




Вводная лекция по дисциплине «Разработка мобильных приложений»

Автор: Леонтьев Д.А.

denis.leontev92@yandex.ru

Введение

-  Мобильные приложения — основная форма взаимодействия с цифровым миром.
-  Навыки Android-разработки востребованы — от стартапов до крупных компаний.
-  Вы научитесь создавать реальные приложения, которые можно запускать на своём телефоне.

Как будет проходить дисциплина

- Лекции: одна вводная (эта).
- Основной упор — на **практику**: лабораторные работы в **IntelliJ IDEA** и **Android Studio**.
- Вы изучите:
 - основы **Kotlin**,
 - создание UI через **Jetpack Compose**,
 - запуск на **эмуляторе** и реальном устройстве,
 - базовую архитектуру **Android-приложения**.

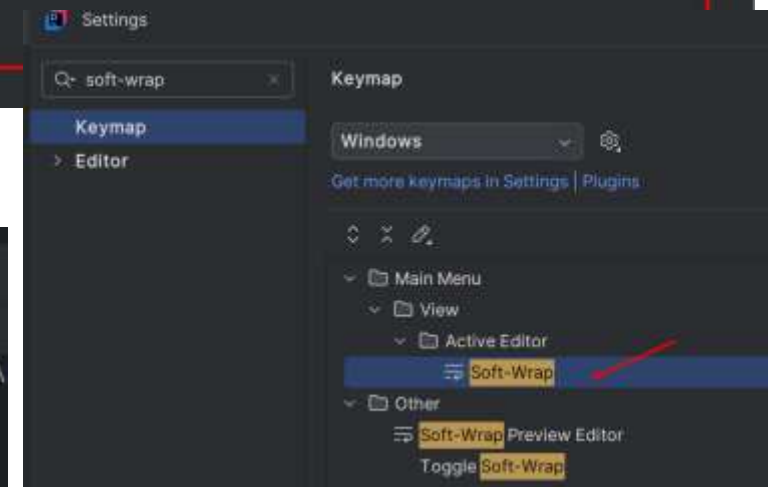
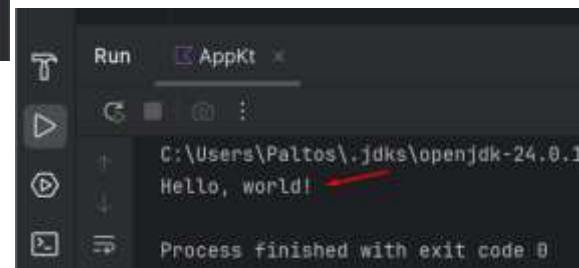
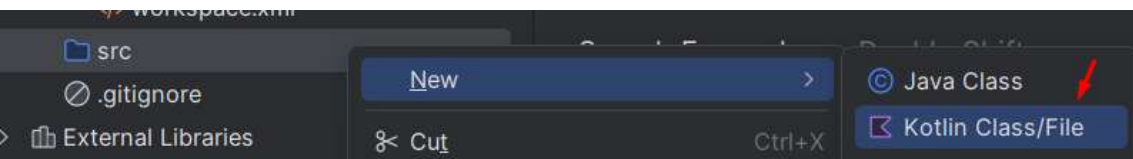
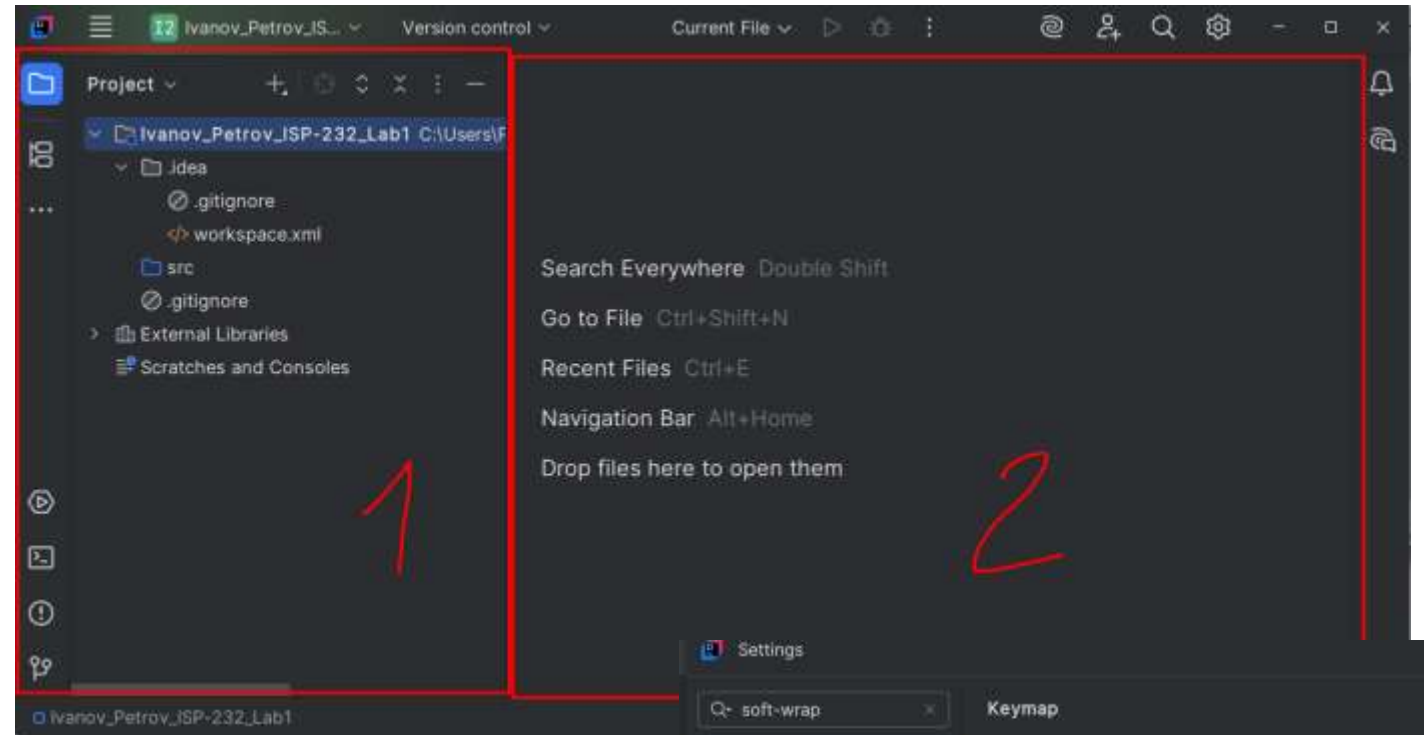
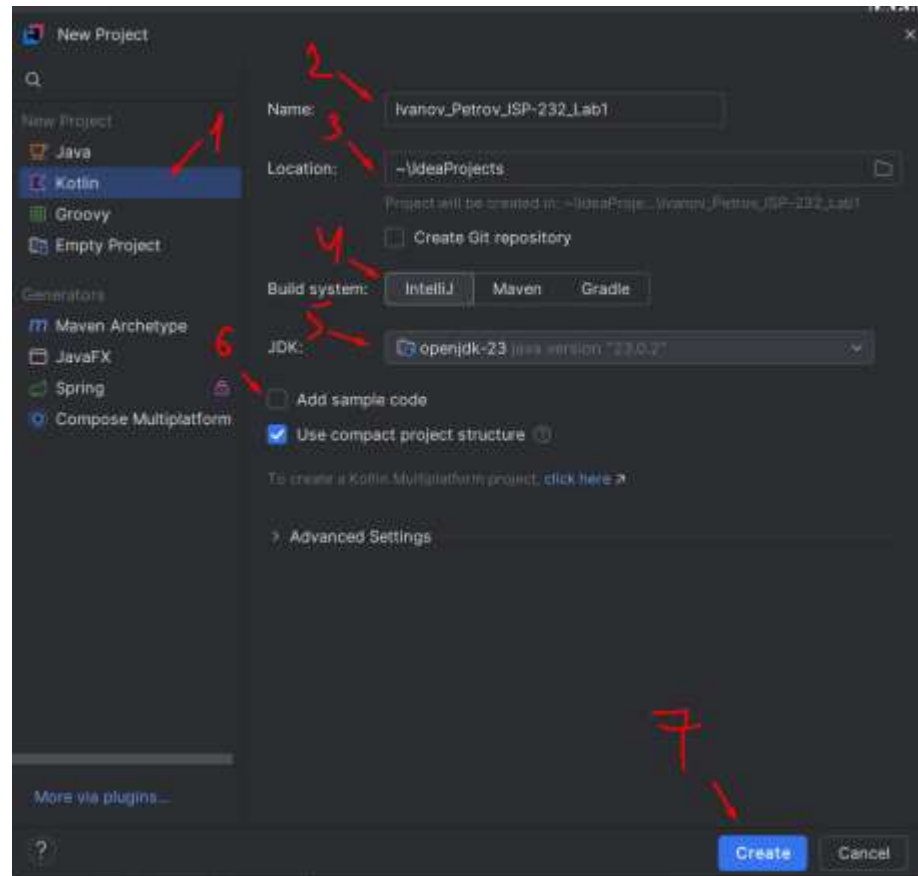
Почему Kotlin, а не Java

- Kotlin — официальный язык разработки под Android (с 2017 года).
- Современный, лаконичный, безопасный (null-безопасность, лямбды, корутины).
- Поддерживается Google и JetBrains.
- Быстрее учить Kotlin с нуля, чем Java + XML + View Binding и т.д.



Что будем изучать. 1 Семестр

- Научиться работать в IntelliJ IDEA



Что будем изучать. 1 Семестр

```
fun main(){
    val name = "Denis" // объявить переменную с именем
    'name' и присвоить ей значение "Denis"
    val age = 18
    println("Hello")
    println("My name is $name")
    println("My age is $age")
    val a = 10
    val b = 15
    val c = a + b + 8
    val str = c.toString()
    println(str)
    val numList = arrayOf(1, 2, 3)
    var x = 0
    while (x < 3) {
        println("Item $x is ${numList[x]}")
        x = x + 1
    }
}
```

- Изучить базовый синтаксис Kotlin
- Изучить стиль кодирования в Kotlin.

name inputs

```
fun main() {
    println("Hello, world!") ← body
}
```

val

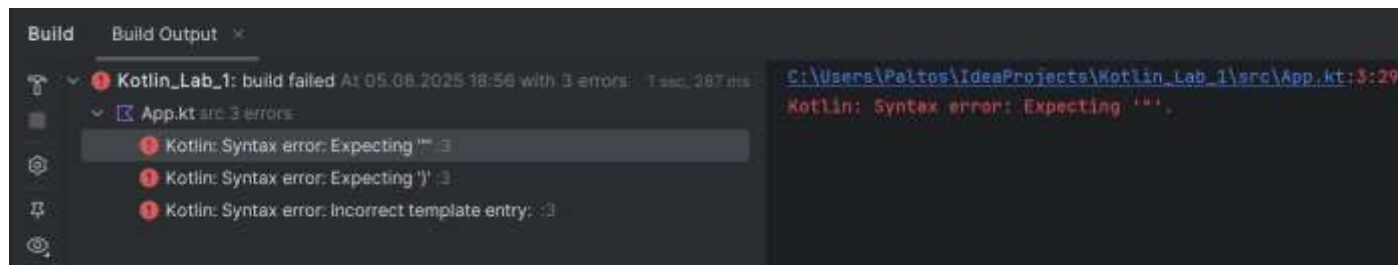
name

:

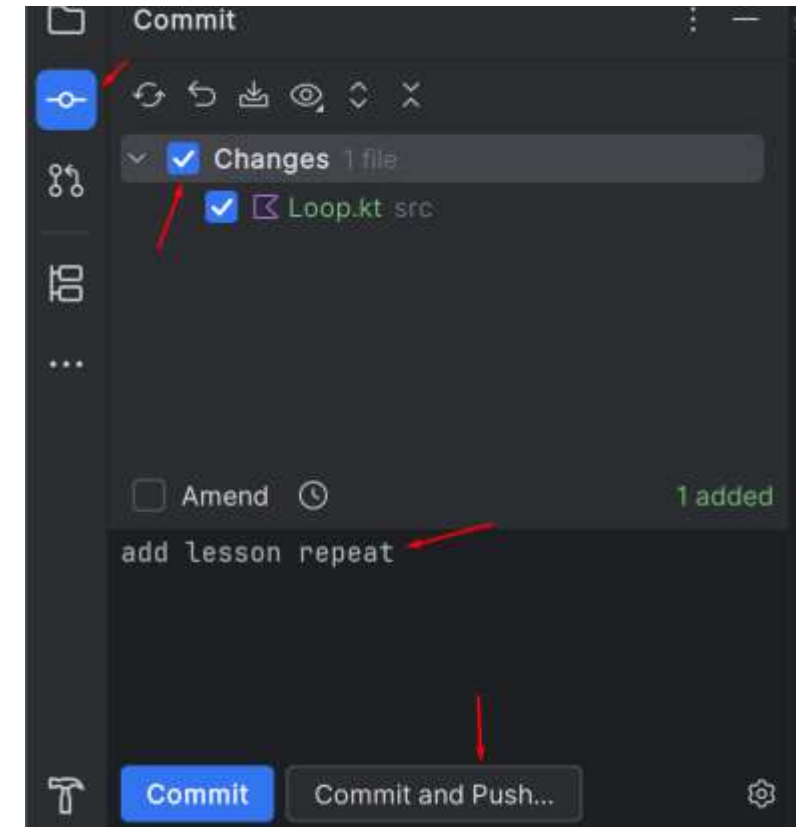
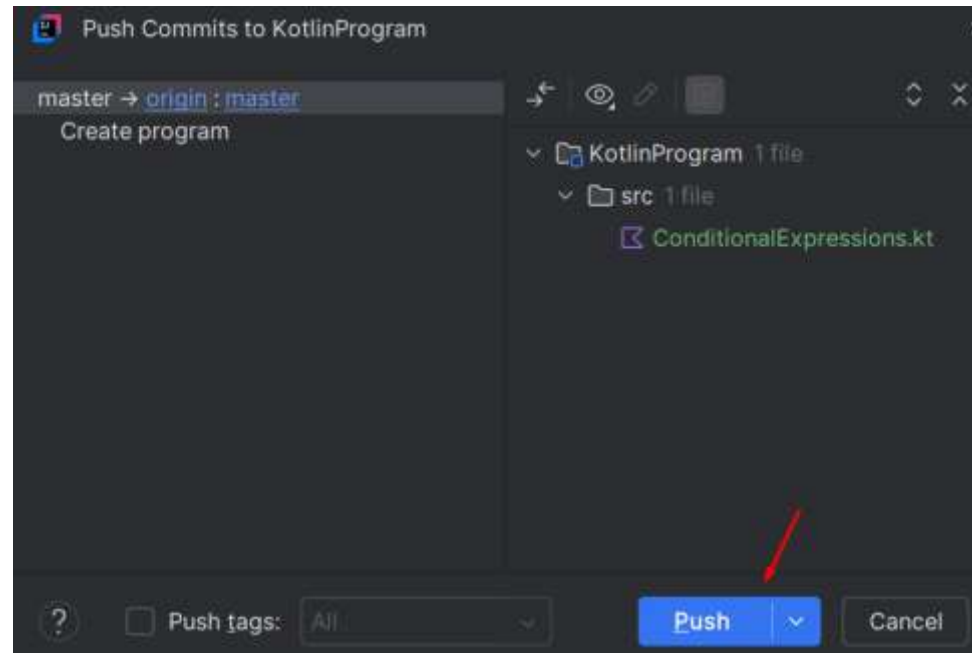
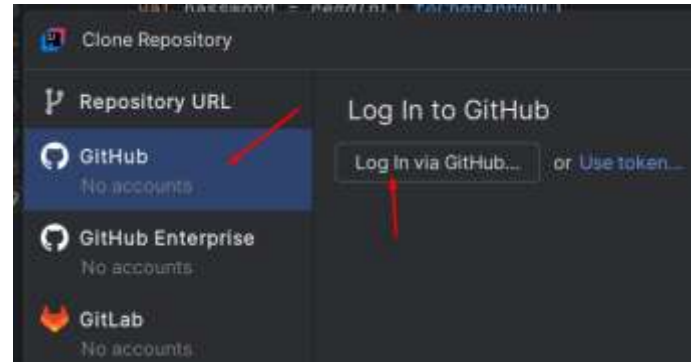
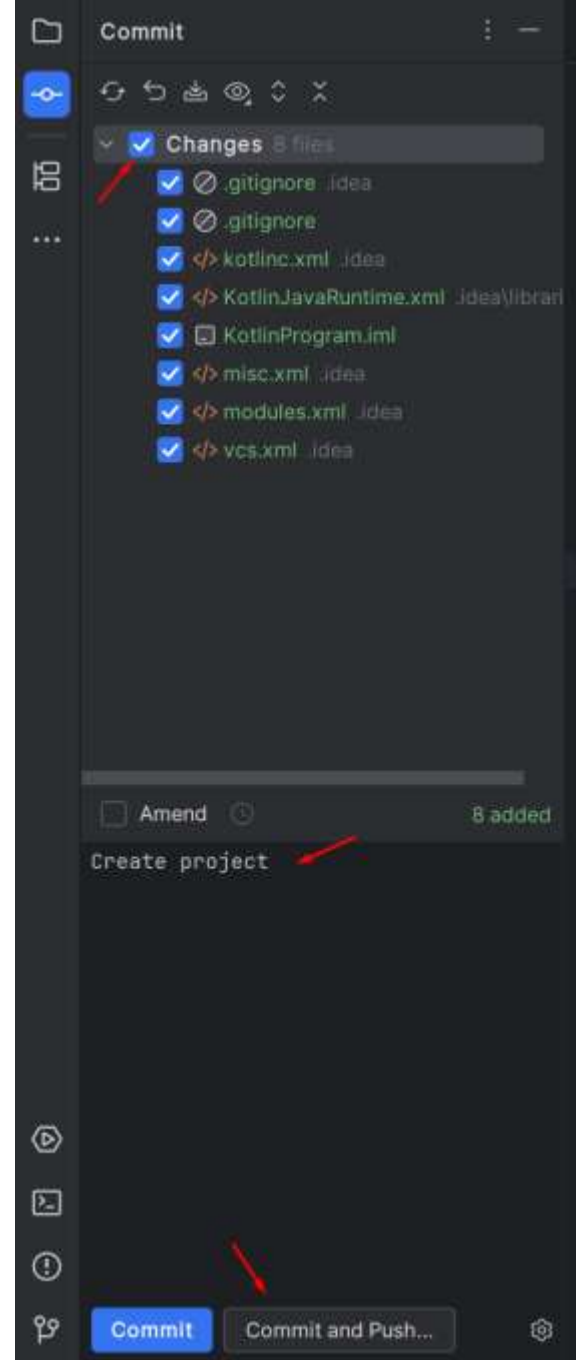
data type

=

initial value



- Научиться работать с **Git-репозиторием**.




```
fun main() { new *
    val numbers = 1..100
    val symbol = 'a'.. 'z'
}
```

- Изучить конструкции **if**, **when** и диапазоны **..**, **in**.
- Освоить цикл **repeat** и базовую работу с коллекциями.

```
fun main() { new *
    println("Начинаем ритуал защиты!")
    repeat(times = 5) {
        println("Заклинание защиты активировано!")
    }
    println("Замок защищен!")
}
```

```
val months = listOf("Python", "Kotlin", "C#", "C++", "Java")
```

```
if (index == 1) {
    month = "Январь"
} else if (index == 2) {
    month = "Февраль"
} else if (index == 3) {
    month = "Март"
} else if (index == 4) {
    month = "Апрель"
} else if (index == 5) {
    month = "Май"
}
```

```
else if (index == 6) {
    month = "Июнь"
} else if (index == 7) {
    month = "Июль"
} else if (index == 8) {
    month = "Август"
} else if (index == 9) {
    month = "Сентябрь"
} else if (index == 10) {
    month = "Октябрь"
} else if (index == 11) {
    month = "Ноябрь"
} else if (index == 12) {
    month = "Декабрь"
}
```



```
when (index) {
    1 -> month = "Январь"
    2 -> month = "Февраль"
    3 -> month = "Март"
    4 -> month = "Апрель"
    5 -> month = "Май"
    6 -> month = "Июнь"
    7 -> month = "Июль"
    8 -> month = "Август"
    9 -> month = "Сентябрь"
    10 -> month = "Октябрь"
    11 -> month = "Ноябрь"
    12 -> month = "Декабрь"
}
```


- Научиться использовать **циклы**, **генерацию случайных чисел** в Kotlin.

```
while (условие) {  
    // Код, который будет повторяться  
}
```

```
while (number <= 10) {  
    println(number)  
    number++  
    if (number == 5) {  
        println("Останавливаемся на $number")  
        break  
    }  
}
```

```
fun main() {  
    for (i in 1..5) {  
        println("Шаг $i")  
    }  
}
```

```
do {  
    // Блок кода  
} while (условие)
```

```
import kotlin.random.Random  
  
fun main() {  
    new *  
    println(Random.nextLong())  
    println(Random.nextFloat())  
    println(Random.nextDouble())  
}
```

```
val part1 = arrayOf("Опытный", "Безумный", "Легендарный",  
    "Скрытый", "Гигачад")  
val part2 = arrayOf("стрелок", "геймер", "воин",  
    "волшебник", "трейдер")  
val part3 = arrayOf("из CS60", "на максималках", "из  
    будущего", "в бане у Габена", "с проклятым лутом")
```

- Освоить синтаксис функций
- Перегрузка функций
- Анонимные функции и лямбда-выражения

```
fun main() {
    greetPlayer() // Вызов функции
    greetPlayer() // Вызов ещё раз
}

// Определение функции
fun greetPlayer() { 2 Usages
    println("Добро пожаловать в игру!")
}
```

```
fun main() {
    showPlayerStats(name = "Aragorn", health = 100)
    showPlayerStats(name = "Legolas", health = 85)
}

fun showPlayerStats(name: String, health: Int) { 2 Usages
    println("Игрок: $name | Здоровье: $health HP")
}
```

```
val greet = fun(name: String) { 1 Usage
    println("Привет, $name!")
}

fun main() {
    greet("Игрок") // Вывод: Привет, Игрок!
}
```

```
fun sum(a: Int, b: Int) : Int{ new *
    return a + b
}

fun sum(a: Double, b: Double) : Double{ new *
    return a + b
}

fun sum(a: Int, b: Int, c: Int) : Int{ new *
    return a + b + c
}

fun sum(a: Int, b: Double) : Double{ new *
    return a + b
}

fun sum(a: Double, b: Int) : Double{ new *
    return a + b
}
```

```
fun main() {
    val hello = {println("Hello Kotlin")}
    hello()
    hello()
}
```

• Введение в ООП

```
class Hero { 2 Usages
    var name: String = "Неизвестный" 2 Usages
    var gender: String = "Не указан" 2 Usages
    var role: String = "Бродяга" 2 Usages
    var hp: Int = 100 2 Usages
    var mp: Int = 50 2 Usages
}
```

```
fun showStats() { new *
    println("Имя: $name | Класс: $role | HP: $hp | MP: $mp |
        Level: $level | Element: $element")
}
```

```
fun init(title: String, duration: Int, reward: Int, difficulty: String) {
    this.title = title
    this.duration = duration
    this.reward = reward
    this.difficulty = difficulty
}
```

```
quest.init(
    title = "Охота на тролля",
    reward = 500,
    duration = 3,
    difficulty = "Средний"
)
```

```
fun main() { ② Denis
    val naruto = Hero()
    naruto.name = "Наруто"
    naruto.role = "Шиноби"
    naruto.element = "Ветер"
    naruto.mp = 100

    val kakashi = Hero()
    kakashi.name = "Какаши"
    kakashi.role = "Шиноби"
    kakashi.element = "Молния"
    kakashi.mp = 100

    val orochimaru = Enemy()
    orochimaru.name = "Орочимару"
    orochimaru.element = "Ветер"

    naruto.castSpellOn( enemy = orochimaru, spellName = "Расенган", damage = 30)
    naruto.duel( opponent = kakashi)
```


- Первичный Конструктор

```
class Spell( 5 Usages new *  
    val name: String,  
    val width: Int,  
    val height: Int,  
    val symbol: String  
) {  
  
    // Конструктор для квадратных заклинаний (width = height)  
    constructor(name: String, size: Int, symbol: String) :  
        this(name, width = size, height = size, symbol) new *  
  
    // Конструктор по умолчанию  
    constructor() : this(name = "", width = 0, height = 0,  
        symbol = "") new *
```

- Ключевое слово **this**

```
// Основной конструктор, который инициализирует все свойства  
constructor(name: String, width: Int, height: Int, symbol: String) {  
    this.name = name  
    this.width = width  
    this.height = height  
    this.symbol = symbol  
}
```

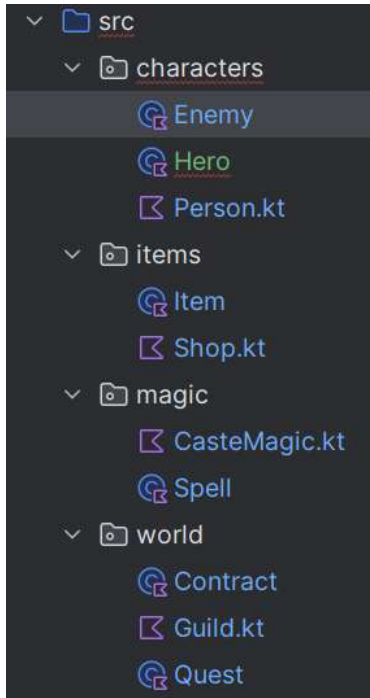
- Конструкторы

```
constructor (title: String, duration: Int, reward: Int, difficulty: String) {  
    this.title = title  
    this.duration = duration  
    this.reward = reward  
    this.difficulty = difficulty  
}
```

- Перегрузка конструкторов

```
// Основной конструктор, который инициализирует все свойства  
constructor(name: String, width: Int, height: Int, symbol: String) {  
    this.name = name  
    this.width = width  
    this.height = height  
    this.symbol = symbol  
}  
  
// Конструктор для квадратных заклинаний (width = height)  
constructor(name: String, size: Int, symbol: String) : this(name,  
    width = size, height = size, symbol) new *  
  
// Конструктор по умолчанию  
constructor() : this(name = "", width = 0, height = 0, symbol = "")
```

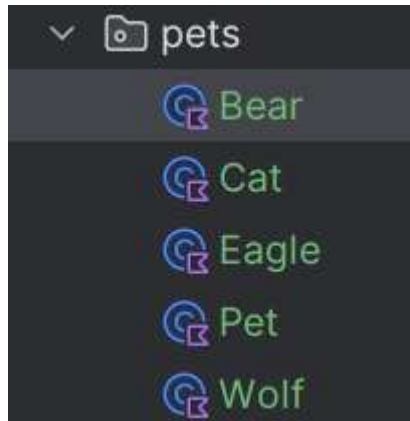
- Пакеты



- Наследование

```
open class Pet {
```

```
class Wolf: Pet()
```

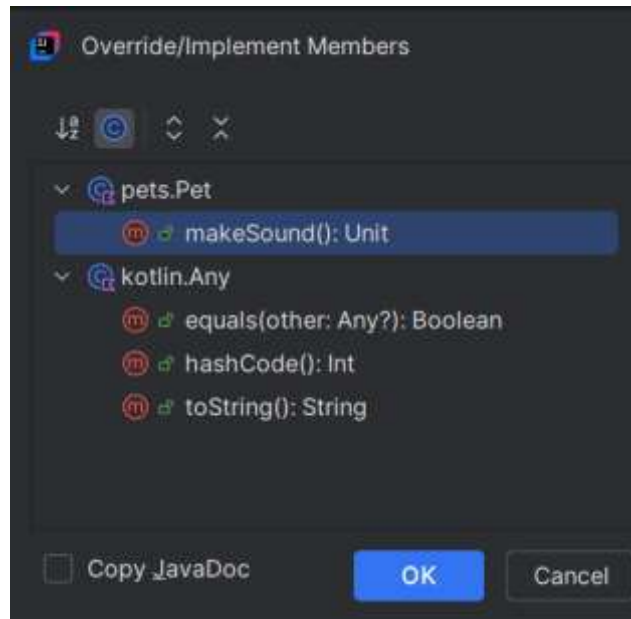


- Upcast, Downcast, Smartcast

```
(enemy as Enemy).takeDamage( amount = 20)  
enemy.takeDamage( amount = 20)  
println(enemy.name)
```

- Полиморфизм

```
6 override fun makeSound() { new *
```



- Перечисления (enum class)

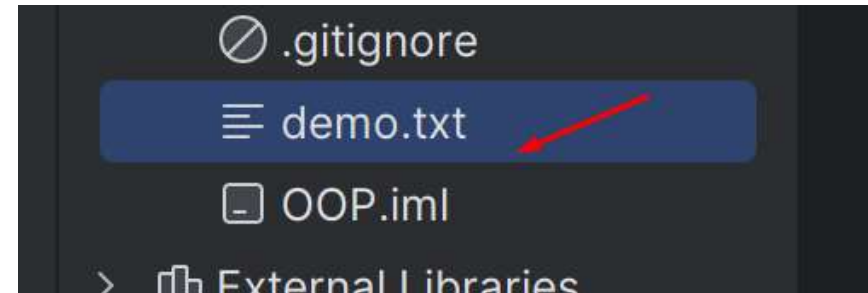
```
enum class QuestType(val description: String) { 5 Usages
    DELIVERY(description = "Доставка предмета"),
    ELIMINATION(description = "Устранение противника"),
    ESCORT(description = "Сопровождение персонажа"),
    EXPLORE(description = "Исследование новой территории"),
    BOSSFIGHT(description = "Битва с боссом")
}
```

- Работа с пакетами

```
package characters
import world.Quest
```

- Работа с файлами

```
println("Добавим новую строку с помощью appendText...")
file.appendText(text = "Это новая строка!\n")
println("Строка добавлена.")
println("Текущее содержимое файла:")
println(file.readText())
```



- Создание простого **текстового менеджера задач (To-Do)** с сохранением в файл

```
=== МЕНЮ ЗАДАЧ ===  
1 - Добавить задачу  
2 - Показать все задачи  
3 - Удалить задачу  
0 - Выход  
Выберите действие:
```

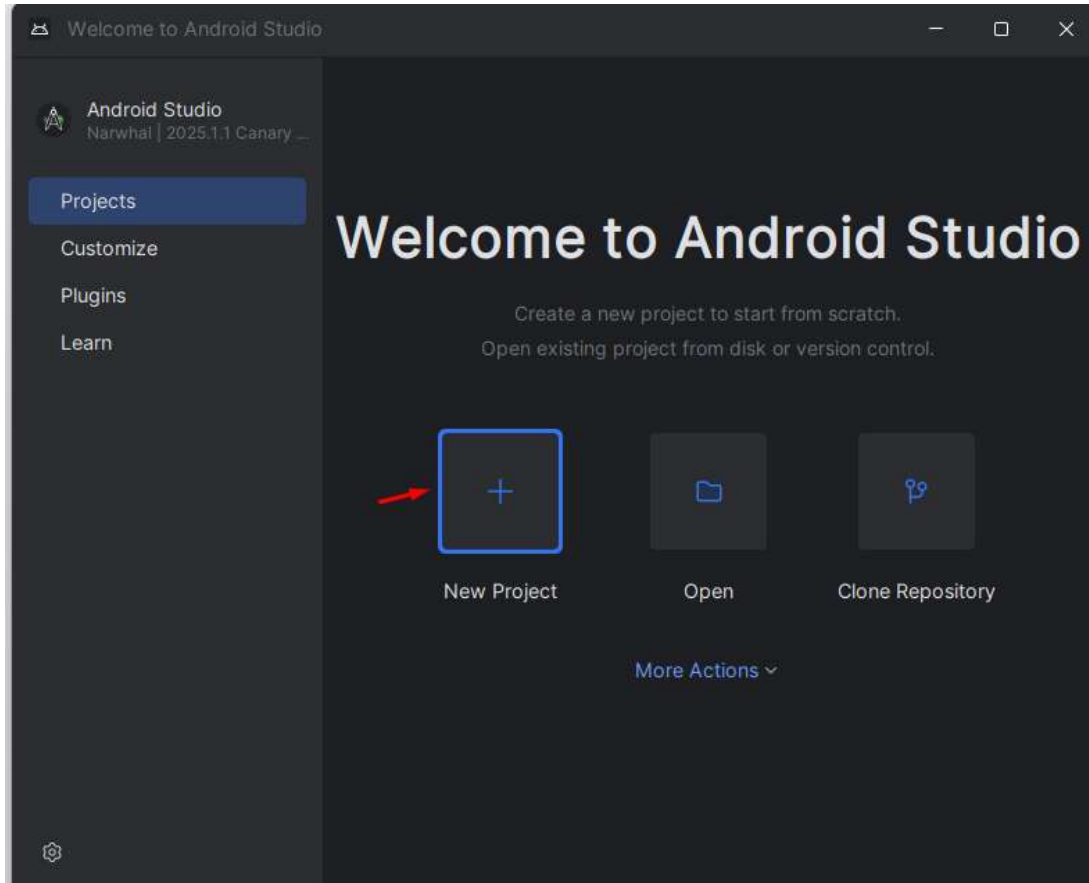
```
Выберите действие: 1  
Введите новую задачу: Поучить Kotlin  
Задача добавлена.
```

```
Выберите действие: 2  
Все задачи:  
1. Поучить Kotlin
```

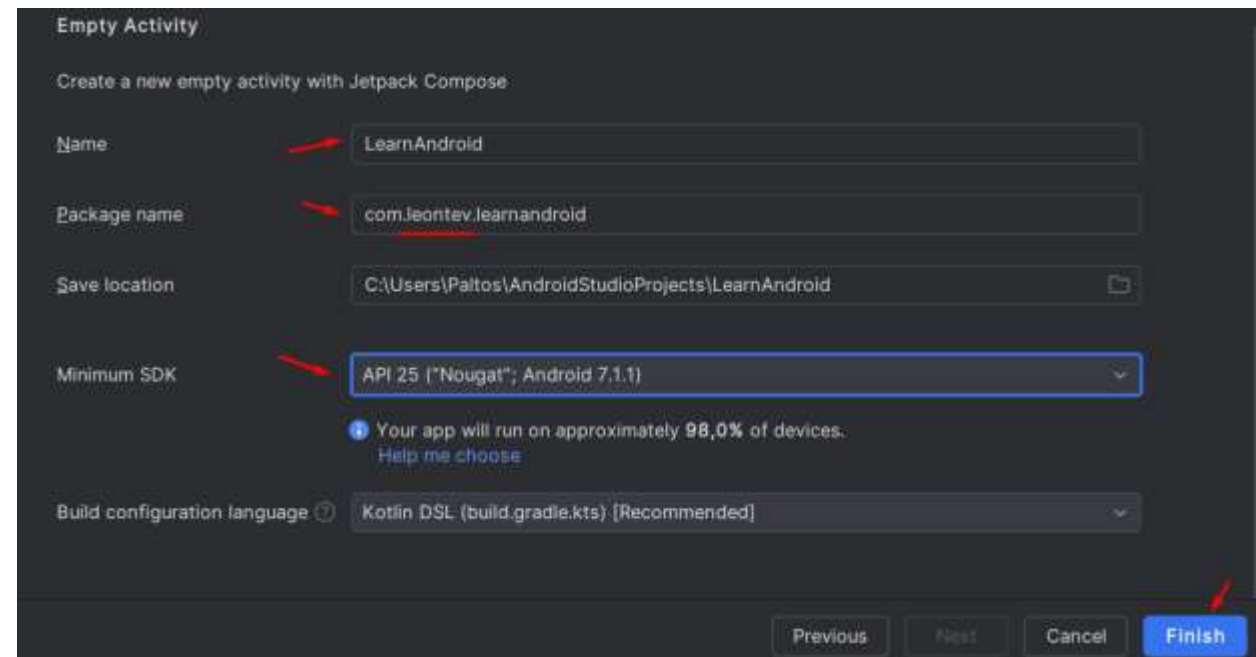
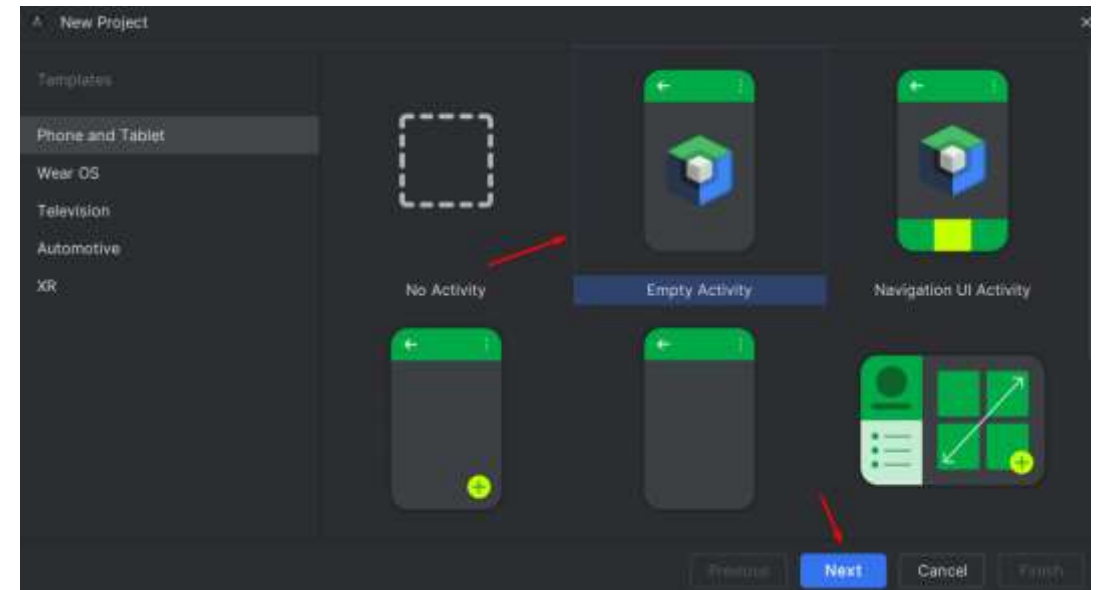
```
Выберите действие: 3  
Задачи для удаления:  
1. Поучить Kotlin  
Введите номер задачи для удаления: 1  
Задача "Поучить Kotlin" удалена.
```

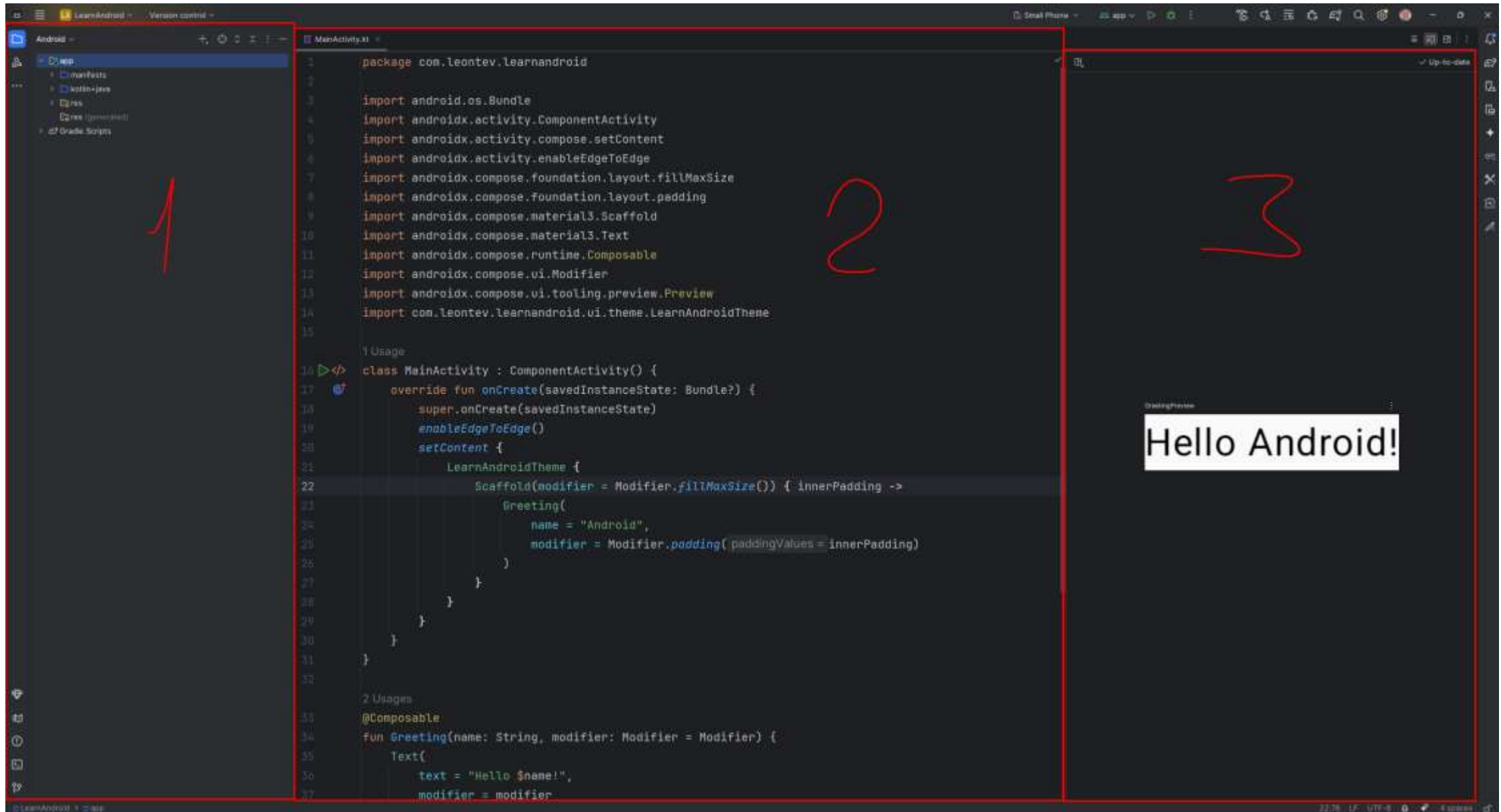
```
Выберите действие: 0  
Выход из программы.
```


Android Studio

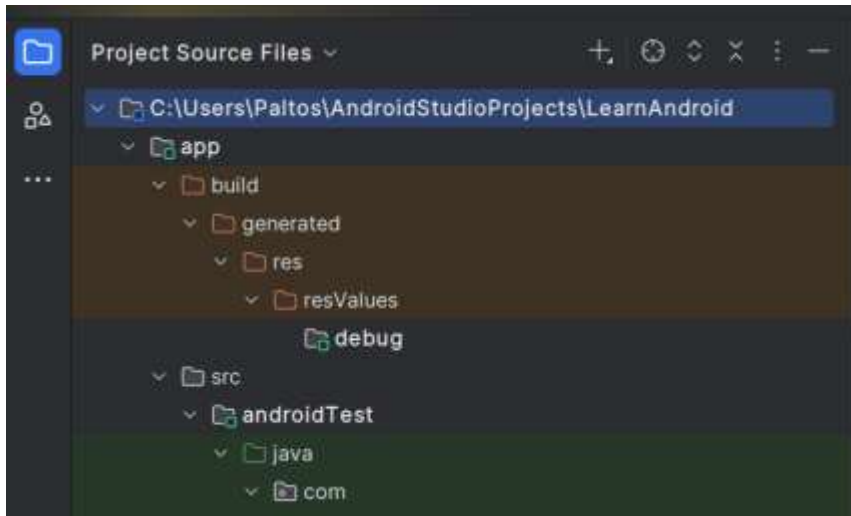


- Создание проекта

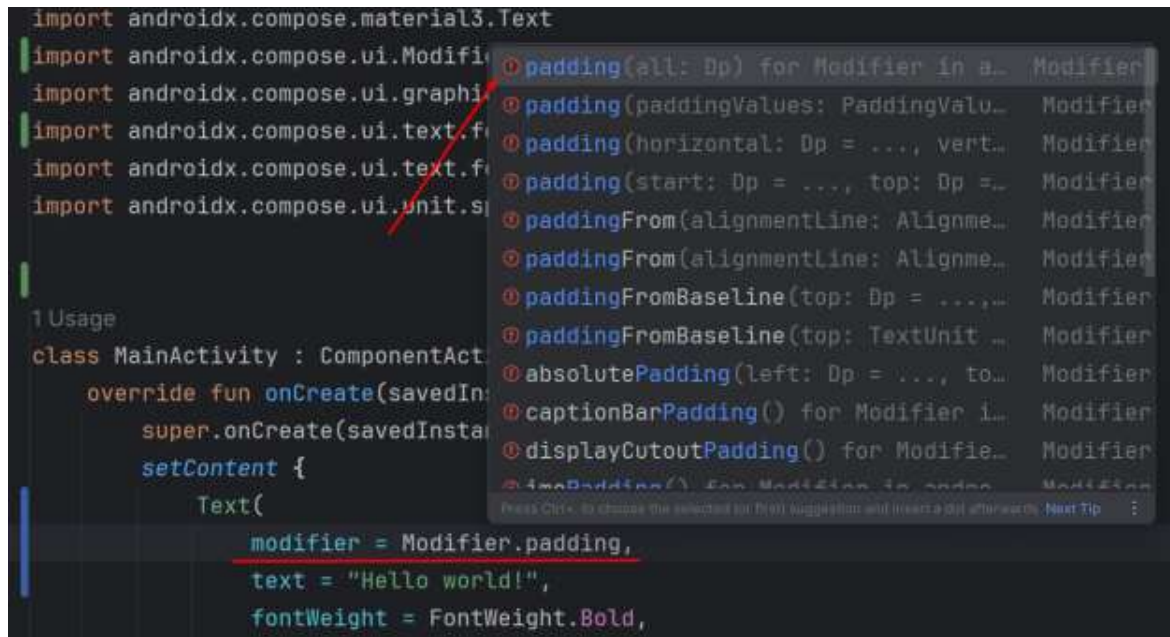




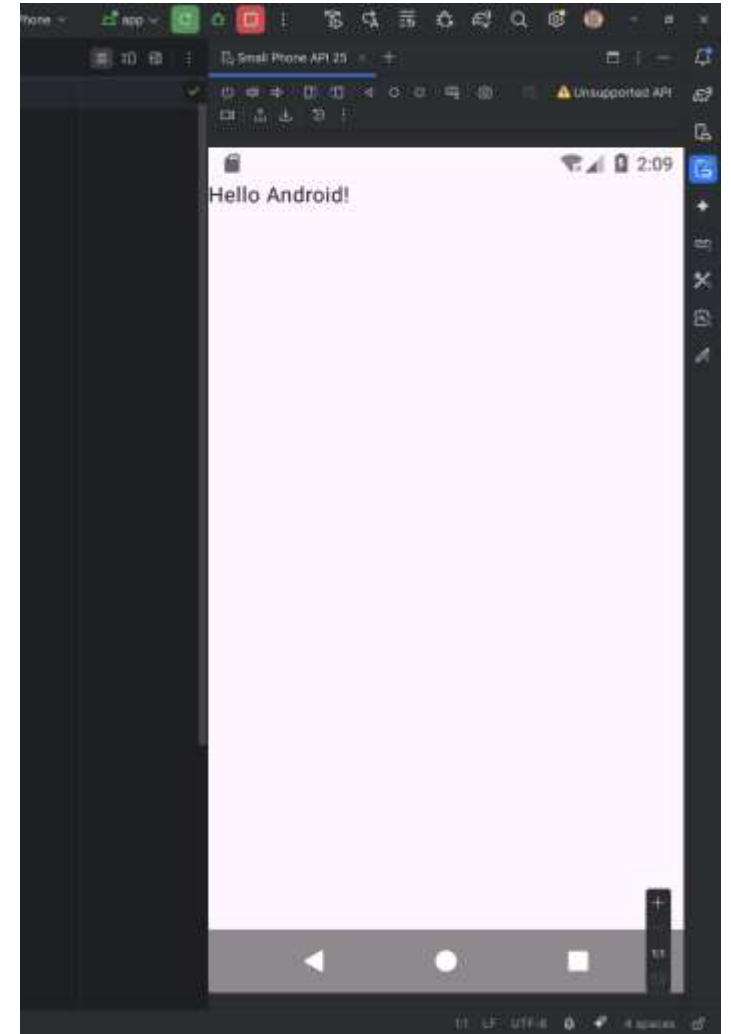
- **Android Studio**



- Устройство проекта



- Работа с кодом

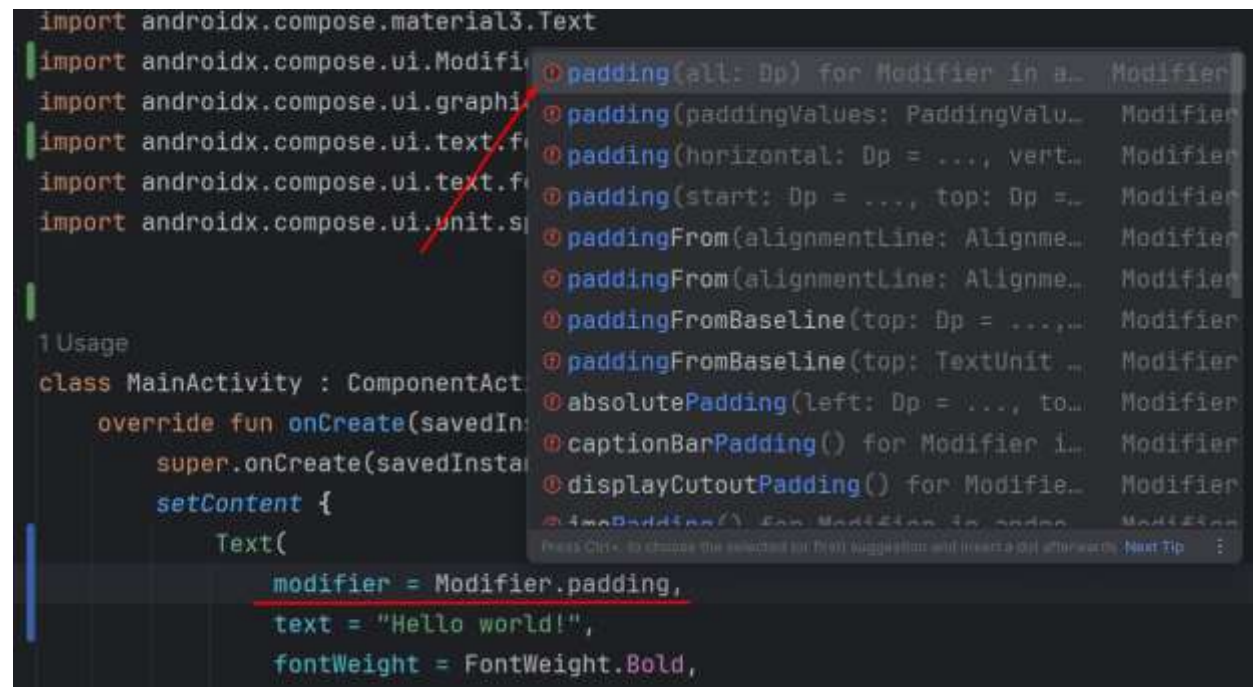


- Эмулятор

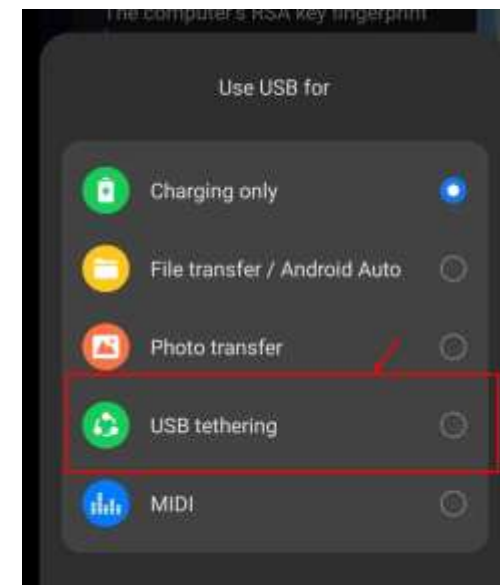
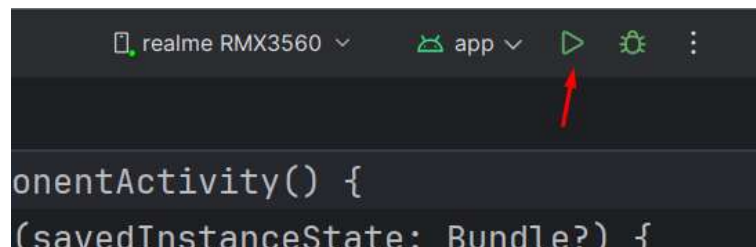
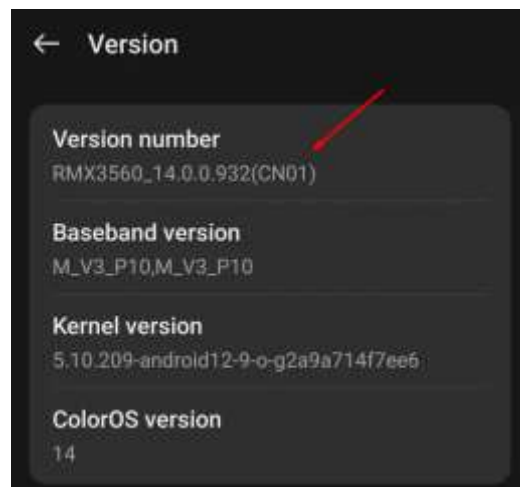
- Modifier



```
modifier = Modifier
    .background(color = Color.Red)
    .padding(start = 30.dp, top = 16.dp, end = 25.dp, bottom = 32.dp),
```



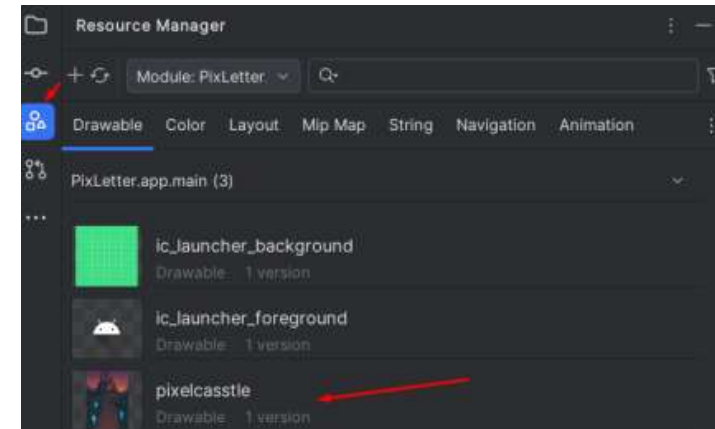
- Запуск на реальном устройстве



• Создание простого приложения



```
@Composable
fun GreetingImage(message: String, from: String, modifier: Modifier = Modifier) {
    val image = painterResource(id = R.drawable.pixelcastle)
    Box {
        Image(
            painter = image,
            contentDescription = null
        )
    }
}
```



```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            PixLetterTheme {
                GreetingText(message = "Royal Decree!\n Dear Denis\nHis Majesty the King\nInvites you to a great feast!\nCome to the castle before sunset,\notherwise, you risk losing your head!", from = "From King Arthur")
            }
        }
    }
}
```

```
<resources>
    <string name="app_name">PixLetter</string>
    <string name="king_message_text">Royal Decree!\n Dear ,
    \nDenis\nHis Majesty the King\nInvites you to a great ,
    \nfeast!\nCome to the castle before sunset,\notherwise, you
    \nrisk losing your head!</string>
</resources>
```

Баллы

- Посещение занятий;
- Работа на парах;
- Выполнение домашних работ;
- Защита мини-проектов.

Разработка мобильных приложений			
Кол-во занятий	Работа на парах (баллы)	ДЗ (баллы)	Инд.проекты (баллы)
1	3	2	15
2	3	2	
3	3	2	
4	3	2	
5	3	2	
6	3	2	
7	3	2	
8	3	2	
9	3	2	25
10	3	2	
11	3	2	
12	3	2	
Итог:	36	24	40
Общий итог:			100

Необходимо создать учётную запись на **github**.
Все домашние работы высылать **ссылкой** на
репозиторий (предварительно не забудьте
сделать его **публичным**).

Лучше заранее создать google учётную запись

Нативная vs Кроссплатформенная разработка. Выбор стратегии

Критерий	Нативная разработка	Кроссплатформенная разработка
Подход	Используются родные языки и инструменты для каждой платформы.	Единая кодовая база на одном языке компилируется под несколько платформ.
Для Android	Kotlin/Java + Android SDK	
Для iOS	Swift/Objective-C + iOS SDK	Flutter (Dart), React Native (JS), KMP (Kotlin)
Аналогия	Строить два отдельных особняка по индивидуальным проектам.	Строить два модульных дома из одних и тех же готовых блоков.
Главный вопрос	Максимальная производительность и глубокая интеграция с ОС?	Скорость разработки и единая команда?

Плюсы и минусы подходов

	Нативная разработка	Кроссплатформенная разработка
Преимущества	<ul style="list-style-type: none">• Максимальная производительность• Прямой доступ ко всем фичам ОС• Идеальное следование гайдлайнам<ul style="list-style-type: none">• Меньше багов и проблем с совместимостью	<ul style="list-style-type: none">• Скорость разработки: один код на две платформы<ul style="list-style-type: none">• Единая команда разработчиков• Синхронизация фич и дизайна на iOS и Android<ul style="list-style-type: none">• Легче поддерживать проект
Недостатки	<ul style="list-style-type: none">• Две команды разработчиков (iOS + Android)• Выше стоимость и время разработки• Риск расхождения фич и дизайна между платформами	<ul style="list-style-type: none">• Производительность ниже нативной• Доступ к новым фичам ОС с задержкой• Сложность отладки специфичных багов на платформах• Дополнительный слой абстракции (риск "выстрелить себе в ногу")

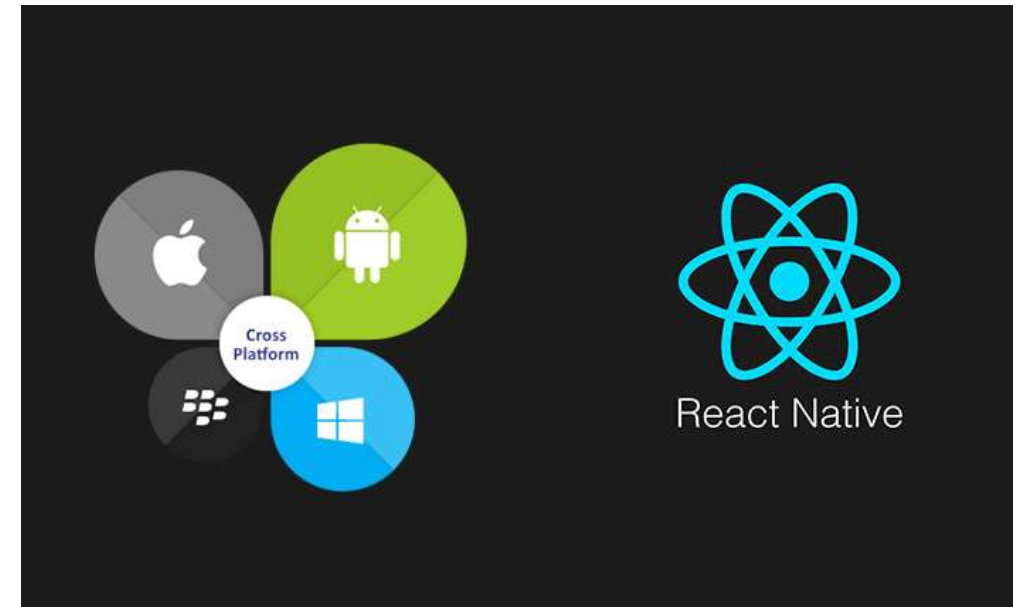
React Native

Подход: **"Learn once, write anywhere"**.

Использует JavaScript/TypeScript и родные компоненты.

Как работает: Ваш JS-код выполняется в отдельном потоке и общается с нативными модулями через "Мост" (Bridge). Интерфейс рендерится настоящими нативными View.

```
1 import { AppRegistry, Text, View, Button } from 'react-native';
2 import React from 'react';
3
4 const HelloWorldApp = () => {
5   const [count, setCount] = React.useState(0);
6
7   const incrementCount = () => {
8     setCount((prevCount) => prevCount + 1);
9   };
10
11   return (
12     <View>
13       <Text>Hello world!</Text>
14       <Text>{count}</Text>
15       <Button onPress={incrementCount} title="Increase Count" />
16     </View>
17   );
18 };
19
20 export default HelloWorldApp;
21
22 AppRegistry.registerComponent('HelloWorld', () => HelloWorldApp);
```



Плюсы:

- Огромное комьюнити, много готовых решений.
- Легко войти веб-разработчикам.
- "Горячее обновление" (Hot Reloading).

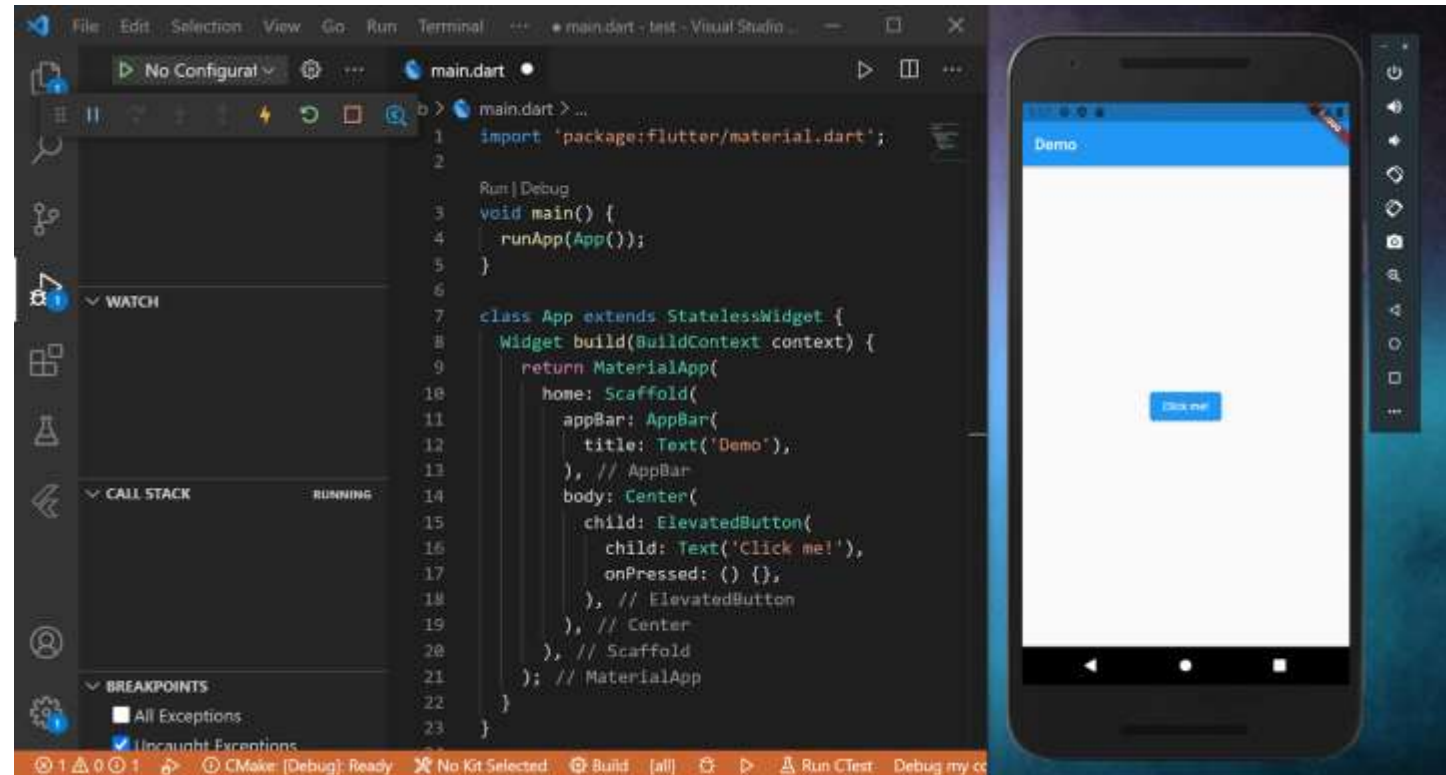
Минусы:

- Производительность может уступать из-за асинхронного моста.
- Сложность с нативными модулями, которые нужно писать отдельно.
- Зависимость от Meta.

Flutter (Google)

Подход: "**Build everything**". Использует язык Dart и собственный механизм рендеринга (Skia). Не использует нативные компоненты UI.

Как работает: Не требует моста. Код компилируется в нативный машинный код (AOT). Flutter сам рисует каждый пиксель на экране, эмулируя виджеты Material/Cupertino.



Плюсы:

- Высокая производительность, близкая к нативной.
- Полная consistency UI на всех платформах и версиях ОС.
- Огромные темпы развития и мощная поддержка Google.
- Горячая перезагрузка (Hot Reload) — лучшая на рынке.

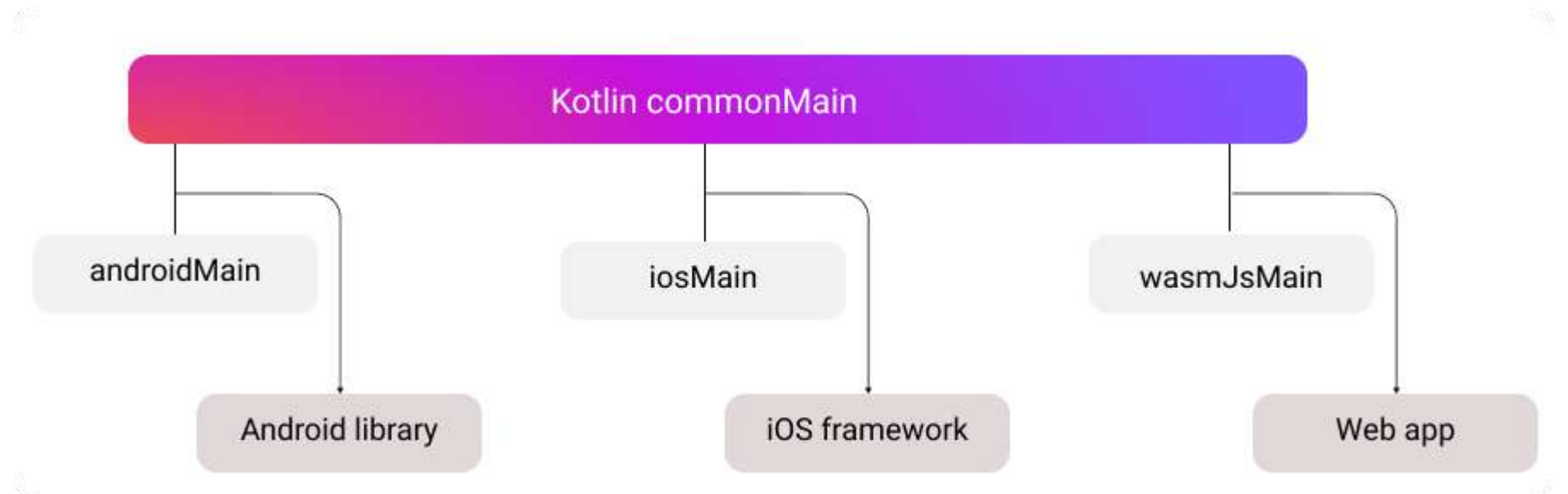
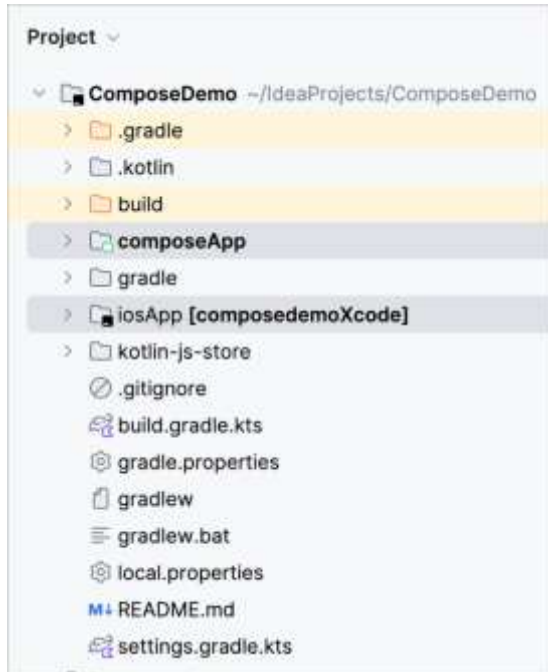
Минусы:

- Большой размер приложения ("вшитый" движок).
- UI не является нативно нативным, может отличаться в мелочах.
- Меньше вакансий, чем у React Native (но рост очень быстрый).

Kotlin Multiplatform (KMP/JetBrains)

Подход: **"Share business logic, keep UI native"**. Это не UI-фреймворк, а технология для переиспользования кода.

Как работает: Вы пишете на Kotlin общую бизнес-логику (сетевые запросы, кэширование, вычисления), которая компилируется в нативный код для JVM (Android), JavaScript (Web) и в бинарники (iOS). UI делается нативно на Swift и Kotlin.



Плюсы:

- Максимальная производительность, нет overhead как у UI-фреймворков.
- Нативные UI на всех платформах — идеальное следование гайдлайнам.
- Идеален для миграции большого нативного Android-приложения на iOS.

Минусы:

- Не подходит для быстрого прототипирования полного приложения.
- Нужны отдельные разработчики под каждую платформу (или fullstack-нативы).
- Молодая и более нишевая технология (но это меняется).

Критерий	React Native	Flutter	Kotlin Multiplatform
Язык	JavaScript / TypeScript	Dart	Kotlin
Подход к UI	Нативные компоненты	Собственный движок	Нативный UI (общая логика)
Производительность	Средняя (есть Bridge)	Высокая	Максимальная (нативный код)
Кривая обучения	Низкая (для веб-разработчиков)	Средняя (новый язык, концепции)	Высокая (требует знаний нативной разработки)
Комьюнити	Огромное	Очень большое	Растущее (сильное в Android-мире)
Идеальный кейс	Приложения с простым/стандартным UI, стартапы	Высокопроизводительные приложения с кастомным UI, MVP	Большие проекты, где нужно разделить логику и UI, миграция

Востребованность на рынке труда (2025, глобальные тренды)

Нативная разработка (Kotlin/Swift):

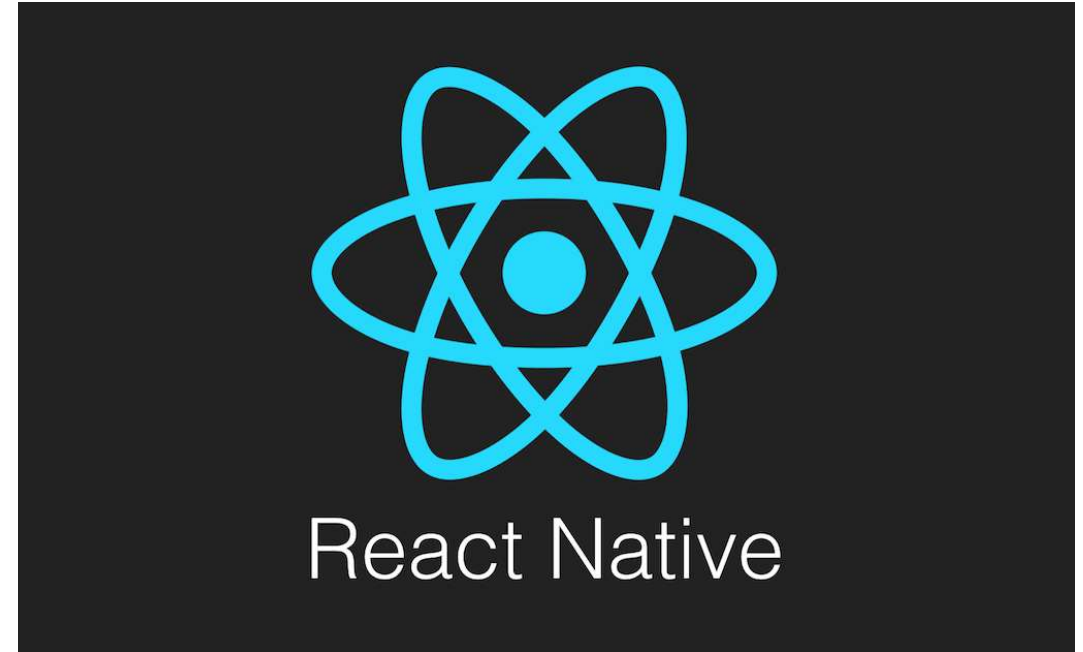
Основа рынка. ~60-70% всех вакансий.

Требуется в крупных продуктах (банки, маркетплейсы, соцсети), где критичны производительность и надежность.

Зарплаты: Стабильно высокие, высокий "потолок" для senior-специалистов.

React Native:

- Лидер по количеству вакансий в кроссплатформе. Около **~20%** рынка.
- Востребован в стартапах и проектах, где нужна быстрая разработка MVP.
- Много legacy-проектов, требующих поддержки.



Flutter:

- Самый быстрорастущий спрос.
~**10-15%** рынка и активно растет.
- Любим стартапами и компаниями, которые доверяют Google.
- Часто требуется как дополнительный навык к нативному разработчику.



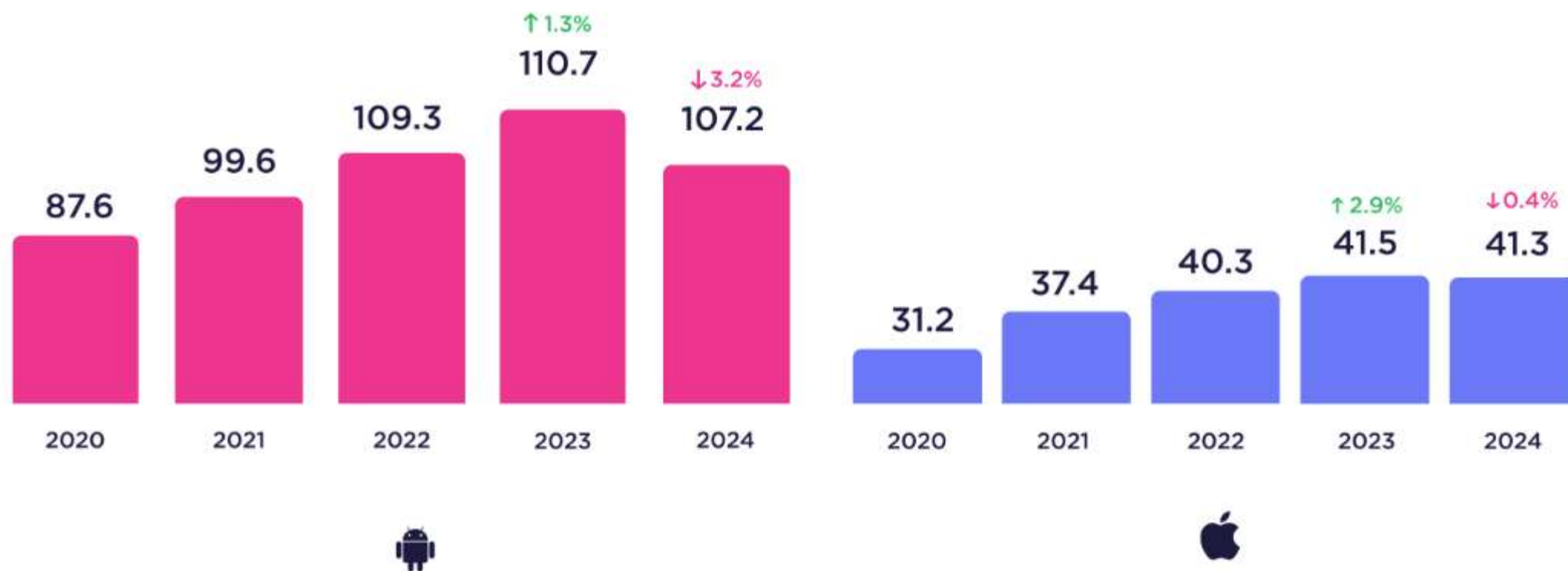
Flutter

Kotlin Multiplatform:

- Нишевая, но перспективная технология. ~<5% вакансий, но интерес к ней растет.
- Пока чаще встречается в крупных IT-корпорациях и банках, которые хотят делиться кодом между платформами.
- Ценный и высокооплачиваемый навык для Android-разработчиков.



Общее количество загрузок приложений в Google Play и App Store (в миллиардах)



Общий доход от приложений в Google Play и App Store (в миллиардах)

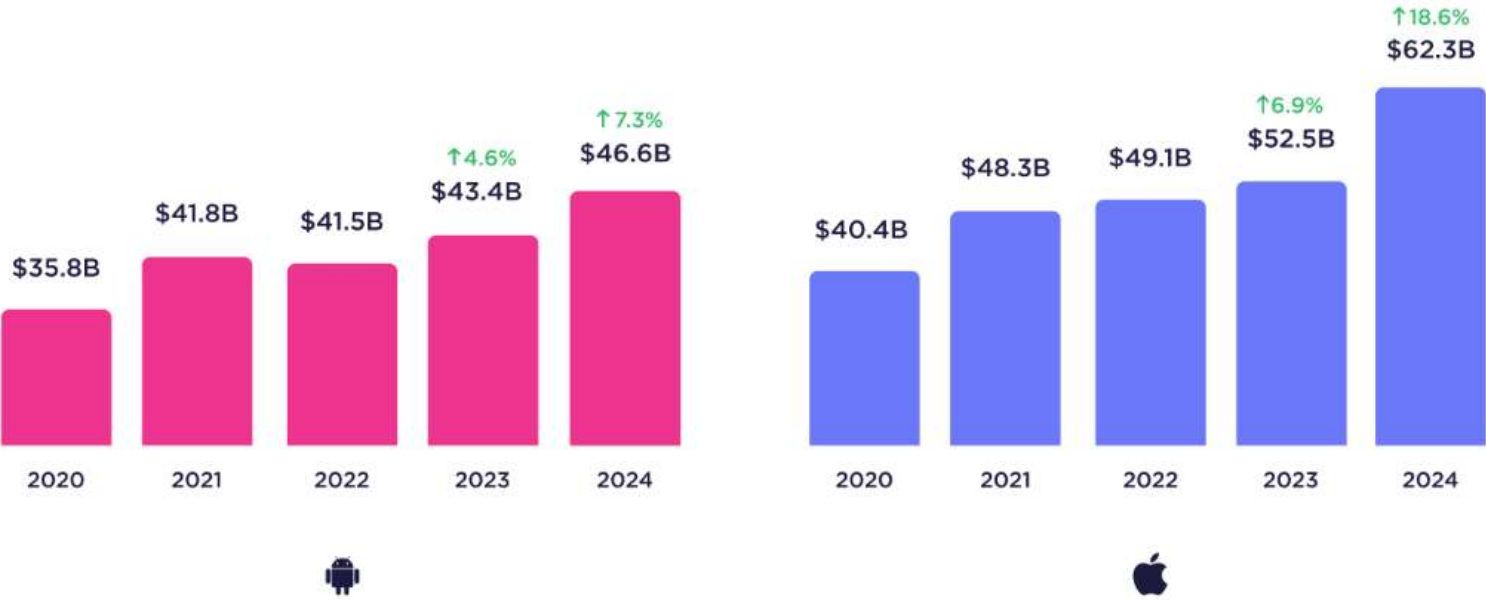


Диаграмма 1. Динамика изменения выручки крупнейших игроков рынка от услуг мобильной корпоративной разработки 2012-2023 гг., млрд руб.



Источник: CNews Analytics

Consumer Spend by Apps vs. Games

iOS and Android



Почему именно нативная разработка?

- Нативные приложения = высокая производительность, доступ ко всем возможностям устройства (Bluetooth, камера, GPS, датчики).
- Web-приложения и кроссплатформенные решения (Flutter, React Native) хороши, но ограничены.
- Android — самая распространённая мобильная ОС в мире (более 70% рынка).

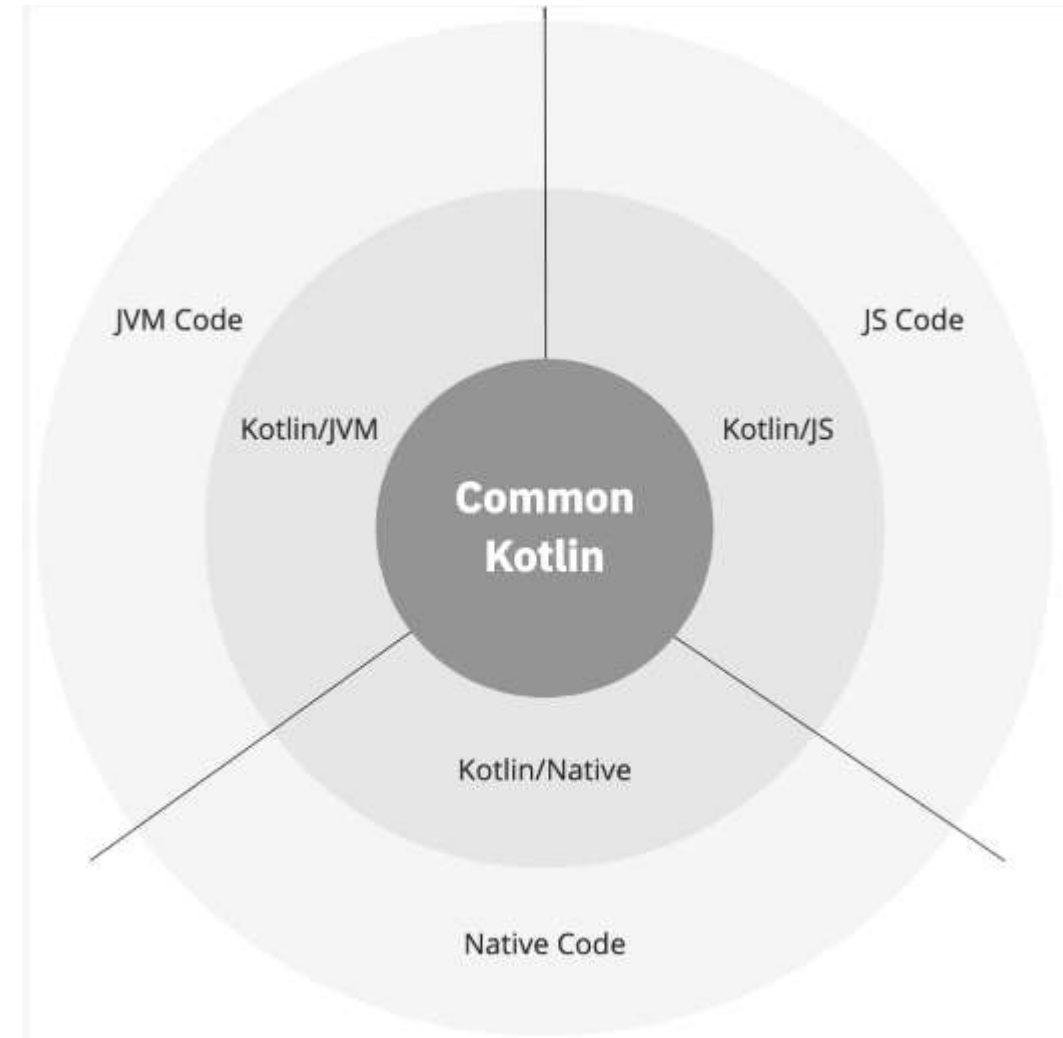
Философия Kotlin

- Kotlin — это pragmatic language
- Создан JetBrains (создателями IDE IntelliJ IDEA) для решения реальных практических задач.
- Главная цель: Сделать код красивым, читаемым, безопасным и при этом полностью совместимым с Java.
- Девиз: «Kotlin — это лаконичная, безопасная и совместимая альтернатива Java».
- Официальный статус: С 2019 года — предпочтительный язык для разработки под Android по версии Google.

Kotlin - это не просто очередной язык программирования. На сегодняшний день это целая экосистема:

Ядро этой экосистемы - **Common Kotlin**, которое включает в себя собственно язык, основные библиотеки и базовые инструменты для построения программ.

Для взаимодействия с конкретной платформой имеются предназначенные для этой платформы версия Kotlin: **Kotlin/JVM**, **Kotlin/JS** и **Kotlin/Native**. Эти специфические версии представляют расширения для языка Kotlin, а также специфичные для конкретной платформы библиотеки и инструменты разработки.



Сравнение Kotlin с языком Python

Kotlin vs Python: Общие черты (Почему он будет вам понятен)

То, что вы уже оценили в **Python**, есть и в **Kotlin**:

Особенность	Python	Kotlin	Выгода
Лаконичность	<code>print("Hello")</code>	<code>println("Hello")</code>	Меньше кода -> меньше ошибок
Null-безопасность	<code>None</code>	<code>null</code> (но безопаснее!)	Защита от частых ошибок
Умный вывод типов	<code>a = 10</code>	<code>val a = 10 // Int</code>	Не нужно писать тип везде
Data-классы	<code>@dataclass</code>	<code>data class User(...)</code>	Автоматические методы для данных
Функции высшего порядка	<code>lambda x: x*2</code>	<code>{ it * 2 }</code>	Мощные инструменты для работы с коллекциями

Kotlin vs Python: Ключевые отличия

А вот где **Kotlin** становится мощнее и надежнее:

Критерий	Python (интерпретируемый, динамический)	Kotlin (компилируемый, статический)	Преимущество Kotlin
Типизация	Динамическая (тип проверяется при запуске)	Статическая (тип проверяется при компиляции)	Ошибки типов находятся ДО запуска программы. Надежнее!
Безопасность NPE	None может вызвать ошибку в runtime	Система типов отличает nullable от non-null (String vs String?)	Компилятор НЕ ДАСТ вам совершить ошибку с null
Производительность	Медленнее (интерпретатор)	Быстро (компилируется в байт-код JVM, как Java)	Высокая скорость выполнения, как у нативных приложений
Мультипарадигменность	ООП, ФП	ООП, ФП + Coroutines (лучшая асинхронность)	Удобнее писать современный асинхронный код
Применение	Скрипты, ML, бэкенд, автоматизация	Профессиональная разработка: Android, серверы, десктоп	Промышленная мощь для больших и сложных проектов

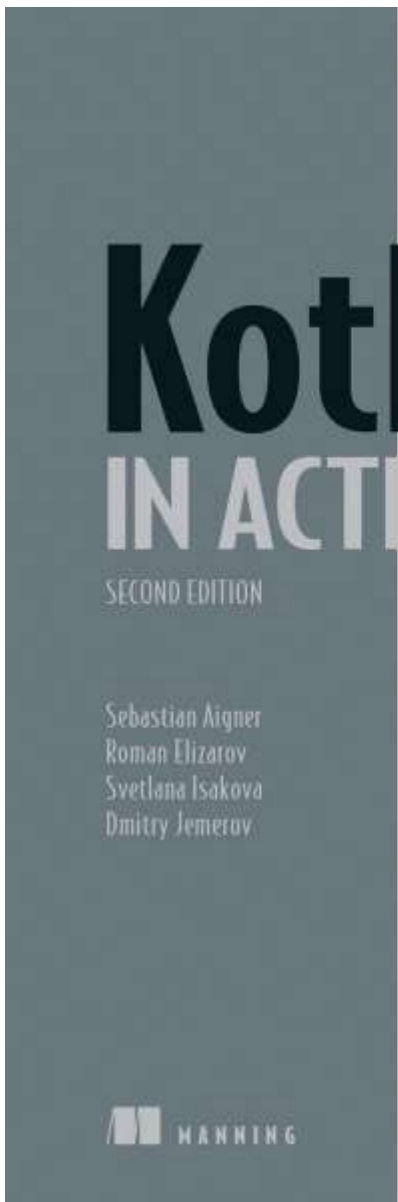
Kotlin vs C#

Аспект	Kotlin	C#	Что это значит для вас
Синтаксис	Лаконичный, минималистичный	Более вербозный , требует больше служебных слов	Kotlin учится быстрее, код читается легче
Null Safety	Встроена в язык с самого начала	Появилась позже (Nullable типы)	В Kotlin система проще и логичнее
Совместимость	100% совместим с Java	Работает в экосистеме .NET	Вы получаете доступ ко всем библиотекам Java
Цель	Практичность, лаконичность	Мощь, универсальность	Kotlin сфокусирован на решении задач разработчика

Почему Kotlin — это здорово?

- 1.Безопасный:** Компилятор ловит ошибки за вас.
- 2.Лаконичный:** Писать мало — читать легко.
- 3.Совместимый:** Весь мир Java-библиотек в вашем распоряжении.
- 4.Выразительный:** Код читается как предложение на английском.
- 5.Современный:** Все последние тренды (корутины) встроены в язык.
- 6.Официальный:** Поддержка Google — это стандарт для Android.

Рекомендуемая литература и ресурсы



Для начинающих



O'REILLY®

Программирование на Kotlin для Android



Для продолжающих

Пьер-Оливье Лоранс,
Аманда Хинчман-Домингес,
Дж. Блейк Мик, Майк Данн

O'REILLY®

Head First Kotlin

Руководство для начинающих программистов

Как избежать
идиотских
ошибок
в лямбда-
выражениях



Пишем
функции
высшего
порядка
не-от-мира-
сего



Все, что вы
хотели знать
о джене-
риках



Как
Элвис
может
изменить
вашу
жизнь

Коллек-
ции под
микро-
скопом



Развлечения
с Kotlin
Standard
Library



Дон Гриффитс и Дэвид Гриффитс



O'REILLY®

Head First

Программирование для Android на Kotlin

Дон Гриффитс
Дэвид Гриффитс



Третье
издание



ПОДАРОК ДЛЯ МОЗГА

Может зайти, но немного устарели

Manifest Android Interview

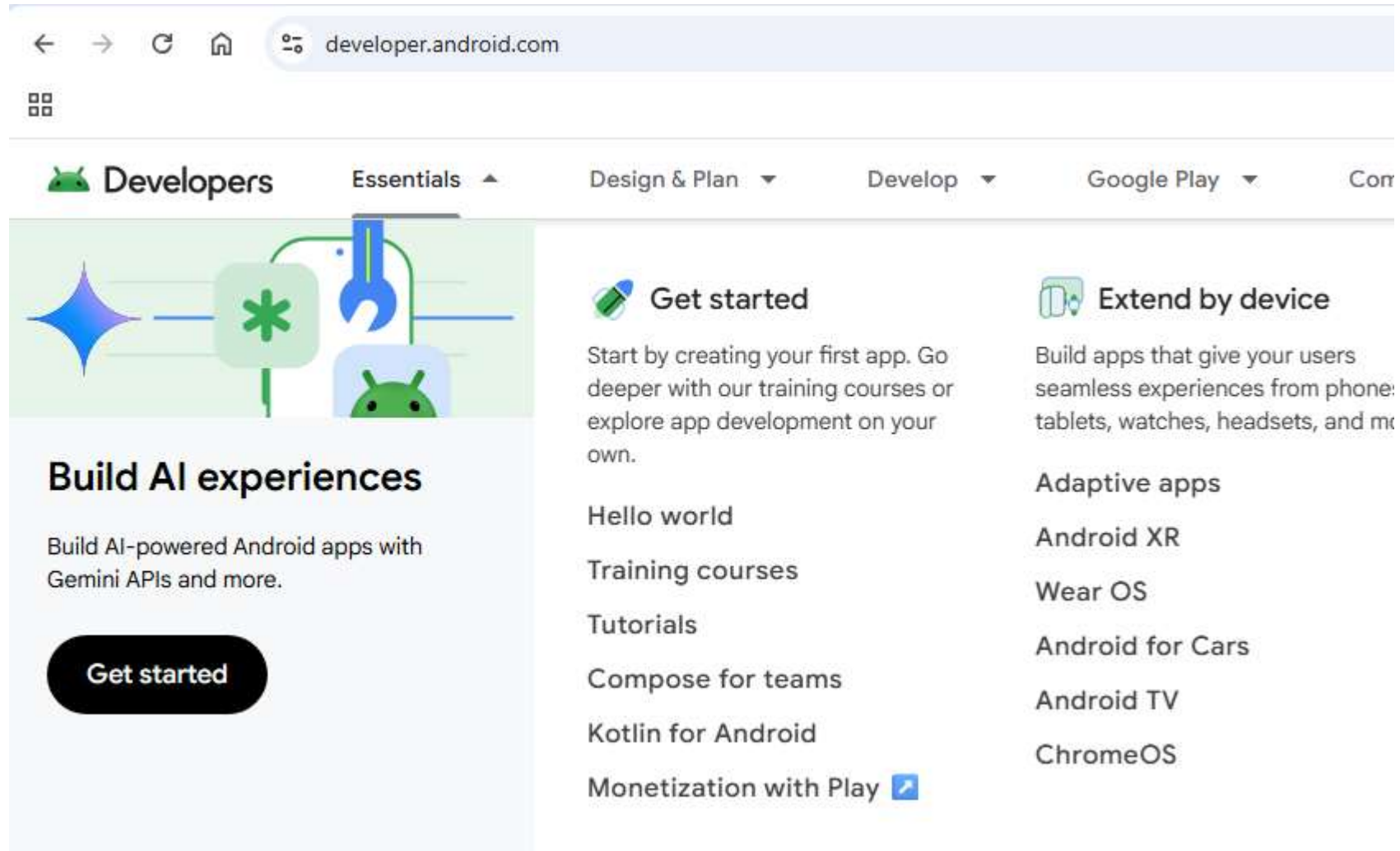
The ultimate guide to cracking
Android technical interviews

Jaewoong Eum (skydoves)

Неплохой справочник

Android developer

<https://developer.android.com/>




The screenshot shows the Android Developer website. At the top, there's a navigation bar with the Android logo, 'Developers', 'Essentials', 'Design & Plan', 'Develop', 'Google Play', and 'Corr'. Below this, there's a large hero section on the left with the title 'Build AI experiences' and a 'Get started' button. To the right of this, there are two columns of links. The first column is titled 'Get started' and includes links for 'Hello world', 'Training courses', 'Tutorials', 'Compose for teams', 'Kotlin for Android', and 'Monetization with Play'. The second column is titled 'Extend by device' and includes links for 'Adaptive apps', 'Android XR', 'Wear OS', 'Android for Cars', 'Android TV', and 'ChromeOS'.

← → ↺ 🏠 🔍 developer.android.com

☰


🤖 Developers Essentials ▲ Design & Plan ▼ Develop ▼ Google Play ▼ Corr



Build AI experiences


Build AI-powered Android apps with Gemini APIs and more.


Get started



Get started

Start by creating your first app. Go deeper with our training courses or explore app development on your own.

- Hello world
- Training courses
- Tutorials
- Compose for teams
- Kotlin for Android
- Monetization with Play 



Extend by device

Build apps that give your users seamless experiences from phone: tablets, watches, headsets, and more.

- Adaptive apps
- Android XR
- Wear OS
- Android for Cars
- Android TV
- ChromeOS

Developers

Essentials

Design & Plan

Develop

Google Play

Community

PLATFORM

OverviewReleasesTechnology

Filter

Platform Architecture

Android 5G

Android for enterprise

Google Play Instant

Guide for building secure government apps

Generic System Images

Dynamic System Updates

SDK Extensions

Android Developers > Essentials > Platform > Technology

Platform architecture

Android is an open source, Linux-based software stack created for a wide range of devices. This section shows the major components of the Android platform.

System Apps

DialerEmailCalendarCamera...

Java API Framework

Content ProvidersView SystemManagersActivityLocationPackageNotificationResourceTelephonyWindow

Native C/C++ Libraries

WebkitOpenMAX ALLibcMedia FrameworkOpenGL ES...

Android Runtime

Android Runtime (ART)

Core Libraries

Considering the common architectural principles mentioned in the previous section, each application should have at least two layers:

- The *UI layer* that displays application data on the screen.
- The *data layer* that contains the business logic of your app and exposes application data.

You can add an additional layer called the *domain layer* to simplify and reuse the interactions between the UI and data layers.

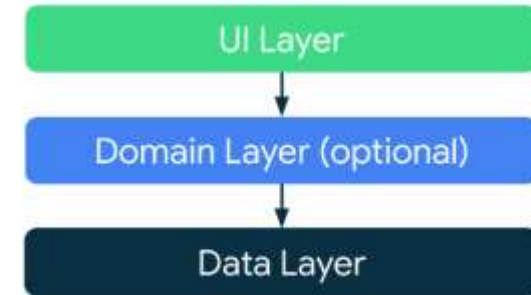


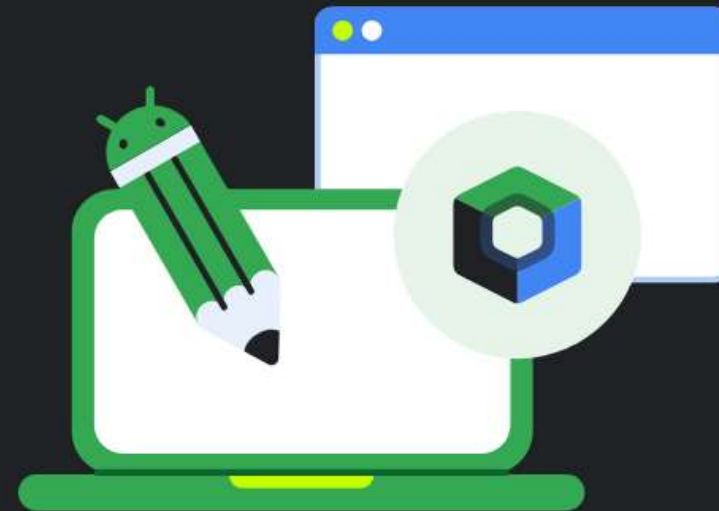
Figure 1. Diagram of a typical app architecture.

Подробная документация

Android Basics with Compose

Android Basics with Compose is a self-paced, online course on how to build Android apps using the latest best practices. It covers the basics of building apps with Jetpack Compose, the recommended toolkit for building user interfaces on Android.

Start course



Course goals

- Understand the fundamentals of Kotlin and Jetpack Compose
- Build a series of basic Android apps
- Use Android app architecture, Jetpack libraries and work with data according to Modern Android Development practices



Prerequisites

- Basic computer skills
- Basic math skills
- A computer that can run Android Studio (see system requirements)
- (Optional) Android device



Хорошие курсы

Документация по Kotlin

<https://kotlinlang.org/docs/home.html>



Solutions

Docs

API

Communi

Home

Get started

Take Kotlin tour

▸ Kotlin overview

▸ What's new in Kotlin

▸ Kotlin evolution and roadmap

▸ Basics

▸ Concepts

▸ Data analysis

▸ Platforms

Develop with Kotlin Multiplatform ↗

▸ Standard library

▸ Official libraries

▸ API reference

▸ Language reference

▸ Tools

▸ Compiler and plugins

▸ Learning materials

▸ Early access preview (EAP)

Kotlin docs

Latest stable version: 2.2.10



Get started with Kotlin

Create your first Kotlin project for a platform of your choice in an IDE: IntelliJ IDEA or Android Studio



Try Kotlin online

Write, edit, run, and share Kotlin code right in the browser

First steps

Basic syntax

A quick introduction to Kotlin syntax: keywords, operators, program structure

Kotlin tour

Take a tour of the fundamentals of the Kotlin programming language

Hyperskill

<https://hyperskill.org/courses>

The screenshot displays the Hyperskill website's course catalog. At the top, the 'Hyperskill' logo is on the left, and navigation links for 'Catalog', 'Study plan', 'Map', and 'My Learning' are in the center. A search bar with the placeholder 'Find topic' and a user profile icon with '93' are on the right. The main content area is divided into four columns: 'Languages' (Python, Java, Kotlin & Android, SQL and Databases, Go & C++), 'Career paths' (Web Dev, Backend, DevOps, Data Science & Analysis), 'Subjects' (AI & AI Coding Tools, ML & Math), and 'Resources' (Blog, University, Guide, Bootcamps). An 'Early Access' section with 'Drafts' is also visible. A 'Full catalog' button is located below the 'Subjects' column. At the bottom, the 'JetBrains Academy' logo and name are shown, along with text stating they are the creators of popular IDEs and authors of Kotlin. A dark blue box on the right contains the text: 'More than 11.4 million developers and 300,000 companies worldwide use JetBrains IDEs to create their products'.

Languages

- Python
- Java
- Kotlin & Android
- SQL and Databases
- Go & C++

Career paths

- Web Dev
- Backend
- DevOps
- Data Science & Analysis

Subjects

- ✨ AI & AI Coding Tools
- ML & Math


Early Access

- Drafts

[Full catalog](#)

Resources

- Blog
- University
- 💡 Guide
- Bootcamps

Provided by  **JetBrains Academy**

Creators of the most popular IDEs: IntelliJ IDEA, PyCharm & others
Authors of Kotlin, Google's preferred language for Android

More than 11.4 million developers and 300,000 companies worldwide use JetBrains IDEs to create their products

What do you want to learn today?

Select a course that fits your interests and experience level

All courses 59

Most popular 6

Python 13

Java 9

Kotlin & Android 7

🔥 AI & AI Coding Tools 6

Web Dev 6

Backend 10

DevOps 4

Data Science & Analysis 10

ML & Math 4

SQL and Databases 6

Go & C++ 3

Certificate

Kotlin Core

★ 4.6

📁 28 projects ⌚ 134 hours

Kotlin, developed by JetBrains, is Google's preferred language for Android app development. Master the language under the guidance of its original creators.



JetBrains Academy

90K already learning

Certificate

Kotlin Developer

★ 4.6

📁 35 projects ⌚ 187 hours

Learn Kotlin for backend, frontend, and Android app development to expand your reach across multiple platforms with a concise language developed by JetBrains.



JetBrains Academy

46K already learning

Certificate

Android Developer with Kotlin

★ 4.5

📁 13 projects ⌚ 90 hours

Learn about UI design, data storage, dynamic screen building, and leverage the capabilities of Android Studio to start your career in mobile app industry.



JetBrains Academy

11K already learning

Certificate

Introduction to Kotlin

★ 4.5

📁 9 projects ⌚ 28 hours

Start your first exciting journey with Kotlin programming! Discover how to work with variables, control code flow, and grasp essential concepts of object-oriented programming.



JetBrains Academy

6K already learning

Certificate

Jetpack Compose for Android Developers



Learned topics: 7 / 148 4%



Applied topics: 0 / 128 0%



Course

Project



My First Project with Kotlin

[Change project →](#)

Imagine that you're the owner of a new small corner shop. You are doing well and want to determine how much you earned in the first month. Create a simple tool that will calculate your net income. Upon completi...

Stages: 0/3 ☐ ☐ ☐

▼ Stage 1/3: Print the prices



2 completed activities

[Expand](#)

Writing first program

Problem of the day



⌚ 24 seconds 💎 Reward

Solve a random problem to practice what you've learned.

Next problem in 11h 32m 48s

[Solve unlimited](#)

Personalize your study plan



⌚ 5 minutes 💎 + 25

Solve a few problems to help the platform adapt to your level.

Repeat what you've learned



⌚ 15 minutes 💎 + 10

We recommend you to repeat 5 topics.

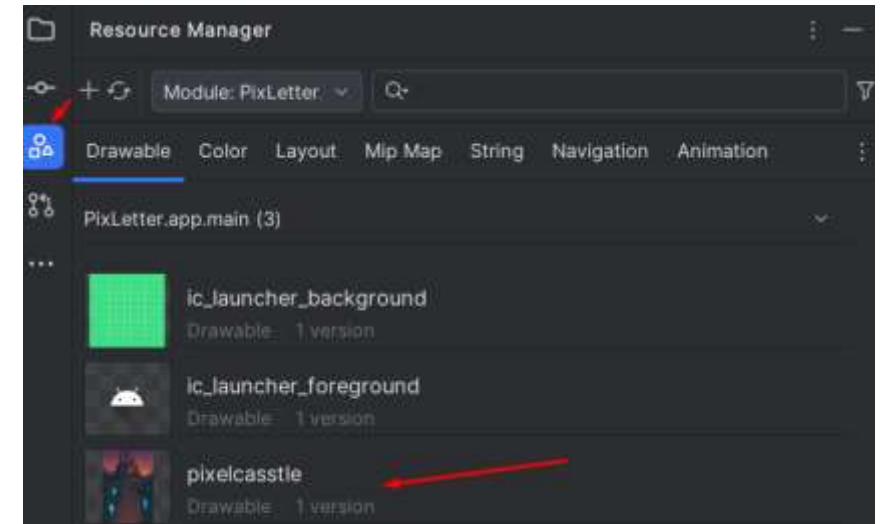
[Repeat unlimited](#)

Что такое Android-приложение

Android-приложение — это...

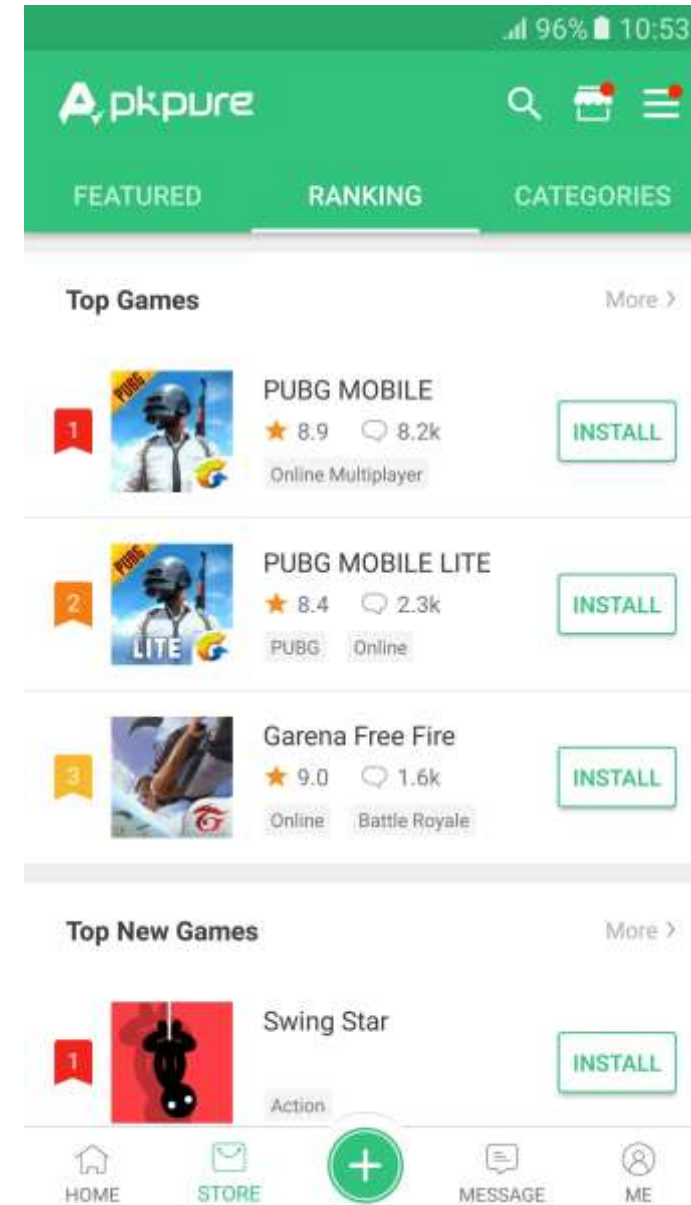
Не просто иконка на экране. Это сложный программный комплекс. Набор скомпилированного кода (обычно на Java или Kotlin), ресурсов (картинки, строки) и мета-данных.

```
16  Usage
17  class MainActivity : ComponentActivity() {
18      override fun onCreate(savedInstanceState: Bundle?) {
19          super.onCreate(savedInstanceState)
20          enableEdgeToEdge()
21          setContent {
22              LearnAndroidTheme {
23                  Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
24                      Greeting(
25                          name = "Android",
26                          modifier = Modifier.padding(paddingValues = innerPadding)
27                      )
28                  }
29              }
30          }
31      }
32  }
```



Android-приложение — это...

Архивный файл (**APK** — Android Package Kit или AAB — Android App Bundle), который содержит всё необходимое для установки и работы на устройстве.



Android-приложение — это...

Набор компонентов, которые система Android может запускать и использовать.

Ключевая особенность: В Android нет единой точки входа (main() функция). Приложение — это набор независимых компонентов, которые система запускает по мере необходимости.

Compiled java code to machine code.



```
A 002000 C2 30 REP #E30
A 002002 1B CLC
A 002003 FB SED
A 002004 AS 34 12 LJA #01234
A 002007 69 21 43 ADC #04321
A 00200A 0F 03 7F 01 STA #017F03
A 00200E 36 CLD
A 00200F E2 30 SEP #430
A 002011 00 BRK
A 2012

r
PS PC Mmu012C .A .X .Y SP .IP .00
: 00 E012 00110000 0000 0000 0002 CFF 0000 00
S 2000
BREAK
PS PC Mmu012C .A .X .Y SP .IP .00
: 00 2013 00110000 5555 0000 0002 CFF 0000 00
M 7103 7103
007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

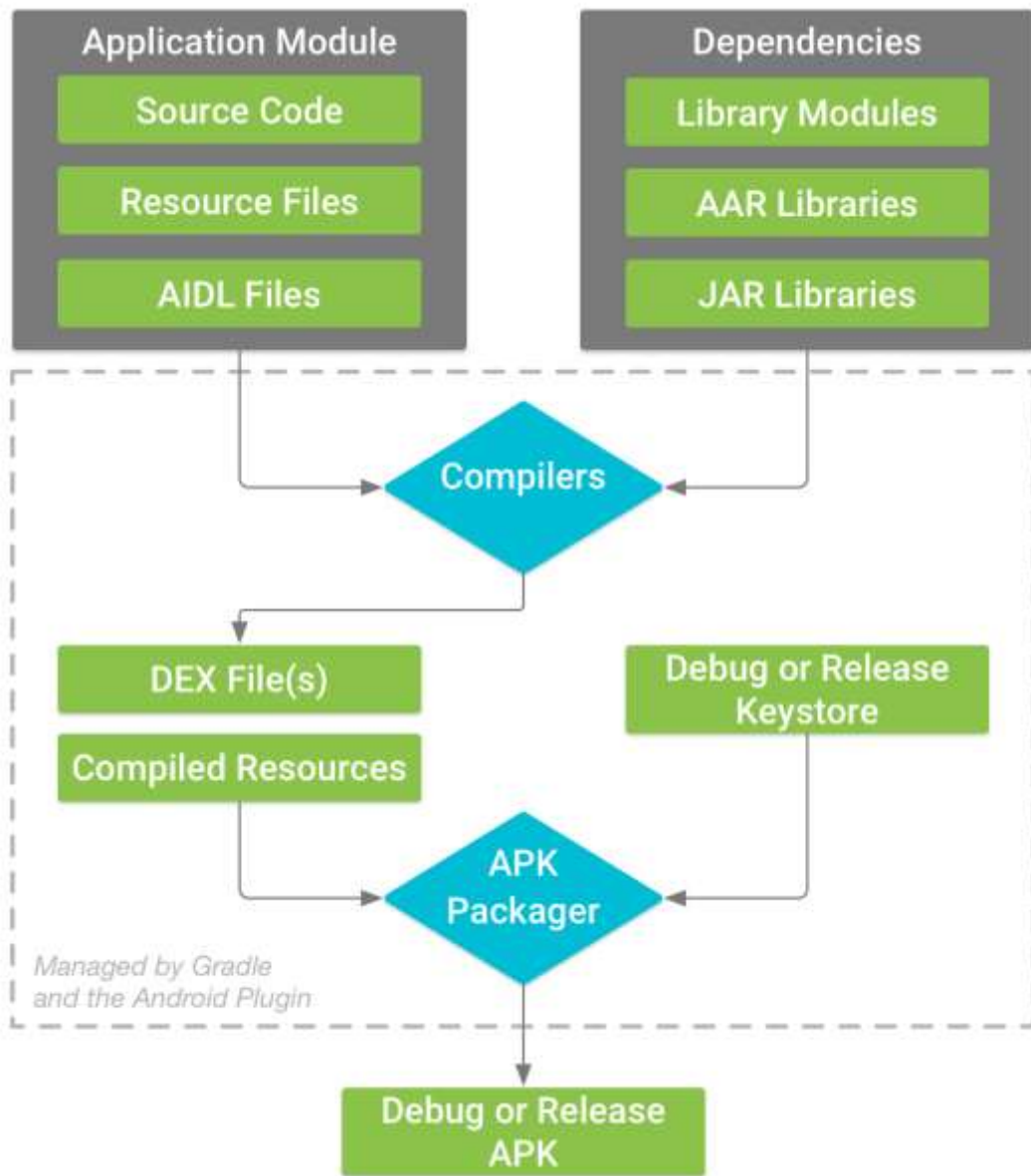
By using the JVM, Different types of CPU architecture are satisfied.

```
javac compiler
[getz@optix]~$ find
./HelloWorld.java
./lib
./lib/gson-1.7.1.jar
[getz@optix]~$ javac -cp lib/gson-1.7.1.jar HelloWorld.java
[getz@optix]~$ find
./HelloWorld.class
./HelloWorld.java
./lib
./lib/gson-1.7.1.jar
[getz@optix]~$
```

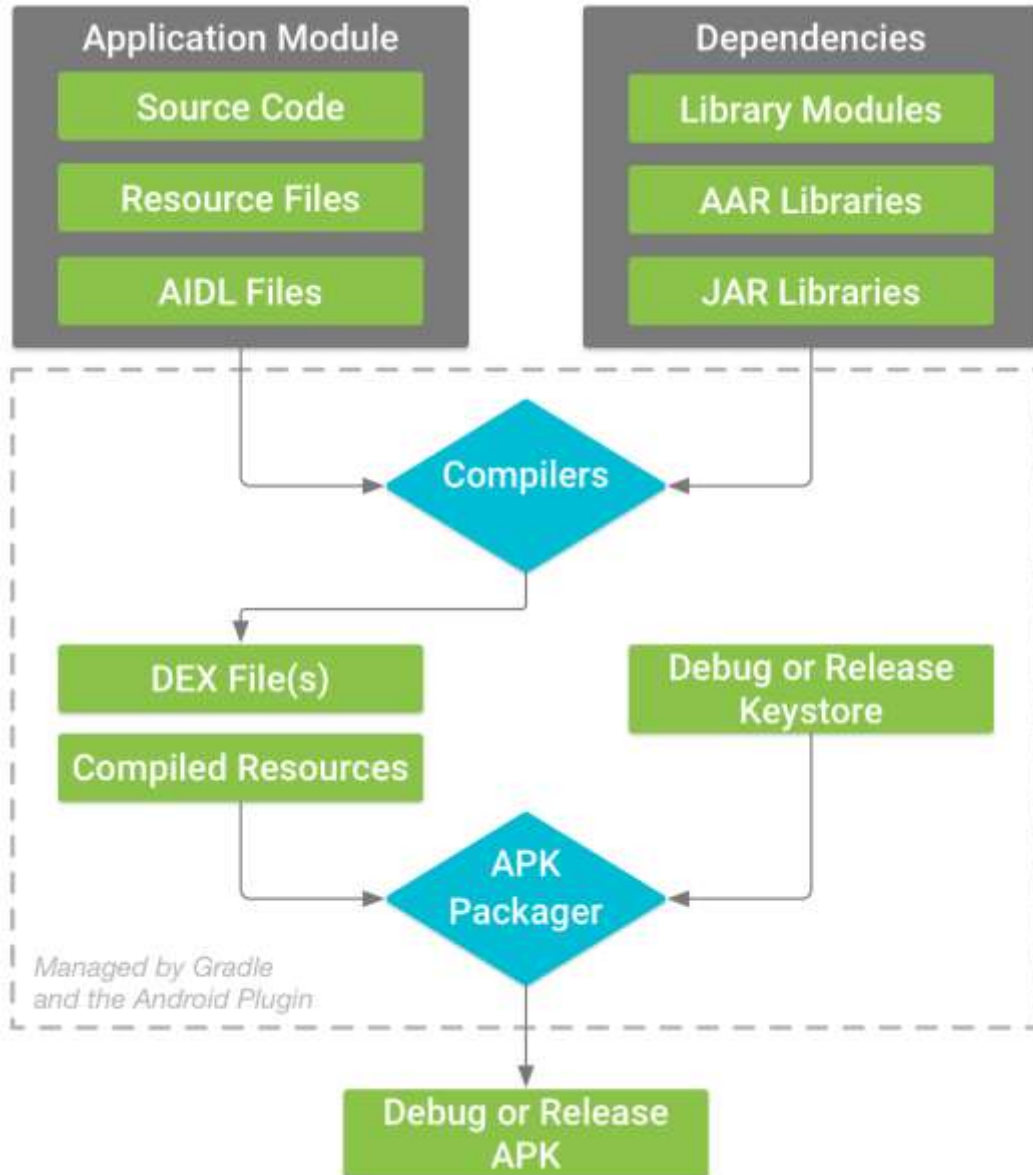
```
JetBrains / kotlin
kotlin compiler
Code Pull requests Actions Security Insights
kotlin / compiler / cli / bin / kotlin
yandex Kotlin, breakpoint, Kotlin, commonmark, kotlin, Kotlin, Kotlin, Kotlin
```

```
// Bytecode stream: 03 3b 04 00 01 1a 05
60 3b a7 ff f9
// Disassembly:
iconst_0 // 03
istore_0 // 3b
iinc 0, 1 // 04 00 01
iload_0 // 1a
iconst_2 // 05
imul // 60
istore_0 // 3b
goto -7 // a7 ff f9
```

Bytecode



Процесс сборки (Build Process)



Исходный код (Kotlin/Java) -> Компилятор -> Java Bytecode (.class files)

Java Bytecode + Библиотеки -> Инструмент dx/d8 -> Dalvik Bytecode (.dex files) (исполняется Android Runtime)

Ресурсы (res/) -> Компилятор ресурсов (aapt2) -> Скомпилированные ресурсы

Все вместе (.dex + скомпилированные ресурсы + манифест) -> Упаковывается и подписывается -> APK или AAB файл.

[Код] -> [Компиляция] -> [Упаковка] -> [APK]

**Любое Android приложение
состоит из четырех
основных типов
компонентов**

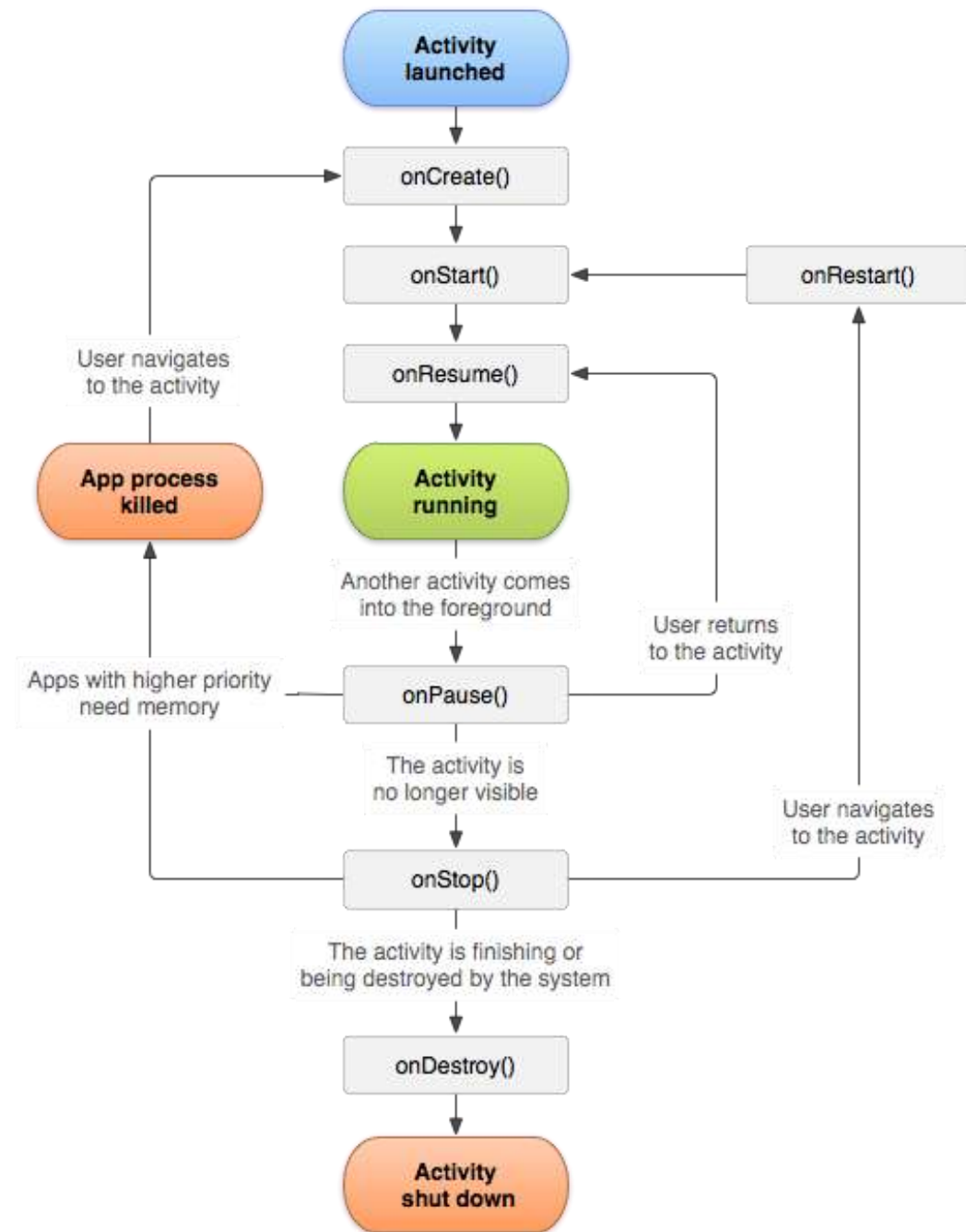
Activity (Активность)

Что это?

Окно приложения, один экран с пользовательским интерфейсом.

Аналогия: Отдельная страница в веб-браузере.

Пример: MainActivity (главный экран),
SettingsActivity (экран настроек).



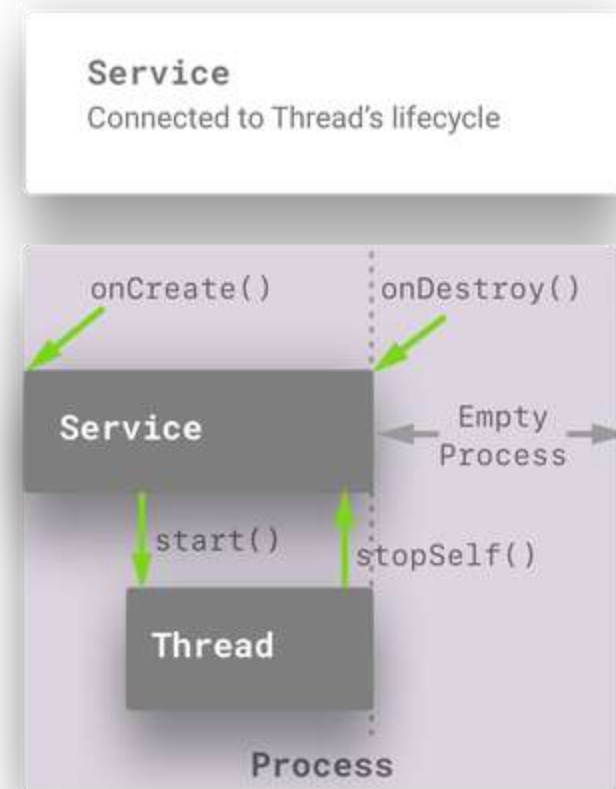
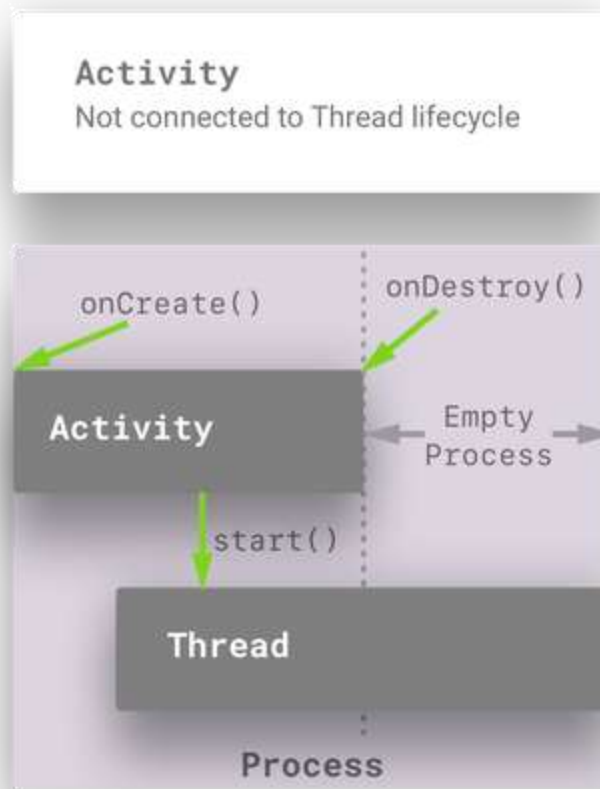
Service (Служба)

Что это?

Компонент для выполнения долгих операций в фоне без пользовательского интерфейса.

Аналогия: Музыкальный проигрыватель, который продолжает играть музыку, даже когда вы свернули приложение.

Пример: `MusicPlayerService`, `DownloadService`.



Broadcast Receiver (Приемник широковещательных сообщений)

Что это?

Компонент, который слушает системные или глобальные события.

Аналогия: Система оповещения о новостях. Вы подписываетесь на событие (например, "разряжена батарея") и получаете уведомление.

Пример: Реакция на входящий звонок, смену режима "в самолете", зарядку устройства.



Content Provider

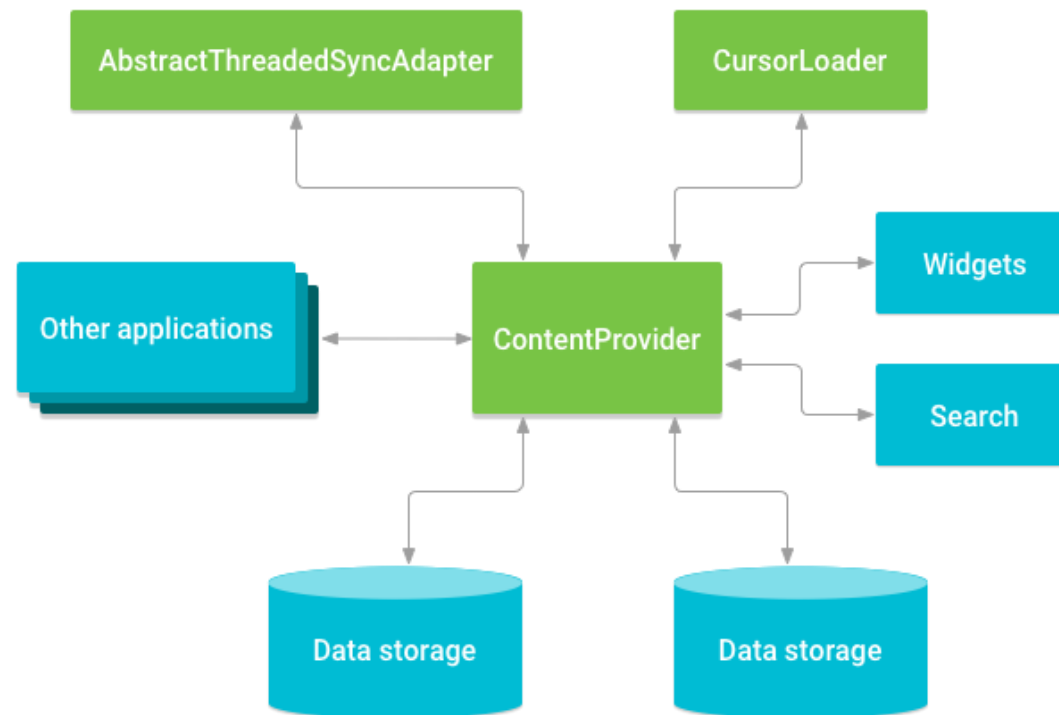
(Поставщик контента)

Что это?

Компонент для управления общим набором данных и предоставления к ним доступа другим приложениям.

Аналогия: Общая база данных, к которой можно предоставить безопасный доступ с запросами (как к SQL-базе).

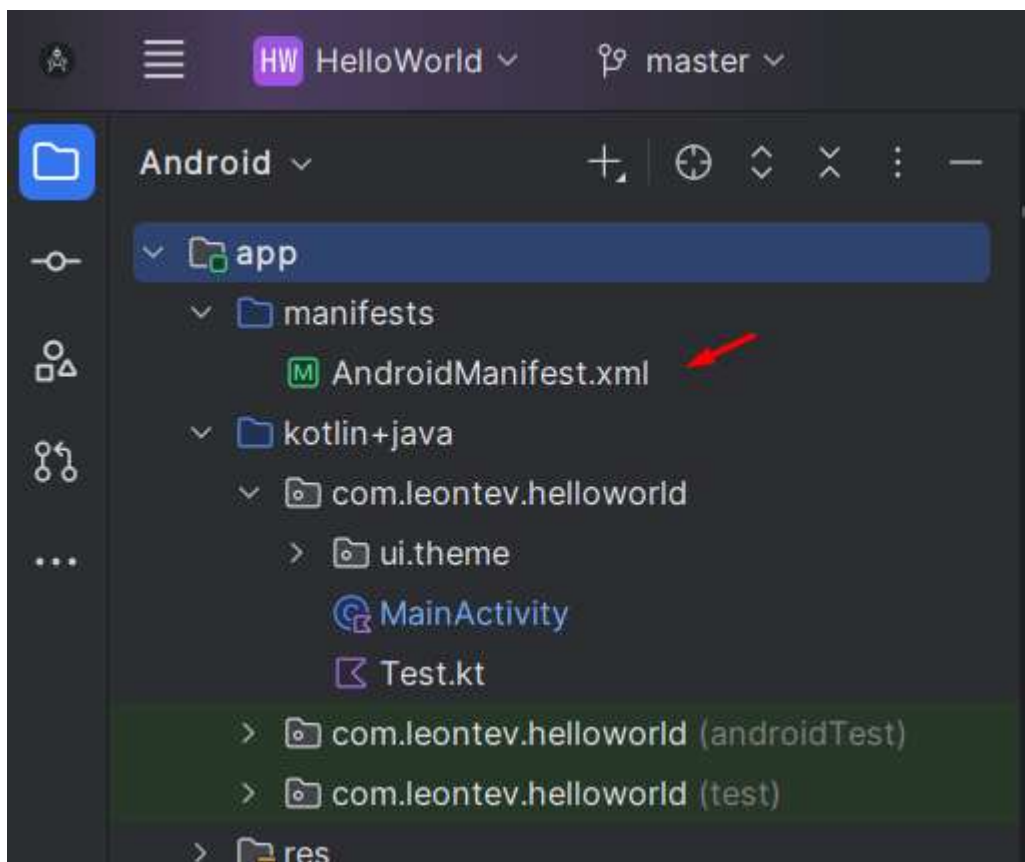
Пример: Приложение "Контакты" предоставляет доступ к своим данным другим приложениям (мессенджерам, соцсетям).



Файл AndroidManifest.xml

Это паспорт вашего приложения.

Он обязательно находится в корне проекта.



Что в нем объявляется?

Все компоненты приложения
(все Activity, Service, etc.).

Разрешения (Permissions),
которые запрашивает
приложение (доступ к
интернету, камере,
контактам).

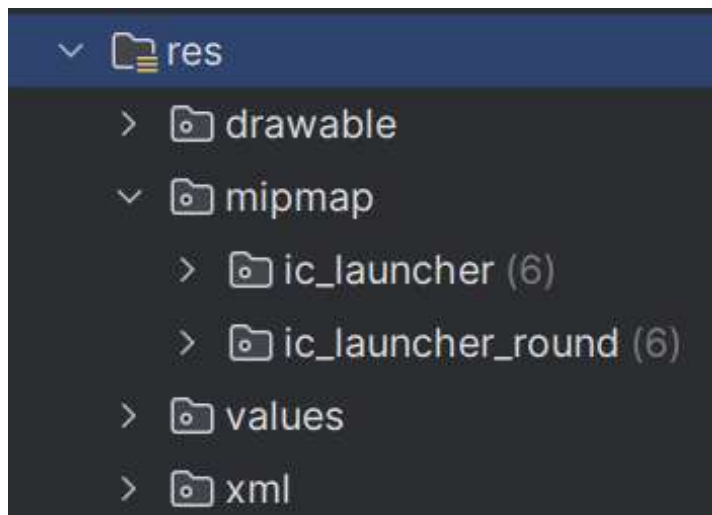
Минимальная и целевая
версия Android API.

Иконка приложения и
название.

Главная Activity, которая
запускается первой.

```
MainActivity.kt  AndroidManifest.xml x
2      <manifest xmlns:android="http://schemas.android.com/apk/res/android"
4
5          <application
6              android:allowBackup="true"
7              android:dataExtractionRules="@xml/data_extraction_rules"
8              android:fullBackupContent="@xml/backup_rules"
9              android:icon="@mipmap/ic_launcher"
10             android:label="HelloWorld"
11             android:roundIcon="@mipmap/ic_launcher_round"
12             android:supportsRtl="true"
13             android:theme="@style/Theme.HelloWorld">
14             <activity
15                 android:name=".MainActivity"
16                 android:exported="true"
17                 android:label="HelloWorld"
18                 android:theme="@style/Theme.HelloWorld">
19                 <intent-filter>
20                     <action android:name="android.intent.action.MAIN" />
21
22                     <category android:name="android.intent.category.LAUNCHER" />
23                 </intent-filter>
24             </activity>
25         </application>
26
27     </manifest>
```

Из чего еще состоит приложение?



Некоторые подпапки **Res-папки**:

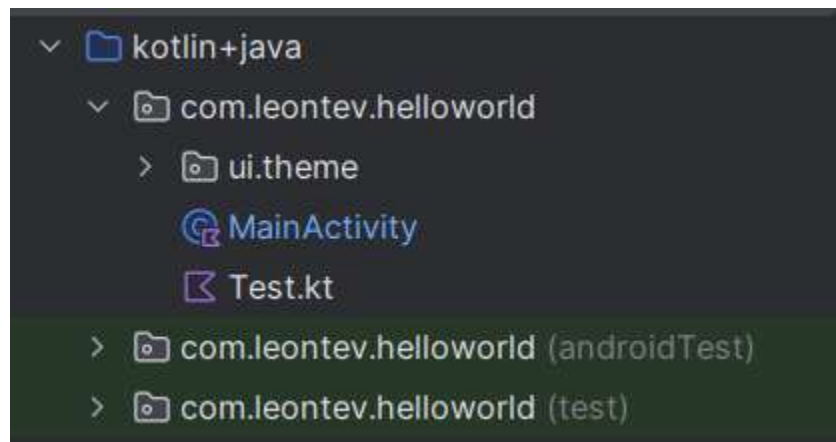
Drawable (res/drawable). Содержит изображения, XML-файлы, определяющие формы, градиенты или другие графические ресурсы.

Values (res/values). XML-файлы, содержащие простые значения, такие как строки, целые числа и цвета.

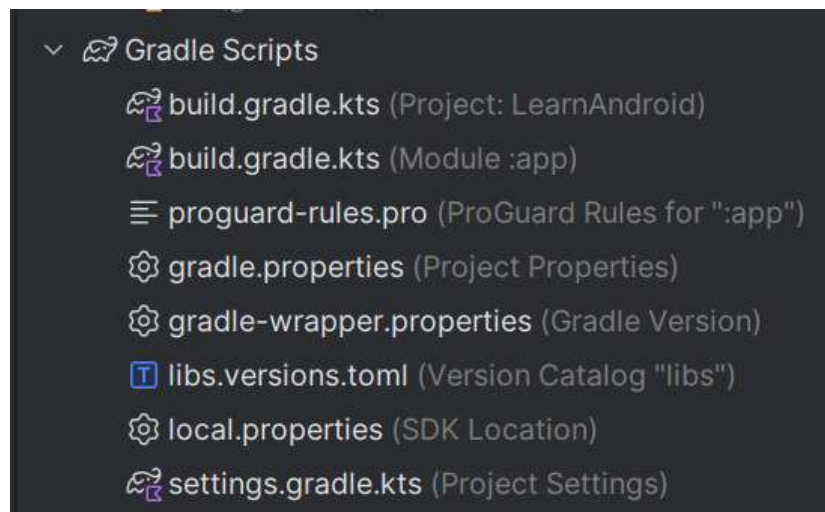
Mipmap (res/mipmap). Используется для иконок запуска приложения разных размеров для различных плотностей экрана.

Xml (res/xml). Произвольные XML-файлы, которые можно прочитать во время выполнения, вызвав `Resources.getXML()`.

Из чего еще состоит приложение?



Папка в которой мы будем писать код — **kotlin+java**. Здесь находятся все файлы на языке **Kotlin** или **Java**.



Папка **Gradle Scripts** содержит конфигурационные файлы, которые управляют сборкой проекта, зависимостями, версиями SDK и плагинами.

Резюме:

- Приложение — это набор компонентов (Activity, Service, etc.), а не единая программа.
- Манифест (AndroidManifest.xml) — это важнейший файл, который рассказывает системе о приложении.
- Приложение четко разделено на логику (код) и внешнее представление (ресурсы).
- Процесс сборки преобразует ваш человекочитаемый код в оптимизированный APK-файл, готовый к установке.
- Система Android — менеджер: Она сама решает, когда создать, запустить, приостановить или уничтожить компоненты вашего приложения в зависимости от действий пользователя и состояния системы.

История Android — от стартапа до глобального доминирования



Как зеленый робот покори́л мир...

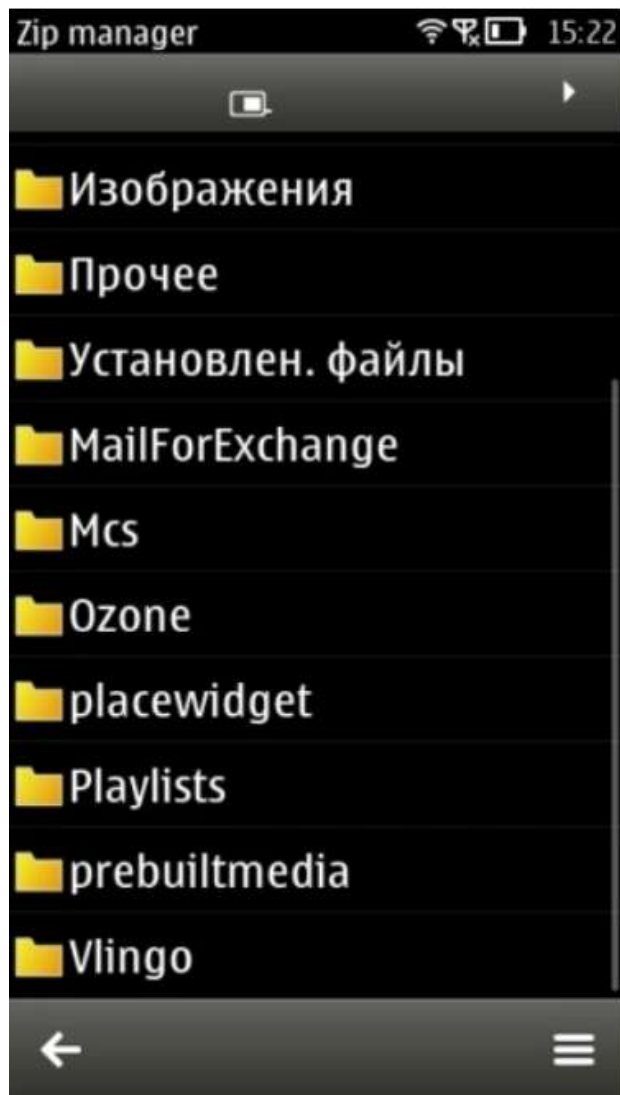
Эпоха до Android

- Доминация «закрытых» ОС: Nokia (Symbian), BlackBerry (RIM), Windows Mobile.

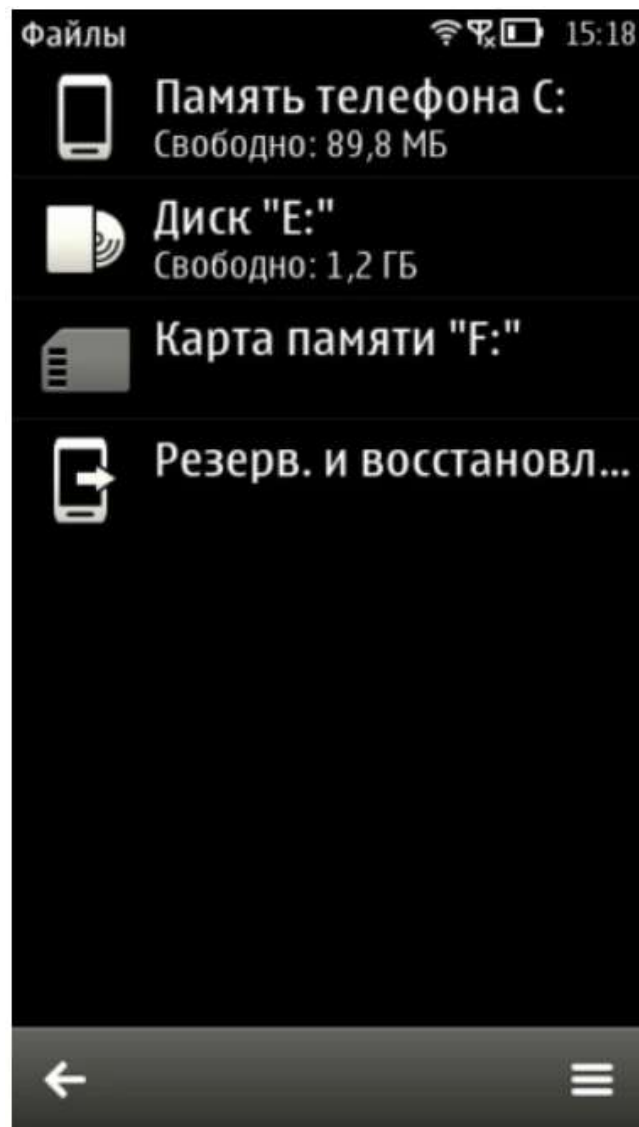




Отличие Symbian от iOS и Android в том, что он не управляет памятью сам - а даёт эту prerogative пользователю. Для Symbian в смартфоне избыточное количество ОЗУ - 512мб, что позволяет не выгружать приложения из памяти. Если памяти будет не хватать - смартфон "прибьёт" самое жрущее приложение, при этом не создавая иллюзию того, что оно открыто.



встроенный архиватор



файловый менеджер



меню

Эпоха до Android

- Проблема фрагментации: У каждого производителя была своя ОС, что усложняло жизнь разработчикам.



Эпоха до Android

- Появление iPhone (2007): Переопределил понятие «смартфон» с помощью мультитач-экрана, простого UI и App Store. Это был вызов всей индустрии.
- Назревавшая потребность: Нужна была универсальная, открытая и бесплатная мобильная ОС, которая могла бы объединить разных производителей против Apple.



Зарождение (2003-2005)

- Октябрь 2003: В Пало-Альто (Калифорния) Энди Рубин, Рич Майнер, Ник Сирс и Крис Уайт основывают компанию Android Inc..



Энди Рубин

- Первоначальная идея: Разработать продвинутую операционную систему для цифровых камер. Идея быстро сменилась на создание ОС для мобильных телефонов.
- Философия Энди Рубина: Открытость, возможность бесплатной кастомизации для производителей.

Приобретение и стратегический ход (2005-2007)

- 2005: Google приобретает Android Inc. примерно за \$50 млн. Энди Рубин и его команда присоединяются к Google.
- Стратегия: Google понимает, что будущее — за мобильным интернетом. Чтобы их сервисы (Поиск, Карты, Gmail) оставались на вершине, им нужно контролировать платформу, на которой они работают.
- Создание Open Handset Alliance (ОНА): 5 ноября 2007 года Google объявляет о создании альянса из 34 компаний (производители, операторы, чипмейкеры). В тот же день представлена Android 1.0 Beta.



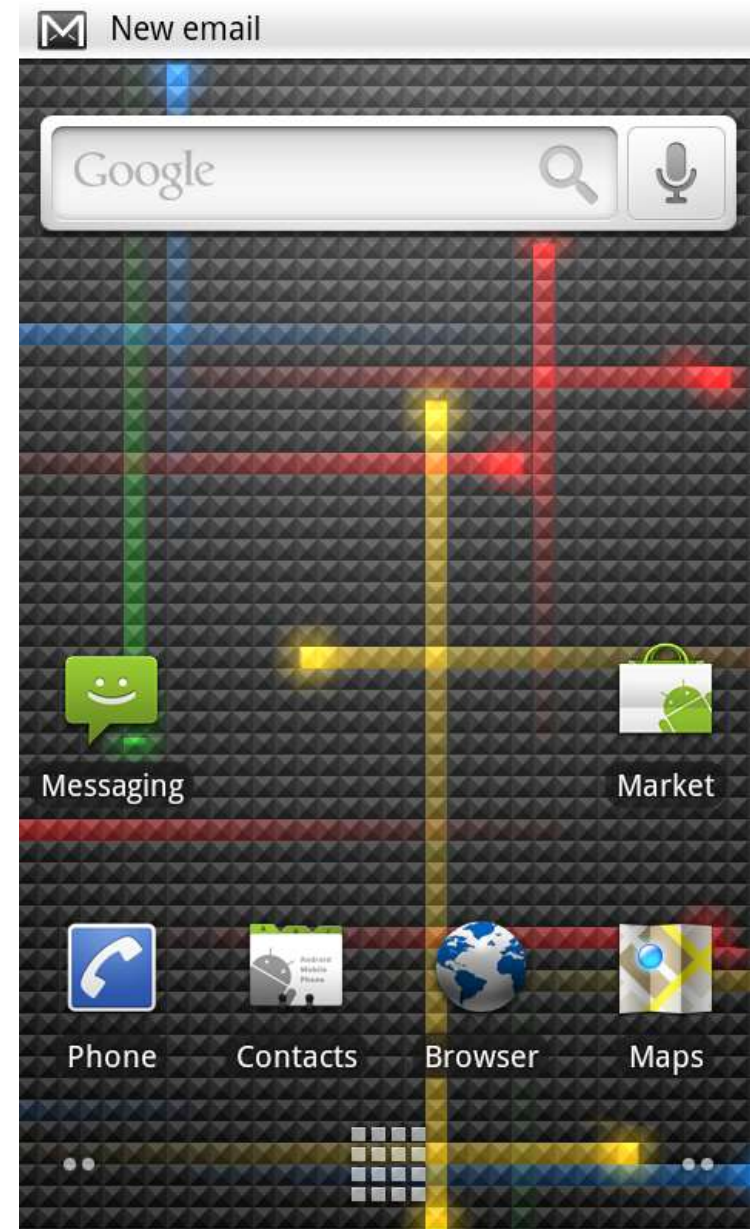
Первый девайс и ранние версии (2008-2009)

- 23 сентября 2008: Выходит первый коммерческий телефон на Android — HTC Dream (T-Mobile G1).
- Физическая QWERTY-клавиатура, трекбол, сенсорный экран.
- Уже были: браузер, Gmail, YouTube, Google Maps и Android Market (всего 50 приложений на старте).
- Версии 1.x (Apple Pie, Banana Bread...): Ранние, «сырые» версии. Доказывали жизнеспособность концепции.



Формирование идентичности: Эпоха десертов (2009-2013)

- Android 2.0 Eclair (2009): Настоящий прорыв. Поддержка нескольких аккаунтов, навигация в Google Maps, голосовые команды.



Формирование идентичности: Эпоха десертов (2009-2013)

- Линейка Nexus: Google начинает сотрудничество с производителями для создания эталонных устройств (Nexus One от HTC, Nexus S от Samsung). Цель — показать «идеальный» Android.



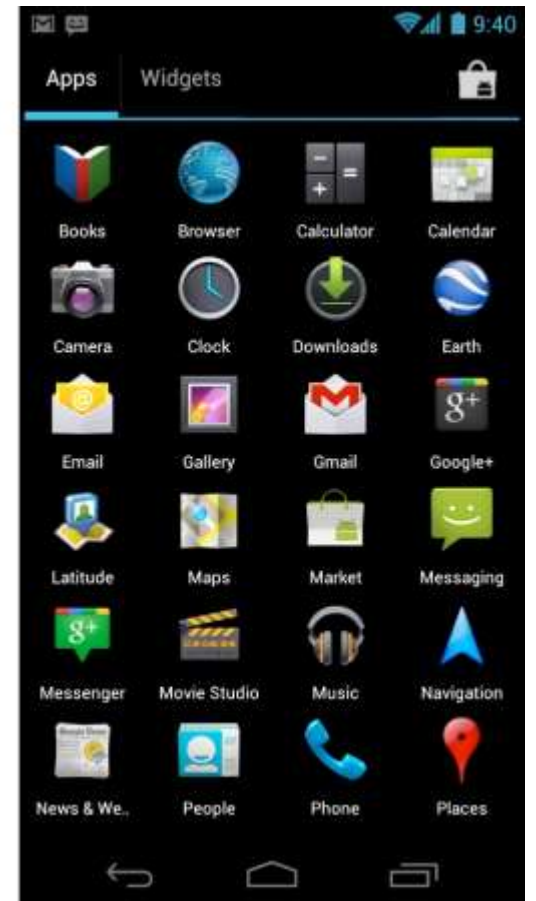
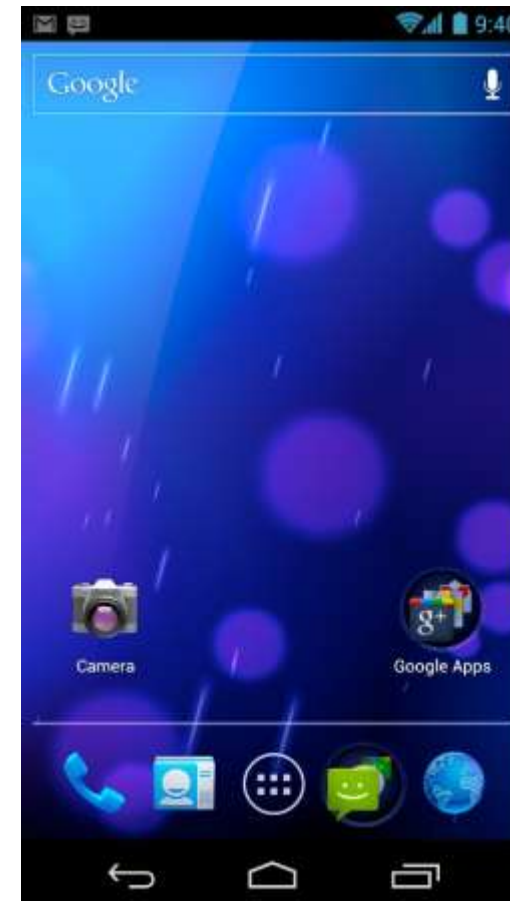
Формирование идентичности: Эпоха десертов (2009-2013)

- Android 3.0 Honeycomb (2011): Провальная версия, сделанная в спешке для планшетов. Неудачный интерфейс, но заложила основы дизайна для будущего.



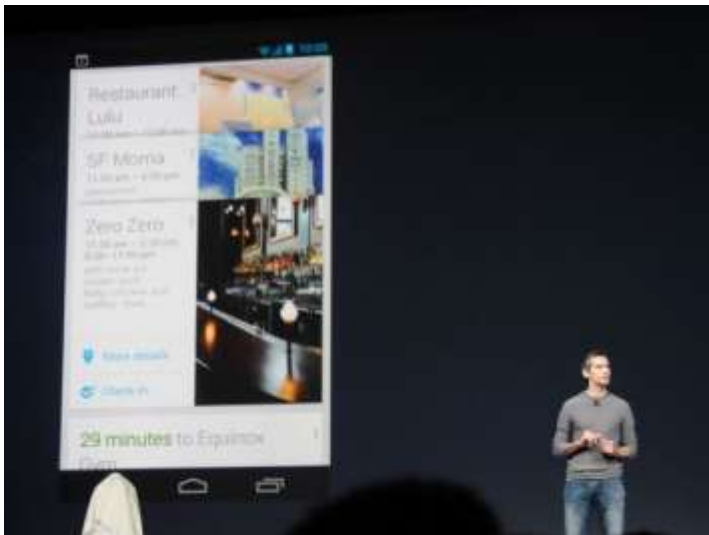
Android 4.x Ice Cream Sandwich & Jelly Bean

- Android 4.0 Ice Cream Sandwich (2011): Объединила лучшие черты Phone и Tablet UI. Новый, современный дизайн Holo.



Android 4.x Ice Cream Sandwich & Jelly Bean

- Android 4.1 Jelly Bean (2012): Ключевое обновление!
- Project Butter: Сделала интерфейс «маслянисто» плавным (60 FPS).
- Google Now: Интеллектуальный ассистент.
- Уведомления стали интерактивными.
- Ключевой тренд: Android догоняет и даже превосходит iOS по функциональности, но все еще отстает по плавности и дизайну.

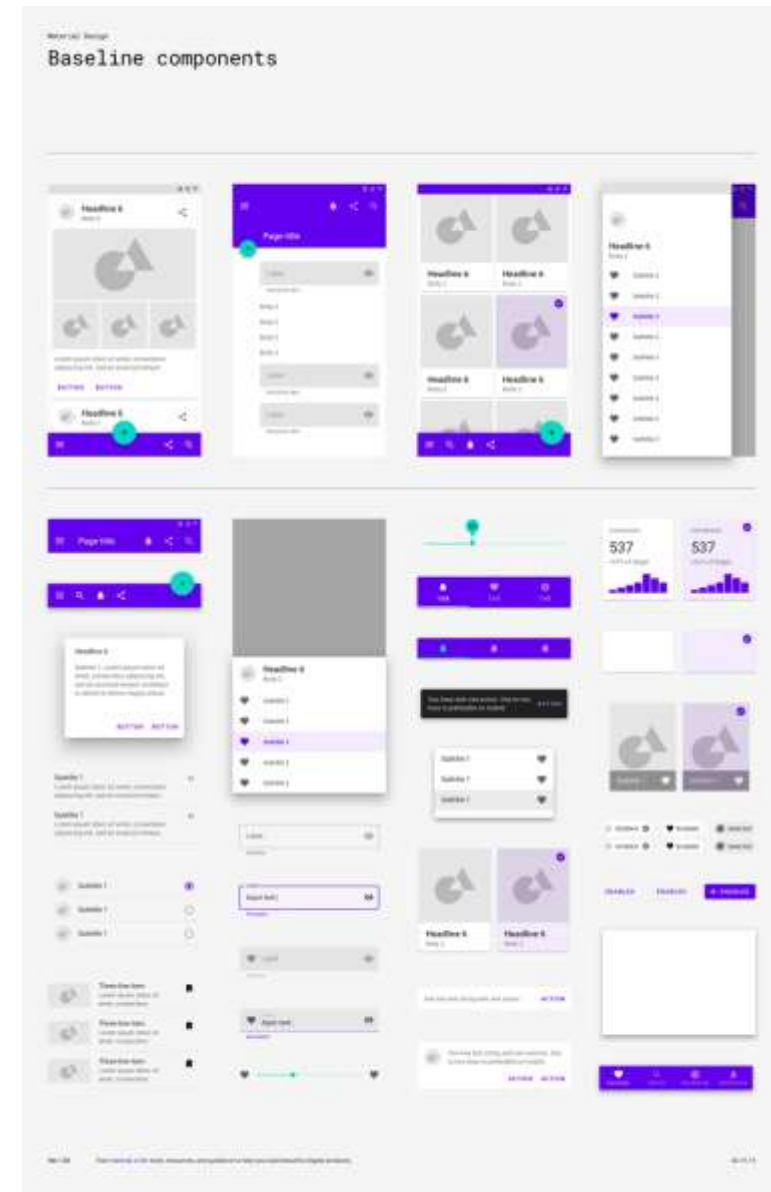


Зрелость и доминанция (2014-2019)

- 2014: Android 5.0 Lollipop — революция в дизайне.
- Material Design: Единый, понятный, осязаемый язык дизайна от Google. Цель — согласованность на всех устройствах и платформах.

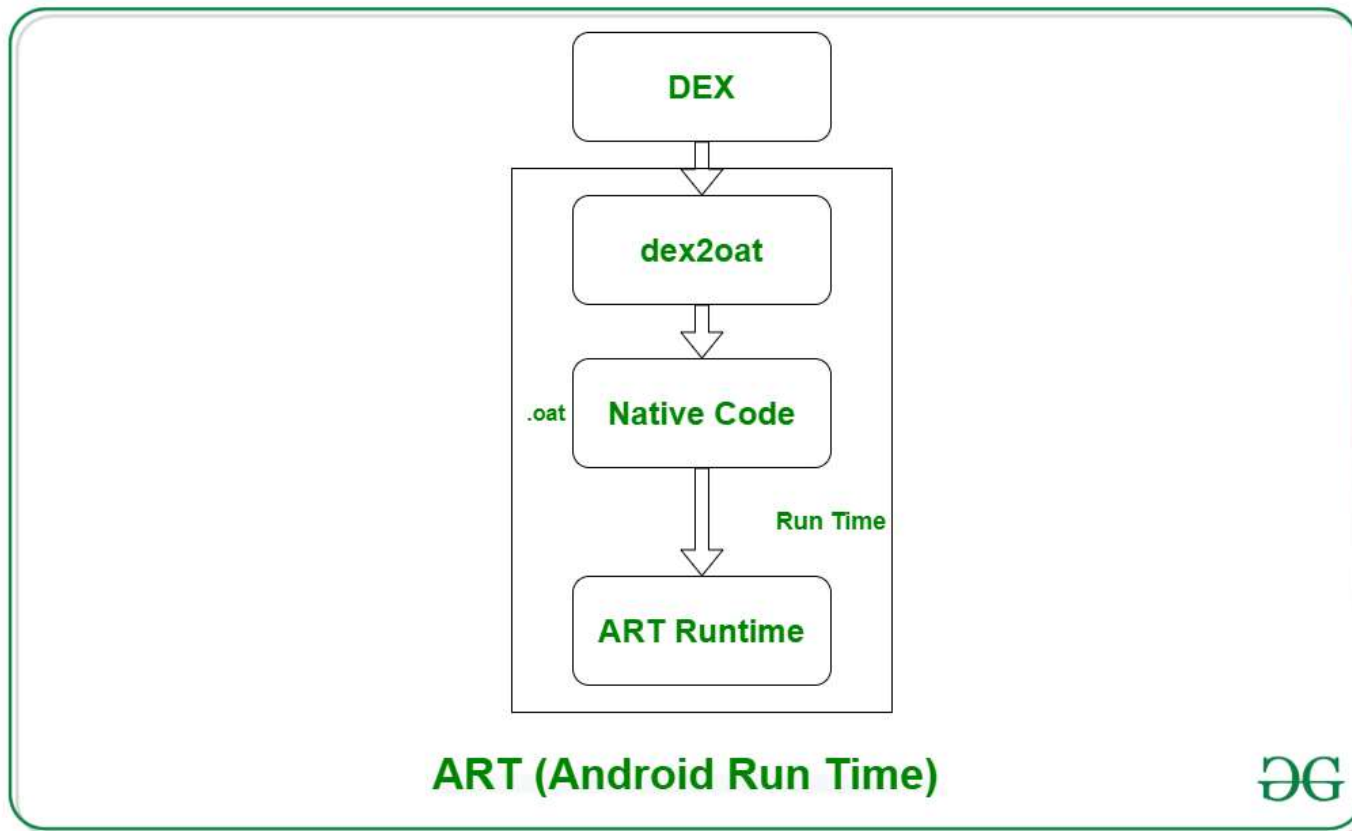


Android 5.0, Lollipop



Зрелость и доминация (2014-2019)

- Android Runtime (ART) заменил Dalvik, что повысило производительность.



Зрелость и доминация (2014-2019)

- Рост экосистемы: Android TV, Wear OS, Android Auto.



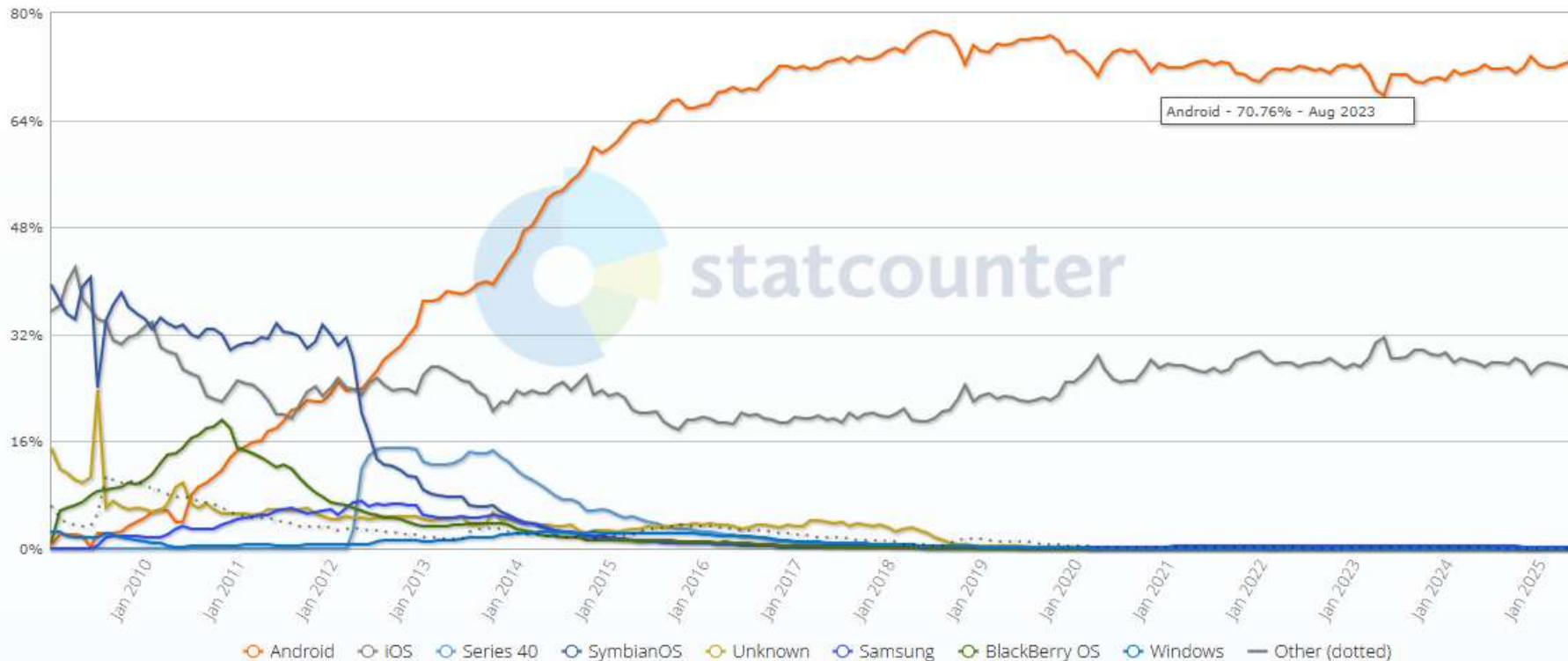
Зрелость и доминация (2014-2019)

- Статистика: Android становится самой популярной ОС в мире по количеству устройств.

Mobile Operating System Market Share Worldwide

Jan 2009 - July 2025

Edit Chart Data



Борьба с фрагментацией и новые вызовы

- Проблема: Из-за открытости тысячи устройств работают на старых версиях ОС. Это головная боль для разработчиков и угроза безопасности.

Решение от Google:

- Вынесение ключевых компонентов в отдельные сервисы (Google Play Services), которые обновляются через магазин, минуя производителей.
- Project Treble (Android 8.0 Oreo): Архитектурное изменение, упрощающее производителям выпуск обновлений.
- Появление конкурента: Huawei представляет собственную ОС HarmonyOS после санкций США.

Современная эра (2020 — настоящее время)

- Отказ от сладостей: С Android 10 (2019) Google отказывается от названий десертов в пользу номеров версий — более глобальный и профессиональный подход.

Android Oreo
Android Pie
Android 10
Android 11
Android 12
Android 12L
Android 13
Android 14
Android 15
Android 16

Современная эра (2020 — настоящее время)

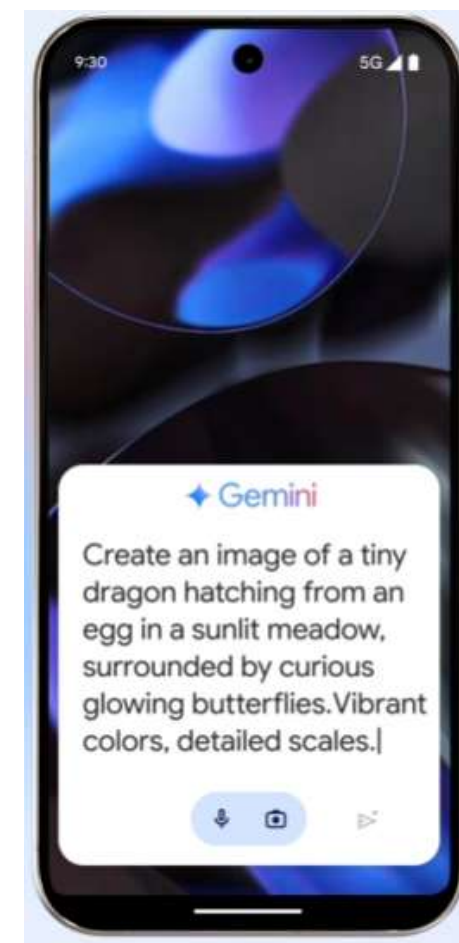
- Фокус на конфиденциальности: Усиленный контроль разрешений, индикатор доступа к камере/микрофону.
- Большие экраны: Android активно адаптируется для складных телефонов, планшетов, хромбуков.



Huawei MateBook Fold – первый складной ноутбук 2025 года

Современная эра (2020 — настоящее время)

- Тенденция: Интеграция ИИ, улучшение персонализации, плавность и стабильность в приоритете над добавлением горы новых функций.



Ключевые факторы успеха Android

- Открытая модель (Open Source): Бесплатный код привлёк производителей.
- Сильная экосистема Google: Интеграция с популярными сервисами.
- Разнообразие выбора: Устройства на любой вкус и кошелек.
- Свобода для пользователей и разработчиков: Кастомизация, альтернативные магазины приложений.
- Стратегический альянс (ОНА): Объединение лидеров индустрии.

Jetpack Compose вместо XML

Раньше UI создавался в XML, и это было сложно.
Теперь: UI пишется на Kotlin, как обычные функции →
декларативный подход.
Это проще, современнее и быстрее.

```
<TextView  
    android:id="@+id/helloText"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Hello World!" />
```

XML

```
2 Usages  
33     @Composable  
34     fun Greeting(name: String, modifier: Modifier = Modifier) {  
35         Text(  
36             text = "Hello $name!",  
37             modifier = modifier  
38         )  
39     }
```

Text(text = "Hello World!")

Jetpack Compose