

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

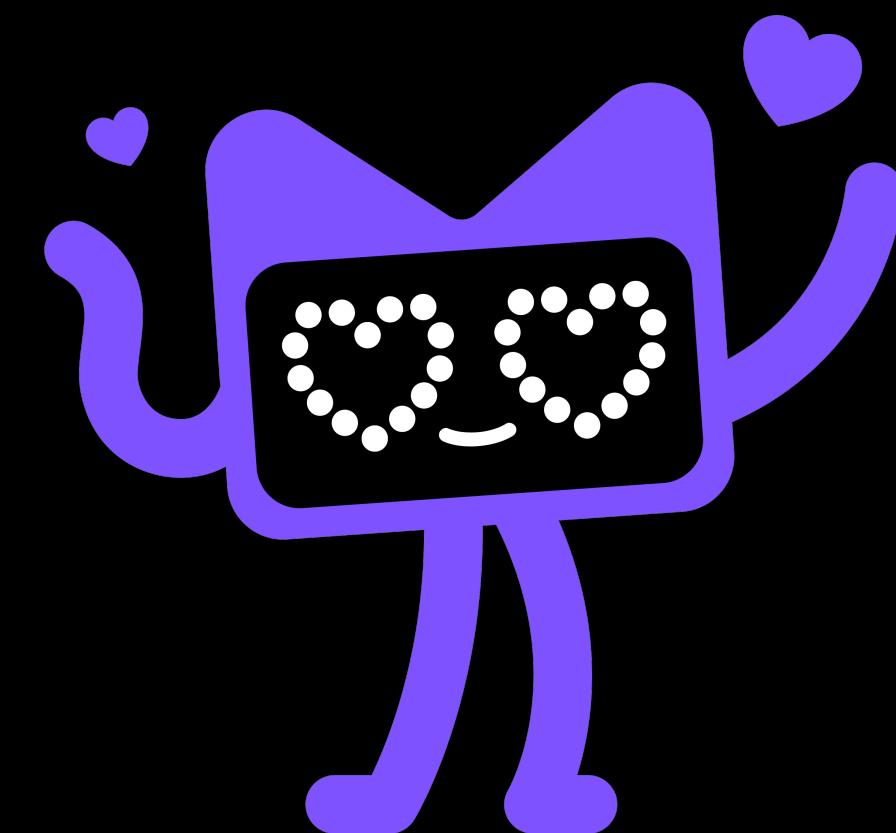
November 30, 2023 – 4:00 PM UTC

Access to Resources

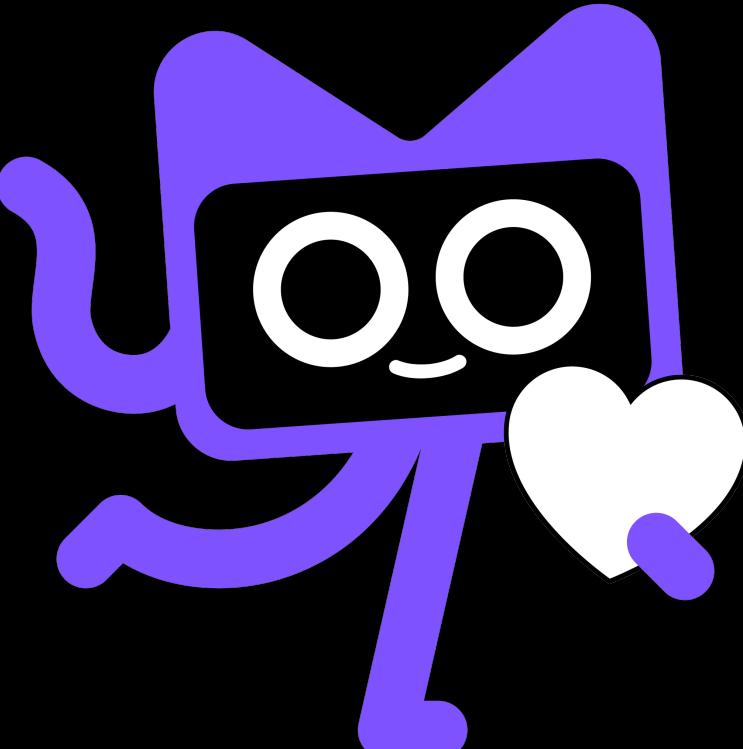
- This webinar is being recorded
- Slides can be found here: kotlinlang.org/ios-dev-kmp

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: Conditional unwrapping (if let / guard let) 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

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

Kotlin

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

Variables And Constants

Swift

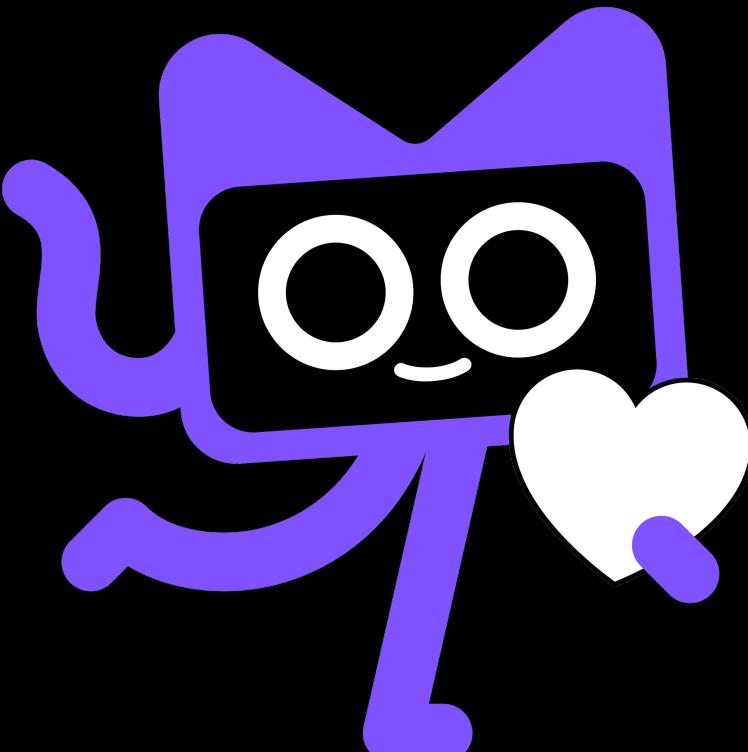
```
var myVariable = 42  
myVariable = 50  
let myConstant = 42
```

Kotlin

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

Explicit Types

Current State of Kotlin/Swift Interop



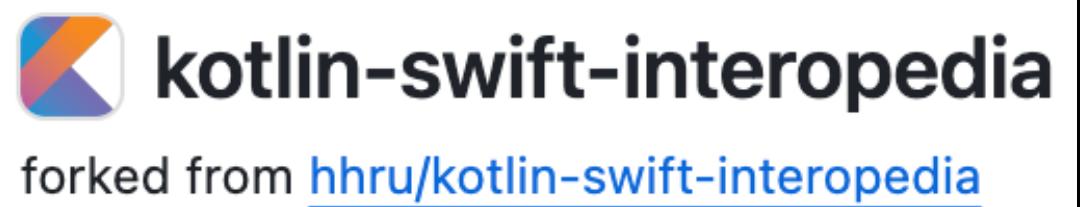
The Kotlin team is working on
direct Swift interoperability.

Current state

- Kotlin Multiplatform supports direct interoperability with Objective-C
- Swift interoperability **via Objective-C bridge**
- This means:
 - Most features work out of the box
 - Some features need a slight modification
 - Some features need a workaround or a plugin

Kotlin/Swift Interopedia

kotlin.in/interopedia



Overview	
Top-level functions	You can access a top-level function via the wrapper class: <code>TopLevelFunctionKt.topLevelFunction()</code> .
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 <code>NSError</code> and must be handled.
Public API	Public classes, functions, and properties are visible from Swift. Marking classes, functions, and properties <code>internal</code> 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 (<code>@property</code>) 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.

A screenshot of a mobile application titled 'Kotlin/Swift Interop Playground'. The top bar shows the time as 2:03 and battery level. The main screen has a sidebar with navigation links: OVERVIEW, Classes and functions, Top-level functions, Types, Collections, Exceptions, Public API, and KDoc Comments. The 'OVERVIEW' link is currently selected. Below the sidebar, there's a section titled 'FUNCTIONS AND PROPERTIES' with links to Member functions, Constructors, and Read-only member properties.

What is SKIE?

- Kotlin compiler 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

SKIE setup

shared/build.gradle.kts

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

What is KMP-NativeCoroutines?

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

KMP-NativeCoroutines setup

shared/build.gradle.kts

- Add plugins

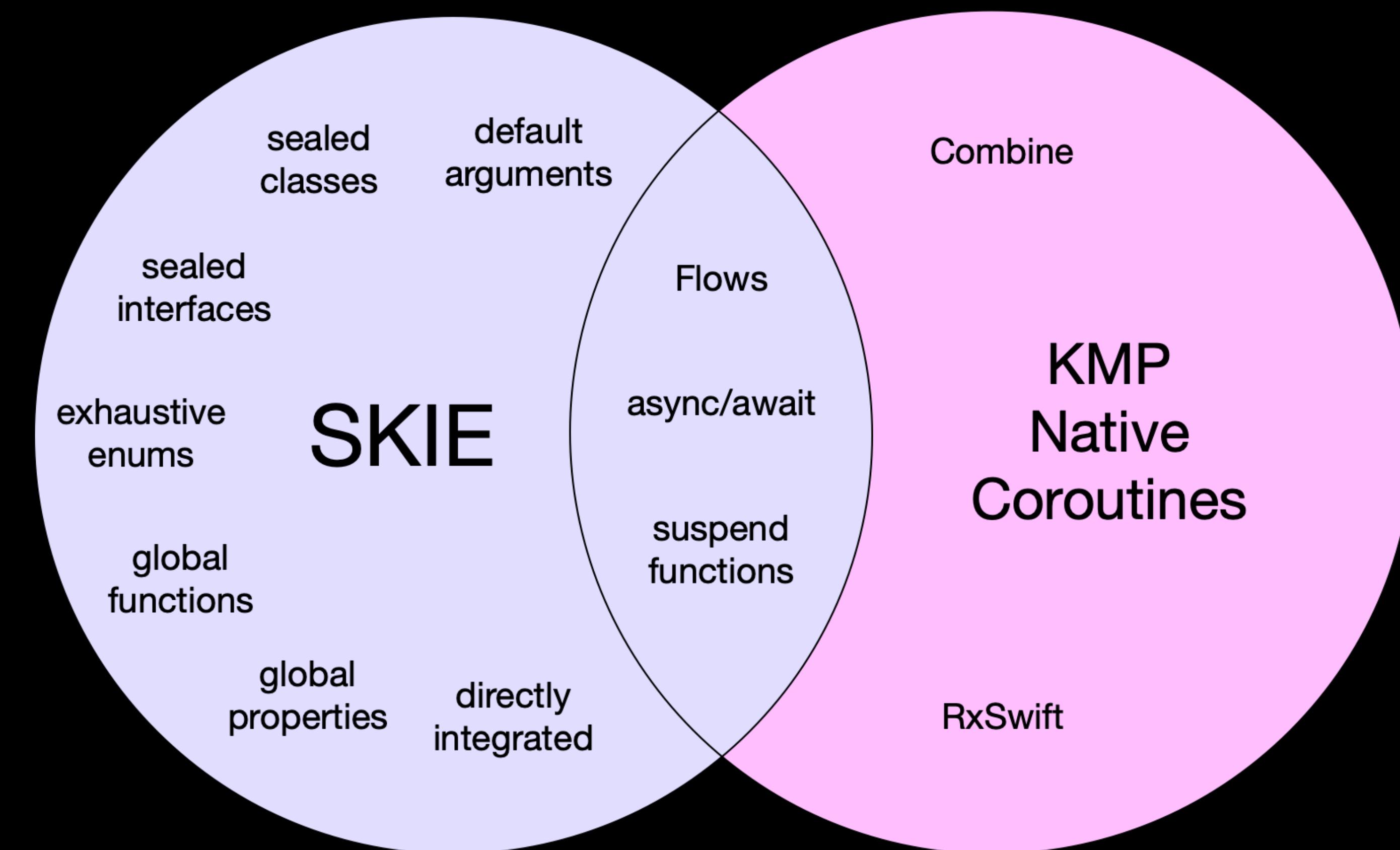
```
plugins {  
    id("com.google.devtools.ksp") version "1.9.10-1.0.13"  
    id("com.rickclephas.kmp.nativecoroutines") version "1.0.0-ALPHA-18"  
}
```

- Opt in to language setting

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

- Add dependency via Cocoapods / SPM in Xcode

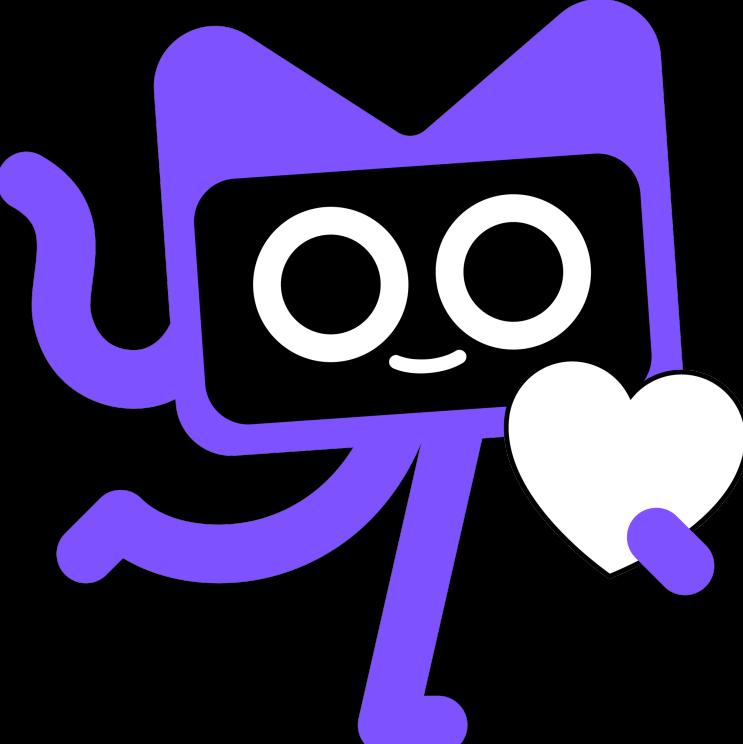
Features





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)

Most features just work

Basics

Classes, properties, functions

Kotlin

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

Swift

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

Some features need
a slight modification

Basics

Top-level properties, functions

Kotlin

In Constants.kt

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

Swift

In Constants.kt

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

In Messages.kt

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

```
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 be mapped 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

Basics

Optional basic types

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

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 or a plugin

Functions

Default arguments

Kotlin

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

Swift

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

Default arguments with SKIE

Generated overloads

Kotlin

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

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

- Inherit and implement Kotlin declarations in Swift
- Interfaces become @protocols
- objects and companion objects accessed via auxiliary objects
- Xcode won't warn when abstract override is missing

Object-oriented programming

Sealed interfaces/classes

Kotlin

```
sealed class SealedClass {  
    object Object : SealedClass()  
  
    class Simple(val param1: String) : SealedClass()  
  
    data class Data(val param1: String, val param2: Boolean) : SealedClass()  
}
```

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 and type-safe

Kotlin

```
sealed class SealedClass {  
    object Object : SealedClass()  
  
    class Simple(val param1: String) : SealedClass()  
  
    data class Data(val param1: String, val param2: Boolean) : SealedClass()  
}
```

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

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

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

- Not cancellable
 - Important for handling async operations in UI lifecycle-aware manner
- Must be called on the main thread - use `@MainActor`



SKIE and KMP-NativeCoroutines
got your back!

KMP-NativeCoroutines

Annotate suspend function (in 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)

```
@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

Cancellation and background threads

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

KMP-NativeCoroutines

Annotate flow function (in Kotlin)

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

KMP-NativeCoroutines

Wrap function call (in 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

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

Swift

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

Experimental language features - annotations

@HiddenFromObjC

Kotlin

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

Swift

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

Experimental language features - annotations

@ShouldRefineInSwift

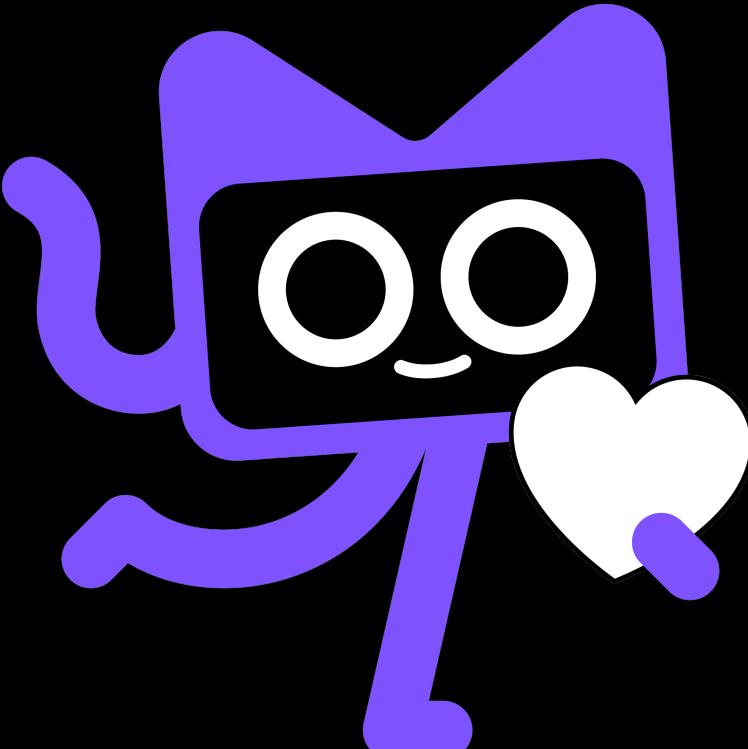
Kotlin

```
interface Person {  
    @ShouldRefineInSwift  
    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

```
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.

The screenshot shows a dark-themed web page for 'Kotlin Multiplatform Development'. At the top left is the Jet Brains logo and the text 'Kotlin Multiplatform Development'. A search icon is at the top right. The main content area has a heading 'Make your Android application work on iOS – tutorial'. Below it is a paragraph about the tutorial's purpose: 'Learn how to make your existing Android application cross-platform so that it works both on Android and iOS. You'll be able to write code and test it for both Android and iOS only once, in one place.' A note below states: 'This tutorial uses a sample Android application ↗ with a single screen for entering a username and password. The credentials are validated and saved to an in-memory database.' To the right of the main content is a sidebar with several links: 'Make your Android application work on iOS – tutorial', 'Prepare an environment for development', 'Make your code cross-platform', 'Make your cross-platform application work on iOS', 'Enjoy the results – update the logic only once', 'What else to share?', and 'What's next?'. On the far left of the main content area, there is a vertical navigation bar with sections like 'Get started', 'Kotlin Multiplatform overview', 'Create an app with shared logic and native UI', 'Create an app with shared logic and UI', 'Make your app multiplatform' (which is highlighted in blue), 'Develop with Kotlin Multiplatform', 'Use platform-specific APIs', 'Choose project configuration', 'Share data access layer', 'Test your multiplatform app', 'Publish your application', 'Samples', 'Compose Multiplatform UI framework', 'Kotlin Multiplatform technology ↗', 'IDEs and tools', and 'Multiplatform Gradle DSL reference ↗'.

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

Philipp Lackner



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

To Watch

ATOM / Talking Kotlin feat. iOS Engineers



Introduce your team guide

Multiplatform development / Introduce cross-platform development to your team

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

Introduce cross-platform development to your team

Start with empathy

Explain how it works

Show the value

Offer proof

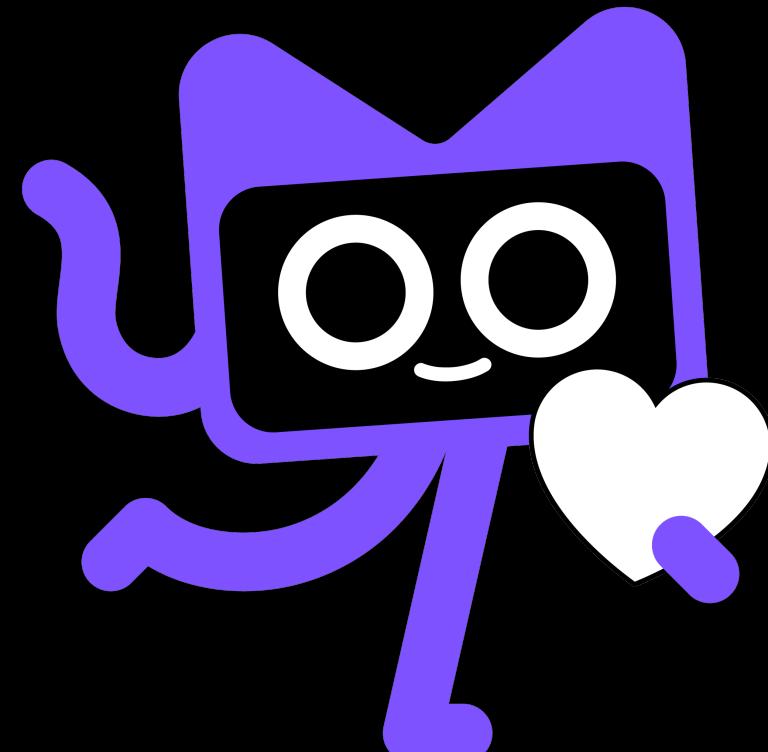
Prepare for questions

Be supportive



We did it!

Questions?



Thank you ❤

👤 Pamela Hill @pamelahill

👤 Tadeas Kriz @TadeasKriz@mastodon.social

