Мультиплатформенные проекты в Kotlin 1.3

Матвеев Илья, JetBrains
ilmat192@gmail.com
Mobius 2019 Piter

О чём будем говорить?

- Лирика: общие концепции MPP*
- Теория: как это выглядит в языке и тулинге
- Практика: приложение iOS + Android

*MPP - мультиплатформенное программирование

Немного про Котлин

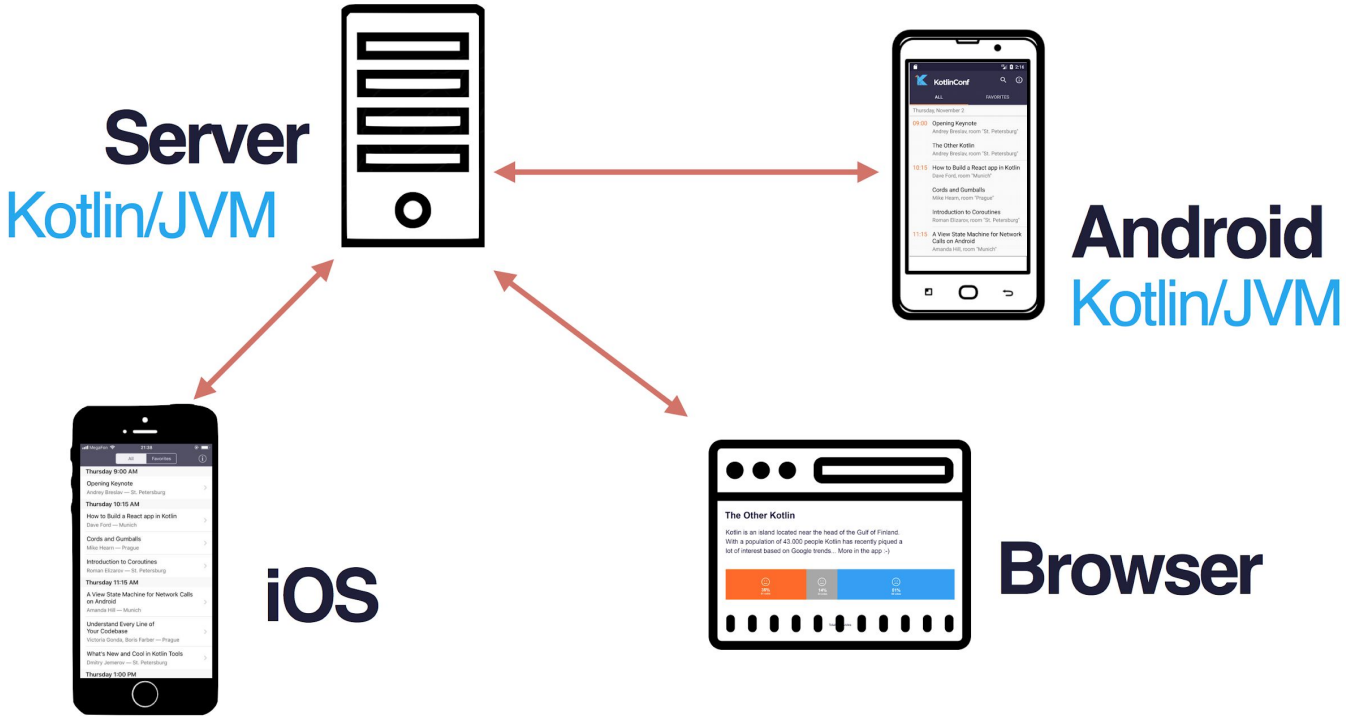
- Начало разработки - 2010. Первый релиз - 2016.
- Ключевые слова:
  - Безопасность
  - Лаконичность
  - Прагматичность

Немного про Котлин

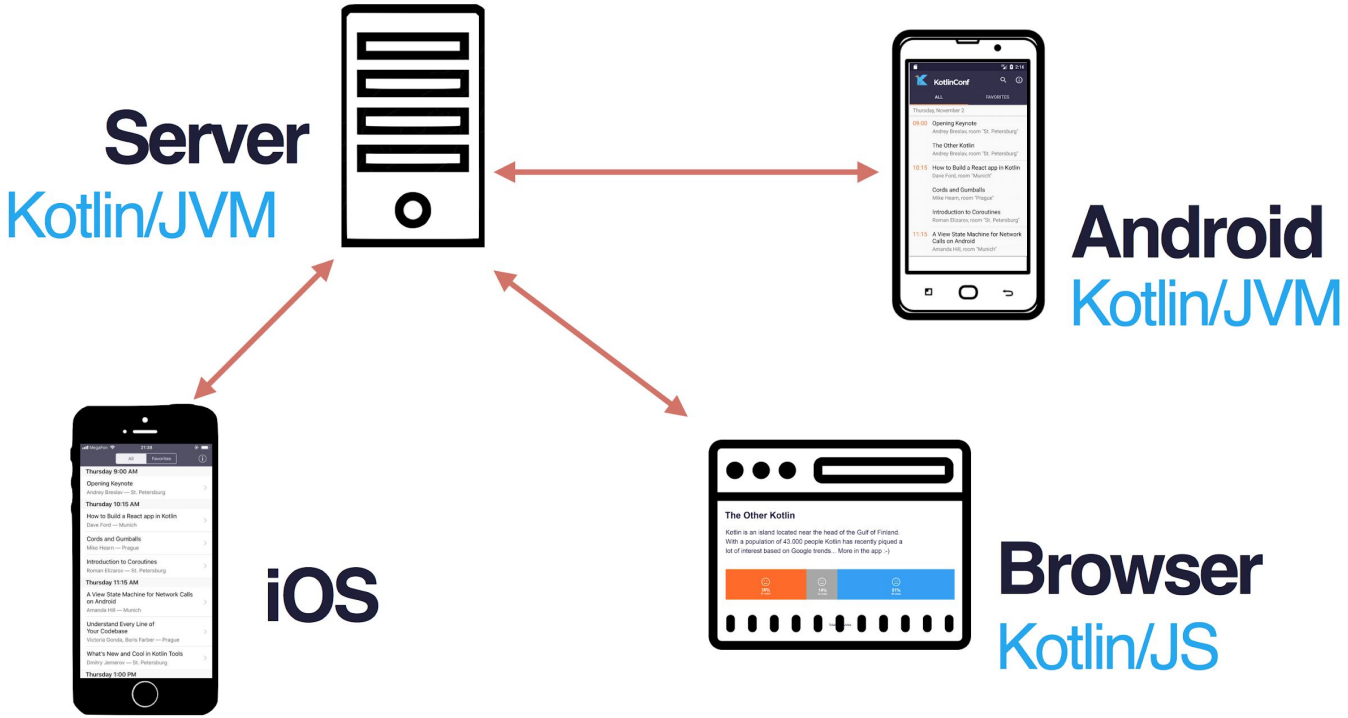
- Начало разработки - 2010. Первый релиз - 2016.
- Ключевые слова:
  - Безопасность
  - Лаконичность
  - Прагматичность

- Изначально - JVM-based язык

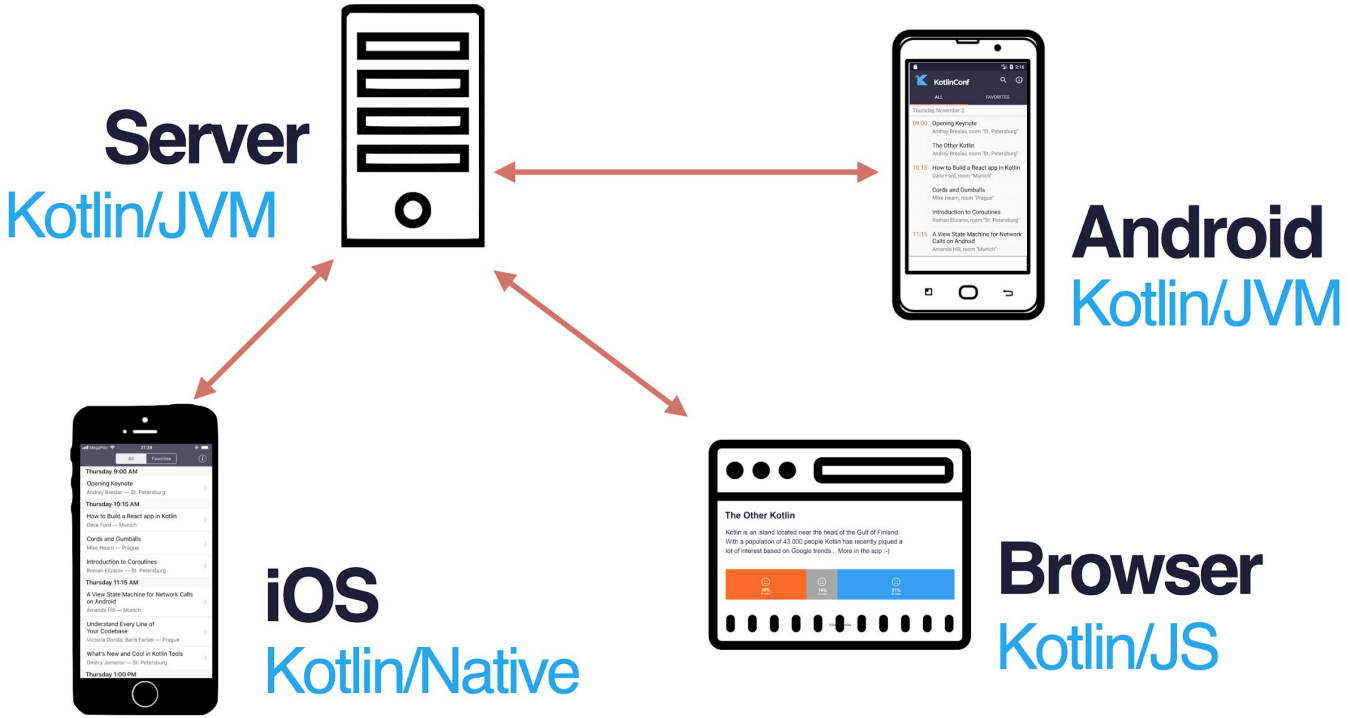
Мультиплатформенные проекты



Мультиплатформенные проекты



Мультиплатформенные проекты



Мультиплатформенные проекты

Kotlin/Native

≠

Kotlin Multiplatform

Что хотим?

- Переиспользовать код на разных платформах

Что хотим?

- Переиспользовать код на разных платформах
- Иметь доступ к платформенно-специфичным API

Что хотим?

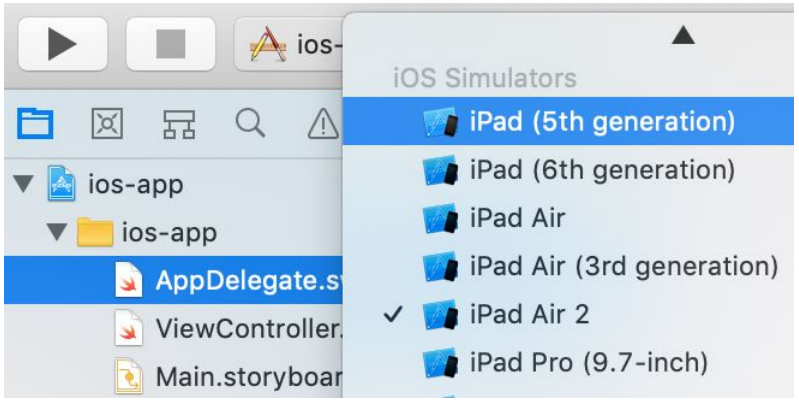
- Переиспользовать код на разных платформах
- Иметь доступ к платформенно-специфичным API
- Пользоваться поддержкой со стороны IDE

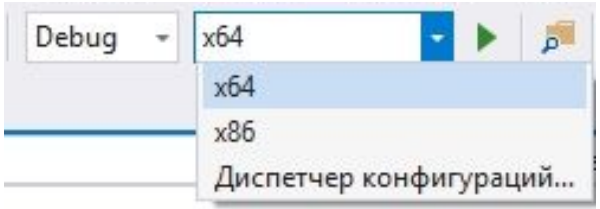
Что не хотим?

```
// Common logic.
printf("Platform: ");

#ifdef __i386__
    // x86-only logic.
    printf("x86\n");
#endif
```

Что не хотим?


Available Resolve Contexts
hello [x64 Debug]
hello [x64 Release]
hello [x86 Debug]
hello [x86 Release]
File will be resolved in the selected context
UTF-8
Context: hello [x64D]


iOS Simulators
iPad (5th generation)
iPad (6th generation)
iPad Air
iPad Air (3rd generation)
iPad Air 2
iPad Pro (9.7-inch)
ios-app
ios-app
AppDelegate.s
ViewController.
Main.storyboar


Debug
x64
x64
x86
Диспетчер конфигураций...

Что хотим?

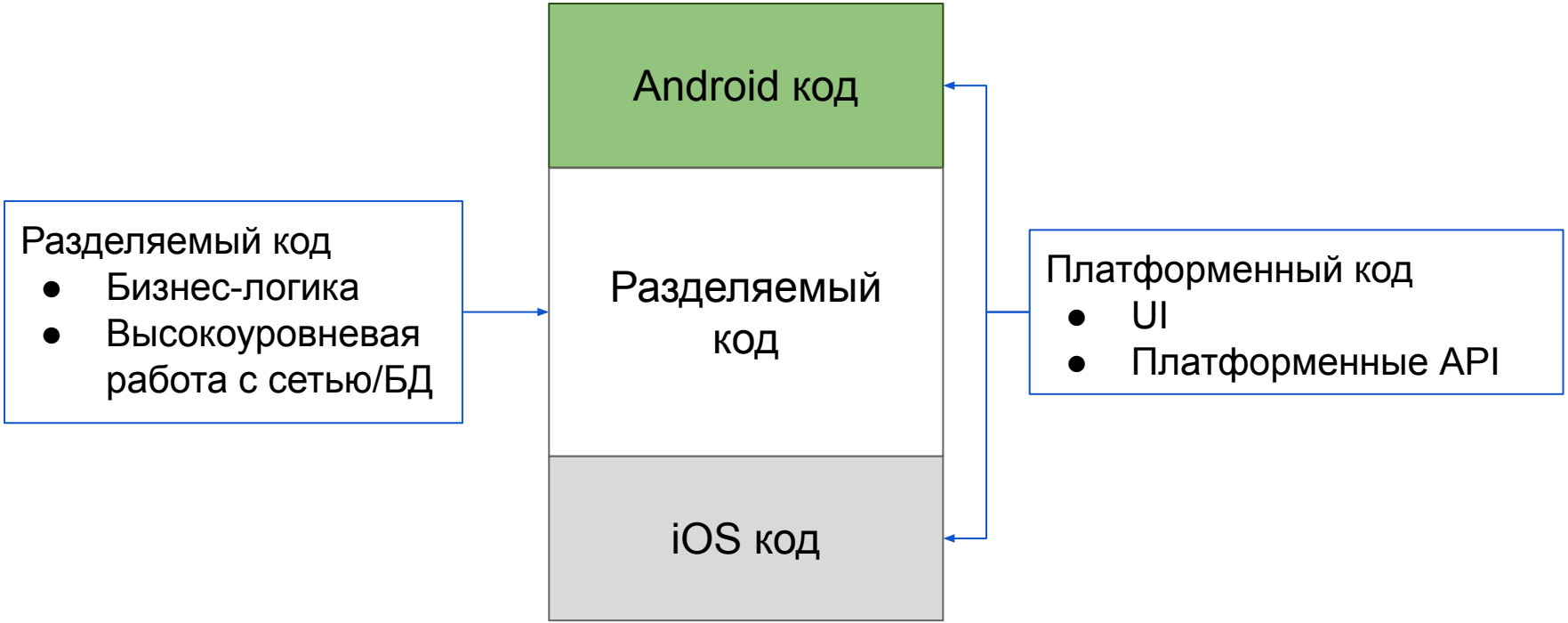
- Переиспользовать код на разных платформах
- Иметь доступ к платформенно-специфичным API
- Пользоваться поддержкой со стороны IDE

Что хотим?

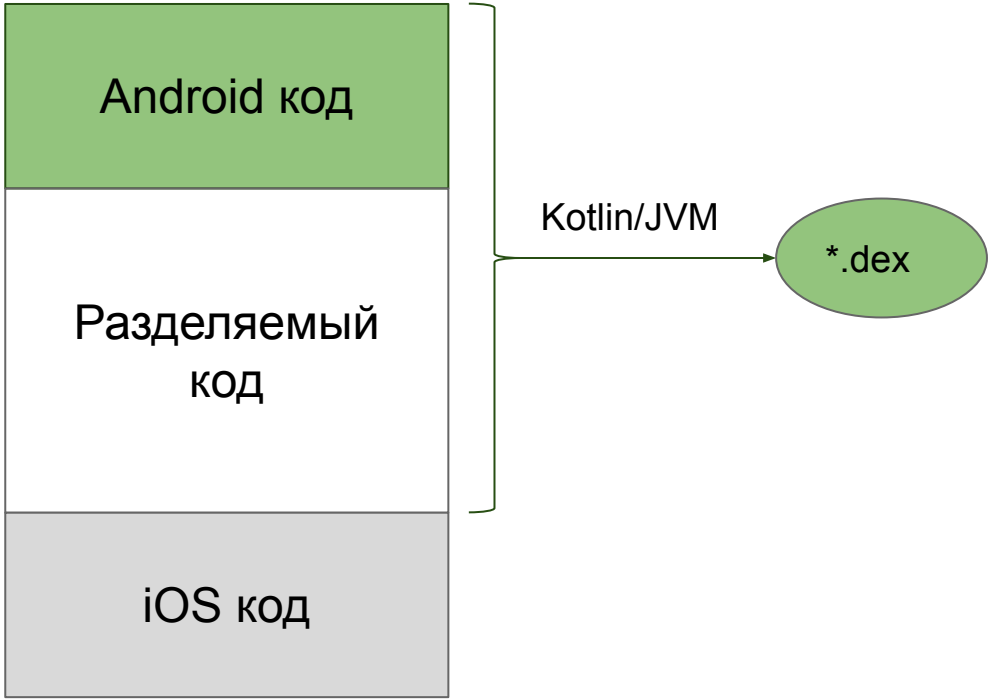
- Переиспользовать код на разных платформах
- Иметь доступ к платформенно-специфичным API
- Пользоваться поддержкой со стороны IDE
- Пользоваться поддержкой со стороны билд-системы

Как это работает

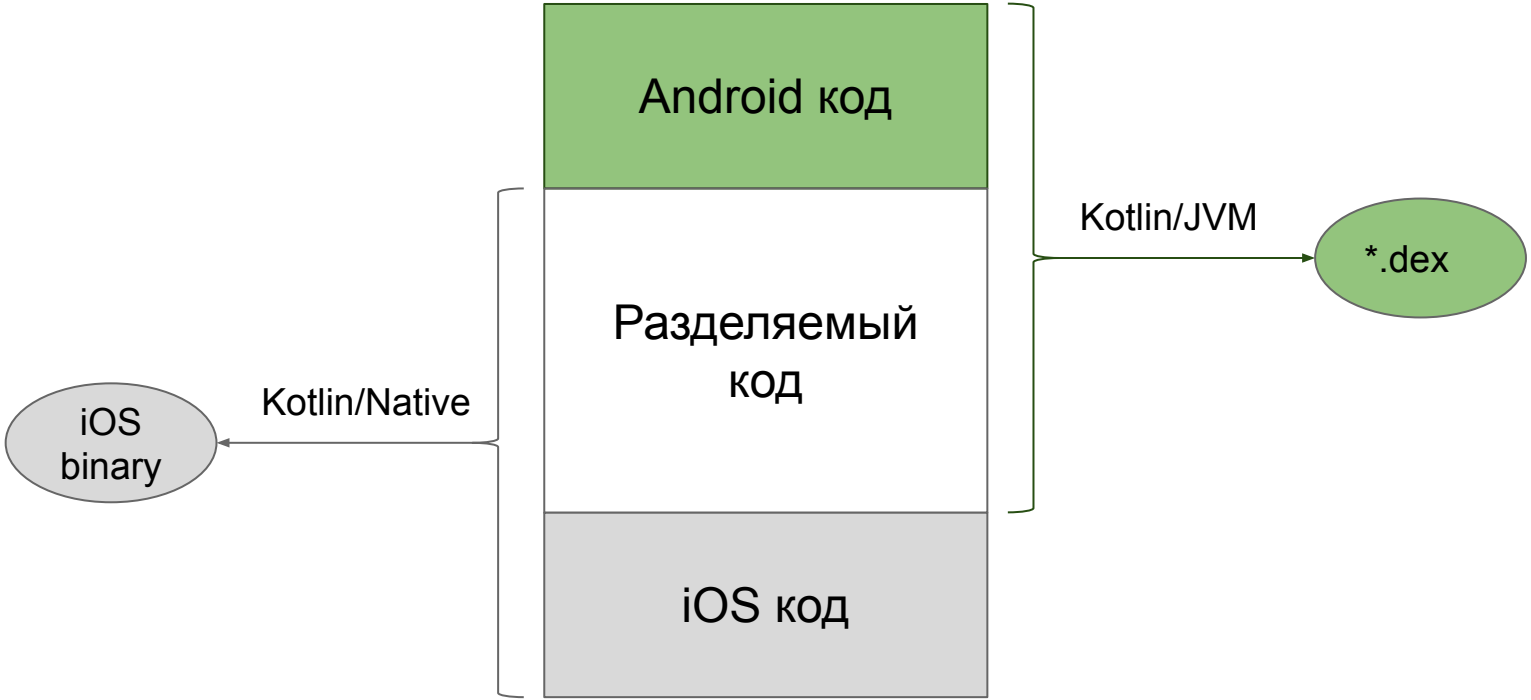
Как это работает



Как это работает



Как это работает



MPP и синтаксис

Expect/actual

```
expect class Logger {
    fun log(message: String)
}
```

Expect/actual

```
expect class Logger {
    fun log(message: String)
}
```

```
import platform.Foundation.*

actual class Logger() {
    fun log(message: String) {
        NSLog(message)
    }
}
```

```
import android.util.Log

actual class Logger {
    fun log(message: String) {
        Log.i("Tag", message)
    }
}
```

Почему не интерфейсы?

Почему не интерфейсы?

```
class LoggerFactory {
    fun createLogger(): Logger { /* Magic. */ }
}

// ...
```

Сложный dependency injection

```
val logger = LoggerFactory().createLogger()
logger.log("Hello!")
```

Громоздкое использование

Почему не интерфейсы?

```
expect class Logger {
    constructor(tag: String)
    // ...
}

// ...
```

Expect-класс может иметь конструктор

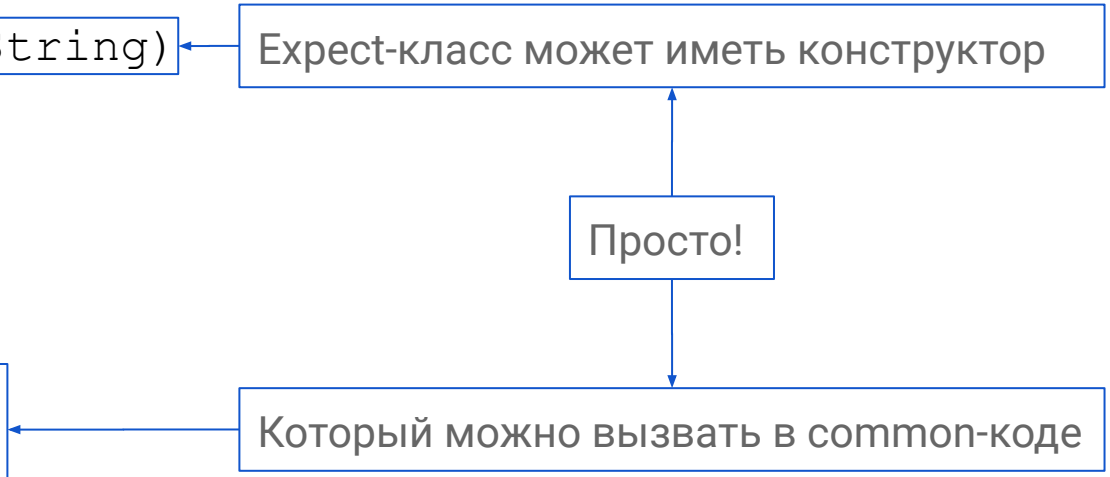
```
val logger = Logger()
logger.log("Hello!")
```

Который можно вызвать в common-коде

Почему не интерфейсы?

```
expect class Logger {
    constructor(tag: String)
    // ...
}

// ...
```

```
val logger = Logger()
logger.log("Hello!")
```



Почему не интерфейсы?

```
expect class Logger {
    constructor(tag: String)
    // ...
}
```

Expect-класс может иметь конструктор

```
expect fun withLogger(action: Logger.() -> Unit)
expect fun Logger.logError(exception: Throwable)
```

Можно объявлять expect-функции

Expect/actual

```
package kotlin.collections

expect class ArrayList<E> : MutableList<E>, RandomAccess {
    /* ... */
}
```

Expect/actual

```
package kotlin.collections

expect class ArrayList<E> : MutableList<E>, RandomAccess {
    /* ... */
}
```

```
package kotlin.collections

actual typealias ArrayList<E> = java.util.ArrayList<E>
```

Expect/actual

```
package kotlin.test

expect annotation class Test
```

Expect/actual

```
package kotlin.test

expect annotation class Test
```

```
package kotlin.test

actual typealias Test = org.junit.Test
```

JUnit

Expect/actual

```
package kotlin.test

expect annotation class Test
```

```
package kotlin.test

actual typealias Test = org.junit.Test
```

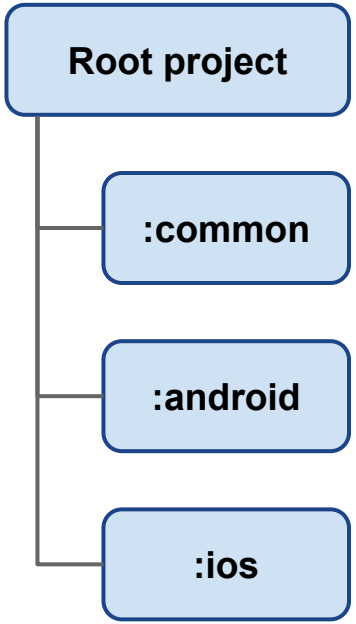
JUnit

```
package kotlin.test

actual typealias Test = org.testng.annotations.Test
```

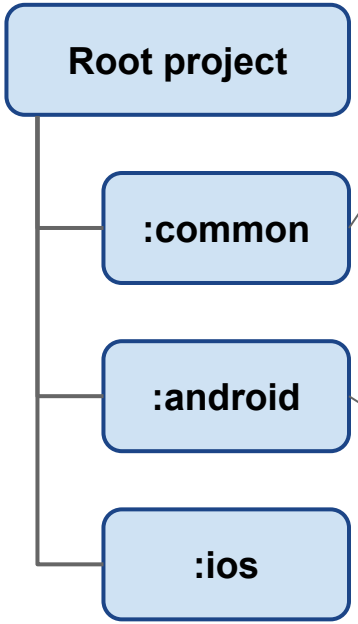
TestNG

MPP и Gradle (1.2)

Мультиплатформа и Gradle (1.2)



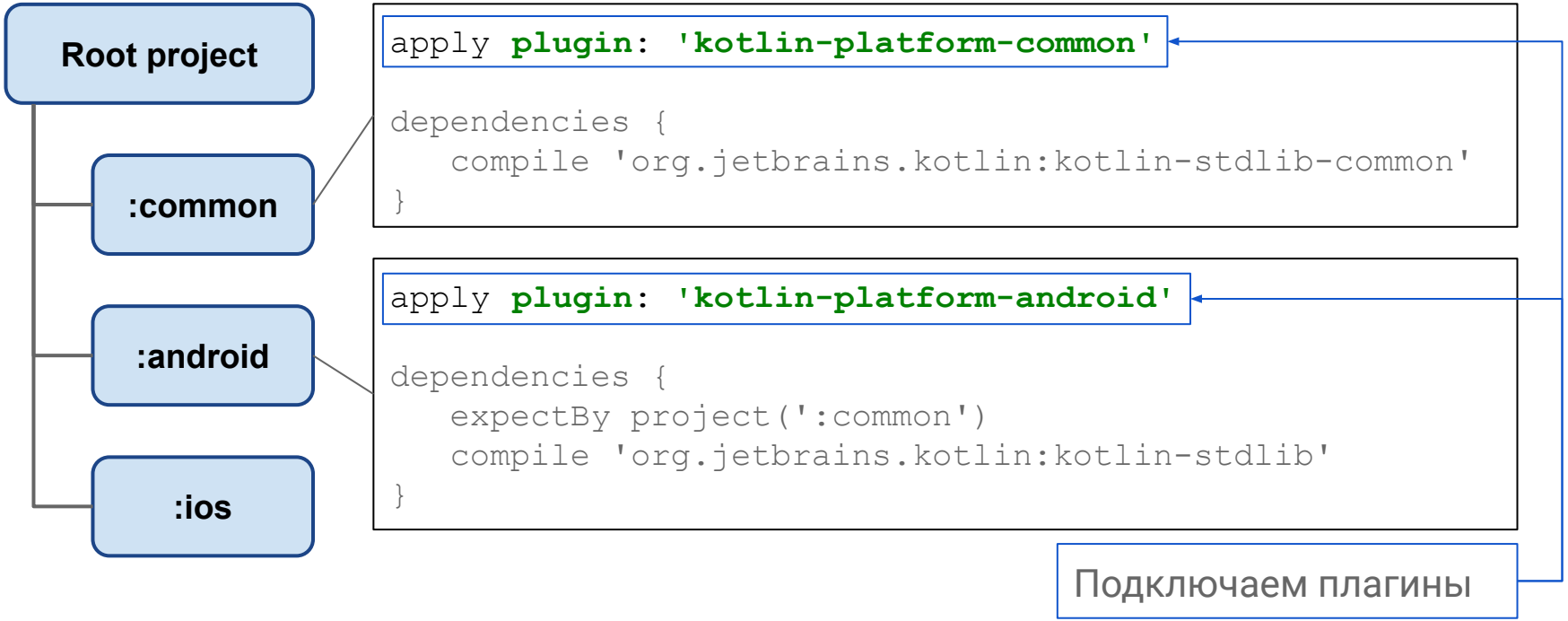
Мультиплатформа и Gradle (1.2)



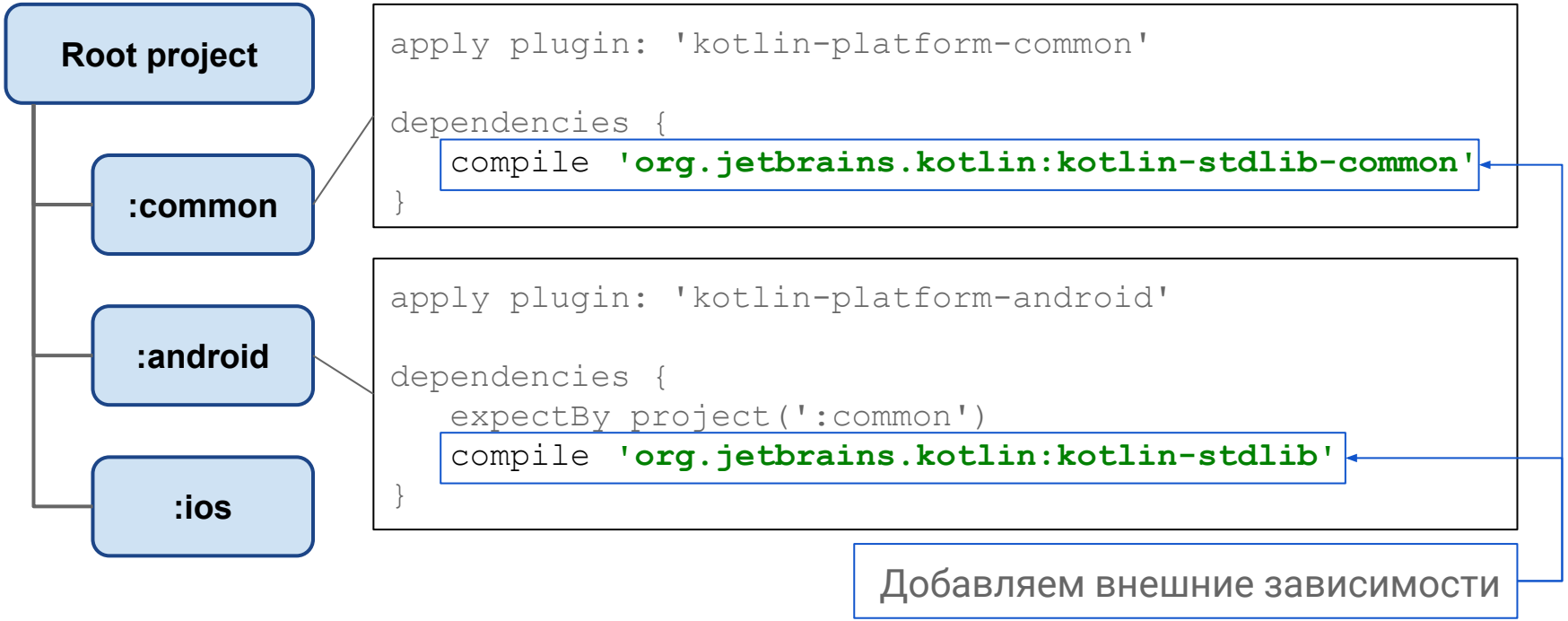

```
apply plugin: 'kotlin-platform-common'

dependencies {
    compile 'org.jetbrains.kotlin:kotlin-stdlib-common'
}
```

```
apply plugin: 'kotlin-platform-android'

dependencies {
    expectBy project(':common')
    compile 'org.jetbrains.kotlin:kotlin-stdlib'
}
```

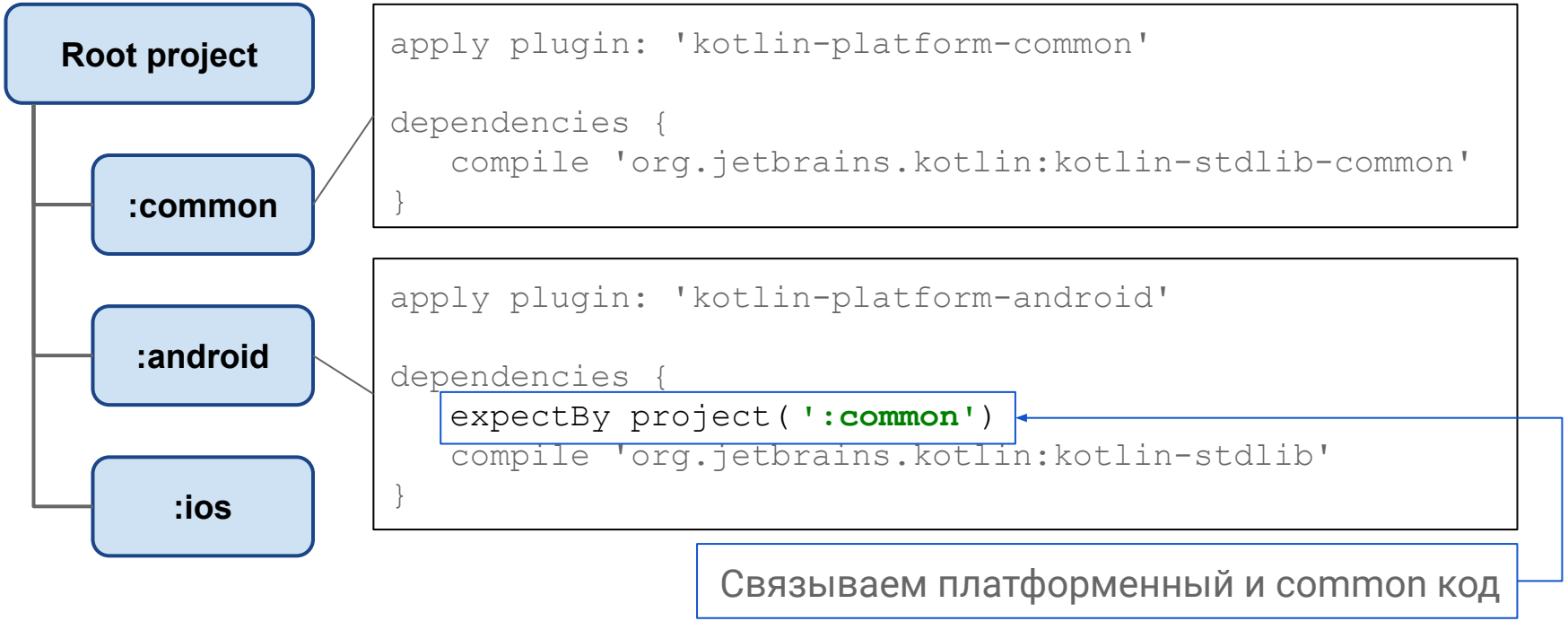
Мультиплатформа и Gradle (1.2)



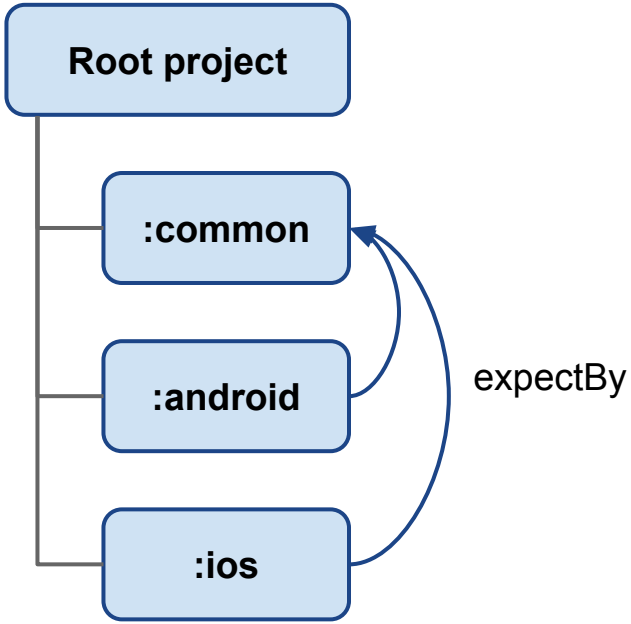
Мультиплатформа и Gradle (1.2)



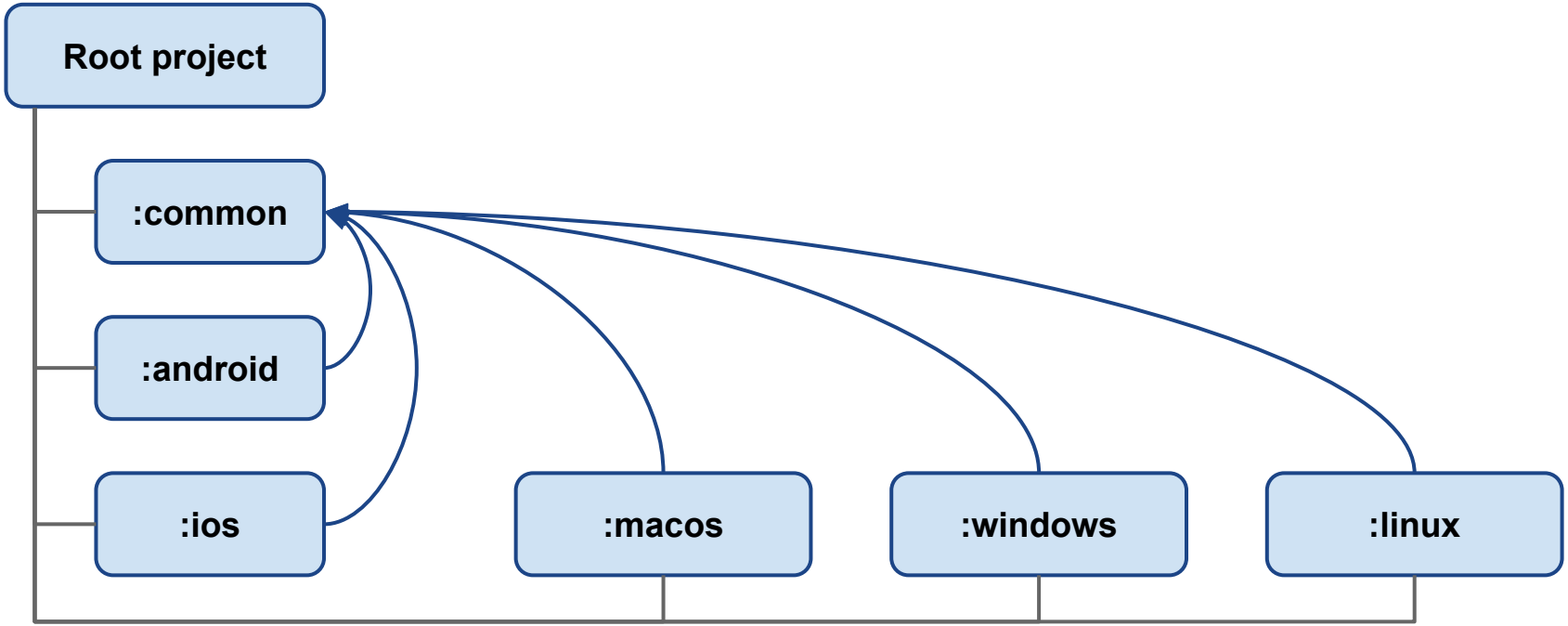
Мультиплатформа и Gradle (1.2)



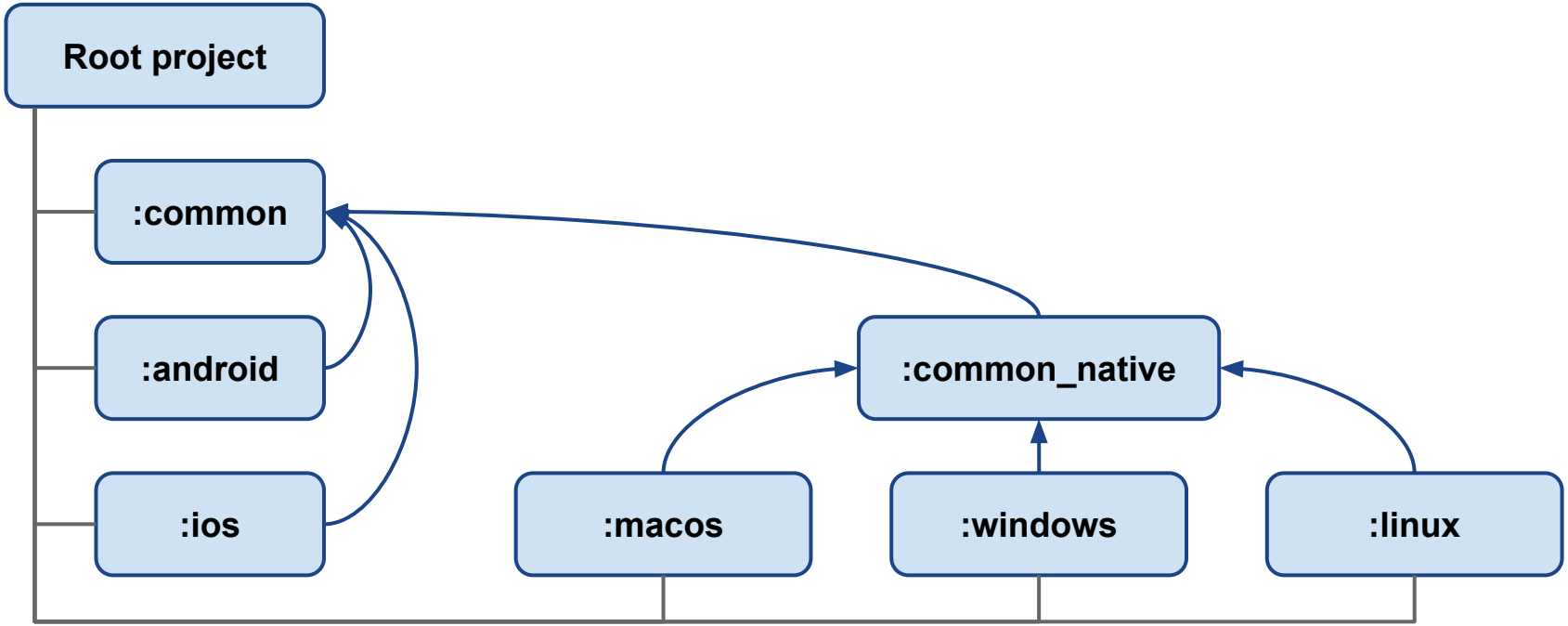
Мультиплатформа и Gradle (1.2)



Проблемы модели 1.2



Проблемы модели 1.2



Проблемы модели 1.2

https://youtrack.jetbrains.com/issue/KT-23930

Scott Pierce ● commented 23 Apr 2018 06:37

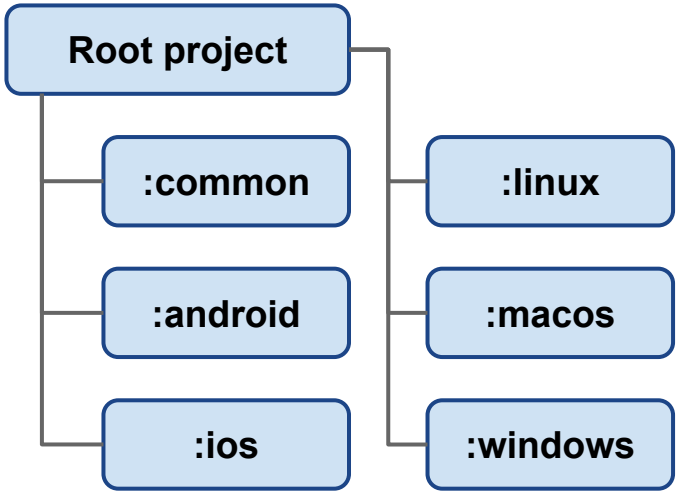
What would normally be 17 gradle modules has become 48 modules because of mpp. The project is fairly new (<2 months old), and will only continue to grow in size. We're being pretty good about separating our code into potentially re-usable libraries, (i.e. a MPP logging library). Eventually we'll split some of the libraries off into their own projects potentially, but still.

Проблемы модели 1.2

1. Число модулей растет с числом платформ

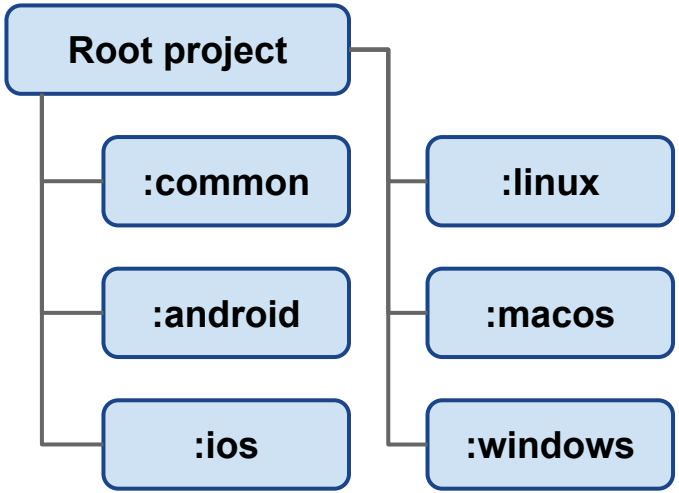
Проблемы модели 1.2

org.example:my-library:1.0



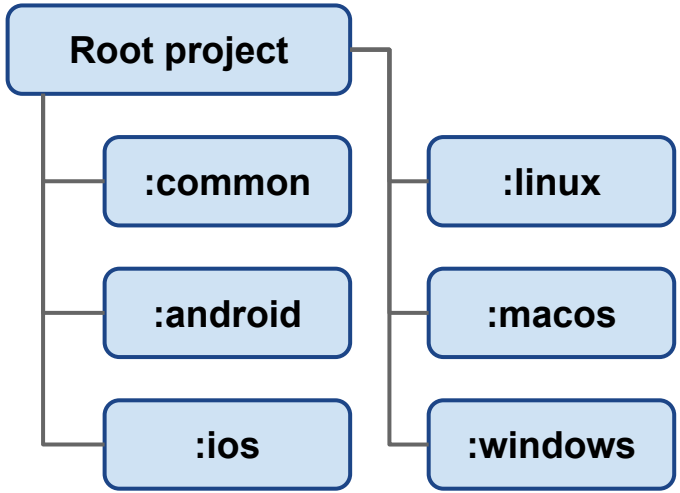
Проблемы модели 1.2

org.example:my-library:1.0



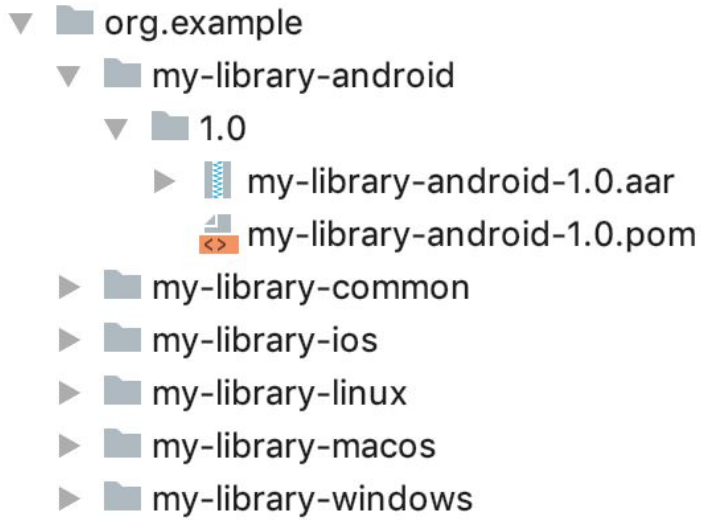
Depends on

org.example:my-application:1.0



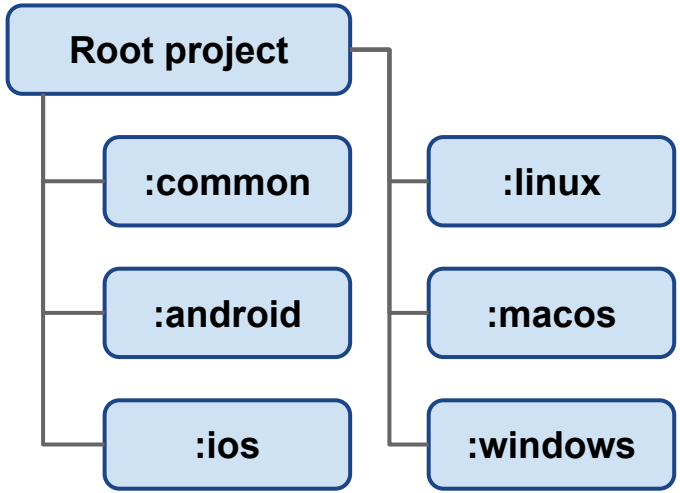
Проблемы модели 1.2

org.example:my-library:1.0



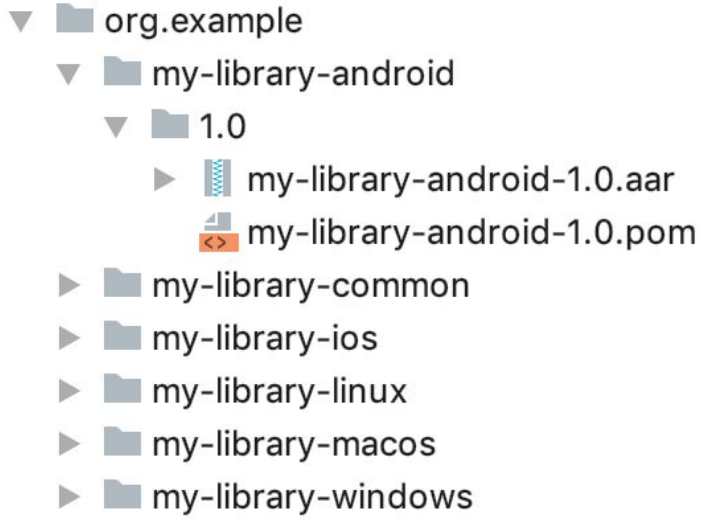
Depends on

org.example:my-application:1.0



Проблемы модели 1.2

org.example:my-library:1.0




```
dependencies {
    compile 'org.example:my-library-common'
}
```

```
dependencies {
    compile 'org.example:my-library-ios'
}
```

```
dependencies {
    compile 'org.example:my-library-linux'
}
```

...

Проблемы модели 1.2

1. Число модулей растет с числом платформ
2. Зависимости настраиваются вручную

MPP и Gradle (1.3)

Все платформы в одном модуле

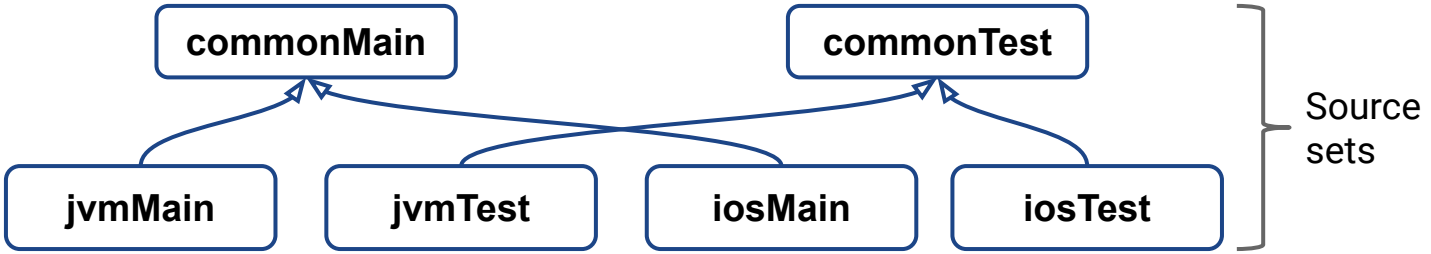
Все платформы в одном модуле

1. Нужно управлять исходниками и их зависимостями

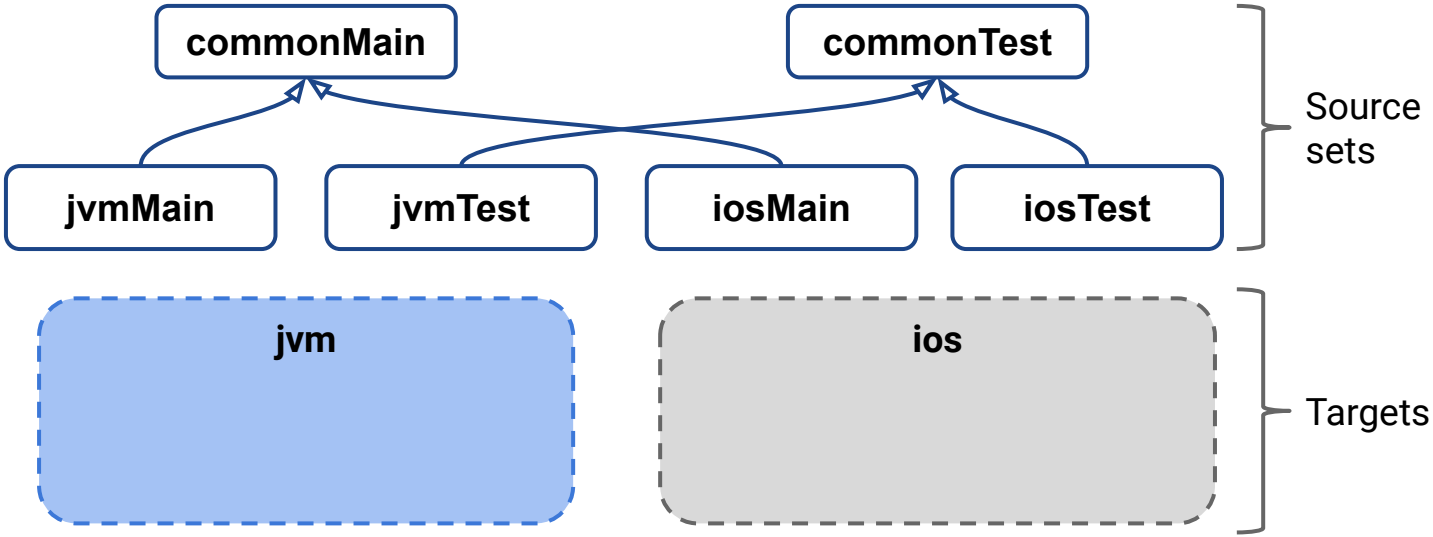
Все платформы в одном модуле

1. Нужно управлять исходниками и их зависимостями
2. Нужно управлять запуском компилятора для разных платформ

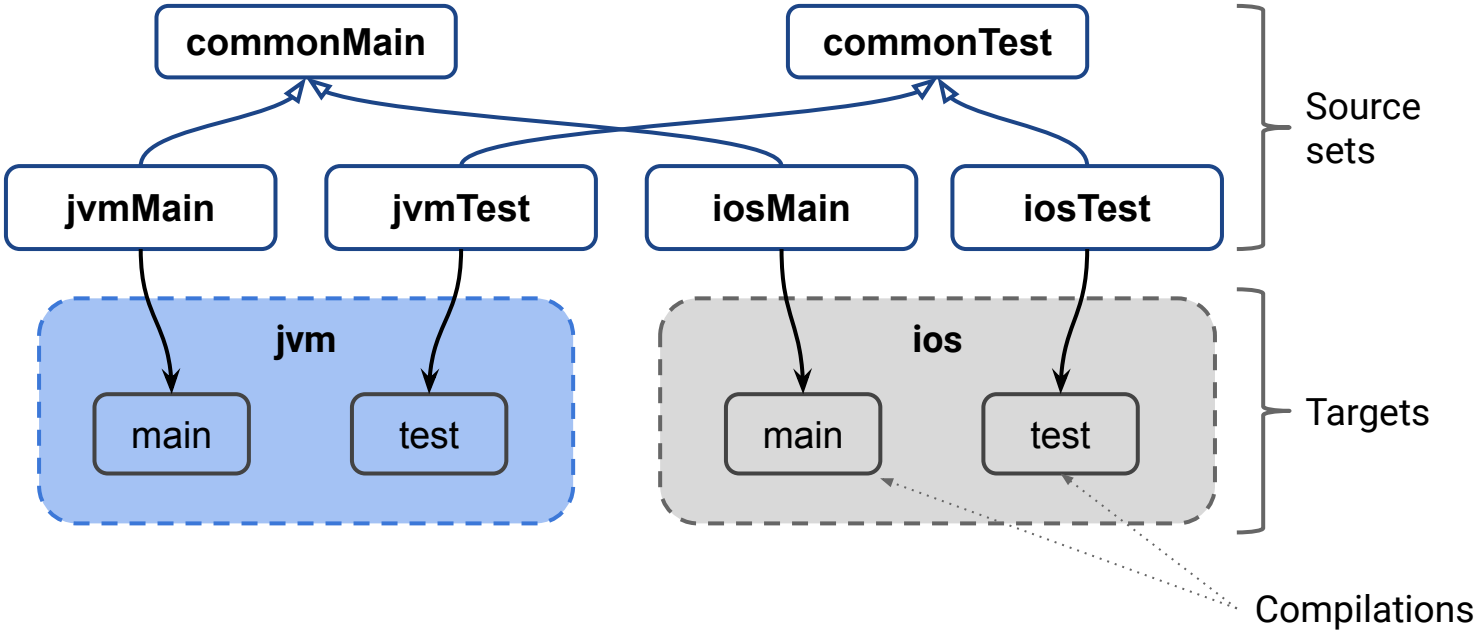
Мультиплатформа и Gradle в 1.3



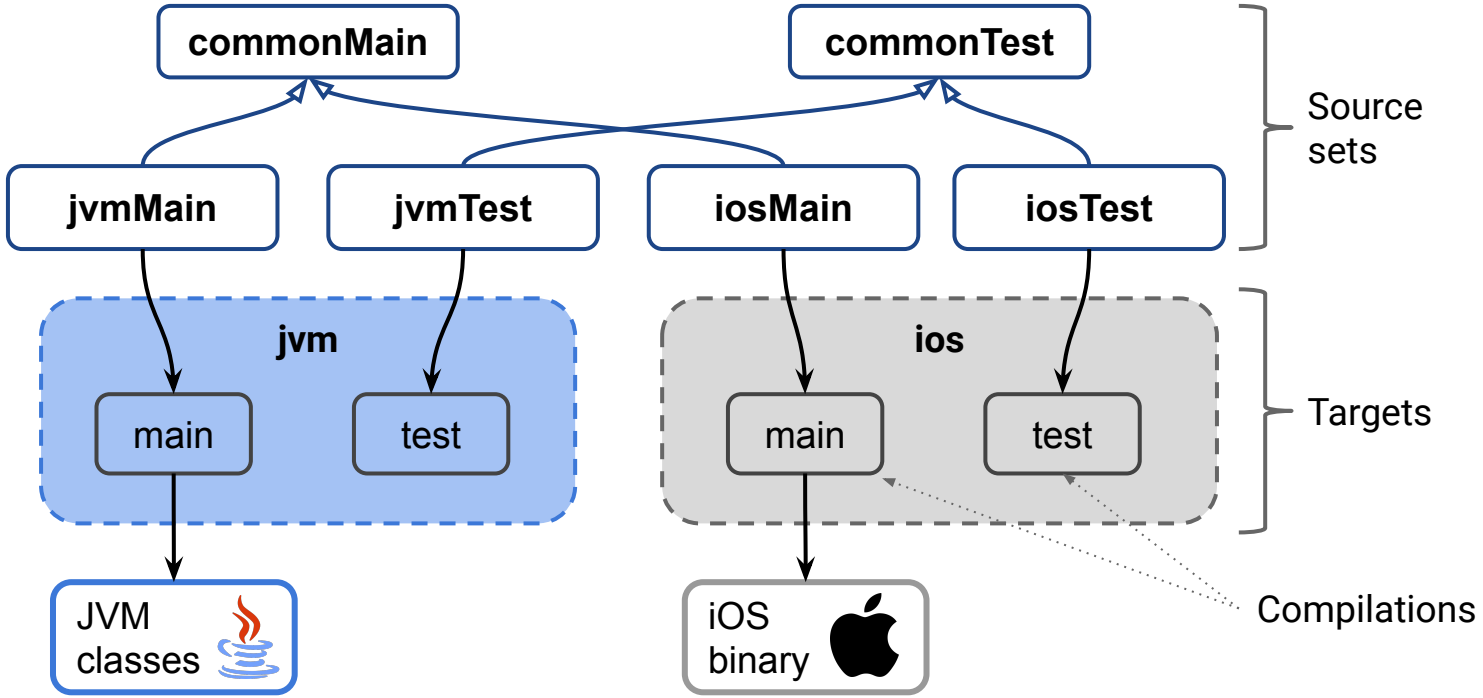
Мультиплатформа и Gradle в 1.3



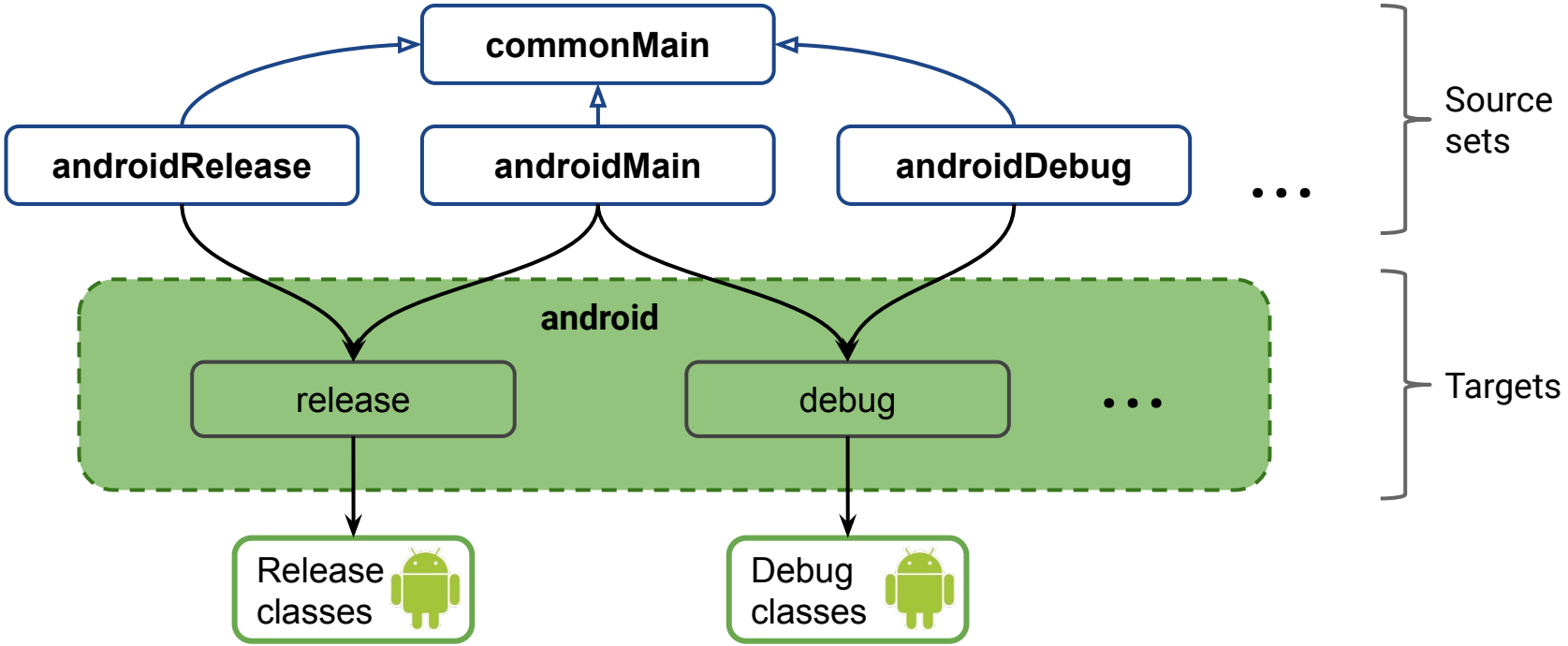
Мультиплатформа и Gradle в 1.3



Мультиплатформа и Gradle в 1.3



Мультиплатформа и Gradle в 1.3



Мультиплатформа и Gradle в 1.3

```
apply plugin: "kotlin-multiplatform"

kotlin {
    sourceSets.commonMain {
        dependencies {
            implementation "org.jetbrains.kotlin:kotlin-stdlib-common"
        }
    }

    android  { /* Android-specific configuration */ }
    iosArm64 { /* iOS-specific configuration     */ }
}
```

Мультиплатформа и Gradle в 1.3

```
apply plugin: "kotlin-multiplatform"
```

Подключаем плагин

```
kotlin {
    sourceSets.commonMain {
        dependencies {
            implementation "org.jetbrains.kotlin:kotlin-stdlib-common"
        }
    }

    android  { /* Android-specific configuration */ }
    iosArm64 { /* iOS-specific configuration     */ }
}
```

Мультиплатформа и Gradle в 1.3

```
apply plugin: "kotlin-multiplatform"

kotlin {
    sourceSets.commonMain {
        dependencies {
            implementation "org.jetbrains.kotlin:kotlin-stdlib-common"
        }
    }

    android  { /* Android-specific configuration */ }
    iosArm64 { /* iOS-specific configuration     */ }
}
```

Конфигурируем source set-ы

Мультиплатформа и Gradle в 1.3

```
apply plugin: "kotlin-multiplatform"

kotlin {
    sourceSets.commonMain {
        dependencies {
            implementation "org.jetbrains.kotlin:kotlin-stdlib-common"
        }
    }

    android  { /* Android-specific configuration */ }
    iosArm64 { /* iOS-specific configuration     */ }
}
```

Конфигурируем source set-ы

Настраиваем зависимости

Мультиплатформа и Gradle в 1.3

```
apply plugin: "kotlin-multiplatform"

kotlin {
    sourceSets.commonMain {
        dependencies {
            implementation "org.jetbrains.kotlin:kotlin-stdlib-common"
        }
    }

    android  { /* Android-specific configuration */ }
    iosArm64 { /* iOS-specific configuration     */ }
}
```

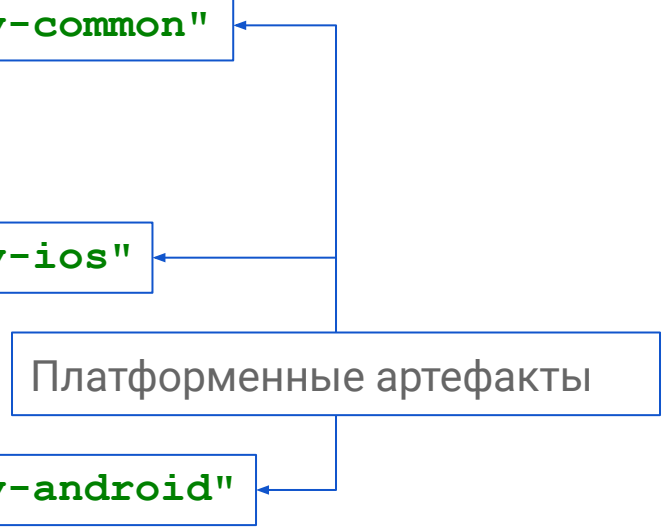
Конфигурируем таргеты

Проблемы модели 1.2

1. ~~Число модулей растет с числом платформ~~
2. Зависимости настраиваются вручную

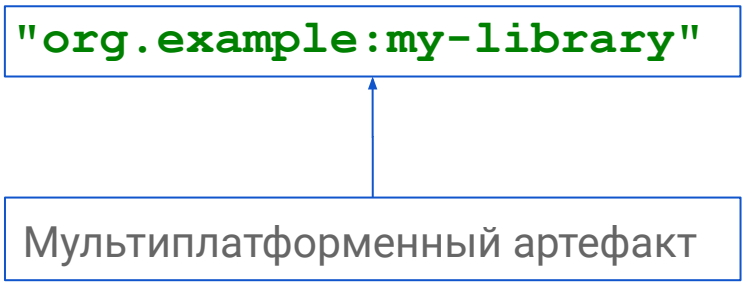
Управление зависимостями

```
commonMain.dependencies {
    implementation "org.example:my-library-common"
}

iosMain.dependencies {
    implementation "org.example:my-library-ios"
}

androidMain.dependencies {
    implementation "org.example:my-library-android"
}
```



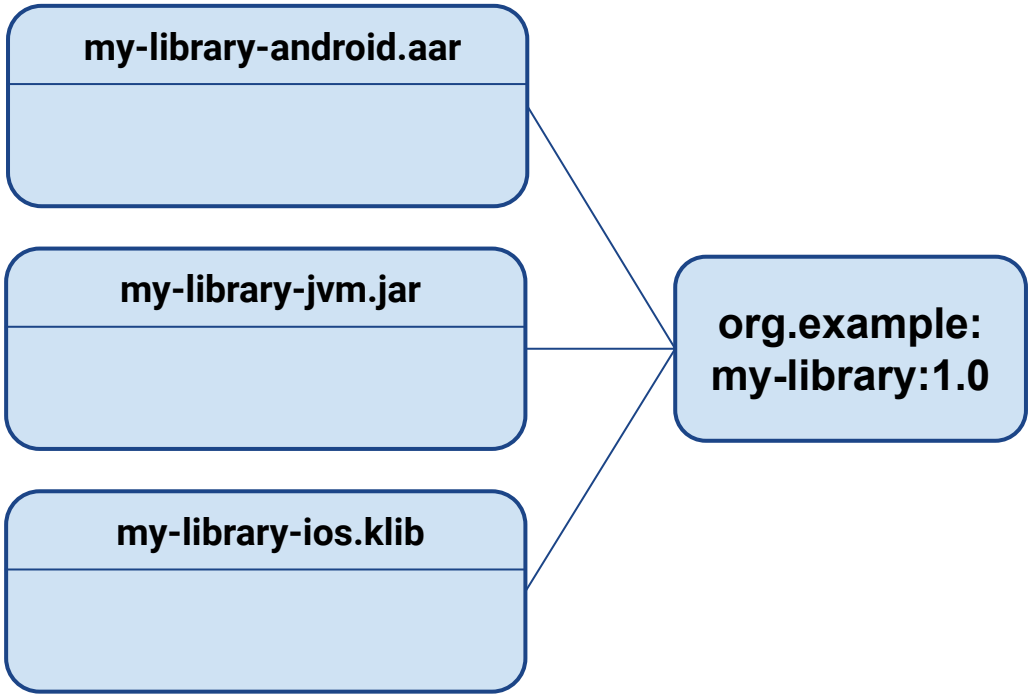
Управление зависимостями

```
commonMain.dependencies {
    implementation "org.example:my-library"
}
```

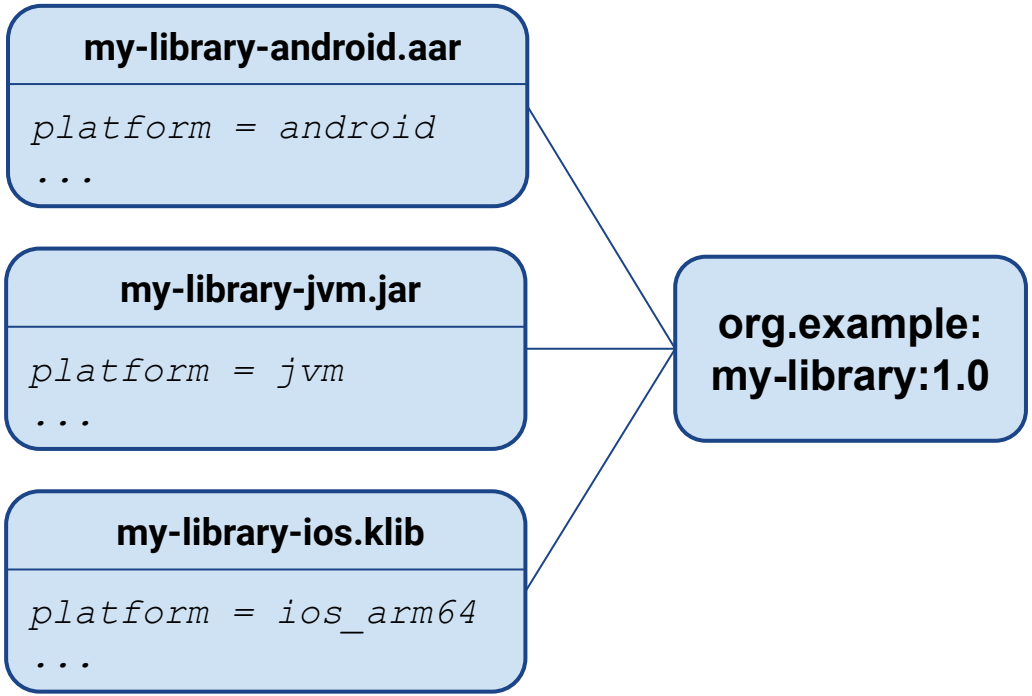


Variant-aware dependency management

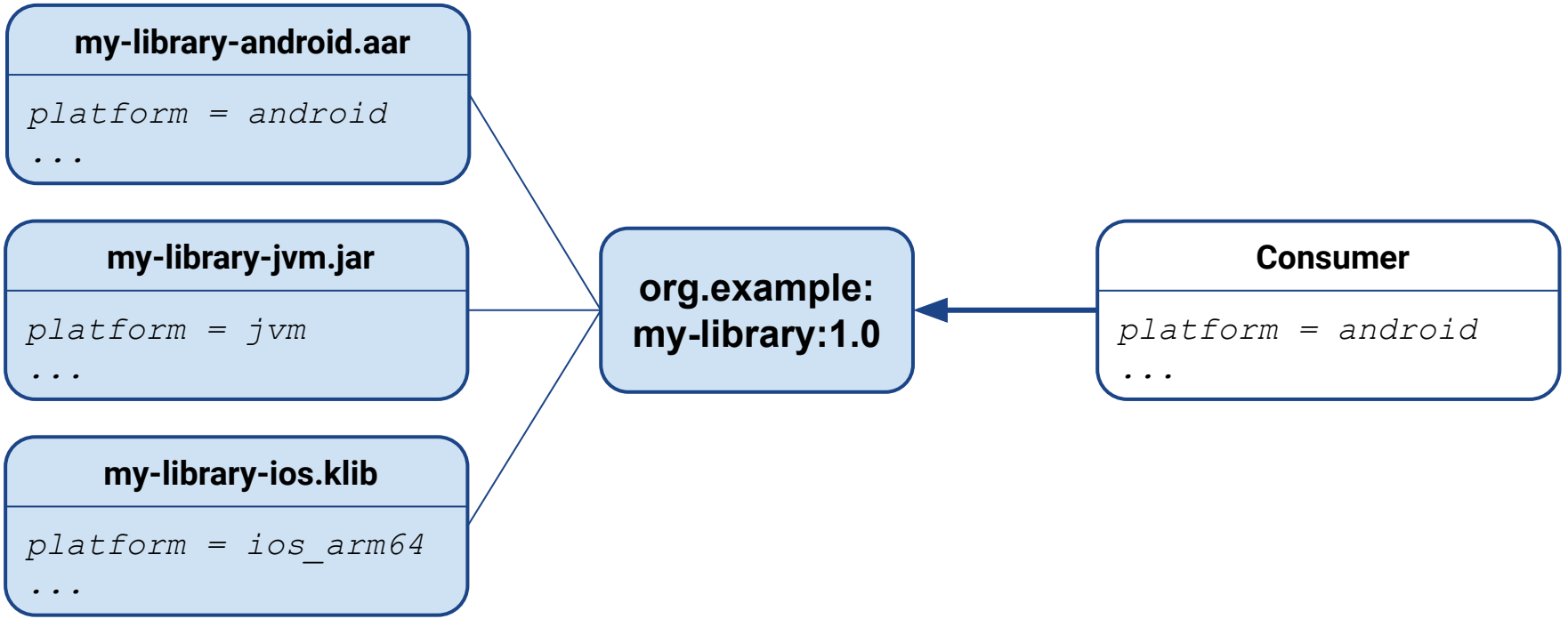
Variant-aware dependency management



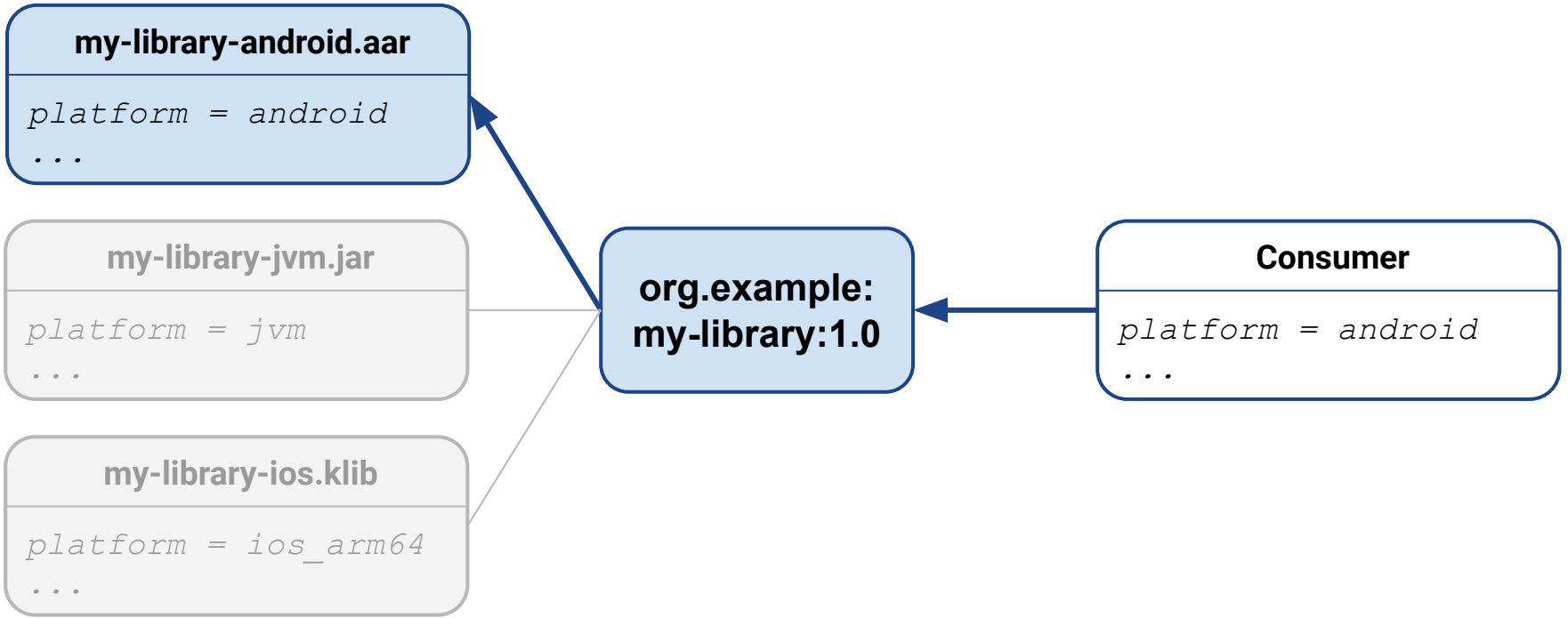
Variant-aware dependency management



Variant-aware dependency management

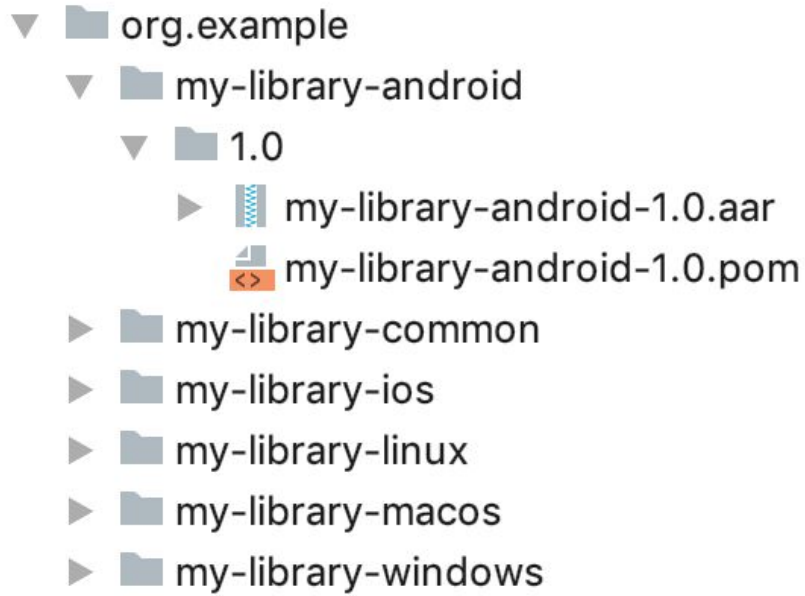


Variant-aware dependency management



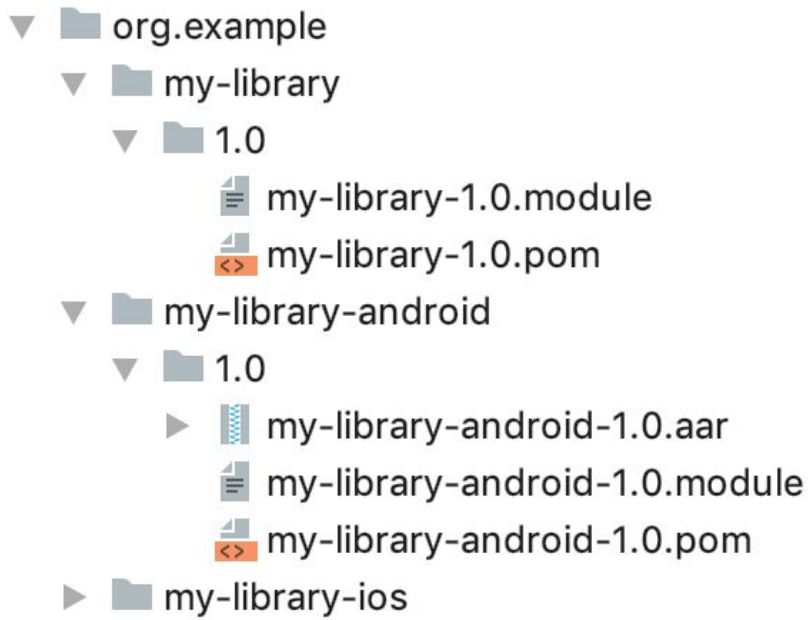
Публикация

org.example:my-library:1.0



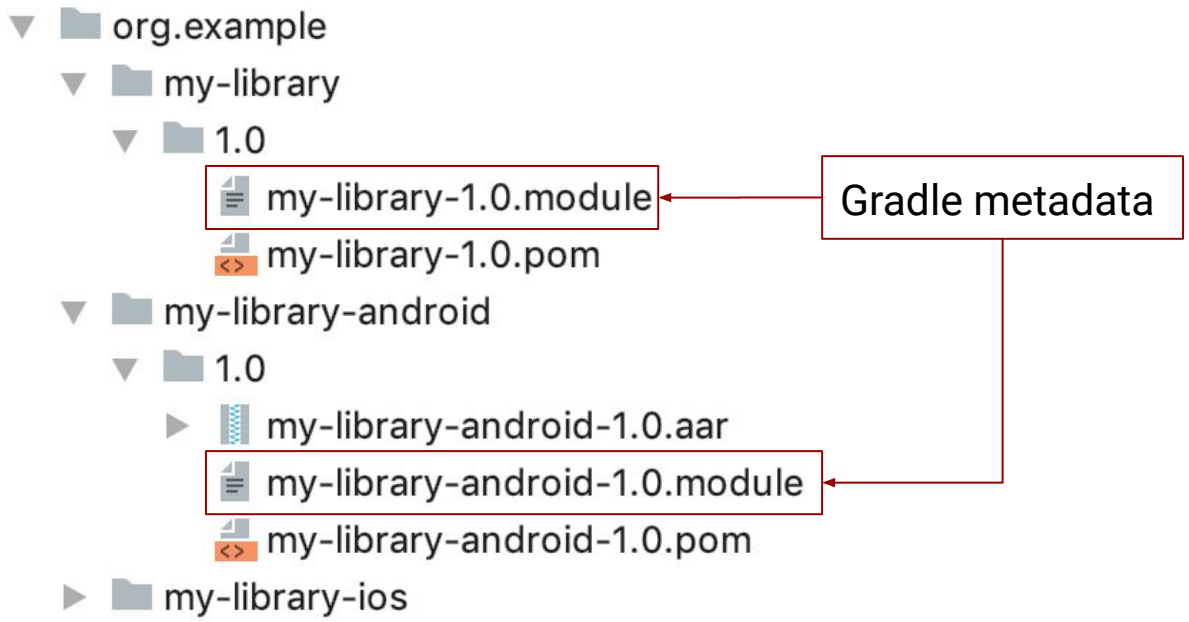
Публикация

org.example:my-library:1.0

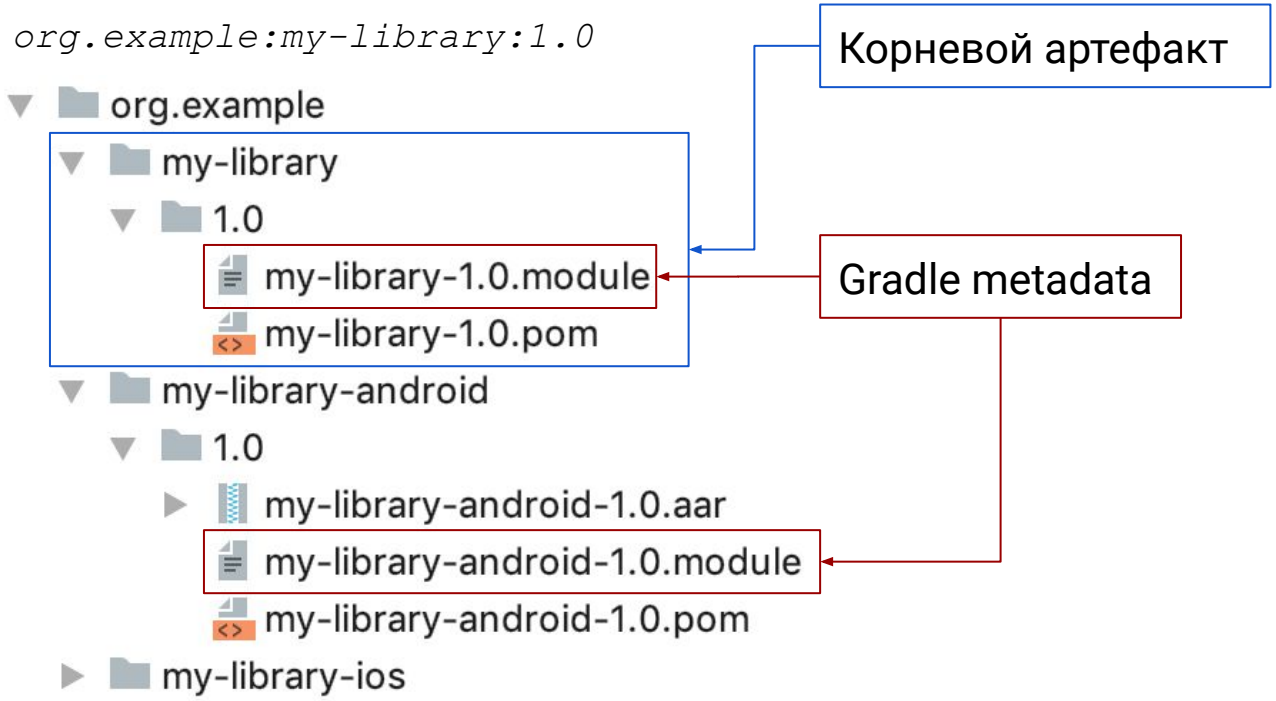


Публикация

org.example:my-library:1.0



Публикация


org.example:my-library:1.0
org.example
my-library
1.0
my-library-1.0.module
my-library-1.0.pom
my-library-android
1.0
my-library-android-1.0.aar
my-library-android-1.0.module
my-library-android-1.0.pom
my-library-ios
Корневой артефакт
Gradle metadata

Gradle metadata

```
"variants": [
    {
        "name": "android-releaseApiElements",
        "attributes": {
            "org.gradle.usage": "java-api",
            "org.jetbrains.kotlin.platform.type": "androidJvm",
            ...
        },
        "available-at": {
            "url": "../../my-library-android/1.0/my-library-android-1.0.module",
            ...
        }
    },
    ...
]
```

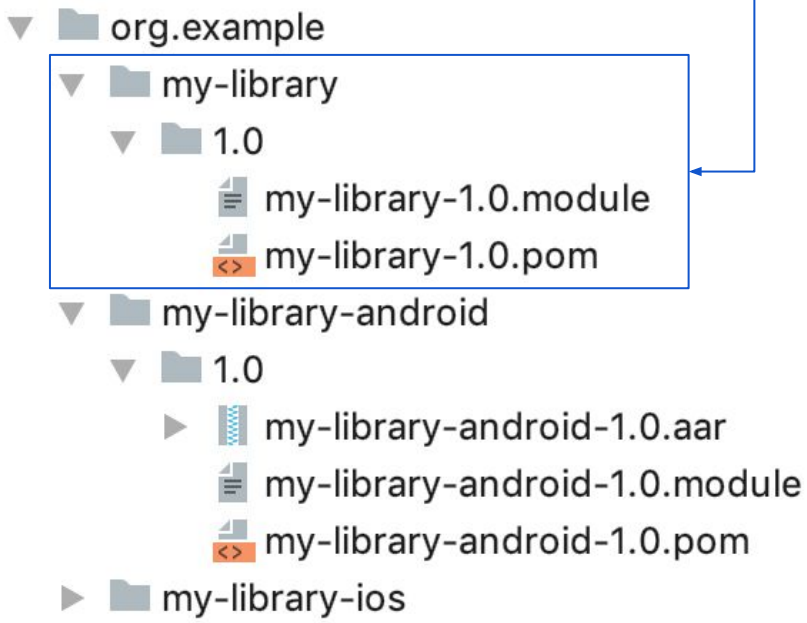
Имя варианта

Атрибуты

Адрес артефакта

Публикация

org.example:my-library:1.0



Корневой артефакт

```
commonMain.dependencies {
    api "org.example:my-library"
}
```

Проблемы модели 1.2

1. ~~Число модулей растет с числом платформ~~
2. ~~Зависимости настраиваются вручную~~

Gradle metadata

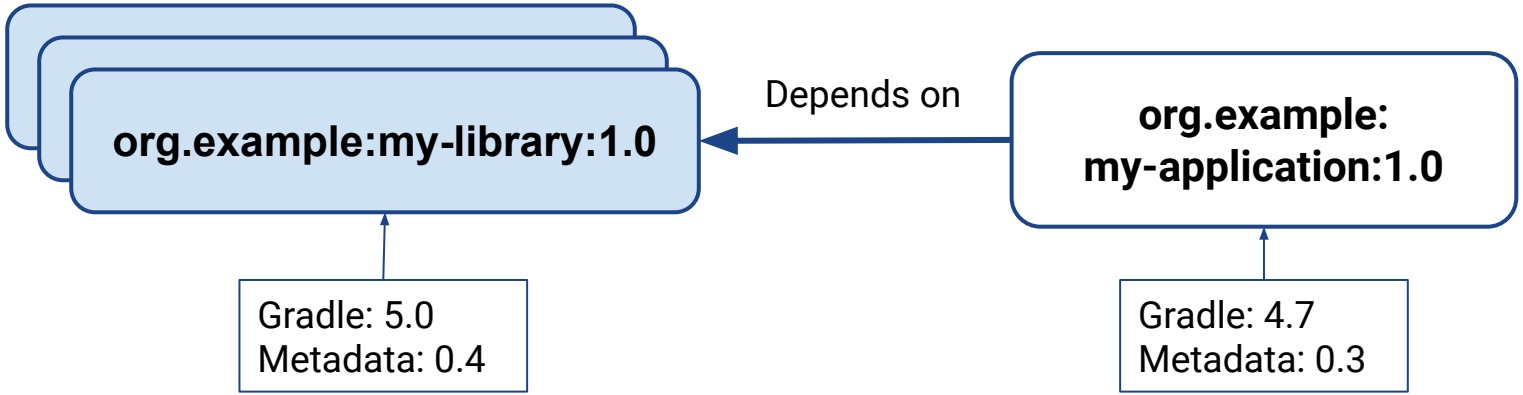
Gradle metadata имеет экспериментальный статус

→

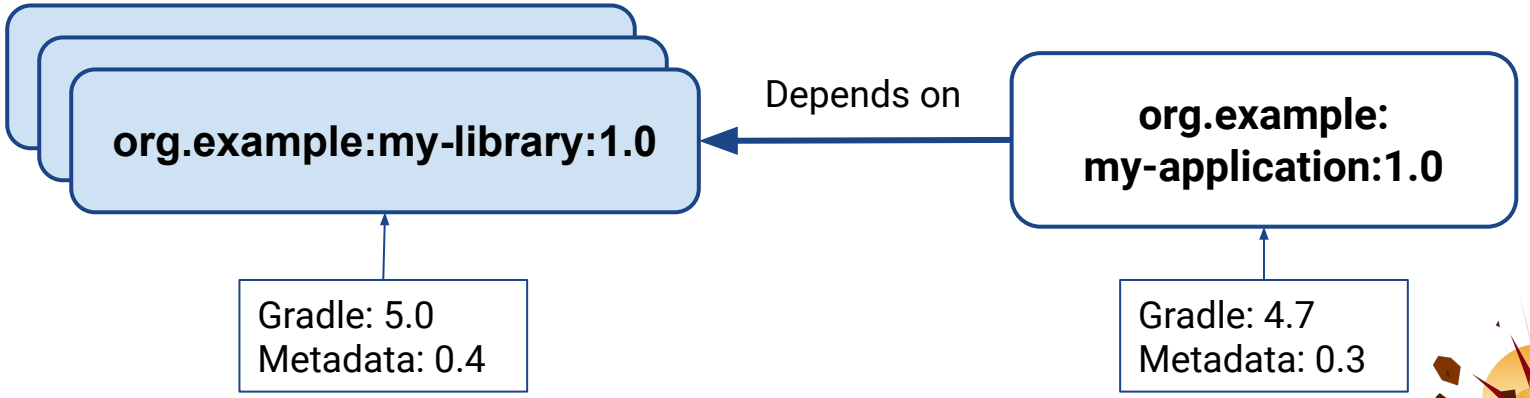
Нет совместимости между разными версиями*

*До версии 5.3: https://docs.gradle.org/5.3/release-notes.html

Gradle metadata



Gradle metadata




```
FAILURE: Build failed with an exception.
...
> Could not parse module metadata my-library-1.0.module
   > Unsupported format version '0.4' specified in module metadata.
     This version of Gradle supports format version 0.3 only.
```

Gradle metadata

1. Не критично для мультиплатформенных проектов
    - Метаданные можно вырубить

Gradle metadata

1. Не критично для мультиплатформенных проектов
   - Метаданные можно вырубить

2. Критично для библиотек, доступных вне MPP

Gradle metadata

1. Не критично для мультиплатформенных проектов
   - Метаданные можно вырубить

2. Критично для библиотек, доступных вне MPP
   - JVM/JS публикуем без метаданных
   - Native - с метаданными

Поддержка в IDE

Поддержка в IDE

1. Поддержан импорт новой проектной модели из Gradle

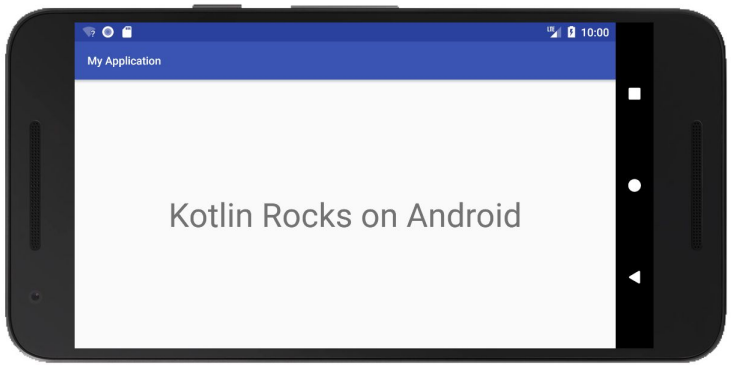
Поддержка в IDE

1. Поддержан импорт новой проектной модели из Gradle
2. Поддержан Kotlin/Native

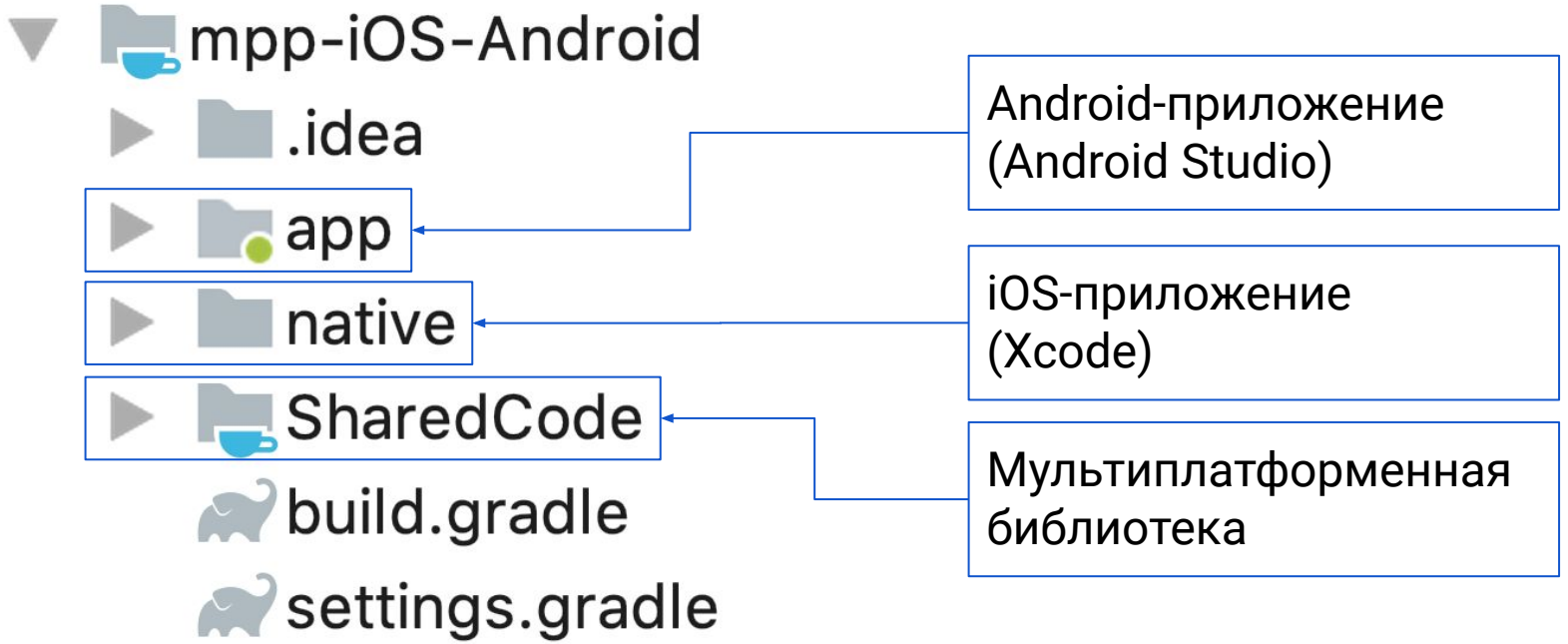
Приложение Android + iOS

Пример: приложение Android + iOS

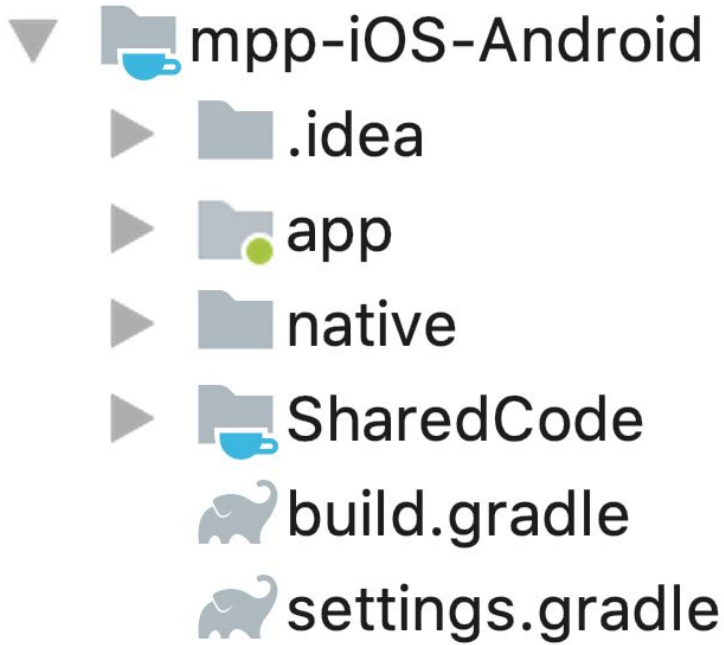
https://kotlinlang.org/docs/tutorials/native/mpp-ios-android.html



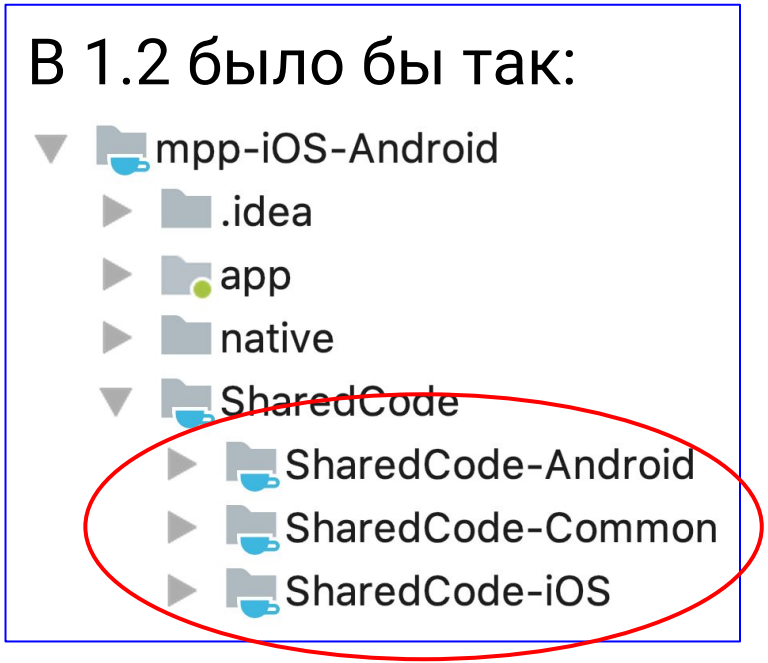
Структура проекта



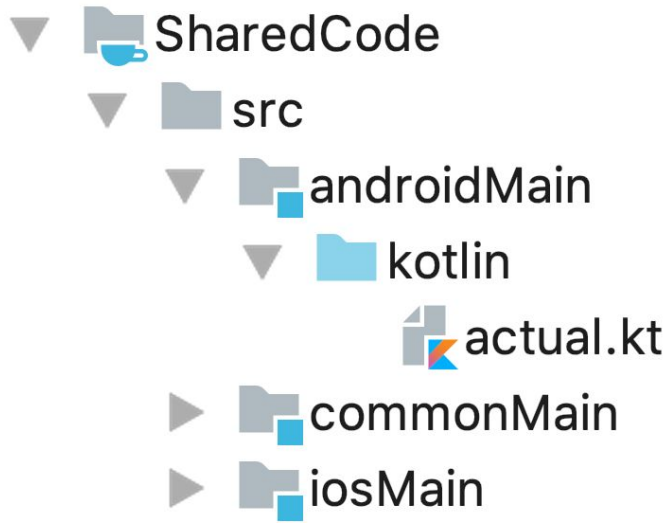
Структура проекта



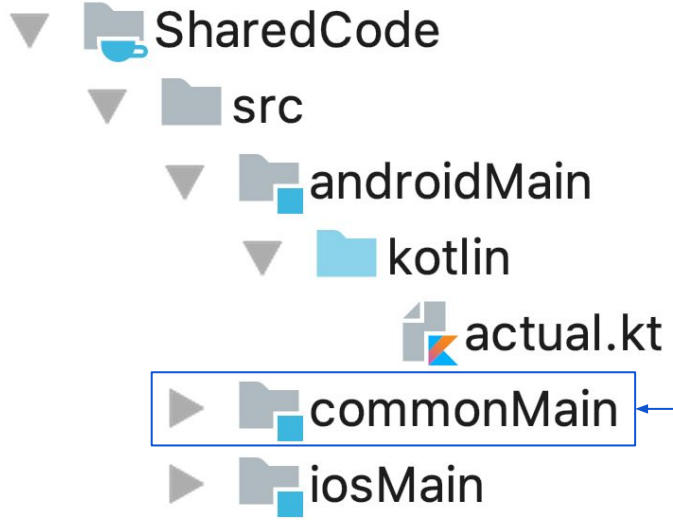
В 1.2 было бы так:



Библиотека SharedCode

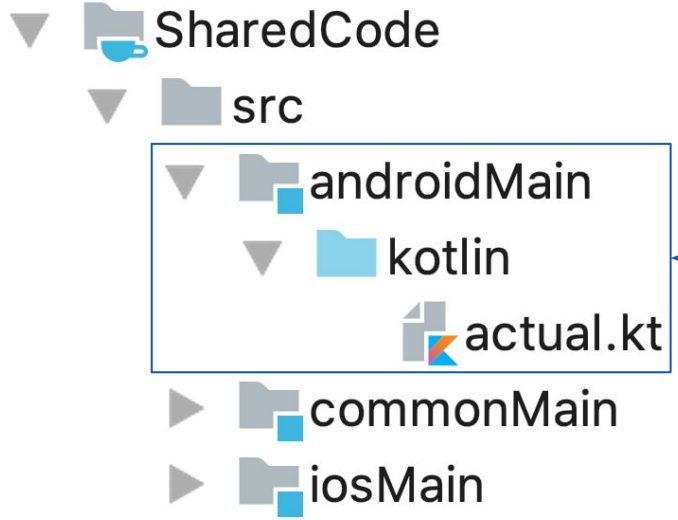


Библиотека SharedCode



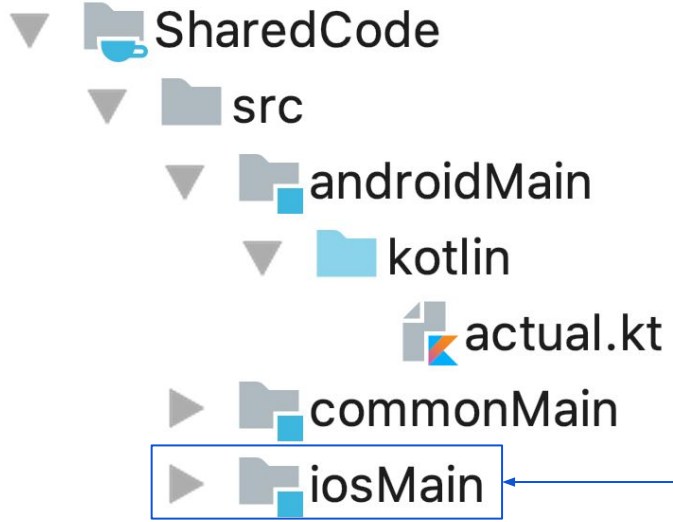

```
expect fun platformName(): String

fun screenMessage(): String =
    "Kotlin Rocks on ${platformName()}"
```

Библиотека SharedCode



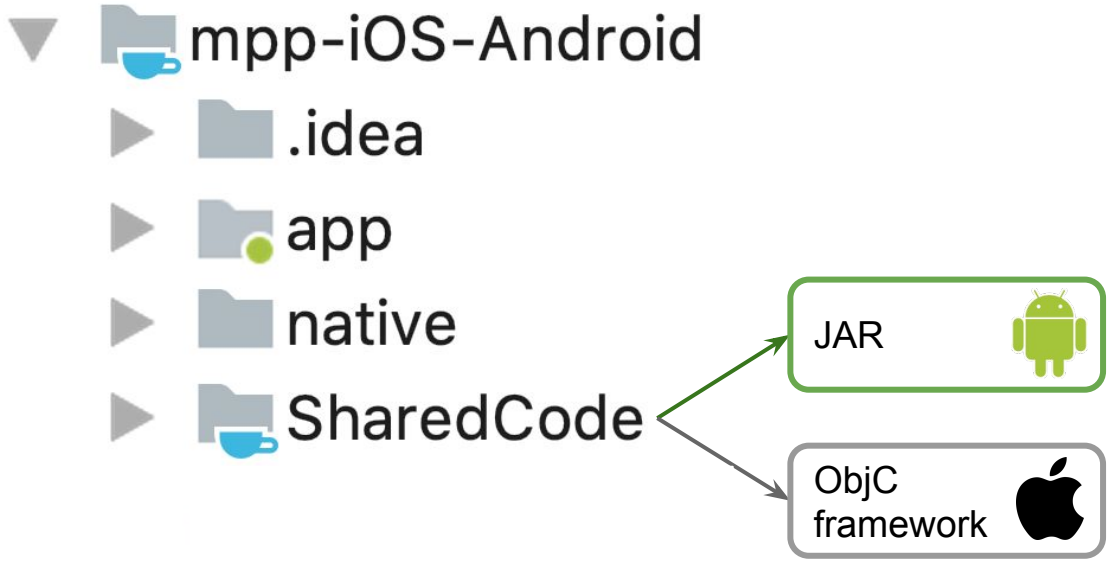

```
actual fun platformName() = "Android"
```

Библиотека SharedCode

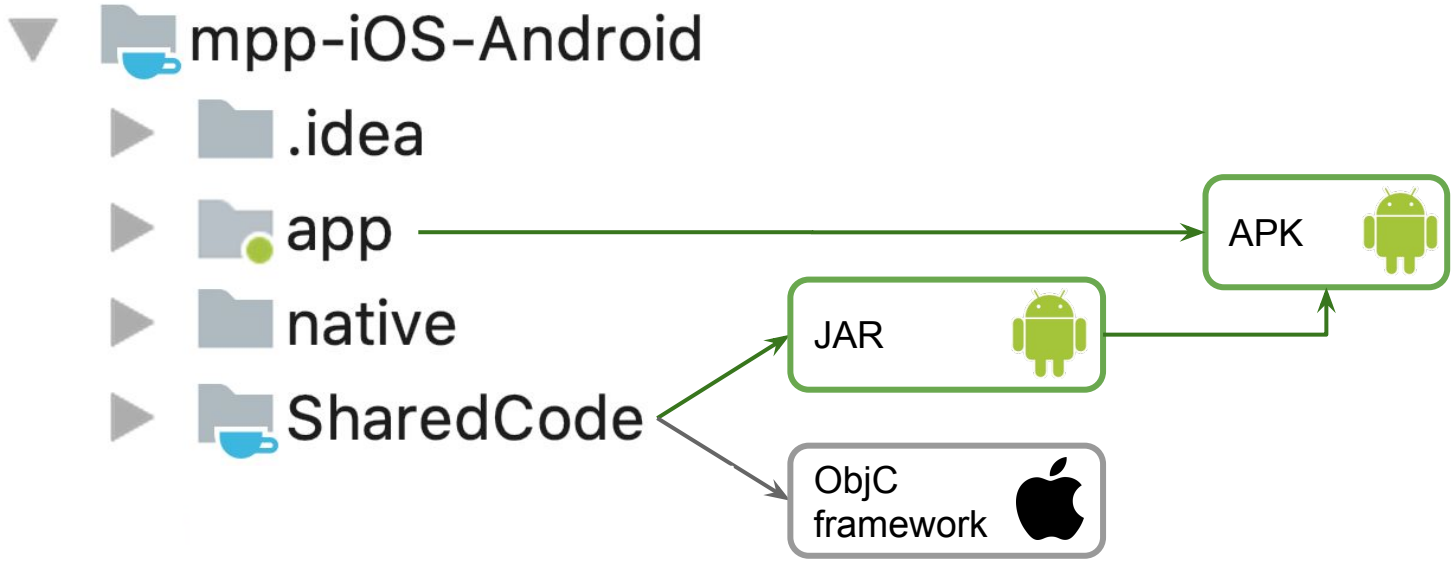



```
// Call iOS platform API.
actual fun platformName() =
    with(UIDevice.currentDevice) {
        "$systemName $systemVersion"
    }
```

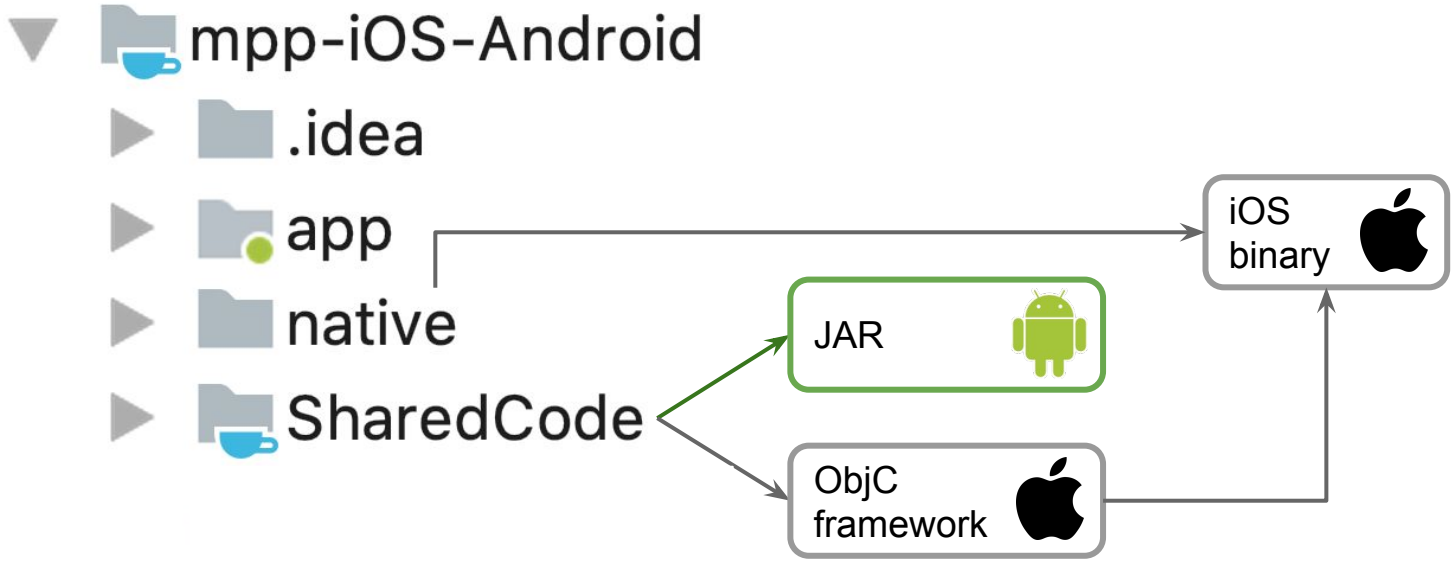
Сборка проекта



Сборка проекта

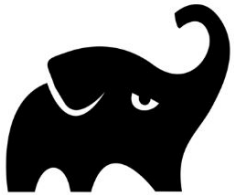


Сборка проекта



Android-приложение

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'

dependencies {
    implementation project(':SharedCode')
}
```

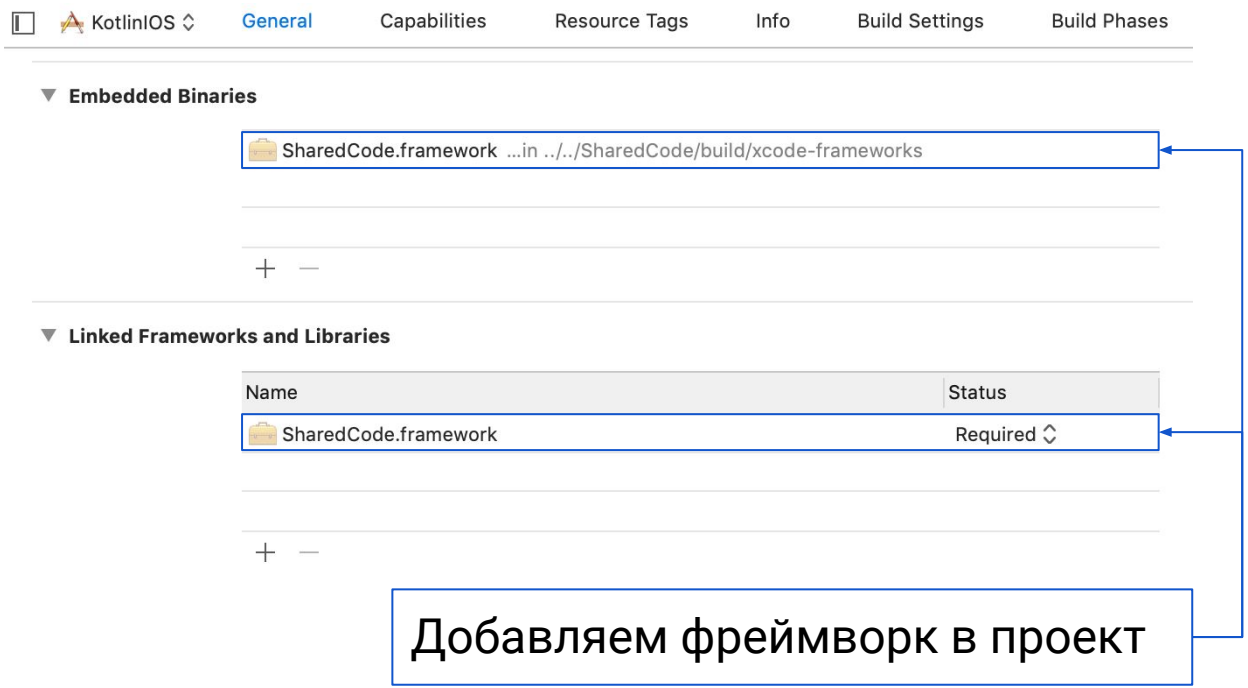


Android-приложение

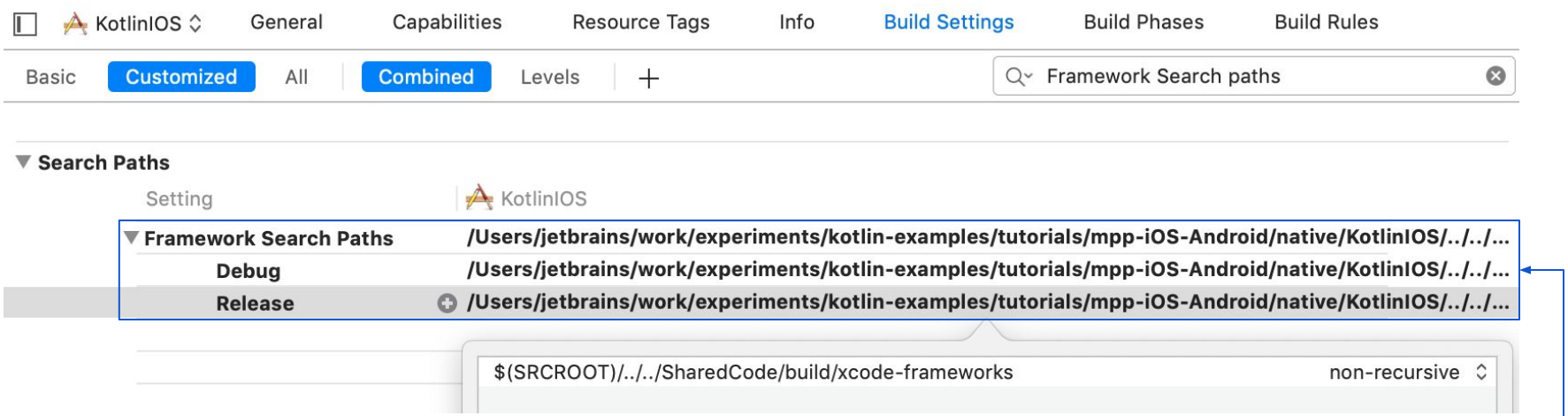
```
import org.kotlin.mpp.mobile.*

// ...

val view = findViewById<TextView>(
        R.id.main_text
)
view.text = screenMessage()
```

iOS-приложение

iOS-приложение

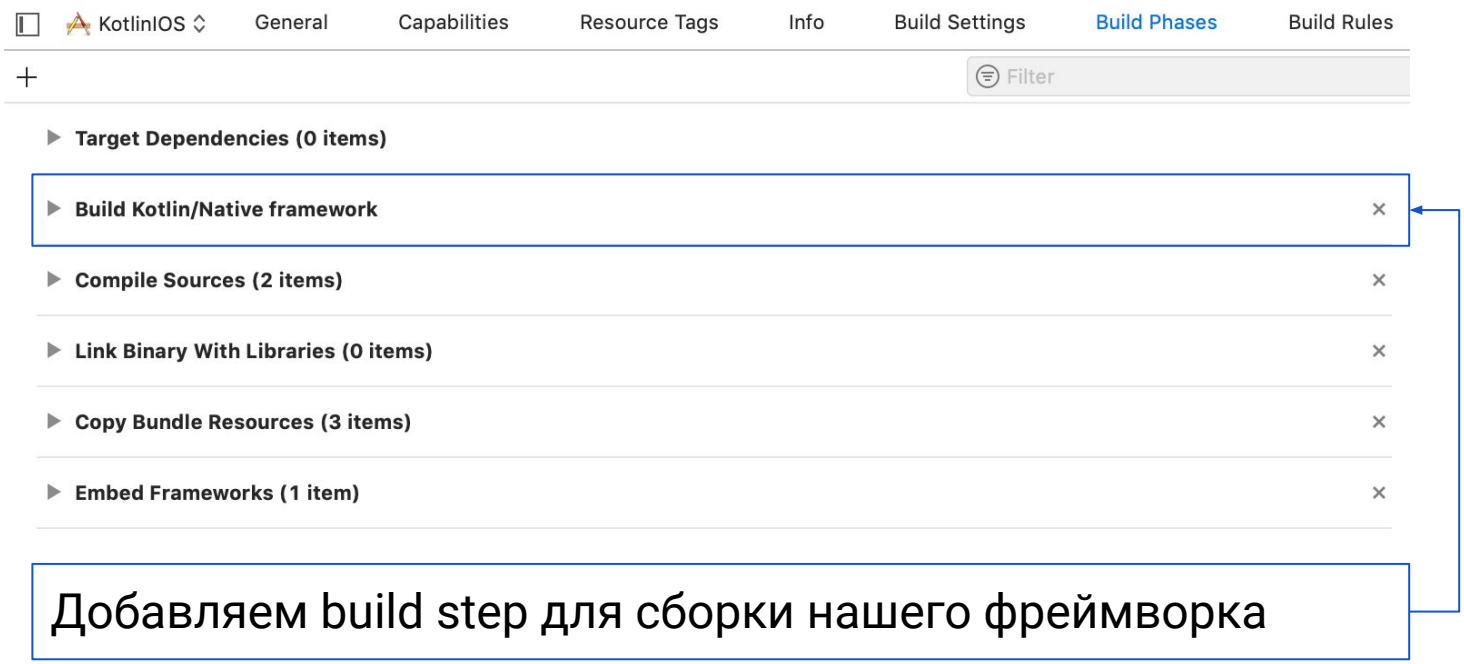


iOS-приложение



Настраиваем пути, чтобы Xcode смог найти наш фреймворк

iOS-приложение



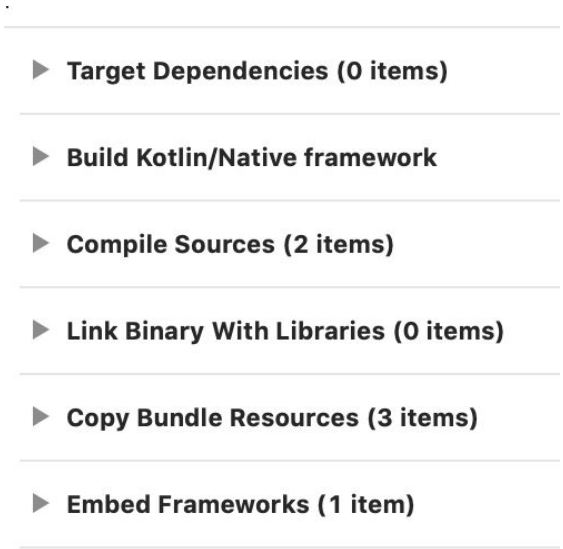
iOS-приложение

Shell /bin/sh

```
1  cd "$SRCROOT/../../SharedCode/build/xcode-frameworks"
2  ./gradlew :SharedCode:packForXCode \
3      -PXCODE_CONFIGURATION="${CONFIGURATION}"
4
```

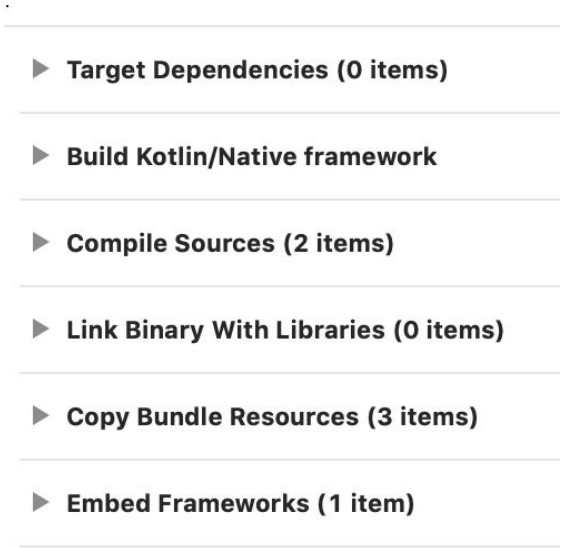
Копируем фреймворк в нужную директорию

iOS-приложение



1. Собираем Kotlin-фреймворк (делегируем в Gradle)
2. Собираем приложение
3. Копируем используемые фреймворки

iOS-приложение



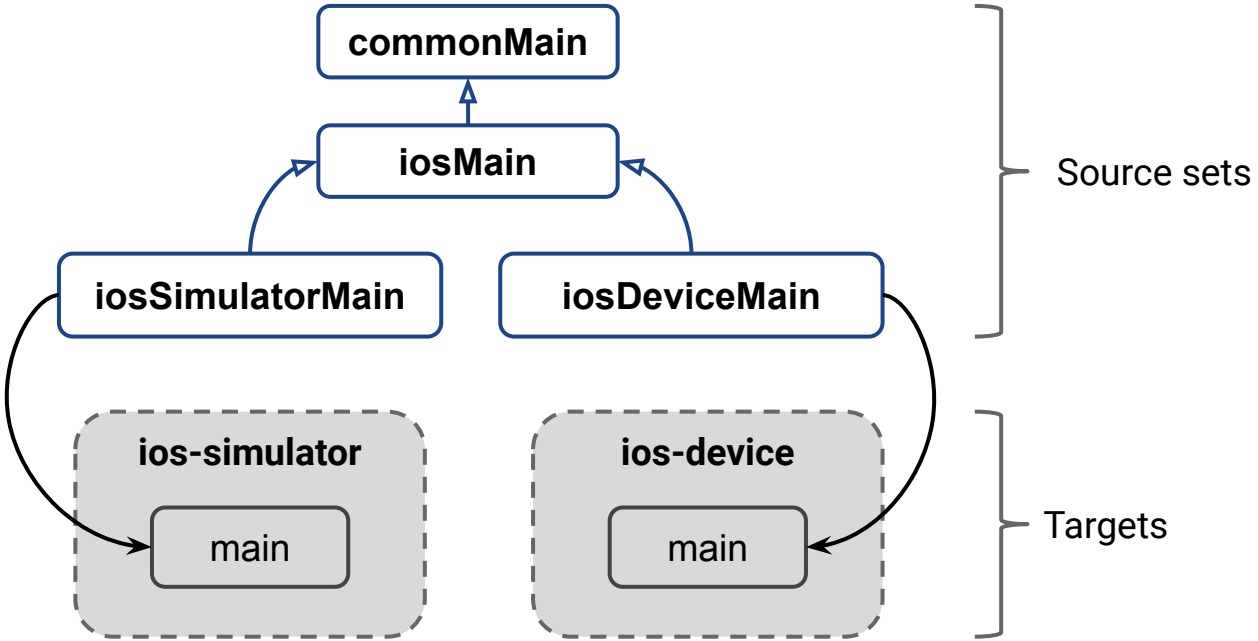
1. Собираем Kotlin-фреймворк (делегируем в Gradle)
2. Собираем приложение
3. Копируем используемые фреймворки

Дефолтные шаги

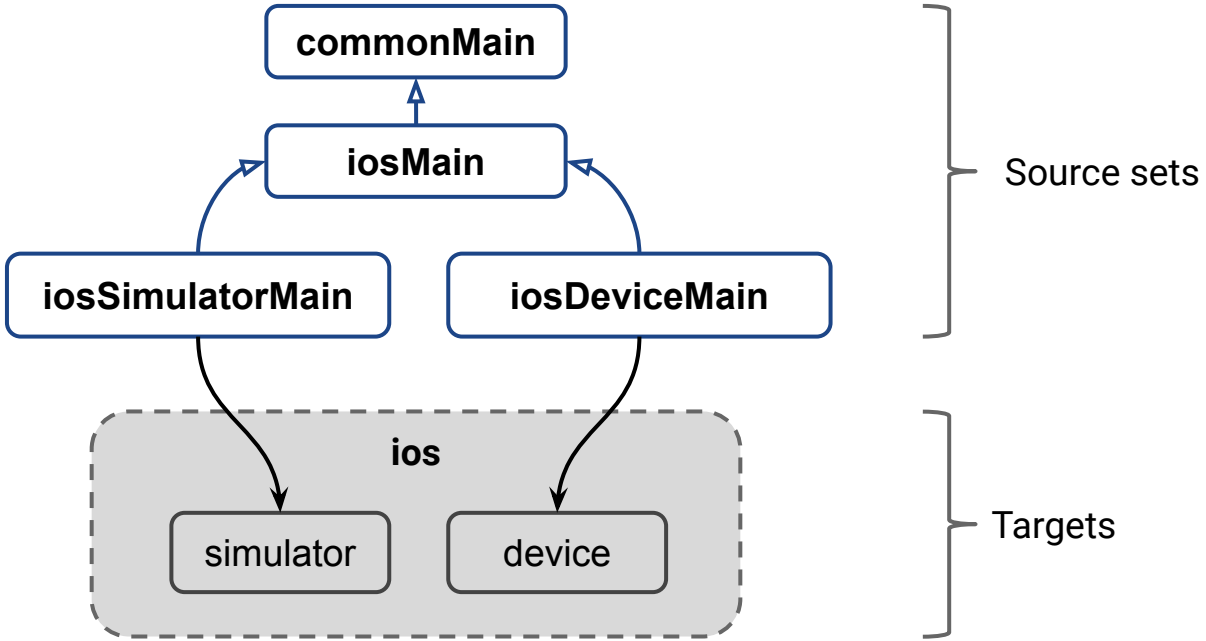
iOS-приложение

```
import SharedCode

// ...

@IBOutlet weak var label: UILabel!

// …

label.text = CommonKt.screenMessage()
```

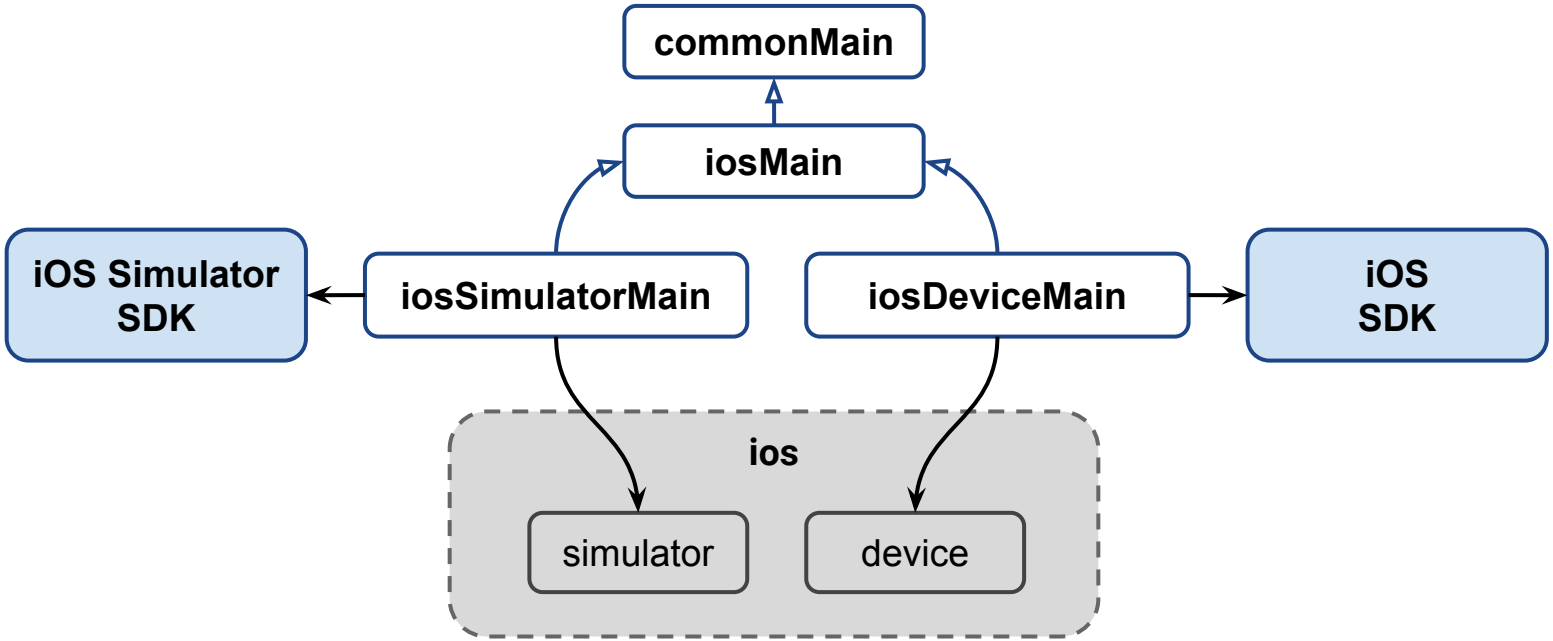
iOS-приложение



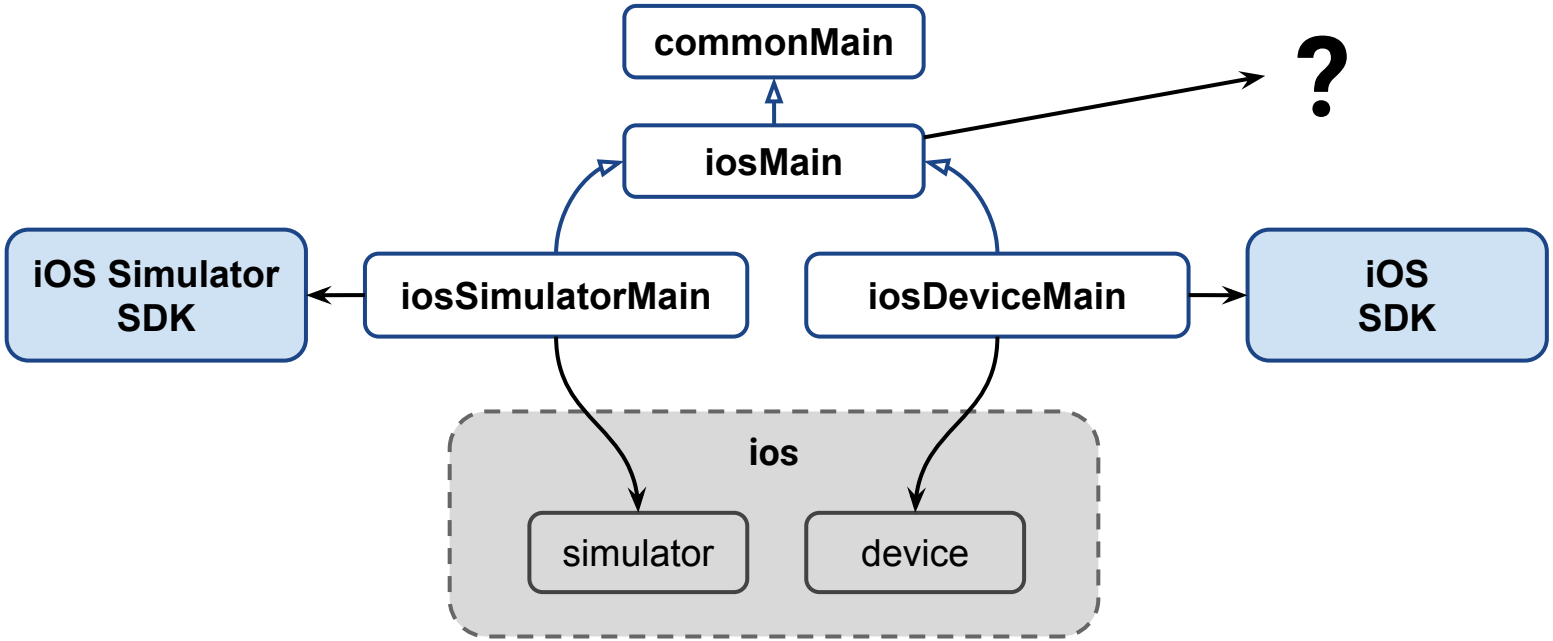
iOS-приложение



iOS-приложение

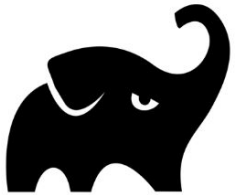


iOS-приложение



iOS-приложение

```
kotlin {
    if (System.getenv('SDK_NAME')?.startsWith("iphoneos")) {
        iosArm64('iOS')
    } else {
        iosX64('iOS')
    }

    targets.iOS { /* Configure iOS. */ }
}
```



Заключение

Больше примеров

- Kotlin/Native samples:
  https://github.com/JetBrains/kotlin-native/tree/master/samples
- Kotlinx-coroutines:
  https://github.com/Kotlin/kotlinx.coroutines/
- Ktor
  https://github.com/ktorio/ktor
- Kotlin/Native обертка для libui
  https://github.com/msink/kotlin-libui

Обратная связь

- Kotlin Slack: kotlinlang.slack.com (канал #multiplatform)
- Баг-трекер: youtrack.jetbrains.com/issues/KT
- GitHub:
  - github.com/JetBrains/kotlin
  - github.com/JetBrains/kotlin-native

Мультиплатформенные проекты в Kotlin 1.3

Матвеев Илья, JetBrains
ilmat192@gmail.com
Mobius 2019 Piter