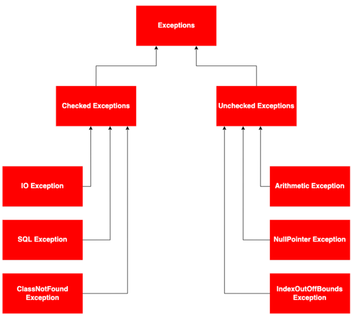
Title

Java Exceptions

. . .

Merhaba, bu yazımda Java Exception tarafına ufak bir giriş yapacağım ve Checked Exceptions ile Unchecked Exceptions kavramlarına değineceğim.

. . .



Görseli incelediğimizde de gördüğümüz gibi **Exception** türleri **Checked** ve **Unchecked** olarak ikiye ayrılır. Exceptionları ayırmamızın sebebi ise bu exceptionların farklı zaman diliminde ortaya çıkması. Gelin biraz daha detayına inelim ve örneklerle açıklamaya çalışalım.

. . .

Checked Exceptions

Checked Exceptionlar, derleyiciler tarafından anlık olarak denetlenir ve mevcut kod hatalarını derleme zamanında yakalayıp bize haber verir.

E şimdi bu ne demek ?

Şöyle bir senaryomuz olsun;

Bir dosya okuma işlemi yapmak istiyoruz ve kodumuzu Java dili kurallarına uygun yazıyoruz:

```
import java.io.FileReader;

public class Main {

    public static void main(String[] args) {
        final FileReader file = new FileReader( fileName: "example.txt");
    }
}
```

Add exception to method signature
Surround with try/catch
Surround with try-with-resources block
Press ⇧⇧ to open preview

Görünürde kodumuzda hata yoktu ve bu kod derlenecek ve çalışacak gibi duruyordu. Fakat tam bu noktada araya **Checked Exception** kavramı giriyor.

Derleyici bize, kodu try-catch bloğu arasına almamızı ya da ilgili exception'ı method imzası olarak belirtmemiz gerektiğini söylüyor.

Bunu istemesinin sebebi şu;
Biz kodumuzu doğru şekilde yazdık ama ya üzerinde işlem yapmak istediğimiz dosya mevcut değilse ?

Derleyicinin bizi uyardığı şekilde kodumuzu düzenliyoruz:

```
import java.io.FileNotFoundException;
import java.io.FileReader;

public class Main {

    public static void main(String[] args) throws FileNotFoundException {
        final FileReader file = new FileReader( fileName: "example.txt");
    }
}
```

ya da:

```
import java.io.FileNotFoundException;
import java.io.FileReader;

public class Main {

    public static void main(String[] args) {
        try {
            final FileReader file = new FileReader( fileName: "example.txt");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

Bu sayede artık kodumuz doğru bir şekilde derlenecektir ve eğer belirtilen dosya mevcut ise hata vermeden çalışacaktır.

Özetlemek gerekirse örneğimizdeki gibi derleme zamanında ortaya çıkan istisnalar **Checked Exceptions** kavramı altında ele alınmaktadır.

. . .

Unchecked Exceptions

Unchecked Exceptionlar, derleyiciler tarafından denetlenemez ve mevcut kod hatalarını çalışma zamanında yakalayıp bize haber verir.

E şimdi bu ne demek ?

Şöyle bir senaryomuz olsun;

Bir dosya okuma işlemi yapmak istiyoruz ve kodumuzu Java dili kurallarına uygun yazıyoruz fakat dosyamızı bu sefer senaryomuz gereği *null* olarak işaretleyelim:

```
import java.io.FileReader;
import java.io.IOException;

public class Main {

    public static void main(String[] args) {
        try {
            FileReader file = null;
            assert false;
            file.read();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Görünürde kodumuzda hata yok ve bu kod derlenecek ve çalışacak gibi duruyor. Fakat tam bu noktada araya **Unchecked Exception** kavramı giriyor.

Kod derleme zamanında herhangi bir exception belirtmedi fakat çalışma zamanında ne oldu dersiniz ?

```
Exception in thread "main" java.lang.NullPointerException: Create breakpoint : Cannot invoke "java.io
    .FileReader.read()" because "file" is null
    at Main.main(Main.java:10)
```

NullPointerException evet, aramıza hoşgeldiniz :)

Özetlemek gerekirse örneğimizdeki gibi derleme zamanında gözden kaçabilen fakat çalışma zamanında ortaya çıkan istisnalar **Unchecked Exceptions** kavramı altında ele alınmaktadır.

. . .

Umarım bu makale Java geliştiricilerine bir nebze de olsa katkı sağlamıştır.

Başka bir yazıda görüşmek üzere.

İstek ve önerileriniz için bana batuhankiltac@gmail.com adresinden ulaşabilirsiniz.