Дзен

Android driven KMM

О том, как мы внедряли KMM с минимальным участием со стороны iOS

Сёмочкин Константин

План доклада

- Зачем нужен KMM
- Почему Android driven
- Выбор задач
- Экспертиза iOS
- Конфигурация проекта
- Модуляризация
- Бесшовный опыт для iOS
- Мультиплатформенные тесты
- Итоги

Зачем нужен KMM ?

Преимущества

- Пишем код 1 раз
- Пишем на Котлине
- «Хороший» код*

Дзен

Почему Android driven

Дзен:

- Est. 2015
- 60 миллионов пользователей
- 60+ мобильных разработчиков

Когда внедрять KMM

- В октябре 2022 года KMM выходит в бету
- JetBrains официально заявляет, что внедрять KMM безопасно

Отношение к KMM

75% iOS против

- Новая технология
- Overhead

65% Android за

- Новая технология
- MacOS

Почему Android driven?

- Скептическое отношение к технологии со сторноы iOS
- Запаздывание iOS по некоторым направлениям

Выбор задач

Скептическое отношение
к технологии со стороны iOS

Выбираем задачи на
чистую бизнес логику

Экспертиза iOS

Ищем Android разработчика,
который разбирается в iOS

Не находим

Создаем

Некоторые особенности разработки

Kotlin

- Байт код
- Пакеты
- GC

Swift

- Нативный код
- Единое пространство имен
- ARC

Конфигурация проекта

- Доставка кода
  - Android (Gradle): модуль, библиотека
  - IOS (SPM, CocoaPods): Framework (aka библиотека)
- Варианты репозиториев
  - Shared + Android + iOS
  - Shared + Android
  - Shared

Хостимся в Android repo

- Android driven
- Легкая ингетрация в инфру

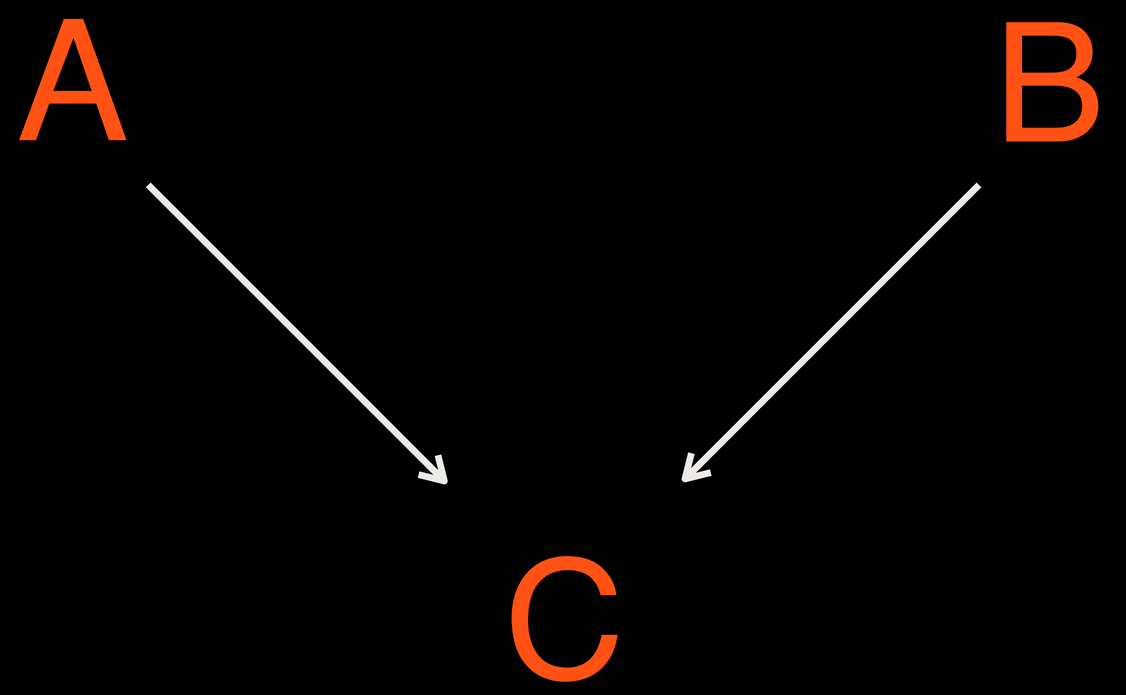
Дзен

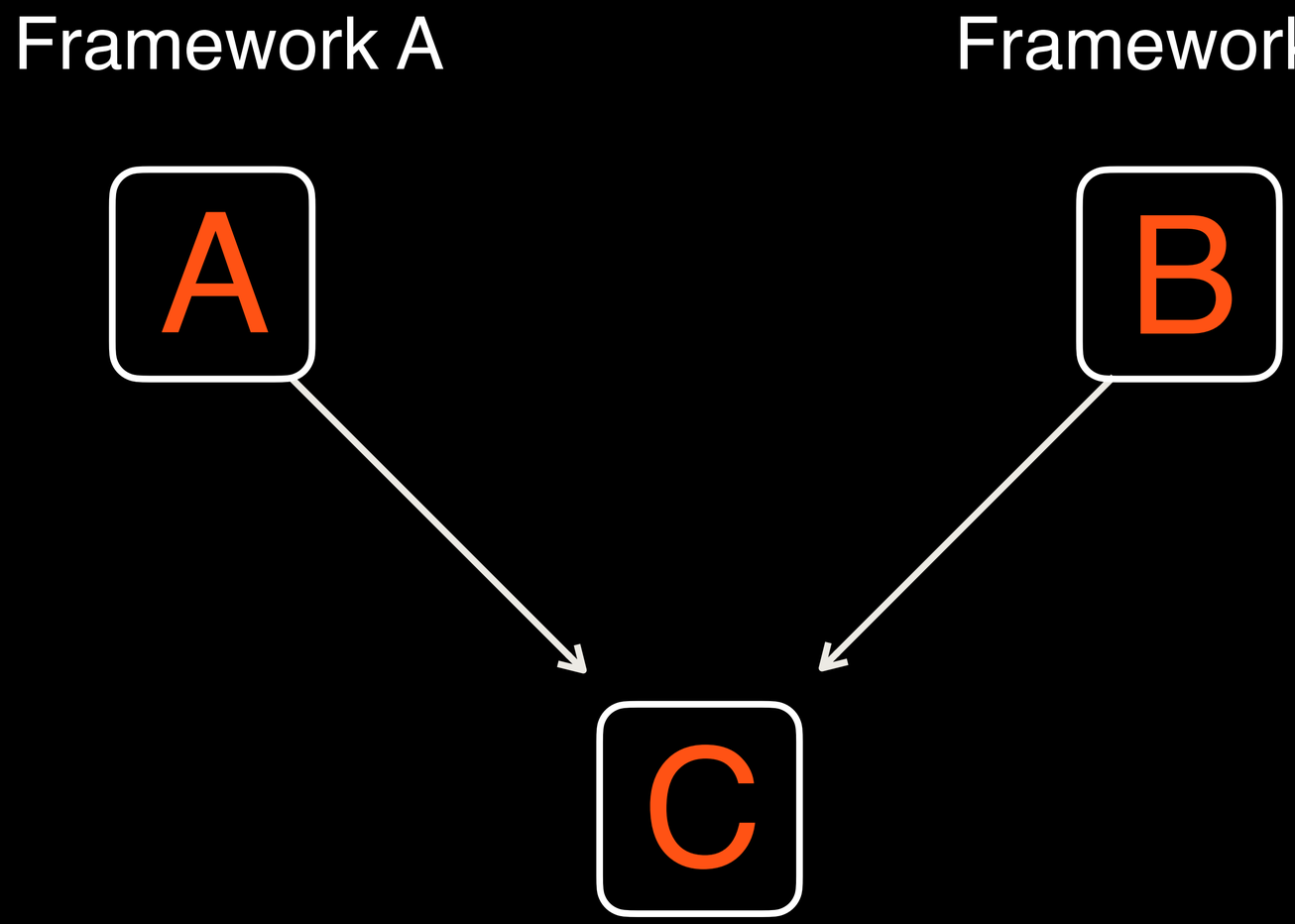
Модуляризация

Пространство имен при экспорте в Framework

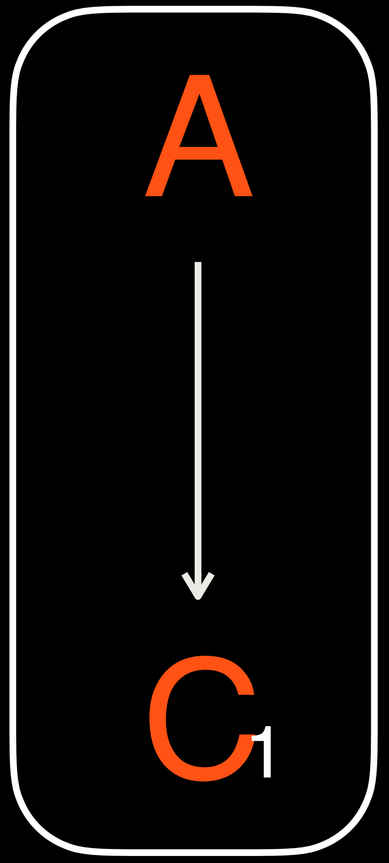
Kotlin		Swift
com.sample.package1.SampleClass	→	SampleClass
com.sample.package2.SampleClass		SampleClass_

Организация в Kotlin коде

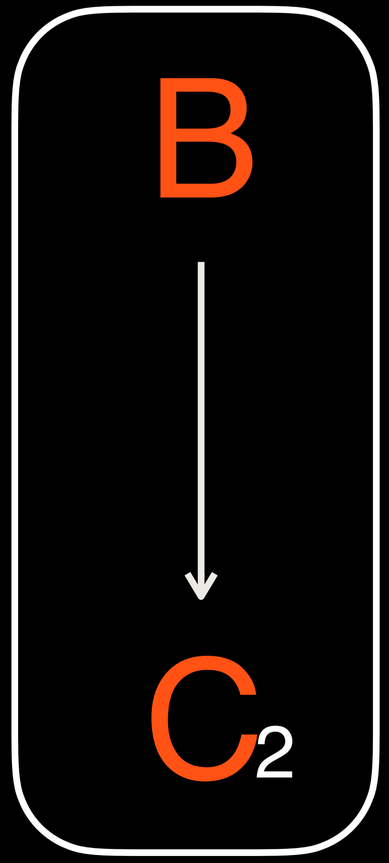




Framework A



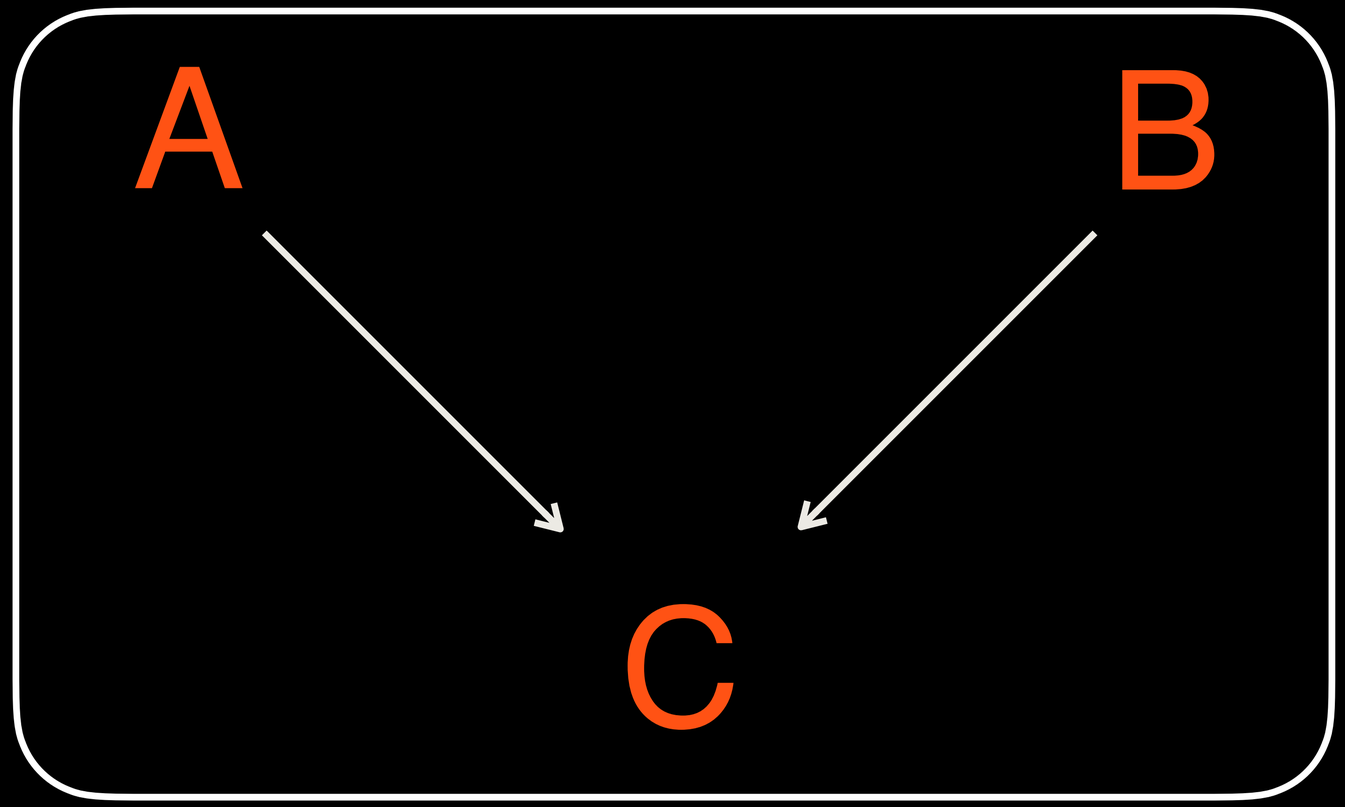
Framework B



Дзен

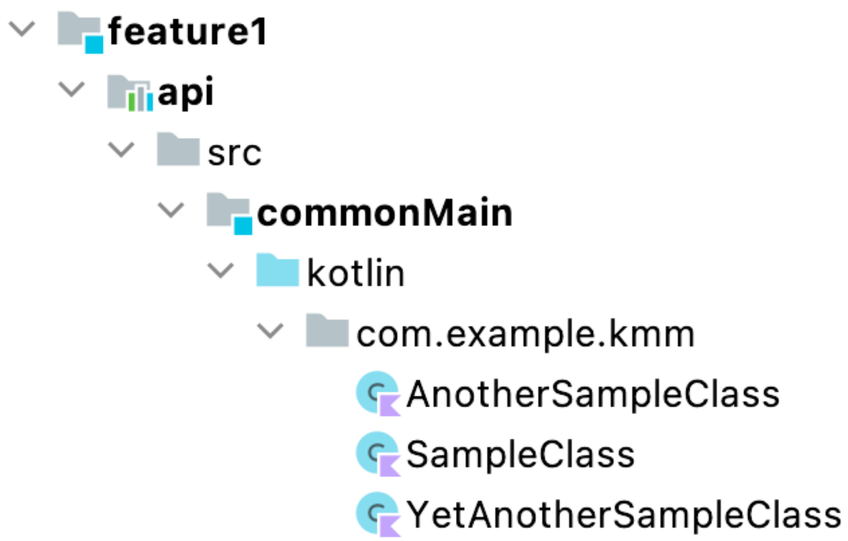
Используем единый артефакт

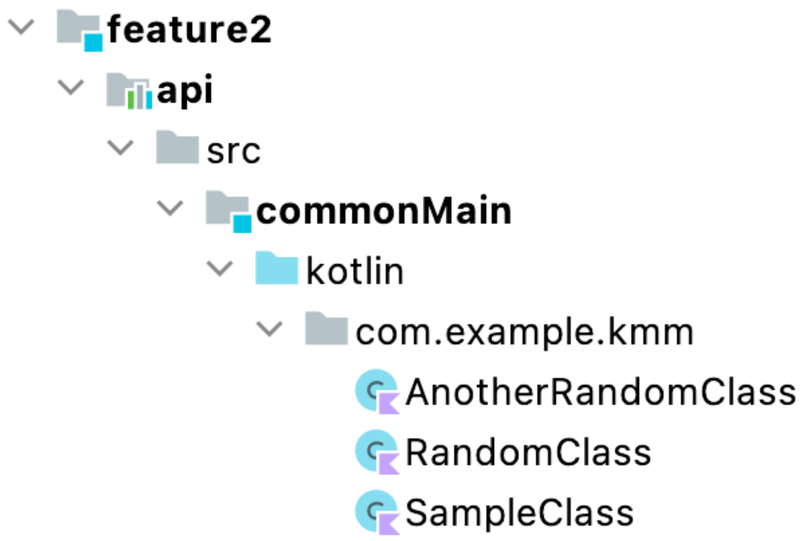
Umbrella Framework



Дзен

Решение: Единый пакет для сущностей



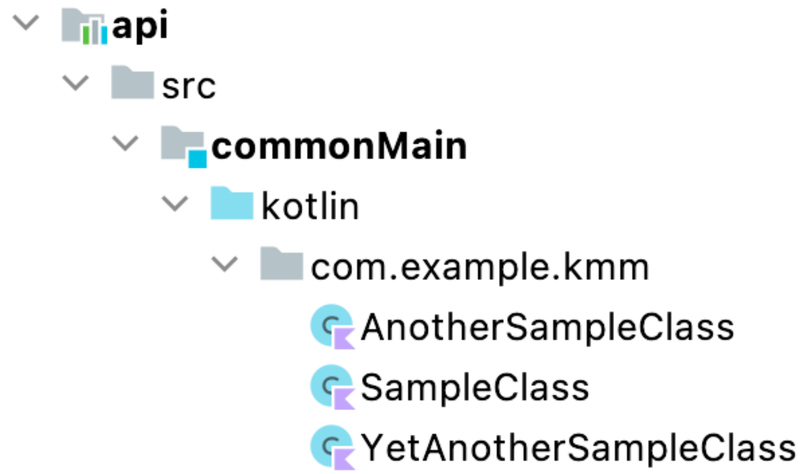


При одинаковых названиях

```
> Task :androidApp:mergeLibDexDebug FAILED
AGPBI: {"kind":"error","text":"Type com.example.kmm.SampleClass is defined multiple times: /Users/forcharc/Androi
com.android.builder.dexing.DexArchiveMergerException: Error while merging dex archives:
Learn how to resolve the issue at https://developer.android.com/studio/build/dependencies#duplicate_classes.
Type com.example.kmm.SampleClass is defined multiple times: /Users/forcharc/AndroidStudioProjects/TestKmmApplicat
    at com.android.builder.dexing.D8DexArchiveMerger.getExceptionToRethrow(D8DexArchiveMerger.java:151)
    at com.android.builder.dexing.D8DexArchiveMerger.mergeDexArchives(D8DexArchiveMerger.java:138)
    at com.android.build.gradle.internal.tasks.DexMergingWorkAction.merge(DexMergingTask.kt:859)
    at com.android.build.gradle.internal.tasks.DexMergingWorkAction.run(DexMergingTask.kt:805)
    at com.android.build.gradle.internal.profile.ProfileAwareWorkAction.execute(ProfileAwareWorkAction.kt:74)
    at org.gradle.workers.internal.DefaultWorkerServer.execute(DefaultWorkerServer.java:63)
    at org.gradle.workers.internal.NoIsolationWorkerFactory$1$1.create(NoIsolationWorkerFactory.java:66)
    at org.gradle.workers.internal.NoIsolationWorkerFactory$1$1.create(NoIsolationWorkerFactory.java:62)
    at org.gradle.internal.classloader.ClassLoaderUtils.executeInClassloader(ClassLoaderUtils.java:100)
    at org.gradle.workers.internal.NoIsolationWorkerFactory$1.lambda$execute$0(NoIsolationWorkerFactory.java:62)
    at org.gradle.workers.internal.AbstractWorker$1.call(AbstractWorker.java:44)
```

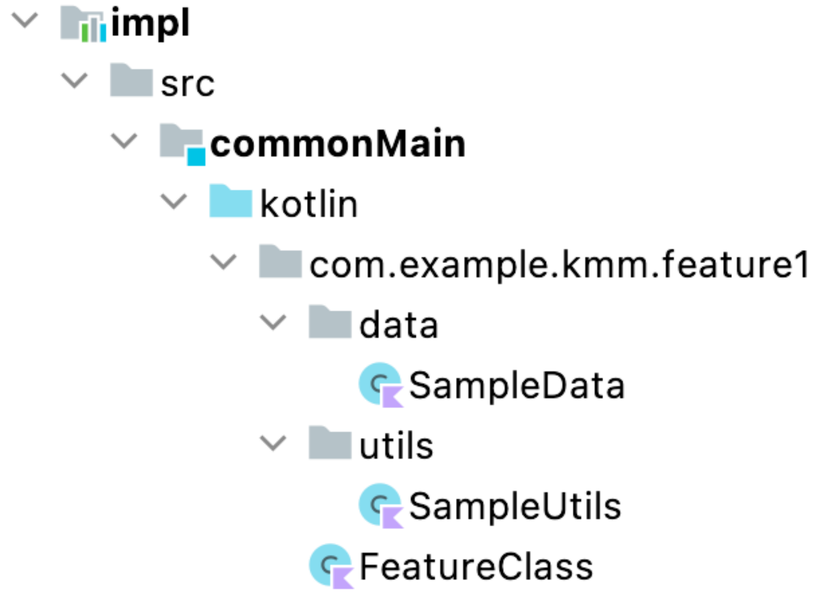
Api модуль

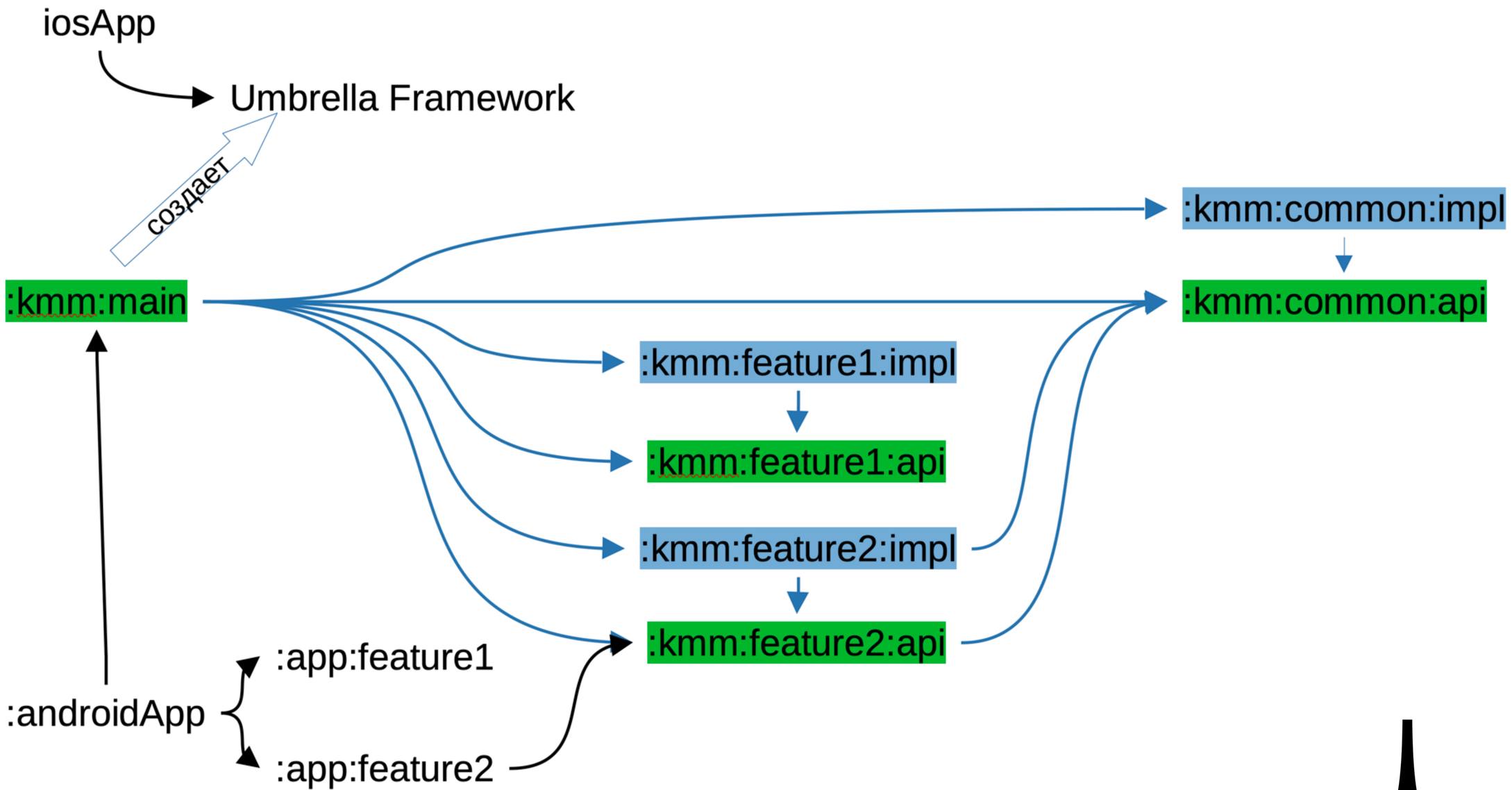
- Доступен извне
- Сущности в едином пространстве имен
- Зависит только от других api модулей

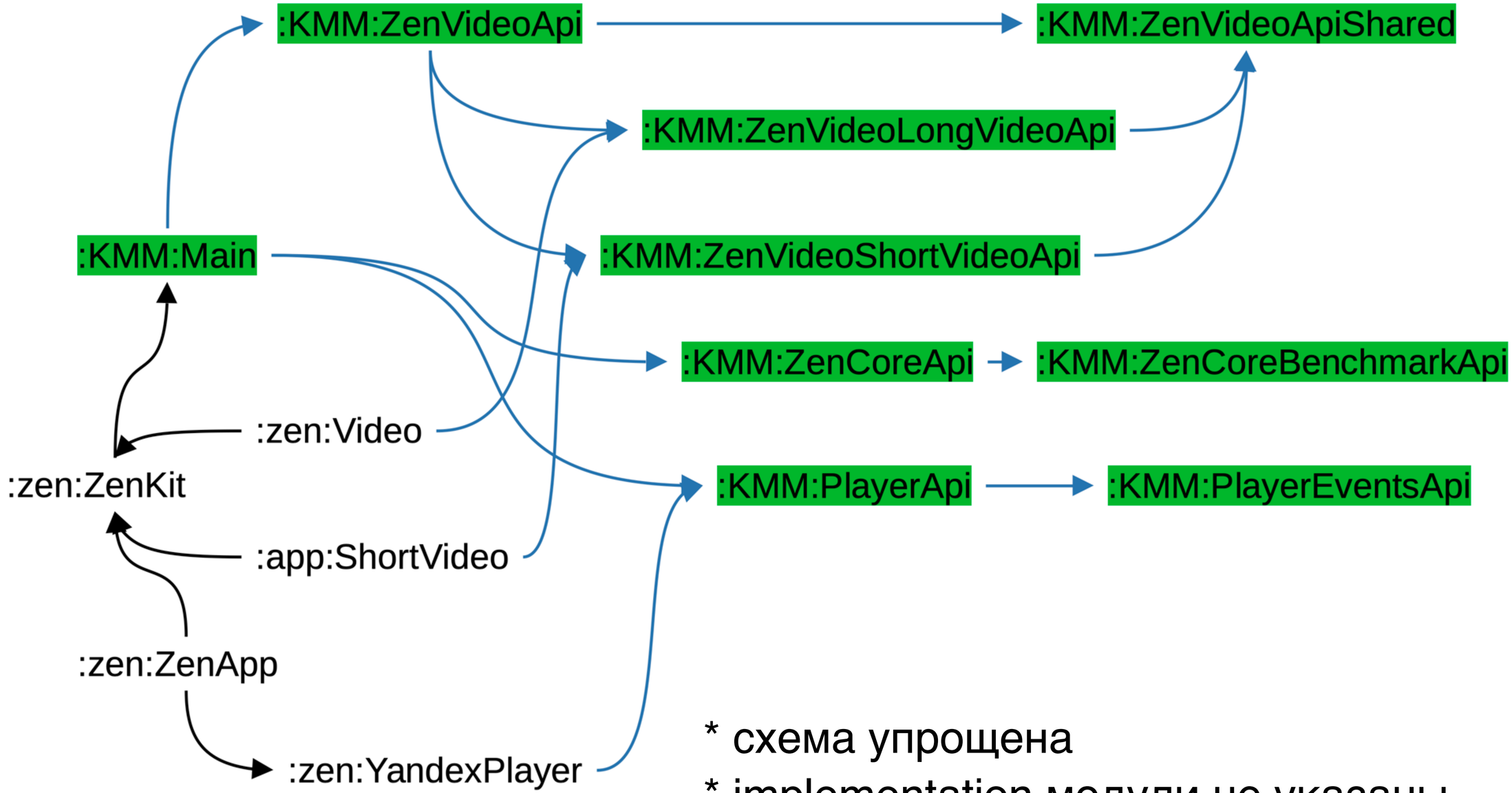


Implementation модуль

- Скрыт внутри библиотеки
- Спокойно пользуемся пакетами
- Может зависить от api модулей




iosApp
Umbrella Framework
создает
:kmm:main
:kmm:common:impl
:kmm:common:api
:kmm:feature1:impl
:kmm:feature1:api
:kmm:feature2:impl
:kmm:feature2:api
:androidApp
:app:feature1
:app:feature2



* схема упрощена
* implementation модули не указаны

Дзен

build.gradle.kts (:kmm:main)

```
kotlin { this: KotlinMultiplatformExtension
    android()
    val frameworkName = "KMM"
    val umbrellaFramework = XCFramework(frameworkName)
    listOf(
        iosX64(),
        iosArm64(),
        iosSimulatorArm64()
    ).forEach { it: KotlinNativeTarget
        it.binaries.framework { this: Framework
            baseName = frameworkName
            export(project( path: ":kmm:common:api"))
            export(project( path: ":kmm:features:feature1:api"))
            export(project( path: ":kmm:features:feature2:api"))
            umbrellaFramework.add(this)
        }
    }

    sourceSets {...}
}
```

```
sourceSets { this: NamedDomainObjectContainer<KotlinSourceSet>
    val commonMain by getting { this: KotlinSourceSet!
        dependencies { this: KotlinDependencyHandler
            implementation(project( path: ":kmm:common:impl"))
            implementation(project( path: ":kmm:features:feature1:impl"))
            implementation(project( path: ":kmm:features:feature2:impl"))
            api(project( path: ":kmm:common:api"))
            api(project( path: ":kmm:features:feature1:api"))
            api(project( path: ":kmm:features:feature2:api"))
        }
    }
    val iosX64Main by getting
    val iosArm64Main by getting
    val iosSimulatorArm64Main by getting
    val iosMain by creating {...}
}
```

Итог

- Решили проблему с коллизиями
- Создали удобную иерархию кода
- Ускорили сборку приложения

Дзен

Бесшовный опыт для iOS

Документация

Подробная документация Api модулей

```
/**
 * Бенчмарк старта приложения. По полученным событиям измеряет:
 * 1. Время до готовности контента
 * 2. Время до отрисовки первого кадра
 * */
interface StartupBenchmark {

    /**
     * Событие готовности контента. Отправляется сразу когда данные готовы к показу.
     * Это значит что данные получены(из кеша или сети) + распрашены + преобразованы в конечные
     * модели которые будут использоваться в UI
     *
     * type - имя экрана на котором это произошло.
     * */
    fun onContentReady(type: StartupBenchmarkContentType)

    /**
     * События показа контента. Отправляется когда в системе произошло
     * первое событие show(см. StatEvents)
     * */
    fun onContentShow()
}
```

Экспорт документации в Obj-C

Используем аргумент компилятора -Xexport-kdoc

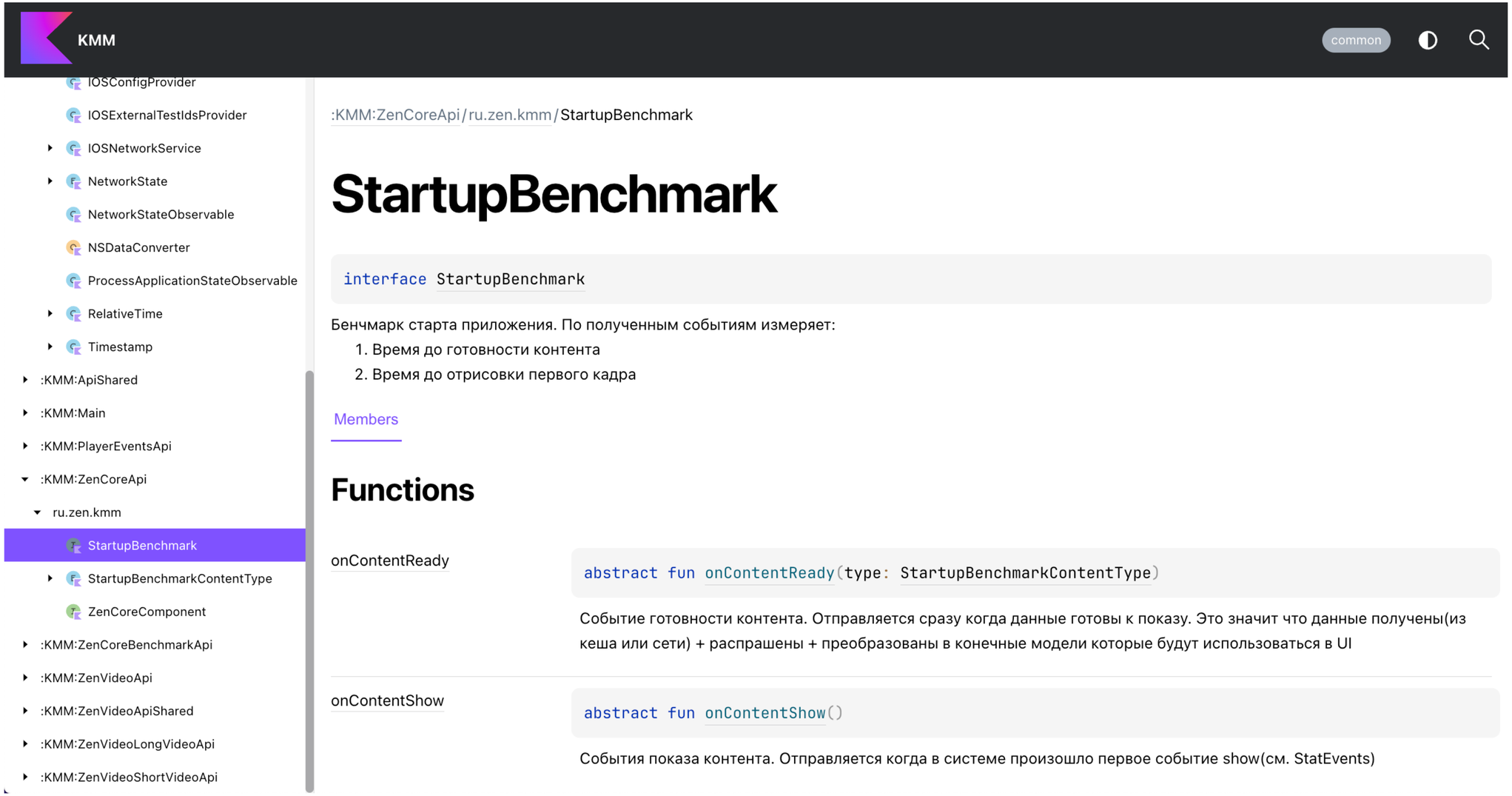
```
kotlin { this: KotlinMultiplatformExtension
    targets.withType<org.jetbrains.kotlin.gradle.plugin.mpp.KotlinNativeTarget> { this: KotlinNativeTarget
        compilations["main"].compilerOptions.options.freeCompilerArgs.add("-Xexport-kdoc")
    }
}
```

В итоге комментарии прорастают в header-ы Obj-C

```
/**
 * Бенчмарк старта приложения. По полученным событиям измеряет:
 * 1. Время до готовности контента
 * 2. Время до отрисовки первого кадра
 * */
__attribute__((swift_name("StartupBenchmark")))
@protocol KMMStartupBenchmark
@required

/**
 * Событие готовности контента. Отправляется сразу когда данные готовы к показу.
 * Это значит что данные получены(из кеша или сети) + распрашены + преобразованы в конечные
 * модели которые будут использоваться в UI
 * */
- (void)onContentReadyType:(KMMStartupBenchmarkContentType *)type __attribute__((swift_name("onContentReady(type:)")));
```

Dokka


KMM
common
IOSConfigProvider
IOSExternalTestIdsProvider
IOSNetworkService
NetworkState
NetworkStateObservable
NSDataConverter
ProcessApplicationStateObservable
RelativeTime
Timestamp
:KMM:ApiShared
:KMM:Main
:KMM:PlayerEventsApi
:KMM:ZenCoreApi
ru.zen.kmm
StartupBenchmark
StartupBenchmarkContentType
ZenCoreComponent
:KMM:ZenCoreBenchmarkApi
:KMM:ZenVideoApi
:KMM:ZenVideoApiShared
:KMM:ZenVideoLongVideoApi
:KMM:ZenVideoShortVideoApi
:KMM:ZenCoreApi/ru.zen.kmm/StartupBenchmark
StartupBenchmark
interface StartupBenchmark
Бенчмарк старта приложения. По полученным событиям измеряет:
1. Время до готовности контента
2. Время до отрисовки первого кадра
Members
Functions
onContentReady
abstract fun onContentReady(type: StartupBenchmarkContentType)
Событие готовности контента. Отправляется сразу когда данные готовы к показу. Это значит что данные получены(из кеша или сети) + распрашены + преобразованы в конечные модели которые будут использоваться в UI
onContentShow
abstract fun onContentShow()
События показа контента. Отправляется когда в системе произошло первое событие show(см. StatEvents)

Дзен

Бесшовный опыт для iOS

- Как можно больше логов
- @HiddenFromObjC и @ObjCName

Алиасы для KMM кода

- Не ипортируем KMM библиотоку из IOS кода напрямую

  ~~**import KMM**~~

- Используем для этого swift Alias’ы, которые лежат в отдельном файле

  public typealias SimpleClass = KMM.SimpleClass

  - Не засоряем Autocomplete
  - Легко уйти с KMM релизации

Дзен

Мультиплатформенные тесты

Положение дел

- Android driven KMM
- Задачи на чистую бизнес логику
- Отсутствие пользовательского интерфейса
- Острая необходимость писать тесты

Способы запуска тестов

Android

Локальные тесты → JVM

Инструментальные тесты → Эмулятор или реальное устройство

IOS

Локальные тесты

Инструментальные тесты

} Симулятор или реальное устройство

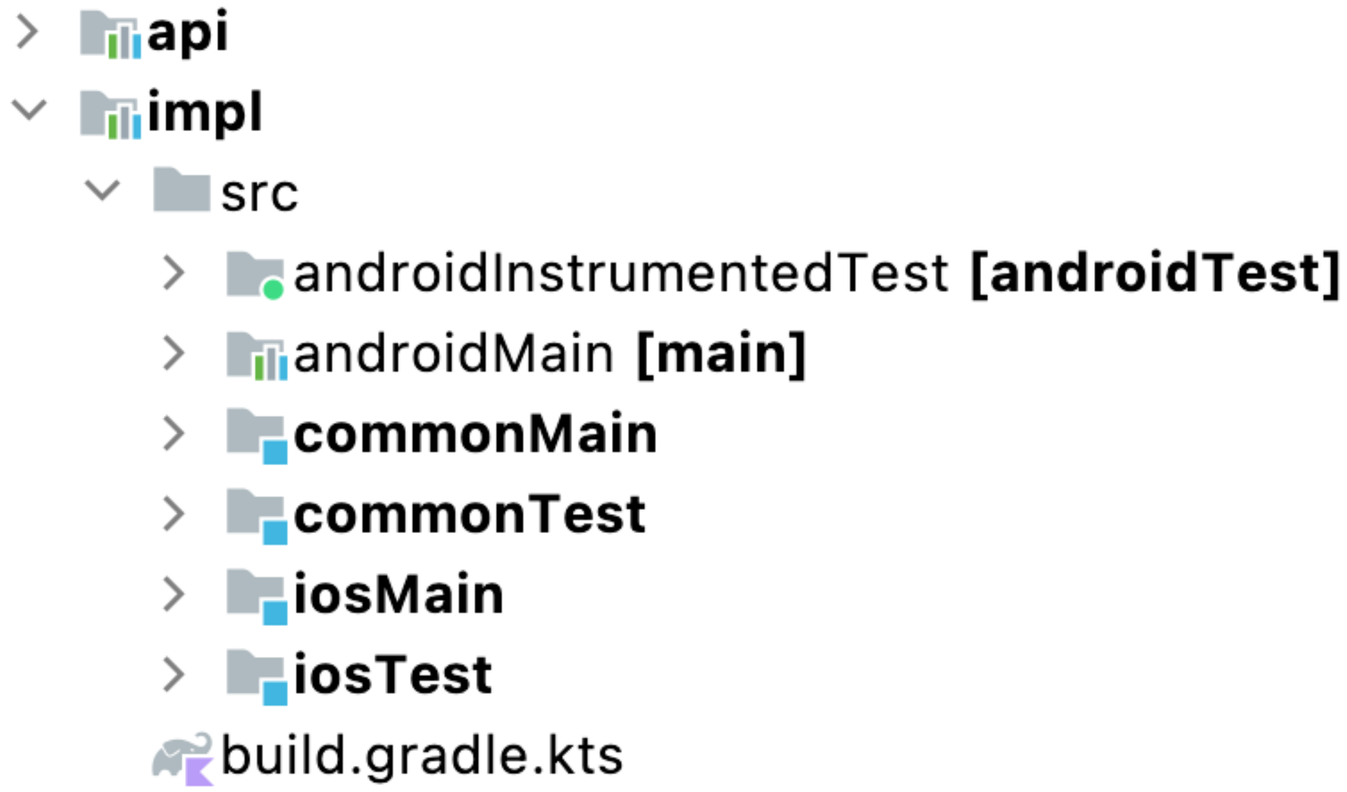
build.gradle.kts

```
kotlin { this: KotlinMultiplatformExtension
...
    sourceSets { this: NamedDomainObjectContainer<KotlinSourceSet>
...
        val commonTest by getting { this: KotlinSourceSet!
            dependencies { this: KotlinDependencyHandler
                implementation(kotlin( simpleModuleName: "test"))
            }
        }
        val androidInstrumentedTest by getting { this: KotlinSourceSet!
            dependsOn(commonTest)
            dependencies { this: KotlinDependencyHandler
                implementation( dependencyNotation: "androidx.test:runner:1.5.2")
            }
        }
        val iosX64Test by getting
        val iosArm64Test by getting
        val iosSimulatorArm64Test by getting
        val iosTest by creating { this: KotlinSourceSet
            dependsOn(commonTest)
            iosX64Test.dependsOn( other: this)
            iosArm64Test.dependsOn( other: this)
            iosSimulatorArm64Test.dependsOn( other: this)
        }
    }
}
```

build.gradle.kts

```
android { this: LibraryExtension
...
    defaultConfig { this: LibraryDefaultConfig
...
        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }
    sourceSets["androidTest"].manifest.srcFile( srcPath: "src/androidInstrumentedTest/AndroidManifest.xml")
}
```

Расположение тестовых исходников



Тестим платформенные имплементации «в лоб»

Дзен

Контрактное тестирование

подставляем реализацию

- iOS реализация
- Android релаизация
- Fake реализация

src/commonMain

```
interface FileStorage {
    fun saveOnDisk(fileName: String, content: String)
    fun getFromDisk(fileName: String): String
}
```

src/commonTest

```
class FakeFileStorage: FileStorage {
    private val fileToContent = mutableMapOf<String, String>()

    override fun saveOnDisk(fileName: String, content: String) {
        fileToContent[fileName] = content
    }
    override fun getFromDisk(fileName: String): String {
        return fileToContent[fileName] ?: ""
    }
}
```

src/commonTest

```
abstract class FileStorageTestContract {
    abstract val fileStorage: FileStorage
    @Test
    fun check() {
        val testText = "test_text"
        val testFile = "test_file"
        fileStorage.saveOnDisk(testFile, testText)
        assertEquals(testText, fileStorage.getFromDisk(testFile))
    }
}
```

src/commonTest

```
class FakeFileStorageTest: FileStorageTestContract() {
    override val fileStorage: FileStorage = FakeFileStorage()
}
```

- Для запуска на JVM

  gradle :module_name:testDebugUnitTest --tests *FakeFileStorageTest

- Для запуска на iOS симуляторе

  gradle :module_name:iosX64Test --tests *FakeFileStorageTest

src/iosMain

```
class IosFileStorage : FileStorage {
    private val fileManager
        get() = NSFileManager.defaultManager()

    override fun saveOnDisk(fileName: String, content: String) {
        val byteArray = content.encodeToByteArray()
        fileManager.createFileAtPath(
            path = NSURL(string = fileName).path!!,
            contents = memScoped { this: MemScope
                NSData.create(
                    bytes = allocArrayOf(byteArray),
                    length = byteArray.size.toULong()
                )
            },
            attributes = null,
        )
    }
...
```

```
...
    override fun getFromDisk(fileName: String): String {
        val data = fileManager.contentsAtPath(NSURL(string = fileName).path!!)
        return data?.let { it: NSData
            val length = data.length.toInt()
            val byteArray = ByteArray(length).apply { this: ByteArray
                if (length > 0) {
                    usePinned { it: Pinned<ByteArray>
                        memcpy(it.addressOf( index: 0), data.bytes, data.length)
                    }
                }
            }
            byteArray.decodeToString() ^let
        } ?: ""
    }
}
```

src/androidMain

```
class AndroidFileStorage(private val context: Context) : FileStorage {
    override fun saveOnDisk(fileName: String, content: String) {
        File(context.filesDir, fileName).writeText(content)
    }
    override fun getFromDisk(fileName: String): String {
        return File(context.filesDir, fileName).readText()
    }
}
```

src/iosTest

```
class IosFileStorageTest: FileStorageTestContract() {
    override val fileStorage: FileStorage = IosFileStorage()
}
```

src/androidInstrumentedTest

```
class AndroidFileStorageTest: FileStorageTestContract() {
    private val application
        get() = InstrumentationRegistry.getInstrumentation()
            .targetContext.applicationContext
    override val fileStorage: FileStorage = AndroidFileStorage(application)
}
```

Gradle таски

- Android:

```
gradle :module_name:connectedDebugAndroidTest
-Pandroid.testInstrumentationRunnerArguments.class=
path.to.AndroidFileStorageTest
```

- IOS

```
gradle :module_name:iosX64Test --tests *IosFileStorageTest
```

Преимущества контрактного тестирования

- TDD
- Единое поведение
- Время

Дзен

Итоги

- Появилась фича на IOS
- Технология разрослась на всю команду
- Продолжаем унифицировать логику (bandwith meter, аналитика в плеере)

Разделение на UI и бизнес

+

Тщательная документация

+

Разделение модулей
на Api и Implementation

+

Тестирование

=

«Хороший» код