13.10.2025 Ders Notları

Meta Android Developer Professional Certificate - Ders Notları

Tarih: 13.10.2025

İçerdiği Konular: Singleton pattern, object declaration, companion object, nested class, inner class, constant değerler

---

Singleton Pattern

Singleton denen şey aslında bir classtan sadece tek bir instance (örnek) oluşturabilmemizi sağlayan bir pattern. Yani bir nevi "tek instance kuralı" gibi düşünebiliriz.

Normalde class'lardan istediğimiz kadar instance oluşturabiliyoruz ya, işte singleton buna izin vermiyor. **Sadece bir tane instance** olabiliyor.

Neden Kullanıyoruz?

- Aynı resource'a (kaynağa) erişimi kontrol altına almak için
- Çakışmaları önlemek için
- Her yerden erişilebilir tek bir nokta istediğimizde

Mesela Little Lemon restoranının müşteri listesi gibi düşün - bu listeye her yerden erişebilmek istiyorsun ama sadece bir tane liste olmalı. İşte burada singleton kullanmak mantıklı.

Kotlin'de Nasıl Yapılıyor?

`class` yerine `object` keyword'unu kullanıyoruz:

```
object RestaurantTables {
    val customers = mutableListOf<String>()

    fun addCustomer(customer: String) {
        customers.add(customer)
    }

    fun removeCustomer(customer: String) {
        customers.remove(customer)
    }
}
```

Burada dikkat: Bu object'i instantiate edemiyorsun! Yani `RestaurantTables()` diyemiyorsun, direk `RestaurantTables.addCustomer("Ahmet")` şeklinde kullanıyorsun.

Not: Singleton'ın constructor'ı private olmalı ama Kotlin'de object declaration bunu otomatik hallediyor zaten.

Companion Object - Static Gibi Bir Şey

Kotlin'de **static** keyword'ü yok, onun yerine **companion object** kullanıyoruz.

Companion Object Ne İşe Yarıyor?

- Tüm instance'lar arasında paylaşılan değerler tutmak için
- Class'a ait "sabit" şeyler için
- Factory method'lar oluşturmak için

Örnek: Waiter class'ı düşünelim - her garsonun kendi adı, ID'si var ama tüm garsonlar aynı şubenin çalışanı. İşte şube adı companion object'te olmalı.

```
class Waiter : Personnel {
    var id: Int = -1
    var name: String = ""

    override fun serveCustomer() {
        // müşteri servis etme kodu
    }

    companion object {
        var branchName: String = "Little Lemon Main Branch"
        var branchAddress: String = "New York, Fifth Avenue, 25"
    }
}
```

Kullanımı: `Waiter.branchName` şeklinde - instance oluşturmaya gerek yok!

Not: Her class'ın sadece bir tane companion object'i olabilir. İsim vermek zorunlu değil, genelde isimsiz kullanılıyor.

Constant Değerler ve Fonksiyonlar

Companion object içinde **sabit değerler** (constant) ve **fonksiyonlar** tanımlayabiliyoruz.

Real-life Örnek: Restaurant Faturası

Little Lemon için fatura yazılımı yaptığımızı düşünelim:

```
class Order(val amountBeforeTax: Double) {
    companion object {
        const val TAX_PERCENTAGE = 5

        fun getNetOrderAmount(amountBeforeTax: Double): Double {
            val taxAmount = amountBeforeTax * TAX_PERCENTAGE / 100
            return amountBeforeTax + taxAmount
        }
    }
}
```

Kullanım:

```
val netAmount = Order.getNetOrderAmount(500.0) // 525.0 döndürür
```

Bunu daha iyi anlamak için pratik yapmam lazım - companion içindeki fonksiyonların ne zaman kullanılacağı kafamı karıştırabiliyor.

Inner Class ve Nested Class

Nested Class (İç İçe Class)

Bir class'ın içinde başka class tanımlamak:

```
class Order {
    class DeliveryDetails {
        var deliveryAddress: String = ""
        var deliveryFee: Double = 0.0
    }
}
```

Önemli: Nested class, outer class'ın member'larına erişemez! Static gibi davranır.

Inner Class

Eğer nested class'ın outer class'ın member'larına erişmesini istiyorsak `inner` keyword kullanıyoruz:

```
class Order(val orderAmount: Double) {
    inner class TaxDetails {
        fun calculateTax(): Double {
            return orderAmount * 0.05 // outer class'ın orderAmount'ına
erişebiliyor!
        }
    }
}
```

Farkı: Inner class outer class'ın tüm property'lerine erişebilir, nested class erişemez.

Inner class konusu biraz karmaşık, bunu projede kullanınca daha iyi anlayacağım sanırım.

Android Studio'da Singleton Oluşturma

1. Package'a sağ tıkla → New → Kotlin Class/File
2. Açılan pencereden **Object** seç
3. İsmini ver ve içini doldur

Kod stili için not: Companion object'leri class'ın en altına yazmak convention (genel kabul). Daha okunaklı oluyor.

Kişisel Yorum

Singleton ve companion object gerçekten çok kullanışlı, özellikle Android development'ta sık karşılaşıyorum. Ama ne zaman singleton, ne zaman companion object kullanacağım konusunda daha pratik yapmam gerekiyor.

Tekrarlamak gerekirse:

- **Singleton** → Tek instance lazımsa
- **Companion Object** → Static benzeri davranış lazımsa
- **Inner Class** → Outer class'ın member'larına ihtiyaç varsa
- **Nested Class** → Sadece gruplama için
