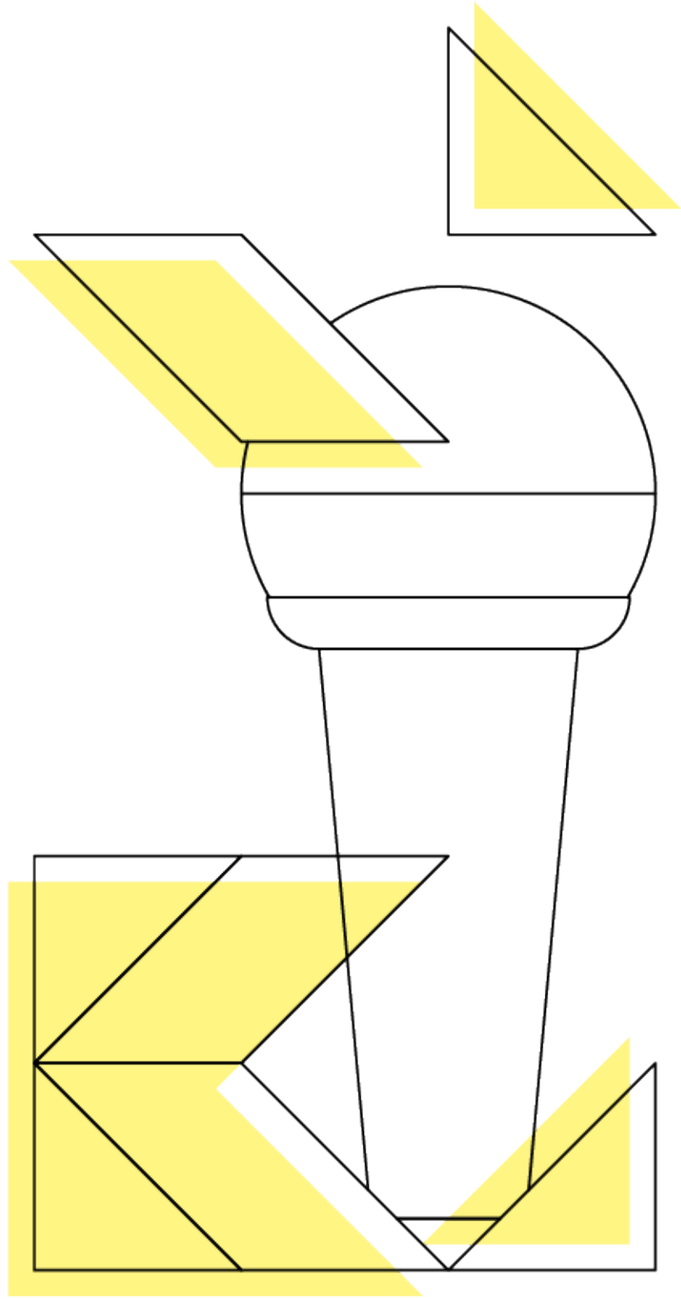
Kotlin Multiplatform

for Android/iOS devs

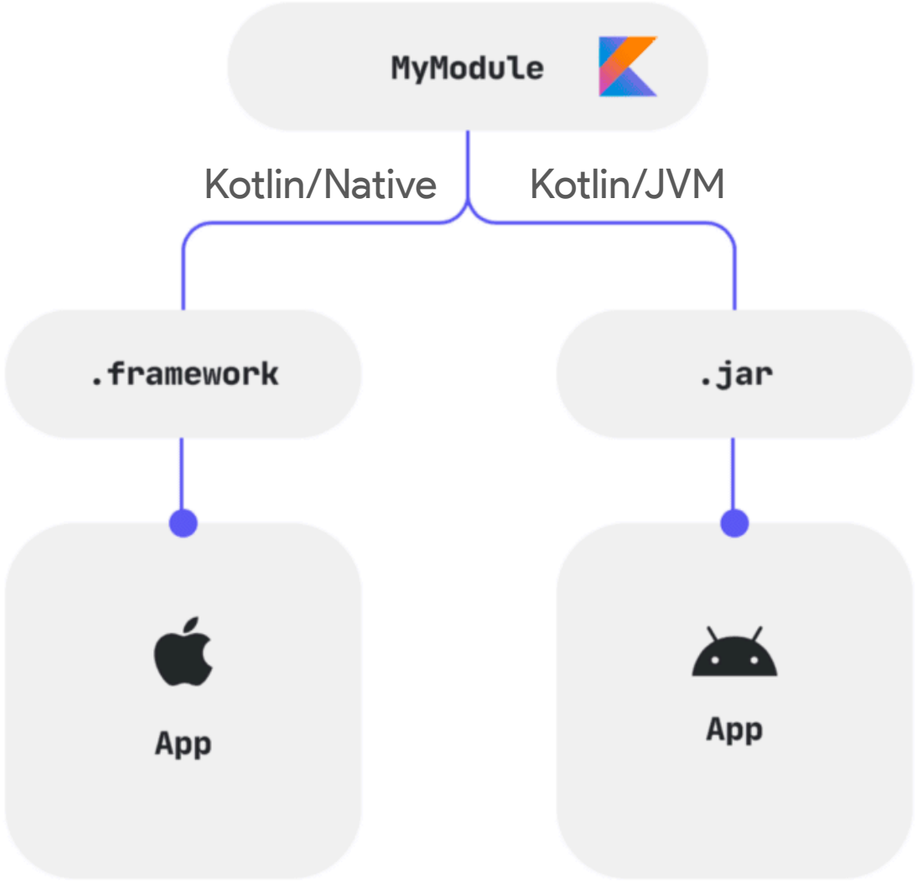
FOSDEM 2023

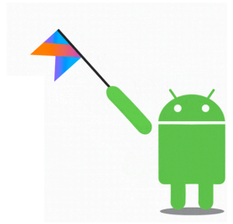
Anna Labellarte
Mobile developer @ Nextome

Paolo Rotolo
Mobile developer @ Nextome



Kotlin Multiplatform Mobile





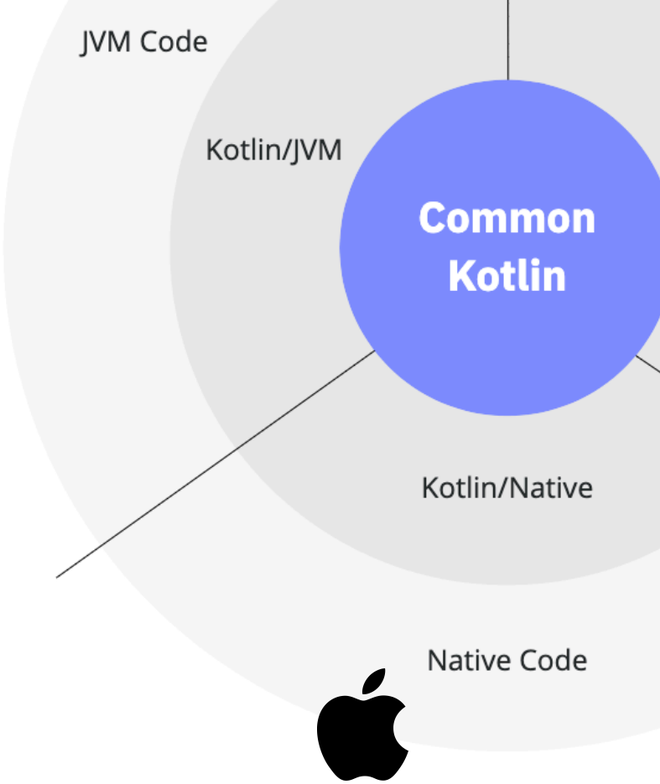


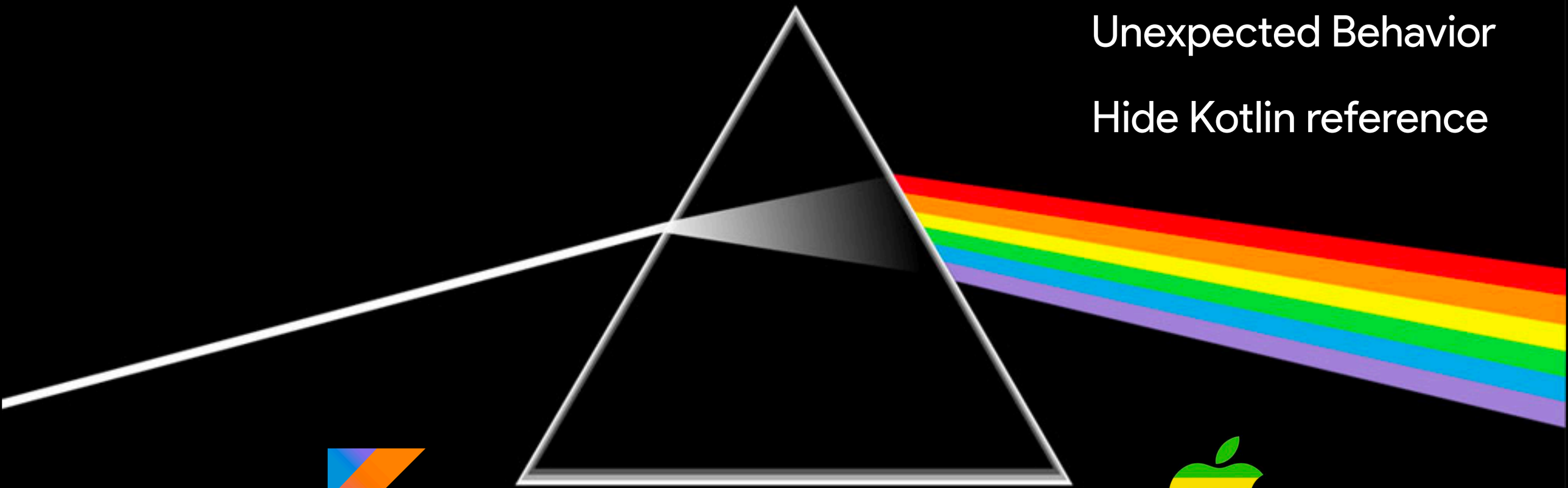
Image taken from: https://dev.to/anioutkajarkova/kotlin-multiplatform-practical-multithreading-part-1-4357

The Dark side of KMM

Swift-Friendly APIs

Unexpected Behavior

Hide Kotlin reference



Coroutines

```
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

Coroutines

```
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

```
scope.launch {
    val todo = TodoRepository().fetchTodo()
}
```

Coroutines

```kotlin
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

```swift
TodoRepository().fetchTodo {todos, error in

}
```

```swift
func loadTodo() async throws {
    let todo = try await TodoRepository().fetchTodo()
}
```

Coroutines

```
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

```
scope.launch {
    val todo = TodoRepository().fetchTodo()
}
```

Coroutines

```
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

```
scope.launch {
    val todo = TodoRepository().fetchTodo()
}

scope.cancel()
```

Koru

Inspired by
https://touchlab.co/kotlin-coroutines-rxswift/

Koru

Inspired by
https://touchlab.co/kotlin-coroutines-rxswift/

```
plugins {
    //add ksp and koru compiler plugin
    id("com.google.devtools.ksp") version "1.6.21-1.0.6"
    id("com.futuremind.koru").version("0.11.1")
}

kotlin {

    sourceSets {

        val commonMain by getting {
            dependencies {
                // add library dependency
                implementation("com.futuremind:koru:0.11.1")
            }
        }

        val iosMain by creating {
            ...
        }

    }

}

koru {
   nativeSourceSetNames = listOf("iosMain")
}
```

Coroutines

```
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

Coroutines

```
@ToNativeClass(name = "TodoRepositoryIos")
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

```
// build/generated/ksp/TodoRepositoryIos.kt

public class TodoRepositoryIos(
  private val wrapped: TodoRepository,
  private val scopeProvider: ScopeProvider?,
) {
  public constructor(wrapped: TodoRepository) :
      this(wrapped,exportedScopeProvider_mainScopeProvider)

  public fun fetchTodo(): SuspendWrapper<List<Todo>> =
     SuspendWrapper(scopeProvider, false) {
      wrapped.fetchTodo()
  }
}
```

Coroutines

```
@ToNativeClass(name = "TodoRepositoryIos")
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

Coroutines

```
@ToNativeClass(name = "TodoRepositoryIos")
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

```
let repo = TodoRepositoryIos(
    wrapped: TodoRepository(), scope: coroutineScope)

repo.getTodoWrapped().subscribe(
        onSuccess: { (array: NSArray?) -> () in

        }, onThrow: { throwable in

        })
```

Coroutines

```kotlin
@ToNativeClass(name = "TodoRepositoryIos")
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

```swift
let repo = TodoRepositoryIos(
    wrapped: TodoRepository(), scope: coroutineScope)

repo.getTodoWrapped().subscribe(
        onSuccess: { (array: NSArray?) -> () in

        }, onThrow: { throwable in

        })
```

SuspendWrapper<List<Todo>>

```kotlin
@ToNativeClass(name = "TodoRepositoryIos")
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}
```

```kotlin
@ToNativeClass(name = "TodoRepositoryIos",
    launchOnScope = MainScopeProvider::class)
class TodoRepository {
    suspend fun fetchTodo(): List<Todo> {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return todos
    }
}

@ExportedScopeProvider
class MainScopeProvider : ScopeProvider {
    override val scope : CoroutineScope = MainScope()
}
```

```kotlin
@ToNativeClass(name = "TodoRepositoryIos",
    launchOnScope = MainScopeProvider::class)
class TodoRepository {
    suspend fun fetchTodoList(): TodoList {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return TodoList(todos)
    }
}

@ExportedScopeProvider
class MainScopeProvider : ScopeProvider {
    override val scope : CoroutineScope = MainScope()
}

data class TodoList(val list: List<Todo>)
```

Coroutines

```
@ToNativeClass(name = "TodoRepositoryIos",
    launchOnScope = MainScopeProvider::class)
class TodoRepository {
    suspend fun fetchTodoList(): TodoList {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return TodoList(todos)
    }
}
```

```
let repo = TodoRepositoryIos(wrapped: TodoRepository())

repo.fetchTodoList().subscribe(onSuccess: { (list: TodoList?) in

}, onThrow: { (throwable: KotlinThrowable) in

})
```

Coroutines

```kotlin
@ToNativeClass(name = "TodoRepositoryIos",
    launchOnScope = MainScopeProvider::class)
class TodoRepository {
    suspend fun fetchTodoList(): TodoList {
        val todos = getTodoFromServer()
        saveToDb(todos)
        return TodoList(todos)
    }
}
```

```swift
let repo = TodoRepositoryIos(wrapped: TodoRepository())

let job = repo.fetchTodoList().subscribe(onSuccess: { (list: TodoList?) in

}, onThrow: { (throwable: KotlinThrowable) in

})

job.cancel(cause: KotlinCancellationException(message: "Stop it!"))
```

Flow

```
class RandomIntGenerator {
    fun generateEach(interval: Long) = flow {
        while(true) {
            delay(interval)
            emit(Random.nextInt())
        }
    }
}
```

Flow

```
class RandomIntGenerator {
    fun generateEach(interval: Long) = flow {
        while(true) {
            delay(interval)
            emit(Random.nextInt())
        }
    }
}
```

```
lifecycleScope.launch {
    RandomIntGenerator().generateEach(ONE_SECOND).collect {
        print(it)
    }
}
```

Flow

```kotlin
class RandomIntGenerator {
    fun generateEach(interval: Long) = flow {
        while(true) {
            delay(interval)
            emit(Random.nextInt())
        }
    }
}
```

```swift
RandomIntGenerator()
    .generateEach(interval: Int64(1000))
    .collect(collector: Collector()) { error in

    }


class Collector: Kotlinx_coroutines_coreFlowCollector{
    func emit(value: Any?, completionHandler: @escaping (Error?) -> Void) {
        print(value)
        completionHandler(nil)
    }
}
```

```
RandomIntGenerator()
    .generateEach(interval: Int64(1000))
    .collect(collector: Collector()) { error in

    }


class Collector: Kotlinx_coroutines_coreFlowCollector{
    func emit(value: Any?, completionHandler: @escaping (Error?) -> Void) {
        print(value)
        completionHandler(nil)
    }
}
```

```
RandomIntGenerator()
    .generateEach(interval: ONE_SECOND)
    .collect(collector: Collector<KotlinInt>() { value in
        print(value)
    }) { (error) in
        print(error?.localizedDescription)
    }


class Collector<T>: Kotlinx_coroutines_coreFlowCollector {

    let callback:(T) -> Void

    init(callback: @escaping (T) -> Void) {
        self.callback = callback
    }


    func emit(value: Any?, completionHandler: @escaping (Error?) -> Void) {
        callback(value as! T)

        completionHandler(nil)
    }
}
```

```
class RandomIntGenerator {
    fun generateEach(interval: Long) = flow {
        while(true) {
            delay(interval)
            emit(Random.nextInt())
        }
    }
}
```

```
class RandomIntGenerator {
    fun generateEach(interval: Long) = flow {
        while(true) {
            delay(interval)
            emit(Random.nextInt())
        }
    }
}


fun <T> Flow<T>.wrap(): CFlow<T> = CFlow(this)

class CFlow<T>(private val origin: Flow<T>) : Flow<T> by origin {
    fun watch(block: (T) -> Unit): Closeable {
        val job = Job()

        onEach {
            block(it)
        }.launchIn(CoroutineScope(Dispatchers.Main + job))

        return object : Closeable {
            override fun close() {
                job.cancel()
            }
        }
    }
}
```

```
class RandomIntGenerator {
    fun generateEach(interval: Long): CFlow<Int> = flow {
        while(true) {
            delay(interval)
            emit(Random.nextInt())
        }
    }.wrap()
}


fun <T> Flow<T>.wrap(): CFlow<T> = CFlow(this)

class CFlow<T>(private val origin: Flow<T>) : Flow<T> by origin {
    fun watch(block: (T) -> Unit): Closeable {
        val job = Job()

        onEach {
            block(it)
        }.launchIn(CoroutineScope(Dispatchers.Main + job))

        return object : Closeable {
            override fun close() {
                job.cancel()
            }
        }
    }
}
```

```kotlin
class RandomIntGenerator {
    fun generateEach(interval: Long): CFlow<Int> = flow {
        while(true) {
            delay(interval)
            emit(Random.nextInt())
        }
    }.wrap()
}
```

```swift
RandomIntGenerator()
    .generateEach(interval: ONE_SECOND)
    .watch { (value: KotlinInt?) in
      print(value)
    }
```

```
class RandomIntGenerator {
    fun generateEach(interval: Long): CFlow<Int> = flow {
        while(true) {
            delay(interval)
            emit(Random.nextInt())
        }
    }.wrap()
}
```

```
let disposable = RandomIntGenerator()
  .generateEach(interval: ONE_SECOND)
  .watch { (value: KotlinInt?) in
    print(value)
  }

disposable.close()
```

Exception Handling

UNCHECKED
EXCEPTIONS

CHECKED
EXCEPTIONS

Exception Handling

```
suspend fun fetchTodo(): List<Todo> {
    val todos = getTodoFromServer()
    saveToDb(todos)
    return todos
}
```

Exception Handling

```
suspend fun fetchTodo(): List<Todo> {
    val todos = getTodoFromServer()
    saveToDb(todos)
    return todos
}
```

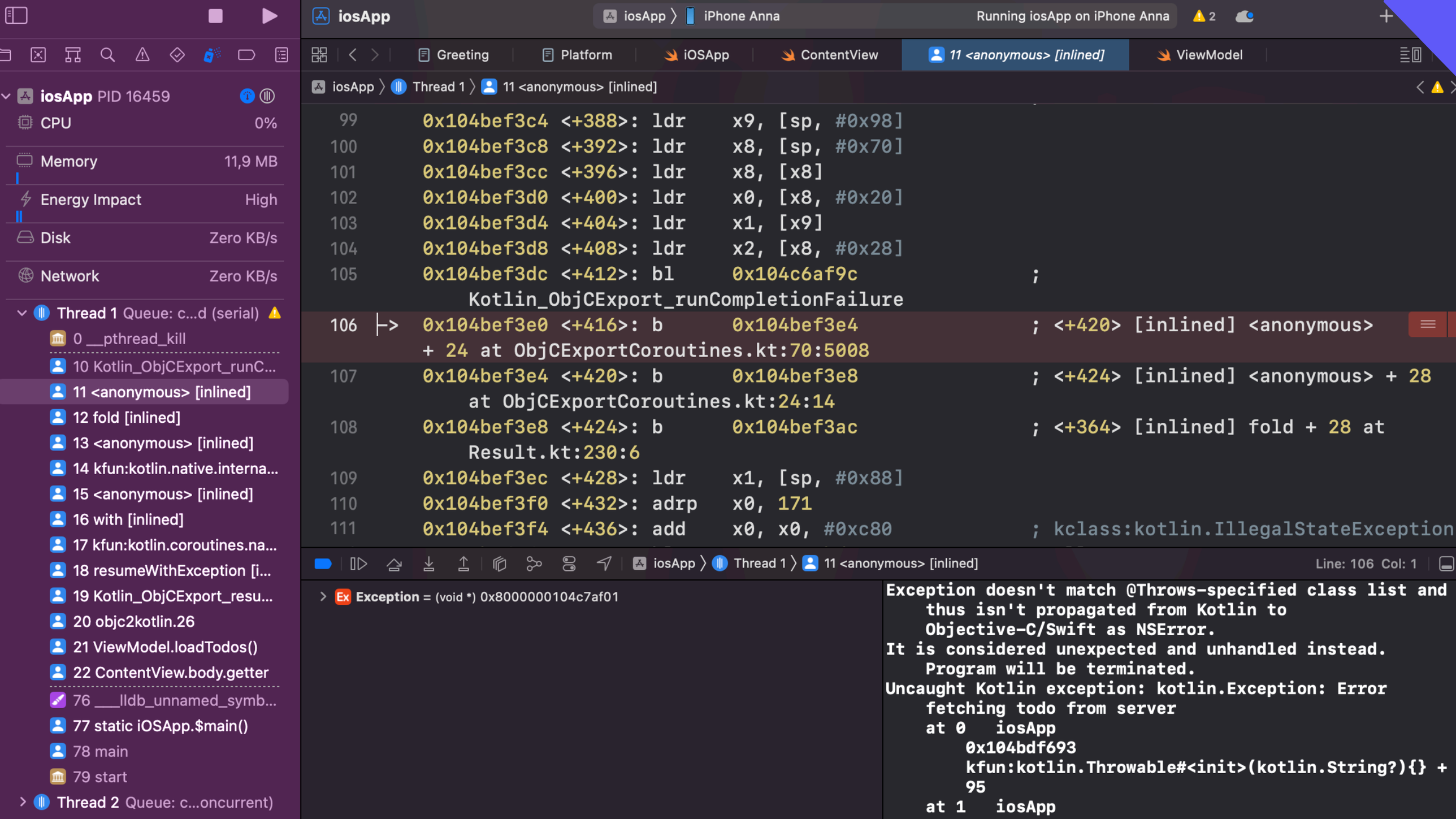
```
suspend fun loadTodos(){
    try{
        ToDoRepository().fetchTodo()
    }catch(e: Exception){
        handleTodoError(e)
    }
}
```

Exception Handling

```kotlin
suspend fun fetchTodo(): List<Todo> {
    val todos = getTodoFromServer()
    saveToDb(todos)
    return todos
}
```

```swift
func loadTodos(){
    ToDoRepository().fetchTodo{todos, error in

    }
}
```


iosApp
iosApp 〉 iPhone Anna
Running iosApp on iPhone Anna
2
Greeting
Platform
iOSApp
ContentView
11 <anonymous> [inlined]
ViewModel
iosApp 〉 Thread 1 〉 11 <anonymous> [inlined]
99 0x104bef3c4 <+388>: ldr x9, [sp, #0x98]
100 0x104bef3c8 <+392>: ldr x8, [sp, #0x70]
101 0x104bef3cc <+396>: ldr x8, [x8]
102 0x104bef3d0 <+400>: ldr x0, [x8, #0x20]
103 0x104bef3d4 <+404>: ldr x1, [x9]
104 0x104bef3d8 <+408>: ldr x2, [x8, #0x28]
105 0x104bef3dc <+412>: bl 0x104c6af9c ;
Kotlin_ObjCExport_runCompletionFailure
106 -> 0x104bef3e0 <+416>: b 0x104bef3e4 ; <+420> [inlined] <anonymous>
+ 24 at ObjCExportCoroutines.kt:70:5008
107 0x104bef3e4 <+420>: b 0x104bef3e8 ; <+424> [inlined] <anonymous> + 28
at ObjCExportCoroutines.kt:24:14
108 0x104bef3e8 <+424>: b 0x104bef3ac ; <+364> [inlined] fold + 28 at
Result.kt:230:6
109 0x104bef3ec <+428>: ldr x1, [sp, #0x88]
110 0x104bef3f0 <+432>: adrp x0, 171
111 0x104bef3f4 <+436>: add x0, x0, #0xc80 ; kclass:kotlin.IllegalStateException
iosApp 〉 Thread 1 〉 11 <anonymous> [inlined]
Line: 106 Col: 1
iosApp PID 16459
CPU 0%
Memory 11,9 MB
Energy Impact High
Disk Zero KB/s
Network Zero KB/s
Thread 1 Queue: c...d (serial)
0 __pthread_kill
10 Kotlin_ObjCExport_runC...
11 <anonymous> [inlined]
12 fold [inlined]
13 <anonymous> [inlined]
14 kfun:kotlin.native.interna...
15 <anonymous> [inlined]
16 with [inlined]
17 kfun:kotlin.coroutines.na...
18 resumeWithException [i...
19 Kotlin_ObjCExport_resu...
20 objc2kotlin.26
21 ViewModel.loadTodos()
22 ContentView.body.getter
76 ___lldb_unnamed_symb...
77 static iOSApp.$main()
78 main
79 start
Thread 2 Queue: c...oncurrent)
Exception = (void *) 0x8000000104c7af01
Exception doesn't match @Throws-specified class list and
thus isn't propagated from Kotlin to
Objective-C/Swift as NSError.
It is considered unexpected and unhandled instead.
Program will be terminated.
Uncaught Kotlin exception: kotlin.Exception: Error
fetching todo from server
at 0 iosApp
0x104bdf693
kfun:kotlin.Throwable#<init>(kotlin.String?){} +
95
at 1 iosApp

Exception Handling

```kotlin
@Throws(Exception::class)
suspend fun fetchTodo(): List<Todo> {
    val todos = getTodoFromServer()
    saveToDb(todos)
    return todos
}
```

```swift
func loadTodos(){
    ToDoRepository().fetchTodo{todos, error in

    }
}
```

Error Handling

```kotlin
@Throws(Exception::class)
fun enterTheDungeon(name: String): String{
    if(name == "Anna"){
        throw Exception("You shall not pass!")
    }
    return "Hello $name!"
}
```

```swift
func enter(){
    do{
        try manager.enterTheDungeon(name: "Anna")
    }catch{
        print(error.localizedDescription)
    }
}
```

Sealed Class

Sealed Class

```
sealed interface UIState {
    object Loading : UIState
    data class Data(val value: List<Todo>) : UIState
    data class Error(val throwable: Throwable) : UIState
}
```

Sealed Class

```
sealed interface UIState {
    object Loading : UIState
    data class Data(val value: List<Todo>) : UIState
    data class Error(val throwable: Throwable) : UIState
}
```

```
when (uiState) {
    is UIState.Data → TODO()
    is UIState.Error → TODO()
    is UIState.Loading → TODO()
}
```

Sealed Class

```
sealed interface UIState {
    object Loading : UIState
    data class Data(val value: List<Todo>) : UIState
    data class Error(val throwable: Throwable) : UIState
}
```

```
ToDoRepository().fetchTodoEnum{uiStatus, error in
    if let status = uiStatus{
        switch status{
            case is UIStateLoading:
                print("loading")
            case let error as UIStateError:
                print("Error: \(error)")
            case let success as UIStateData:
                let count = success.value.count
                print("Retrieved \(count) todos")
            default:
                print("Never called")
        }
    }
}
```

Sealed Class

```
sealed interface UIState {
    object Loading : UIState
    data class Data(val value: List<Todo>) : UIState
    data class Error(val throwable: Throwable) : UIState
}
```

Enum

```
enum UIState{
    case Loading, Error(Error), Success([Todo])
}
```

KSwift

Sealed Class

```
sealed interface UIState {
    object Loading : UIState
    data class Data(val value: List<Todo>) : UIState
    data class Error(val throwable: Throwable) : UIState
}
```

```
ToDoRepository().fetchTodoEnum{uiStatus, error in
    if let status = uiStatus{
        let statusKs = UIStateKs(status)
        switch (statusKs){
        case .loading:
            print("Loading...")
        case .error(let error):
            print(error.throwable.message ?? "")
        case .data(let data):
            let count = data.value.count
            print("Retrieved \(count) todo")

        }
    }
}
```

Context

```
MyLibrary.getInstance(
    applicationContext)
```

Context

```
val myLib = MyLibrary.getInstance(
   applicationContext)


myLib.doSomething()
```

```
let myLib = MyLibrary.getInstance()


myLib.doSomething()
```

App Startup

```
internal lateinit var applicationContext: Context
    private set

public object MyLibrary

class MyLibraryInitializer: Initializer<MyLibrary> {
    override fun create(context: Context): MyLibrary {
        applicationContext = context.applicationContext
        return MyLibrary
    }

    override fun dependencies(): List<Class<out Initializer<*>>> {
        return listOf()
    }

}
```

Context

```
val myLib = MyLibrary.getInstance(
   applicationContext)

myLib.doSomething()
```

```
let myLib = MyLibrary.getInstance()

myLib.doSomething()
```

Context

```
val myLib = MyLibrary.getInstance()

myLib.doSomething()
```

```
let myLib = MyLibrary.getInstance()

myLib.doSomething()
```

1.8.0

Improved ObjC-Swift Interoperability

@ObjCName

```
package com.nextome.lib.dto


class Todo {
    val name: String = ""
    val id: Long = -1
    val date: Long = -1
}
```

```
package com.nextome.lib.entity


class Todo {
    val name: String = ""
    val deadLineDate: String = ""
}
```

@ObjCName

```kotlin
package com.nextome.lib.dto


class Todo {
    val name: String = ""
    val id: Long = -1
    val date: Long = -1
}
```

```kotlin
package com.nextome.lib.entity

@ObjCName(“TodoEntity")
class Todo {
    val name: String = ""
    val deadLineDate: String = ""
}
```

```swift
let entity = Todo()
entity.name

let dto = Todo_()
dto.deadLineDate
```

@ObjCName

```
package com.nextome.lib.dto


class Todo {
    val name: String = ""
    val id: Long = -1
    val date: Long = -1
}
```

```
package com.nextome.lib.entity

@ObjCName(“TodoEntity")
class Todo {
    val name: String = ""
    val deadLineDate: String = ""
}
```

```
let entity = Todo()
entity.name

let dto = TodoEntity()
dto.deadLineDate
```

@ObjCName

```
fun deleteTodosForUser(userId: String){
    TODO()
}
```

@ObjCName

```
fun deleteTodosForUser(userId: String){
    TODO()
}
```

```
func deleteTodosForUser(userId: String){

}
```

@ObjCName

```
fun deleteTodosForUser(userId: String){
    TODO()
}
```

```
func deleteTodos(forUser userId: String){

}
```

@ObjCName

```
@ObjCName("deleteTodos")
 fun deleteTodosForUser(@ObjCName("forUser") userId: String){
    TODO()
 }
```

```
func deleteTodos(forUser  userId: String){

}
```

@HiddenFromObjC

@ShouldRefineInSwift

@ShouldRefineInSwift

```
fun insertNewTodo(
    title: String,
    priority: Int?
)
```

@ShouldRefineInSwift

```
@ShouldRefineInSwift
fun insertNewTodo(
    title: String,
    priority: Int?
)
```

```
repository.insertNewTodo(
                title: String,
                priority:KotlinInt?)
```

⚠️

@ShouldRefineInSwift

```
func insertNewTodo (title: String, priority: Int?){

    __ insertNewTodo(title: String , priority: KotlinInt? )

}
```

⚠️

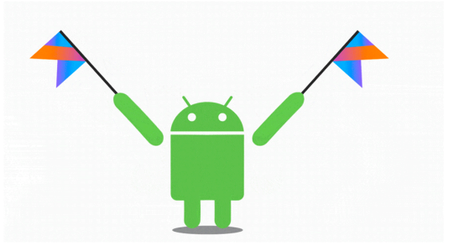
@ShouldRefineInSwift

```
func insertNewTodo (title: String, priority: Int?){

    __ insertNewTodo(title: title  , priority: priority.toKotlinInt() )


}
```

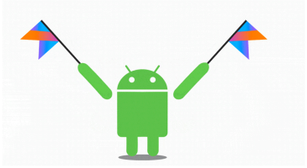
What about the future



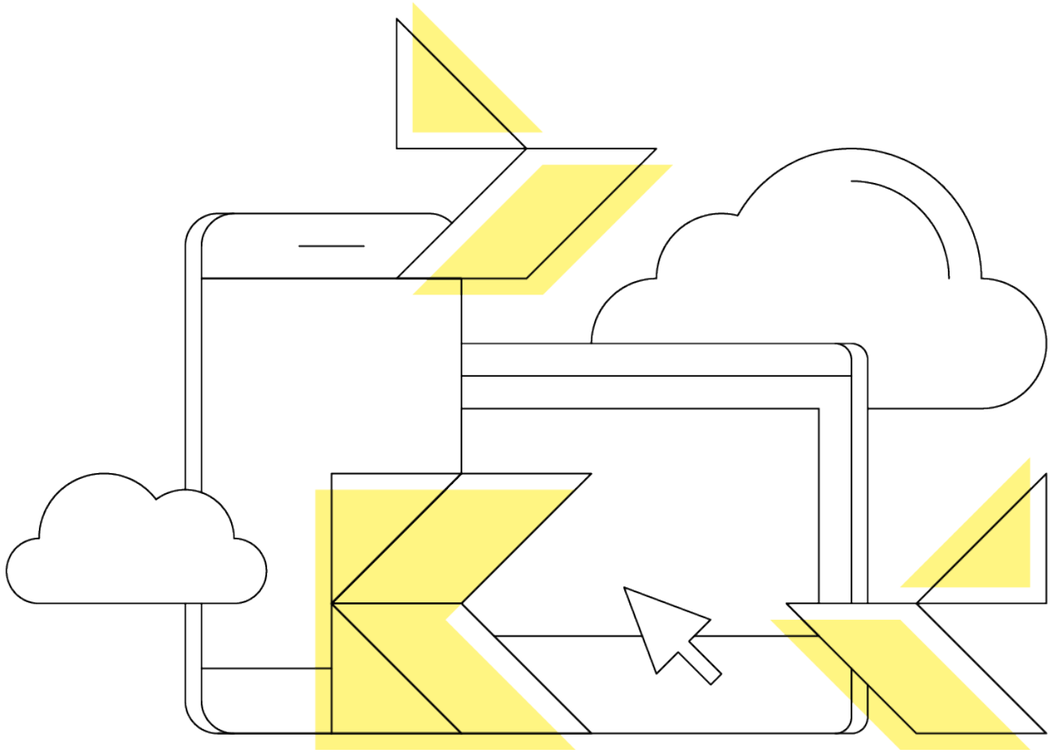
Kotlin roadmap

Edit page Last modified: 01 February 2023

- 🆕 Promote Kotlin Multiplatform Mobile to Stable ↗
- Improve exporting Kotlin code to Objective-C ↗
- Improve Kotlin/Native compilation time ↗
- 🆕 Provide a Kotlin API guide for libraries authors ↗



Questions?



Thanks!