Throw ve Throws

throw ifadesinin çalışma zamanında bir Throwable nesnesini JVM'e teslim etmesi ile throws bildiriminin yalnızca derleme-zamanı bir sözleşme olması arasındaki fark -- fail-fast doğrulama, yakalanan bir exception'ı cause parametresiyle farklı bir türle yeniden fırlatmak, ve checked exception'ın çağrı zinciri boyunca yakalanmadan yayılması. Exception Handling serisinin 5.'si.

Başlangıç 20 dk

Son güncelleme: 23 Ağustos 2026
Çevrimiçi oku: https://www.learnforgex.com/tr/topics/throw-and-throws

---

Bu seri boyunca `throw` ve `throws`'u zaten geçerken kullandın — bir örnekte bir `throw`, bir metot imzasında `throws IOException`. Bu ders ikisine de tek tek durup bakıyor: `throw` çalıştığında gerçekte ne yapar, `throws` ne bildirir ve ne YAPMAZ, ve ikisi bir exception'ı ortaya çıktığı yerden yakalandığı yere taşımak için nasıl birlikte çalışır.

Throw ve Throws Nedir?

`throw` bir İFADEDİR (statement) — kodunda belirli bir noktada çalışır ve çalışma zamanında, tam o anda, bir `Throwable` nesnesini doğrudan JVM'e teslim eder. `throws` ise bir BİLDİRİMDİR — bir metodun imzasında yer alır ve derleyiciye (ve metodu okuyan herkese) bu metottan hangi checked exception'ların çıkabileceğini söyler; `throws` yazmak, tek başına, hiçbir şey fırlatmaz ya da hiçbir kod çalıştırmaz.

Neden Var?

`throw` olmasaydı, kodun "bir şey ters gitti" demesinin tek yolu sihirli bir dönüş değeri olurdu (`-1` ya da `null` gibi) — tam olarak "Exception'lara Giriş"in açtığı sorun. `throws` olmasaydı, çağrı zincirinin derinliklerinde fırlatılan checked bir exception'ın, onu yakalaması gereken kişiye derleyici tarafından doğrulanmış bir yolu olmazdı — aradaki her metot onu sessizce unutabilirdi. Birlikte, `throw` başarısızlığı YARATIR, `throws` ise (checked exception'lar için) derleyicinin bu başarısızlığın bir handler'a giden yolunu takip etmesini sağlar.

throw İfadesi

`throw`, tek bir `Throwable` nesnesi alır — genelde `new` ile o anda oluşturduğun bir nesne — ve kontrolü o noktadan hemen uzaklaştırır. Koşulsuz bir `throw`'dan hemen sonra yazılan herhangi bir kod ERİŞİLEMEZDİR (unreachable) ve derleyici bunu doğrudan reddeder.

```
public class ThrowStatementBasicsExample {

    public static void main(String[] args) {
        try {
            reject();
        } catch (IllegalStateException e) {
            System.out.println("Caught: " + e.getMessage());
        }
    }

    static void reject() {
        // "throw" is a statement: it executes right now, immediately handing
        // a Throwable instance to the JVM. Execution of this method stops here.
        throw new IllegalStateException("this operation is not allowed right now");

        // Any code placed here would be UNREACHABLE and would not even compile
        // -- the compiler knows the throw above always transfers control away.
    }
}
```

`reject()` içinde, çalışma `throw` satırını hiçbir zaman geçmez — "return" edecek hiçbir şey yoktur, ve o blokta ondan sonra kod koyacak bir yer de yoktur.

Erken Başarısız Olmak İçin Fırlatmak (Fail Fast)

`throw`'un çok yaygın bir kullanımı, gerçek iş başlamadan ÖNCE, metodun en başında argümanları doğrulamak ve geçersizlerse hemen fırlatmaktır. Buna "erken başarısız olma" (fail fast) denir — hata, gerçek kaynağında raporlanır, başka bir yerde kafa karıştırıcı bir belirti olarak sonradan ortaya çıkmak yerine.

```
public class FailFastValidationExample {

    public static void main(String[] args) {
        System.out.println(applyDiscount(100.0, 10));

        try {
            applyDiscount(100.0, -5);
        } catch (IllegalArgumentException e) {
            System.out.println("Rejected: " + e.getMessage());
        }
    }

    static double applyDiscount(double price, int percent) {
        // Fail fast: validate arguments first and throw immediately, before
        // any real work happens -- this keeps the rest of the method free of
        // defensive checks and makes the bad input's origin obvious.
        if (percent < 0 || percent > 100) {
            throw new IllegalArgumentException("percent must be between 0 and 100, was " +
percent);
        }

        return price - (price * percent / 100);
    }
}
```

`applyDiscount(...)`, herhangi bir hesaplama yapmadan ÖNCE `percent`'i kontrol eder — bu kontrol kaldırılsaydı, geçersiz bir yüzde hemen çökmezdi; sessizce yanlış bir fiyat üretirdi, bu da çok daha zor izlenebilecek bir hata olurdu.

> Geçersiz girdi için bir `throw` yazarken, bunu metodun yaptığı İLK şey olarak koy. Metodun ortasına gömülmüş bir doğrulama kontrolü kaçırılması kolaydır ve metot sonradan refactor edildiğinde yanlışlıkla atlanması kolaydır.

Yeniden Fırlatmak (Rethrowing): Bir Exception'ı Yakalayıp Başka Birini Fırlatmak

Bazen bir exception'ı yakaladıktan sonra doğru tepki onu ele almak değil, yerine daha anlamlı, farklı bir exception `throw` etmektir — kendi metodunun VAAT ETTİĞİ şey cinsinden tanımlanan, bağımlı olduğu iç bir detay cinsinden değil.

```
public class RethrowingCaughtExceptionExample {

    public static void main(String[] args) {
        try {
            loadConfiguration("settings.cfg");
        } catch (IllegalStateException e) {
            System.out.println("Startup failed: " + e.getMessage());
        }
    }

    static void loadConfiguration(String path) {
        try {
            parse(path);
        } catch (NumberFormatException e) {
            // Rethrowing: catch a specific, low-level exception and throw a
            // different, higher-level one in its place -- the caller sees a
            // failure described in terms this method's contract, not in
            // terms of an internal detail (a malformed number) it never
            // promised to expose. The original is kept as the "cause" so
            // no diagnostic information is lost.
            throw new IllegalStateException("configuration file is corrupt: " + path, e);
        }
    }

    static void parse(String path) {
        Integer.parseInt("not-a-number");
    }
}
```

`loadConfiguration(...)`, `parse(...)`'tan gelen düşük seviyeli `NumberFormatException`'ı yakalar ve kendi çağıranı için gerçekten bir anlam ifade eden bir `IllegalStateException` fırlatır — "yapılandırma dosyası bozuk" bu metodun dışında anlamlıdır; "bir sayı ayrıştırılamadı" değildir. İkinci constructor argümanına dikkat et: orijinal exception'ı `cause` olarak geçirmek, onun mesajını ve stack trace'ini bağlı tutar, bu yüzden farklı bir türle yeniden fırlatmak hiçbir şey kaybettirmez. Bu tam deseni — `IllegalStateException` gibi hazır türleri yeniden kullanmak yerine kendi exception türlerini tasarlamayı — bir sonraki derste resmileştireceksin.

throws Bildirimi ve Yayılma (Propagation)

Bir metot imzasındaki `throws`, tek başına hiçbir şey çalıştırmaz — yalnızca derleyiciye, bu metottan checked bir exception'ın, metot onu yakalamadan ÇIKABİLECEĞİNİ söyler. "Checked ve Unchecked Exception'lar"da gördüğün gibi, checked-exception sözleşmesini uygulanabilir kılan tam olarak budur: çağrı zincirindeki her metot ya exception'ı `catch` etmeli ya da kendi `throws`'unu eklemelidir, en yukarıya kadar.

```
import java.io.FileNotFoundException;

public class ThrowsDeclarationPropagationExample {

    public static void main(String[] args) {
        try {
            step1();
        } catch (FileNotFoundException e) {
            System.out.println("Caught at the top: " + e.getMessage());
        }
    }

    // Each method here declares "throws" instead of catching -- the checked
    // exception simply passes through, unhandled, all the way up the call
    // chain until something finally catches it in main. "throws" costs
    // nothing to write and does not stop execution by itself; it is a
    // compile-time declaration, not a runtime action like "throw".
    static void step1() throws FileNotFoundException {
        step2();
    }

    static void step2() throws FileNotFoundException {
        step3();
    }

    static void step3() throws FileNotFoundException {
        throw new FileNotFoundException("report.csv");
    }
}
```

Burada, gerçek bir `throw` içeren tek metot `step3()`'tür — `step1()` ve `step2()`, `FileNotFoundException`'a hiç dokunmaz, yalnızca `throws FileNotFoundException` bildirir ve onun doğrudan geçmesine izin verir. O bildirim yüzünden `step1()` ya da `step2()` içinde farklı hiçbir şey çalışmaz; bu tamamen, exception'ın nerede sonlanabileceği konusunda derleyiciyi dürüst tutan, derleme-zamanı bir muhasebedir.

> `throws SomeException` bildirmek, exception'ı hiçbir şekilde yakalamaz ya da azaltmaz — yalnızca derleyicinin yükümlülüğünü bu metodu çağıran kişiye kaydırır. Çağrı zincirinde yukarıda hiçbir yer onu `catch` etmezse, exception yine de yakalanmadığında programı sonlandırır — tıpkı "Exception'lara Giriş"te gördüğün gibi.

throw vs throws: Temel Ayrım

İsimlerin benzerliği kafa karıştırdığı için farkı doğrudan belirtmekte fayda var: `throw`, bir metodun GÖVDESİNİN yaptığı bir şeydir, belirli bir satırda, çalışma zamanında, ve ona ulaşan bir çalışma yolu başına yalnızca bir kez görünebilir. `throws` ise bir metodun İMZASININ bildirdiği bir şeydir, derleme zamanında, ve tek bir metot, yukarıdaki `step1()` ve `step2()`'nin gösterdiği gibi kendisi hiç `throw` çağırmadan bile ihtiyaç duyduğu kadar çok exception türünü (virgülle ayırarak) bildirebilir.

Best Practices

- Argümanları doğrula ve fırlat işlemini metodun en başında yap — erken başarısız ol, sorunun gerçek kaynağında.
- Farklı bir exception türüyle yeniden fırlatırken, hiçbir tanı bilgisinin kaybolmaması için orijinalini her zaman `cause` olarak geçir.
- `throws`'u yalnızca metodun (ya da çağırdığı bir şeyin) gerçekten üretebileceği checked exception'lar için ekle — "ne olur ne olmaz" diye savunmacı bir şekilde bildirme.
- Fırlattığın exception türünü kendi uygulama detaylarına doğru değil, ÇAĞIRANA anlamlı olacak şekilde seç.

Yaygın Hatalar

- Koşulsuz bir `throw`'dan sonra kod yazıp derleyicinin bunu erişilemez diye reddetmesine şaşırmak.
- Bir metottaki `throws` bildiriminin exception'ı bir şekilde "ele aldığını" sanmak — hiçbir şeyi yakalamaz ya da bastırmaz.
- Yakalanan bir exception'ı yeni bir türle yeniden fırlatırken orijinalini `cause` olarak geçirmemek, stack trace'ini sessizce kaybetmek.
- Ne olduğunu gerçekten anlatan bir tür ve mesaj yerine genel, düşük bilgili bir exception fırlatmak (`throw new RuntimeException("error")` gibi).

Özet, Cheat Sheet ve Terimler Sözlüğü

Özet

- `throw`, bir `Throwable` nesnesini hemen JVM'e teslim eden ve kontrolü uzaklaştıran, çalışma zamanı ifadesidir.
- `throws`, bir metot imzasında yer alan, checked bir exception'ın orada yakalanmadan yayılmasına izin veren derleme-zamanı bildirimidir.
- Erken başarısız olmak, girdiyi doğrulamak ve sorunun gerçek kaynağında, metodun en başında fırlatmak demektir.
- Farklı, daha anlamlı bir exception türüyle yeniden fırlatmak (orijinali `cause` olarak geçirerek) yaygın ve güvenli bir desendir.
- `throw`, belirli bir satırda yol başına bir kez çalışır; `throws` birden fazla exception türünü listeleyebilir ve aynı metotta bir `throw` gerektirmez.

Cheat Sheet

```
// throw: çalışma zamanı ifadesi
void reject() {
    throw new IllegalStateException("not allowed");
}

// Fail fast
void applyDiscount(double price, int percent) {
    if (percent < 0 || percent > 100) {
        throw new IllegalArgumentException("percent must be between 0 and 100, was " +
percent);
    }
}

// cause ile yeniden fırlatma
try {
    parse(input);
} catch (NumberFormatException e) {
    throw new IllegalStateException("configuration file is corrupt", e);
}

// throws: derleme-zamanı bildirimi, catch olmadan yayılma
void step1() throws FileNotFoundException {
    step2(); // burada catch yok -- yalnızca geçiyor
}
```

Terimler Sözlüğü

- **throw**: bir `Throwable` nesnesini hemen JVM'e teslim edip kontrolü uzaklaştıran ifade.
- **throws**: bir metodun yayılmasına izin verebileceği checked exception'ları listeleyen metot imzası bildirimi.
- **Fail fast**: geçersiz girdi tespit edildiği anda, sorunun gerçek kaynağında hemen fırlatmak.
- **Yeniden fırlatma (rethrowing)**: bir exception'ı yakalayıp yerine farklı bir tanesini fırlatmak, genelde orijinalini `cause` olarak geçirerek.
- **Yayılma (propagation)**: checked bir exception'ın, her biri onu `throws` ile bildiren bir metot zincirinden ele alınmadan geçmesi.