

# Flutter ile Uygulama Geliştirme Kursu | Android & IOS

## İleri Dart

Kasım ADALAN

Elektronik ve Haberleşme Mühendisi

Android - IOS Developer and Trainer

# Eğitim İçeriği

1. Try – Catch Yapısı
2. Asenkron İşlemler

# Exception Nedir ?

- Derleyici Hatası (Compiler Error) : Derleme öncesi yakalanan hatalar
  - Örn: karakter hataları, sentaks hatası, ...
- Hata (Exception): Çalışma anında (runtime) gerçekleşen hatalar
  - Örn: Sistem hataları, cihaz hataları, dosya bulunamadı, dizi indeksi aşıldı, ...

Exception Hata Ayıklama

# try catch

- Derleme sırasında oluşabilecek hatalar için kullanılır.
- Genelde kotlin input – output işlemleri için kullanılır. Yani veri alışveriş işlemlerinde kullanılır.
- Kullanılacak yer mutlaka hata fırlatmalıdır.

```
try{
```

```
//Kontrol edilecek kodlama buraya yazılır.
```

```
}catch (e){
```

```
//Hata oluşunca burası çalışır.
```

```
}
```

# try catch bloğu

```
var sayilar = List<int>();
```

```
sayilar.add(34); //0. indeks
```

```
sayilar.add(67); //1. indeks
```

```
try{
```

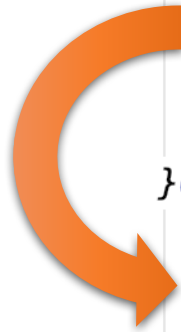
```
    sayilar[2] = 45;
```

```
    print("İşlem tamam");
```

```
}catch(e){
```

```
    print("Listenin boyutunu aştınız");
```

```
}
```



Asenkron işlemler

# Future, async ve await

- **Future** : Fonksiyon tanımlamasında fonksiyon isminden önce gelir. Metodun asenkron olarak çalışacağını ve **await** metodu ile karşılaştığı zaman çalışmasını askıya alacağını belirtir.
- **async** : Fonksiyon isminden sonra gelir ve asenkron çalıştırmak istediğimiz yapılarda kullanırız, yani aynı anda birden fazla işlem yaptırmak için kullanırız.
  - **Örn** : Dosya işlemleri , İnternette veri alırken gibi.
- **await** : Sadece async fonksiyonların içinde kullanılır. Amaç asenkron işlem yaparken yarım kalan bazı kodlamalar hata oluşturabilir , hata oluşturmaması için await kullanılır ve asenkron işlem içinde o kodlamanın bitmesi beklenir.



# Asenkron İşlemler

## Örnek

//Async olursa satır önceliği değişir.  
 //await ile bir satırın çalışmasını işini bitirene kadar bekletebiliriz.  
 //Örnek olarak var veri = await veritabanındanVeriAl(); çalışmasını, bitene kadar bekletmezsek  
 //print('Alınan veri : \$veri'); satırı daha önce çalışmak isteyebilir ve hata oluşur.  
 //Bunun sebebi : veritabanındanVeriAl() metodunun işi bitmediği için veri değişkenine ilgili veriyi aktaramaz.  
 //Veriyi aktaramadığı için print('Alınan veri : \$veri'); satırında hata oluşur çünkü veri boştur.

```
Future<void> main() async { await kullanacağımız için bu metod async olmalıdır.
  print('Verilerin alınması bekleniyor...');
  var veri = await veritabanındanVeriAl();
  print('Veri alınıyor...');
  print('Alınan veri : $veri');
}
```

```
Future<String> veritabanındanVeriAl() async {
  //İki adet Future kodlaması yer aldığı için asenkron çalışırlar.
  //Her ikiside ayrı ayrı işlemler yaparlar.

  for (var i = 1; i <= 5; i++) {//Temsili olarak veri alınma miktarını gösterir.
    Future.delayed(Duration(seconds: i), () => print("Alınan veri miktarı : ${i*20}"));
  }

  //Süre dolduğunda metodun geri dönüş değerini return ile metodun kullanıldığı yere iletilir.
  return Future.delayed(Duration(seconds: 5), () => 'Veritabanı veri kümesi');
}
```

```
Verilerin alınması bekleniyor...
Alınan veri miktarı : %20
Alınan veri miktarı : %40
Alınan veri miktarı : %60
Alınan veri miktarı : %80
Alınan veri miktarı : %100
Veri alınıyor...
Alınan veri : Veritabanı veri kümesi
```

await olmazsa

```
Verilerin alınması bekleniyor...
Veri alınıyor...
Alınan veri : Instance of 'Future<String>'
Alınan veri miktarı : %20
Alınan veri miktarı : %40
Alınan veri miktarı : %60
Alınan veri miktarı : %80
Alınan veri miktarı : %100
```

Teşekkürler...



kasım-adalan



kasimadalan@gmail.com



kasimadalan