Koleksiyonlarla Generics

List<T>, Set<T> ve Map<K,V> gibi generic koleksiyonlar, koleksiyonlarla tür güvenliği, List<Object>'in neden List<String> OLMADIĞI (generics değişmezliği, tam işleniş), ve diamond operatörü/var ile tür çıkarımı. Generics serisinin 5.'si.

Orta 25 dk

Son güncelleme: 7 Kasım 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/generics-with-collections

---

Bu serideki her mekanik şimdiye kadar — generic sınıflar, generic metotlar, sınırlar, wildcard'lar — çoğunlukla `Box<T>` gibi küçük, uydurma türler kullanarak odağı mekanizmanın kendisinde tuttu. Ama pratikte, generics'i gerçekten kullanacağın en yaygın yer Java'nın kendi koleksiyon çerçevesidir. Bu ders özellikle `List<T>`, `Set<T>` ve `Map<K, V>` 'ye bakıyor, ve "Wildcard'lar"ın yalnızca geçerken motive ettiği bir soruyu kapatıyor: `List<String>` ve `List<Object>` neden tam olarak ilgisiz türlerdir.

Generic Koleksiyonlar: List, Set ve Map

`List<T>`, `Set<T>` ve `Map<K, V>`, "Generics'e Giriş"te işlenen tam olarak aynı mekanizmayla inşa edilmiş, kendileri de sıradan generic türlerdir — `List` 'in elemanları için bir tür parametresi vardır, `Map` 'in ise ikisi vardır, biri key'leri biri value'ları için.

```
import java.util.List;
import java.util.Map;
import java.util.Set;

public class GenericCollectionApisExample {

    public static void main(String[] args) {
        // List<T>: an ordered, index-accessible sequence of one element type.
        List<String> names = List.of("Alice", "Bob", "Charlie");

        // Set<T>: an unordered collection of one element type, with no duplicates.
        Set<String> uniqueNames = Set.copyOf(List.of("Alice", "Bob", "Alice")); // the
duplicate collapses

        // Map<K, V>: an association from one key type to one value type.
        Map<String, Integer> ages = Map.of("Alice", 30, "Bob", 25);

        System.out.println("List: " + names);
        System.out.println("Set size: " + uniqueNames.size());
        System.out.println("Map: " + ages);

        // The exact same three interfaces -- List, Set, Map -- are generic
        // over ANY type; nothing here is specific to String or Integer.
        List<Integer> scores = List.of(90, 85, 78);
        Set<Boolean> flags = Set.of(true, false);
        System.out.println("Scores: " + scores);
        System.out.println("Flags: " + flags);
    }
}
```

Aynı üç interface, ne tuttuklarından bağımsız olarak birebir aynı şekilde çalışır — `List<String>` ve `List<Integer>`, aynı `List`'tir, `Set<String>` ve `Set<Boolean>`, aynı `Set`'tir. Koleksiyon API'lerinin kendisiyle ilgili hiçbir şey değişmez; yalnızca tür argümanı değişir.

Koleksiyonlarla Tür Güvenliği

"Generics'e Giriş"te genel olarak işlenen derleme-zamanı kontrolü, her koleksiyon işlemine — `add`, `put`, `get` — uygulanır, yalnızca oluşturmaya değil.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class CollectionTypeSafetyExample {

    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add("Alice");
        // names.add(42); // would NOT compile -- List<String> only accepts String

        String first = names.get(0); // no cast needed -- the compiler already knows it's a
String

        Map<String, Integer> ages = new HashMap<>();
        ages.put("Alice", 30);
        // ages.put("Alice", "thirty"); // would NOT compile -- the value must be an Integer
        // ages.put(42, 30);            // would NOT compile -- the key must be a String

        int age = ages.get("Alice"); // no cast needed here either

        System.out.println(first + " is " + age);
    }
}
```

Beklenen Çıktı

```
Alice is 30
```

Bir `List<String>` üzerinde `names.add(42)` ve bir `Map<String, Integer>` üzerinde `ages.put("Alice", "thirty")`, ikisi de derleme zamanında reddedilir, iki hata da hiç çalışan bir programa ulaşamadan önce. `names.get(0)` ya da `ages.get("Alice")` ile geri okumak da aynı nedenle hiçbir cast gerektirmez — derleyici tam olarak hangi türün saklandığını zaten bilir.

List
Neden List Değildir
"Wildcard'lar" bu kuralı motivasyon olarak kısaca tanıttı; işte daha eksiksiz resim. Generics DEĞİŞMEZDİR (invariant): `String` bir `Object` OLSA bile, `List<String>` ve `List<Object>`, hiçbir yönde birbirinin yerine geçemeyen, tamamen ilgisiz iki tür olarak ele alınır.

```
import java.util.ArrayList;
import java.util.List;

public class ListInvarianceExample {

    // If List<Object> WERE a List<String> (it isn't), this method would
    // let you sneak an Integer into a list that was really created as a
    // List<String> -- a broken promise the compiler will never allow.
    static void addNumber(List<Object> list) {
        list.add(42);
    }

    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add("Alice");

        // addNumber(names); // would NOT compile -- List<String> is not a
        //     List<Object>, even though String IS an Object. If this line
        //     were allowed, addNumber(...) could insert an Integer into
        //     what the caller believes is purely a list of Strings --
        //     exactly the unsafety generics exist to prevent.

        List<Object> anything = new ArrayList<>();
        addNumber(anything); // fine -- this really is a List<Object>

        System.out.println(names);
        System.out.println(anything);
    }
}
```

Beklenen Çıktı

```
[Alice]
[42]
```

`List<String>`'in bir `List<Object>` beklenen yerde geçirilmesine İZİN VERİLSEYDİ, `addNumber(...)`, çağıranın yalnızca `String`'lerden oluştuğuna inandığı bir listeye bir `Integer` ekleyebilirdi — tür sisteminin daha sonra yakalamanın hiçbir yolu olmayan, bozulmuş bir söz. Değişmezlik, tam olarak bunu önleyen şeydir: `List<String>`, yalnızca bir `List<String>`'in (ya da "Wildcard'lar"ın işlediği gibi, her `List`'in zaten sağladığı bir `List<? extends Object>`'in) beklendiği yerde geçirilebilir.

> **Bir metodun bilinmeyen ya da ilişkili bir eleman türünden bir `List` i gerçekten kabul etmesi gerektiğinde, "Wildcard'lar"daki wildcard formlarına (`List<?>`, `List<? extends T>`, `List<? super T>`) başvur — bu değişmezlik kuralının gerekli kıldığı tam olarak bu araçtır.**

Koleksiyonlarla Tür Çıkarımı

Bir koleksiyon oluşturmak, tür argümanının iki kez tekrarlanmasını gerektirmez — Java bunu bağlamdan iki yaygın şekilde çıkarır.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class DiamondOperatorInferenceExample {

    public static void main(String[] args) {
        // The diamond operator, <>, lets the constructor's type argument be
        // INFERRED from the variable's declared type -- no need to repeat
        // "String" on both sides.
        List<String> names = new ArrayList<>(); // infers ArrayList<String>
        names.add("Alice");

        Map<String, Integer> ages = new HashMap<>(); // infers HashMap<String, Integer>
        ages.put("Alice", 30);

        // "var" infers the variable's type from the right-hand side instead
        // -- here the compiler works out that scores is a List<Integer>,
        // purely from what List.of(...) returns.
        var scores = List.of(90, 85, 78);

        System.out.println(names);
        System.out.println(ages);
        System.out.println(scores);
    }
}
```

Diamond operatörü, `<>`, bir constructor'ın tür argümanını atandığı değişkenden çıkarır — bir `List<String>` değişkenine atanan `new ArrayList<>()`, `String`'i tekrar yazmadan bir `ArrayList<String>` olur. `var` ise bunun yerine değişkenin kendi türünü, sağ

taraftaki her neyse ondan çıkarır — `var scores = List.of(90, 85, 78)`, `scores`'a, tamamen `List.of(...)`'un argümanlarından çıkarılan `List<Integer>` türünü verir.

Pratik Bir Örnek

Generic koleksiyonlar, günlük Java kodunun bel kemiğidir — sayma, gruplama ve arama neredeyse her zaman bir `Map` ya da bir `List` üzerinden geçer.

```
import java.util.HashMap;
import java.util.Map;

public class PracticalWordFrequencyExample {

    // A practical, everyday use of Map<K, V>: counting how many times
    // each word appears -- the kind of code that shows up constantly in
    // real applications.
    static Map<String, Integer> countWords(String[] words) {
        Map<String, Integer> frequency = new HashMap<>();
        for (String word : words) {
            frequency.merge(word, 1, Integer::sum);
        }
        return frequency;
    }

    public static void main(String[] args) {
        String[] words = {"apple", "banana", "apple", "cherry", "banana", "apple"};

        Map<String, Integer> counts = countWords(words);
        for (Map.Entry<String, Integer> entry : counts.entrySet()) {
            System.out.println(entry.getKey() + ": " + entry.getValue());
        }
    }
}
```

`countWords(...)`, bir kelime array'inden `merge(...)` kullanarak her kelimenin sayısını artıran bir `Map<String, Integer>` inşa ediyor — bu derste işlenen tür güvenliğine ve çıkarımına tamamen dayanan, hiçbir yerde cast olmayan sıradan, pratik kod.

Best Practices

- **Değişken ve parametre türü olarak somut implementasyonlar (`ArrayList`, `HashMap`) yerine koleksiyon interface'lerini (`List`, `Set`, `Map`) tercih et — bu, "Interface"in genel kılavuzunu yansıtır ve generic koleksiyon türleri için de aynı derecede geçerlidir.**
- **Açıkça türlenmiş bir değişkenle bir koleksiyon oluştururken varsayılan olarak diamond operatörünü kullan — tür argümanını her iki tarafta da tekrarlamak için nadiren bir neden vardır.**
- **Bir koleksiyonun türü zaten başlatıcısından belli olduğunda `var`'a başvur, ama belirgin olmayan bir durumda okunabilirliği artırdığında açık türü koru.**
- **Bir metodun ilişkili ama aynı olmayan bir eleman türünden bir `List`i kabul etmesi gerektiğinde, değişmezliği başka bir şekilde aşmaya çalışmak yerine ("Wildcard'lar"dan) bir wildcard kullan.**

Yaygın Hatalar

- **Bir `List<String>`'i bir `List<Object>` beklenen yerde geçirmeye çalışıp derleyicinin bunu reddetmesine şaşırmak — bu, bir derleyici kısıtı değil, değişmezliğin tam olarak tasarlandığı gibi çalışması.**
- **Bir `Map<K, V>`'nin tür güvenliğinin key'leri ve value'ları birbirinden bağımsız olarak kapsadığını unutmak — `put(...)` ve `get(...)`, ikisi de kendi tür parametrelerine göre kontrol edilir.**
- **Bir bildirimin sağ tarafına diamond operatörü yerine tam generic türü yazmak (`List<String> names = new ArrayList<String>();`), saf bir tekrar eklemek.**
- **`var`'ın bir değişkenin türünü "daha az sıkı" yaptığını ya da tür güvenliğini kaldırdığını varsaymak — yalnızca türü**

YAZMA ihtiyacını kaldırır; derleyici, sanki açıkça yazılmış gibi onu aynen zorlamaya devam eder.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- `List<T>`, `Set<T>` ve `Map<K, V>`, herhangi bir özel generic sınıfla aynı mekanizmayla inşa edilmiş, sıradan generic türlerdir.
- Koleksiyon işlemleri (`add`, `put`, `get`), hepsi koleksiyonun bildirilen tür argümanlarına göre derleme zamanında kontrol edilir.
- `List<String>` ve `List<Object>`, generics değişmez olduğu için ilgisiz türlerdir -- bu, "Wildcard'lar"daki wildcard formlarını baştan gerekli kılan şeydir.
- Diamond operatörü (`<>`), bir constructor'ın tür argümanını bağlamdan çıkarır; `var`, bir değişkenin tüm türünü başlatıcısından çıkarır.
- Generic koleksiyonlar, generics'in günlük Java kodunda gerçekten en yaygın kullanıldığı yerdir.

Cheat Sheet

```
// Üç temel generic koleksiyon türü
List<String> names = List.of("Alice", "Bob");
Set<String> unique = Set.of("Alice", "Bob");
Map<String, Integer> ages = Map.of("Alice", 30);

// Derleme zamanında kontrol edilen tür güvenliği
List<String> list = new ArrayList<>();
list.add("ok");
// list.add(42); // reddedilir

// Diamond operatörü vs var
List<String> a = new ArrayList<>();       // diamond, ArrayList<String>'i çıkarır
var b = List.of(1, 2, 3);                  // var, List<Integer>'ı çıkarır

// Değişmezlik
// List<Object> o = names; // reddedilir -- List<String>, List<Object> değildir
```

Terimler Sözlüğü

- **Generic koleksiyon: tuttuğu tür(ler)le parametrelenmiş bir koleksiyon türü (`List`, `Set`, `Map`).**
- **Değişmezlik (invariance): `A` ve `B` aynı tür olmadıkça, ilişkili olsalar bile `List<A>` ve `List<B>`'nin ilgisiz türler olduğu kuralı.**
- **Diamond operatörü: `<>`, bir constructor'ın tür argümanının atandığı değişkenden çıkarılmasını sağlar.**
- **var: bir yerel değişkenin tüm bildirilen türünü, hiçbir derleme-zamanı tür kontrolünü kaldırmadan, başlatıcısından çıkaran bir anahtar kelime.**