Deep Dive Into Kotlin Multiplatform

Advanced Techniques for Seamless Code Sharing

JETBRAINS

About Your Instructor - Pamela

Kotlin Developer Advocate at **JetBrains**

Living in **Cape Town**, South Africa

About Your Mentor - Kostya

Software Developer in Compose Multiplatform at **JetBrains**

Living in **Leiden**, The Netherlands

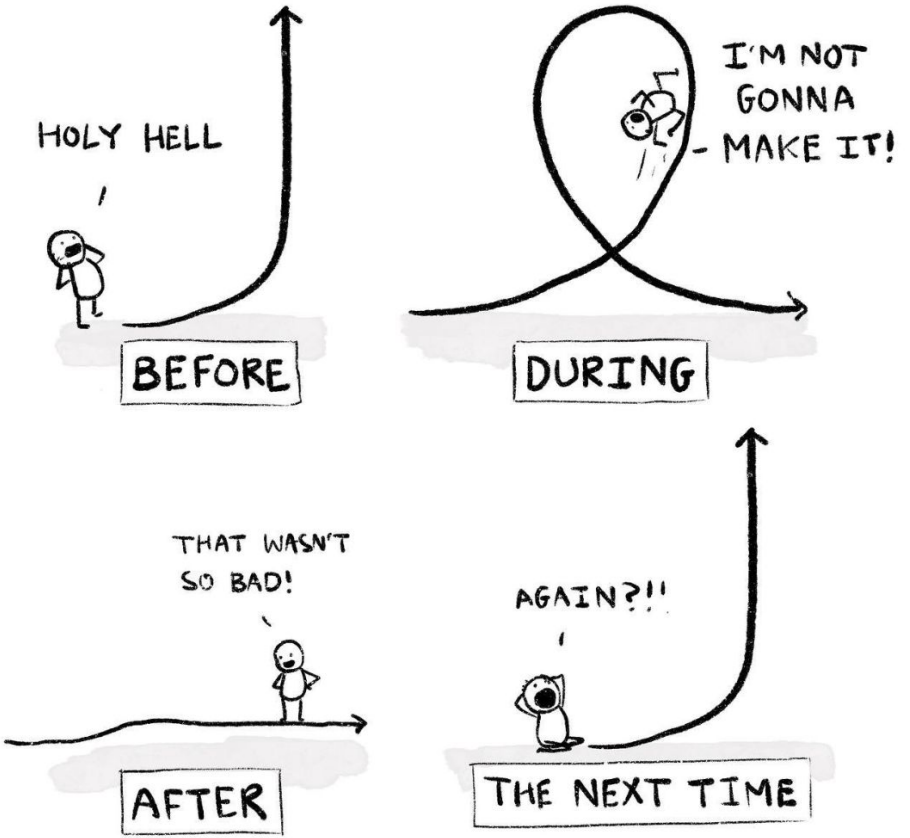
Morning Agenda

- Before we start: Setup verification
- 9:00 Welcome and speaker introduction
- 9:01 A Gentle(r) introduction
- 9:30 Native integrations + practical 1
- 10:30 Coffee break
- 11:00 Additional setup and independent lookaround
- 11:30 Advanced Kotlin Multiplatform for iOS Targets + practical 2
- 12:30 Lunch

Afternoon Agenda

- 13:30 Code quality
- 14:30 App quality
- 15:00 Coffee break
- 15:30 App quality + CI/CD notes
- 16:00 AMA with Kostya
- 16:30 Discussion time, photo, goodbye

HOW LEARNING CURVES FEEL...



@STEINBERGDRAWSCARTOONS

A Gentle(r) Introduction

What we'll be looking at:

- Introducing the first example project.
- Explaining project structure and architecture.
- Highlighting a few key libraries used in the project.
- Afterwards, there will be an opportunity to have a look around in the code.

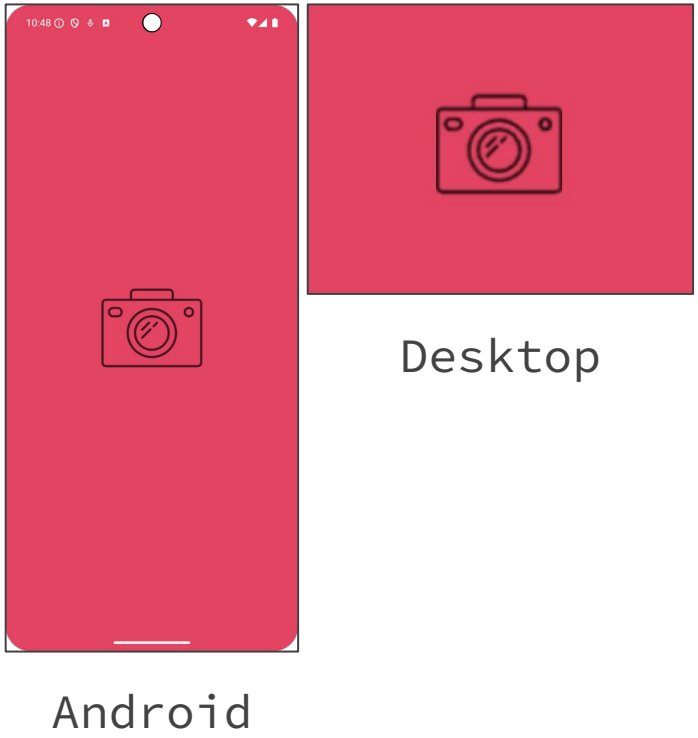
Introducing the example project

Camera app

- KMP/CMP app.
- Runs on Android / iOS / Desktop(JVM).
- Functionality:
  1. Allows you to get an image:
     a. (Android/iOS Device) Take a photo using your native camera, OR
     b. (iOS Sim/Desktop) Pick an image with a file picker.
  2. Apply some image filters to the image file.
  3. Create a local notification with the image file.

Step -1: Show the splash screen

- Done natively.
- No animations - simple image with a background.
- Illustrates how to do native splash screens (“Native integrations” section).

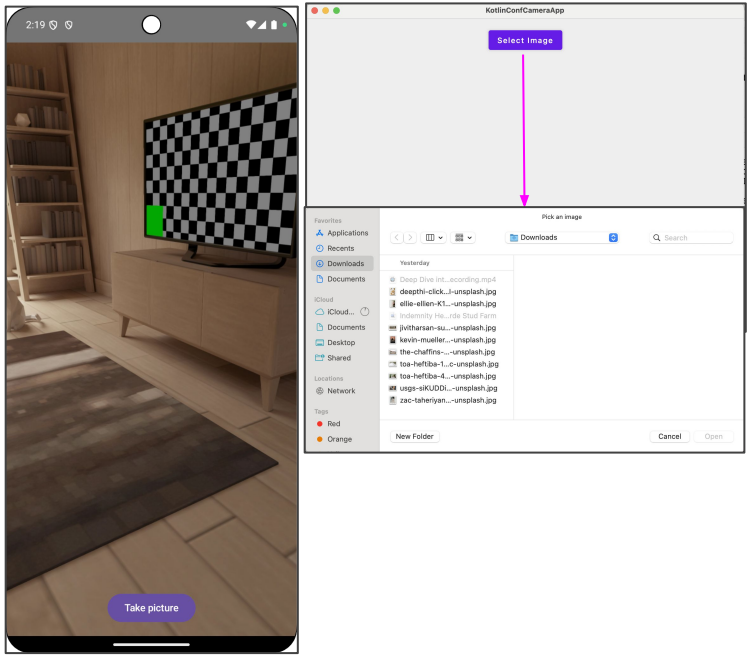


Step 0: Permissions screen

- Certain operations on Android/iOS need permissions.
- For example: camera, notifications, gallery.
- Android/iOS with CMP and library.
- Desktop doesn’t need permissions.

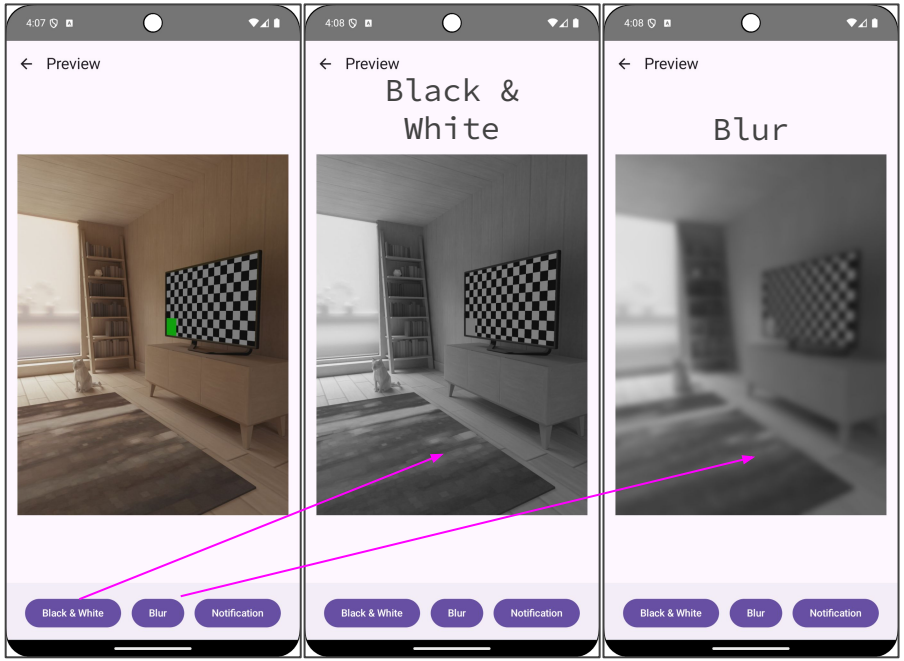
Step 1: Camera screen

- Native cameras on Android/iOS device - live preview on back-facing camera.
- “Select image” button that opens file picker in user directory on Desktop/iOS sim.



Step 3: Pictures screen

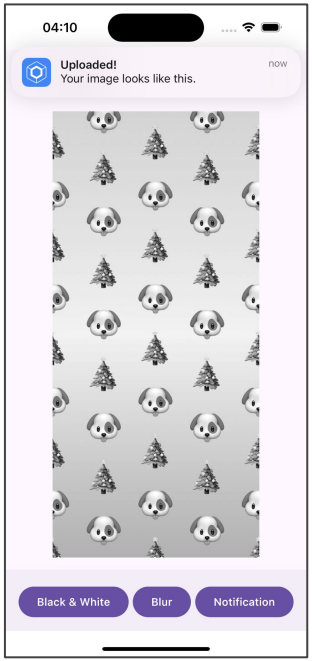
- Preview of image taken/selected in previous screen.
- Two filters: black & white, blur.
- Illustrates handling files natively (“Native integrations” section).



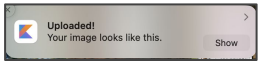
Step 3: Notifications popup

- Notifications done natively using a library for Android/iOS/Desktop.
- Illustrates doing native notifications (“Native integrations” section).

Android



iOS

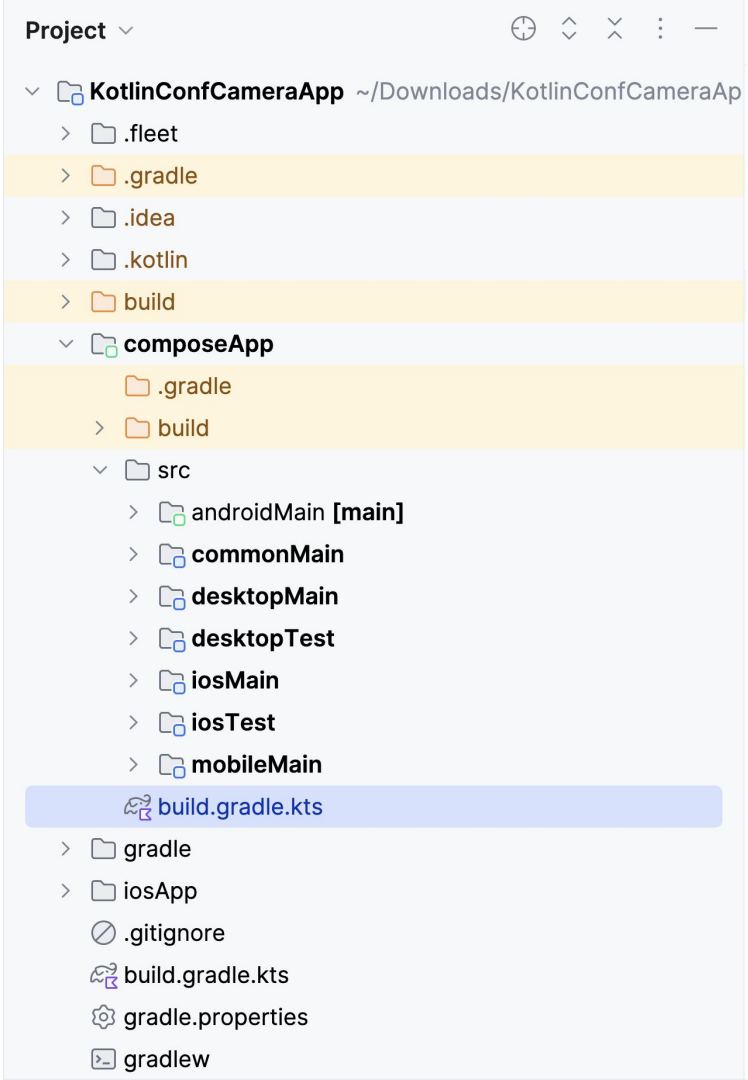


Desktop

Explaining project structure and architecture

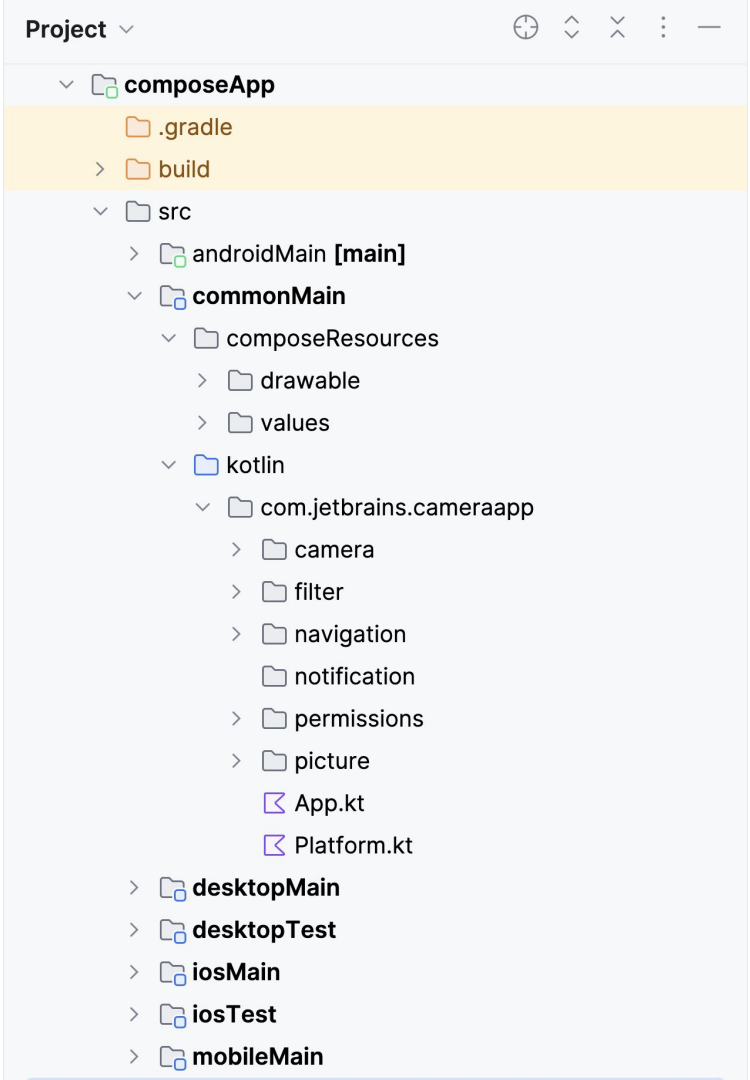
Project structure [top-level]

- Standard “web wizard” (kmp.jetbrains.com) structure.
- composeApp - module with shared & target-specific source sets.
  - build.gradle.kts - for Gradle project setup.
- gradle - directory with Gradle wrapper, version catalog etc.
- iosApp - directory with iOS application.



Project structure [commonMain]

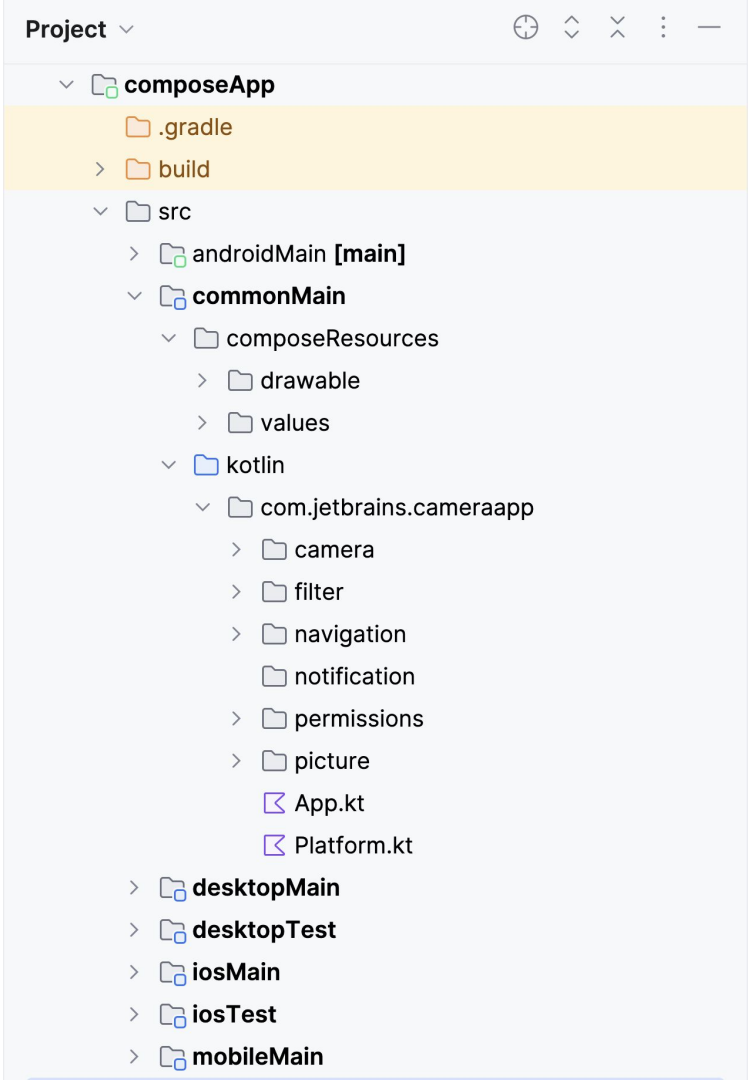
- Shared code and resources.
- We tried to put as much code here as possible.
- Packaged-by-feature rather than packaged-by-layer or type.



Project structure [other source sets]

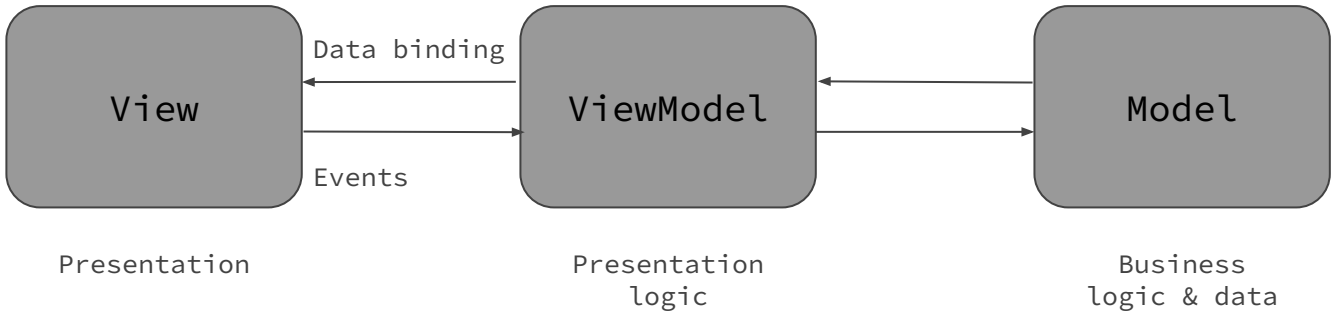
- androidMain - Android application AND Android-specific code*.
- desktopMain - Desktop application AND Desktop-specific code*.
- iosMain - iOS-specific code*.
- mobileMain - intermediate source set for Android/iOS-specific code* (related to permissions). Why?
  - Our library doesn't support Desktop
  - Desktop handles permissions differently

* Think actuals etc.



Pattern: MVVM

- Model-View-ViewModel.
- Exposes state to the UI, encapsulates business logic.
- On Android, `ViewModel` survives configuration changes.
- Other options are MVI / Clean architecture etc.



Extra resources for architectural patterns

- MVI:
  - Philipp Lackner (video) - 2023.
  - Arkadii Ivanov (library) - MVIKotlin.
- Clean:
  - Philipp Lackner (video) - 2022 | Philipp Lackner (video) - 2024.

Highlighting a few key libraries used in the project

Libraries used in the project [JetBrains]

- Kotlin Multiplatform (plugin).
- Compose Multiplatform (plugin).
- Navigation Compose + Kotlinx-serialization: Implementing navigation - docs.
  - Now in Beta.
  - BackHandler / Predictive Back / Deeplinks.
  - Will it become the de facto standard for navigation?
  - Alternatives: Voyager | Decompose (also has navigation) | Appyx | Circuit | PreCompose etc.
- Lifecycle: Handles lifecycle for components in platform-dependent way - docs (includes platform mappings).
- ViewModel for Compose: Shared ViewModels - docs.
  - Alternatives: KMP-ObservableViewModel | Voyager (also has VM) | Moko MVVM (also has VM) etc.

Libraries used in the project [Multiplatform Community]

- Coil - fancy async image loader
  - Used Multiplatform version (v3+).
  - Disabled disk/memory cache as we're working with latest version on disk (after filters).
  - Alternatives: Kamel | compose-imageloader etc.
- Moko Permissions - permissions for Android/iOS (need to ask before using)
  - You need to manually add text to iOS for the dialog.
    - "Privacy - Camera Usage Description" in the plist file.

Libraries used in the project

- Koin
  - Simple, popular.
  - Coding challenge - convert to Koin Annotations 2.0 for compile-time safety.
  - Alternatives: Kodein | kotlin-inject (with or without anvil) | Metro | Do-it-yourself.
- KMPNotifier
  - Local / push notifications. For our purposes - just local.
  - You do need to do some Firebase setup.
    - iOS: plist download, Firebase SPM packages addition.
    - Android: google-services.json.
  - Desktop: run with runDistributable.

Libraries used in the project [Android-specific]

- Splash screen:
  - My article (static + animated) for Android/iOS.
- Camera2 (CameraX)
  - Works ok on simulator.
- Androidx.graphics
  - Used for Black & White Filter.
- RenderScript
  - Used for Blur.

Libraries used in the project [iOS-specific]

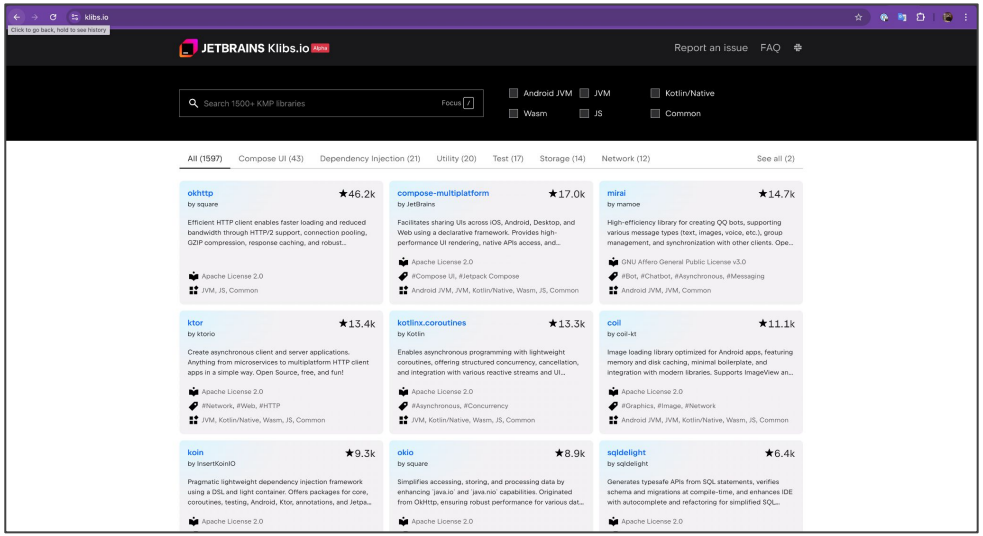
- AVFoundation - Camera - automatic bindings available.
- SwiftUI + UIKit - CMP <= UIKit <= SwiftUI app.
- CoreGraphics - for filters.
- FirebaseMessaging - local notifications (crash without the plist file).
  - Configuration instructions.

Libraries used in the project [Desktop-specific]

- FileKit - Multiplatform file operations, pickers, dialogs.
  - We used the picker for Desktop/iOS Sim, but easy to extend to other platforms.
  - Better native look than Swing components.

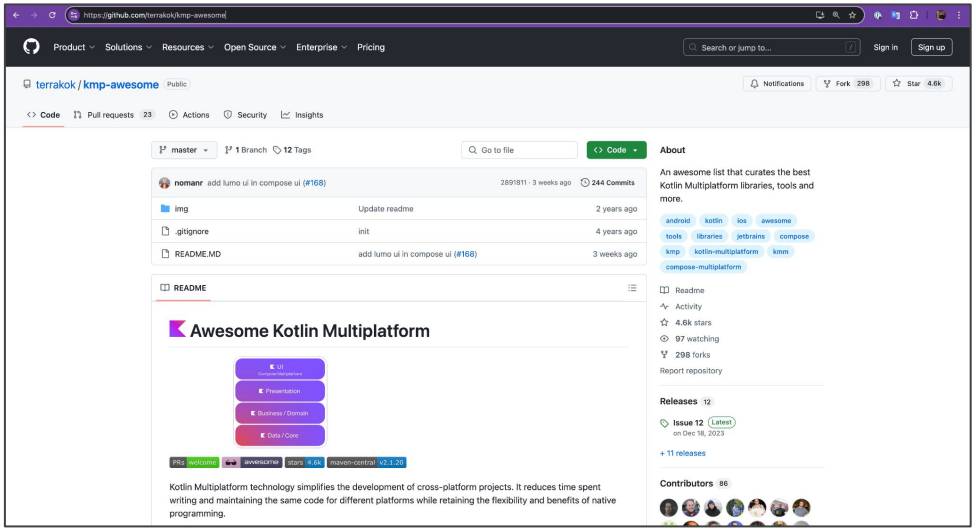
How to get libraries [klibs.io]

- Official from JetBrains.
- Only GitHub libraries thus far.
- Automatically added (see FAQ).



How to get libraries [kmp-awesome]:

- GitHub repo by Kostya and community.
- Submit PR to add your library.
- Can include libs on other platforms than GitHub.



Native Integrations

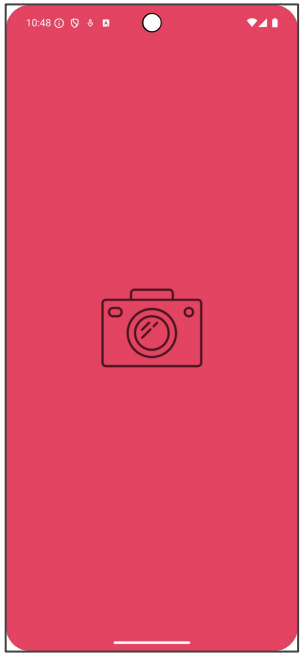
What we'll be looking at:

- Adding splash screens.
- Working with files (updating underlying image file with filter).
- Sending a local notification.

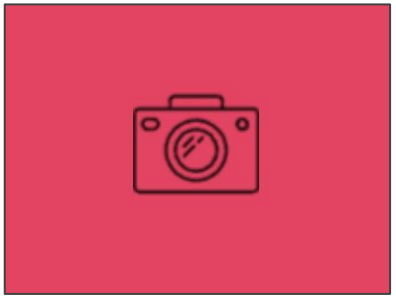
Adding Splash Screens

Splash screens

- Splash screens are there for a quick flash of branding, **not** for a lot of loading data.
- Troubleshooting:
  - Android / iOS: You may need to kill the app (swipe up) / even reinstall.
  - Desktop: make sure you are running with **runDistributable**.



Android

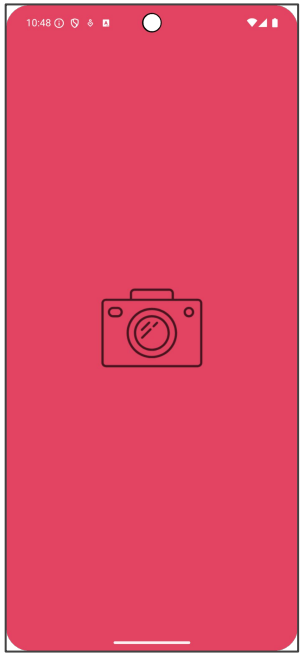


Desktop

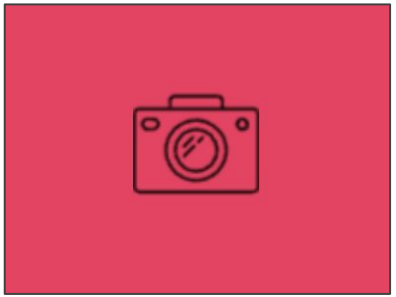
Splash screens

- Splash screens are there for a quick flash of branding, **not** for a lot of loading data.
- Troubleshooting:
  - Android / iOS: You may need to kill the app (swipe up) / even reinstall.
  - Desktop: make sure you are running with **runDistributable**. So:

```
./gradlew composeApp:runDistributable
```



Android



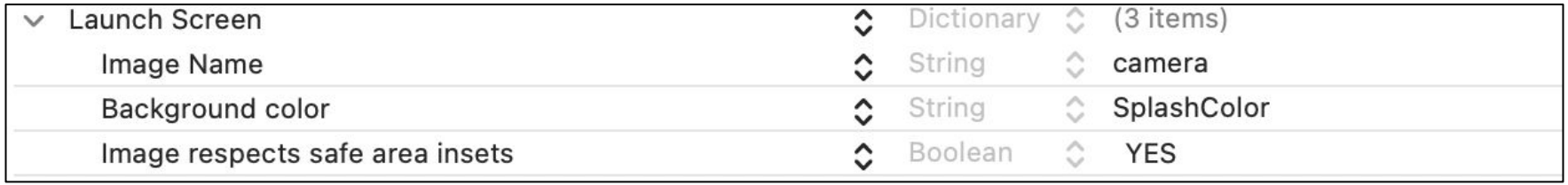
Desktop

Steps for Android [static]

1. Add the splash screen library
2. Import the .svg file as a vector asset
3. Make the vector drawable fit according to the Google guidelines
4. Create a splash screen theme
5. Apply the splash screen theme to the application and first Activity
6. Install the splash screen
7. Run & profit

Steps for iOS [static]

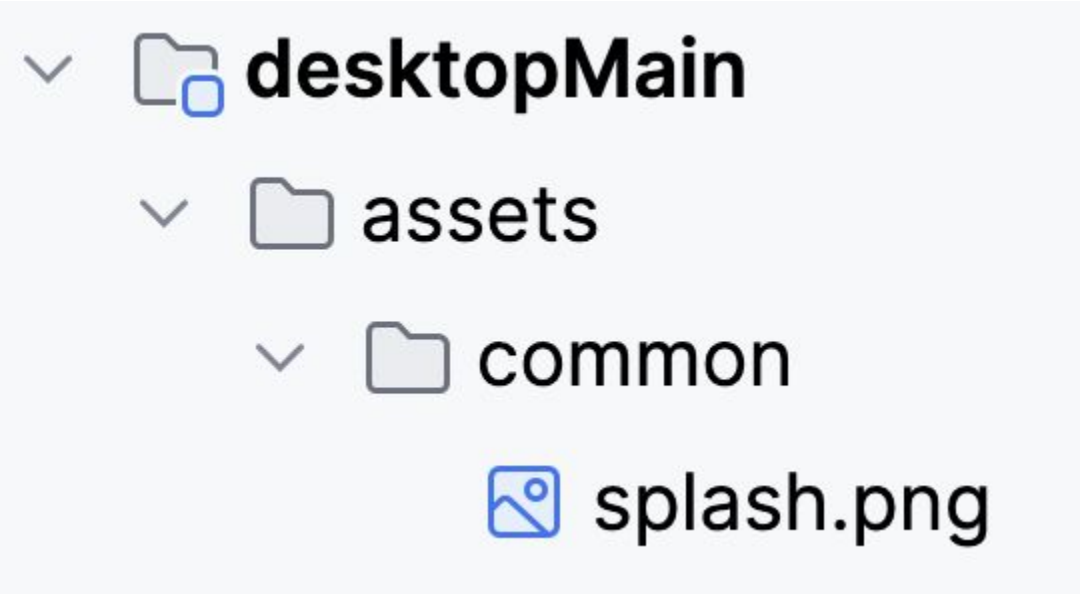
1. Open project in Xcode
2. Add the image to your Xcode project
3. Create the Launch Screen in Xcode properties
4. Run & profit



Steps for Desktop [static]

1. Add the png image to assets/common in desktopMain
2. Update the build.gradle.kts file for compose.desktop:
    a. Point to appResourcesRootDir to the assets directory
    b. Add a jvmArg for the splash screen
3. ./gradlew composeApp:runDistributable and profit

Add the PNG



Steps for Desktop [static]

```
compose.desktop {
    application {
        …
        nativeDistributions {
            …
            appResourcesRootDir = layout.projectDirectory.dir("src/desktopMain/assets")
            jvmArgs += "-splash:${'$'}APPDIR/resources/splash.png"
        }
    }
}
```

Extra resources for splash screens

- My article: How to Add a Splash Screen to a Compose Multiplatform App
- Philipp Lackner videos:
  - How to Build an Animated Splash Screen on Android - The Full Guide (2023)
  - Create a Splash Screen in Compose Multiplatform for iOS & Android - KMP for Beginners (2024)

Practical 1: Splash Screens [40 mins]

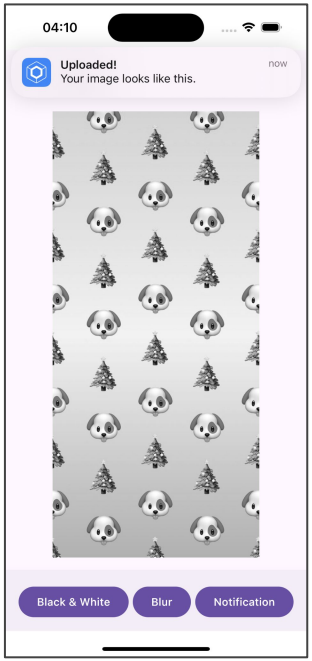
Do something you’re not familiar with / animate something.

Notifications

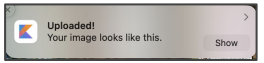
Local Notifications

- Notifications done using a library for Android/iOS/Desktop.
- Troubleshooting:
  - Desktop: make sure you are running with **runDistributable**.

Android



iOS



Desktop

Demo time

KMPNotifier by Mirzamehdi Karimov

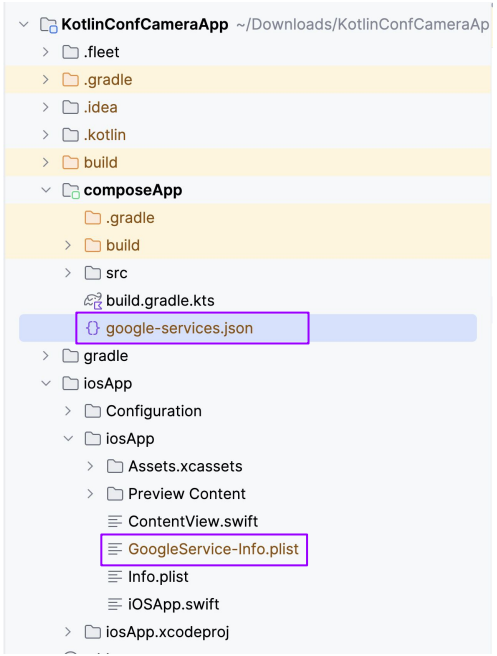
- Simple & easy-to-use.
- Supports local and push notifications (we'll only look at local ones).
- Local: Android, iOS, Desktop and Web (JS and Wasm).
- Remote: Android, iOS.

Steps to use KMPNotifier

0. Setup project in Firebase Console.
   - Download and add google-services.json file for Android.
   - Download and add GoogleService-Info.plist for iOS.
1. Initialize the library.
2. Send the notification.

Setup project in Firebase Console

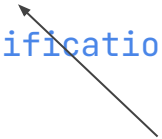
- Go to Firebase Console.
- Create the project as usual in Firebase.
- Download and add google-services.json file for Android.
- Download and add GoogleService-Info.plist for iOS.



Initialize the library - Android

- “First” thing in application onCreate:

```
NotifierManager.initialize(
    configuration = NotificationPlatformConfiguration.Android(
        notificationIconResId = R.drawable.camera_splash,
        showPushNotification = true,
    )
)
```

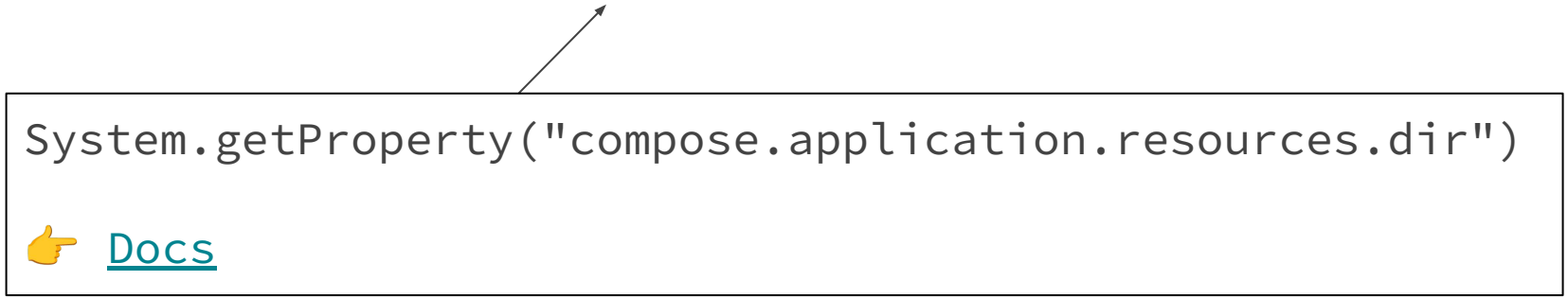


Branding icon: top bar and left of the notification body.

Initialize the library - Desktop

- “First” thing in **application** lambda:

```
NotifierManager.initialize(
    NotificationPlatformConfiguration.Desktop(
        showPushNotification = true,
        notificationIconPath = composeDesktopResourcesPath() + File.separator + "splash.png"
    )
)
```



Initialize the library - iOS - Create a delegate class

- In iOSApp.swift:

```
class AppDelegate: NSObject, UIApplicationDelegate {
    func application(_ application: UIApplication,
            didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey : Any]? = nil) -> Bool
{

        FirebaseApp.configure() //important

        NotifierManager.shared.initialize(configuration: NotificationPlatformConfigurationIos(
               showPushNotification: true,
               askNotificationPermissionOnStart: false,
               notificationSoundName: nil
        ))
       return true
    }

    func application(_ application: UIApplication,
            didRegisterForRemoteNotificationsWithDeviceToken deviceToken: Data) {
        Messaging.messaging().apnsToken = deviceToken
    }

}
```

UIApplicationDelegate is a protocol in iOS development that defines a set of methods for managing and responding to the **lifecycle events** of an iOS application.

It acts as the central hub for handling events such as app launch, termination, entering background or foreground, receiving push notifications, and more.

Initialize the library - iOS - Setup delegate in app

- In iOSApp.swift:

Swift property wrapper (similar to property delegate in Kotlin)

```
@main
struct iOSApp: App {

    @UIApplicationDelegateAdaptor(AppDelegate.self) var delegate

    var body: some Scene {
        WindowGroup {
            ContentView()
        }
    }
}
```

Send the notification

- In commonMain:

```
val notifier = NotifierManager.getLocalNotifier()
notifier.notify {
    id= Random.nextInt(0, Int.MAX_VALUE)
    title = "Uploaded!"
    body = "Your image looks like this."
    image = NotificationImage.File(imagePath)
}
```

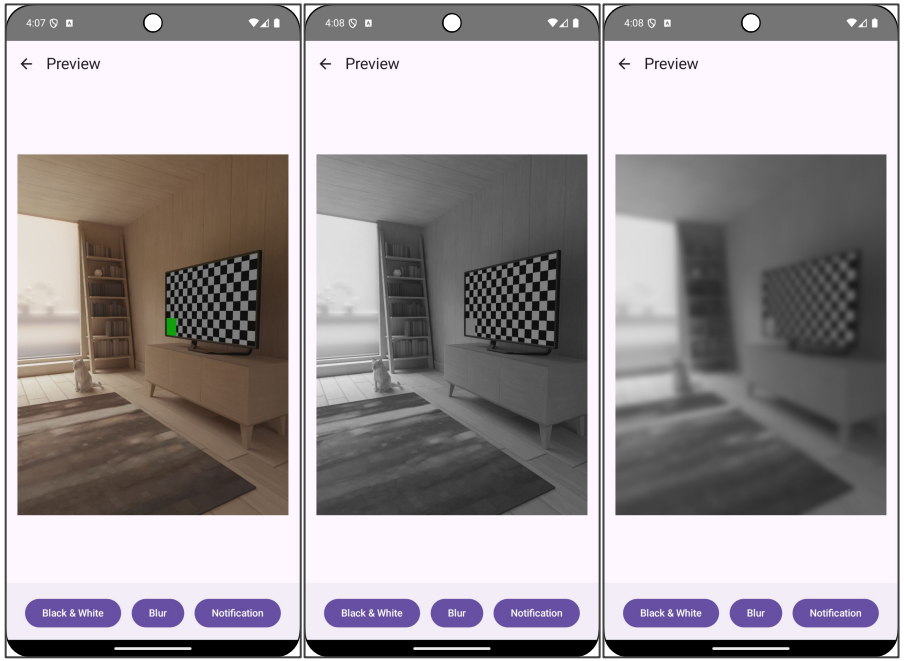

Other option is Url

Managing Files

Managing Files - Read/Filter/Write

Applying filters:

1. Check exists
2. Open, read
3. Filter based on filter type and platform
4. Save file in same spot
5. AsyncImage is updated - no caching!



Demo time

General how to decide - some advice from Kostya

- What to use? Depends on your tasks!
  - If you need a raw file management (read/write or something else), then kotlinx-io.
  - If you need a platform Filepicker UI to select a directory or file, then it's easier with FileKit.
  - If you need something else, then the native.

Our thought process

- We wanted to do **image filtering**, but having some platforms support **file picker** would be nice.
- What options do we have?
  - Kotlinx-io feels too low-level for image filtering.
  - Using a file utility like FileKit, writing the filters ourselves.
  - Using an image filter library like Peekaboo (but Android/iOS only).
  - Native for each platform, all the filters we want is already supported per platform.

Our thought process

- What options do we have?
  - Kotlinx-io feels too low-level for image filtering.
  - Using a file utility like FileKit, writing the filters ourselves.
  - Using an image filter library like Peekaboo (but Android/iOS only).
  - **Native for each platform, all the filters we want is already supported per platform - there was some repetition per platform we could reuse.**

Android Solution

1. Validation - Android
2. Load bitmap - Android
3. Create (copy) bitmap - Android
4. Create canvas - Android
5. Apply filter - unique
6. Draw bitmap - unique
7. Save bitmap - Android

For example - B&W filter:
android.graphics

```
val paint = Paint().apply {
    // Create a color matrix that
    // converts to grayscale
    val colorMatrix = ColorMatrix().apply {
        setSaturation(0f)
    }
    colorFilter =
        ColorMatrixColorFilter(colorMatrix)
}
```

Gaussian blur: RenderScript

Desktop Solution

---

1. Validation - Desktop
2. Load image - Desktop
3. Apply filter - unique
4. Save image - Desktop

For example - B&W filter:
java.awt.image

```
val blackAndWhiteImage =
BufferedImage(width, height,
BufferedImage.TYPE_BYTE_GRAY)
```

Gaussian blur: also java.awt.image with own Gaussian kernel.

iOS Solution

1. File validation - iOS
2. Rest - varies a bit but UIKit & CoreGraphics

B&W: Color space for greyscale

Gaussian blur: Filter

Additional Setup and Quick Lookaround

Setup + lookaround suggestions [30 mins]

- Project is in: **full-example** directory.
- Look in the **docs** directory for the Practical .instructions. They provide a guide if you'd like it.
- Look:
  - How the UI is organised and navigation is set up.
  - How the platform-specific cameras work (note the @Composable expect/actual fun.)
  - How the platform-specific photo filters work.

Advanced Kotlin Multiplatform for iOS Targets

What we'll be looking at:

- Debugging with xcode-kotlin.
- Further debugging with LLDB.
- Understanding and improving Kotlin/Swift interoperability.

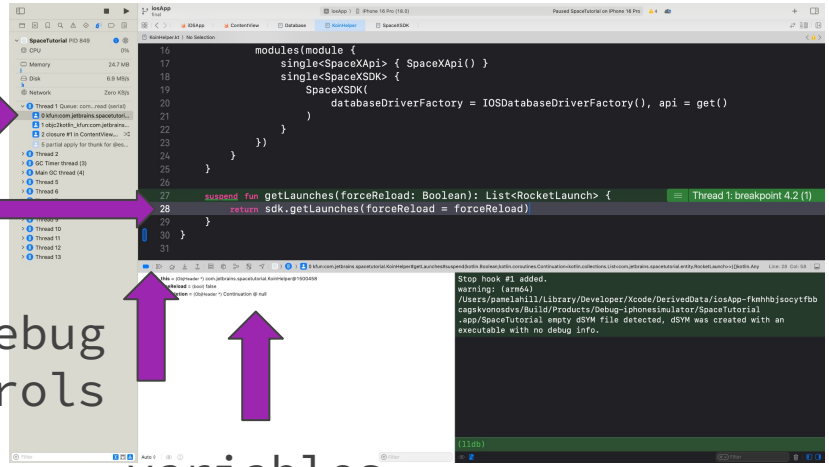
Debugging in Xcode

backstack

line numbers (can create breakpoints)

debug controls

variables



this is draggable

LLDB console

Debugging with xcode-kotlin

Introduction to xcode-kotlin

- Open-source tool from Touchlab.
- Plugin to improve the Kotlin/Swift debugging experience.
- Supports some common debugging activities.
- Doesn’t support state inside suspend functions so well.

Installation / Syncing (Terminal)

- Installation using Homebrew:

```
brew install xcode-kotlin
xcode-kotlin install
```

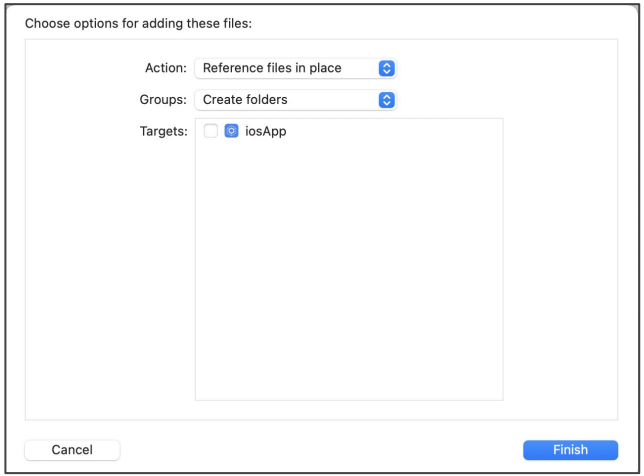
- Syncing with new Xcode versions:

```
xcode-kotlin sync
```

https://touchlab.co/xcodekotlin

Demo time

Setup of sources in xcode-kotlin



1. Right-click -> New Group.
2. Right-click -> Add files to iOS App.
3. Select commonMain and iosMain.
4. Set Action to “Reference files in place”.
5. Set Groups to “Create folders”.
6. Make sure iosApp is **unchecked** in Targets.
7. Finish.

Extra resources

- https://touchlab.co/xcodekotlin
- https://touchlab.co/xcode-kotlin-2-0
- #touchlab-tools channel on Kotlinlang

Debugging with LLDB

Introduction to LLDB

- Powerful debugger environment for Apple platforms.
- Debug Swift / Objective-C / C / C++ code.
- Only on macOS platforms, from terminal / Xcode.

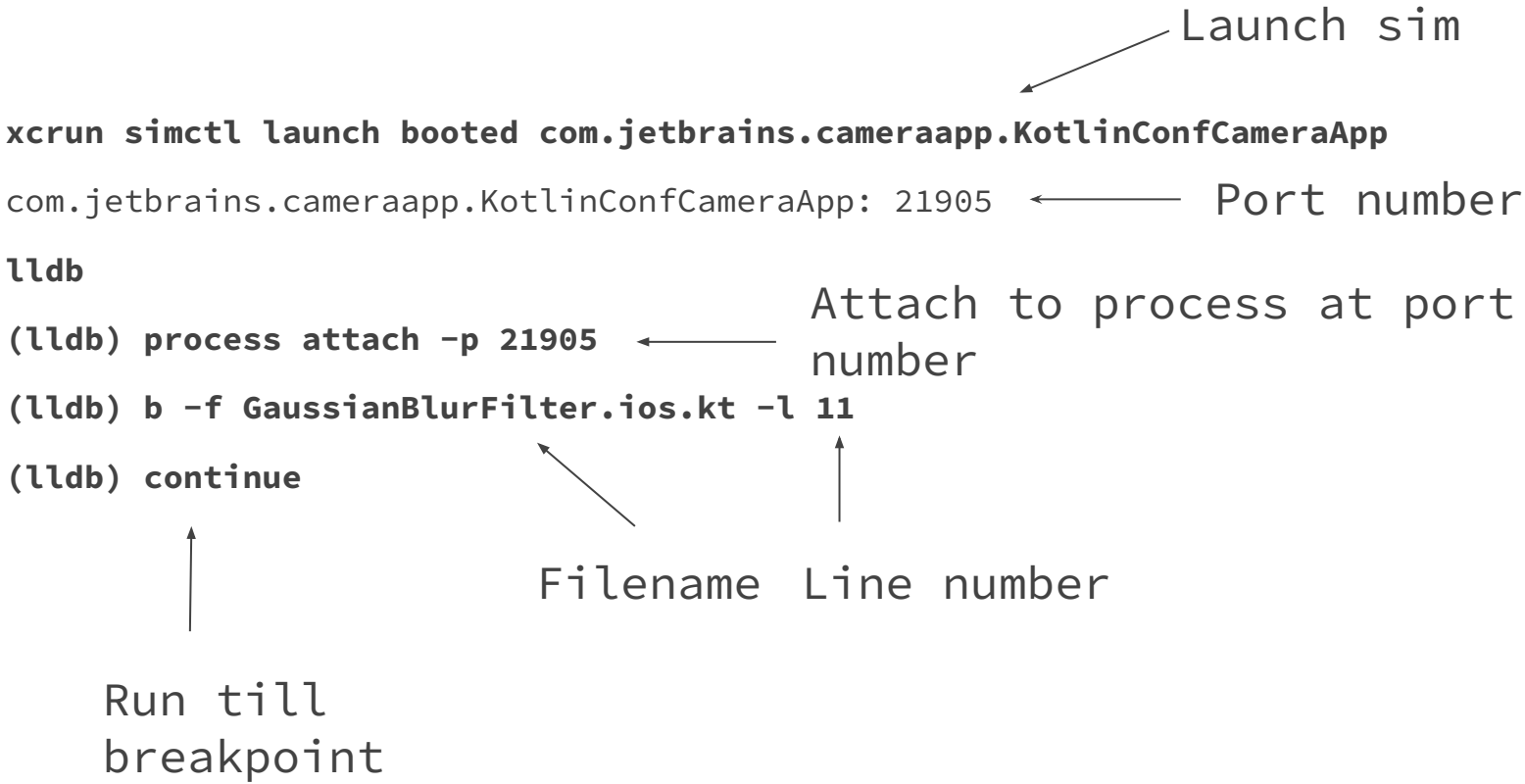
Simple breakpoint commands

- Breakpoint at line number: b line_number
- Breakpoint at function: b function_name
- Breakpoint in file at line: b -f filename -l linenumber
- Breakpoint list: breakpoint list (the first number is the id of the breakpoint)
- Disable breakpoint: breakpoint disable id
- Enable breakpoint: breakpoint enable id
- Delete breakpoint: breakpoint delete id

Simple debugging

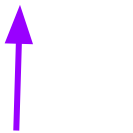
- Help: help
- Continue (all threads): c
- Next step with into (source-level): step
- Step over (source-level): n
- Step out (source-level): finish
- Print variable or expression: p variable|expression
- Print expression with formatting: po expression
- Backtrace: bt

Connecting to a simulator, creating a breakpoint



Loading konan_lldb.py for pretty printing values

```
(lldb) command script import
/Users/pamelahill/.konan/kotlin-native-prebuilt-macos-aarch64
-2.1.20-RC2/tools/konan_lldb.py
```



Version of Kotlin you used

Your username

Your architecture

But what is the best way? UI vs CLI

- It depends! BUT
- Learn to use LLDB because
  - UI doesn’t support everything, for example:
    - Regex breakpoints
    - Commands for understanding the loaded libraries: look at **target modules** commands (use help)
    - Expression evaluation for Kotlin/Native
  - If you want to shell script LLDB itself in the future

Practical 2: Debugging Kotlin/Swift [20 mins]

Other resources

- (Kotlin docs) Debugging Kotlin/Native
- Documentation from Apple
- Learn the lldb debugger basics in 11 minutes | 2021 (Also works on M1 Apple Silicon) - Mike Shah

Understanding and Improving Kotlin/Swift Interoperability

What we'll be looking at:

- Quick recap of my talk (KotlinConf 2024).
- Discuss and demo Swift Export (current state).
- Discuss Touchlab's SKIE.
- But, what about Swift Import?

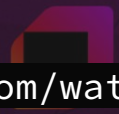
Kotlin Multiplatform Alchemy

Making Gold out of Your Swift Interop

Quick recap

Pamela Hill

Developer Advocate - JetBrains



https://www.youtube.com/watch?v=P_5ZEtK05kc

Kotlin/Swift interop

=

Calling Kotlin declarations from Swift

What exactly is the problem?

- Decision made in ~2018 to be interoperable with Objective-C, not Swift
- Why?
  - Can be used in all iOS projects
  - Swift still on the road to maturity/adoption at that stage

This means:
Kotlin is **not directly** interoperable with Swift
but **indirectly** via an Objective-C bridge.

What this means for Swift developers

Some Kotlin features:

- Work exactly as expected
- Work with a small workaround
- Work better with a community solution
- Don’t work optimally right now
- Don’t work


Member properties
(Readonly or Mutable)
Top-level properties (Read-only or Mutable)
Classes and functions
Top-level functions
Annotations
Inline functions
Exceptions
Functions with default arguments
Functions expecting lambda with receiver
Function overloads
Collections with custom types
Generic classes
Higher-order functions
(lambda as Parameter
or Return)
Flows
Functions with receivers
Suspend functions
Collections with basic types
Optional basic types
Data classes
Mutable / immutable collections
Fun interfaces
Unit and Nothing
Generic functions
Sealed classes
Objects
Contravariant generics
Basic types
Enum classes
Abstract classes
Open classes
Inner classes
Companion objects
Covariant generics
Interfaces
Extension function over platform class
Sealed interfaces
Extension function over usual class
Bounded generics
Extension properties over platform class
Reified functions
Generic interfaces
Extension properties over usual class
Inline classes
Extension properties for companion object of
usual class
Star projections
Functions with value class parameter
Extension properties for companion object of
platform class
Functions with a vararg parameter

forked from hhru/kotlin-swift-interopedia

Kotlin/Swift Interopedia [kotl.in/interopedia]

- Interoperability encyclopedia
- Explains how Kotlin features currently work from Swift
- Lots of code samples
- Gives workarounds, community solutions
- Includes playground app with runnable samples

Overview

Classes and functions	You can instantiate Kotlin classes and call Kotlin functions from Swift: SimpleClass().simpleFunction().
Top-level functions	You can access a top-level function via the wrapper class: TopLevelFunctionKt.topLevelFunction().
Types	Simple types and custom types can be passed as arguments and returned from function calls.
Collections	Kotlin and Swift have very similar kinds of collections and can be mapped between each other.
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.
Interop annotation - @ObjCName	Gives better Objective-C/Swift names to Kotlin constructs like classes, functions and so on, without actually renaming the Kotlin constructs. Experimental.
Interop annotations - @HiddenFromObj	Hides a Kotlin declaration from Objective-C/Swift. Experimental.
Interop annotations - @ShouldRefineInSwift	Helps to replace a Kotlin declaration with a wrapper written in Swift. Experimental.
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.

Current interop tricks

- **@ObjCName** - handling renaming of Kotlin declarations in Swift.
- **@ShouldRefineInSwift** - refining declarations better in Swift.
- **@HiddenFromObjC** - not exporting declaration to Objective-C/Swift.
- **KDoc** - documenting shared Kotlin API.
- **@Throws** - listing exceptions which should be caught.
- **Abstract classes** instead of typed interfaces - to retain type argument.
- Suspend functions / Flows - **libraries** like KMP-NativeCoroutines & SKIE.

Swift Export

Goal:
To deliver a more pleasant experience
for iOS developers out-of-the-box

We want to solve the following problems:

- Generated Objective-C API is considered unaesthetic
- Support more language features
- Better integration of compiler with other parts of developer experience

The team is still hard at work!

Follow their progress on YouTrack: KT-64572

Supported features (pre-alpha)

- Multi-modules
- Top-level functions (and so also extension functions and properties)
- Type aliases
- Overloaded functions

Known limitations

- No inheritance from Kotlin interfaces/classes from Swift
- Coroutines / Flows aren’t supported yet
- Generics have type erasure
- Only creates Swift sources, not binaries

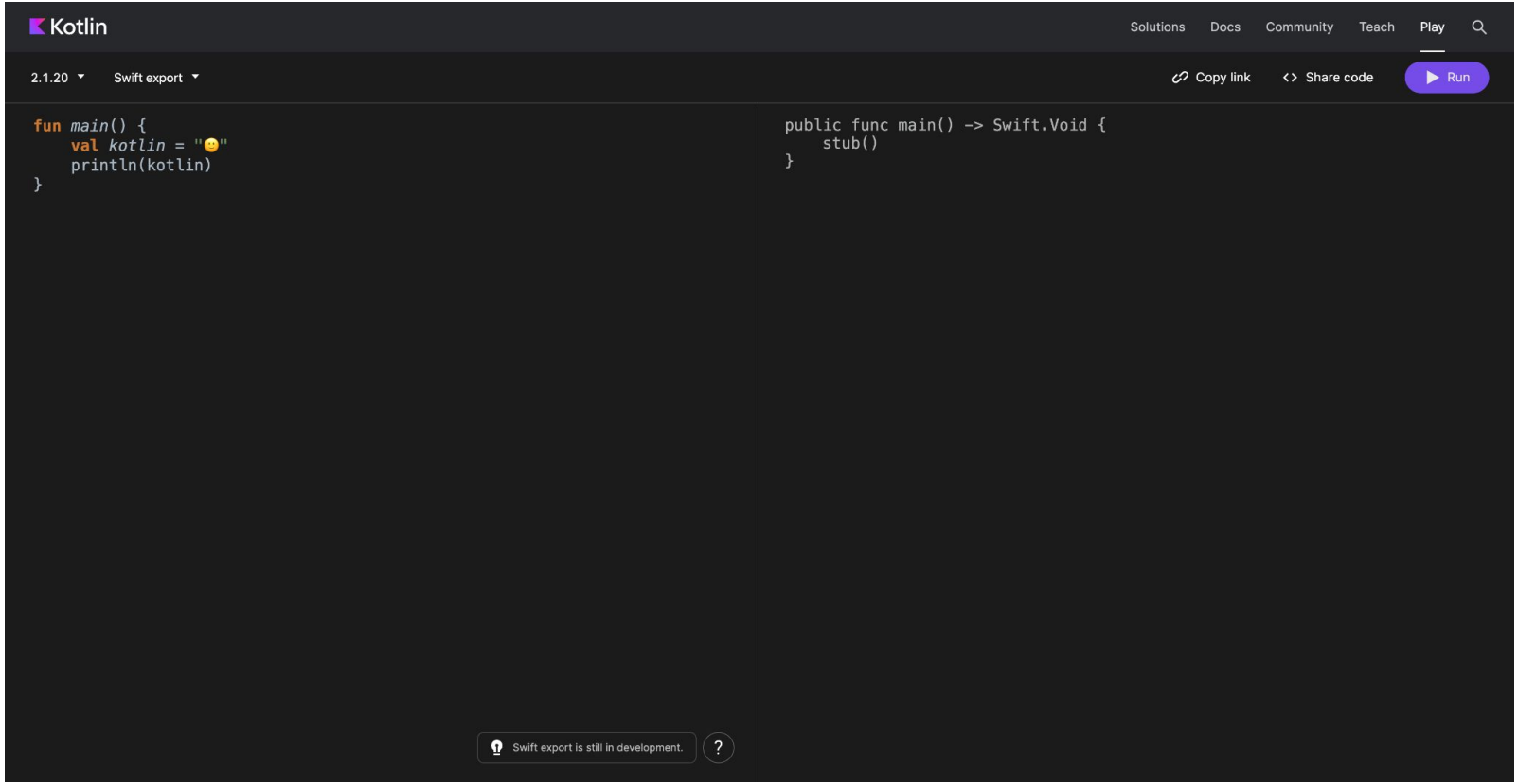
Best way to currently experiment

https://github.com/Kotlin/swift-export-sample

Note: Some unusual setup required

Demo time

```
kotlin.experimental.swift-export.enabled=true
```



https://play.kotlinlang.org/

Swift Export will improve what is possible to improve

Touchlab's Swift Kotlin Interface Enhancer (SKIE)

Supported features

- Enumerations - generates code that will be exhaustive for ‘switch’ etc.
  - Enums
  - Sealed Classes and Interfaces
- Functions
  - Default Arguments - should be used with caution (compile times)
  - Global Functions
  - Interface Extensions
  - Overloaded Functions - leaves parameter name untouched
- Coroutines / Reactive Interface - main thread safety out-of-box, handles cancellation correctly.
  - Suspend Functions
  - Flows - AsyncSequence
- Swift Code Bundling - write custom Swift wrappers for Kotlin code.

What to choose

- Swift Export’s feature set overlaps with SKIE.
- Unclear when/if we’ll cover same features SKIE is offering.
- SKIE is maintained Touchlab, and might require updates for each Kotlin compiler update.
- It’ll depend on your project’s needs which one you’ll want to use.

What about Swift Import?

spmForKmp plugin

- Alternative for the aging CocoaPods plugin
- No 1st-party support yet, it will only happen after Swift Export
- Import your own / 3rd-party SPM dependencies for use in your Kotlin code

Swift Package Manager For Kotlin Multiplatform

The Swift Package Manager for Kotlin Multiplatform Plugin, aka `spmForKmp` Gradle Plugin, is an **alternative of the dying CocoaPods Plugin** used by KMP cocoapods plugin.

It will help you to integrate Swift Package and simplify communication between Swift/Kotlin Multiplatform projects targeting the **Apple platform**.

The plugin uses the embedded Swift Package Manager, so **no third-party dependency is needed**, and it's less intrusive than CocoaPods.

> **Please Be Aware**
>
> Pure Swift packages can't be exported to Kotlin; the plugin will help you to create a bridge to bypass this issue.
>
> It's a manual job, but until the Swift-import is (not currently planned) available in KMP, it's the only way.

Features

- **Create a Swift<->Kotlin bridge**: Import your own Swift code for functionality that can't be done in Kotlin.
- **Use SPM third-Party Dependency**: Add external dependency and use it inside your bridge
- **Import Swift-compatible code to Kotlin**: Enable SPM dependencies and your own Swift code to be exposed directly in your Kotlin code (if compatible).

Support My Project ⭐

If you find this project useful, please consider giving it a star or a coffee!

https://frankois944.github.io/spm4Km

Code Quality

What we'll be looking at:

- Adding code coverage (JVM-only) with Kover.
- Modularizing your application.

Code coverage with Kover (JVM-only)

Why code coverage?

- Functionality that has not been properly tested
- Edge cases
- Dead code

BUT NOT

- Correctness of the program or test

What is JaCoCo? Why not use it?

- Well-known code-coverage tool from Java world.
- For Kotlin code, some quirks like inline functions.

What about Kotlin Multiplatform projects?

- Different locations of files.
- Common sources easily lost.

A quick intro to Kover

- Gradle plugin from JetBrains.
- Support for Kotlin/JVM, Kotlin Multiplatform, Kotlin/Android projects.
- Support for instrumentation on JVM, not Android. Otherwise just unit tests.
- Supports multiple modules in your projects.
- HTML/XML reports - feed it into 3rd party tools like codecov (quality tool).

Setup:

1. Add the following to project-level build.gradle.kts file:

```
plugins {
    …
    id("org.jetbrains.kotlinx.kover") version "0.9.1"
}
```

2. Add the following to the shared build.grade.kts file:

```
plugins {
    …
    id("org.jetbrains.kotlinx.kover")
}
```

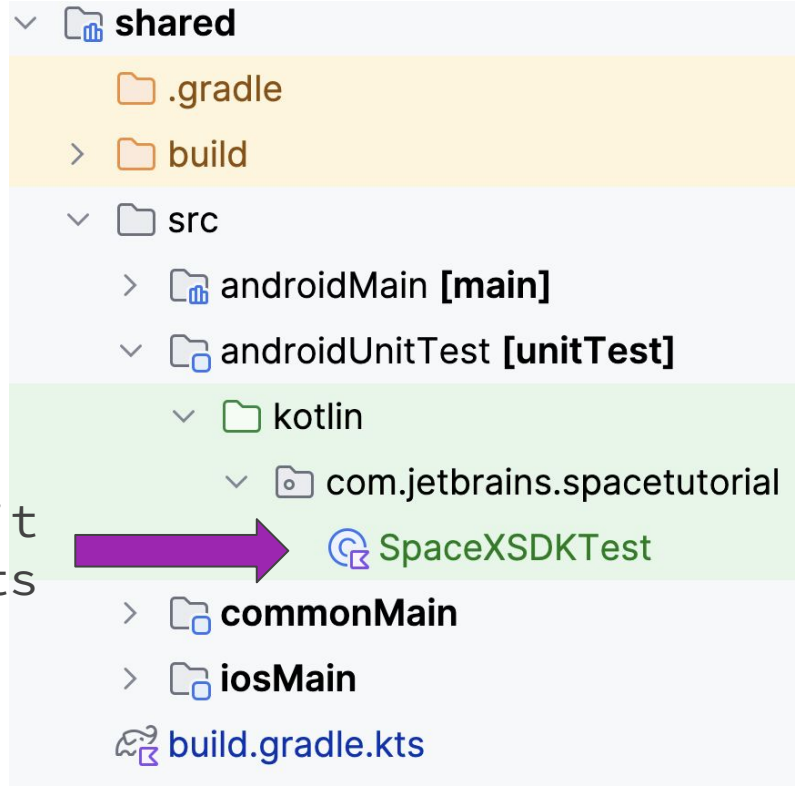
3. Sync Gradle

Example: Setup unit tests dependencies

```
kotlin {
    …
    sourceSets {
        …
        val androidUnitTest by getting {
            dependencies {
                implementation(kotlin("test"))
                implementation(kotlin("test-junit"))
                implementation("org.jetbrains.kotlinx:kotlinx-coroutines-test:1.7.3")
                implementation("io.mockk:mockk:1.13.8")
            }
        }
    }
}
```

Add unit tests:

Four unit tests



Creating the report:

1. Run **./gradlew koverHtmlReportDebug**
   a. Where to find the report:

```
> Task :shared:koverHtmlReportDebug
Kover: HTML report for ':shared' file:///Users/pamelahill/StudioProjects/deep-dive-into-kmp/kover/shared/build/reports/kover/htmlDebug/index.html
```

   b. How many tests were run:

```
BUILD SUCCESSFUL in 5s
19 actionable tasks: 4 executed, 15 up-to-date
```

Current scope: shared | all classes

shared: Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %	Instruction, %
all classes	56.2% (9/16)	23.8% (10/42)	16.7% (4/24)	24.4% (30/123)	23.9% (165/690)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %	Instruction, %
com.jetbrains.spacetutorial	100% (1/1)	100% (2/2)	100% (4/4)	100% (8/8)	100% (46/46)
com.jetbrains.spacetutorial.cache	60% (3/5)	15% (3/20)	0% (0/6)	6.3% (5/79)	5.7% (18/314)
com.jetbrains.spacetutorial.cache.shared	66.7% (2/3)	33.3% (2/6)		33.3% (3/9)	50% (16/32)
com.jetbrains.spacetutorial.entity	50% (3/6)	33.3% (3/9)	0% (0/14)	70% (14/20)	32.3% (85/263)
com.jetbrains.spacetutorial.network	0% (0/1)	0% (0/5)		0% (0/7)	0% (0/35)

generated on 2025-05-03 04:56

Current scope: shared | all classes | com.jetbrains.spacetutorial

Coverage Summary for Package: com.jetbrains.spacetutorial

Package	Class, %	Method, %	Branch, %	Line, %	Instruction, %
com.jetbrains.spacetutorial	100% (1/1)	100% (2/2)	100% (4/4)	100% (8/8)	100% (46/46)

Class	Class, %	Method, %	Branch, %	Line, %	Instruction, %
SpaceXSDK	100% (1/1)	100% (2/2)	100% (4/4)	100% (8/8)	100% (46/46)

generated on 2025-05-03 04:56

Current scope: shared | all classes | com.jetbrains.spacetutorial

Coverage Summary for Class: SpaceXSDK (com.jetbrains.spacetutorial)

Class	Method, %	Branch, %	Line, %	Instruction, %
SpaceXSDK	100% (2/2)	100% (4/4)	100% (8/8)	100% (46/46)
SpaceXSDK$getLaunches$1				
Total	100% (2/2)	100% (4/4)	100% (8/8)	100% (46/46)

```
 1 package com.jetbrains.spacetutorial
 2
 3 import com.jetbrains.spacetutorial.cache.Database
 4 import com.jetbrains.spacetutorial.cache.DatabaseDriverFactory
 5 import com.jetbrains.spacetutorial.entity.RocketLaunch
 6 import com.jetbrains.spacetutorial.network.SpaceXApi
 7
 8 class SpaceXSDK(databaseDriverFactory: DatabaseDriverFactory, val api: SpaceXApi) {
 9     private val database = Database(databaseDriverFactory)
10
11     @Throws(Exception::class)
12     suspend fun getLaunches(forceReload: Boolean): List<RocketLaunch> {
13         val cachedLaunches = database.getAllLaunches()
14         return if (cachedLaunches.isNotEmpty() && !forceReload) {
15             cachedLaunches
16         } else {
17             api.getAllLaunches().also {
18                 database.clearAndCreateLaunches(it)
19             }
20         }
21     }
22 }
```

Demo time

Additional resources

- Unit & UI Testing With Compose Multiplatform - KMP for Beginners - Philipp Lackner
- Kover - The Code Coverage Plugin
- Kover Documentation

Modularization

What is modularization?

- Organizing code into loosely coupled and self contained parts - **modules**.
- Each module is independent and has a clear purpose.
- By dividing the problem domain into smaller subproblems, reducing the design/maintenance complexity of the larger system.

Why modularization?

- **Reusability** of modules between apps.
- **Strict visibility controls** between modules (using internal/private modifiers) .
- **Scalability** - changes are mostly localised.
- **Possibly ownership** - dedicated owner for a module responsible for maintenance and further development.
- **Encapsulation** - isolated code is easier to read and understand.
- **Testability** - components can be easily tested in isolation.
- **Build time improvements** - some Gradle functionality can leverage modularity.
  - Incremental build, build cache, parallel build

Strategies

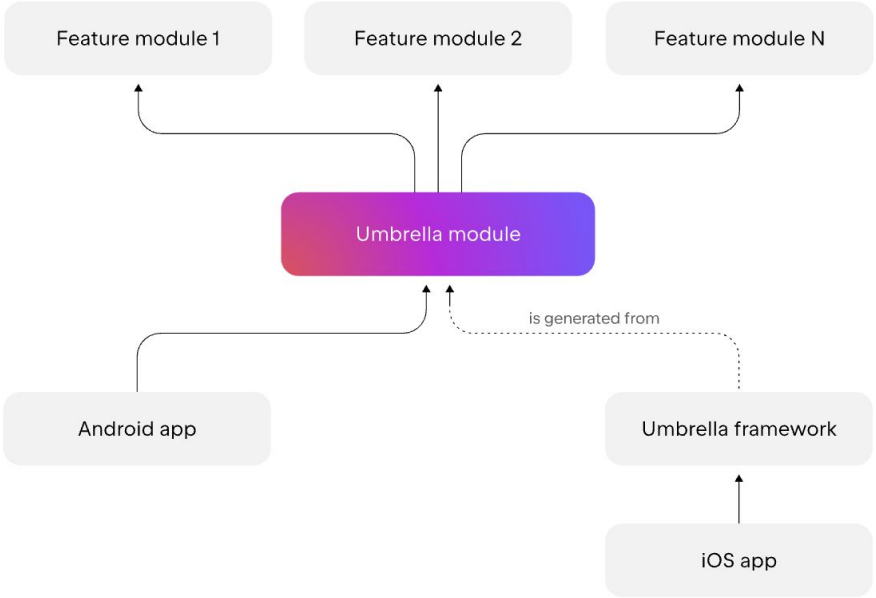
- Layer division - application’s architectural layers such as presentation / domain / data.
- Feature division - different features like camera / filters / upload.

Common pitfalls

- Too fine-grained - many small modules.
  - Each module has overhead of build complexity/boilerplate.
  - Complex/distributed build configuration is difficult to keep consistent across modules.
  - Too much boilerplate makes codebase cumbersome/difficult to maintain.
- Too coarse-grained - too large modules, monolith.
  - Miss benefits of modularization.
- Too complex - size of codebase doesn’t warrant it.
  - The codebase is small and won’t grow too much.

Umbrella framework ☔

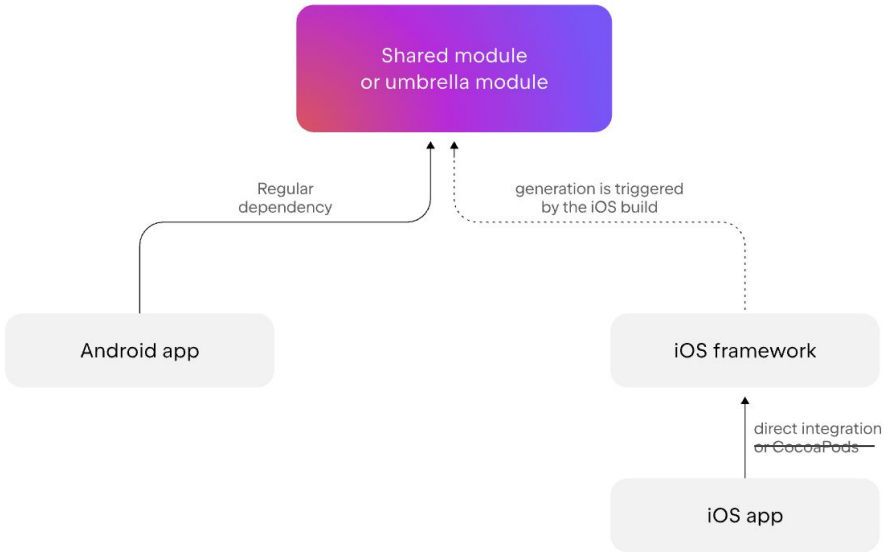
- iOS apps can only depend on one KMP framework.
- For several modules, you need an extra module depending on all of the modules you’re using - an **umbrella module.**
- An **umbrella framework** is a framework containing all the project’s shared modules. This framework is imported into the iOS app.
- Android app can depend on Umbrella module for consistency.



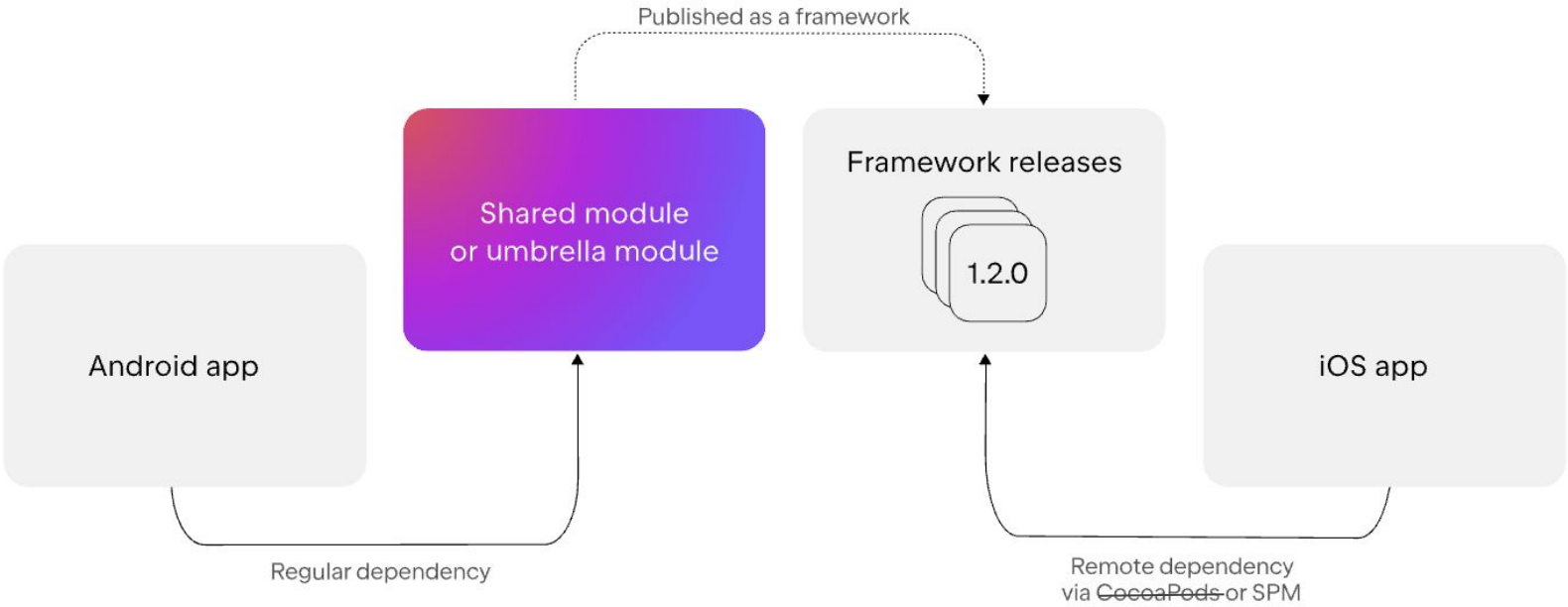
Why an umbrella framework ☔?

- When a KMP module is compiled into a framework, the resulting framework **includes all of its dependencies.**
- Whenever two or more modules use the same dependency and are exposed to iOS as separate frameworks, the Kotlin/Native compiler **duplicates the dependencies** (for example, the Kotlin stdlib might be duplicated).
- This duplication causes a number of **issues:**
  - iOS app size is unnecessarily inflated.
  - A dependency's code structure is incompatible with the duplicated dependency's code structure.

How to distribute (Local)



How to distribute (Remote)



Practical 3: Modularisation [40 mins]

Additional resources

- Guide to Android app modularization
- Using Kotlin from local Swift packages
- Swift package export setup
- Choosing a configuration for your Kotlin Multiplatform project
- KMMBridge

App Quality

What we'll be looking at:

- Fixing memory leaks.
- App size optimization (for iOS).

Fixing memory leaks

Memory management in Kotlin

- Garbage collection.
- Automatic memory management where the GC reclaims memory occupied by unreferenced (ie. not strongly referenced) objects.
- This frees up memory for other objects.
- The GC runs according to policies, but you shouldn’t try to guess when/if it will run.

Memory management in Objective-C and Swift

- Reference counting.
- Stores the number of references to an object, and releases the memory when it reaches zero.

Brainteaser 🤯 - where and why is this code leaking memory?

```
val superview = UIView()
val subview = object : UIView() {
    val ref = superview
}
…

superview.addSubview(subview)
```

What about now? 🤯

```
                        iOS
val superview = UIView()  iOS
val subview = object : UIView() {
    val ref = superview
}
…
                    iOS
superview.addSubview(subview)
```

Some terminology

- A **retain cycle** occurs when a program consistently references an object, and this chain of references is **never broken.**
- **Strong references** prevent an object from being deallocated.
- **Weak references** do not prevent an object from being deallocated.
- **Soft references**? Weak ref but will not be collected until memory constraints require it
- There are also **unowned references**

Retain cycle between Kotlin and Obj-C/Swift

```
                    iOS
val superview = UIView()  iOS
val subview = object : UIView() {
    val ref = superview
}
…
                  iOS
superview.addSubview(subview)
```

Primary source of the leak

Kotlin (strong) references an ObjC object and ObjC references Kotlin object. This causes a retain cycle!

Breaking the cycle with WeakReference

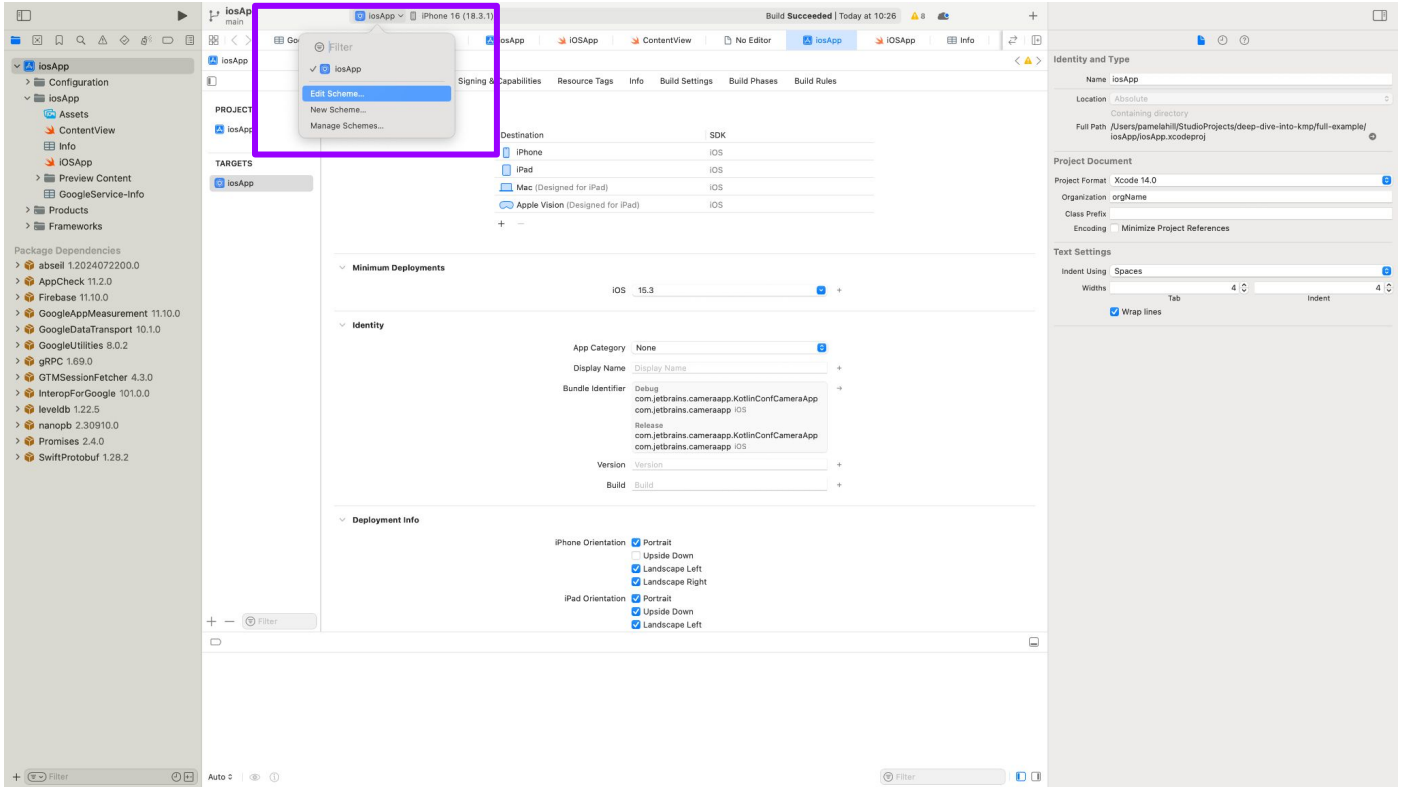
```
                        iOS
val superview = UIView()  iOS
val subview = object : UIView() {
    val ref = WeakReference(superview)
}
…
                    iOS
superview.addSubview(subview)
```

Demo time

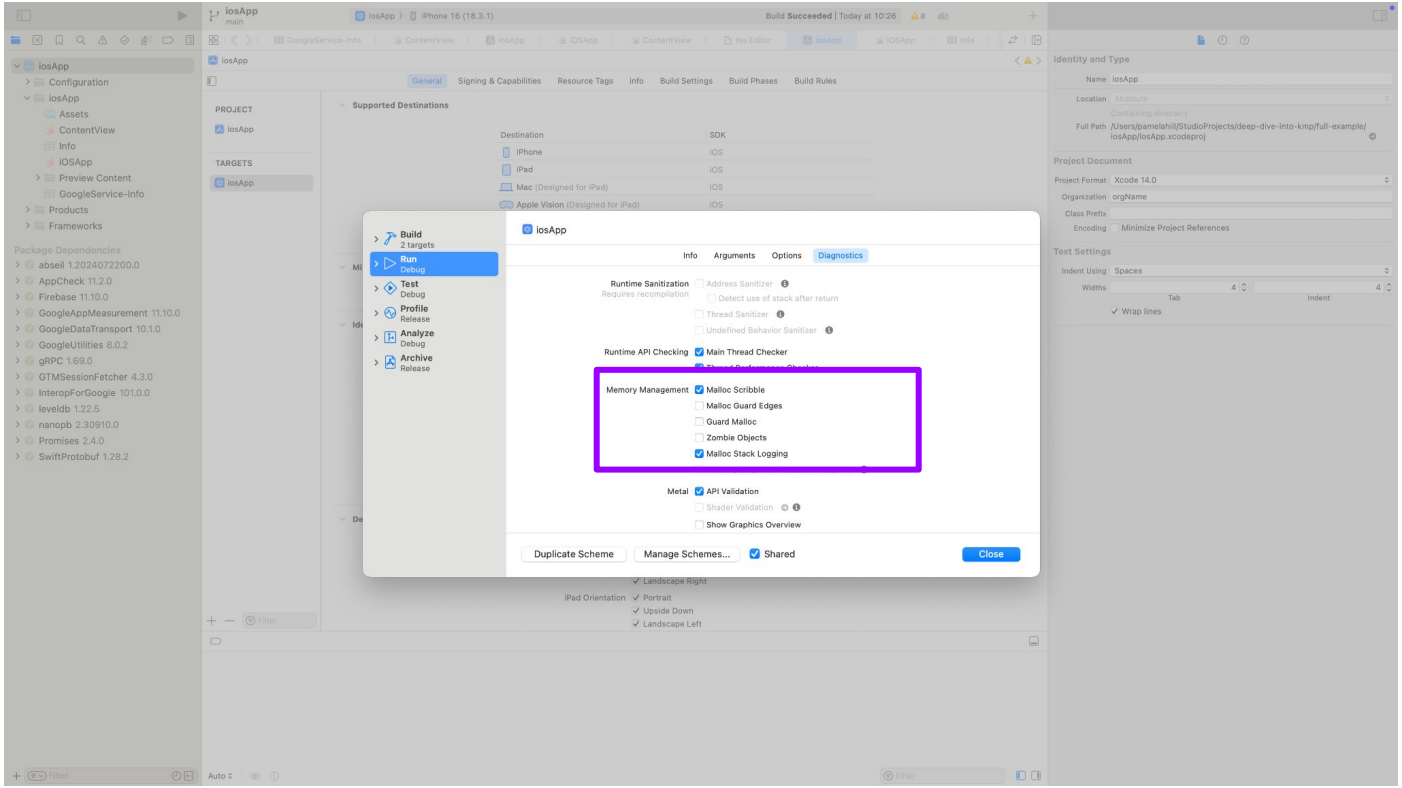
Steps to debug a memory leak in Xcode

1. Enable malloc debugging info on the scheme.
2. Run the application.
3. Open the debug navigator (memory section).
4. Create the debug memory graph.
5. Filter to only the leaked allocations.
6. Select the leak.
7. View leak backtrace.
8. Fix the memory leak and review!

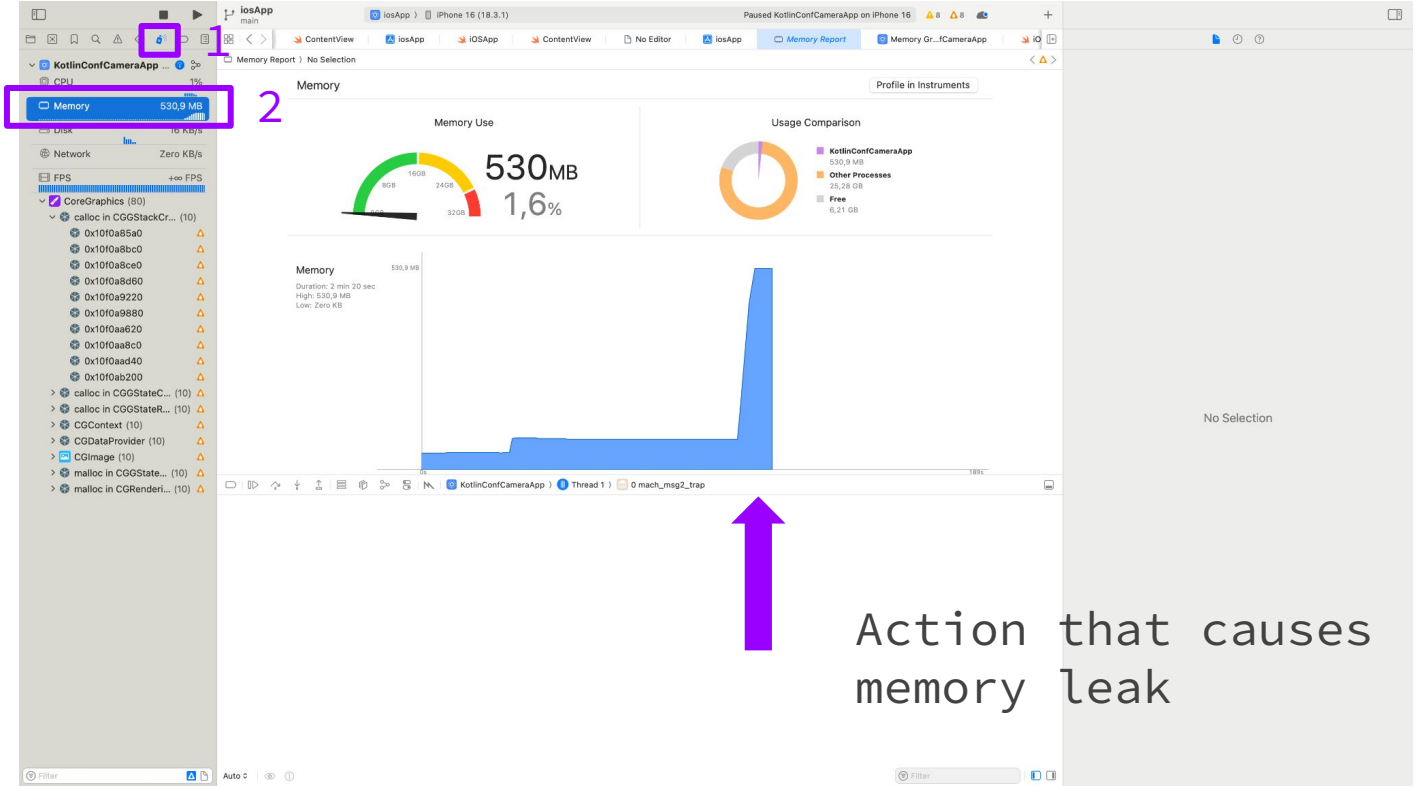
Enabling malloc debugging info



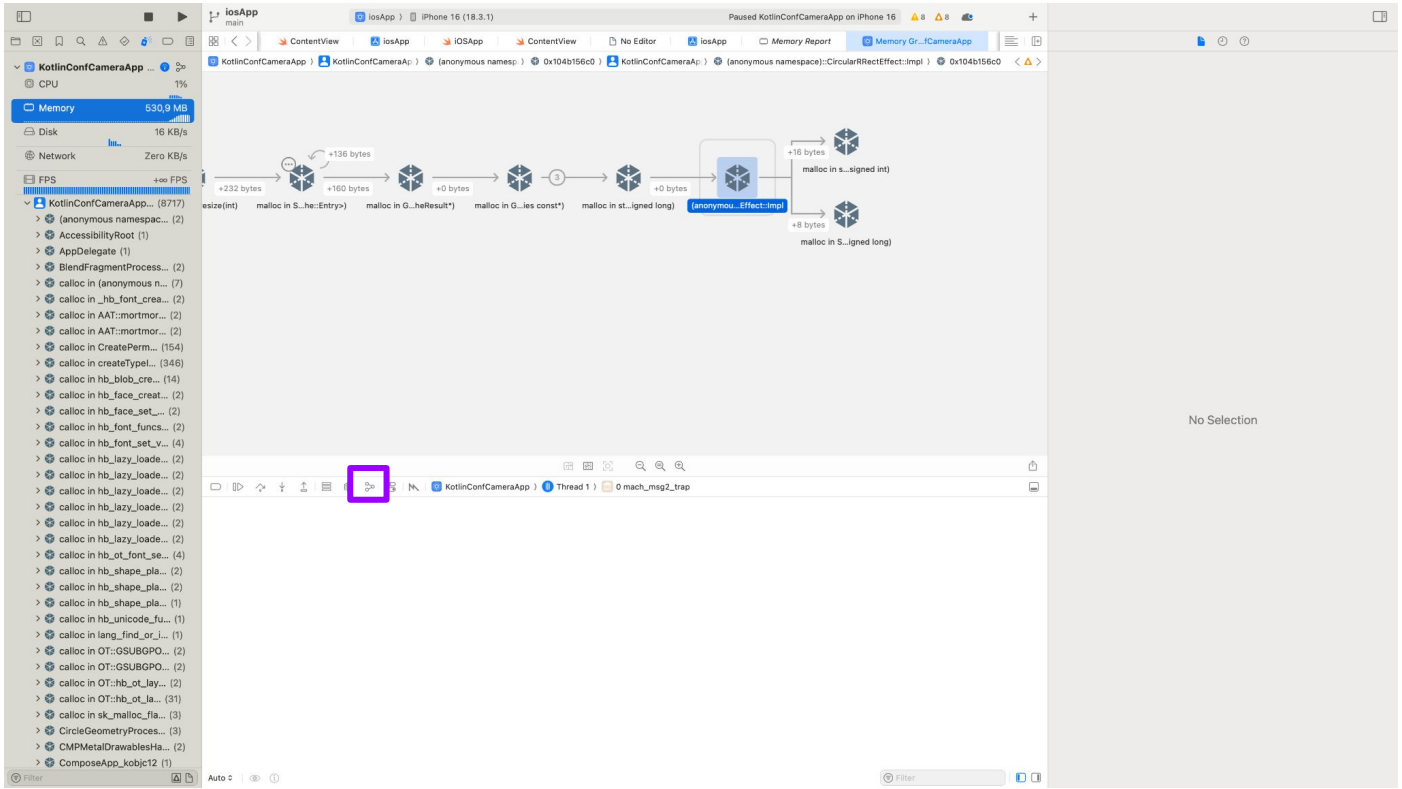
Enabling malloc debugging info



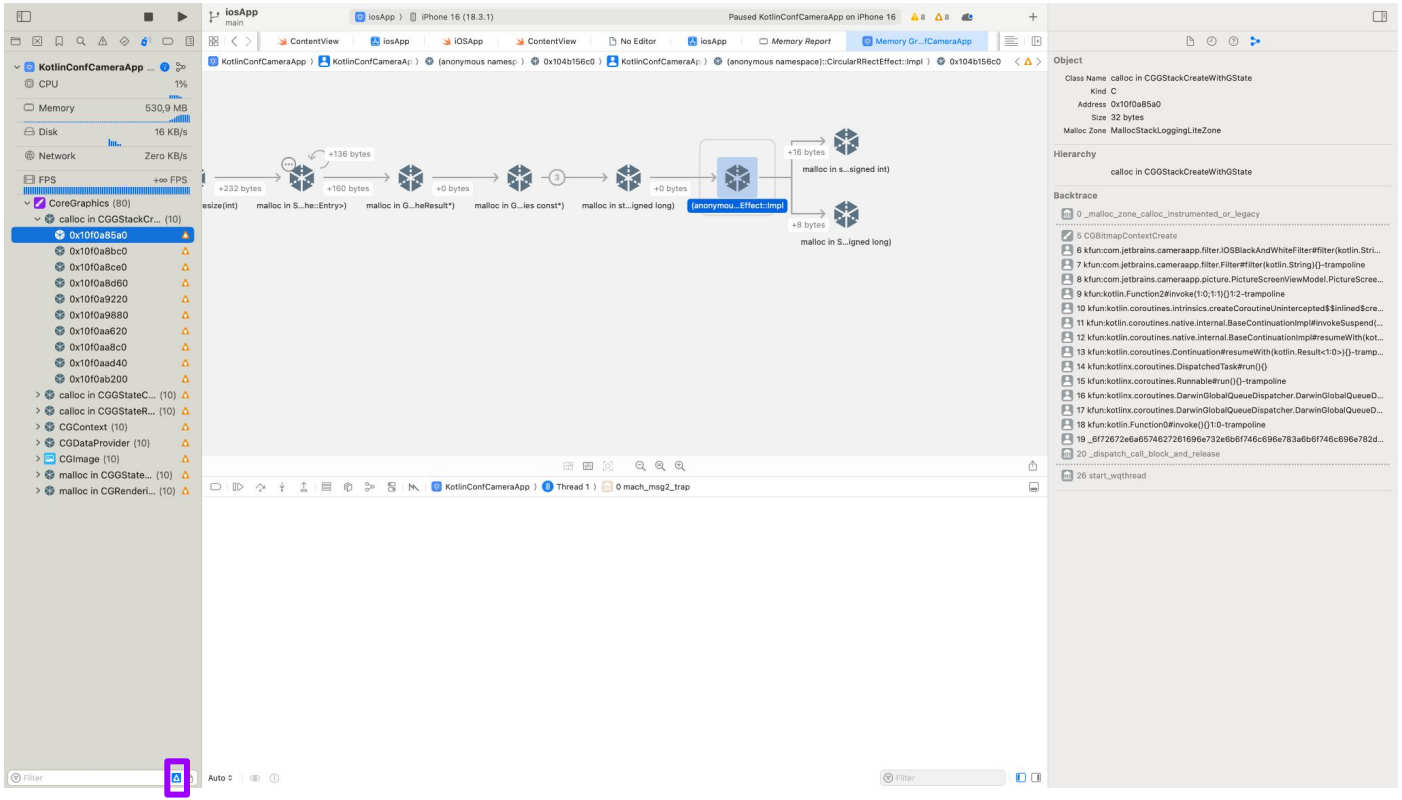
Run, then look at memory in Debug Navigator



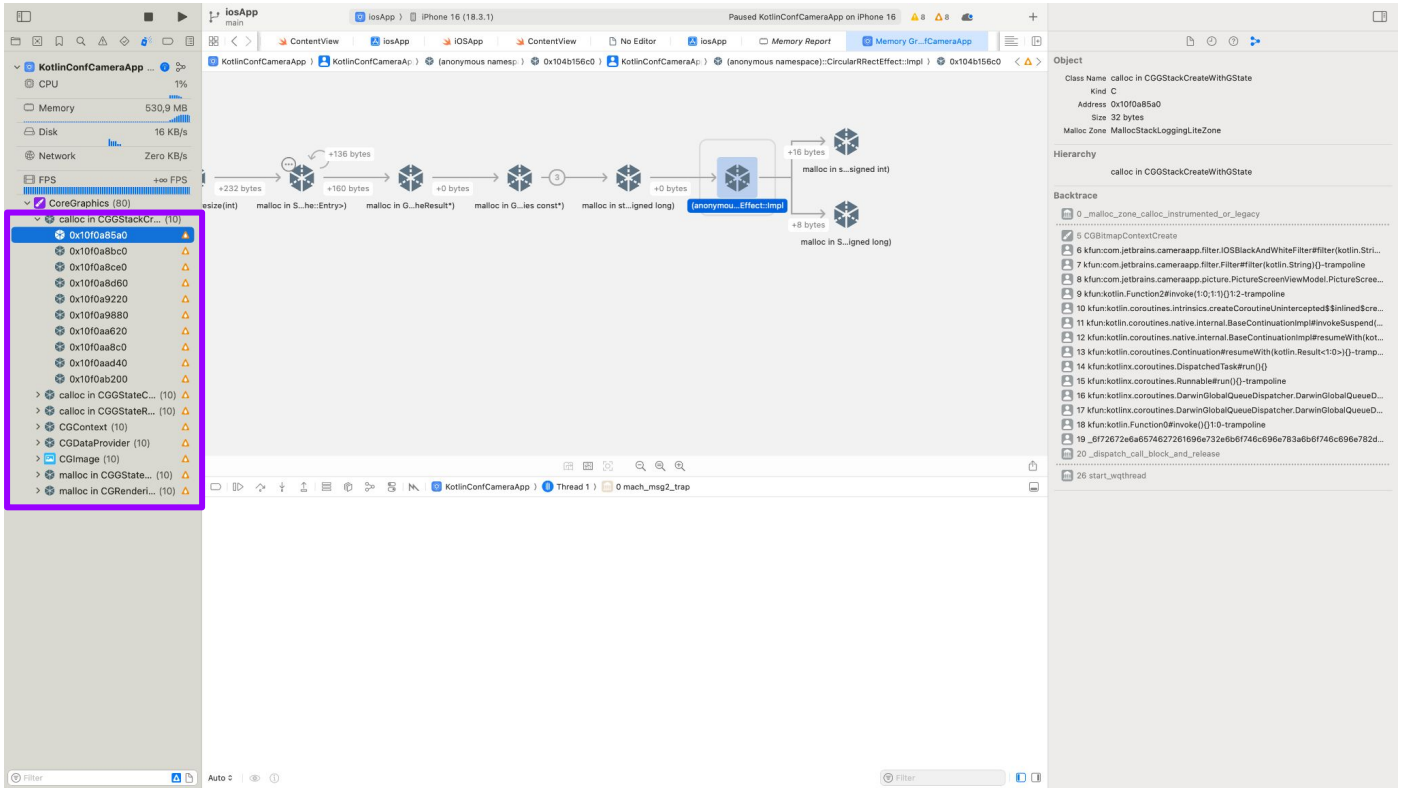
Create Debug Memory Graph



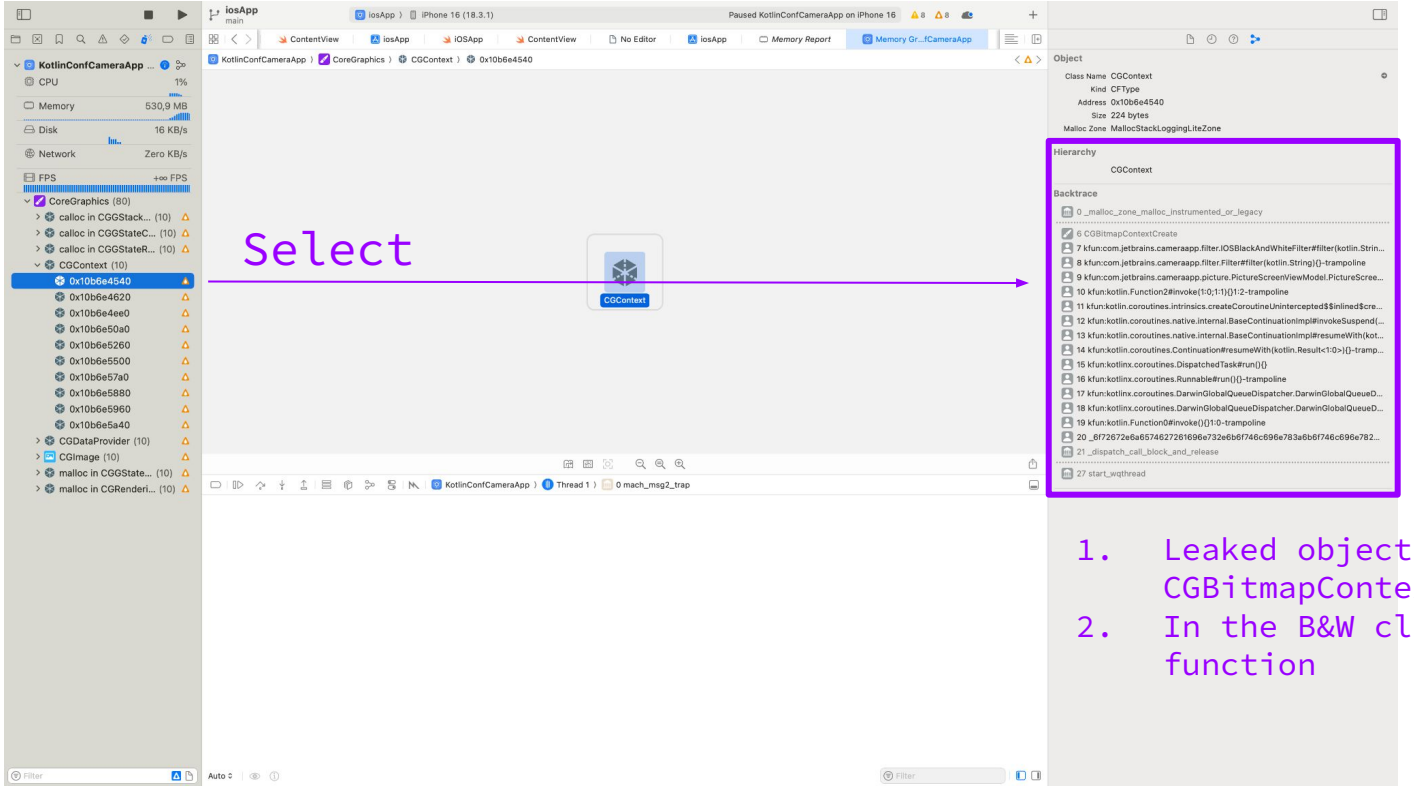
Show only leaked allocations



Select leak


KotlinConfCameraApp
CPU 1%
Memory 530,9 MB
Disk 16 KB/s
Network Zero KB/s
FPS +∞ FPS
CoreGraphics (80)
calloc in CGGStackCr... (10)
0x10f0a85a0
0x10f0a8bc0
0x10f0a8ce0
0x10f0a8d60
0x10f0a9220
0x10f0a9880
0x10f0aa620
0x10f0aa8c0
0x10f0aad40
0x10f0ab200
calloc in CGGStateC... (10)
calloc in CGGStateR... (10)
CGContext (10)
CGDataProvider (10)
CGImage (10)
malloc in CGGState... (10)
malloc in CGRenderi... (10)
+232 bytes
+136 bytes
+160 bytes
+0 bytes
+0 bytes
+16 bytes
+8 bytes
malloc in S...he::Entry>)
malloc in G...heResult*)
malloc in G...ies const*)
malloc in st...igned long)
(anonymou...Effect::Impl
malloc in s...signed int)
malloc in S...igned long)
Object
Class Name calloc in CGGStackCreateWithGState
Kind C
Address 0x10f0a85a0
Size 32 bytes
Malloc Zone MallocStackLoggingLiteZone
Hierarchy
calloc in CGGStackCreateWithGState
Backtrace

View leak backtrace



1. Leaked object type is CGBitmapContextCreate
2. In the B&W class' filter function

Fix the leak by releasing the context manually

```
@OptIn(ExperimentalForeignApi::class)
class IOSBlackAndWhiteFilter : BlackAndWhiteFilter{
    override fun filter(imagePath: String) {
        …
        val context = CGBitmapContextCreate(
            …
        )

        if (context ≠ null) {
            …
            CGContextRelease(context)
        } else {
            …
        }
    }
}
```

Reminder of the steps

1. Enable malloc debugging info on the scheme.
2. Run the application.
3. Open the debug navigator (memory section).
4. Create the debug memory graph.
5. Filter to only the leaked allocations.
6. Select the leak.
7. View leak backtrace.
8. Fix the memory leak and review!

Practical 4: Fixing a memory leak [30 mins]

Additional resources

- GC you later, Allocator - Jesse Wilson
- (Kotlin docs) Integration with Swift/Objective-C ARC
- Memory Management in Swift - Furkan Yildirim

iOS App Size Optimization

A few tricks from Kostya

- Make sure your API surface is minimal - use internal or @HiddenFromObjC keywords.
- Using a big library for a small function? Rather rewrite it.

Additional resources

- (Apple docs) Reducing your app's size
  - How to consistently, correctly measure app size.
  - Links to articles on basic/advanced strategies.
    - Asset optimization for images/video/audio

Managing Workflows

Additional resources

- Building a CI Pipeline for Kotlin Multiplatform Mobile Using GitHub Actions - Nate Ebel
- Building & Deploying a simple KMP app — Part 5: Develop CI on Github - Robert Munro
- Building & Deploying a simple KMP app — Part 6: Release CI on Github - Robert Munro
- How to publish a Kotlin Multiplatform Android app on Play Store with GitHub Actions - Marco Gomiero
- How to publish a Kotlin Multiplatform iOS app on App Store with GitHub Actions - Marco Gomiero
- How to publish a Kotlin Multiplatform macOS app on the App Store with GitHub Actions - Marco Gomiero

Discussion Time

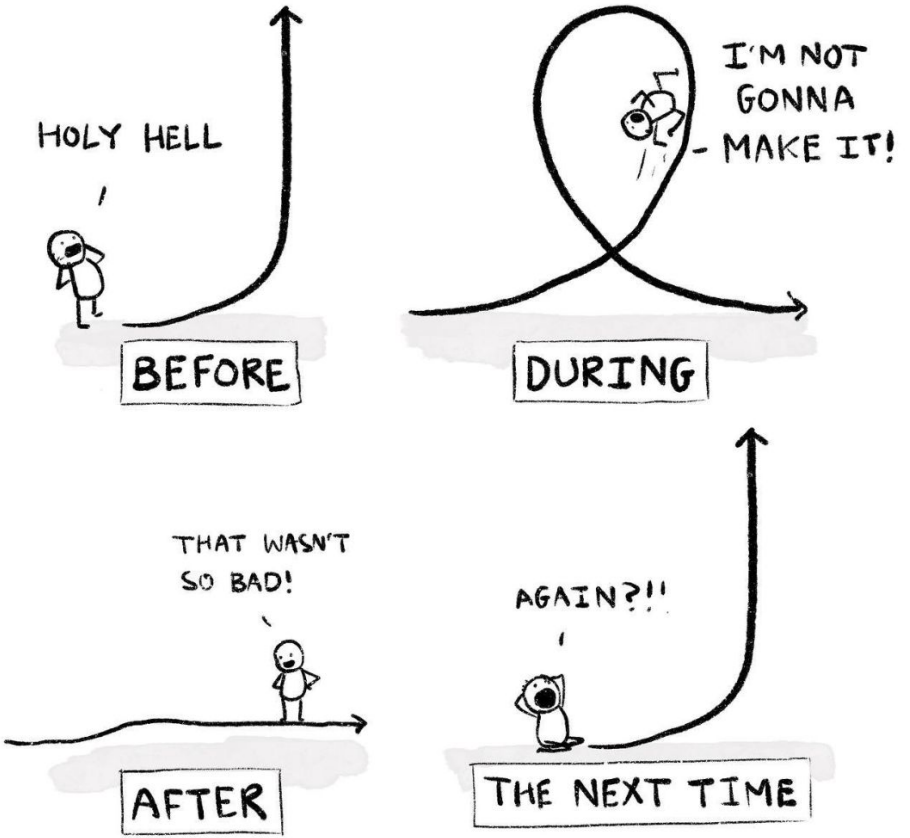
What did you like learning about the most?

What are your next steps  ?

What are you taking back to your team 🤝?

Closing

HOW LEARNING CURVES FEEL...



@STEINBERGDRAWSCARTOONS

Group picture 

Let me know if you don’t want to be photographed

Come and see my lightning ⚡ talk!

State of Kotlin/Wasm and Compose Multiplatform for Web - Pamela Hill

22 May, 10:45
Auditorium 10 (Lightning talks)

Thank you!

Stay in touch:

📧 - pamela.hill@jetbrains.com

🦋 - @pamelaahill.bsky.social

👩‍💻 - pamelaahill.com

JETBRAINS