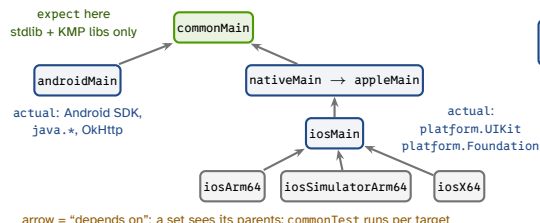
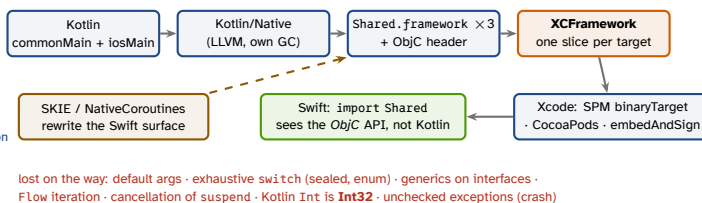


In one line: KMP compiles **one Kotlin codebase** to each platform's native artifact — JVM bytecode for Android, a **Kotlin/Native** (LLVM) binary shipped as an **Objective-C framework** for iOS — so you share **logic** (models, networking, persistence, state) and keep **native UI** unless you opt into Compose Multiplatform. Swift sees Kotlin **through an ObjC header**: that header is where every interop trap lives.

Source sets (default hierarchy, Kotlin 1.9.20+)



How Swift gets it



What KMP is — and is not

- Targets: `androidTarget()`, `iosArm64()` (device), `iosSimulatorArm64()`, `iosX64()` (Intel sim), `jvm()`, JS, Wasm. **Stable** since Kotlin 1.9.20 (Nov 2023); Google backs it for shared logic.
- Share bottom-up: models, DTOs, networking, validation, repositories, state machines, DB. Keep platform: UI, permissions, push token, Keychain, pickers.
- `commonMain` cannot touch `java.*` (use `kotlinx-datetime` etc.). A library must **publish klibs for every target** — Maven Central $\neq$ KMP.
- expect/actual: functions, classes, objects, properties, typealias. Prefer a common interface + injected impl (testable, can be written in Swift); expect for thin platform primitives.
- vs Flutter/RN: no own renderer, no JS bridge — compiled logic under native UI.

Example — shared API, Swift call site

```
// commonMain
expect fun platformName(): String
sealed interface Ui
data class Loaded(val items: List<Device>) : Ui
class DevicesSdk(private val api: Api) {
    @Throws(ApiError::class, CancellationException::class)
    suspend fun refresh(): List<Device> = api.devices()
    val state: StateFlow<Ui> = /* ... */
}
// iosMain
actual fun platformName() = UIDevice.currentDevice.systemName()

let list = try await sdk.refresh() // cancel won't reach Kotlin
switch sdk.state.value {          // Any? (generic erased)
case let s as Loaded: show(s.items) // a cast, not an enum case
default: break                    // no exhaustiveness
```

Compose Multiplatform (briefly)

- Shares the *UI* too: `@Composable` in `commonMain`, wrapped on iOS by `ComposeUIViewController { App() }` and hosted from SwiftUI via `UIViewControllerRepresentable`. Skia draws the pixels — no UIKit controls. iOS stable since CMP 1.8 (2025).
- Worth it for form/data-heavy UI, a Kotlin-heavy team, parity over idiom; otherwise keep SwiftUI and share logic only.

What Swift sees

Kotlin	Swift (via the ObjC header)
<code>class Greeting</code>	<code>Greeting (ObjC: SharedGreeting);</code> a clash gets a <code>_</code> suffix
<code>top-level fun f() in Util.kt</code>	<code>UtilKt.f()</code>
<code>object X · companion</code>	<code>X.shared · X.companion</code>
<code>suspend fun</code>	<code>completion handler → async throws</code> ; no cancellation inward
<code>Flow / StateFlow</code>	<code>opaque protocol: implement a FlowCollector; .value is Any?</code>
<code>sealed class/interface</code>	<code>base class + subclasses</code> ; cast with <code>as</code> , need default
<code>enum class</code>	<code>class with static instances</code> , not a Swift enum
<code>generic class Box<T></code>	<code>ObjC lightweight generic, T: AnyObject</code>
<code>generic interface, generic fun</code>	erased to <code>Any?</code>
<code>Int · Long · Int?</code>	<code>Int32 · Int64 · KotlinInt?</code> (boxed)
<code>List<T> · Map</code>	<code>[T] · [K: V] (NSArray / NSDictionary)</code>
<code>Unit callback</code>	<code>KotlinUnit</code>
<code>data class</code>	<code>NSObject subclass: == via isEqual, copy → doCopy(...)</code> (all args); no value semantics / Codable
<code>default arguments</code>	dropped — Swift passes every argument
<code>exception, no @Throws</code>	crashes the app
<code>@ObjCName · @HiddenFromObjC</code>	<code>rename · hide from the header</code>

Tools that fix the surface

- SKIE** (Touchlab, Gradle plugin): sealed → Swift enum via `onEnum(of:)`, Kotlin enums → Swift enums, `Flow` → `AsyncSequence`, `suspend` → `async with cancellation`.
- KMP-NativeCoroutines**: `@NativeCoroutines` → Swift `asyncFunction(for:)`, `asyncSequence(for:)`, `Combine` publishers.
- Still design a **Swift-shaped facade**: concrete types, `@Throws`, no defaults, map to Swift structs at the edge. Swift export (no ObjC) is experimental.

Interview traps

- ! “KMP shares the UI” — core KMP shares logic; UI is Compose Multiplatform.
- ! `.framework` = one slice: device-only fails in the simulator — ship an **XCFramework**.
- ! Kotlin `Int` arrives as `Int32`; nullable primitives as `KotlinInt?`.
- ! Kotlin cannot import Swift: Swift implements a Kotlin interface and passes it in.

Remember

Share logic, keep UI · common sees nothing platform · Swift sees ObjC · wrap suspend, Flow, sealed.

Likely questions

- expect/actual vs interface? — compile-time per platform vs injected (fakeable).
- Consume a Flow in SwiftUI? — SKIE / NativeCoroutines `AsyncSequence`, or a collector + cancel handle.
- Why is my switch not exhaustive? — sealed bridges as subclasses.