Generics'e Giriş

Generics'in neden var olduğu (pre-generics raw type/cast/ClassCastException sorunu), tür güvenliğinin derleme zamanında nasıl sağlandığı, generic sınıflar ve interface'ler, ve tür parametresi adlandırma kuralı. Generics serisinin 1.'si.

Orta 25 dk

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

---

Bu kurs boyunca kullandığın Java sınıfları ve interface'leri — `List`, `Optional`, "Interface" ve "Abstract Class" derslerindeki kendi sınıfların — bir kez yazılıp her yerde yeniden kullanılıyor. Generics tam olarak bu fikri bir adım daha ileri götürüyor: bir sınıfı her TÜR için ayrı ayrı yazmak yerine, herhangi bir tür için TEK SEFERDE yazıyorsun, ve derleyici onun her kullanımını sanki elle özel bir versiyon yazmışsın gibi sıkı biçimde kontrol etmeye devam ediyor. Bu ders, tam olarak bu mekanizma üzerine yeni bir seri açıyor.

Generics Nedir?

Generics, bir sınıfın, interface'in ya da metodun, tıpkı bir metodun sıradan değerlerle parametrelendirilmesi gibi, bir TÜR ile parametrelendirilmesine izin verir. `List<String>` ve `List<Integer>`, ikisi de tam olarak aynı `List` sınıfından inşa edilir — açılı parantezlerin içindeki kısım, TÜR ARGÜMANI, derleyiciye bu belirli kullanımın hangi türü tutması gerektiğini söyler, ayrı bir `StringList` ve `IntegerList` sınıfına gerek kalmadan.

Neden Var?

Generics'ten önce (Java 5'te, 2004'te tanıtıldı), `List` gibi genel amaçlı bir konteyner, hangi türde eleman tuttuğunu hatırlamanın bir yolunu bulamıyordu — yalnızca `Object`'i, her şeyin ortak atasını saklayabiliyordu. Bir elemanı geri okumak açık bir cast gerektiriyordu, ve baştan yanlış türde bir şey eklemeni engelleyen hiçbir şey yoktu; hata daha sonra, genelde yanlış elemanın gerçekten eklendiği yerden çok uzakta, bir `ClassCastException` olarak ortaya çıkıyordu.

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

public class PreGenericsCastingProblemExample {

    public static void main(String[] args) {
        // A raw (non-generic) List: the compiler has no idea what type of
        // element it's supposed to hold, so it accepts anything...
        List names = new ArrayList();
        names.add("Alice");
        names.add("Bob");
        names.add(42); // ...including something that clearly doesn't belong here.

        // Reading requires an explicit, unchecked cast -- the compiler
        // trusts you that the element really is a String.
        for (Object item : names) {
            String name = (String) item; // throws at runtime on the int 42
            System.out.println(name.toUpperCase());
        }
    }
}
```

Bu raw (generic olmayan) `List`, hiçbir itiraz etmeden önce bir `String` sonra bir `Integer` kabul ediyor — hata ancak daha sonra, döngü yanlış konumlandırılmış elemana ulaştığında, cast noktasında ortaya çıkıyor. Generics, tam olarak bu sınıf hatayı program hiç çalışmadan önce, DERLEME zamanında yakalamak için var.

Tür Parametreleri

Açılı parantezlerin içindeki harf — `Box<T>`'deki `T`, `Pair<K, V>`'deki `K` / `V` — TÜR PARAMETRESİ olarak adlandırılır: sınıf gerçekten kullanıldığında doldurulacak bir tür için yer tutucu bir isim. Kural olarak, Java kodu kısa, tek büyük harfli isimler kullanır: genel bir generic tür için `T`, bir koleksiyon elemanı için `E`, bir map'in key ve value'su için `K` ve `V`, bir sayı için `N`. Dilde bu harfleri özellikle zorlayan hiçbir şey yok, ama okuyacağın her Java kod tabanı bunları bekler.

Generic Sınıflar

Bir sınıf, isminin hemen ardından bir ya da daha fazla tür parametresi bildirerek ve bu parametreyi normalde somut bir türün göründüğü her yerde — alan türleri, metot parametreleri, dönüş türleri — kullanarak generic olur.

```
public class GenericBoxClassExample {

    // A generic class: T is a type parameter, a placeholder for whatever
    // real type is supplied when Box is used. The class is written ONCE,
    // but works correctly for any type without any casting or duplication.
    static class Box<T> {
        private T content;

        void set(T content) {
            this.content = content;
        }

        T get() {
            return content;
        }
    }

    public static void main(String[] args) {
        Box<String> stringBox = new Box<>();
        stringBox.set("hello");
        String value = stringBox.get(); // no cast needed -- the compiler already knows
it's a String

        Box<Integer> intBox = new Box<>();
        intBox.set(42);
        int number = intBox.get(); // same class, different type argument, still no cast

        System.out.println(value.toUpperCase());
        System.out.println(number * 2);
    }
}
```

Beklenen Çıktı

```
HELLO
84
```

`Box<T>` tam olarak bir kez yazılıyor, ama `Box<String>` ve `Box<Integer>` tamamen ayrı, tam tür-güvenli iki sınıf gibi davranıyor — `stringBox.get()` hiçbir cast gerekmeden bir `String` döndürüyor, ve derleyici bir `Integer`'ı bir `Box<String>`'e `set(...)` etmeye çalışmayı reddeder.

Bir sınıf tek bir tür parametresiyle sınırlı değil — tasarımın ihtiyaç duyduğu kadar çoğu, virgülle ayrılarak bildirilebilir.

```
public class GenericPairClassExample {

    // A generic class with TWO type parameters -- K and V here are just
    // names (by convention, K for "key", V for "value"); nothing forces
    // this specific naming beyond readability.
    static class Pair<K, V> {
        private final K key;
        private final V value;

        Pair(K key, V value) {
            this.key = key;
            this.value = value;
        }

        K getKey() {
            return key;
        }

        V getValue() {
            return value;
        }

        @Override
        public String toString() {
            return key + " = " + value;
        }
    }

    public static void main(String[] args) {
        Pair<String, Integer> ageEntry = new Pair<>("Alice", 30);
        Pair<Integer, String> idEntry = new Pair<>(101, "order-created");

        System.out.println(ageEntry);
        System.out.println(idEntry);
    }
}
```

Beklenen Çıktı

```
Alice = 30
101 = order-created
```

`Pair<K, V>`, iki bağımsız tür parametresi kullanıyor — `Pair<String, Integer>` ve `Pair<Integer, String>`, ikisi de tam olarak aynı sınıfın geçerli, birbiriyle ilgisiz kullanımları, her biri kendi iki türü için derleyici tarafından tam olarak kontrol ediliyor.

Generic Interface'ler

Interface'ler, tıpkı sınıflar gibi tür parametreleri bildirebilir — tür parametresi daha sonra interface'in bildirdiği her metoda akar.

```
public class GenericInterfaceExample {

    // A generic interface: the type parameter T flows through the method
    // signatures exactly like it would in a generic class.
    interface Repository<T> {
        void save(T item);
        T findLatest();
    }

    // A concrete class implementing the interface supplies a real type
    // argument -- every method in the implementation now deals with
    // Order specifically, no casting required anywhere.
    static class InMemoryOrderRepository implements Repository<Order> {
        private Order latest;

        @Override
        public void save(Order item) {
            latest = item;
        }

        @Override
        public Order findLatest() {
            return latest;
        }
    }

    record Order(int id, double total) {
    }

    public static void main(String[] args) {
        Repository<Order> repository = new InMemoryOrderRepository();
        repository.save(new Order(1, 49.99));

        Order latest = repository.findLatest(); // already an Order, no cast
        System.out.println(latest);
    }
}
```

`Repository<T>`, `save(T item)` ve `T findLatest()`'i `T` cinsinden bildiriyor. `InMemoryOrderRepository implements Repository<Order>`, gerçek tür argümanını sağlıyor — o implementasyondaki her metot artık özel olarak `Order` ile ilgileniyor, ve `findLatest()`, `main` içinde hiçbir yerde cast gerekmeden bir `Order` döndürüyor.

Derleme Zamanında Tür Güvenliği

Yukarıdakilerin somut faydası, geçersiz kullanımın program hiç çalışmadan reddedilmesi, daha sonra bir çalışma zamanı çökmesi olarak keşfedilmemesidir.

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

public class TypeSafetyCompileTimeCheckExample {

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

        // names.add(42); // does NOT compile -- int cannot be used where
        //                   the compiler expects a String. This line is
        //                   commented out only because it wouldn't compile
        //                   otherwise; uncomment it to see the real error.

        // Reading back requires no cast at all -- the compiler already
        // knows every element is a String, because that's the only thing
        // this particular List<String> was ever allowed to accept.
        for (String name : names) {
            System.out.println(name.toUpperCase(Locale.ROOT));
        }
    }
}
```

Beklenen Çıktı

```
ALICE
BOB
```

Bir `List<String>`'e `add(42)` yapmaya çalışmak basitçe derlenmiyor — unutulacak bir cast, sonradan gerçekleşmeyi bekleyen bir `ClassCastException` yok. Bu, generics'in verdiği temel sözdür: ilk örnekte gösterilen türden hata, baştan yazılamaz hâle geliyor.

> Bir raw type kullanıldığını gördüğünde (açılı parantezleri olmadan referans verilen bir generic sınıf, `List<String>` yerine düz `List` gibi), bunu bir uyarı işareti olarak ele al — derleyici o belirli kullanım için generics-öncesi davranışa geri düşer, bu derste işlenen tür-güvenliği faydalarının tamamını sessizce kaybeder.

Best Practices

- Generic bir sınıf ya da interface kullanırken her zaman bir tür argümanı sağla — yazdığın kodda raw type'lardan tamamen kaçın.
- Tür parametreleri için standart tek-harf adlandırma kuralına (`T`, `E`, `K`, `V`, `N`) uy, böylece diğer Java geliştiricileri rollerini hemen tanır.
- Farklı türler için neredeyse birebir aynı kodu yazdığını fark ettiğinde generic bir sınıfa başvur — bu tekrar, tam olarak generics'in ortadan kaldırdığı şeydir.
- Tür parametresi sayısını az tut; çok fazlası olan bir sınıf, önlemeye çalıştığı tekrardan daha hızlı okunması zor hâle gelir.

Yaygın Hatalar

- Bir raw type kullanmak (`List<String>` yerine `List`) ve daha sonra generics'in önlemesi gereken bir `ClassCastException` 'a şaşırmak.
- `Box<Object>` 'in generics-öncesi kodun yaptığı gibi her şeyi tutabileceğini varsaymak — yalnızca bildirileni tutabilir, ve (sonraki derslerin işlediği gibi) `Box<String>` ile birbirinin yerine kullanılamaz.
- Diğer geliştiricilerin kodu bir bakışta okumasını zorlaştıran alışılmadık tür parametresi isimleri uydurmak.
- Tür PARAMETRESİYLE (`T` , sınıf bildirimindeki yer tutucu) tür ARGÜMANINI (`String` , sınıf kullanıldığında sağlanan gerçek tür) karıştırmak — bu iki terim aynı ilişkinin farklı taraflarını tanımlar.

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- Generics, bir sınıfın, interface'in ya da metodun bir türle parametrelenmesine izin verir, derleyici tarafından tam olarak kontrol edilir.
- Generics'ten önce, genel amaçlı konteynerler `Object` saklıyordu ve kontrolsüz cast gerektiriyordu, tür hatalarını çalışma zamanına erteliyordu.
- Bir tür parametresi (`T` , `K` , `V` , ...), sınıf kullanıldığında gerçek bir tür argümanıyla doldurulan bir yer tutucudur.
- Bir sınıf ya da interface, alanlarında, metot parametrelerinde ve dönüş türlerinde kullanılan bir ya da daha fazla tür parametresi bildirebilir.
- Temel fayda, tür hatalarını daha sonra bir `ClassCastException` olarak keşfetmek yerine derleme zamanında yakalamaktır.

Cheat Sheet

```
// Generic sınıf, tek tür parametresi
class Box<T> {
    private T content;
    void set(T content) { this.content = content; }
    T get() { return content; }
}
Box<String> box = new Box<>();

// Generic sınıf, iki tür parametresi
class Pair<K, V> {
    K getKey() { ... }
    V getValue() { ... }
}
Pair<String, Integer> entry = new Pair<>("Alice", 30);

// Generic interface
interface Repository<T> {
    void save(T item);
    T findLatest();
}
class OrderRepository implements Repository<Order> { ... }
```

Terimler Sözlüğü

- **Generics**: bir sınıfın, interface'in ya da metodun bir türle parametrelenmesine izin veren mekanizma.
- **Tür parametresi (type parameter)**: generic bir sınıf, interface ya da metotta bildirilen bir yer tutucu isim (`T`, `E`, `K`, `V`, `N`).
- **Tür argümanı (type argument)**: generic bir sınıf gerçekten kullanıldığında bir tür parametresi için sağlanan gerçek, somut tür.
- **Raw type**: hiçbir tür argümanı olmadan kullanılan, generics-öncesi davranışa geri düşen bir generic sınıf ya da interface.
- **Tür güvenliği (type safety)**: derleyicinin, program yanlış bir değerle çalışmaya başlayamadan önce, uyumsuz türleri derleme zamanında reddetmesi.