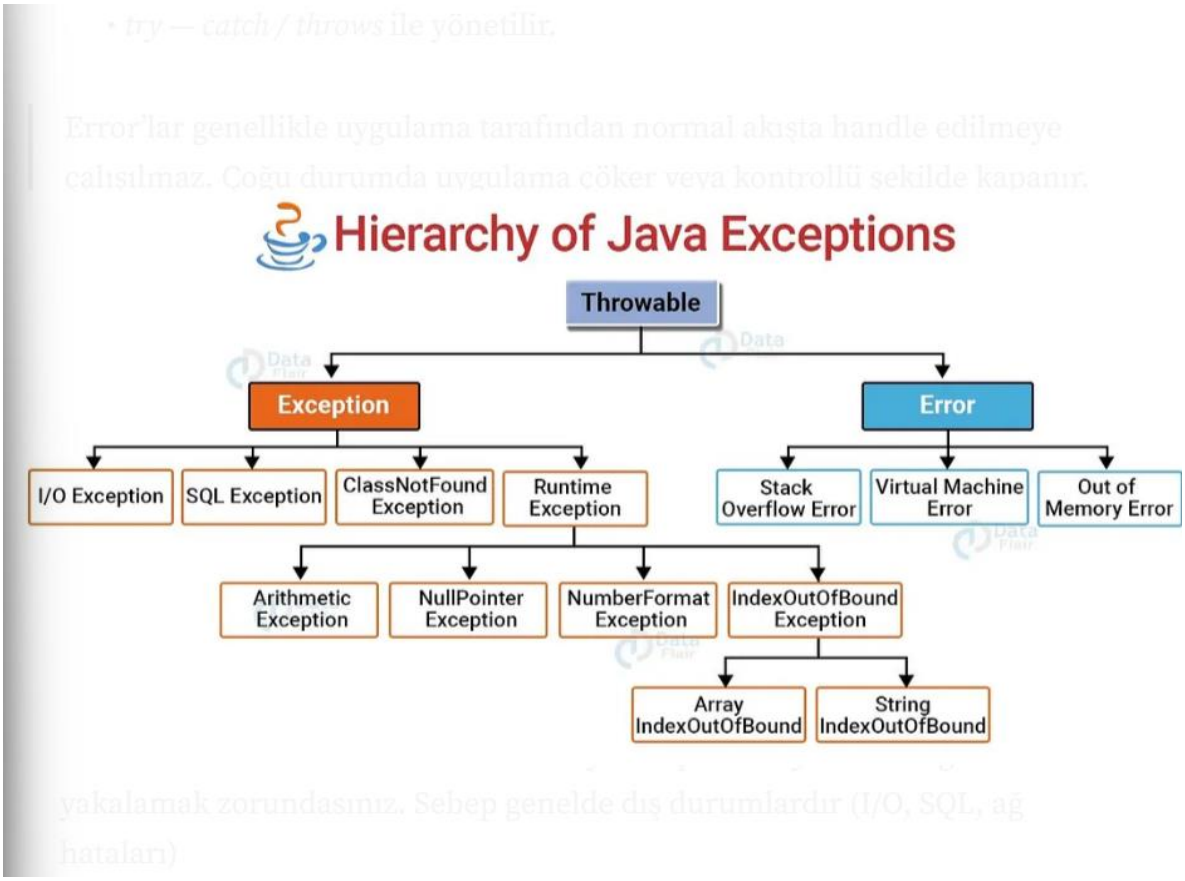
Javada Hata Yönetimi (Exception Handling)

Exception Nedir?

Programları çalıştırırken beklenmedik bir durumla karşılaştığımızda Exception Handling (İstisna Yönetimi) mekanizması devreye girer. Bu mekanizma programı beklenmedik bir hata anında aniden çökmesini engellemek için kullandığımız bir mekanizmadır.

Exception Handling'in başlıca amacı:
→ Programın tamamen çökmesini engellemek
→ Hatanın nasıl oluştuğunu anlamak anlamak
→ Gerekirse kullanıcıya anlamlı bir mesaj vermek
→ Log tutup daha sonra inceleyebilmek



Throwable Java da tüm hata (errror) ve istisnaların (exceptions) en üst sınıfıdır. Yani Java'da fırlatılabilen (*throw edilebilen*) her şey `Throwable` sınıfından türemiştir. Gündelik dilde hata ve istisna kavramları aynı gibi kullanılsa da kod tarafında bunların teknik anlamları farklıdır.

Exception(istisna):Program çalışır haldeyken oluşabilen, programcı tarafından tespit edilip çözülebilen beklenmedik durumlardır.Örneğin 0'a bölme ,var olmayan dosyayı açma ,dizinin sınırlarını aşma. Bunlar try-catch- finally bloklarıyla yönetilir.

Eroror (hata): Genellikle sistem seviyesindeki geri döndürülemez ciddi hatalardır.JVM ilgili sorunlardır. Bu hatalar yakalanmaya çalışılmaz , uygulama çöker. Örneğin Bellek yetersizliği (*OutOfMemoryError*), Stack taşması (*StackOverflowError*).

Exceptions Çeşitleri

Java'da iki tür exception vardır: Checked exception ve Unchecked exception.

Checked Exception:

Checked exception, Java derleyicisi tarafından derleme zamanında kontrol edilir ve programcının yakalamak veya belirtmek zorunda olduğu hataları temsil eder.

Bu tür hatalar, genellikle dosya okuma/yazma işlemleri, ağ bağlantıları, veritabanı işlemleri gibi dış kaynaklarla etkileşimli kod blokları sırasında oluşabilir.

Checked exception'lar, "throws" anahtar kelimesiyle belirtilir ve programcılar, bu tür hataları yakalamak veya belirtmek zorundadır. Aksi takdirde derleme zamanında hata verirler.

Checked exception'lar compile time'da kontrol edilir.
Yani ya try-catch ile yakalanmalı ya da throws ile üst metoda bildirilmelidir.
Aksi halde kod derlenmez.

IOException, SQLException, ClassNotFoundException ve InterruptedException gibi durumlar, Checked exception örnekleridir.

Unchecked Exception:

Unchecked exception, çalışma zamanında ortaya çıkan hataları temsil eder.

Bu tür hatalar, genellikle programcının kodunda yapılan hatalardan kaynaklanır, örneğin: null pointer exception, arithmetic exception vb.

Unchecked exception'lar, programcının açıkça belirtmesi veya yakalaması gerekmez. Bunun yerine, kodun düzeltilmesi veya hataların önlenmesi için tasarlanmıştır.

RuntimeException alt sınıfı, Unchecked exception örneklerini içerir. NullPointerException, IndexOutOfBoundsException, ArrayStoreException ve IllegalArgumentException gibi durumlar, Unchecked exception örnekleridir.

Throws Keyword

Bir metodun imzasında (signature) kullanılan bir anahtar kelimedir. **amacı** Metot içinde oluşabilecek istisnaları (Exceptions) o metodun içinde çözmek yerine, metodu **çağıran yapıya** bildirmektir.

Mesajı: "Ben bu işi yaparken hata çıkabilir, ben çözmüyorum, beni çağıran kişi bu hatayı `try-catch` ile çözsün veya o da başkasına fırlatsın."

Neden Kullanılır?

Sorumluluk Devri: Her hatayı oluştuğu yerde çözmek kod kalabalığına neden olabilir.

Checked Exceptions: Java'da kontrol edilmesi zorunlu olan (dosya okuma, veritabanı bağlantısı gibi) hatalarda kullanılması mecburidir.

Temiz Kod: Hata yönetimini merkezi bir yerde toplam

Throw vs Throws kavramları:

- **throw (Eylemdir):** Metot içinde hatayı **fırlatmak** (manuel hata oluşturmak) için kullanılır.

- **throws (İmzadır):** Metot başında hatayı **beyan etmek** (olabilir demek) için kullanılır.

Custom Exception

Custom Exception (Özel İstisna), programlama dillerinde geliştiricinin kendi hata türünü tanımlamasıdır.

Hazır hata sınıfları (örneğin `NullPointerException`, `IOException` gibi) bazen uygulamaya özel durumları ifade etmek için yeterli olmaz. Bu durumda **kendi hata sınıfımızı yazarız.**

Neden Custom Exception Kullanırız?

✓ Daha anlamlı hata mesajı vermek için
✓ Uygulamaya özel iş kurallarını kontrol etmek için
✓ Hataları daha düzenli yönetmek için
✓ Kodun okunabilirliğini artırmak için

Örnek olarak şunu verebiliriz:

Normal hata:

```
throw new Exception("Hata oluştu");
```

Custom Exception:

```
throw new YasSiniriAsildiException("Yaş 18'den küçük olamaz!");
```

Neden try-catch yerine throws kullanırız?

Eğer bir kütüphane veya ortak bir araç yazıyorsan, hatanın nasıl yönetileceğine sen karar vermemelisin. Kullanıcı (metodu çağıran kişi) hatayı görünce ekrana mesaj mı basmak ister, veritabanına mı kaydeder yoksa programı mı kapatır? Kararı ona bırakmak için throws ile topu ona atarsın.