iOS Development with Kotlin Multiplatform

Tips & Tricks

Pamela Hill & Tadeas Kriz - 30 November 2023

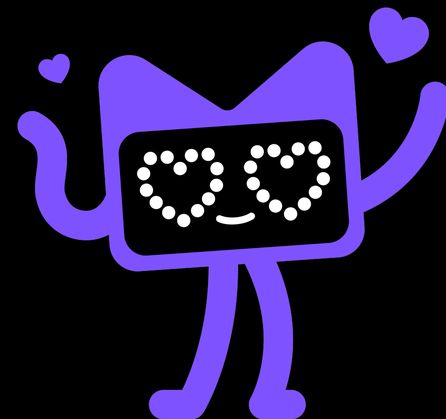

Four Webinars on Kotlin Multiplatform Development
REGISTER
Live Webinars
November 21, 2023 – 5:00 PM UTC
November 23, 2023 – 4:00 PM UTC
November 28, 2023 – 4:00 PM UTC
November 30, 2023 – 4:00 PM UTC

Access to Resources

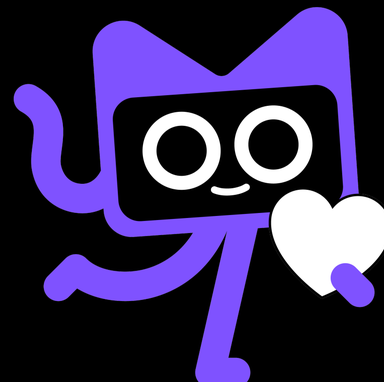
- This webinar is being recorded
- Slides can be found here: kotl.in/ios-dev-kmp

What We Will Discuss

- Kotlin/Swift Comparison: Why Can’t We Be Friends?
- Current State of Kotlin/Swift Interop
- Language Features and How to Tame Them
- Helpful Resources



Kotlin/Swift Comparison



We Have So Much in Common!

Paradigms

- Object-oriented*: classes, interfaces (or protocols), inheritance
- Functional: lambdas (or closures)

We Have So Much in Common!

Type systems

- Similarity of types and collections
- Null (or nil) safety
  - Kotlin: Null-safety checks/operators and !!
  - Swift: Null-safety checks like if let / let guard and !
- Mutability / Immutability* of variables and collections
  - Kotlin: val or var for variables, type for collections
  - Swift: let or var for variables and collections

We Have So Much in Common!

Asynchronous code

- Similar ways of writing asynchronous, parallel code
  - Kotlin: coroutines, suspend functions
  - Swift: Tasks, async functions
- Streams:
  - Kotlin: Flows
  - Swift: AsyncSequence, Combine

BASICS

Hello World

Swift

```swift
print("Hello, world!")
```

Kotlin

```kotlin
println("Hello, world!")
```

Variables And Constants

Swift

```swift
var myVariable = 42
myVariable = 50
let myConstant = 42
```

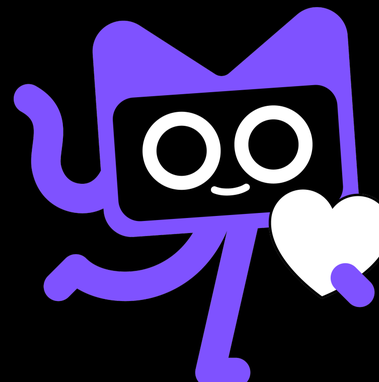
Kotlin

```kotlin
var myVariable = 42
myVariable = 50
val myConstant = 42
```

Explicit Types

nilhcem.com/swift-is-like-kotlin/

Current State of Kotlin/Swift Interop



Current state

- Kotlin Multiplatform supports direct interoperability with Objective-C
- Swift interoperability **via Objective-C bridge**
- This means:
  - Some features work exactly as you expect
  - Some features work with a slight modification
  - Some features need a workaround / plugins or libraries
  - Some features are not supported at all

The Kotlin team is working on direct Swift interoperability but there’s no prototype yet.

Kotlin/Swift Interopedia

kotl.in/interopedia

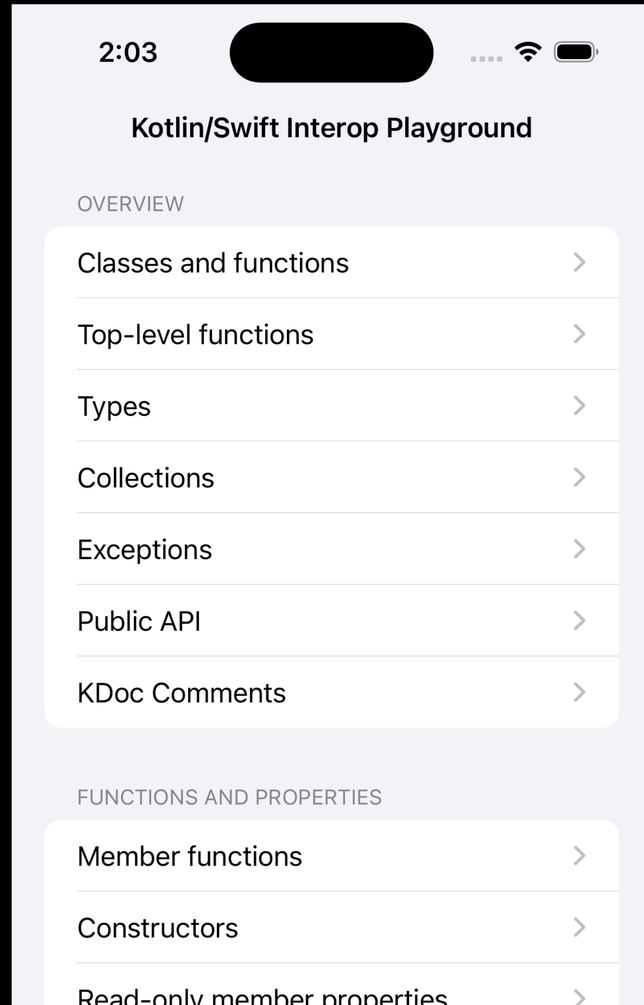
kotlin-swift-interopedia
forked from hhru/kotlin-swift-interopedia

Overview

Top-level functions	You can access a top-level function via the wrapper class: TopLevelFunctionKt.topLevelFunction().
Exceptions	If you invoke a Kotlin function that throws an exception and doesn't declare it with `@Throws`, that crashes the app. Declared exceptions are converted to NSError and must be handled.
Public API	Public classes, functions, and properties are visible from Swift. Marking classes, functions, and properties internal will exclude them from the public API of the shared code, and they will not be visible in Swift.
KDoc comments	You can see certain KDoc comments at development time. In Xcode, use Option+Double left click to see the docs. Note that many KDocs features don't work in Xcode, like properties on constructors (@property) aren't visible. In Fleet, use the 'Show Documentation' action.

Functions and properties

Member functions	You can call public member functions from Swift. Internal or private declarations aren't visible.
Constructor	You call constructors to create Kotlin classes from Swift.



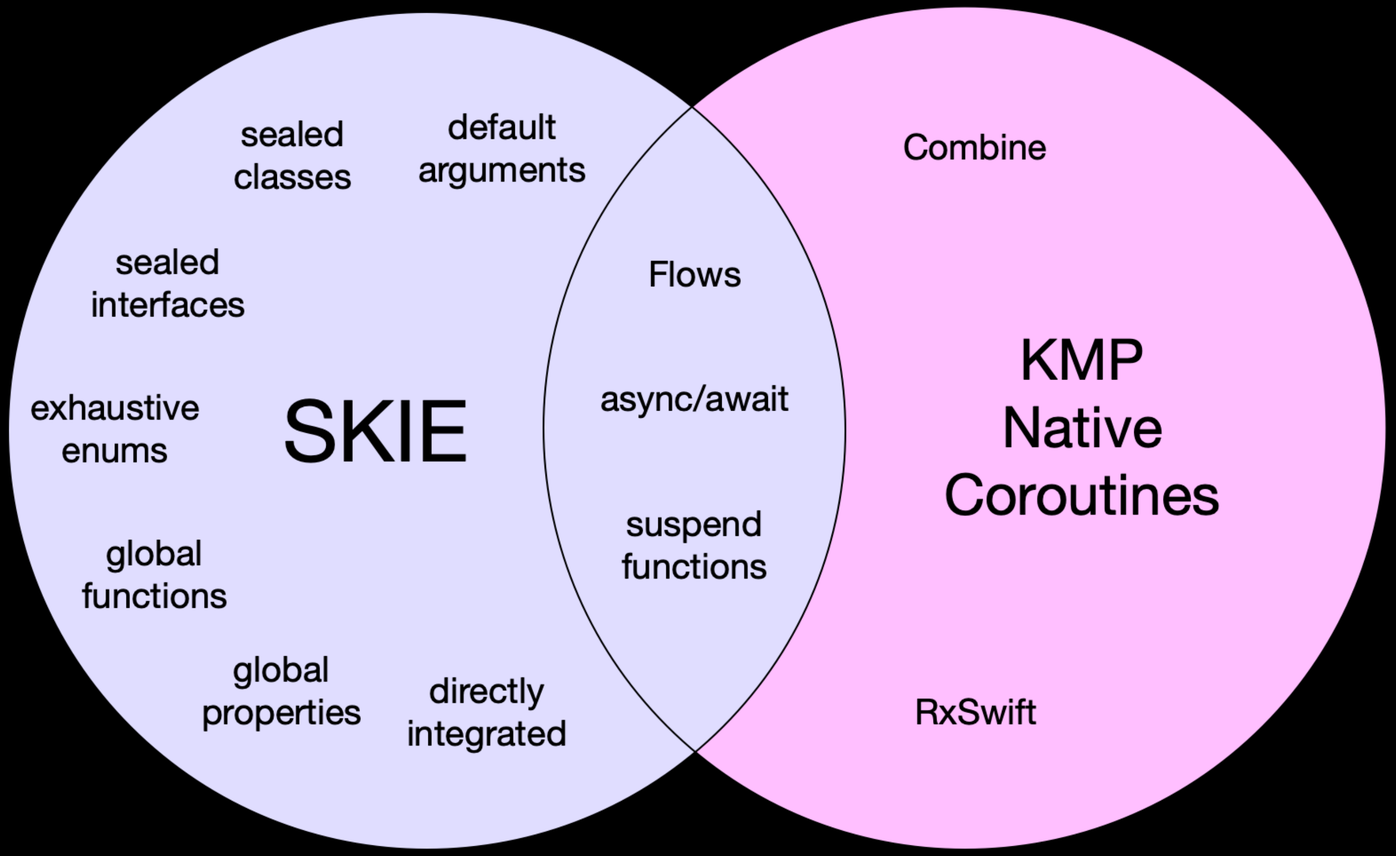
What is SKIE? 

- Plugin by Touchlab
- Narrows gap between Kotlin and Swift
  - Enums, sealed hierarchies, suspend functions, Flows and more
- Directly linked in Kotlin framework
- Thoroughly tested
- Stepping stone to official Swift interop

What is KMP-NativeCoroutines?

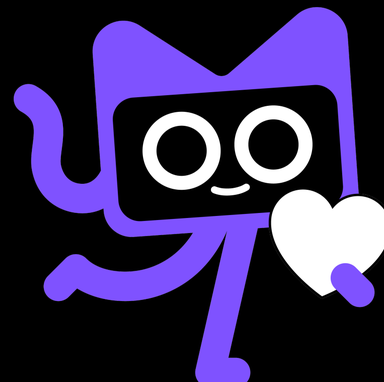
- Plugin and library by Rick Clephas
- Helps wrap suspend functions and Flows

Features


sealed classes
default arguments
sealed interfaces
exhaustive enums
SKIE
global functions
global properties
directly integrated
Flows
async/await
suspend functions
Combine
KMP Native Coroutines
RxSwift

Thank you to our
open source developers!

Language Features and How to Tame Them



Language features we'll be covering

- Basics (Classes, properties, functions / Top-level properties and functions)
- Types (Basic types / Optional basic types / Collections)
- Functions (Default arguments)
- Object-oriented programming (Sealed classes and interfaces)
- Asynchronous programming (Suspend functions / Flows)
- Experimental language features (annotations)

Language features we'll be covering

- Basics (Classes, properties, functions / Top-level properties and functions)
- Types (Basic types / Optional basic types / Collections)
- Functions (Default arguments)
- Object-oriented programming (Sealed classes and interfaces)
- Asynchronous programming (Suspend functions / Flows)
- Experimental language features (annotations)

Some features work exactly as you expect

Basics

Classes, properties, functions

Kotlin

```kotlin
class Printer {

    1 usage
    fun printSomething(){
        println("Something")
    }
}
```

Swift

```swift
let printer = Printer()
printer.printSomething()
```

Some features work with a slight modification

Basics

Top-level properties, functions

Kotlin

In Constants.kt

```kotlin
val apiKey = "A top-level property"
```

In Messages.kt

```kotlin
fun printMessage() {
    println("Hello from top-level function")
}
```

Swift

```swift
func topLevelPropertyExample() {
    print(ConstantsKt.apiKey)
}
```

```swift
func topLevelFunctionExample() {
    MessagesKt.printMessage()
}
```

Language features we'll be covering

- Basics (Classes, properties, functions / Top-level properties and functions)
- Types (Basic types / Optional basic types / Collections)
- Functions (Default arguments)
- Object-oriented programming (Sealed classes and interfaces)
- Asynchronous programming (Suspend functions / Flows)
- Experimental language features (annotations)

Types

Basic types (primitives and String)

- Similar basic types, may require mapping for integer types and Char
  - Kotlin's `Int` is converted to `Int32`, `Long` to `Int64`
  - Using literals or arguments of Swift types requires no additional effort
  - Using Swift type `Int` requires mapping to other Swift types
  - Kotlin's `Char` is converted to `unichar`, which is inconvenient

Basics

Optional basic types

- Some basic types require mapping into special optional (`nil`lable) types
  - Passing literals or `nil` values as arguments requires no additional effort
  - Using Swift types requires additional mapping (say `Int32` -> `KotlinInt`)
  - Using optional Swift types requires a `nil` check

Types
Collections

- Collections types are similar, only primitives need explicit mappings
  - Passing literals to a Kotlin function works requires no additional effort
  - Passing Swift types to a Kotlin function requires a mapping
  - For example: `List<Int>` <-> `[KotlinInt]`

Language features we'll be covering

- Basics (Classes, properties, functions / Top-level properties and functions)
- Types (Basic types / Optional basic types / Collections)
- Functions (Default arguments)
- Object-oriented programming (Sealed classes and interfaces)
- Asynchronous programming (Suspend functions / Flows)
- Experimental language features (annotations)

Some features need a workaround / plugins or libraries

Functions

Default arguments

Kotlin

```kotlin
class Shop {

    2 usages
    fun orderItems(item: String,
                   quantity: Int = 1) {
        print("Ordered")
    }
}
```

Swift

```swift
Shop().orderItems(item: "1", quantity: 100)
```

Default arguments with SKIE

- Overloads available for all combinations
- Disabled by default
- Use annotation to enable
  - `@DefaultInterop.Enable`

SKIE and suspend functions

Setup: shared/build.gradle.kts

```
plugins {this: PluginDependenciesSpecScope
    kotlin("multiplatform")
    id("com.android.library")
    id("co.touchlab.skie") version ("0.5.6")
}
```

Default arguments with SKIE

Kotlin

```kotlin
class Shop {
    @DefaultArgumentInterop.Enabled
    2 usages
    fun orderItems(item: String,
                   quantity: Int = 1) {
        print("Ordered")
    }
}
```

Swift

```swift
Shop().orderItems(item: "1")
```

Language features we'll be covering

- Basics (Classes, properties, functions / Top-level properties and functions)
- Types (Basic types / Optional basic types / Collections)
- Functions (Default arguments)
- Object-oriented programming (Sealed classes and interfaces)
- Asynchronous programming (Suspend functions / Flows)
- Experimental language features (annotations)

Object-oriented programming

Inheritance / objects / interfaces

- You have to override all `abstract` functions, Xcode won't warn here
- `objects` and `companion` objects accessed via auxiliary objects
- `Interfaces` become `@protocols`

Object-oriented programming

Sealed interfaces/classes

Kotlin

```kotlin
sealed class SealedClass {
    object Object : SealedClass()

    class Simple(val param1: String) : SealedClass()

    data class Data(val param1: String, val param2: Boolean) : SealedClass()
}
```

Swift

```swift
func usingKotlinSealedClass(s: SealedClass) {
    switch s {
    case is SealedClass.Object:
        print("object")
    case is SealedClass.Simple:
        print("simple")
    case is SealedClass.Data:
        print("data")
    default:
        print("other")
    }
}
```

Sealed interfaces and classes with SKIE

- Exhaustive switching
- Type-safety
- Supports nested hierarchies

Sealed interfaces and classes with SKIE

Kotlin

```kotlin
sealed class SealedClass {

    1 super
    object Object : SealedClass()


    1 super
    class Simple(val param1: String) : SealedClass()


    1 super
    data class Data(val param1: String, val param2: Boolean) : SealedClass()
}
```

Swift

```swift
func example(s: SealedClass) {
    switch onEnum(of: s) {
        case .object: print("object")
        case .simple(let simple): print("simple \(simple.param1)")
        case .data(let data): print("data \(data.param1) \(data.param2)")
    }
}
```

Language features we'll be covering

- Basics (Classes, properties, functions / Top-level properties and functions)
- Types (Basic types / Optional basic types / Collections)
- Functions (Default arguments)
- Object-oriented programming (Sealed classes and interfaces)
- Asynchronous programming (Suspend functions / Flows)
- Experimental language features (annotations)

Nearly there!

Asynchronous programming

- `suspend` functions have callback / completion handler
- Can now also use Swift's `async/await`
- `Flows` also have a callback / completion handler, and the generic type is lost

Asynchronous programming

Suspend functions

Kotlin

```kotlin
suspend fun getThingSimple(succeed: Boolean): Thing {
    delay(100.milliseconds)
    if (succeed) {
        return Thing("Thing")
    } else {
        error("oh no!")
    }
}
```

Swift

```swift
@MainActor

1 usage
func suspendFunctionExample() {
    Task {
        do {
            let thing = try await ThingRepository().getThingSimple(succeed: true)
            print("Got result: \(thing).")
        }
        catch {
            print("Found error: \(error)")
        }
    }
}
```

Asynchronous programming

What happens here?

```
@MainActor

1 usage
func suspendFunctionWithCancellationExample() {
    Task {
        do {
            let thing = try await ThingRepository().getThingSimple(succeed: true)
            print("Got result: \(thing).")
        }
        catch {
            print("Found error: \(error)")
        }
    }.cancel()
}
```

Asynchronous programming

- No cancellation support
  - Important for handling async operations in UI lifecycle-aware manner
- Must use wrapper / library or plugin to do this
- Must be called on the main thread (Kotlin/Native restriction) - use `@MainActor`

KMP-NativeCoroutines

Setup: `shared/build.gradle.kts`

- Add plugins

```
plugins {
    id("com.google.devtools.ksp") version "1.9.10-1.0.13"
    id("com.rickclephas.kmp.nativecoroutines") version "1.0.0-ALPHA-18"
}
```

- Opt in to language setting

```
sourceSets {
    all {
        languageSettings.optIn("kotlin.experimental.ExperimentalObjCName")
    }
```

- Add dependency via Cocoapods / SPM in Xcode

KMP-NativeCoroutines

Annotate suspend function (in Kotlin)

```kotlin
@NativeCoroutines
suspend fun getThingAnnotated(succeed: Boolean): Thing {
    delay(100.milliseconds)
    if (succeed) {
        return Thing("Thing")
    } else {
        error("oh no!")
    }
}
```

KMP-NativeCoroutines

Wrap function call (in Swift)

```swift
@MainActor
func suspendFunctionKMPNativeCoroutinesExample() {
    Task {
        do {
            let result = try await asyncFunction(for: ThingRepository().getThingAnnotated(succeed: true))
            print("Got result: \(result)")
        } catch {
            print("Failed with error: \(error)")
        }
    }
}
```

SKIE and suspend functions

- Source compatible call-site with Kotlin*
- Full cancellation support
- Usable from any thread

SKIE and suspend functions

Setup: shared/build.gradle.kts

```
func example(){
    Task {
        let result = try await ThingRepository().getThing(succeed: true)
        print("Got result: \(result)")
    }
}
```

KMP-NativeCoroutines

Annotate flow function (in Kotlin)

```kotlin
@NativeCoroutines
fun getNumbersAnnotated(): Flow<Int> = flow {
    for (i in 1..10) {
        emit(i)
        delay(1.seconds)
    }
}
```

KMP-NativeCoroutines

Wrap function call (in Swift)

```swift
@MainActor
func flowKMPNativeCoroutinesExample() {
    Task {
        do {
            let sequence = asyncSequence(for: NumberFlowRepository().getNumbersAnnotated())
            for try await number in sequence {
                print("Got number: \(number)")
            }
        } catch {
            print("Failed with error: \(error)")
        }
    }
}
```

SKIE and Flows

- Bridged to Swift's `AsyncSequence`
- Full type safety
- No wrapping or extra code on Swift side
- `Flow`, `StateFlow`, `MutableStateFlow` support
- Supports cancellation
- Can be used from any thread

SKIE and Flows

```
func example(){
    Task {
        for await it in NumberFlowRepository().getNumbers() {
            print("Got number: \(it)")
        }
    }
}
```

KMP-NativeCoroutines or SKIE

Which is right for my project?

- KMP-NativeCoroutines
  - Generates needed wrappers
  - Requires annotation, function wrappers
  - Supports async/await, Combine, and RxSwift
  - Available for longer than SKIE, and may encounter fewer edge-cases
- SKIE
  - Augments the Objective-C API produced by the Kotlin compiler
  - Requires no additional work after setup
  - Supports async/await, while Combine, RxSwift require adapters
  - Offers other features to produce a Swift-friendly API from Kotlin

Language features we'll be covering

- Basics (Classes, properties, functions / Top-level properties and functions)
- Types (Basic types / Optional basic types / Collections)
- Functions (Default arguments)
- Object-oriented programming (Sealed classes and interfaces)
- Asynchronous programming (Suspend functions / Flows)
- Experimental language features (annotations)

Experimental language features - annotations

Opting in

```
sourceSets { this: NamedDomainObjectContainer<KotlinSourceSet>
    all { this: KotlinSourceSet
        languageSettings.optIn(annotationName: "kotlin.experimental.ExperimentalObjCName")
        languageSettings.optIn(annotationName: "kotlin.experimental.ExperimentalObjCRefinement")
    }
```

Experimental language features - annotations

@ObjCName

Kotlin

```kotlin
@ObjCName(swiftName = "MySwiftArray")
1 usage
class MyKotlinArray {
    @ObjCName(name: "index")
    1 usage
    fun indexOf(@ObjCName(name: "of") element: String): Int = 1
}
```

Swift

```swift
let array = MySwiftArray()
let index = array.index(of: "element")
```

Experimental language features - annotations

@HiddenFromObjC

Kotlin

```kotlin
@HiddenFromObjC
fun myKotlinOnlyFunction(){
    println("Only Kotlin!")
}
```

Swift

```swift
func hiddenFromObjCExample(){
    myKotlinOnlyFunction()
}
```

Experimental language features - annotations

@ShouldRefineInSwift

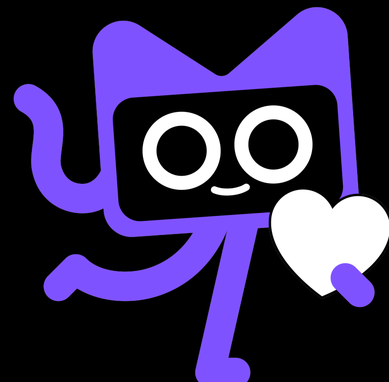
Kotlin

```kotlin
interface Person {
    @ShouldRefineInSwift

    1 implementation
    val namePair: Pair<String, String>
}


1 usage | 1 super
class RealPerson: Person {

    1 super
    override val namePair : Pair<String, String> = "Rick" to "Clephas"
}
```

Swift

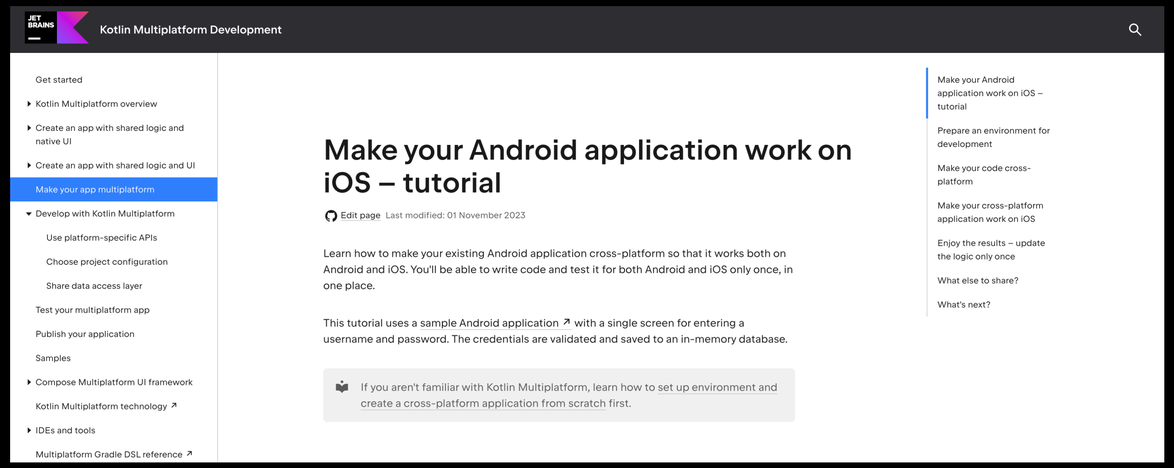
```swift
extension Person {
    1 usage
    var name: (firstName: String, lastName: String) {
        let namePair = __namePair
        return (namePair.first! as String, namePair.second! as String)
    }
}


1 usage
func shouldRefineInSwiftExample(){
    let authorNames = RealPerson().name
    print("Author is: \(authorNames.firstName) \(authorNames.lastName)")
}
```

Helpful Resources



Make your Android app work on iOS - tutorial

Make an **existing** Android app cross-platform so it works on Android and iOS.



jb.gg/kmp-existing-on-ios

How to Migrate an Android Project to Kotlin Multiplatform (KMP)

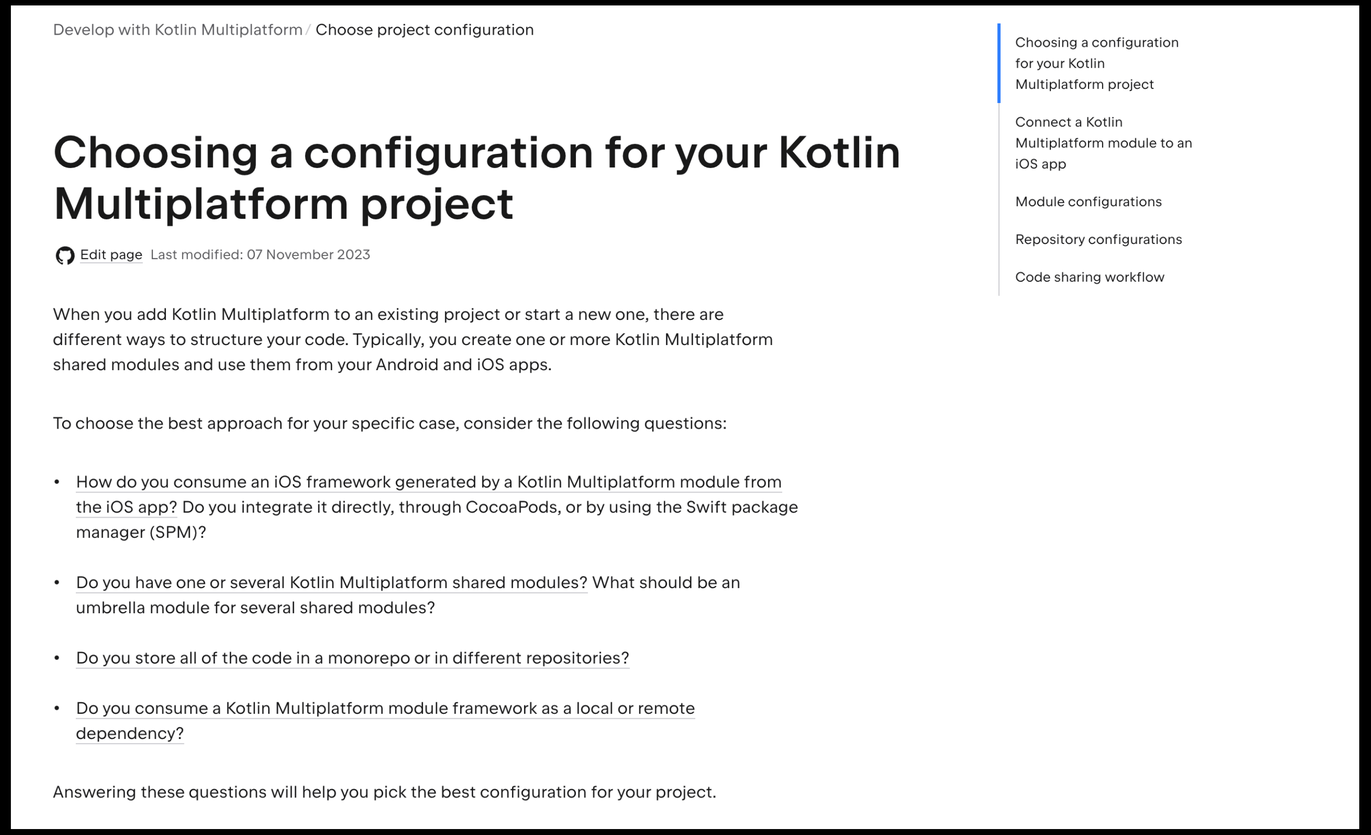
Philipp Lackner

www.youtube.com/watch?v=vb-Pt8SdfEE

Choose your project configuration

Helps you decide:

- How to consume an iOS framework from your KMP module in your iOS app
- One or several KMP modules? Plus: what's an umbrella ☂️ module?
- Monorepo or different repositories?
- Local or remote dependency?



Develop with Kotlin Multiplatform / Choose project configuration

Choosing a configuration for your Kotlin Multiplatform project

Edit page Last modified: 07 November 2023

When you add Kotlin Multiplatform to an existing project or start a new one, there are different ways to structure your code. Typically, you create one or more Kotlin Multiplatform shared modules and use them from your Android and iOS apps.

To choose the best approach for your specific case, consider the following questions:

- How do you consume an iOS framework generated by a Kotlin Multiplatform module from the iOS app? Do you integrate it directly, through CocoaPods, or by using the Swift package manager (SPM)?
- Do you have one or several Kotlin Multiplatform shared modules? What should be an umbrella module for several shared modules?
- Do you store all of the code in a monorepo or in different repositories?
- Do you consume a Kotlin Multiplatform module framework as a local or remote dependency?

Answering these questions will help you pick the best configuration for your project.

Choosing a configuration for your Kotlin Multiplatform project
Connect a Kotlin Multiplatform module to an iOS app
Module configurations
Repository configurations
Code sharing workflow

jb.gg/kmp-project-config

To Watch

ATOM / Talking Kotlin feat. iOS Engineers

www.youtube.com/@Kotlin

Introduce your team guide

Introduce cross-platform development to your team

Edit page Last modified: 27 November 2023

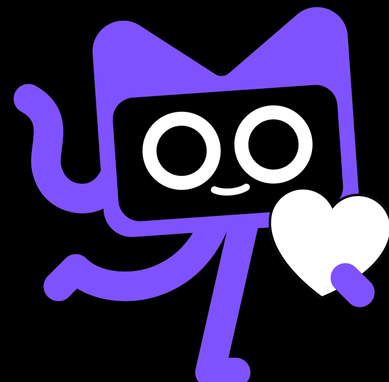
These recommendations will help you introduce your team to Kotlin Multiplatform:

- Start with empathy
- Explain how Kotlin Multiplatform works
- Show the value using case studies
- Offer a proof by creating a sample project yourself
- Prepare for questions from your team

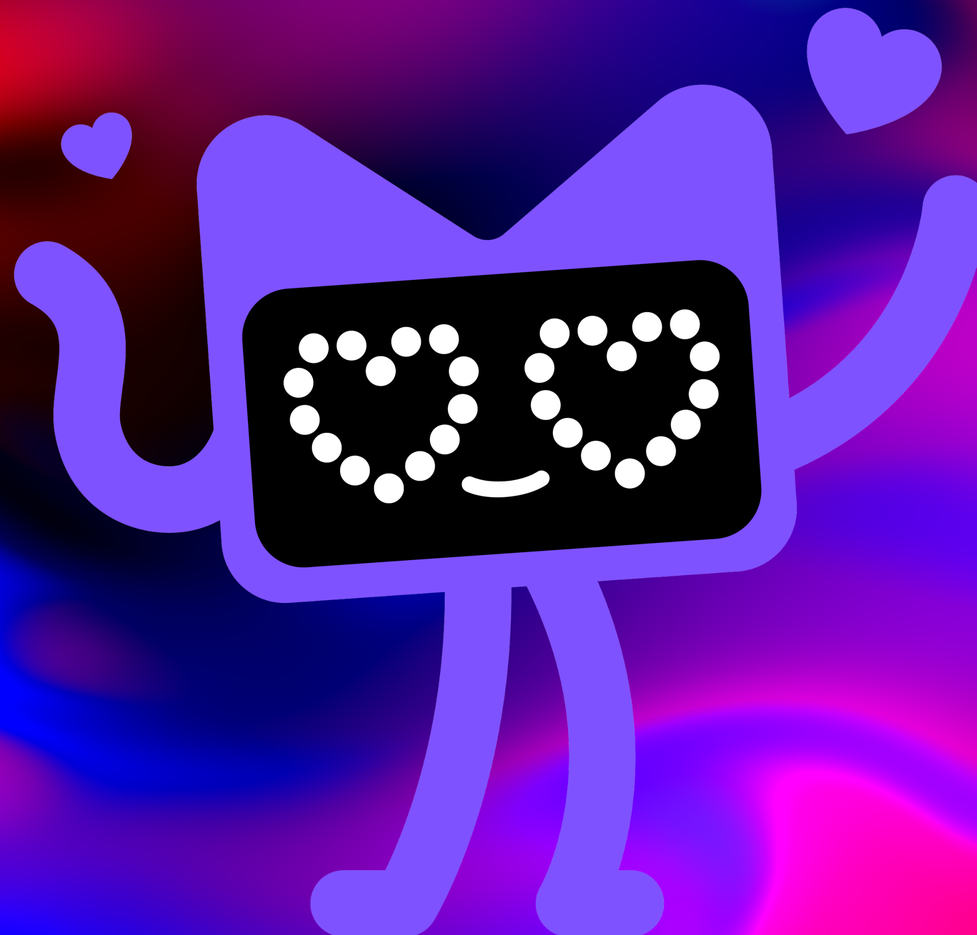
kotl.in/kmp-intro-team

We did it!

Questions?



Thank you 💗



👩🏻‍💻 Pamela Hill @pamelaahill
👨🏼‍💻 Tadeas Kriz @TadeasKriz@mastodon.social