca bir sınıf olduğunda değil, anahtar kelime `extends`tir.

```
import java.util.List;

public class UpperBoundedSumExample {

    // "T extends Number" is an UPPER BOUND: T can be Number itself or any
    // subclass of it (Integer, Double, Long, ...) -- nothing else is
    // allowed. In exchange, the compiler now knows every T value has every
    // method Number declares, like doubleValue().
    static <T extends Number> double sum(List<T> numbers) {
        double total = 0;
        for (T number : numbers) {
            total += number.doubleValue(); // only legal because of the bound
        }
        return total;
    }

    public static void main(String[] args) {
        System.out.println(sum(List.of(1, 2, 3)));       // T = Integer
        System.out.println(sum(List.of(1.5, 2.5)));       // T = Double

        // sum(List.of("a", "b")); // would NOT compile -- String isn't a Number
    }
}
```

Beklenen Çıktı

```
6.0
4.0
```

`T extends Number` ile `sum(List<T> numbers)`, her eleman üzerinde `number.doubleValue()`'yu çağırabilir, çünkü sınır, olası her `T`'nin — `Integer`, `Double`, `Long` ya da başka herhangi bir `Number` alt türü — bu metoda sahip olduğunu garanti eder. `sum(...)`'u bir `List<String>` ile çağırmak basitçe derlenmez, çünkü `String` bir `Number` değildir.

Birden Fazla Sınır

Bir tür parametresi, `&` ile birleştirilmiş birden fazla gereksinimle aynı anda sınırlanabilir. Sınırlardan en fazla biri bir sınıf olabilir, ve varsa, ilk sırada gelmelidir; geri kalanlar interface olmalıdır.

```
import java.util.List;

public class MultipleBoundsExample {

    // Multiple bounds are joined with "&": T must satisfy ALL of them at
    // once. At most one bound may be a class (and it must come first if
    // present); the rest must be interfaces. Here T must be both a Number
    // AND Comparable to itself, so the method can use doubleValue() from
    // Number and compareTo(...) from Comparable in the same method.
    static <T extends Number & Comparable<T>> T max(List<T> values) {
        T largest = values.get(0);
        for (T value : values) {
            if (value.compareTo(largest) > 0) {
                largest = value;
            }
        }
        return largest;
    }

    public static void main(String[] args) {
        System.out.println(max(List.of(3, 7, 2, 9, 4)));      // T = Integer
        System.out.println(max(List.of(1.5, 3.2, 0.8)));       // T = Double
    }
}
```

Beklenen Çıktı

```
9
3.2
```

`<T extends Number & Comparable<T>>`, `T`'nin hem bir `Number` HEM DE kendisiyle karşılaştırılabilir olmasını gerektirir — metot gövdesi, aynı değer üzerinde hem `doubleValue()`'yu (`Number` sınırından) hem `compareTo(...)`'yu (`Comparable` sınırından) serbestçe çağırabilir.

Sınıfla Sınırlamak

Sınır yalnızca bir metotta görünmek zorunda değil — generic bir SINIFIN tür parametresi de sınırlanabilir, o sınıfın her kullanımını aynı şekilde kısıtlar.

```
public class BoundedGenericClassExample {

    // The bound can be declared on a CLASS's type parameter too, not just
    // a method's -- every use of NumericBox is now restricted to Number
    // subtypes, and every method inside the class can rely on that.
    static class NumericBox<T extends Number> {
        private final T value;

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

        boolean isPositive() {
            return value.doubleValue() > 0; // legal, thanks to the class-level bound
        }

        T getValue() {
            return value;
        }
    }

    public static void main(String[] args) {
        NumericBox<Integer> intBox = new NumericBox<>(42);
        System.out.println(intBox.isPositive());

        NumericBox<Double> doubleBox = new NumericBox<>(-3.5);
        System.out.println(doubleBox.isPositive());

        // NumericBox<String> stringBox = new NumericBox<>("hi"); // would NOT
        //     compile -- String does not extend Number, so it fails the bound.
    }
}
```

Beklenen Çıktı

```
true
false
```

`NumericBox<T extends Number>`, `NumericBox<String>`'in basitçe yazılamayacağı anlamına gelir — derlenmez, çünkü `String` sınırı karşılamaz. `NumericBox`'ın içindeki her metot, tıpkı yukarıda `sum(...)`'un yapabildiği gibi, `value`'nun `Number`'ın metotlarına sahip olduğuna güvenebilir.

Interface ile Sınırlamak

Bir sınırın bir sınıfa hiç ihtiyacı yok — yalnızca bir interface ile sınırlamak da en az sınıfla sınırlamak kadar yaygındır, ve genelde daha geneldir, çünkü belirli bir tür hiyerarşisine bağlı değildir.

```
import java.util.List;

public class PracticalMaxFinderExample {

    // Bounding by an INTERFACE alone (no class involved) is just as
    // common as bounding by a class -- here T is only required to be
    // Comparable to itself, which is enough for a general-purpose "find
    // the largest element" utility that works for String, Integer, or any
    // other Comparable type, not just numbers.
    static <T extends Comparable<T>> T max(List<T> items) {
        T largest = items.get(0);
        for (T item : items) {
            if (item.compareTo(largest) > 0) {
                largest = item;
            }
        }
        return largest;
    }

    public static void main(String[] args) {
        System.out.println(max(List.of("banana", "apple", "cherry"))); // T = String
        System.out.println(max(List.of(5, 1, 9, 3)));                  // T = Integer
    }
}
```

Beklenen Çıktı

```
cherry
9
```

`<T extends Comparable<T>>`, kendisini aynı türden bir başkasıyla karşılaştırabilen herhangi bir türü kabul eder — `String`, `Integer` ve pek çok kendi sınıfın, `Number` ile hiçbir ilişkisi olmadan bu koşulu sağlar. Bu, "Wildcard'lar" dersi ilgili ama farklı bir amaç için `<? extends T>`'yi tanıttığında yoğun biçimde kullanıldığını göreceğin aynı şekildir — bir tür parametresini sınırlamak ile bir wildcard'ı sınırlamak aynı `extends` anahtar kelimesini kullanır, ama farklı sorulara cevap verir.

> `<T extends Comparable<T>>`, gerçek Java kodunda göreceğin en yaygın sınırlardan biridir — tek bir generic metodun, yalnızca sayılar için değil, herhangi bir karşılaştırılabilir tür için bir maksimum, bir minimum ya da bir sıralama hesaplamasına izin veren tam olarak budur.

Best Practices

- Generic kodun `Object`'in sunduğunun ötesinde bir metot çağırması gerektiği anda bir sınır ekle — sessizce daha fazlasına ihtiyaç duyan sınırsız bir tür parametresi, sınırın gereksiz olduğunun değil, unutulduğunun bir işaretidir.

- Gereksinim gerçekten "bu işlemi yapabilmek" ise, "literal olarak bu tür ya da onun bir alt türü olmak zorunda" değilse, somut bir sınıf yerine bir interface ile (`Comparable<T>` gibi) sınırlamayı tercih et.
- Sınırları birleştirirken, varsa sınıfın ilk sırada gelmesi, ardından interface'lerin gelmesi ve hepsinin `&` ile birleştirilmesi gerektiğini unutma.
- Bir sınırı, metodun ya da sınıfın gerçekten gerektirdiği kadar dar tut — yalnızca `toString()`'i çağırdığında `Number` ile sınırlamak hiçbir şey kazandırmaz ve çağıranları gereksiz yere kısıtlar.

Yaygın Hatalar

- Sınırı tamamen unutup, gerçekçi her argümanın sahip olacağını bildiğin bir metoda yapılan bir çağrıyı derleyicinin reddetmesine şaşırmak.
- `<T extends Comparable & Number>`'ı interface önce gelecek şekilde yazmak — bu derlenmez; varsa bir sınıf sınırı her zaman ilk sırada gelmelidir.
- Bir sınırın, tür parametresinin yerine hangi türlerin geçebileceğini kısıtlamak yerine, tür parametresinin KENDİ sınıfının ne yapabileceğini kısıtladığını varsaymak — sınır, argümanı tanımlar, generic sınıfın ya da metodun kendisini değil.
- Uygun bir sınır yerine bir geçici çözüm olarak `Object`'e başvurmak, bir sınırın sağlayacağı spesifik metot erişiminin tamamını kaybetmek.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Sınırsız bir tür parametresi yalnızca `Object`'in metotlarını garanti eder; sınırlı biri, hangi türlerin uygun olduğunu kısıtlamak karşılığında daha fazlasını garanti eder.
- `<T extends SomeType>`, hem sınıflar hem interface'ler için `extends` kullanarak bir üst sınır bildirir.
- Birden fazla sınır `&` ile birleştirilir; en fazla biri bir sınıf olabilir, ve ilk sırada gelmelidir.
- Bir sınır, yalnızca tek bir metotta değil, bir sınıfın tür parametresinde de görünebilir, o sınıfın her kullanımını kısıtlar.
- Bir interface ile sınırlamak (`Comparable<T>` gibi) yaygındır ve belirli bir sınıf hiyerarşisinden bağımsız, genel amaçlıdır.

Cheat Sheet

```
// Sınıfla üst sınır
static <T extends Number> double sum(List<T> numbers) {
    double total = 0;
    for (T n : numbers) total += n.doubleValue();
    return total;
}

// Birden fazla sınır: önce sınıf, sonra interface'ler, & ile birleştirilir
static <T extends Number & Comparable<T>> T max(List<T> values) { ... }

// Sınıfın kendi tür parametresinde sınır
class NumericBox<T extends Number> { ... }

// Yalnızca interface ile sınır
static <T extends Comparable<T>> T max(List<T> items) { ... }
```

Terimler Sözlüğü

- **Sınırlı tür parametresi (bounded type parameter)**: herhangi bir türü değil, belirli bir türü (ve onun alt türlerini) kabul edecek şekilde kısıtlanmış bir tür parametresi.
- **Üst sınır (upper bound)**: `extends` ile bildirilen, sınır türünün kendisine ya da alt türlerinden herhangi birine izin veren kısıtlama.
- **Birden fazla sınır (multiple bounds)**: bir türün karşılaması gereken, `&` ile birleştirilmiş iki ya da daha fazla gereksinim.
- **Sınır (bound)**: bir tür parametresinin genişletmesi (extend) ya da uygulaması (implement) gereken tür (sınıf ya da interface).