

# Missing Parts

Павел Галанин  
Весна 2024



Подпись и публикация  
приложения.  
Магазины приложений

• • • •

# Подпись приложения

Android позволяет устанавливать только [подписанные](#) приложения.

Зачем нужна подпись?

1. Для проверки целостности:  
Android не даст обновить приложение, если подписи различаются.
2. Для настроек доступа:
  - можно настроить выдачу разрешений приложениям, подписанным тем же ключом
  - если используется несколько приложений в одном процессе, они все должны быть подписаны одним ключом

Если потерять ключ подписи, приложение нельзя будет обновить, только поставить с нуля. При публикации через маркеты это по сути означает, что нужно зарегистрировать новое приложение.

Не теряйте ключ.



# Виды ключей

## Debug-ключ

Android Studio автоматически генерирует debug-ключ и подписывает им debug-сборку.

Выложить в маркет с debug-ключом нельзя.

Сертификат с debug-ключом живёт 30 лет.

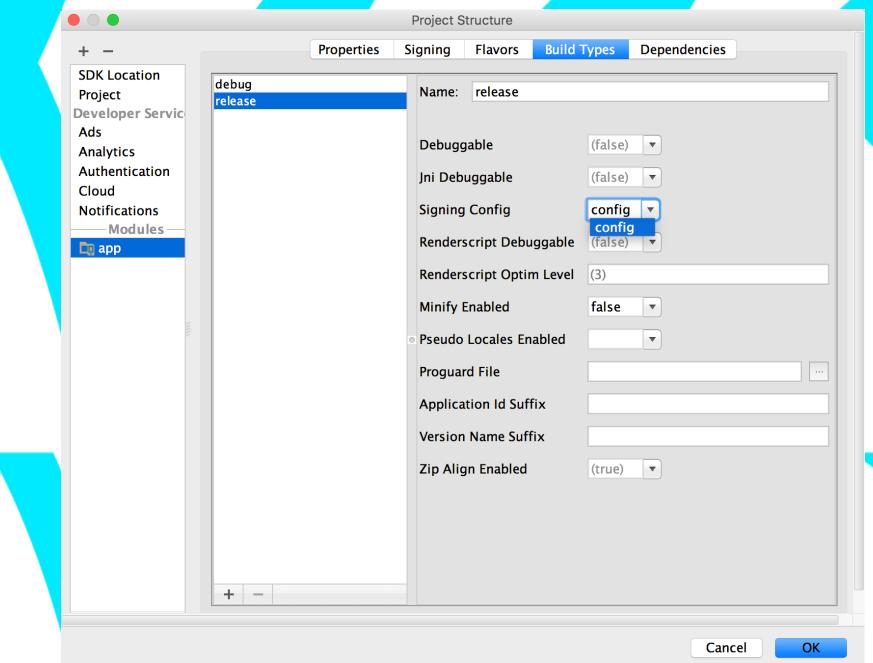
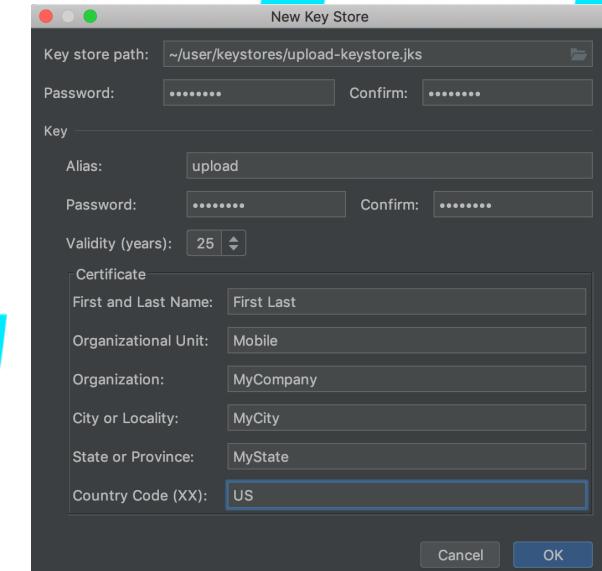
## Release-ключ

С помощью Android Studio можно создать хранилище ключей (keystore) и сам ключ.

Иначе это можно сделать вручную с помощью утилиты [keytool](#).

Подписать приложение можно с помощью Build > Generate Signed Bundle/APK.

Можно настроить автоматическую подпись для release-сборки.



# Конфигурация

Android Studio добавляет информацию о подписи в **открытом** виде в `build.gradle`.

Если планируется где-то публиковать исходный код, стоит вынести эту информацию в локальный `properties`-файл.

```
storePassword=myStorePassword
keyPassword=mykeyPassword
keyAlias=myKeyAlias
storeFile=myStoreFileLocation

...
// Create a variable called keystorePropertiesFile, and initialize it to your
// keystore.properties file, in the rootProject folder.
def keystorePropertiesFile = rootProject.file("keystore.properties")

// Initialize a new Properties() object called keystoreProperties.
def keystoreProperties = new Properties()

// Load your keystore.properties file into the keystoreProperties object.
keystoreProperties.load(new FileInputStream(keystorePropertiesFile))

android {
    signingConfigs {
        config {
            keyAlias keystoreProperties['keyAlias']
            keyPassword keystoreProperties['keyPassword']
            storeFile file(keystoreProperties['storeFile'])
            storePassword keystoreProperties['storePassword']
        }
    }
    ...
}
```

# Google Play App Signing

Когда используется AAB, подпись приложения откладывается до сборки конкретного APK.

Play App Signing использует два ключа:

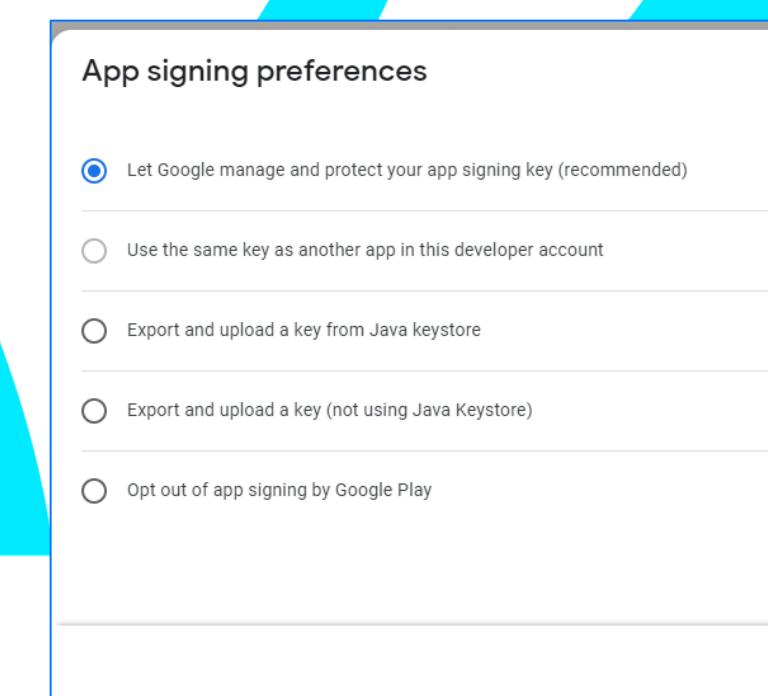
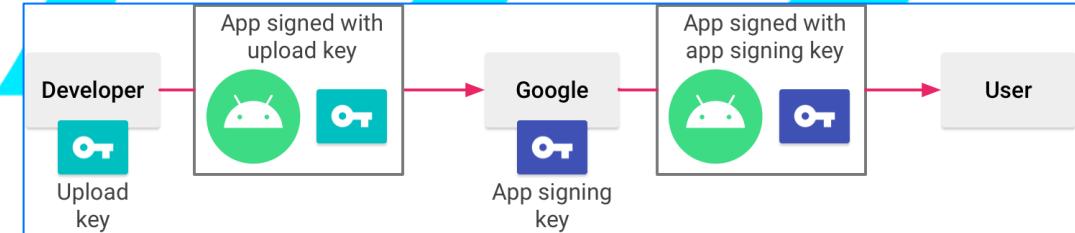
- ключ загрузки ([upload key](#))
- ключ подписи ([signing key](#))

Ключ загрузки создаёт разработчик. Он используется, чтобы загрузить приложение в Google Play. Этот ключ можно заменить без потери возможности обновления приложения.

Создание и использование ключа подписи можно отдать под полное управление Google Play:

- + не нужно тратить силы и время на создание, а главное - на хранение ключа подписи
- не получится обновиться сборкой из другого маркета

Можно использовать один ключ и для загрузки, и для подписи.



# Магазины приложений

От компаний:

- RuStore от VK
- Amazon Appstore

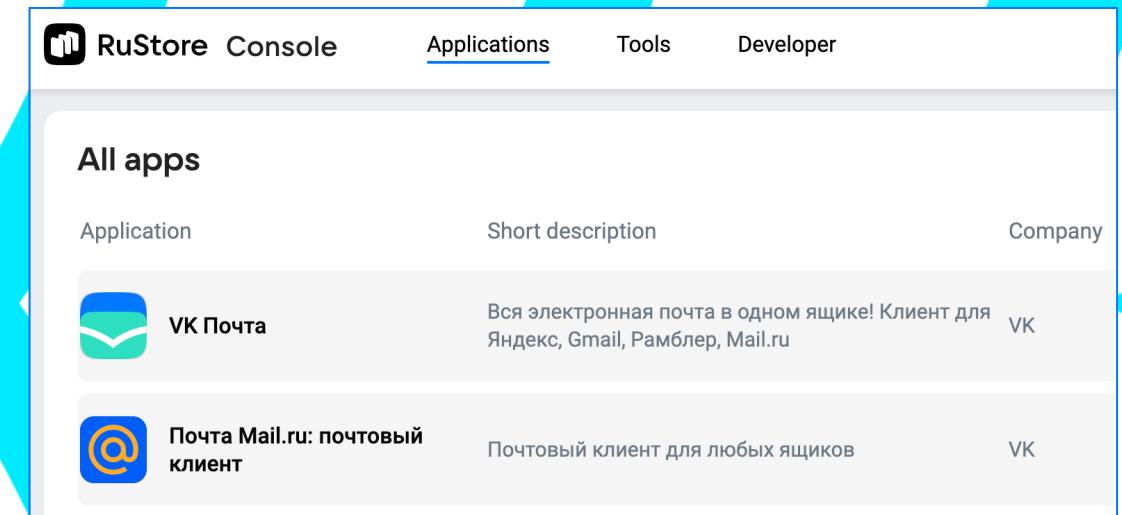
От вендоров:

- Huawei AppGallery
- Samsung Galaxy Store
- Xiaomi GetApps
- Vivo V-Appstore

От сообщества:

- Aptoide
- F-Droid

...



The screenshot shows the RuStore Console interface with a navigation bar at the top. The 'Applications' tab is selected. Below the navigation bar, the text 'All apps' is displayed. A table lists two applications: 'VK Почта' and 'Почта Mail.ru: почтовый клиент'. The table has columns for 'Application', 'Short description', and 'Company'. The 'VK' logo is present in the 'Company' column for both entries.

Application	Short description	Company
 VK Почта	Вся электронная почта в одном ящике! Клиент для Яндекс, Gmail, Рамблер, Mail.ru	VK
 Почта Mail.ru: почтовый клиент	Почтовый клиент для любых ящиков	VK

# Магазины приложений

Большая часть маркетов работает с APK. Некоторые поддерживают AAB, некоторые вводят свои форматы.

Чтобы сохранялась совместимость между приложениями, установленными через разные магазины, приложения должны быть подписаны **одной подписью**.

**Скачанные файлы**

Ссылка для внутреннего доступа к приложению  
Чтобы установить эту версию приложения, перейдите по ссылке на устройстве Android. [Управление доступом](#)  
↪ Копировать ссылку общего доступа

**Объекты**

Файл	Размер скач
Исходный файл	
Подписанный универсальный APK-файл	
Архивированный APK	
Файл сопоставления ReTrace mapping.txt	
Нативные отладочные символы	

**RuStore Консоль** Приложения Инструменты Разработчик

**Почта Mail.ru: почтовый клиент** 14.106.0.66812

Версии / Загрузка новой версии

**Файлы**

Вы можете загрузить несколько файлов, например:  
1. С разными подписями, чтобы исключить ошибки обновления у пользователей. [Подробнее](#)  
2. С Huawei Mobile Services для стабильной работы приложения на устройствах Huawei и Honor, не поддерживающих сервисы Google

Используется для загрузки версий приложения. APK не более 2.5 GB или AAB не более 500 MB [Выберите файл](#)

Подпись не загружена  
Для Android App Bundle требуется подпись, используемая в предыдущей версии. Если у вас её нет, загрузите версию в формате APK. [Подробнее](#)

**Загрузить**

# Согласие на использование данных

- General Data Protection Regulation (GDPR)

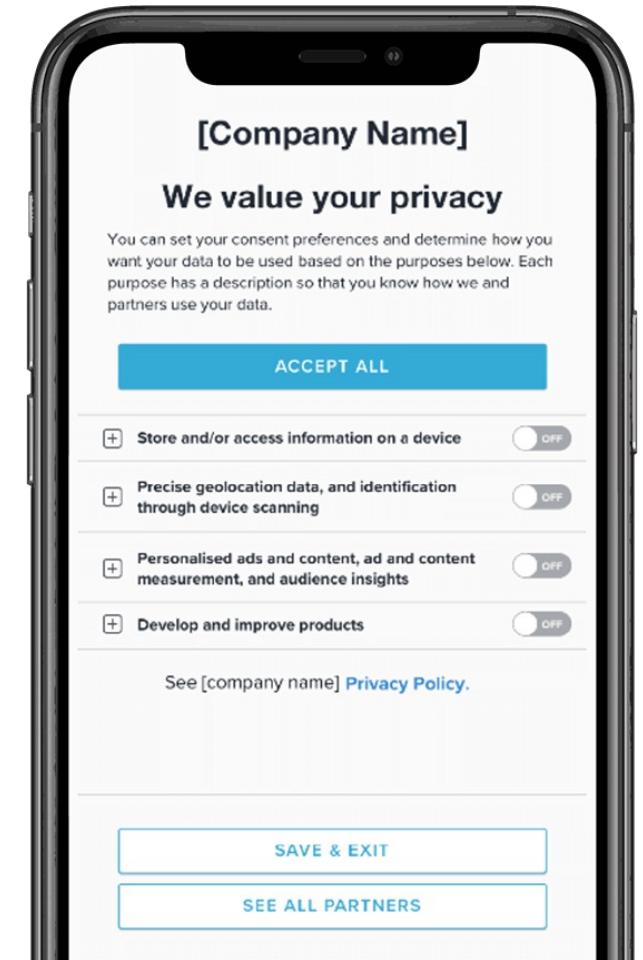
Нужно получить согласие от пользователя на обработку его персональных данных (имя, адрес, данные о личной жизни, финансах, здоровье и т.д.).

Если пользователь не дал согласие на использование персональных данных, нужно, например, в рекламных SDK отключать персонализацию.

- California Consumer Privacy Act (CCPA) - для жителей Калифорнии.

- Children's Online Privacy Protection Act (COPPA) - действует на сервисы для детей до 13 лет

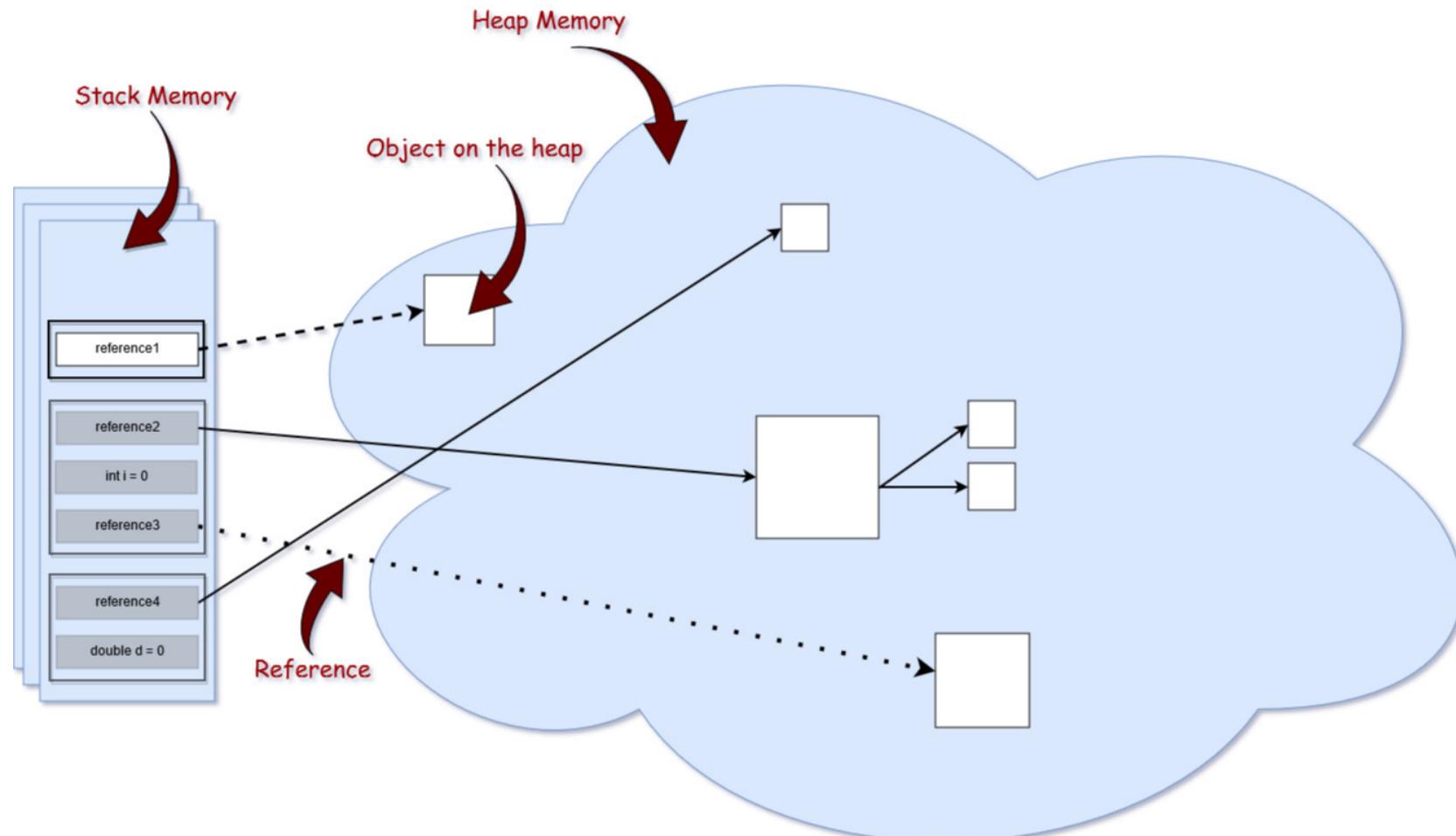
Можно подключить библиотеку, которая предоставляет диалог согласия (например <https://github.com/TrackerControl/auto-app-consent>)



# Работа с памятью в Android. Утечки памяти

• • • •

# Память Java-процесса

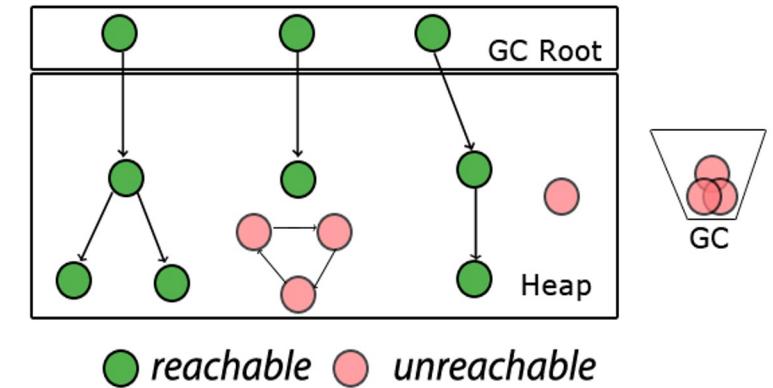
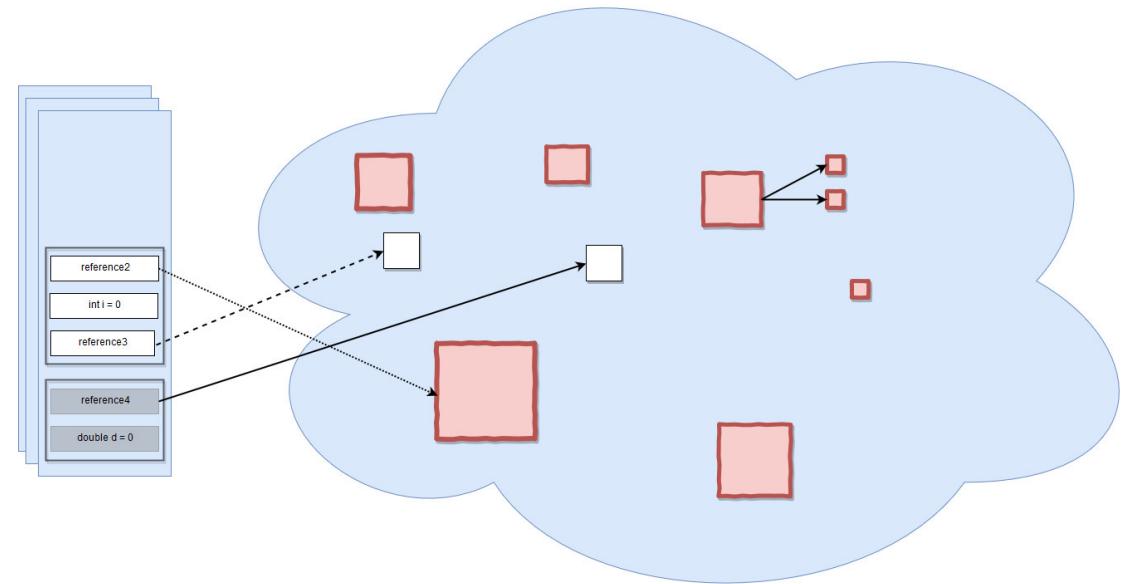


# Мусор

Корень (GC root):

- локальные переменные
- активные потоки и их стек
- статические переменные
- объекты, к которым есть ссылка из нативных методов JNI
- в случае Android экземпляр Application

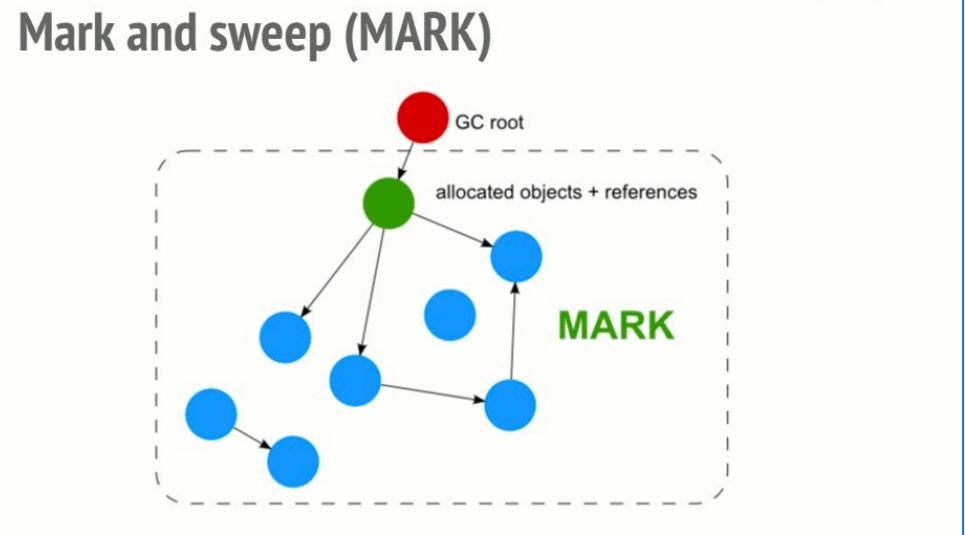
Все, что недоступно из корня, считается **мусором**.



# Сборка мусора

Как правило, сборка мусора состоит из этапов:

1. **Mark**: помечаются неиспользуемые объекты.
  2. **Sweep** («подметать»): объекты удаляются из памяти.
  3. **Compact** - дефрагментация - перемещение объектов с целью «уплотнения» памяти.



# Поколения

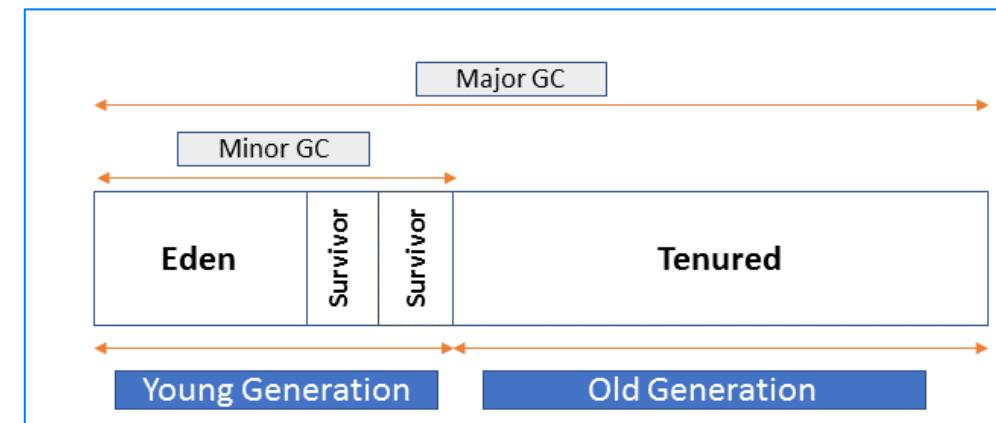
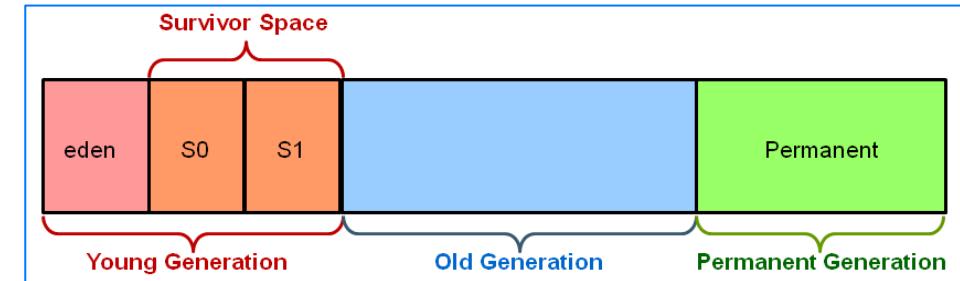
Теория поколений:

Большая часть объектов - короткоживущие. Если объект был создан недавно, с большой вероятностью он скоро станет мусором.

На этом строится оптимизация с поколениями:

Если объект «пережил» сборку мусора, он перемещается в более «взрослое» поколение.

При сборке мусора в первую очередь проверяются молодые поколения. Обращение ко взрослым поколениям выполняется, когда после сборки мусора в молодых поколениях освободилось недостаточно памяти.



# Сборщики

**Serial GC**: однопоточный, Stop-The-World (STW).

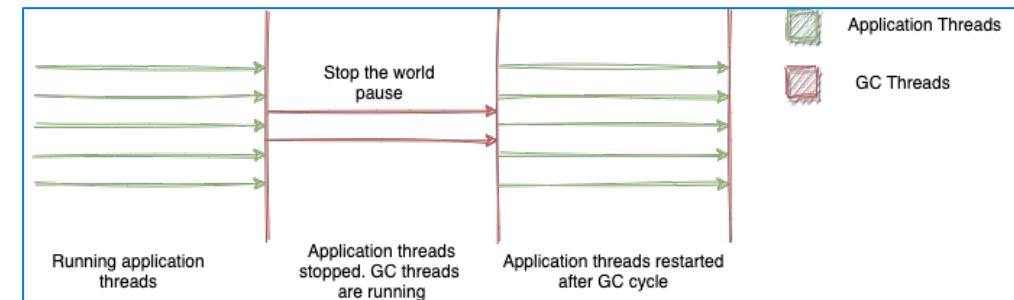
**Parallel GC**: выполняет сборку в несколько потоков в STW.

Concurrent - часть сборки мусора параллельно с приложением:

- **CMS** (Concurrent Mark Sweep) - параллельно с работой приложения отслеживает живые объекты, это снижает время пауз STW (но потребляет ресурсы)
- **G1** - делит память на большое количество регионов; к моменту сборки знает, в каких регионах больше всего мусора - и сперва очищает их (garbage first)
- более новые **ZGC, Shenandoah GC** - ещё меньше паузы STW

**Epsilon GC**: работает без STW, не потребляет ресурсов приложения во время работы. Но есть нюанс...

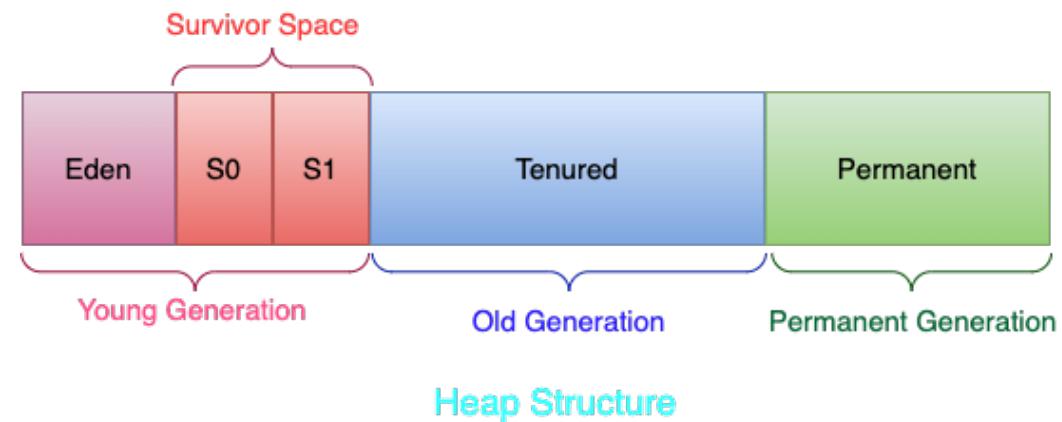
Epsilon GC вообще не собирает мусор: просто завершает работу приложения сразу, как заканчивается память.



# GC в Android

История:

- до KitKat: Dalvik GC - “Stop the World GC”
- Lollipop и Marshmallow: ART GC - “Generational GC”
- Nougat: ART GC - переписали на Assembler
- Oreo: ART GC - “Concurrent Copying GC” без поколений
- с Android 10: ART GC - “Concurrent Copying GC“ с поколениями



Minor GC - сборка мусора в Young Generation

Major GC - сборка мусора в Old Generation

# Виды ссылок

- по умолчанию создаётся обычная ссылка: [строгая](#), [сильная](#)
- [WeakReference](#) - объект будет собран GC, если на него нет других строгих ссылок
- [SoftReference](#) - объект не будет собран GC, пока памяти достаточно
- [PhantomReference](#) - используется для планирования посмертных действий по очистке, в основном внутри платформы

# Утечка

Утечка: объект, который фактически уже не используется в программе, но не может быть собран GC.

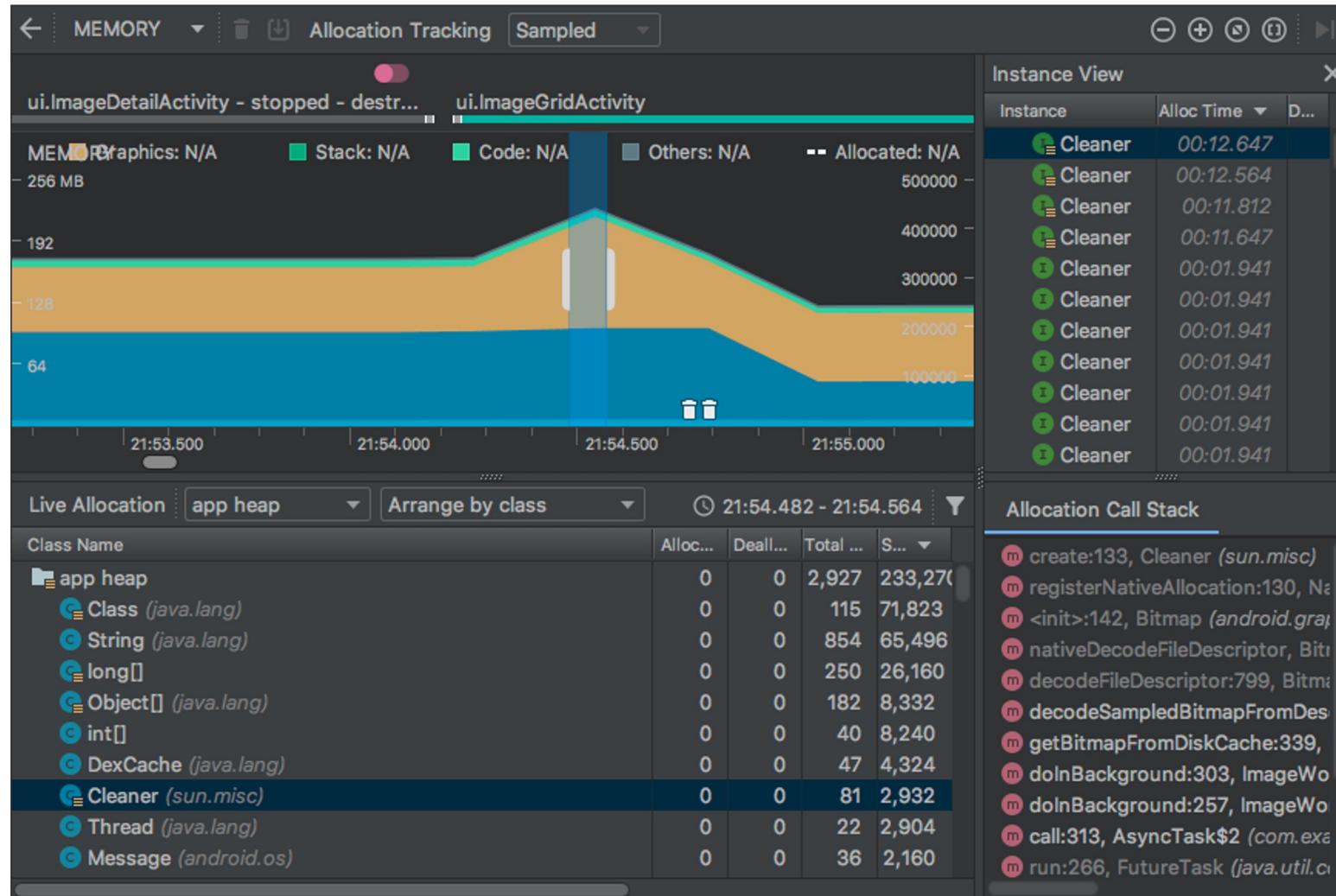
Утечки могут привести к [OutOfMemoryError](#).

Причины утечек и ООМ:

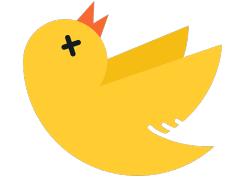
- хранение ссылки на контекст/активити/фрагмент и т.д. дольше времени жизни объекта
- неаккуратная работа со статическими полями
- отсутствие отписки слушателей, колбеков
- неправильное использование внутренних классов
- неправильная работа с битмапами
- отсутствие закрытия/очистки ресурсов (например файловых дескрипторов)
- неправильный интероп с Java (лямбы)

```
----- beginning of crash
05-28 14:45:42.796 29710-29710/buggycompany.com.buggyapp E/AndroidRuntime: FATAL EXCEPTION: main
Process: buggycompany.com.buggyapp, PID: 29710
java.lang.OutOfMemoryError: Failed to allocate a 2095496 byte allocation with 1762016 free bytes and 1720KB until OOM
    at java.util.Arrays.copyOf((Arrays.java:3352)
    at java.lang.AbstractStringBuilder.expandCapacity(AbstractStringBuilder.java:130)
    at java.lang.AbstractStringBuilder.ensureCapacityInternal(AbstractStringBuilder.java:114)
    at java.lang.AbstractStringBuilder.append(AbstractStringBuilder.java:417)
    at java.lang.StringBuilder.append(StringBuilder.java:133)
    at buggycompany.com.buggyapp.MainActivity$1@1$1.run(MainActivity.java:49)
    at android.os.Handler.handleCallback(Handler.java:751)
    at android.os.Handler.dispatchMessage(Handler.java:95)
    at android.os.Looper.loop(Looper.java:154)
    at android.app.ActivityThread.main(ActivityThread.java:6119) <1 internal call>
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:886)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:776)
```

# Как искать причину: Memory Profiler



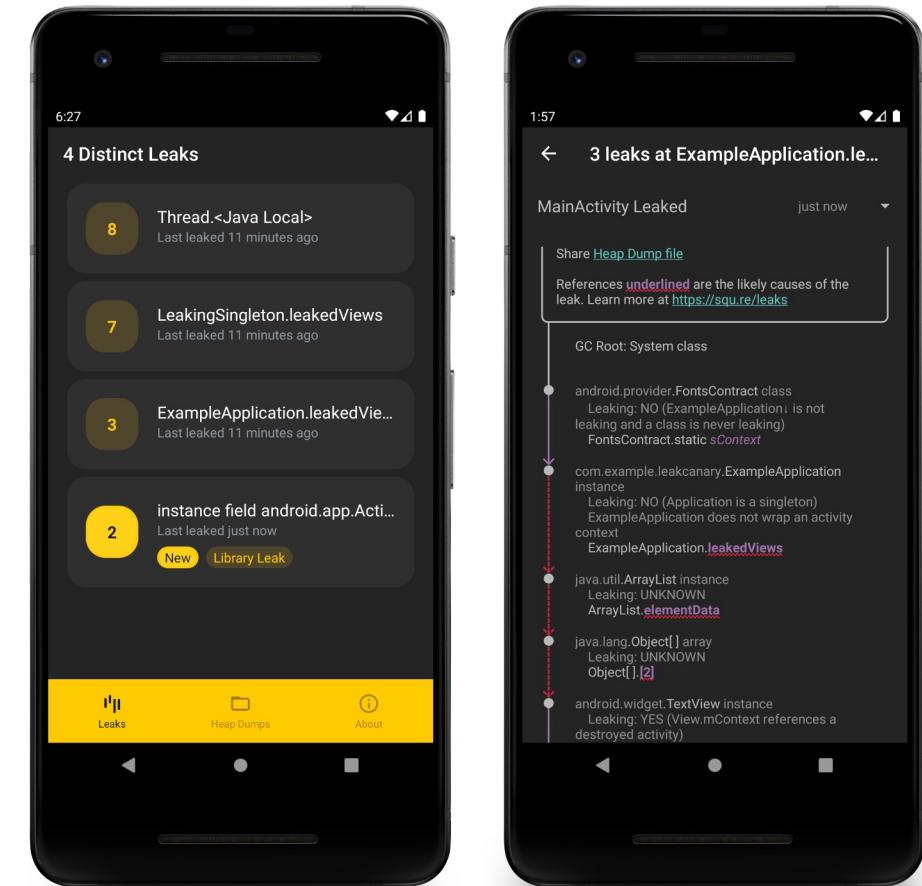
# Как искать причину: LeakCanary



- разработана Square
- подключение в приложение:

```
dependencies {  
    // debugImplementation because LeakCanary should only run in debug builds.  
    debugImplementation 'com.squareup.leakcanary:leakcanary-android:2.14'  
}
```

- будет анализировать ссылки во время работы приложения



# Как исправлять утечку

- избавляться от лишних ссылок
- вовремя очищать ресурсы
- переиспользовать ресурсы (пулы)
- WeakReference



# Работа с native-кодом: NDK, JNI

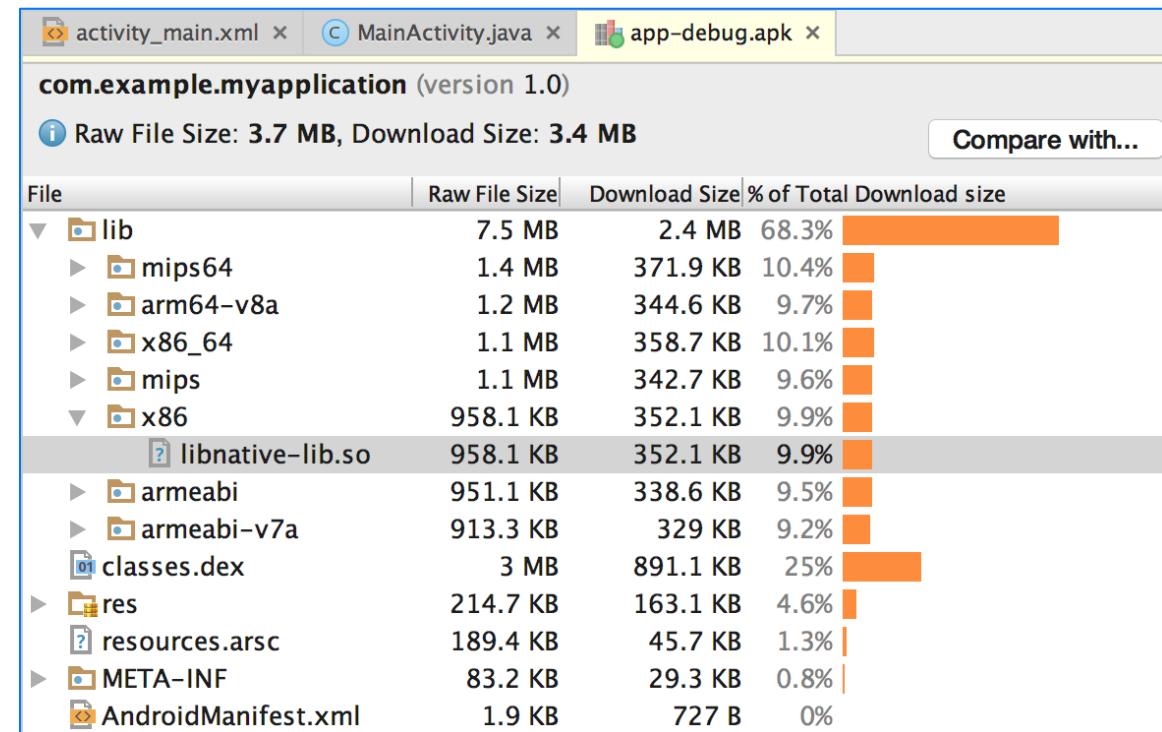
• • • •

# Android NDK

**NDK** (Native Development Kit) - средства для разработки под Android на C/C++ и для взаимодействия с физическими устройствами.

**JNI** (Java Native Interface) - интерфейс взаимодействия между Java/Kotlin-кодом и кодом на C/C++.

Под каждую поддерживаемую платформу собирается своя библиотека.



# Зачем?



- может дать ускорение в вычислениях для критичных по производительности участков кода
- позволяет использовать существующие библиотеки (TensorFlow Lite, OpenGL ES, FFmpeg etc.)
- можно написать общую логику на разные платформы

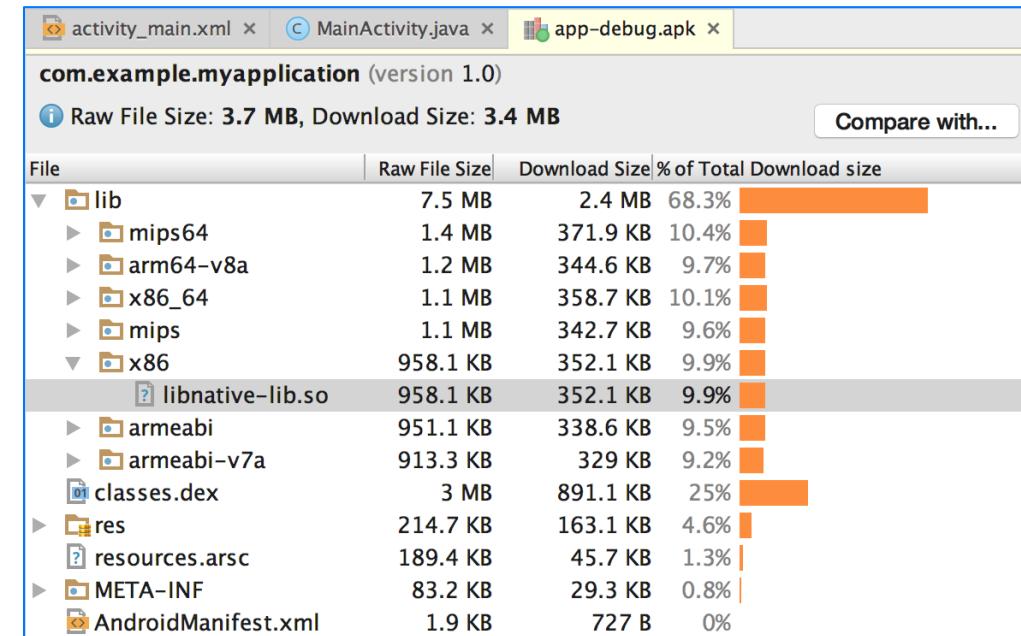


TensorFlow



# Проблемы

- много нюансов при многопоточной работе
- сложная отладка
- увеличение размера APK
- усложнение работы с памятью, потенциальный источник утечек
- проблемы с производительностью при переходе в нативный код - при передаче данных (маршаллинг)
- C/C++ (может быть и плюсом)



# Разработка нативной библиотеки

Для native-разработки понадобится:

1. NDK
2. [CMake](#) - для сборки нативной библиотеки
3. LLDB - для отладки нативного кода



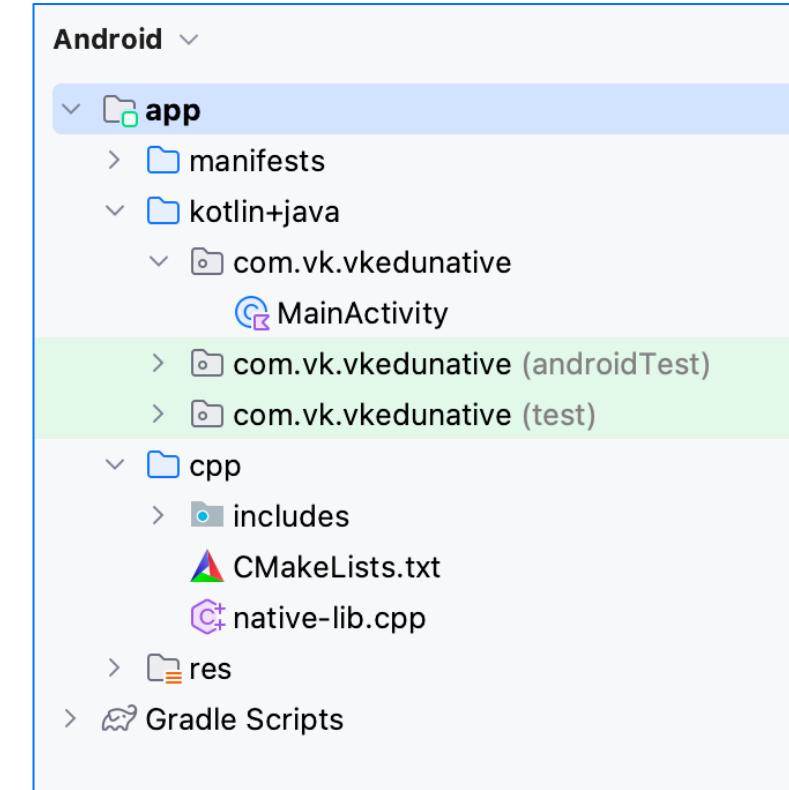
CMake позволяет собирать:

- [shared](#)-библиотеки (.so) - подключаются в рантайме через `System.loadLibrary`
- [статические](#) библиотеки (.a) - подключаются к динамическим библиотекам на этапе компиляции

# Добавление в проект

Native C++ project / New > C/C++ Source File

```
externalNativeBuild { this: ExternalNativeBuild
    cmake { this: Cmake
        path = file( path: "src/main/cpp/CMakeLists.txt")
        version = "3.22.1"
    }
}
```



# CMakeLists.txt

- **add\_library** - добавляет библиотеку в проект: указывается тип (shared, static) и файлы с исходным кодом
- **find\_library** - добавляет собранную библиотеку в проект (liblog, libGLESv3, libvulkan)
- **target\_link\_libraries** - подключает одну нативную библиотеку к другой

```
# Declares the project name. The project name can be accessed via ${PROJECT_NAME},  
# Since this is the top level CMakeLists.txt, the project name is also accessible  
# with ${CMAKE_PROJECT_NAME} (both CMake variables are in-sync within the top level  
# build script scope).  
project("vkedunative")  
  
add_library(${CMAKE_PROJECT_NAME} SHARED  
            native-lib.cpp)  
  
target_link_libraries(${CMAKE_PROJECT_NAME}  
                      android  
                      log)
```

lib**library-name**.so

# Код

```
#include <jni.h>
#include <string>

extern "C" JNIEEXPORT jstring JNICALL
Java_com_vk_vkedunative_MainActivity_stringFromJNI(
    JNIEnv* env,
    jobject MainActivity /* this */) {
    std::string hello = "Hello from C++";
    return env->NewStringUTF( bytes: hello.c_str());
}
```

```
/*
 * A native method that is implemented by the 'vkedunative' native library,
 * which is packaged with this application.
 */
external fun stringFromJNI(): String

companion object {
    // Used to load the 'vkedunative' library on application startup.
    init {
        System.loadLibrary( libname: "vkedunative")
    }
}
```

# JNIEnv

JNIEnv - указатель на таблицу функций - через него можно вызывать функции, доступные в JNI:

- создание объектов, массивов объектов (NewObject, New\*Array)
- работа с объектами (Call\*Method, Get\*Field, Set\*Field)
- сравнение объектов (isSameObject)
- работа со ссылками из нативного кода (NewGlobalRef, DeleteGlobalRef, DeleteLocalRef)
- работа с потоками (AttachCurrentThread, DetachCurrentThread)
- JNI\_OnLoad, JNI\_OnUnload

JNIEnv предназначен для использования в рамках одного потока - не стоит переиспользовать его в других потоках.

```
extern "C" JNIEXPORT jstring JNICALL
Java_com_vk_vkedunative_MainActivity_stringFromJNI(
    JNIEnv* env,
    jobject MainActivity /* this */) {
    std::string hello = "Hello from C++";
    return env->NewStringUTF( bytes: hello.c_str());
}
```

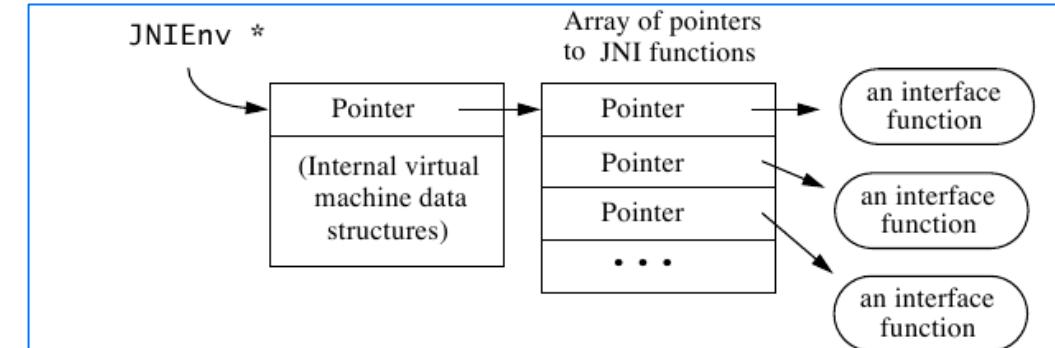


Figure 3.1 The JNIEnv Interface Pointer

# Типы данных

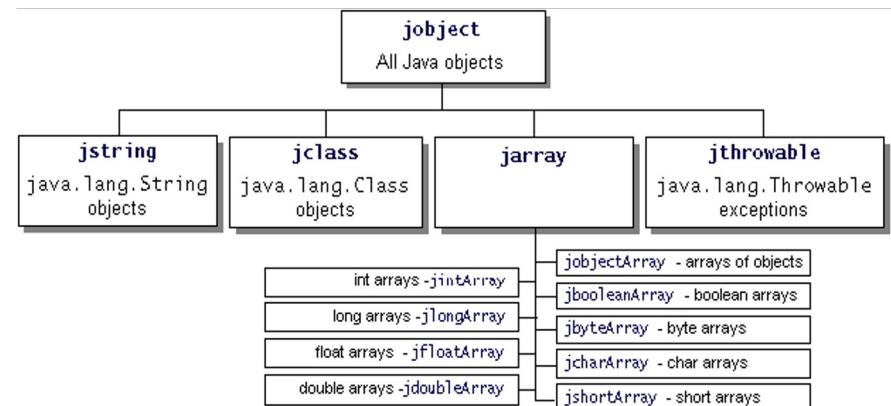
Объекты - `jobject`. Подтипы:

- `jclass` for `Class`
- `jstring`
- `jthrowable`
- `jarray` (для примитивов `jintArray`, `jshortArray`, ...)

Строки:

- `GetStringUTFChars` - сконвертировать JNI-строку в нативный массив символов (после использования нужно освободить ресурсы вызовом `ReleaseStringUTFChars`)
- `NewStringUTF` - создать JNI-строку

Java type	Native type	Native array type	Type code	Array type code
<code>boolean</code>	<code>jboolean</code>	<code>jbooleanArray</code>	<code>Z</code>	<code>[Z</code>
<code>byte</code>	<code>jbyte</code>	<code>jbyteArray</code>	<code>B</code>	<code>[B</code>
<code>char</code>	<code>jchar</code>	<code>jcharArray</code>	<code>C</code>	<code>[C</code>
<code>double</code>	<code>jdouble</code>	<code>jdoubleArray</code>	<code>D</code>	<code>[D</code>
<code>float</code>	<code>jfloat</code>	<code>jfloatArray</code>	<code>F</code>	<code>[F</code>
<code>int</code>	<code>jint</code>	<code>jintArray</code>	<code>I</code>	<code>[I</code>
<code>long</code>	<code>jlong</code>	<code>jlongArray</code>	<code>J</code>	<code>[J</code>
<code>Short</code>	<code>jshort</code>	<code>jshortArray</code>	<code>S</code>	<code>[S</code>
<code>Object</code>	<code>jobject</code>	<code>jobjectArray</code>	<code>L</code>	<code>[L</code>
<code>String</code>	<code>jstring</code>	<code>N/A</code>	<code>L</code>	<code>[L</code>
<code>Class</code>	<code>jclass</code>	<code>N/A</code>	<code>L</code>	<code>[L</code>
<code>Throwable</code>	<code>jthrowable</code>	<code>N/A</code>	<code>L</code>	<code>[L</code>
<code>void</code>	<code>void</code>	<code>N/A</code>	<code>V</code>	<code>N/A</code>



# Работа с объектами

Вызов метода:

1. получение ссылки на класс через GetObjectClass
2. получение ID метода через GetMethodID
3. вызов с помощью CallVoidMethod,  
Call<Type>Method или CallObjectMethod

```
jclass pointClass = env->FindClass( name: "com/vk/vkedunative/Point");
jmethodID constr = env->GetMethodID( clazz: pointClass, name: "<init>", sig: "(II)V");
jmethodID xProp = env->GetMethodID( clazz: pointClass, name: "getX", sig: "()I");
jmethodID yProp = env->GetMethodID( clazz: pointClass, name: "getY", sig: "()I");
```

Обращение к полям выполняется аналогично, только  
используются методы GetFieldID и Get<ТипПоля>Field.

Создание объектов - вызов конструктора, как метода:

1. получение ссылки на класс через FindClass
2. получение ID конструктора через GetMethodID с  
использованием сигнатуры <init>
3. создание объекта через метод NewObject

```
public int foo(String bar, long[][] baz)
```

would become

```
(Ljava/lang/String;[[J)I
```

# Локальные и глобальные ссылки

Объекты, на которые есть ссылки, не могут быть очищены GC.

Каждый аргумент, переданный в нативный код, и почти каждый объект, возвращённый JNI - это [локальная ссылка](#).

- количество ссылок ограничено вместимостью таблицы ссылок
- живут в пределах метода в рамках одного потока (даже если объект фактически живёт дольше)
- при возврате из нативного метода JVM автоматически очищает локальные ссылки в рамках текущего потока
- в нативных потоках нужно явно вызывать `DeleteLocalRef`

[Глобальные ссылки](#) (`newGlobalRef/DeleteGlobalRef`) живут до явного очищения

Две разные ссылки могут ссылаться на один и тот же объект - сравнивать объекты нужно через метод `isSameObject`, а не `==`.

# Потоки

Нативные потоки не в курсе про JVM и JNI - поэтому нужно явно связывать JVM- и нативные потоки:

[AttachCurrentThread](#)

Вызов создаёт `java.lang.Thread` для нативного потока, делает поток видимым для отладки. Теперь в потоке можно использовать `JniEnv`

При завершении работы с потоком нужно вызывать:

[DetachCurrentThread](#)

# Нативная функция

- `extern "C"` - обращение к функции в C-стиле (в C++-стиле могут возникать проблемы из-за переименований функций компилятором)
- `JNIEXPORT` - добавляет функцию в динамическую таблицу, делая её доступной для вызова
- `JNICALL` - конвенция вызова, в Android не используется, нужно для платформенной совместимости
- имя функции в формате `Java_{package_and_classname}_{function_name}`
- второй параметр - ссылка на Java-объект/класс, в котором этот нативный метод объявлен

```
extern "C" JNIEXPORT jstring JNICALL
Java_com_vk_vkedunative_MainActivity_stringFromJNI(
    JNIEnv* env,
    jobject MainActivity /* this */) {
    std::string hello = "Hello from C++";
    return env->NewStringUTF( bytes: hello.c_str());
}
```

# Советы

- избегать лишних вызовов JNI из-за проблем с производительностью
- минимизировать передачу данных из/в нативный код
- избегать излишней работы с потоками
- следить за памятью
- не писать весь кода в одном C/C++ файле, разделять файлы с чистым C/C++ и файлы JNI

# Ссылки

Вспомогательная библиотека Google <https://android.googlesource.com/platform/libnativehelper/> со всякими утилитами.

<https://www.youtube.com/watch?v=9cKO-pEo5No>

Android NDK:

<https://developer.android.com/training/articles/perf-jni#kotlin>

<https://vk.com/@forasoft-kak-ispolzovat-nativnye-biblioteki-v-android>

<https://habr.com/ru/companies/e-legion/articles/487046/>

<https://habr.com/ru/articles/590535/>

<https://www.youtube.com/watch?v=D5Yg-xLAgQE>

<https://www.youtube.com/watch?v=uglvahwOvIM>

<https://medium.com/tompee/android-ndk-jni-primer-and-cheat-sheet-18dd006ec07f>

<https://github.com/android/ndk-samples>

JNI:

<https://docs.oracle.com/javase/8/docs/technotes/guides/jni/spec/jniTOC.html>

<https://www.youtube.com/watch?v=DVTeZdtuHS0>

# Android WebView



# WebView

[WebView](#) - платформенный компонент для отображения веб-контента внутри Android-приложения.

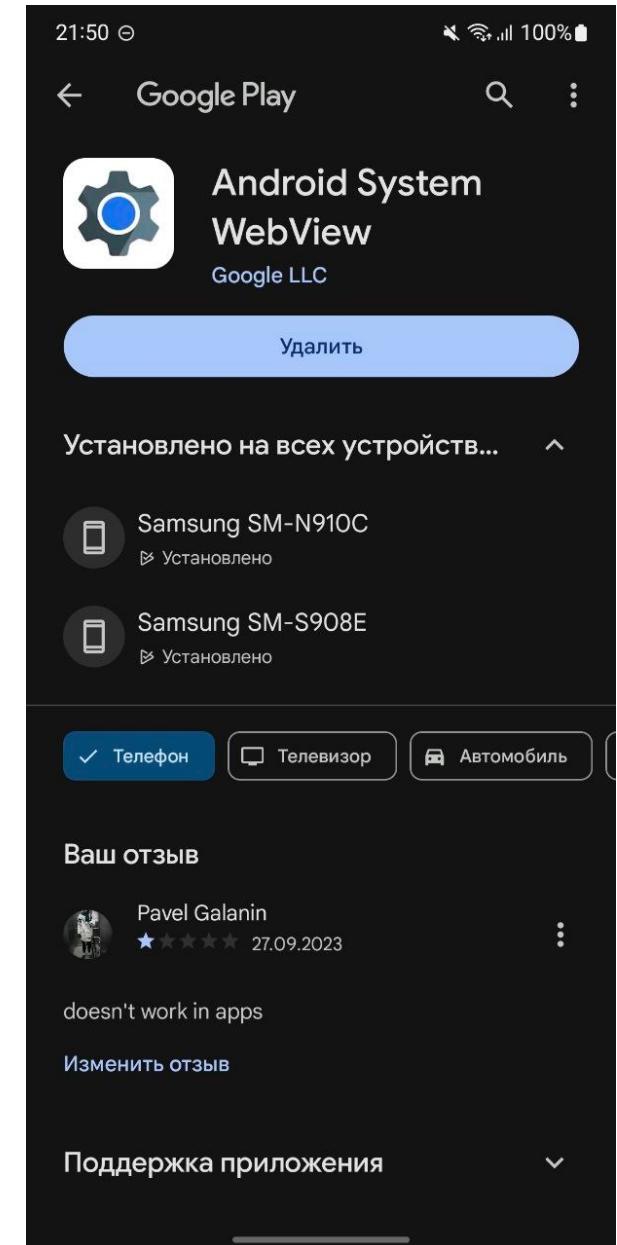
По сути, это небольшой веб-браузер, встроенный в приложение.

Зачем нужна WebView?

- позволяет внедрять веб-контент без необходимости перенаправлять пользователя в браузер
- экономит время на разработку фичи, если она уже есть для веб-платформы - кроссплатформенная технология по своей сути
- ускорение обновления функциональности приложения
- работа с rich-текстом в приложении (HTML, CSS, JavaScript)

Недостатки:

- хуже производительность, чем у нативных компонентов
- ограниченное и усложнённое взаимодействие с платформой
- меньше контроля над приложением, сложная отладка
- зависимость (частичная или полная) от состояния сети
- может привнести веб-уязвимости в приложение
- баги внутри WebView



# Подключение

1. Выдать пермишн на сеть

```
<uses-permission  
    android:name="android.permission.INTERNET" />
```

2. Добавить в разметку WebView

3. Вызвать у объекта WebView loadUrl / loadData

4. Включить поддержку JavaScript:  
`myWebView.settings.javaScriptEnabled = true`

```
<WebView  
    android:id="@+id/webview"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
/>>
```

```
val myWebView: WebView = findViewById(R.id.webview)  
myWebView.loadUrl("http://www.example.com")
```

<https://developer.android.com/develop/ui/views/layout/webapps/webview>

# Взаимодействие Android -> WebView

Можно вызвать JS-функцию со страницы с помощью метода

[evaluateJavascript](#)

```
// !!!!  
webview.settings.apply {  
    javaScriptEnabled = true  
}  
  
// without result  
webview.evaluateJavascript("function1()", null)  
  
// with result  
webview.evaluateJavascript("function2('$param')") {  
    // return String  
}
```

# Взаимодействие WebView -> Android

Чтобы веб-страница могла обратиться к Android-приложению, используется @JavascriptInterface-методы.

С помощью addJavascriptInterface «внутрь» WebView прорывается объект, с которым может взаимодействовать JS-код веб-страницы.

```
/** Instantiate the interface and set the context. */
class WebAppInterface(private val mContext: Context) {

    /** Show a toast from the web page. */
    @JavascriptInterface
    fun showToast(toast: String) {
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show()
    }
}

val webView: WebView = findViewById(R.id.webview)
webView.addJavascriptInterface(WebAppInterface(this), "Android")

<script type="text/javascript">
    function showAndroidToast(toast) {
        Android.showToast(toast);
    }
</script>
```

# Вспомогательные объекты

[WebViewClient](#) - позволяет реализовывать собственную обработку переходов, слушать события загрузки страницы:

- `shouldOverrideUrlLoading`
- `shouldInterceptRequest`
- `onPageStarted` / `onPageFinished`

[WebChromeClient](#) - расширенный набор колбеков:

- `onConsoleMessage`
- `onJsAlert`
- `onShowFileChooser`
- `onShowCustomView`, `onHideCustomView` - поддержка полноэкранного контента (медиаплееров)
- `onReceivedTitle`
- `onProgressChanged`

```
webview.apply {
    webViewClient = SandboxWebClient()
}

class SandboxWebClient: WebViewClient() {
    override fun shouldOverrideUrlLoading(view: WebView?, url: String?): Boolean {
        if (isForbiddenUrl(url)) {
            Toast.makeText(view!!.context, "Forbidden Url: $url", Toast.LENGTH_LONG).show()
            return true
        }
        return false
    }

    protected fun isForbiddenUrl(url: String?): Boolean {
        return !url.isNullOrEmpty() && url.startsWith("http")
    }
}
```

# Кроссплатформенная разработка

• • • •

# Кроссплатформенная разработка

Единая (полностью или частично) кодовая база приложения для разных платформ.

## Преимущества

- + ускорение / удешевление разработки:
- + нужно меньше разработчиков
- + ускорение написания и модификации кода (?)
- + упрощение коммуникаций
- + консистентность поведения между платформами

## Недостатки

- сложнее в изучении, сложнее искать разработчиков
- проблемы с производительностью
- ограничение возможностей разработки



# Фреймворки

## Development frameworks

Framework	Currently owned by	First introduced in	Performance	Development language	Learning curve	Hardware APIs support	Community
React Native	Meta	2015	Average due to the use of bridge	JavaScript	Easy	Limited (direct API's and third-party plugins)	Huge and growing
Flutter	Google	2017	Native-like	Dart	Easy	Extensive (direct API's and third-party plugins)	Fair-sized but rapidly growing
Xamarin	Microsoft	2011	Close to native-like	.NET	Difficult	Extensive (direct API's and third-party plugins)	Fair-sized but shrinking
Ionic Framework	OutSystems	2013	Average due to use of web technologies	Web languages (HTML, CSS, JS)	Easy	Limited (third-party plugins only)	Huge and growing
Apache Cordova	Apache Software Foundation	2011	Average due to use of web technologies	Web languages (HTML, CSS, JS)	Easy	Limited (third-party plugins only)	Huge (possibly larger than Ionic's) but shrinking
NativeScript	OpenJS Foundation	2014	Great (but not as good as Flutter's)	JavaScript or its variants	Fairly easy	Limited (direct access and third-party plugins) but can be expanded easily	Small and fairly stagnant
Kotlin Multiplatform	JetBrains	2017	Native-like (and often better than Flutter)	Kotlin	Moderate	Extensive (expect/actual mechanism)	Small but rapidly growing

# Web-гибридные приложения

Web-гибридные приложения используют web-код (HTML, CSS, JavaScript) для написания нативного приложения.

## Apache Cordova (PhoneGap)

- появился в 2009 году в Nitobi, в 2011 был куплен Adobe
- платформы: iOS, Android, десктоп
- по сути обёртка над WebView, позволяет писать мобильные приложения, используя CSS3, HTML5, и JavaScript
- позволяет подключать плагины, дающие доступ к нативным фичам платформы (геолокация, камера и т.д.)



## Ionic Capacitor

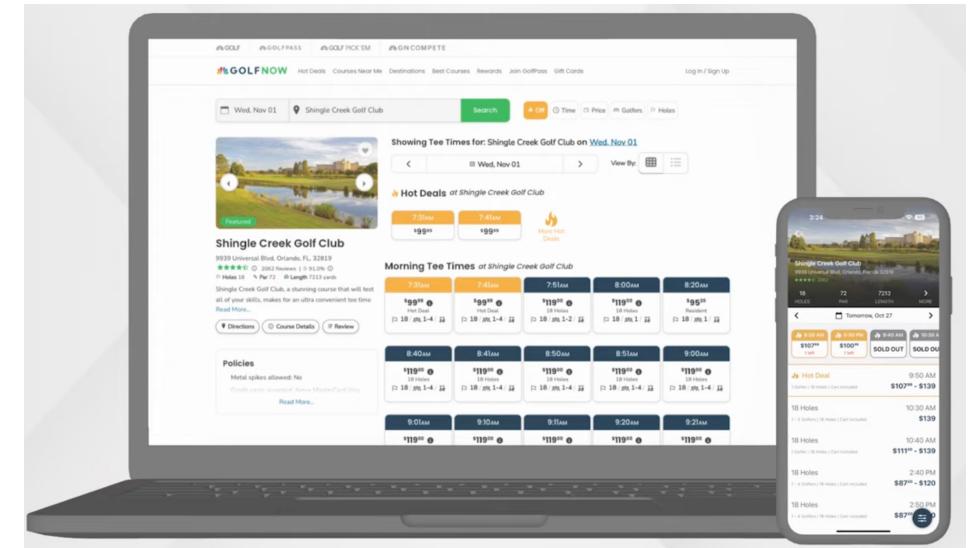
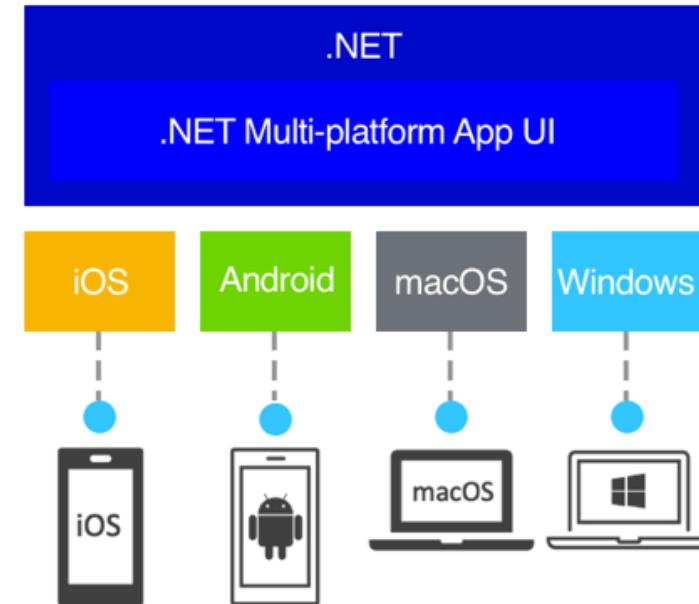
Ionic - верхнеуровневый фреймворк для разработки приложений для различных платформ с использованием web-средств. Предоставляет набор готовых UI-компонентов.

Изначально Ionic работал поверх Apache Cordova, но затем разработчики создали свой фреймворк для мобильных приложений - Capacitor. Capacitor упрощает взаимодействие с нативными API платформ.



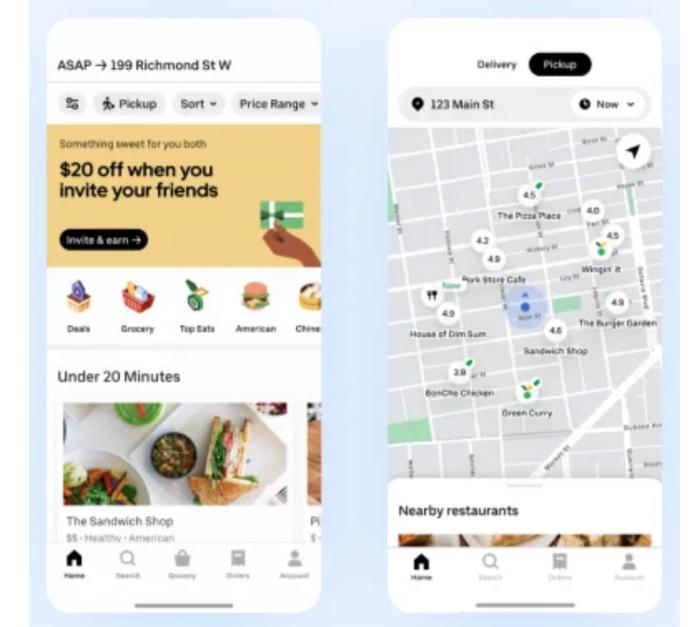
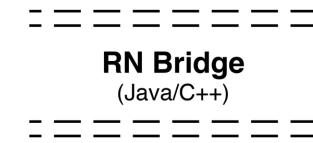
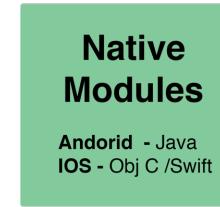
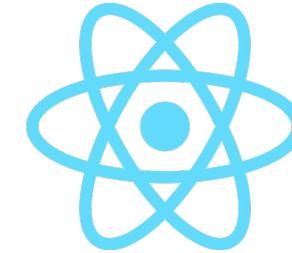
# .NET MAUI / Xamarin

- появился в 2011
- с 2016 года находится под управлением Microsoft
- в 2022 Xamarin стал частью .NET и эволюционировал в [.NET MAUI](#) (Multi-platform App UI)
- платформы: iOS, Android, macOS, Windows
- языки: C#, XAML
- использует нативный рендеринг UI (views, UIKit)
- доступны нативные фичи (GPS, акселерометр, состояние сети т.д.)
- приложения: NBC Sports Next



# React Native

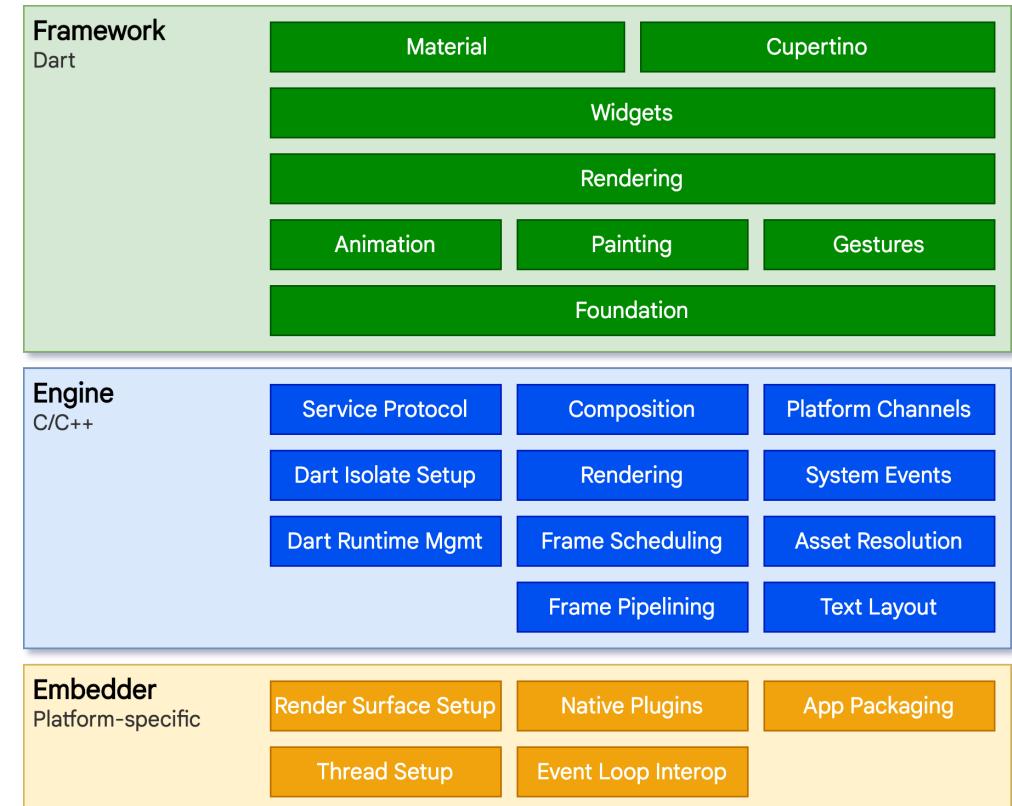
- разработан Facebook, релиз в 2015, но всё ещё не получил версию 1.0
- основан на ReactJS
- языки разработки: JavaScript, JSX
- платформы: Android, iOS, web, десктопы
- использует отдельный поток, на котором работает JavaScript VM - этот поток взаимодействует с UI-потоком через специальный бридж
- поток с JS VM преобразует исходную разметку в нативные UI-компоненты (для Android - во View)
- можно обновлять код на лету (Over The Air, OTA) - загружать код из внешнего сервиса (например App Center CodePush) и обновлять приложение (без обновления приложения в маркете)
- может всё равно требоваться написание нативных модулей (под каждую платформу)
- используется в UberEats, Instagram, Discord, Skype, Airbnb, Pinterest



# Flutter



- платформы: Android, iOS, web, десктоп
- разрабатывается Google, первый релиз в 2017
- язык программирования: Dart
- поддерживает часть нативных фич (геолокация, камера), остальные можно поддержать через Platform Channel
- предоставляет компоненты Material Design
- AOT (ahead-of-time)-компиляция в нативный код (нативные ARM/x86-библиотеки);  
в debug-режиме компилируется в JVM-байткод, чтобы  
работал hot reload
- примеры приложений: eBay, Alibaba, Google Pay



# Отрисовка во Flutter

Изначально Flutter использовал [Skia](#) для отрисовки, но с этого года для мобильных платформ переходит на [Impeller](#).

Skia - компилирует шейдеры во время выполнения - это позволяет адаптироваться к различным графическим API , таким как OpenGL, Vulkan или Metal, и генерировать оптимальные шейдеры для каждого из них.

Impeller разработан командой Flutter исключительно для него. Impeller компилирует шейдеры во время сборки самого движка Flutter.



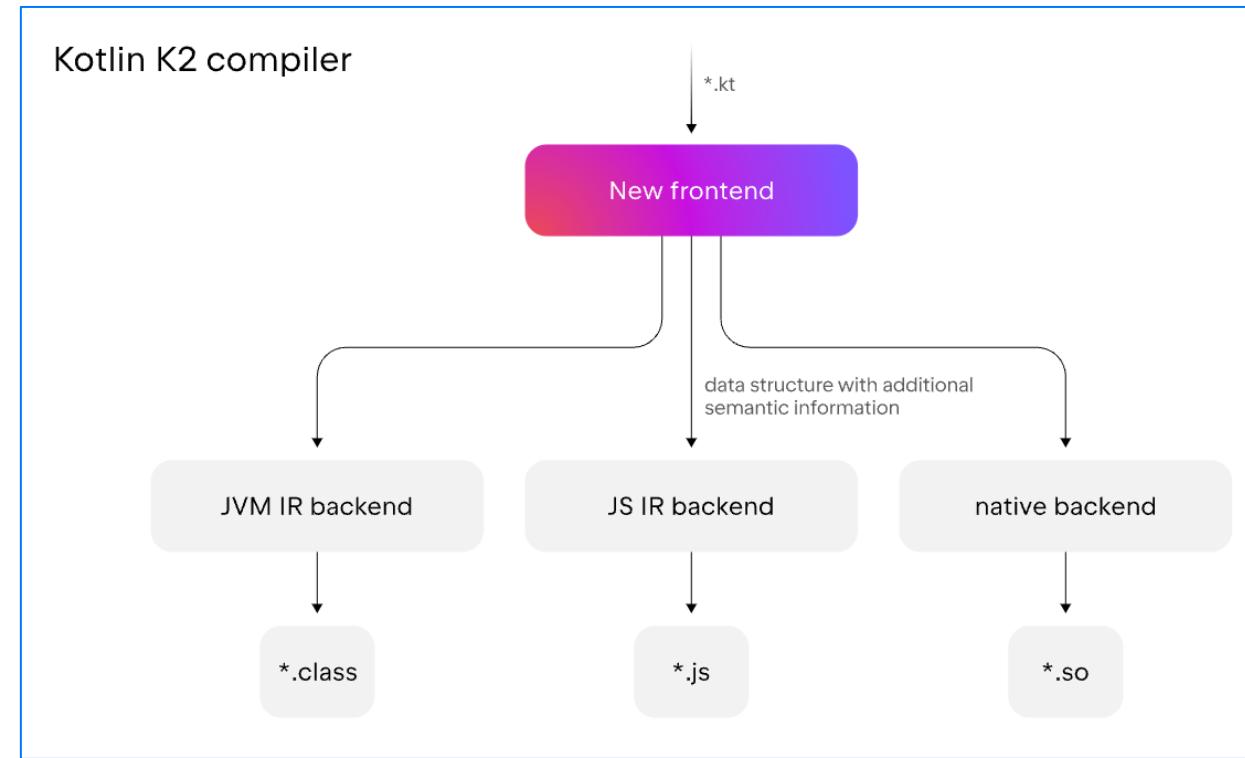
<https://habr.com/ru/articles/795265/>

<https://www.youtube.com/watch?v=vd5NqS01rlA>

# Kotlin Compiler

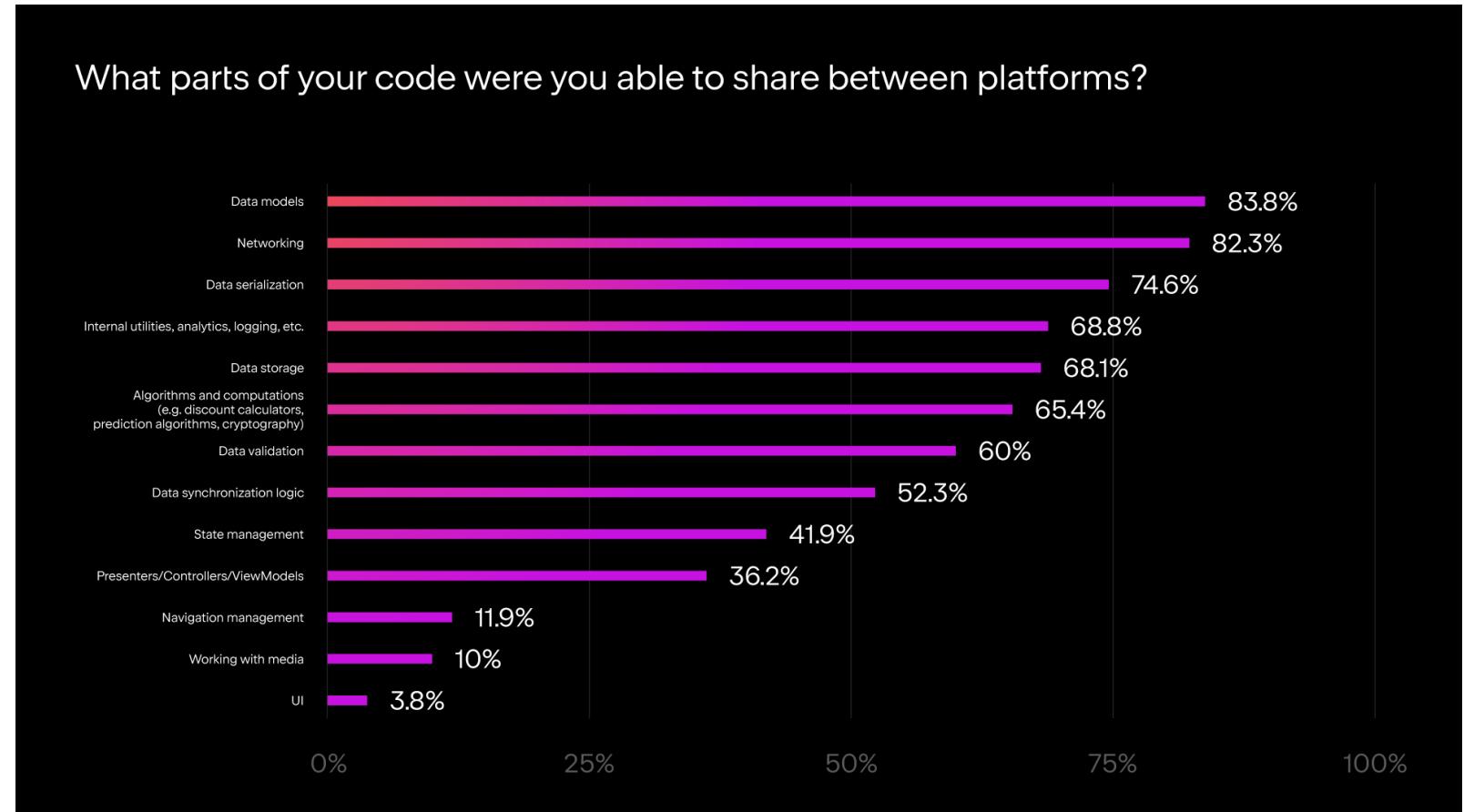
Frontend-компилятор на основе исходного кода составляет Абстрактное Синтаксическое Дерево (AST), а затем компилирует его в более низкоуровневое представление [IR](#) ([Intermediate Representation](#)). IR приближен к машинному коду, но всё ещё является платформонезависимым.

Backend-компилятор производит код, специфичный для платформы.

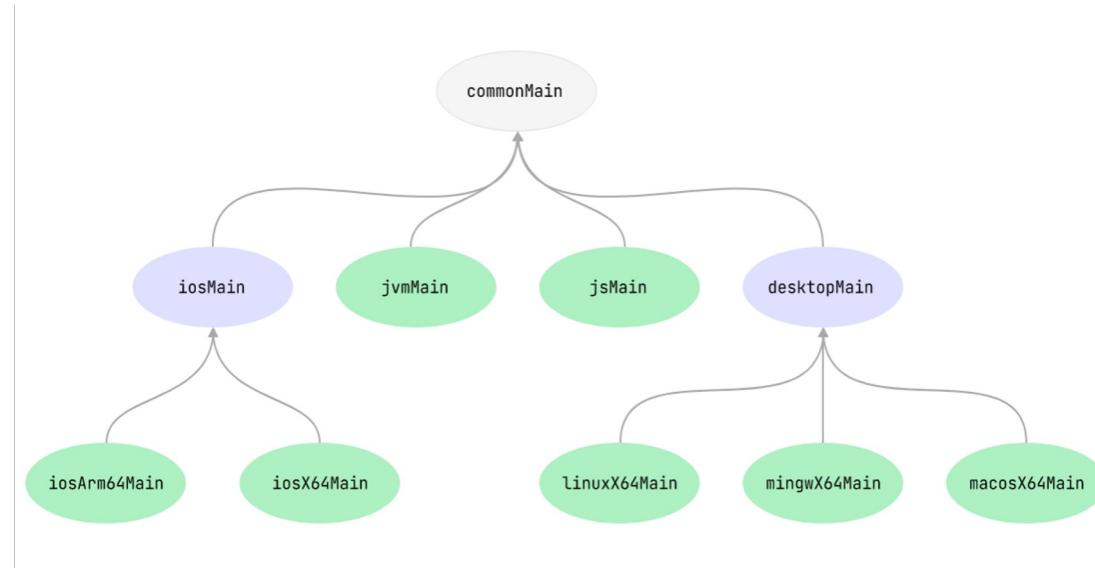


# Kotlin Multiplatform

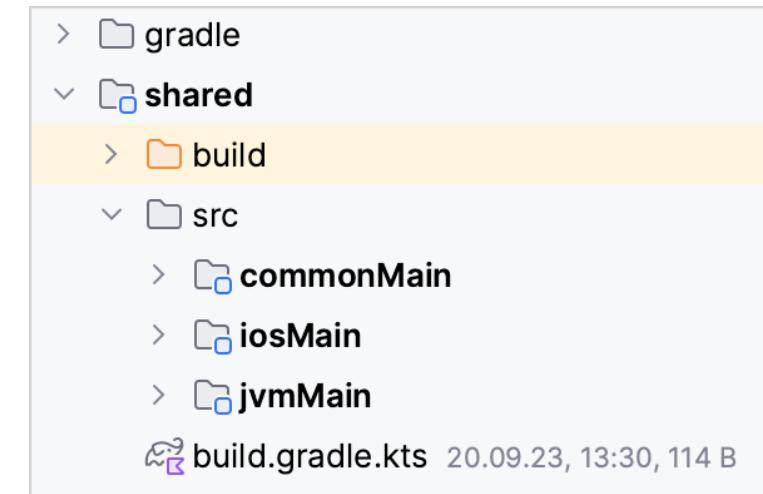
- платформы: Android, iOS, web, desktop и server-side
- языки: Kotlin, Groovy
- позволяет переиспользовать на разных платформах единую кодовую базу
- для использования платформенных возможностей используется механизм `expect/actual`
- для сборки использует Gradle
- используется в приложениях McDonald's, Netflix, Trello, Philips



# Targets



```
kotlin {  
    androidTarget()  
    iosArm64()  
    iosSimulatorArm64()  
  
    sourceSets {  
        iosMain.dependencies {  
            implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:1.7.3")  
        }  
    }  
}
```



# expect / actual

- `expect / actual` - для взаимодействия с платформенными API
- модификатором `expect` можно пометить функцию, свойство, класс, интерфейс, enum-класс или аннотацию
- компилятор подставит нужную реализацию

```
expect fun formatDate(dateString: String, format: String): String
```

## Kotlin/Android

```
import java.text.SimpleDateFormat
import java.util.*

actual fun formatDate(dateString: String, format: String): String {
    val date = if (dateString.isNotEmpty())
        SimpleDateFormat(Constants.normalDateFormat).parse(dateString)
    else
        Date()
    val dateFormatter = SimpleDateFormat(format, Locale.getDefault())
    return dateFormatter.format(date ?: Date())
}
```

## Kotlin/iOS

```
import platform.Foundation.*

actual fun formatDate(dateString: String, format: String): String {
    val dateFormatter = NSDateFormatter().apply {
        dateFormat = Constants.normalDateFormat
    }
    val formatter = NSDateFormatter().apply {
        dateFormat = format
        locale = NSLocale(localeIdentifier = "id_ID")
    }
    return formatter.stringFromDate(
        dateFormatter.dateFromString(dateString) ?: NSDate()
    )
}
```

# Memory Management

- JVM: JVM GC
- JS: JavaScript GC, предоставляемый платформой
- Kotlin/Native - параллельно с приложением работает трассирующий GC

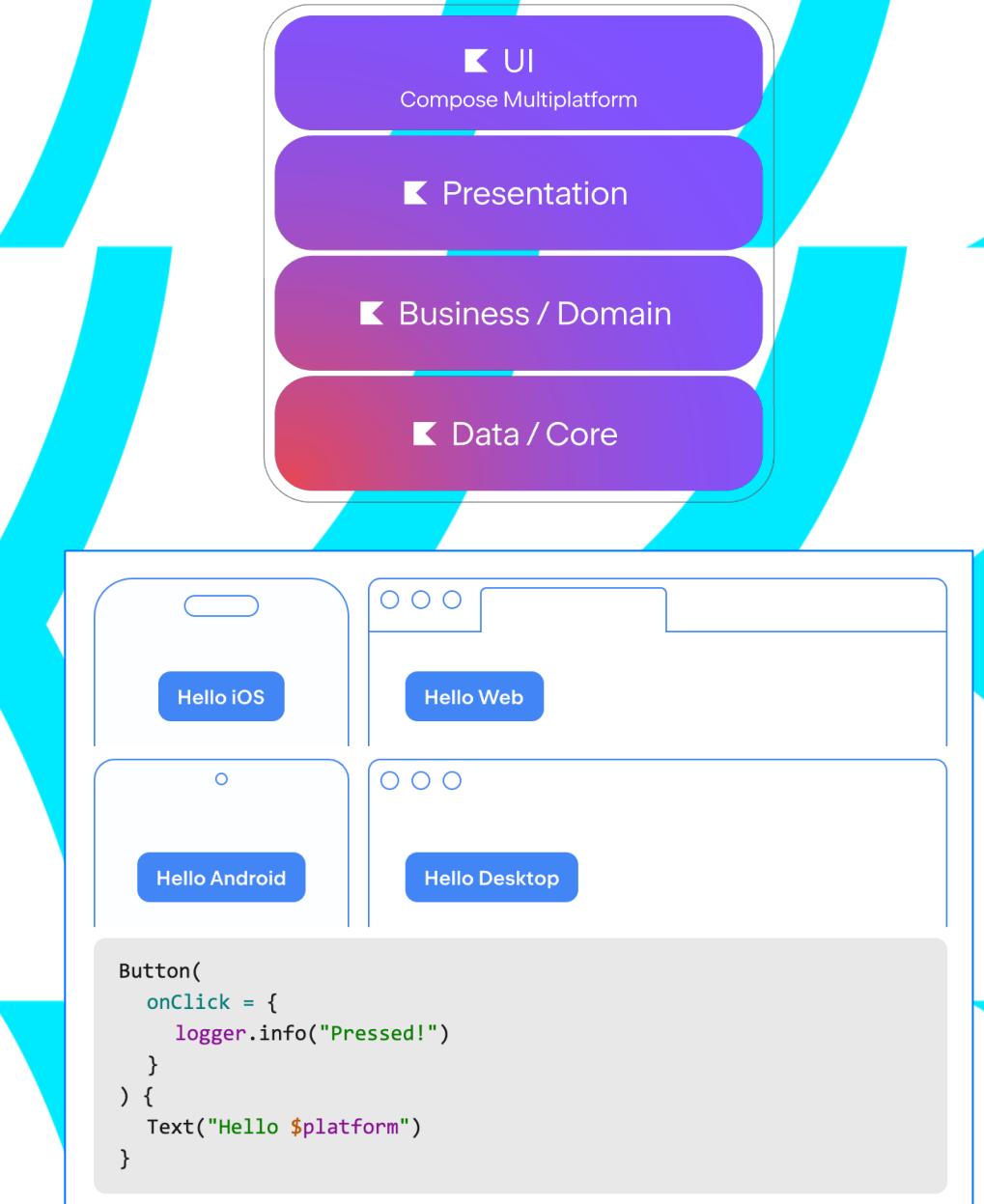
В iOS используется Reference Counting GC - для каждого объекта ведётся подсчёт ссылок на него. Если счётчик равен 0, значит память объекта можно очистить.

# Compose Multiplatform

Для отрисовки использует [Skiko](#) (Skia + Kotlin)

Готовность:

- ✓ Android - релиз (Jetpack Compose)
- ✓ Desktop (Windows, MacOS, Linux) - релиз
- ⚠ iOS - alpha
- 🛠 Web - experimental



# Библиотеки

<https://github.com/Akira/Kotlin-Multiplatform-Libraries>

## Kotlin Multiplatform Libraries

- [Network](#)
- [Repository](#)
- [Serializer](#)
- [Storage](#)
- [DI](#)
- [Image](#)
- [Audio](#)
- [Bluetooth](#)
- [Reactive](#)
- [Utility](#)
- [Debug](#)
- [Test](#)
- [Annotation Processor](#)
- [GUI](#)
- [Command Line Interface](#)
- [Architecture](#)
- [Docs](#)
- [Build & Development Tools](#)
- [Artificial Intelligence](#)
- [Social](#)

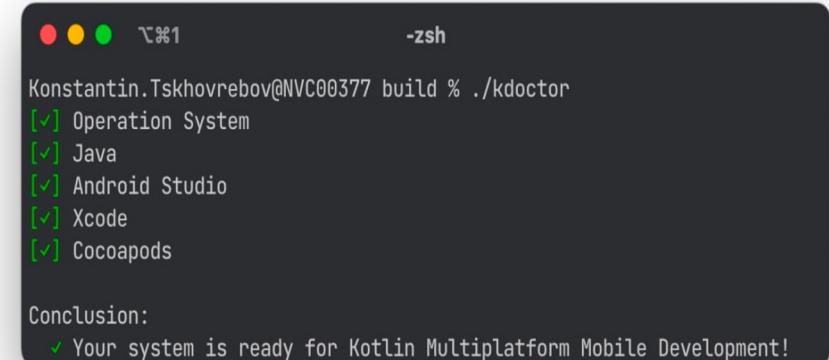


# Kotlin Multiplatform Mobile

Необходимые компоненты для разработки:

1. JDK
2. Android Studio - для разработки и запуска на устройствах/эмуляторах
3. Xcode - для исходной настройки окружения iOS (принятие соглашений и т.д.) и для добавление нативного iOS-кода
4. Плагин Kotlin Multiplatform Mobile для Android Studio

KMM =  
Kotlin Multiplatform + Mobile Features



```
Konstantin.Tskhovrebov@NVC00377 build % ./kdoctor
[✓] Operation System
[✓] Java
[✓] Android Studio
[✓] Xcode
[✓] Cocoapods

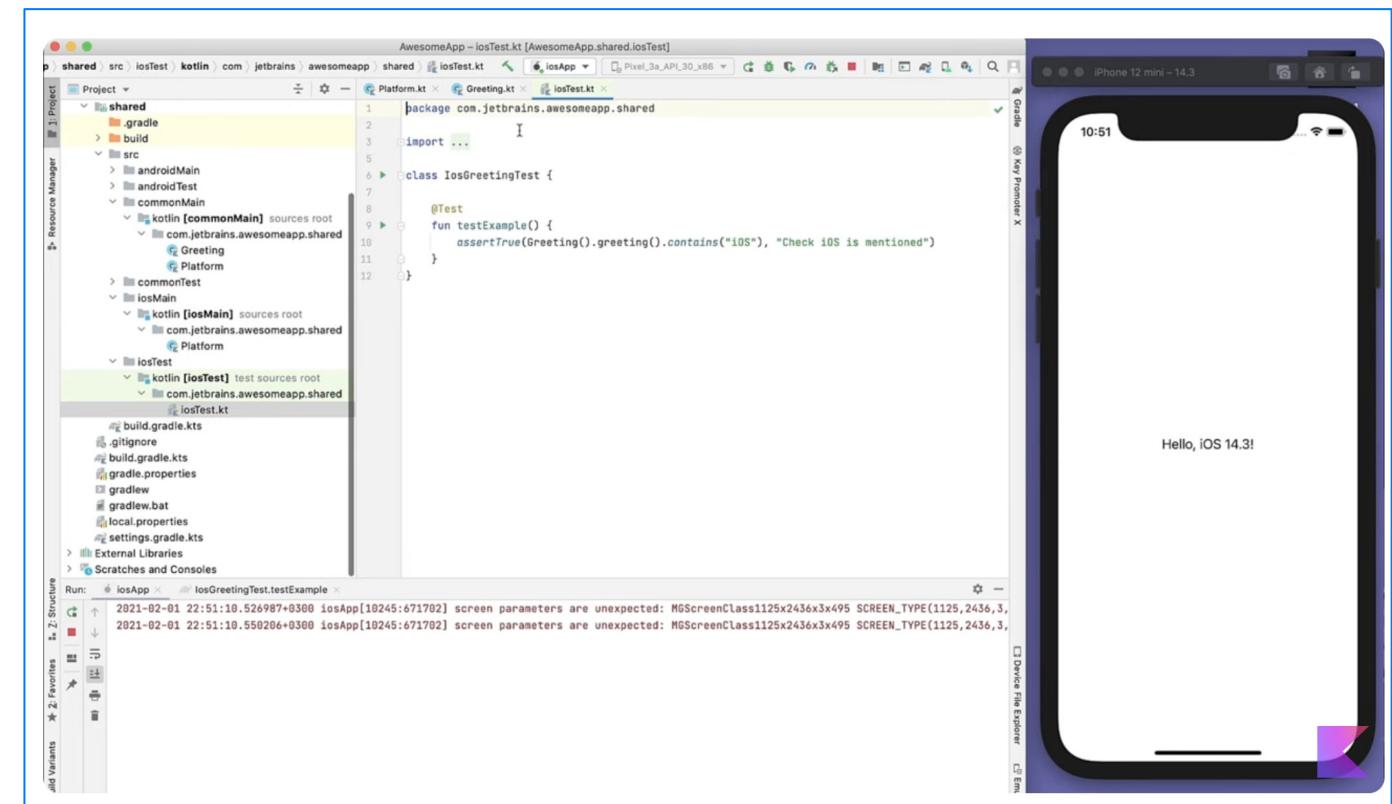
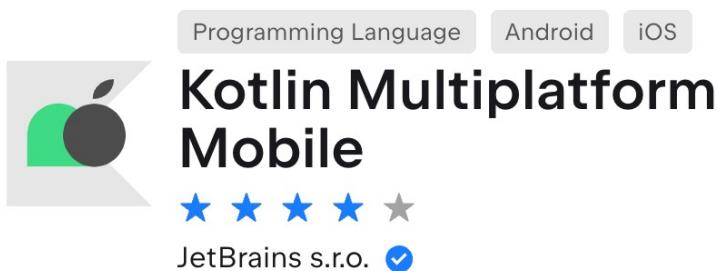
Conclusion:
✓ Your system is ready for Kotlin Multiplatform Mobile Development!
```

<https://github.com/Kotlin/kdoctor>

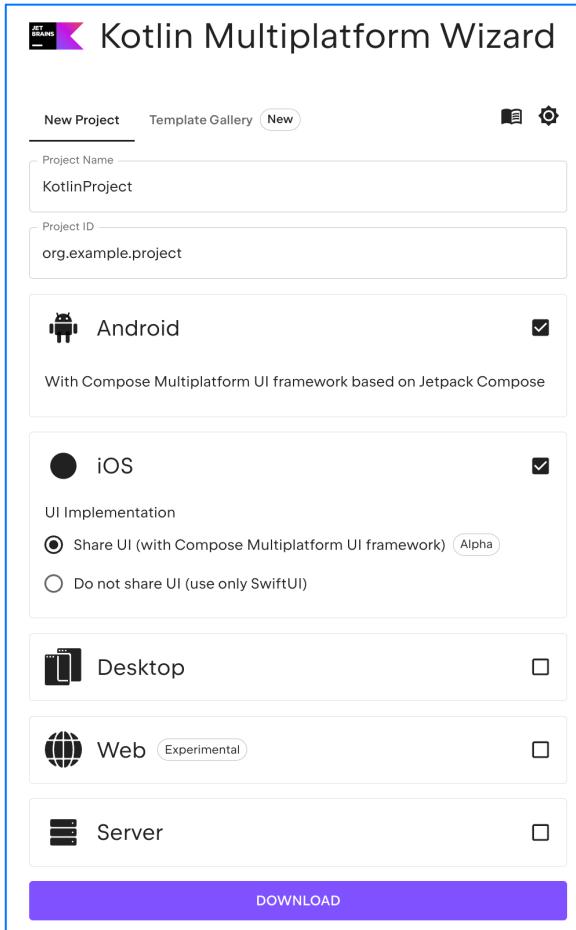
# Плагин Kotlin Multiplatform Mobile

## Плагин Kotlin Multiplatform Mobile:

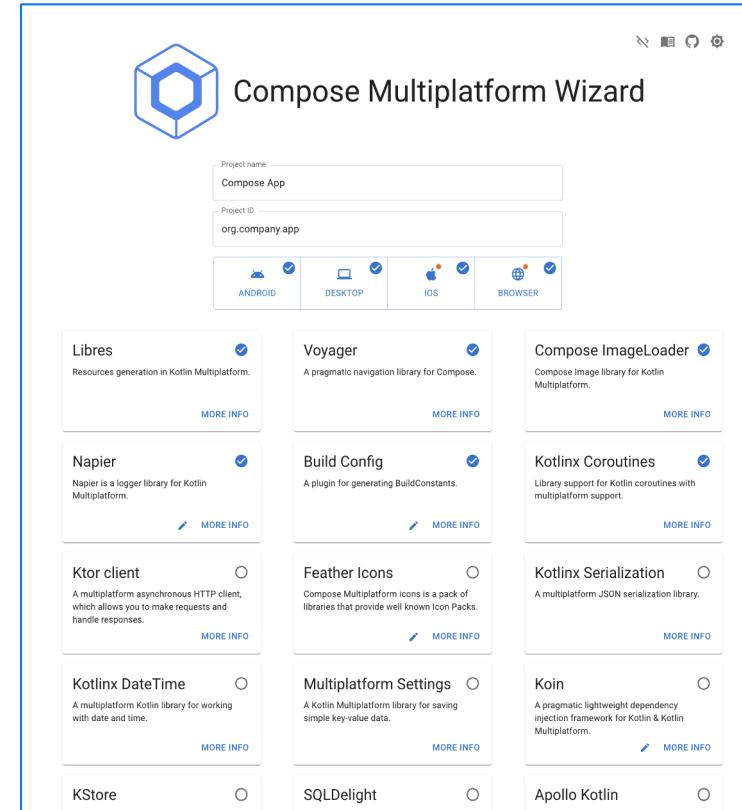
- добавляет визард для создания КММ-проектов
- позволяет отлаживать iOS-приложения в Android Studio



# Полезные инструменты



<https://kmp.jetbrains.com/>



<https://terrakok.github.io/Compose-Multiplatform-Wizard/>

Спасибо за  
внимание!

