



Android lecture 7

Kotlin Multiplatform

January 2024

Lukáš Prokop, Simona Kurňavová



Agenda

- Kotlin Multiplatform
- Architecture of KMM
- iOS development (Swift, SwiftUI)

Kotlin Multiplatform (KMP)

- **Documentation:** <https://kotlinlang.org/docs/multiplatform.html>
- JetBrains
- Announced in 2017
- Cross-platform development
- Allows to write shared business logic in Kotlin and use it across multiple platforms
- IDEs support
- Built on Kotlin/Native - technology and compiler that allows Kotlin code to be compiled directly into native binaries for various platforms, without relying on a virtual machine
- Primary used for iOS and Android (KMM)
- Kotlin Multiplatform Gradle plugin

Kotlin Multiplatform Mobile (KMM)

- Mobile development – iOS, Android
- Shared code (Kotlin)
- Modules dedicated to both platforms (Kotlin, Swift)
- Compose Multiplatform (in Alpha)

Kotlin Multiplatform Mobile (KMM)

- **Pros:**

- Reduced duplicity of code
- Reduced development time
- Consistent architecture
- Shared libraries
- Native code for UI for both platforms (better UX)

- **Cons:**

- Platform specific code often needed
- Complex project (bigger repository, bigger dev team, ...)
- Not all libraries/tools support KMM
- Switching between two IDEs

Prerequisites for KMM project

- MacOS
- Android Studio + Kotlin Multiplatform Mobile plugin
- Xcode
- *Kdoctor (optional):*


















```
Environment diagnose (to see all details, use -v option):
[✓] Operation System
[✓] Java
[✓] Android Studio
[*] Xcode
  * Current command line tools: /Library/Developer/CommandLineTools
  You have to select command line tools bundled to Xcode
  Command line tools can be configured in Xcode -> Settings -> Locations -> Locations
[!] CocoaPods
  ! CocoaPods configuration is not required, but highly recommended for full-fledged development
  * System ruby is currently used
  CocoaPods is not compatible with system ruby installation on Apple M1 computers.
  Please install ruby via Homebrew, rvm, rbenv or other tool and make it default
  Detailed information: https://stackoverflow.com/questions/64901180/how-to-run-cocoapods-on-apple-silicon-m1/66556339#66556339
  * cocoapods not found
  Get cocoapods from https://guides.cocoapods.org/using/getting-started.html#installation













Conclusion:
  * KDoctor has diagnosed one or more problems while checking your environment.
  Please check the output for problem description and possible solutions.
```

Structure of KMM project

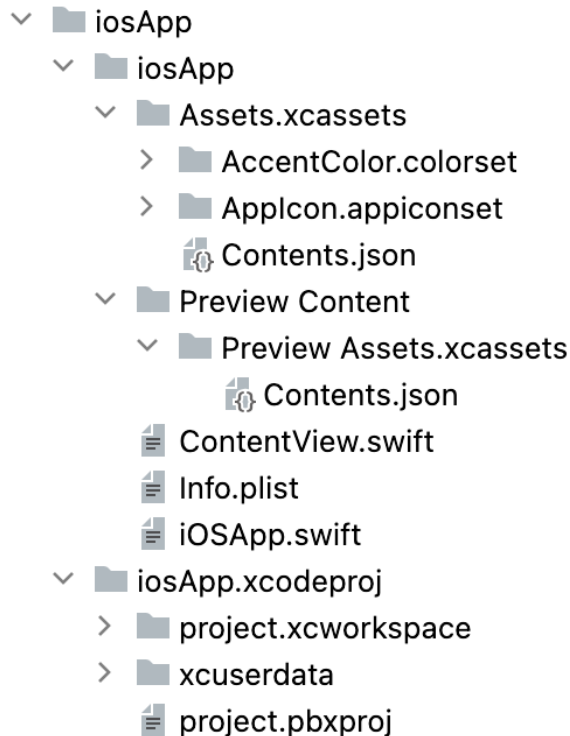
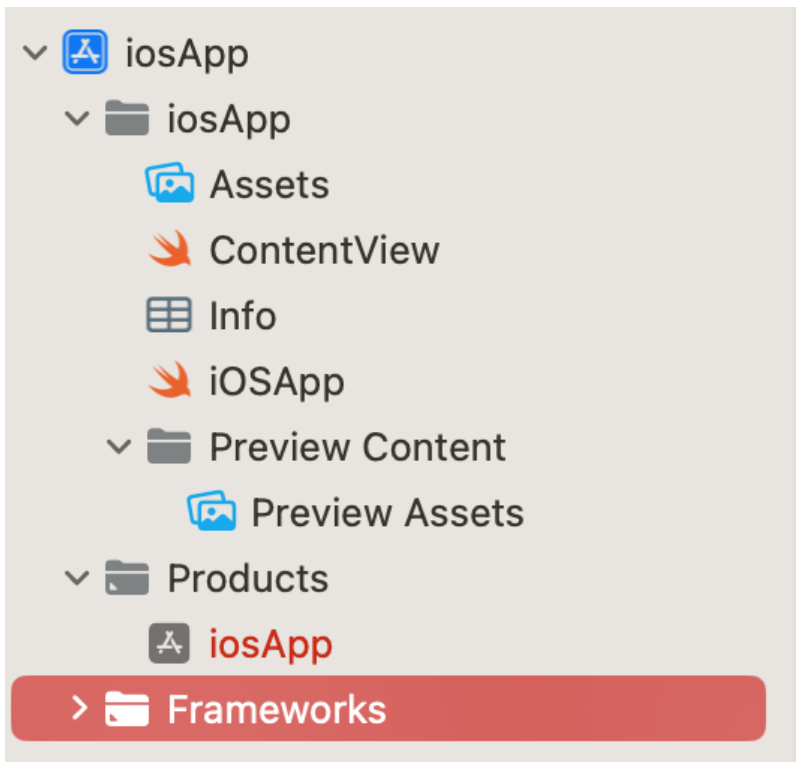
- **shared Module:** business logic, data models, no platform specific code
 - **commonMain:** common Kotlin code shared between Android and iOS.
 - **androidMain:** Android-specific code that extends or implements the common code.
 - **iosMain:** iOS-specific code that extends or implements the common code.
- **androidApp Module:** specific to the Android platform, includes the Android app-specific code and dependencies.
- **iosApp Module:** specific to the iOS platform, includes the iOS app-specific code and dependencies.

Structure of KMM project

- >  .gradle
- >  **androidApp**
- >  gradle
- >  iosApp
- ▼  **shared**
 - ▼  src
 - >  androidMain **[main]**
 - >  **commonMain**
 - >  **iosMain**
 -  build.gradle.kts
-  .gitignore
-  build.gradle.kts
-  gradle.properties
-  gradlew
-  gradlew.bat
-  local.properties
-  settings.gradle.kts

- ▼  **androidApp**
 - ▼  src
 - ▼  **main**
 - ▼  java
 - ▼  com.example.mytestkmm.android
 -  MainActivity.kt
 -  MyApplicationTheme.kt
 - ▼  res
 - ▼  values
 -  styles.xml
 -  AndroidManifest.xml
 -  build.gradle.kts

Structure of KMM project: Xcode vs. Android Studio



expect & actual

```
expect class PlatformSpecificClass { // Shared code in commonMain
    fun platformSpecificFunction(): String
}
```

```
actual class PlatformSpecificClass actual constructor() { // Android-specific in androidMain
    actual fun platformSpecificFunction(): String {
        return "Android implementation"
    }
}
```

```
actual class PlatformSpecificClass actual constructor() { // iOS-specific in iosMain
    actual fun platformSpecificFunction(): String {
        return "iOS implementation"
    }
}
```

Development/Testing in KMM

- Create KMM project in Android Studio
- Shared code - developed in AS (shared module)
 - Kotlin, Gradle, Kotlin/Native, *only* multiplatform libraries
- Android - developed and tested in AS
 - Kotlin, Gradle
- **iOS** – developed and tested in Xcode
 - Swift, CocoaPods, Carthage, SwiftPackageManager

Multiplatform libraries

- Kotlin Coroutines
- Ktor
- Fuel
- kotlinx.serialization
- Kodein-DI
- Koin
- SQLDelight
- <https://github.com/terrakok/kmp-awesome>

Apple iOS

- Used on: iPhone, iPad, iPod touch, Apple TV, Apple Watch
- App Store
- Secure and **closed** ecosystem
- Tight integration with other Apple devices and services through features like Handoff, AirDrop, and Continuity.
- Apple controls hardware and software with their products



Apple iOS – version summary

- iOS 1 (2007): original iOS with first iPhone, Safari, apps primary web-based, third-party development not supported
- iOS 2 (2008): App Store
- iOS 3 (2009): Copy-Paste, Spotlight Search.
- iOS 4 (2010): Multitasking, FaceTime.
- iOS 5 (2011): Notification Center, iMessage.
- iOS 6 (2012): Apple Maps, Passbook.
- iOS 7 (2013): Flat design, Control Center.
- iOS 8 (2014): HealthKit, HomeKit, Extensions.
- iOS 9 (2015): Split View on iPad, Siri improvements.
- iOS 10 (2016): Widgets, iMessage apps.
- iOS 11 (2017): Files app, ARKit.
- iOS 12 (2018): Performance improvements, Screen Time.
- iOS 13 (2019): Dark Mode, Sign In with Apple.
- iOS 14 (2020): Widgets, App Library, App Clips.
- iOS 15 (2021): Changes to Apple maps.
- iOS 16 (2022): Redesigned lock screen.
- iOS 17 (2023): StandBy, Interactive widgets.

iOS development

- **Swift**: introduced in 2014
- **Objective-C**: used in old apps, new apps should use Swift
- **Xcode IDE**: editor, simulation, debugging
- **UIKit, SwiftUI**
- Apps distributed through the Apple App Store
- Apple Developer Program
- Development possible only on MacOS

iOS vs. Android comparison

	iOS	Android
Fragmentation	One manufacturer	Multiple manufacturers and range of devices
App distribution	App store	Google play, third party stores
Updates	Consistent across all devices	Often delays due to updates going through different manufacturers
Technology	Swift, Objective-C, UIKit, SwiftUI	Kotlin, Java, XML, Compose
Development environment	Xcode (on MacOS only)	Android Studio (Windows, MacOS, Linux), IntelliJ, Eclipse

Swift

- **Documentation:** <https://developer.apple.com/swift/>
- Developed by Apple for iOS, macOS, watchOS, and tvOS
- Open source
- Swift playground (ipad, Mac)
- Xcode support

Swift

- Interoperable with Objective-C
- Comparable speed to low-level languages.
- Automatic Reference Counting (ARC) for efficient memory management.
- Typesafe: optionals, type inference, and static typing
- Robust error-handling model

Swift/Kotlin interop

- <https://kotlinlang.org/docs/native-objc-interop.html>
- Kotlin interface -> Swift/ObjC protocols – loose generics
 - Issue with kotlin Flow
- Sealed classes -> simple classes
 - Using Swift switch is cumbersome, non-reachable default branch
- Inline classes
 - i.e. `kotlin.Result` -> Generics in swift downcasted to Any?
- <https://medium.com/@aoriani/list/writing-swiftfriendly-kotlin-multiplatform-apis-c51c2b317fce>

Swift – Syntax

```
var optionalVariable: Int? = 42
let myConstant: String = "Swift"
```

```
if condition {
    // Code for true condition
} else {
    // Code for false condition
}
```

```
for item in array {
    // Code for each item
}
```

```
func greet(name: String = "Alice") -> String {
    return "Hello, \(name)!" // Interpolation
}
```

```
class Person {
    var name: String
    init(name: String) {
        self.name = name
    }
}
```

```
struct Point {
    var x, y: Double
}
```

Swift – Safe access

```
let length: Int? = nullableString?.count
```

```
let result: String = nullableString ?? "Default Value"
```

```
if let nullableString {  
    print("Hey, \(nullableString)")  
} // Doesn't print anything if nullableString is nil.
```

Swift – Protocol, extension

```
protocol Printable {  
    var description: String { get }  
}
```

```
extension String {  
    func reverse() -> String {  
        return String(self.reversed())  
    }  
}
```

Swift – inout and labels

```
func squareInPlace(_ number: inout Int) {  
    number = number * number  
}  
  
var myNumber = 5  
squareInPlace(&myNumber) // myNumber = 25
```

```
func sayHello(to name: String) {  
    print("Hello, \(name)!")  
}  
  
sayHello(to: "Taylor") // Different label for caller
```

Swift – Error handling

```
func checkPassword(_ password: String) throws -> Bool {  
    if password == "password" {  
        throw PasswordError.obvious  
    }  
    return true  
}
```

```
do {  
    try checkPassword("password")  
    print("That password is good!")  
} catch {  
    print("You can't use that password.")  
}
```


SwiftUI

- UI framework by Apple
- Designed to be cross-platform (iOS, macOS, watchOS, tvOS)
- Uses declarative syntax
- Everything is View

SwiftUI: Views

- Lightweight structures that define a portion of the user interface.
- **Core views:** Text, Image, Button, TextField, Toggle, ProgressView, MapView, Divider, Spacer, ...
- **Layouts and stacks:** HStack, VStack, ZStack
- **Modifiers:** operations that change the characteristics of a view. '
 - opacity, font, color, border, padding, ...
 - **The order of modifiers matters!**

SwiftUI: Custom view

```
struct MyView: View {  
    var body: some View {  
        VStack {  
            Text("Hello, World!")  
                .opacity(0.5) // Display partially transparent text.  
                .font(.largeTitle)  
                .bold();  
            Button(action: {}) {  
                Text("OK")  
            }  
        }  
    }  
}
```

SwiftUI: Property Wrappers

Mechanism used for managing state and data flow between different parts of your SwiftUI views.

- **@State** - declares a property whose changes can trigger view updates
- **@Binding** - Used to pass state from a parent view to a child view and allow the child view to modify that state.
- **@ObservedObject** - Used to declare a property that is an instance of an ObservableObject, which notifies the view when its underlying data changes.

SwiftUI: Property Wrappers - @State

```
struct ContentView: View {  
    @State private var counter = 0  
  
    var body: some View {  
        Text("Counter: \(counter)")  
            .onTapGesture {  
                counter += 1  
            }  
    }  
}
```

SwiftUI: Property Wrappers - @Binding

```
struct ParentView: View {  
    @State private var counter = 0  
    var body: some View {  
        ChildView(counter:$counter)  
    }  
}
```

```
struct ChildView: View {  
    @Binding var counter: Int  
    var body: some View {  
        Text "Counter: \(counter)"  
        .onTapGesture {  
            counter += 1  
        }  
    }  
}
```

SwiftUI: Property Wrappers - @ObservedObject

```
class UserSettings: ObservableObject {  
    @Published var darkModeEnabled = false  
}
```

```
struct ContentView: View {  
    @ObservedObject var userSettings = UserSettings()  
  
    var body: some View {  
        VStack {  
            Toggle("Dark Mode", isOn: $userSettings.darkModeEnabled)  
            Text("Dark Mode is \ \(userSettings.darkModeEnabled ? "Enabled" : "Disabled")")  
        }  
    }  
}
```

Demo

- Simple iOS and Android app using KMM

**Thank you for
attention**

Gen™

GenDigital.com

