

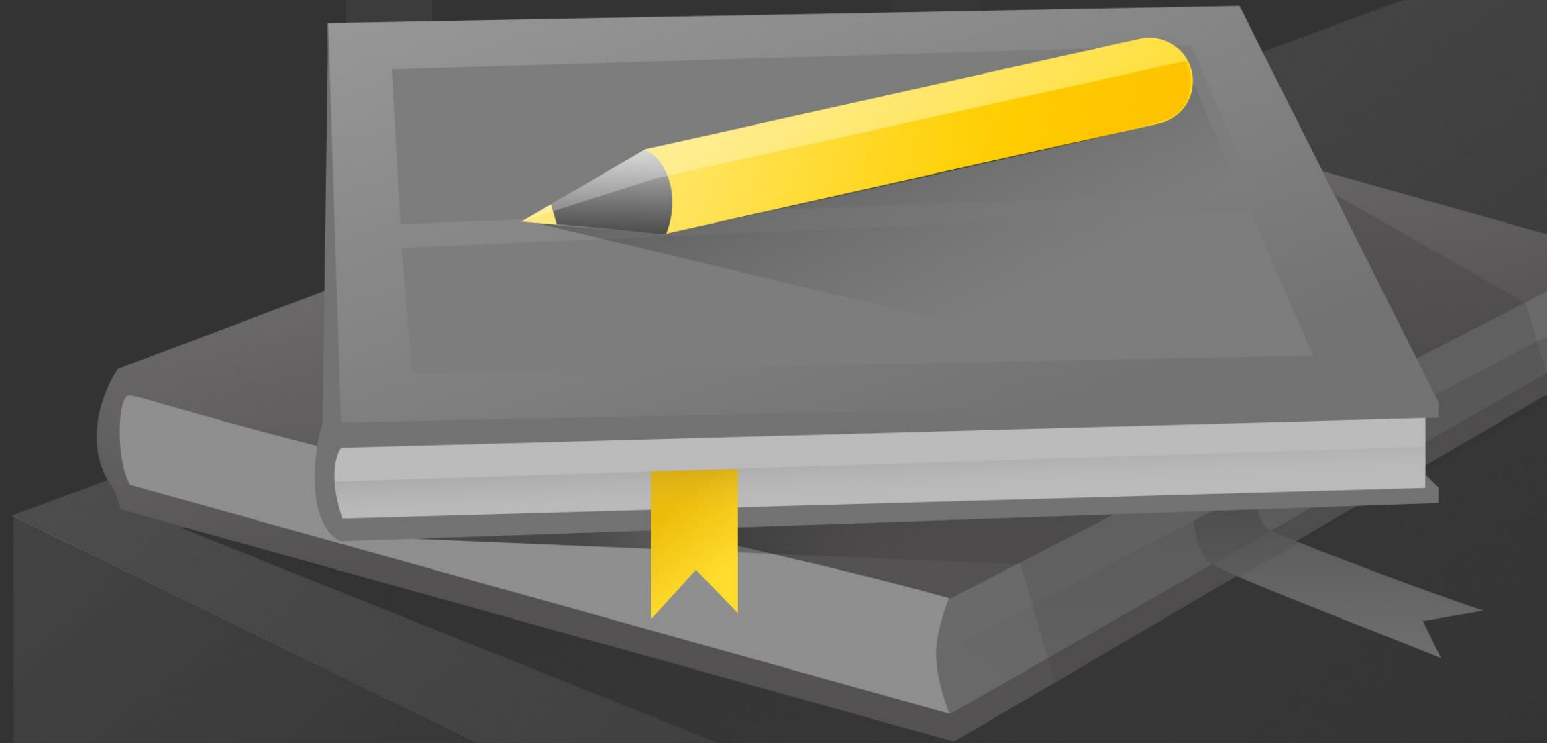
Android разработка

@лкл'зима 2025

Игорь Лизунов

О чем расскажу?

- Что такое Android?
- Kotlin
- Основные фичи Kotlin'а
- Android Studio
- Генератор мемов



Игорь Лизунов

Senior Android Developer



- Лицей №2'21 - математический класс
- Пермский государственный гуманитарно-педагогический университет'24 – прикладная информатика



Мобильный разработчик

И почему именно Android разработка?

Согласно определению из ~~Википедии~~ ChatGPT –

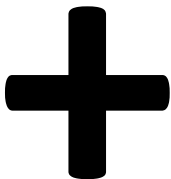
это специалист, который создает приложения для мобильных устройств, таких как смартфоны и планшеты. Эти приложения могут работать на различных операционных системах, например, iOS или Android

Валидация данных банковской карты

Необходимо реализовать логику валидации введенных данных банковской карты.

1. Валидация номера карты
 - i. Номер карты должен состоять из 16 цифр, разделенных пробелами на группы по 4 цифры
 - ii. Номер карты должен содержать только цифры и пробелы
2. Валидация срока действия карты
 - i. Срок действия карты должен состоять из 5 символов и быть в формате `mm/YY`
 - ii. Месяц должен быть в диапазоне от 1 до 12
 - iii. Год должен быть в диапазоне от 00 до 99
 - iv. Срок действия карты должен быть больше текущей даты. Считаем, что на текущий месяц карта еще действительна
3. Валидация CVV
 - i. CVV должен состоять из 3 символов
 - ii. CVV должен содержать только цифры

Если пользователь еще не полностью ввел данные в поле ввода, то сообщение об ошибке не должно отображаться, но оплата должна быть недоступна. Валидация должна происходить только после полного ввода данных в поле ввода. Например, при вводе номера карты `1234 567` сообщение об ошибке не должно отображаться, но оплата должна быть недоступна. После ввода номера карты `1234 5678 1234 5678` сообщение об ошибке не должно быть.



Экран оплаты

9:41

Оплата

Карта

.....

Дата

01/25

CVV

...

Карта

Неверно

.....

Дата

Неверно

mm/YY

CVV

...

Оплатить

1 Заголовок экрана	
Размер текста	18sp
Цвет текста	#000000
Шрифт	sans-serif-medium
Выравнивание	По центру
includeFontPadding	false

2 Заголовок поля ввода	
Оплата	
Размер текста	16sp
Цвет текста	#7A7A7A
Выравнивание	По началу
includeFontPadding	false

3 Подсказка в поле ввода	
Карта	
Размер текста	18sp
Цвет текста	#CBCACB

4 Текст в поле ввода	

==

Оплата

Карта

1234 4567 7890 1234

Дата

Неверно

01/10

CVV

123

Оплатить

Мобильный разработчик

И почему именно Android разработка?

Согласно определению из ~~Википедии~~ ChatGPT –

это специалист, который создает приложения для мобильных устройств, таких как смартфоны и планшеты. Эти приложения могут работать на различных операционных системах, например, iOS или Android

Большое количество устройств работает на Android – примерно 3,3 млрд

Взаимодействие с девайсом и пользовательским интерфейсом

Логика работы самого приложения

Android



Android Inc

Команда, состоящая из Энди Рубина, Ричарда Майнера, Ника Сирса и Криса Уайта, создала компанию Android Inc.

Покупка Google

Google покупает компанию и создает Android Open Source Project (AOSP)

Первая версия

Релиз первой версии Android

Развитие Android

Android является операционной системой не только для мобильных телефонов, но и для часов, телевизоров и других устройств.



Что такое Kotlin?

Это статически типизированный язык программирования, разработанный компанией JetBrains, который совмещает объектно-ориентированные и функциональные возможности, и предназначен для работы на платформе JVM, а также совместим с Java

Kotlin назван в честь острова в Финском заливе

Язык общего назначения

Kotlin является высокоуровневым языком программирования общего назначения, подходящим для широкого спектра задач

Совместимость с Java

Kotlin полностью совместим с Java, позволяя использовать код обоих языков в одном проекте

Универсальность подходов

Kotlin объединяет элементы объектно-ориентированного и функционального программирования

Лаконичность синтаксиса

Kotlin предлагает более краткий и выразительный синтаксис, что позволяет писать меньше кода по сравнению с Java при сохранении его читабельности

Статическая типизация

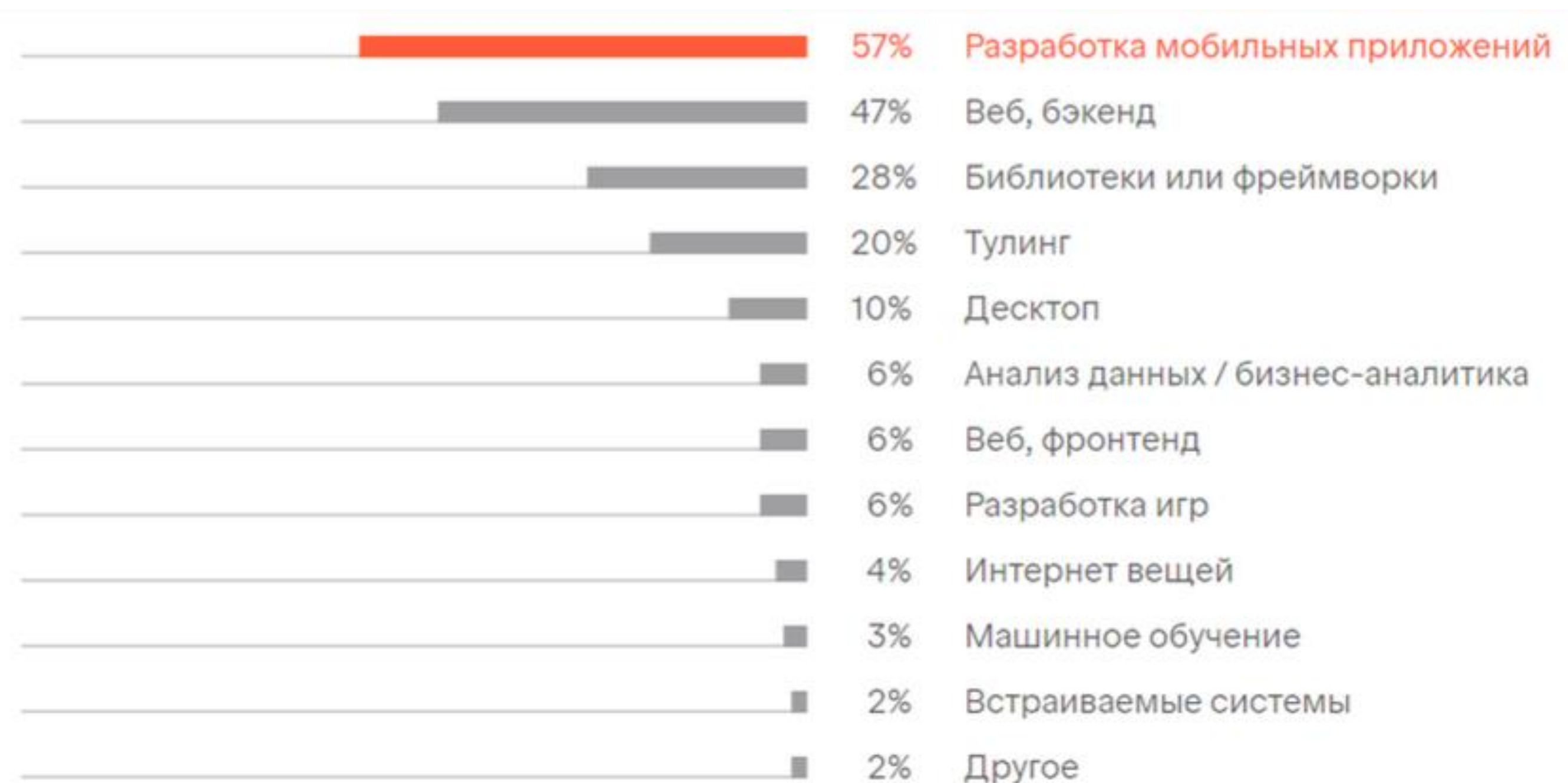
Kotlin обладает статической типизацией, обеспечивая проверку типов на этапе компиляции

Кросс-платформенность

Kotlin можно использовать для разработки под различные платформы: Android, JVM, веб и др.

Где используется Kotlin?

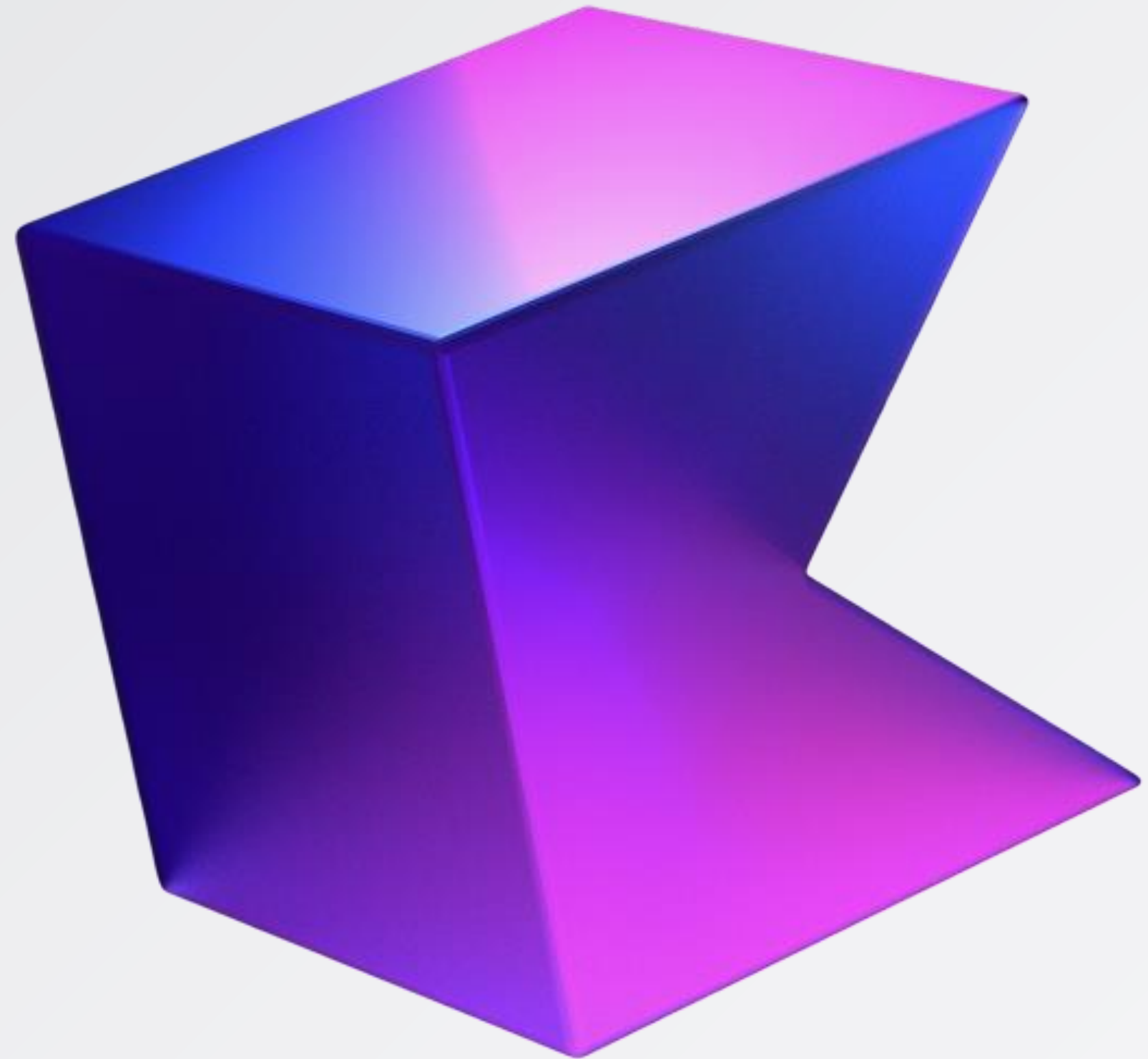
Kotlin используется для создания мобильных приложений, серверных решений и веб-приложений благодаря своей универсальности и совместимости с различными платформами. Он подходит для разработки как клиентских, так и серверных частей ПО

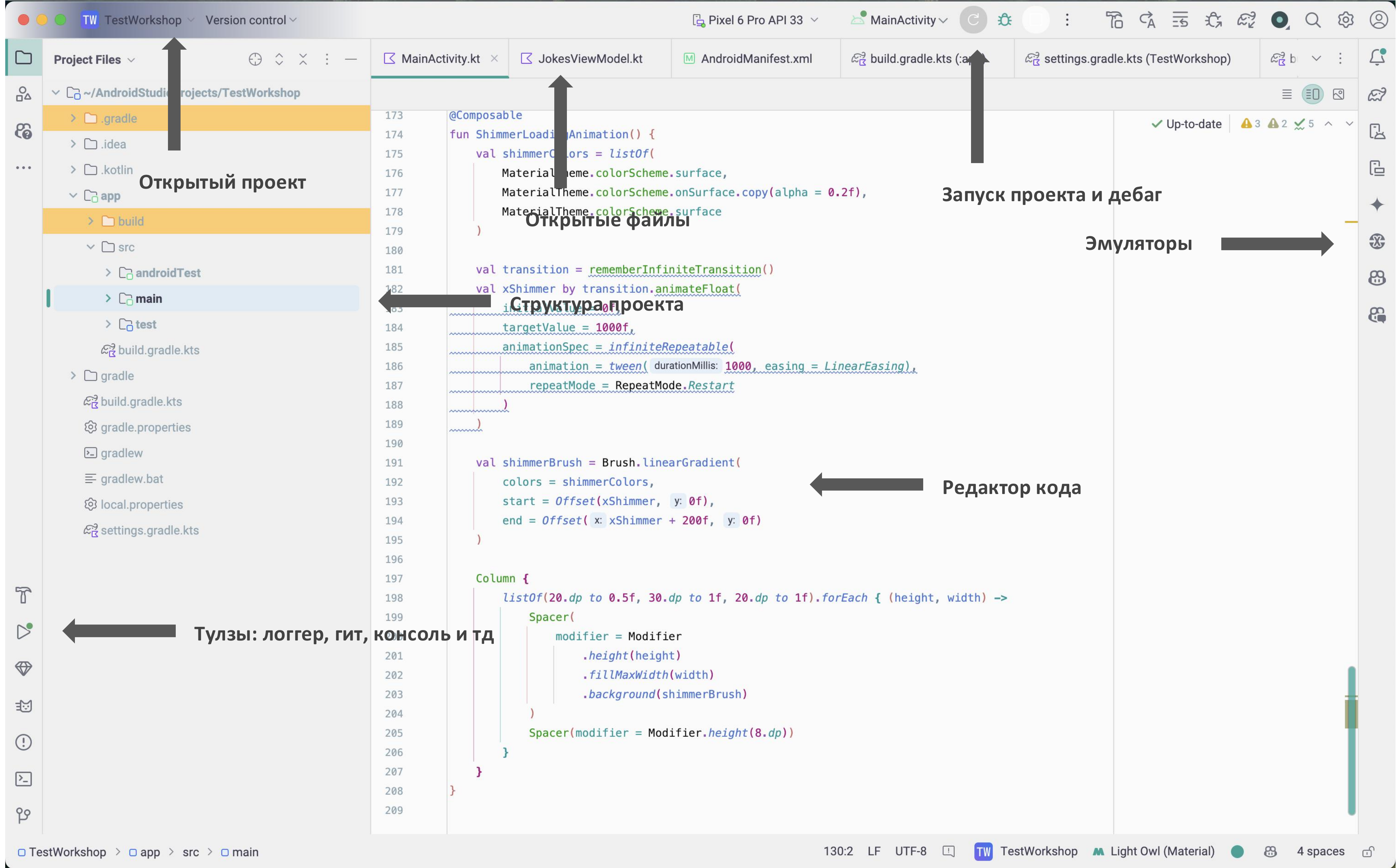


Мобильная разработка для Android стала Kotlin-first

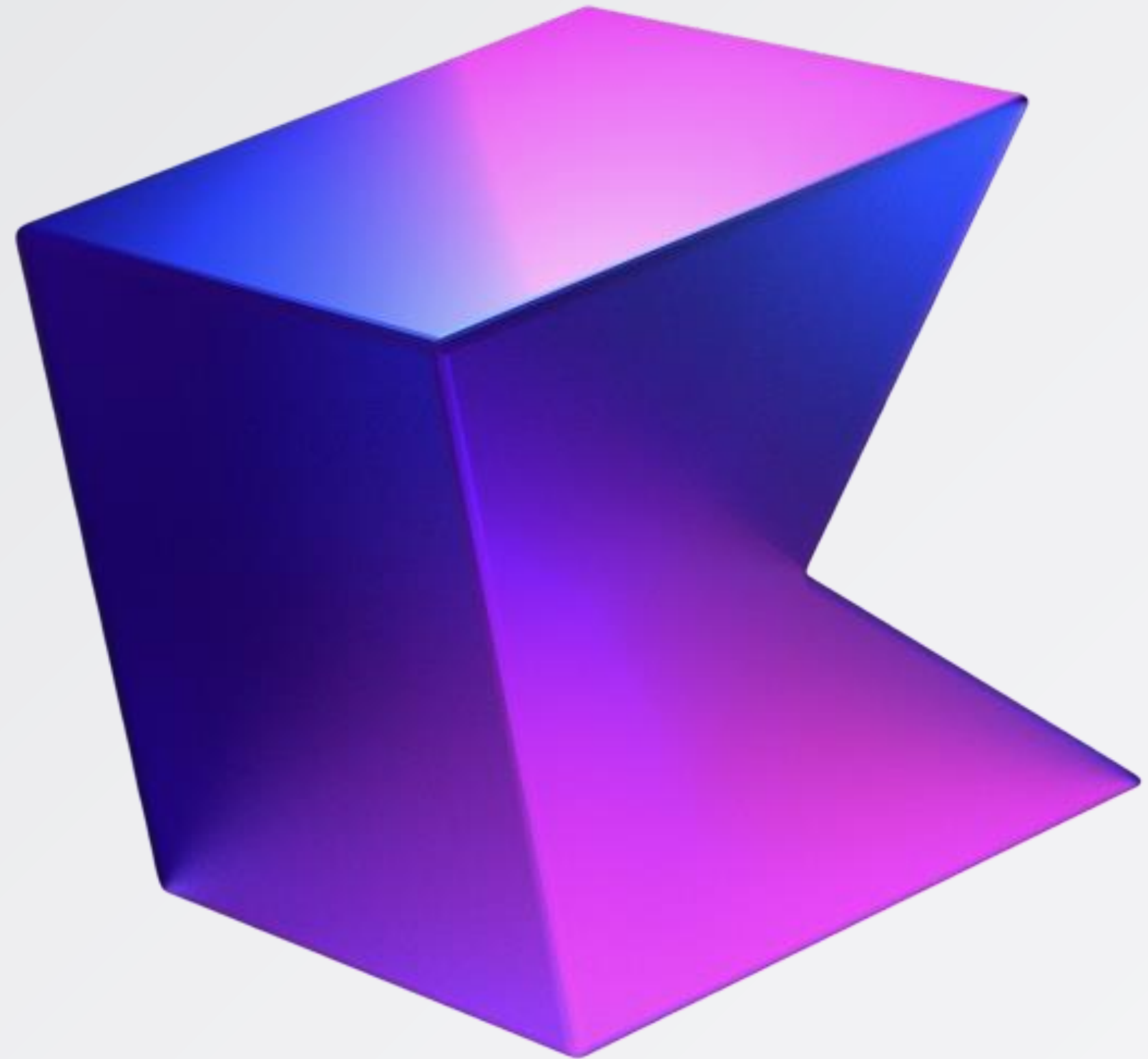
Google на Google I/O 2019 объявило Kotlin основным языком для Android

Android Studio





Gradle



Gradle

01

Projects

Проект представляет собой программный продукт, например приложение или библиотеку. Может включать в себя подпроекты (модули)

02

Build Scripts

Описывает порядок действий для сборки проекта

03

Dependency management

Метод объявления и разрешения зависимостей

04

Tasks

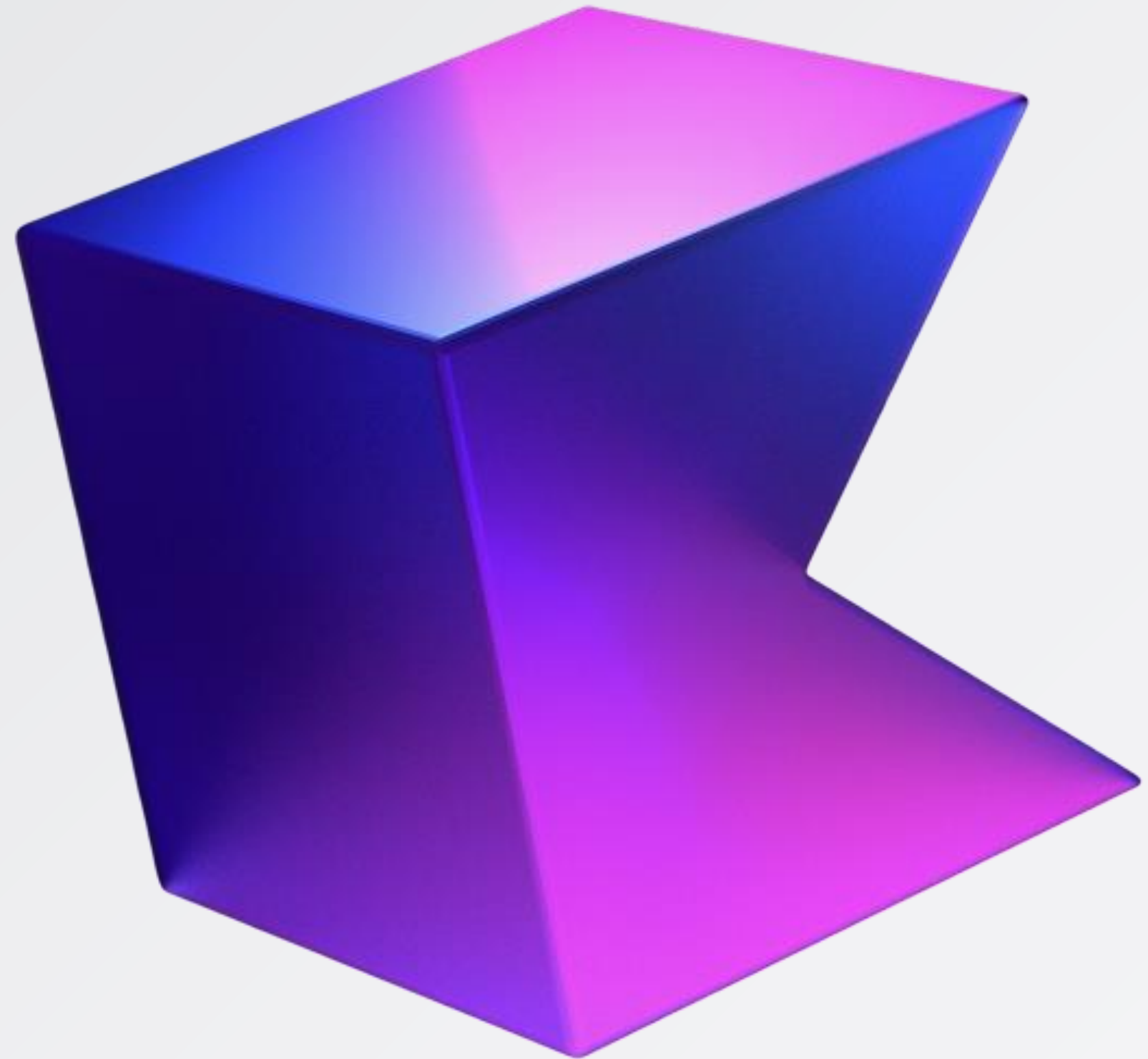
Базовая единица работы. Например компиляция или запуск тестов

05

Plugins

Расширяют возможности gradle, могут добавлять свои задачи

Kotlin



Hello, World!

```
fun main() {  
    println("Hello, World!")  
}
```

Где ; ???

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```


Hello, World!

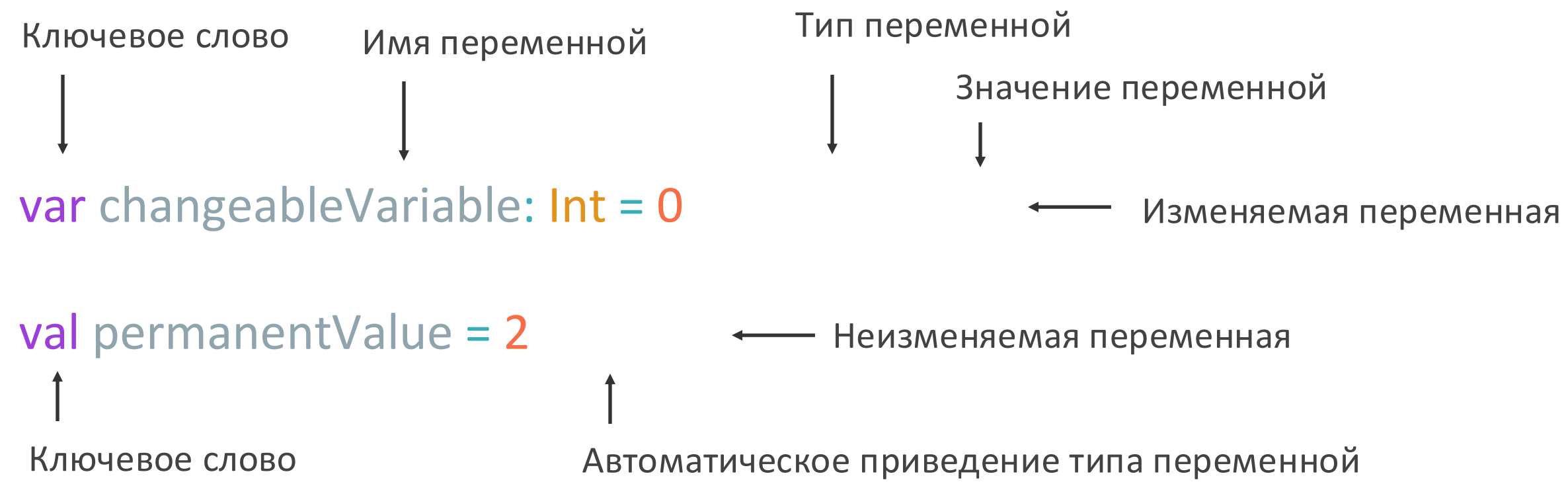
Ключевое слово для объявления функции

Имя функции

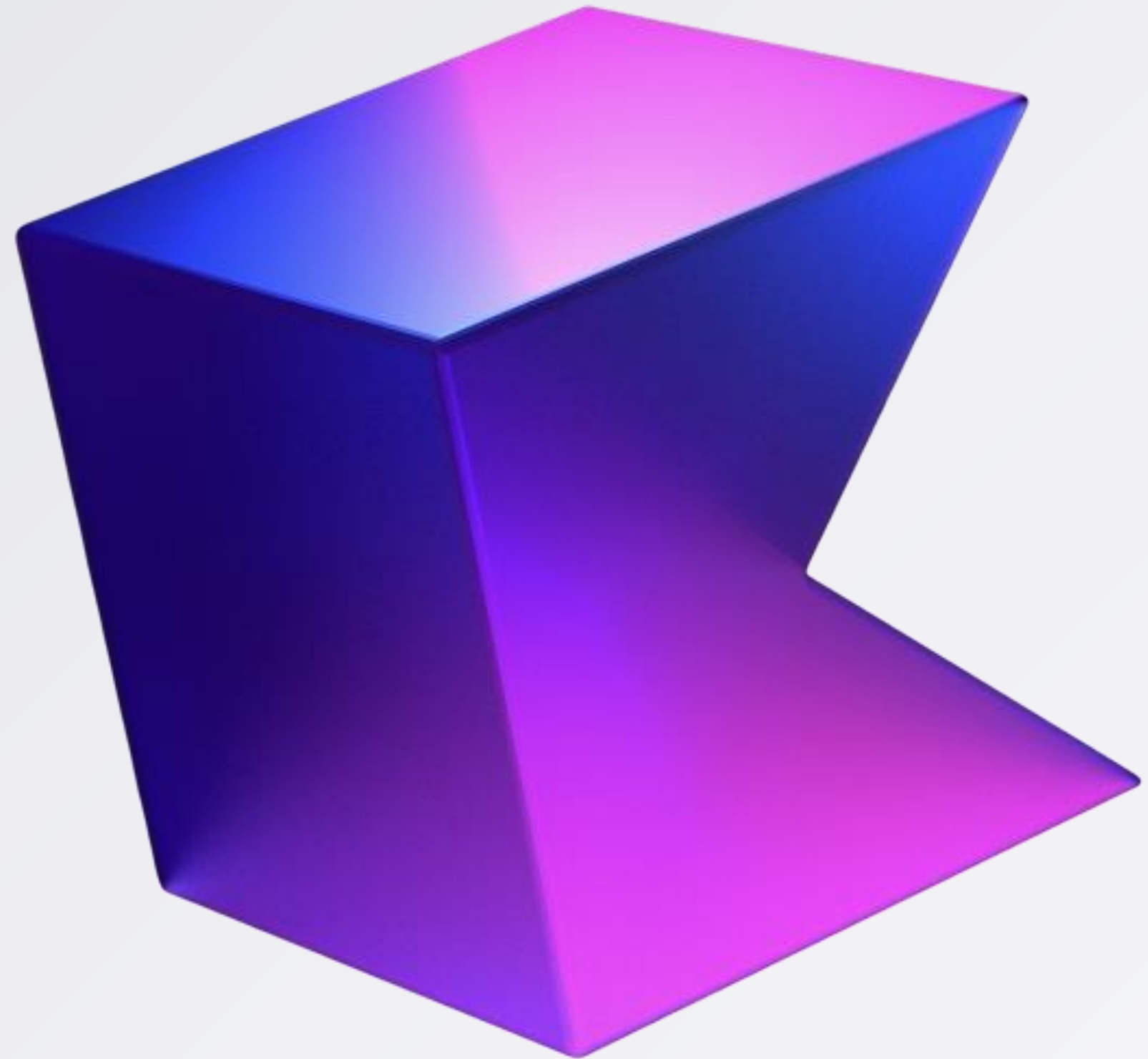
```
fun main() {  
    println("Hello, World!")  
}
```

← Тело функции

Переменные



Null



NullPointerException?

В Kotlin безопасность от null реализована через механизм разделения типов на nullable и non-nullable; это позволяет компилятору автоматически предотвращать многие ошибки, связанные с null

```
val myVariable: Int = 2
```

->

```
val myVariable: Int? = null
```

Делает переменную nullable



Null-safety call

```
val nullableVariable: Int? = null
val answer = nullableVariable?.plus(3)
```

= null



↑
Оператор безопасного обращения

```
if (nullableVariable != null) {
    answer = nullableVariable + 3
} else {
    answer = null
}
```


Elvis operator

```
val nullableVariable: Int? = null  
val answer = nullableVariable?.plus(3) ?: -1
```

= -1



Элвис оператор

```
if (nullableVariable != null) {  
    answer = nullableVariable + 3  
} else {  
    answer = -1  
}
```

Null assertion

```
val nullableVariable: Int? = getIntFromUserInput()
val answer = nullableVariable!!.plus(3)
```

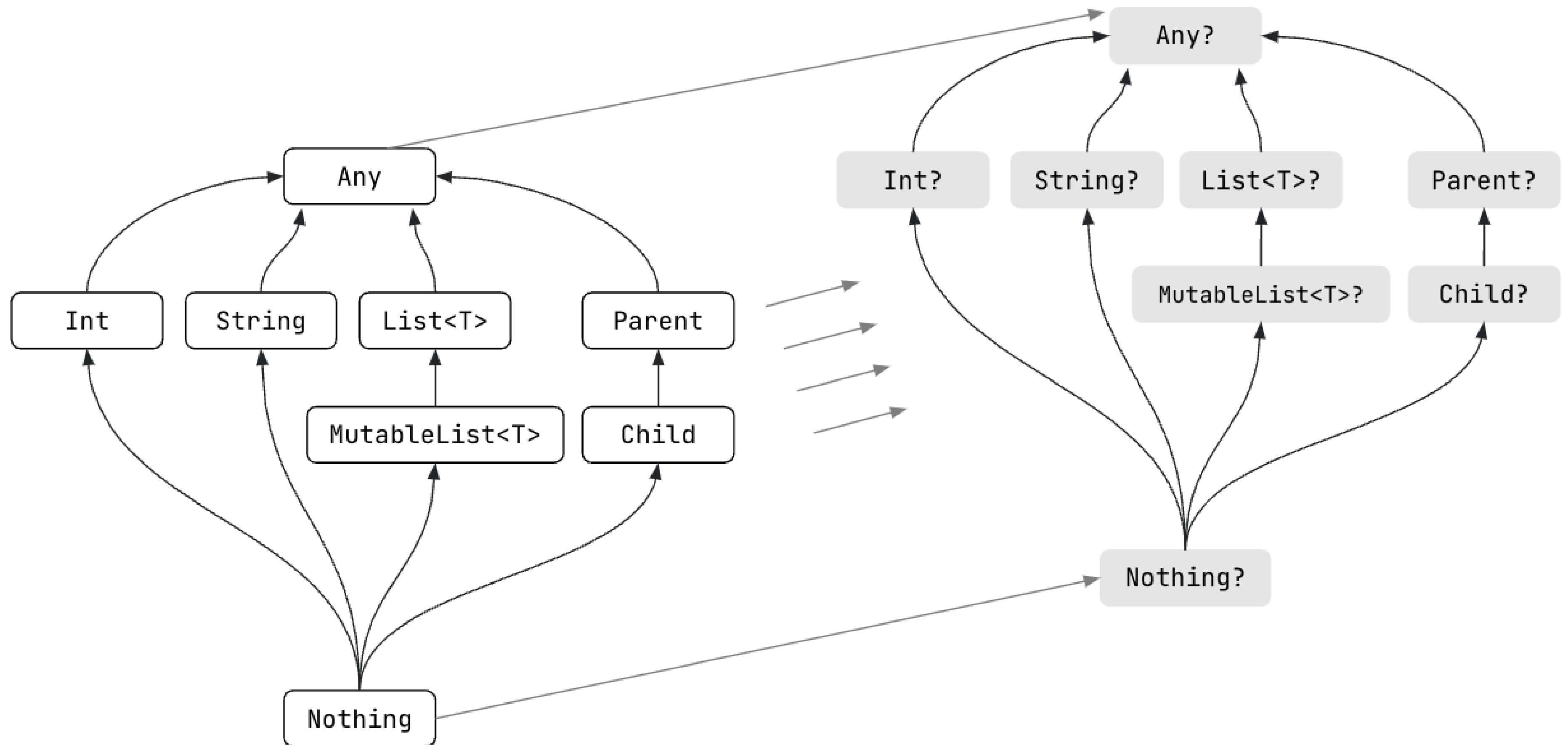
= $x + 3$
= `NullPointerException`



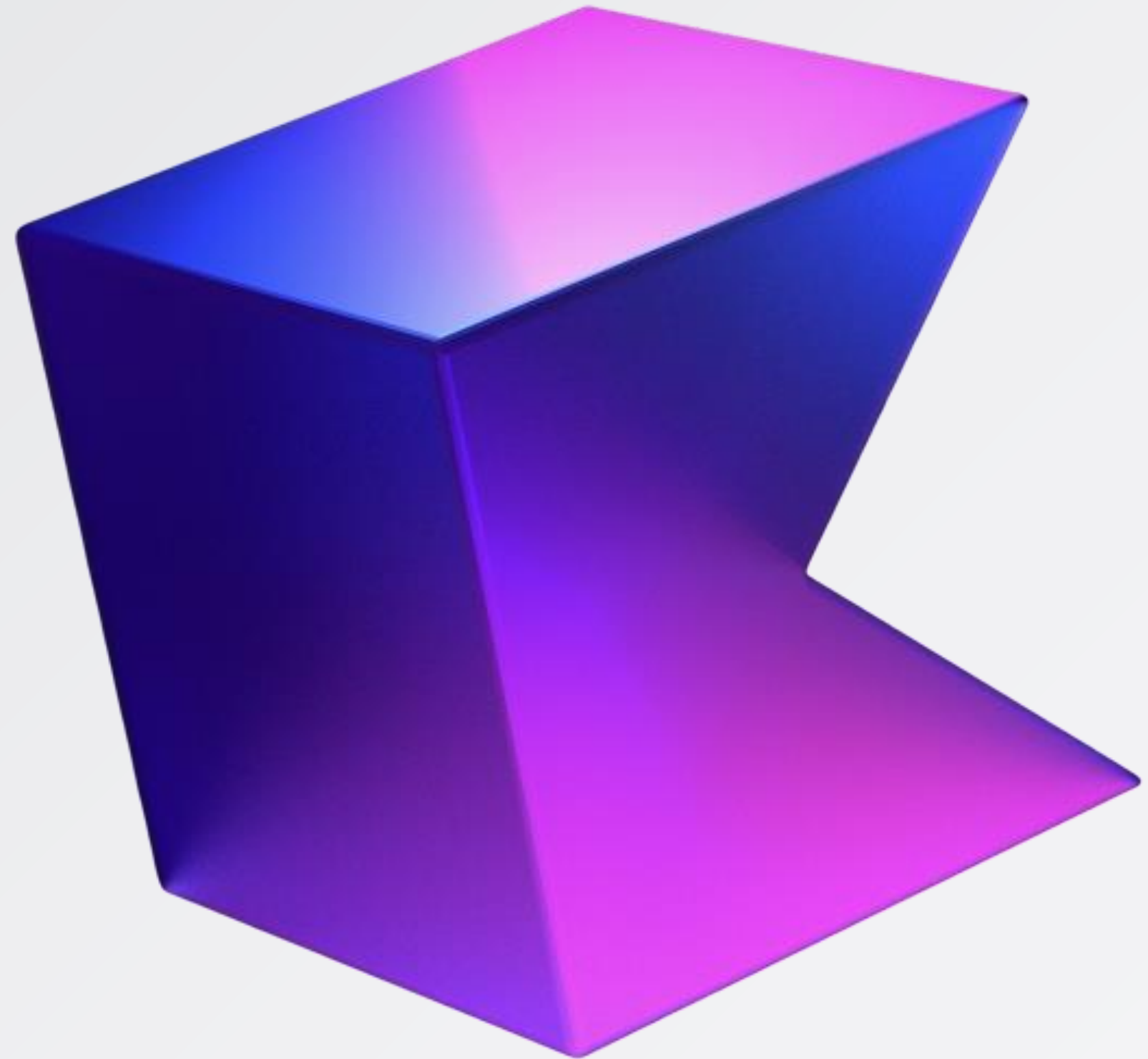
↑
Принудительный каст к not null типу*

```
if (nullableVariable == null) {
    throw NullPointerException()
}
```

Схема системы типов



if else / for



if else

```
fun main() {  
    val userInput = readln().toInt()  
    if (userInput > 0) {  
        println("Your number is greater than 0")  
    } else {  
        println("Your number is less than 0")  
    }  
}
```

Условие

Ветка по умолчанию

← Тело блока

When expression

Ключевое слово

```
fun main(args: Array<String>) {  
    val answer = when (args.size) {  
        0 -> "Arguments are empty"  
        1 -> "One argument here"  
        2 -> "Two argument here"  
        else -> "A lot of arguments"  
    }  
}
```

← Условие

← Тело блока

← Ветка по умолчанию

For

Условие для диапазона [0; 5)



```
for (i in 0 until args.size) {  
    println("Argument ${args[i]}")  
}
```

```
for (i in 0..<args.size) {  
    println("Argument ${args[i]}")  
}
```

For

Условие для итерации по индексам массива



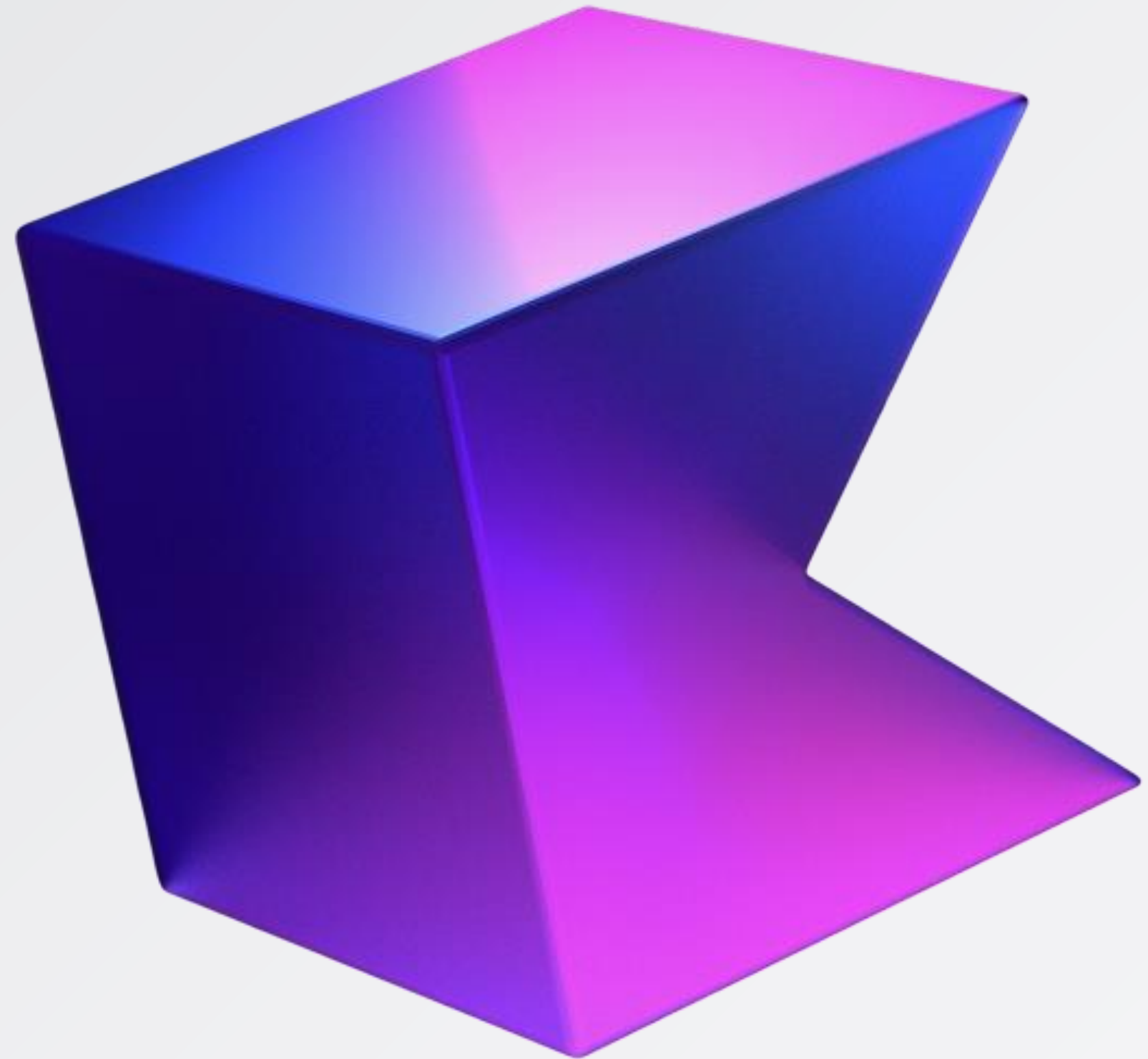
```
for (i in args.indices) {  
    println("Argument ${args[i]}")  
}
```

Условие для итерации по элементам массива



```
for (arg in args) {  
    println("Argument $arg")  
}
```

Class



Class

Ключевое слово



Имя класса



```
class MyClass
```

```
fun main() {  
    val myClassObject = MyClass()  
}
```

Создание экземпляра класса



```
class MyClass {
```

```
}
```



Тело класса

Data class

Data class в Kotlin автоматически генерирует полезные методы, такие как `equals()`, `hashCode()`, `toString()`, а также `copy()` и методы для разбора (`componentN()`), что делает его удобным выбором для хранения данных.

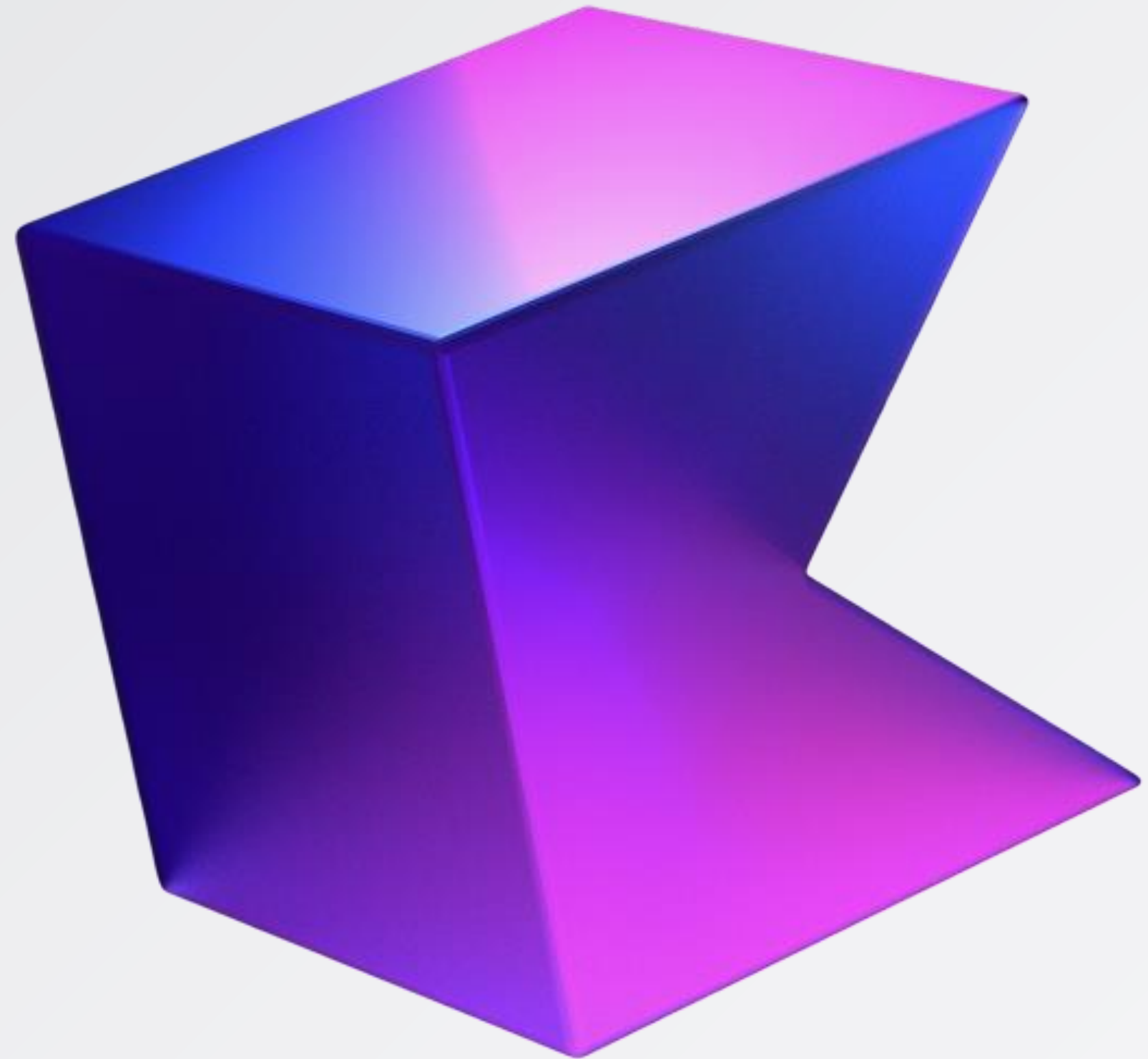
```
data class User(  
    val id: Int,  
    val name: String,  
    val email: String  
)
```

Sealed interface

Sealed interface в Kotlin позволяет определять фиксированное множество реализаций интерфейса в пределах одного файла, обеспечивая безопасность и контроль над иерархией типов при использовании полиморфизма.

```
sealed interface Shape {  
    data class Circle(val radius: Double) : Shape  
    data class Rectangle(val width: Double, val height: Double) : Shape  
}
```


runCatching



runCatching

```
fun main() {  
    runCatching {  
        addBalance(-1)  
    }  
}
```

->

```
inline fun <R> runCatching(block: () -> R): Result<R>
```



Результат выполнения

runCatching

```
fun main() {  
    runCatching {  
        addBalance(-1)  
    }.onSuccess {  
        println("Successfully added balance")  
    }.onFailure {  
        println("Failed to add balance")  
    }  
}
```

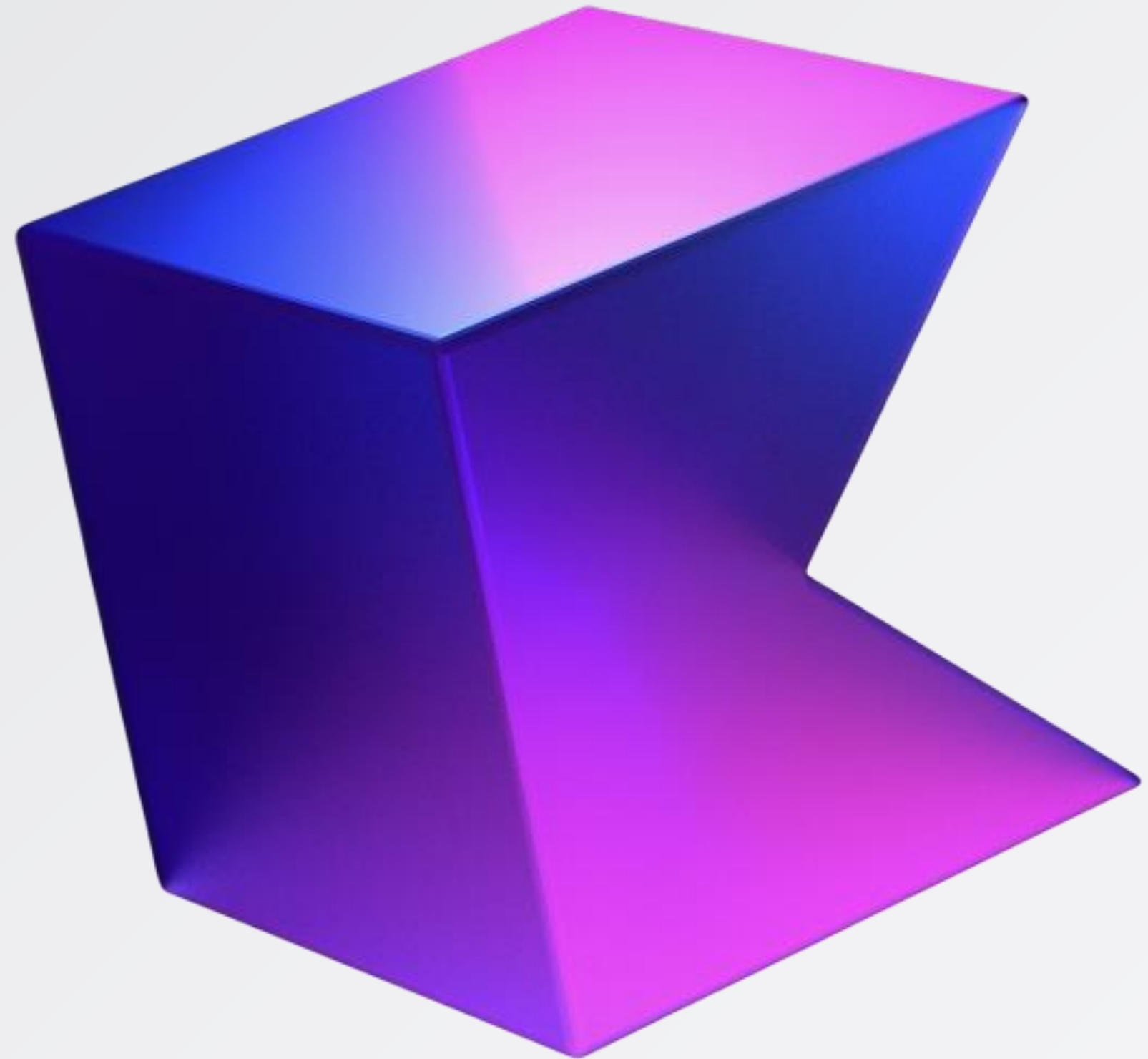
runCatching

```
fun main() {  
    val result = runCatching {  
        addBalance(-1)  
    }  
  
    result.isSuccess  
    result.isFailure  
    result.getOrNull()  
    result.getOrThrow()  
    result.getOrElse { println("Default value") }  
    result.getOrDefault()  
}
```

Result

```
Result.kt
Result
  constructor Result(Any?)
  value: Any?
  isSuccess: Boolean
  isFailure: Boolean
  getOrNull(): T?
  exceptionOrNull(): Throwable?
  toString(): String
  companion object of Result
  Failure
    constructor Failure(Throwable)
    exception: Throwable
    equals(Any?): Boolean
    hashCode(): Int
    toString(): String
  createFailure(Throwable): Any
  throwOnFailure() on Result<*>: Unit
  runCatching(() -> R): Result<R>
  runCatching(T.() -> R) on T: Result<R>
  getOrThrow() on Result<T>: T
  getOrElse((exception: Throwable) -> R) on Result<T>: R
  getOrDefault(R) on Result<T>: R
  fold((value: T) -> R, (exception: Throwable) -> R) on Result<T>: R
  map((value: T) -> R) on Result<T>: Result<R>
  mapCatching((value: T) -> R) on Result<T>: Result<R>
  recover((exception: Throwable) -> R) on Result<T>: Result<R>
  recoverCatching((exception: Throwable) -> R) on Result<T>: Result<R>
  onFailure((exception: Throwable) -> Unit) on Result<T>: Result<T>
  onSuccess((value: T) -> Unit) on Result<T>: Result<T>
```

Лямбды и extensions



Лямбда

Лямбда — это анонимная функция, которая может быть определена в виде выражения и предназначена для передачи функциональной логики как значения, позволяя более лаконично и гибко работать с функциями высшего порядка и коллекциями.

```
val myLambda: () -> Int = { 42 }
```

← Тело лямбды

↑ Возвращаемый тип

↑ Ключевое слово

Extensions

Экстеншен на String



```
public inline fun <R> String.map(transform: (Char) -> R): List<R> {  
    return mapTo(ArrayList<R>(length), transform)  
}
```


Map































```
fun main() {  
    val myString = "String"  
    myString.map { it.uppercase() }  
}
```

↑
Лямбда

-> **STRING**

Extensions

▼  _Strings.kt

- ❗  elementAt(Int) on CharSequence: Char
- ❗  elementAtOrElse(Int, (Int) -> Char) on CharSequence: Char
- ❗  elementAtOrNull(Int) on CharSequence: Char?
- ❗  find((Char) -> Boolean) on CharSequence: Char?
- ❗  findLast((Char) -> Boolean) on CharSequence: Char?
- ❗  first() on CharSequence: Char
- ❗  first((Char) -> Boolean) on CharSequence: Char
- ❗  firstNotNullOf((Char) -> R?) on CharSequence: R
- ❗  firstNotNullOfOrNull((Char) -> R?) on CharSequence: R?
- ❗  firstOrNull() on CharSequence: Char?
- ❗  firstOrNull((Char) -> Boolean) on CharSequence: Char?
- ❗  getOrElse(Int, (Int) -> Char) on CharSequence: Char
- ❗  getOrNull(Int) on CharSequence: Char?
- ❗  indexOfFirst((Char) -> Boolean) on CharSequence: Int
- ❗  indexOfLast((Char) -> Boolean) on CharSequence: Int
- ❗  last() on CharSequence: Char
- ❗  last((Char) -> Boolean) on CharSequence: Char
- ❗  lastOrNull() on CharSequence: Char?
- ❗  lastOrNull((Char) -> Boolean) on CharSequence: Char?
- ❗  random() on CharSequence: Char
- ❗  random(Random) on CharSequence: Char
- ❗  randomOrNull() on CharSequence: Char?
- ❗  randomOrNull(Random) on CharSequence: Char?
- ❗  single() on CharSequence: Char
- ❗  single((Char) -> Boolean) on CharSequence: Char
- ❗  singleOrNull() on CharSequence: Char?
- ❗  singleOrNull((Char) -> Boolean) on CharSequence: Char?
- ❗  drop(Int) on CharSequence: CharSequence
- ❗  drop(Int) on String: String
- ❗  dropLast(Int) on CharSequence: CharSequence

Extensions

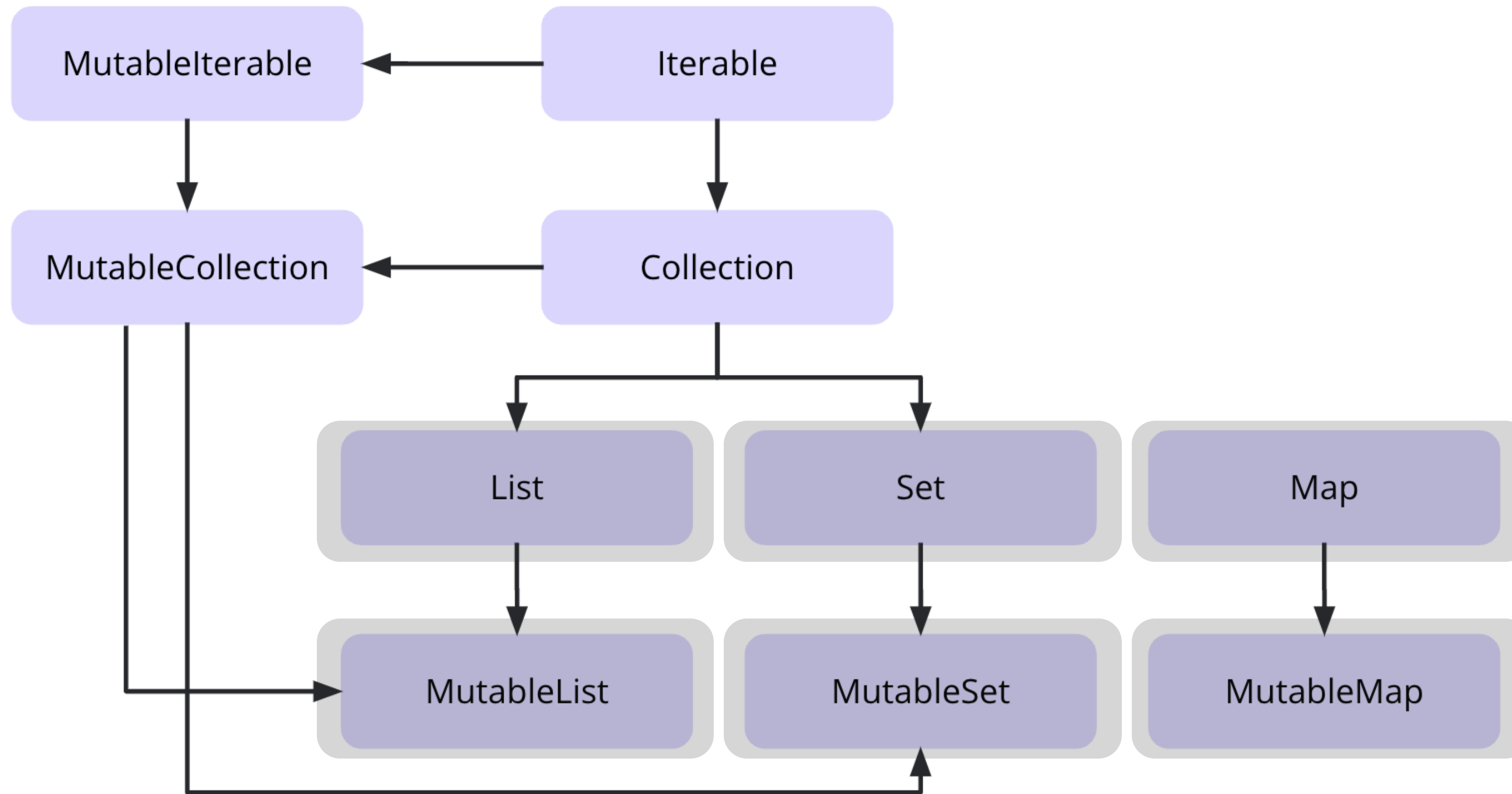
```
fun String.convertNameToGreeting() = "Hello, $this"
```

Extensions

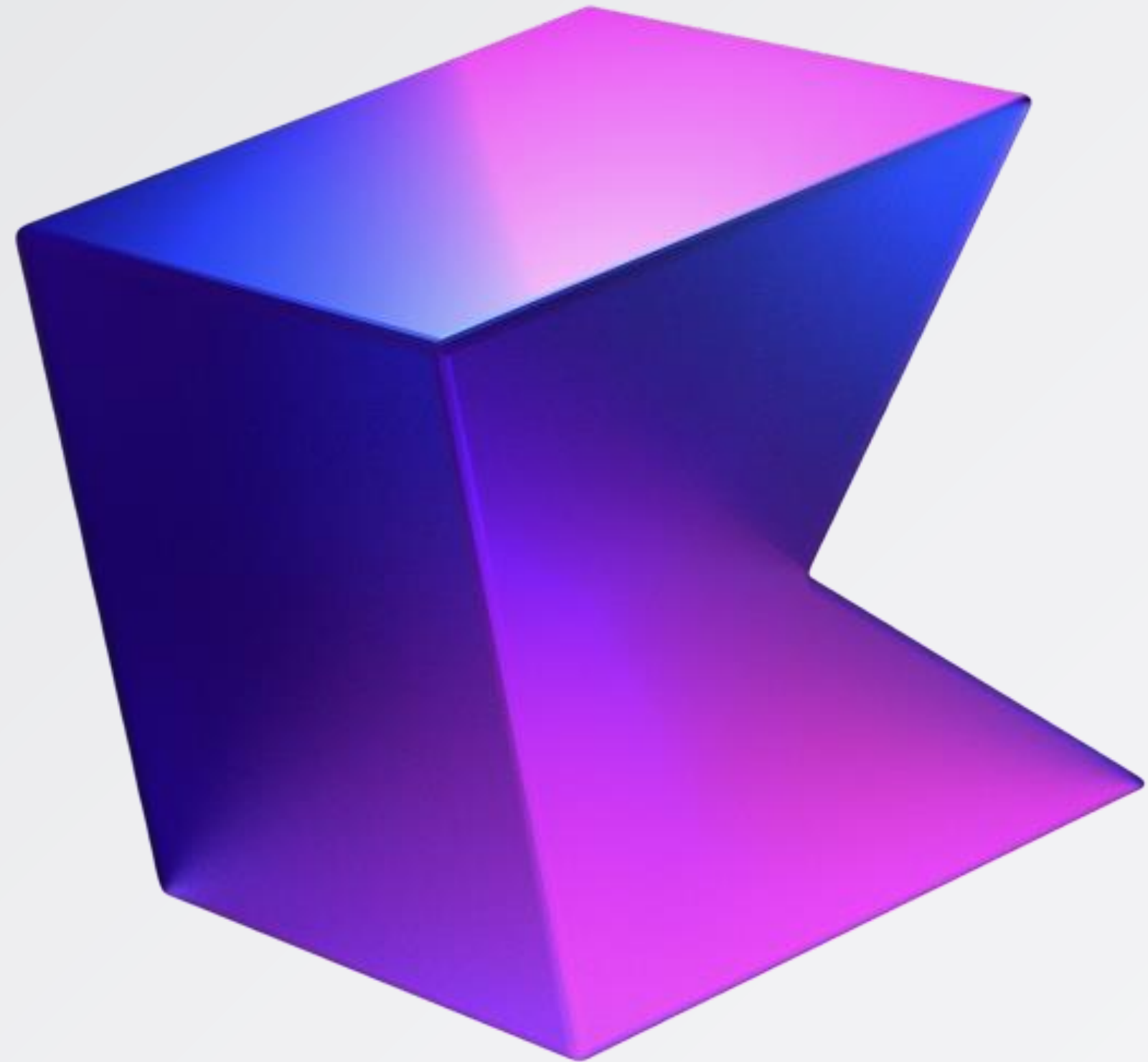
```
fun main() {  
    val numbers = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
  
    val result = numbers  
        .filter { it % 2 == 0 }           ← Фильтруем числа, оставляем только четные  
        .map { it * it }                  ← Возводим в квадрат  
        .sortedDescending()              ← Сортируем по убыванию  
        .take(3)                         ← Берем первые 3 часа  
        .map { it.toString() }           ← Преобразовываем в строку  
  
    println(result)  
}
```

-> 100, 64, 36

Коллекции



Delegates



Delegates

Делегаты — это механизм, позволяющий переадресовать реализацию свойства или поведения на другой объект, облегчая управление логикой и повторное использование кода.

```
val lazyVal by lazy { 5 }
```



Ленивая переменная

Значение в лямбде будет подсчитано только только при первом вызове

```
var notNullVar by Delegates.notNull<Int>()
```



Проверка на null, некая замена lateinit

```
var observableVar by Delegates.observable(  
    initialValue = 0,  
    onChange = { _, _, _ -> println("Value changed") }  
)
```



Вызов коллбека на изменение переменной

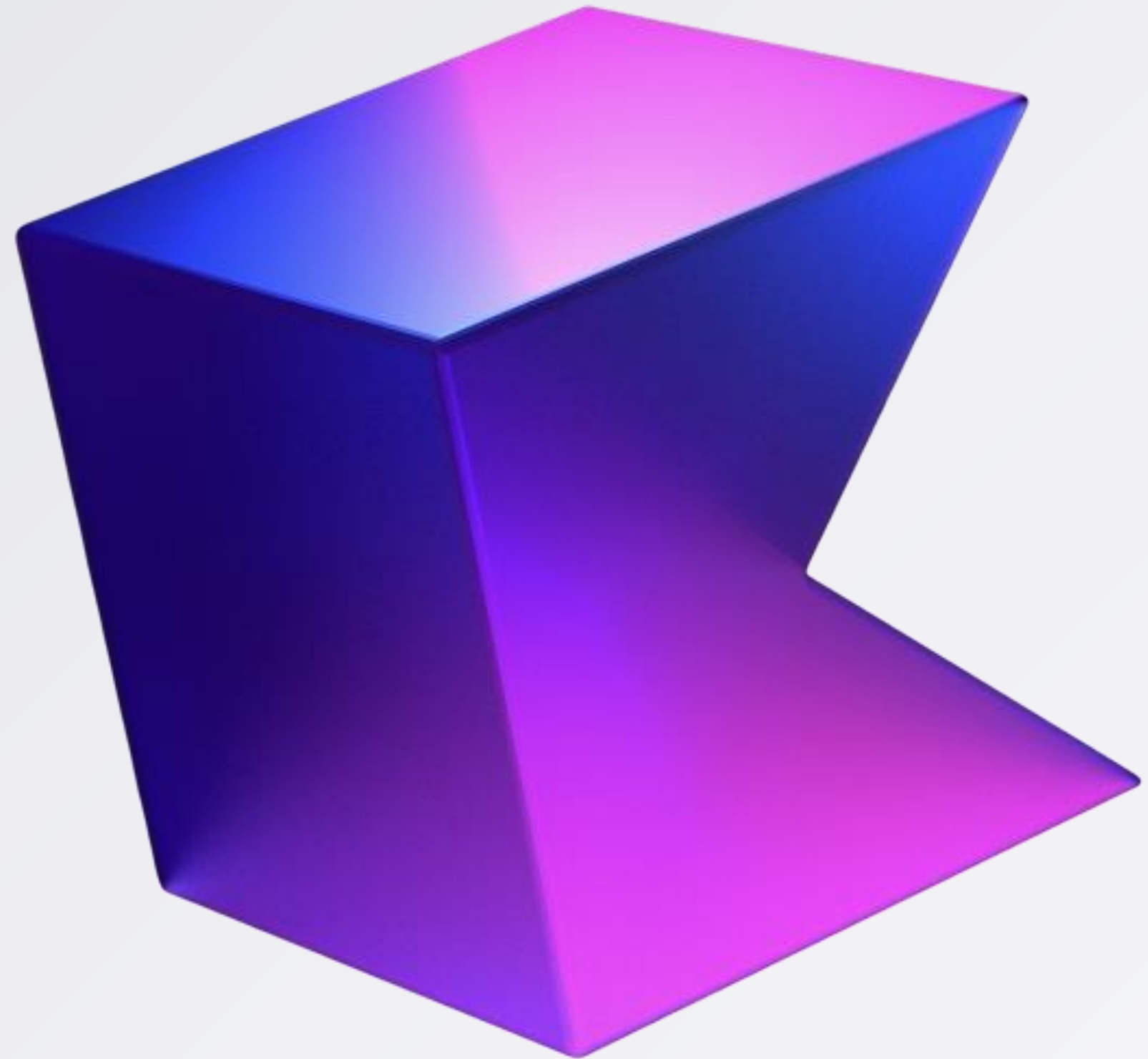
```
var vetoableVar by Delegates.vetoable(  
    initialValue = 0,  
    onChange = { _, _, value ->  
        value > 0  
    }  
)
```

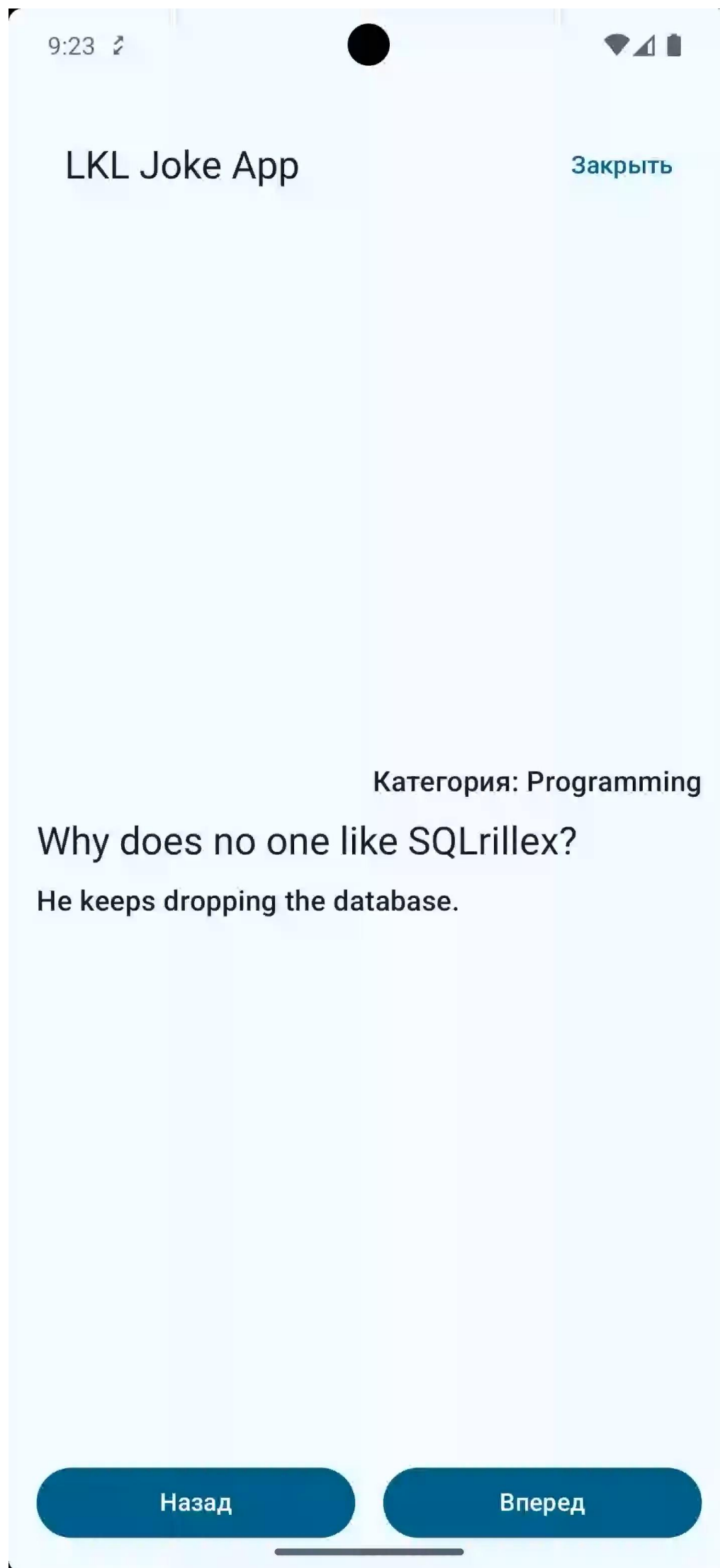


Вызов коллбека на изменение переменной

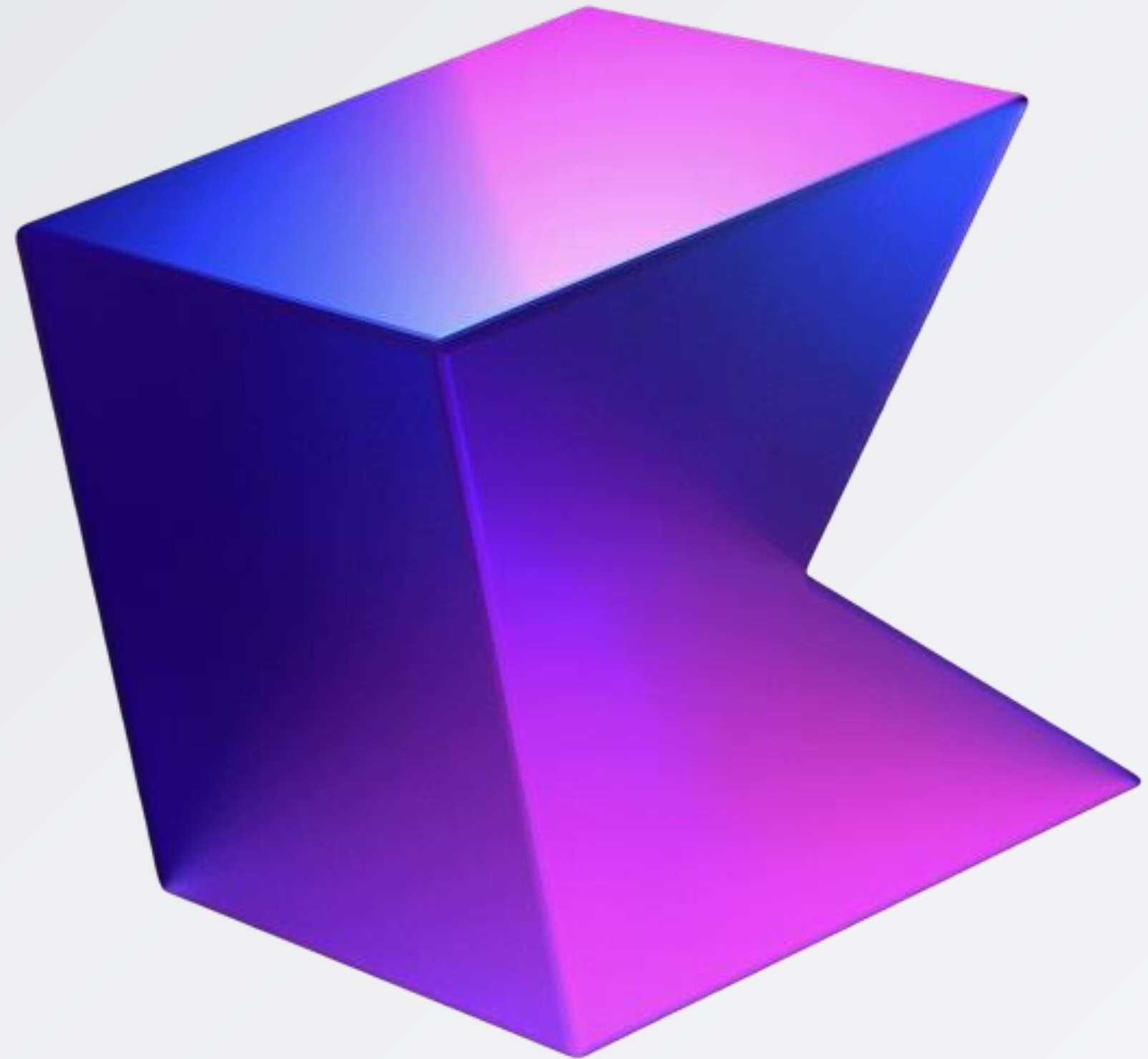
Проверка изменения значения

Приложение с шутками





Activity

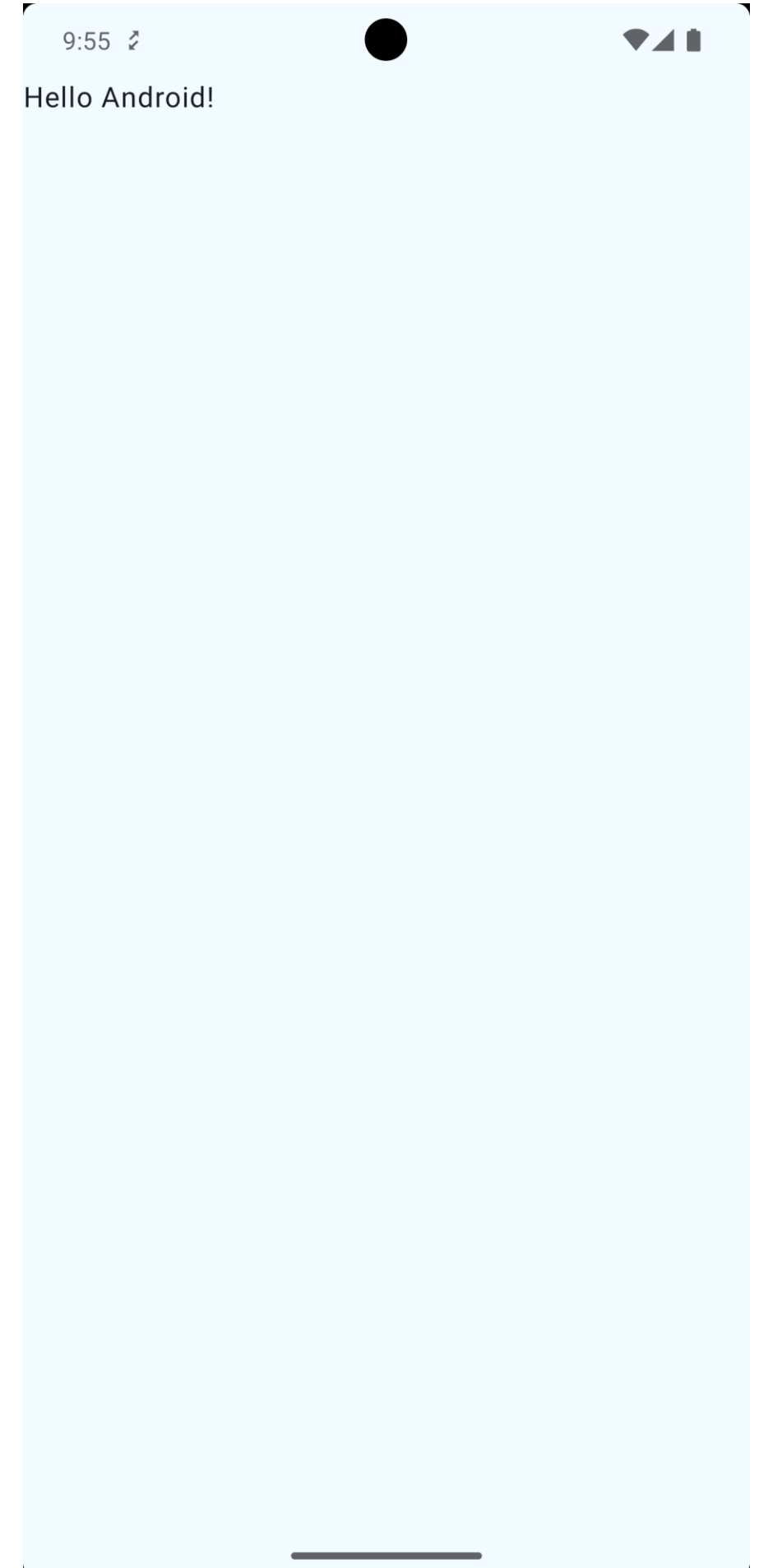


Activity

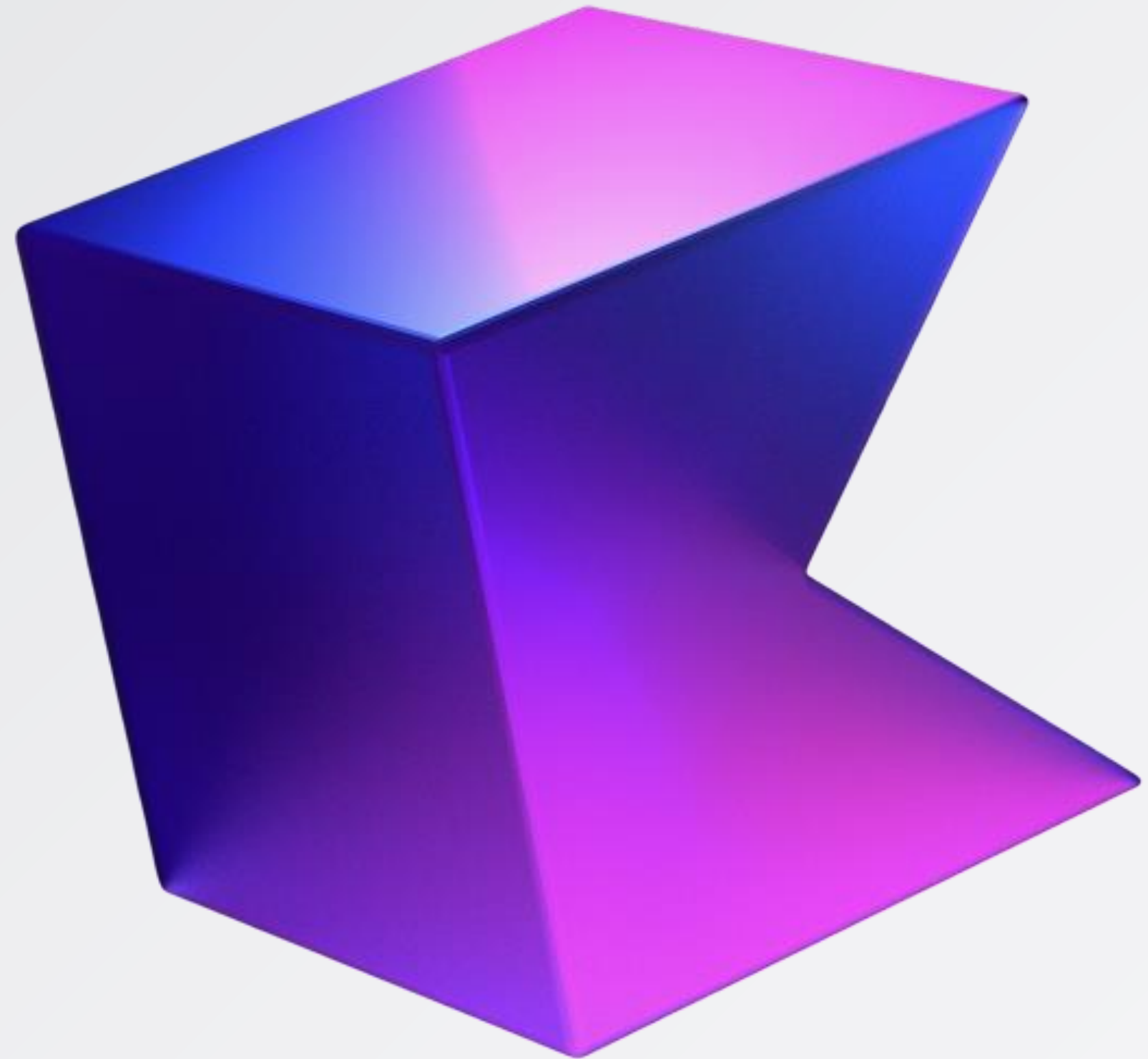
Activity в Android — это как отдельный экран в приложении, который пользователь видит и с которым взаимодействует. Каждое приложение может иметь несколько таких экранов, выполняющих разные функции.

developer.android.com/guide/components/activities/intro-activities

```
class MainActivity : ComponentActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        enableEdgeToEdge()  
        setContent {  
            ...  
        }  
    }  
}
```



Jetpack Compose



Jetpack Compose

Jetpack Compose — это современный способ создания интерфейсов в Android-приложениях, используя Kotlin вместо XML. Он предлагает более простой и интуитивный подход к разработке, упрощая код и управление состоянием по сравнению с традиционным XML-подходом.

developer.android.com/compose

@Composable

```
fun Greeting(name: String, modifier: Modifier = Modifier) {  
    Text(  
        text = "Hello $name!",  
        modifier = modifier  
    )  
}
```

XML

```
<?xml version="1.0" encoding="utf-8"?>
  <androidx.coordinatorlayout.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    tools:context=".MainActivity">

    <com.google.android.material.appbar.AppBarLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:fitsSystemWindows="true">

      <com.google.android.material.appbar.MaterialToolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize" />

    </com.google.android.material.appbar.AppBarLayout>

    <include layout="@layout/content_main" />

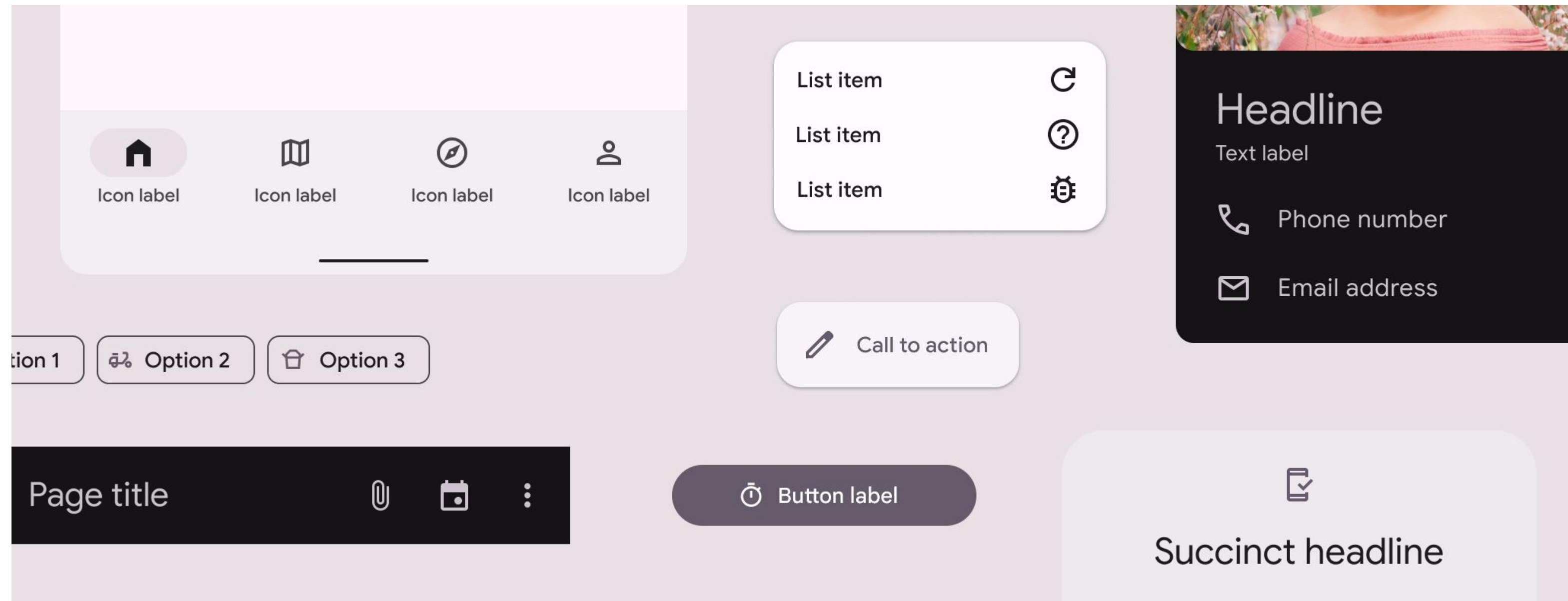
    <com.google.android.material.floatingactionbutton.FloatingActionButton
      android:id="@+id/fab"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="bottom|end"
      android:layout_marginEnd="@dimen/fab_margin"
      android:layout_marginBottom="16dp"
      app:srcCompat="@android:drawable/ic_dialog_email" />

  </androidx.coordinatorlayout.widget.CoordinatorLayout>
```

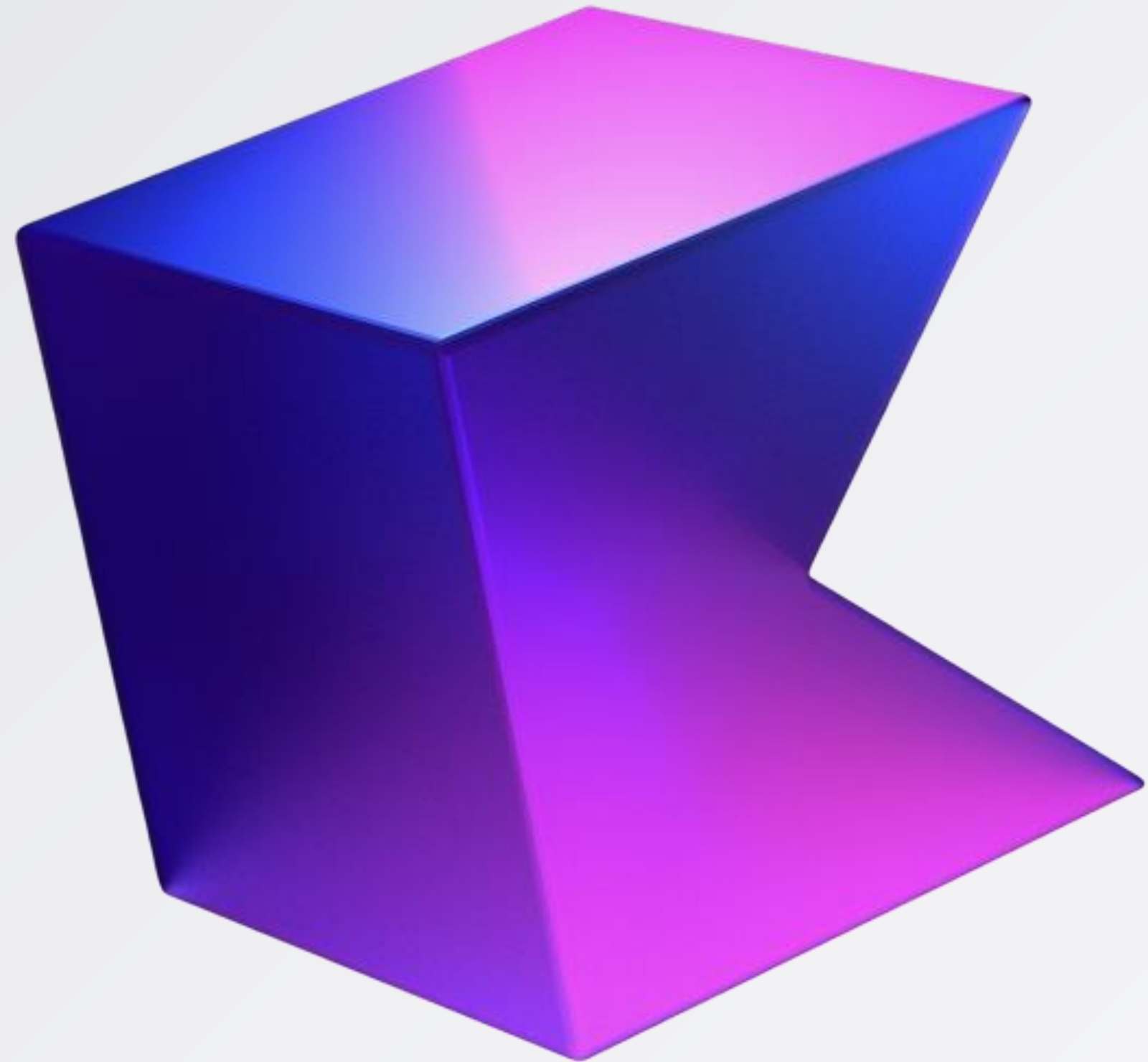
Material UI

Material UI — это стиль и руководство по созданию визуально приятных и последовательных интерфейсов, разработанное Google, которое помогает сделать приложения привлекательными и удобными для пользователя.

m3.material.io



ViewModel



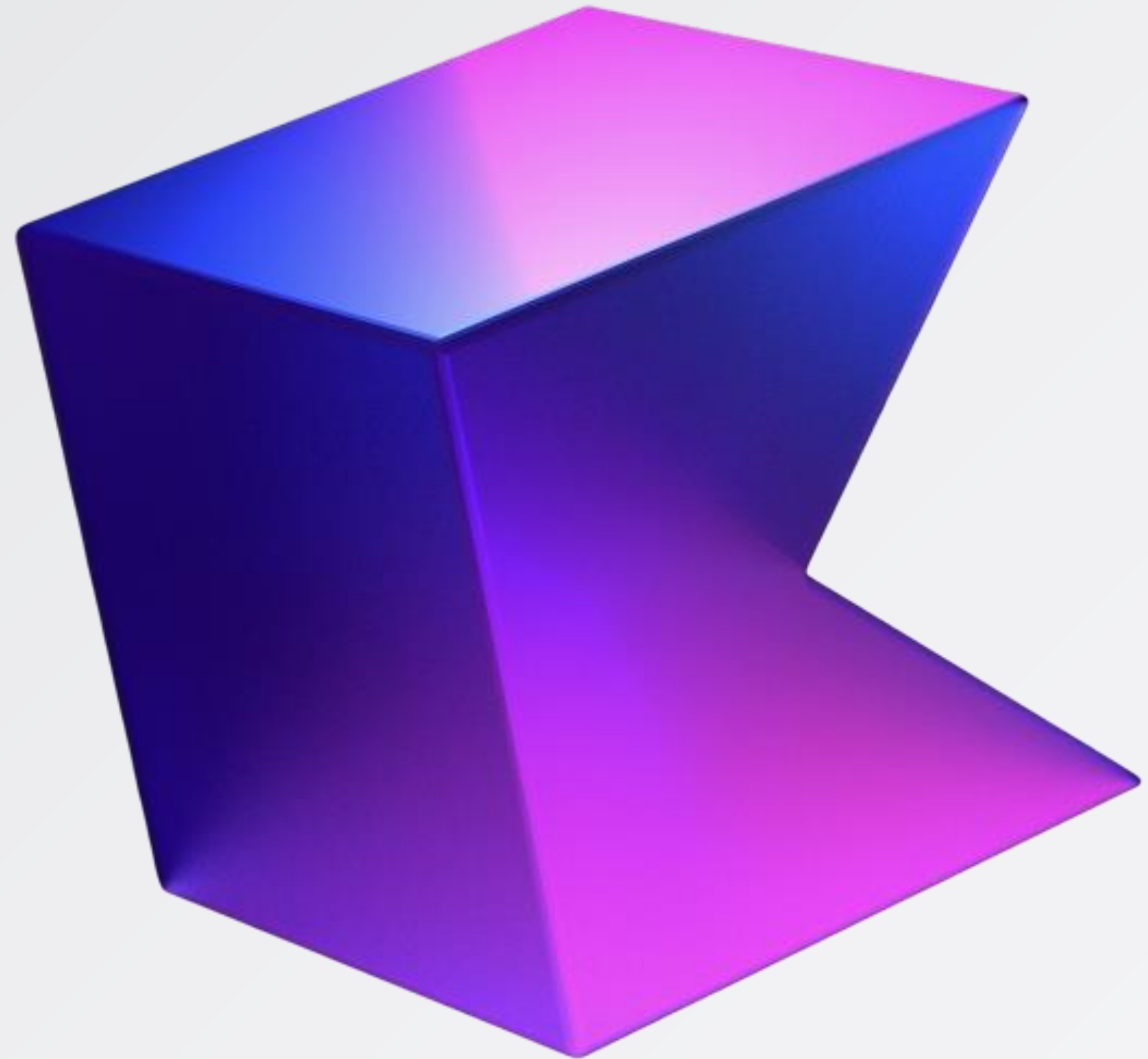
ViewModel

ViewModel — это компонент Android, который помогает хранить и управлять данными, необходимыми для пользовательского интерфейса. Он живет дольше, чем обычные компоненты, и переживает пересоздание Activity, сохраняя состояние данных и избегая повторных загрузок при повороте экрана или других конфигурационных изменениях.

developer.android.com/topic/libraries/architecture/viewmodel

```
class CounterViewModel : ViewModel() {  
    private var _count = 0  
    val count: Int  
        get() = _count  
  
    fun increment() {  
        _count++  
    }  
}
```

Coroutines



Coroutines

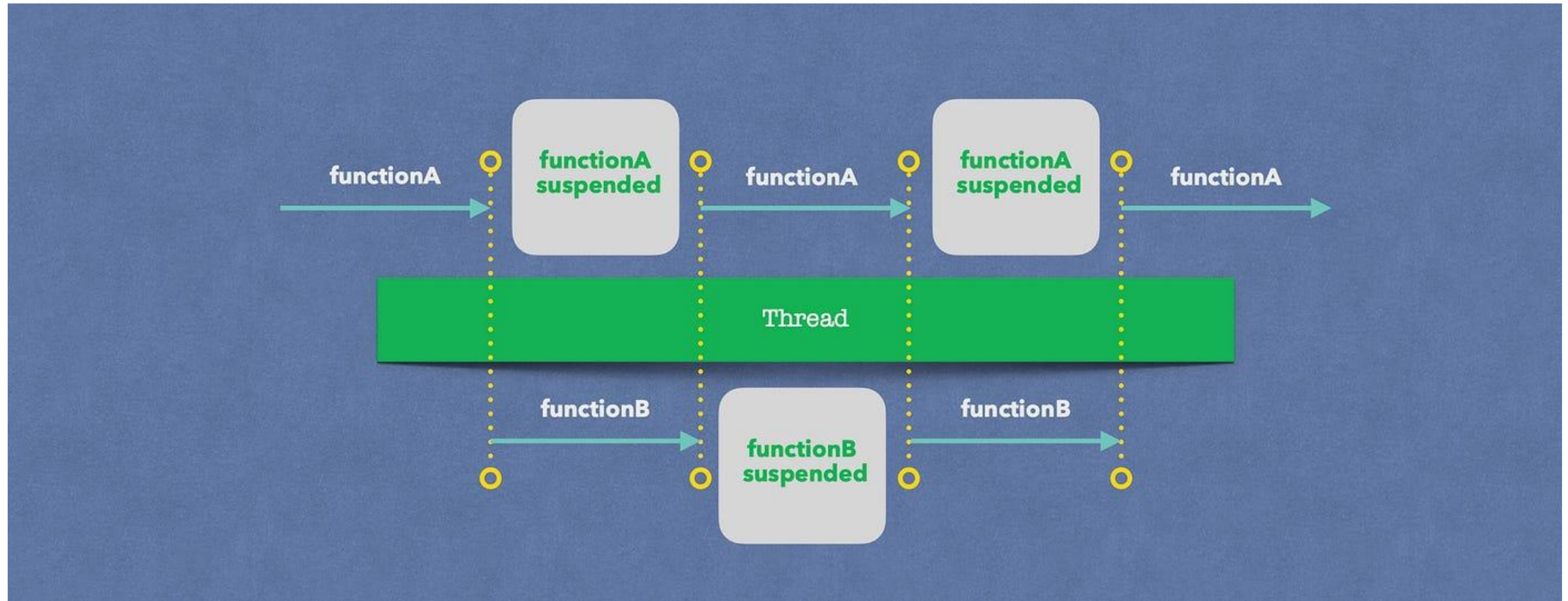
Корутины в Kotlin используются для выполнения асинхронных операций простым и устойчивым к ошибкам способом. Они облегчают выполнение долгих операций, таких как работа с сетью или базой данных, не блокируя основной поток.

kotlinlang.org/docs/coroutines-overview.html

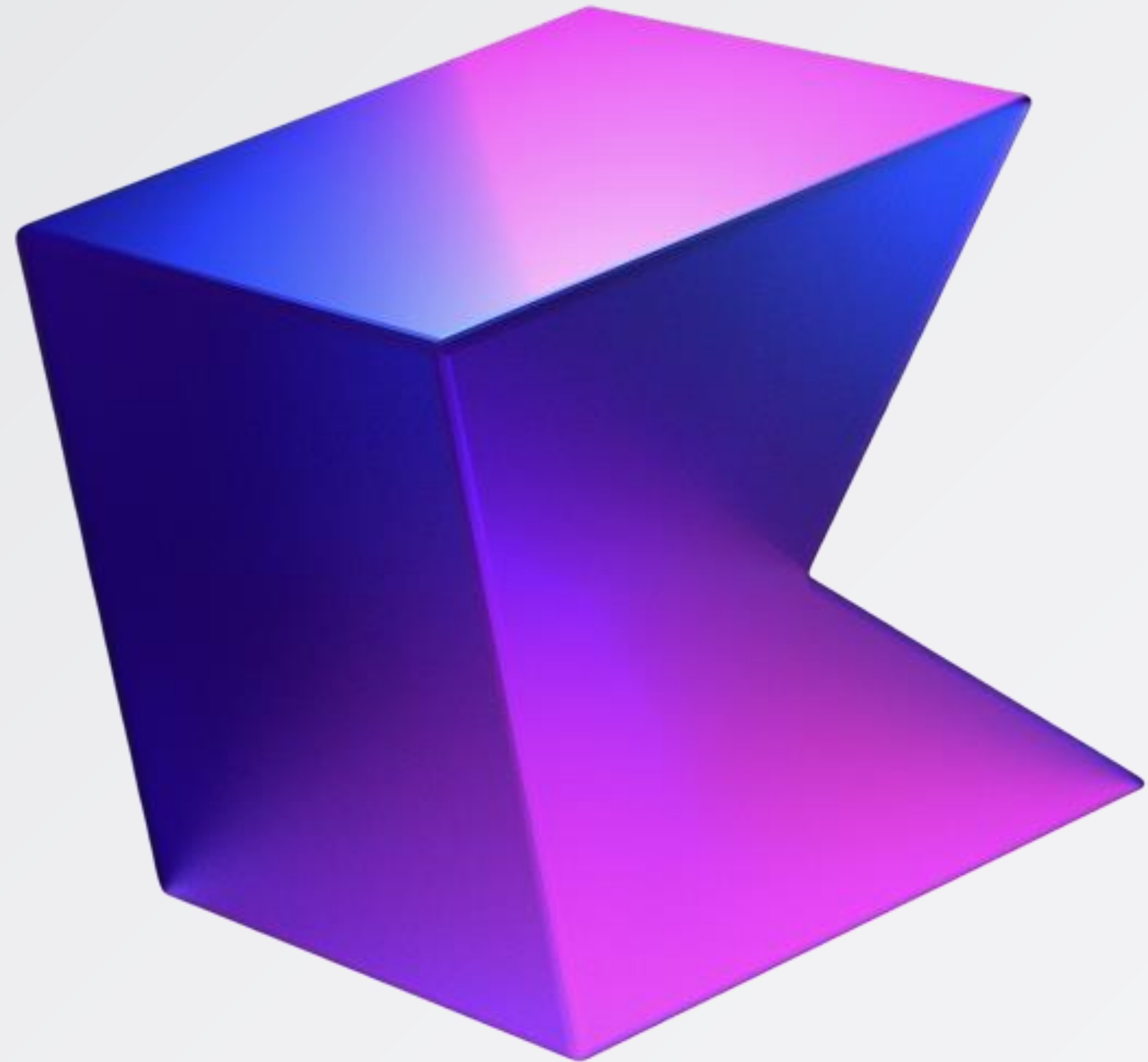
```
fun main() = runBlocking { // Запускаем корутину в потоке для примера
    launch { // Запускаем новую корутину в данном стеке
        performTask()
    }
}
```

```
suspend fun performTask() {
    println("Task started...")
    delay(2000) // Имитация длительной работы
    println("Task completed!")
}
```


Coroutines



Coroutine flow



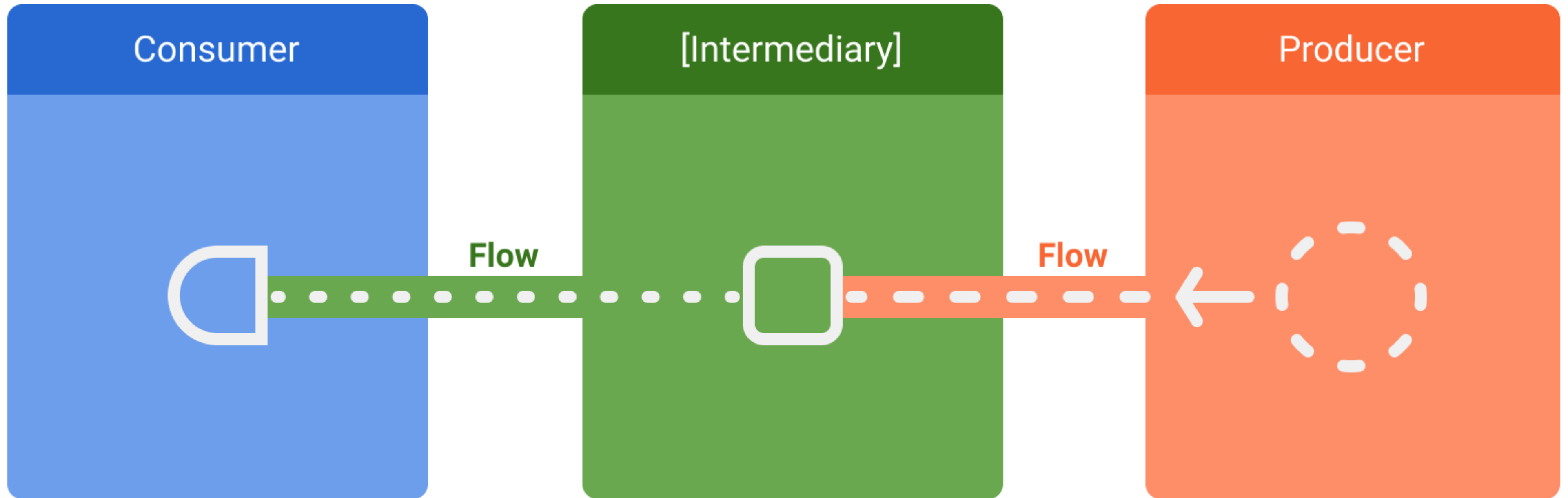
Coroutine flow

Flow в Kotlin — это API для работы с асинхронными потоками данных, который позволяет обрабатывать последовательности значений во времени, используя стиль программирования реактивных потоков. Flow часто используется для наблюдения за данными, которые обновляются или поступают со временем, например, от базы данных или удаленного сервера.

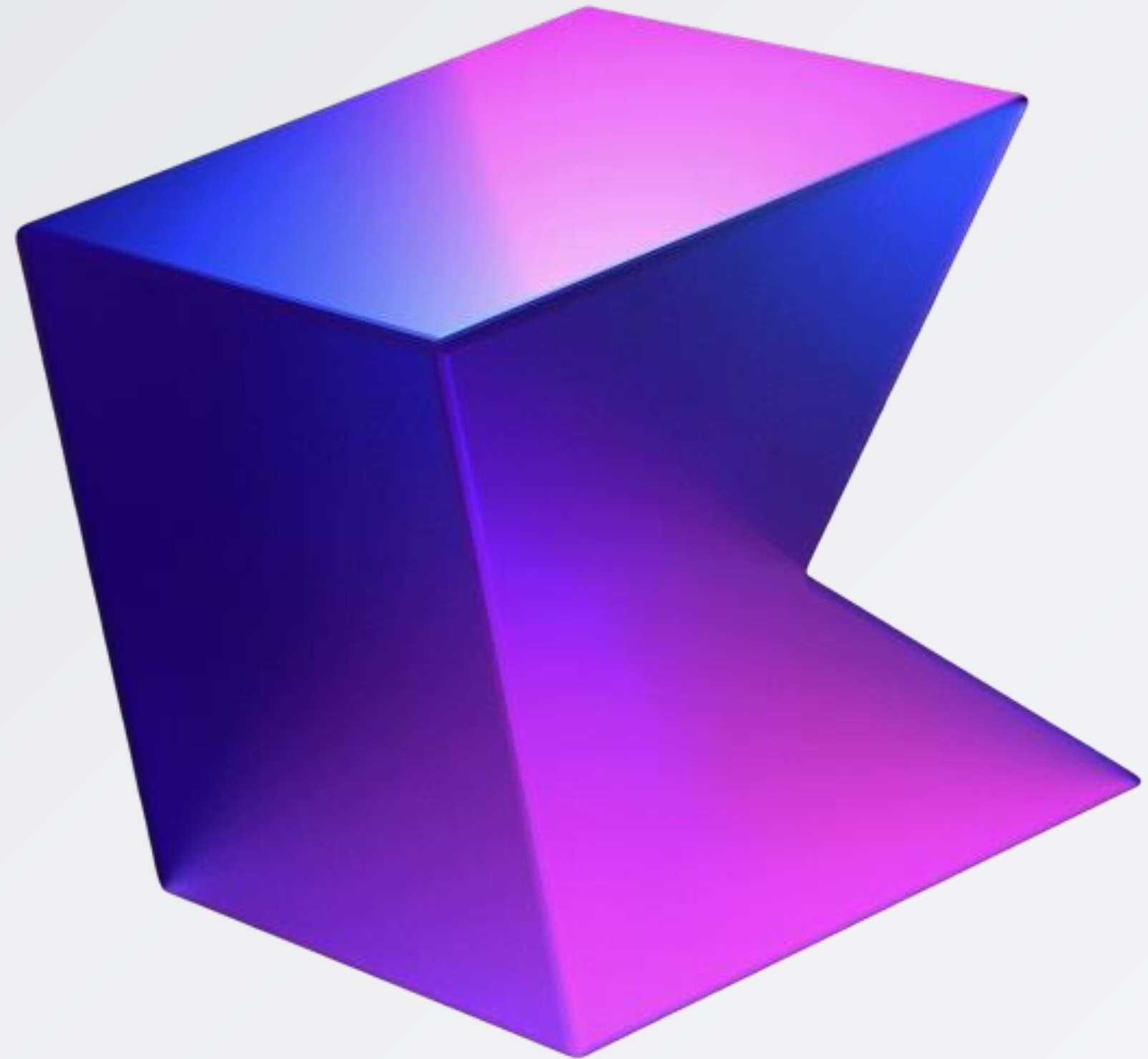
kotlinlang.org/docs/flow.html

```
fun numberFlow(): Flow<Int> = flow {  
    for (i in 1..5) {  
        delay(1000) // Эмулируем задержку  
        emit(i) // Публикуем значение  
    }  
}
```

Coroutine flow



Network



Http методы

HTTP методы — это действия, которые могут быть выполнены над ресурсами веб-сервера. Они указывают серверу, что клиент (например, браузер или приложение) хочет сделать с ресурсом, таким как веб-страница или данные.

Стандартный GET метод используется для запроса данных с сервера, не изменяя их; это самый распространенный метод, который используется для получения информации, отображаемой пользователю.

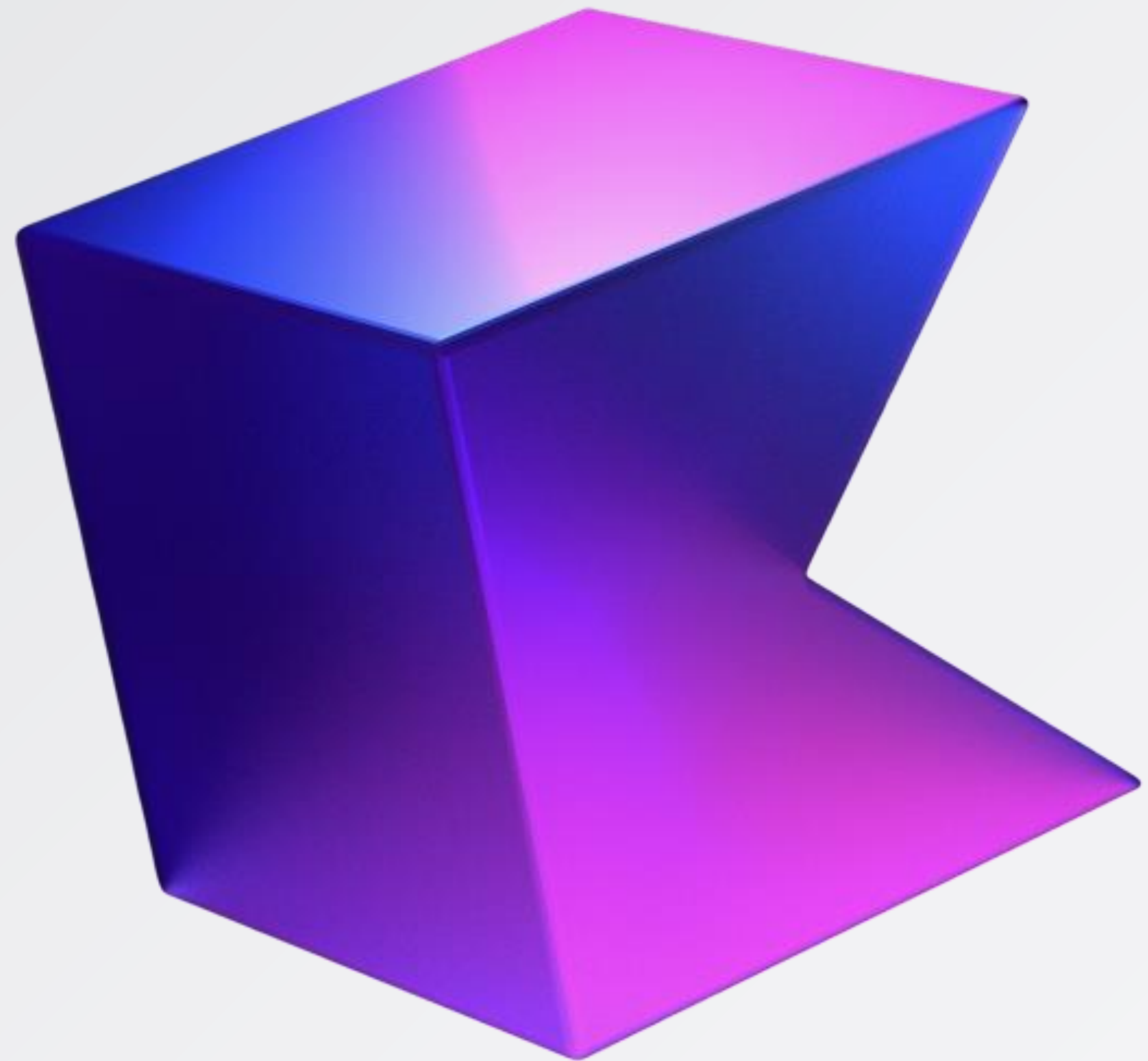
Метод	Что делает
GET	Запрашивает ресурс, расположенный по указанному URI.
POST	Используется для передачи на сервер данных, которые должны быть обработаны ресурсом, указанным в URI. Данные передаются в теле запроса.
PUT	Позволяет сохранить (или перезаписать) передаваемый на сервер ресурс с указанным URI.
DELETE	Используется для удаления ресурса, указанного в URI.
HEAD	Аналогичен GET, но клиенту возвращается только заголовок сообщения ответа. Этот метод можно использовать для проверки доступа к ресурсам.
OPTIONS	Запрашивает характеристики соединения между клиентом и сервером, характеристики сервера, требования для запроса данного ресурса и т. п. Если вместо идентификатора URI стоит символ «*», то запрашивается вся доступная информация о сервере.

JSON

JSON — это текстовый формат передачи данных, который легко читается человеком и машиной, и часто используется для обмена данными между клиентом и сервером. Он представляет собой структуру, основанную на парах ключ-значение и списках, что делает его удобным для передачи объектов и массивов. JSON поддерживает простые типы данных, такие как строки, числа, логические значения, а также вложенные структуры.

```
{  
  "error": false,  
  "category": "Programming",  
  "type": "twopart",  
  "setup": "How did the programmer die in the shower?",  
  "delivery": "He read the shampoo bottle instructions: Lather. Rinse. Repeat.",  
  "flags": {  
    "nsfw": false,  
    "political": false,  
    "explicit": false  
  },  
  "safe": false,  
  "id": 266,  
  "lang": "en"  
}
```

OkHttpClient / Retrofit



OkHttp

OkHttp — это популярная библиотека для Android и Java, используемая для выполнения сетевых операций. Она позволяет выполнять HTTP-запросы и обрабатывать ответы, предлагая возможности для настройки клиента, кэширования ответов, отслеживания запросов и их логирования.

square.github.io/okhttp/

// Создание логирующего интерсептора

```
val logging = HttpLoggingInterceptor().apply {  
    level = HttpLoggingInterceptor.Level.BODY  
}
```

// Настройка клиента с интерцептором

```
val client = OkHttpClient.Builder()  
    .addInterceptor(logging)  
    .build()
```

Retrofit

Retrofit — это библиотека для Android и Java, которая облегчает работу с RESTful веб-сервисами. Она предоставляет удобный способ выполнения HTTP-запросов и обработки ответов, автоматически преобразуя JSON-данные в объекты Kotlin или Java.

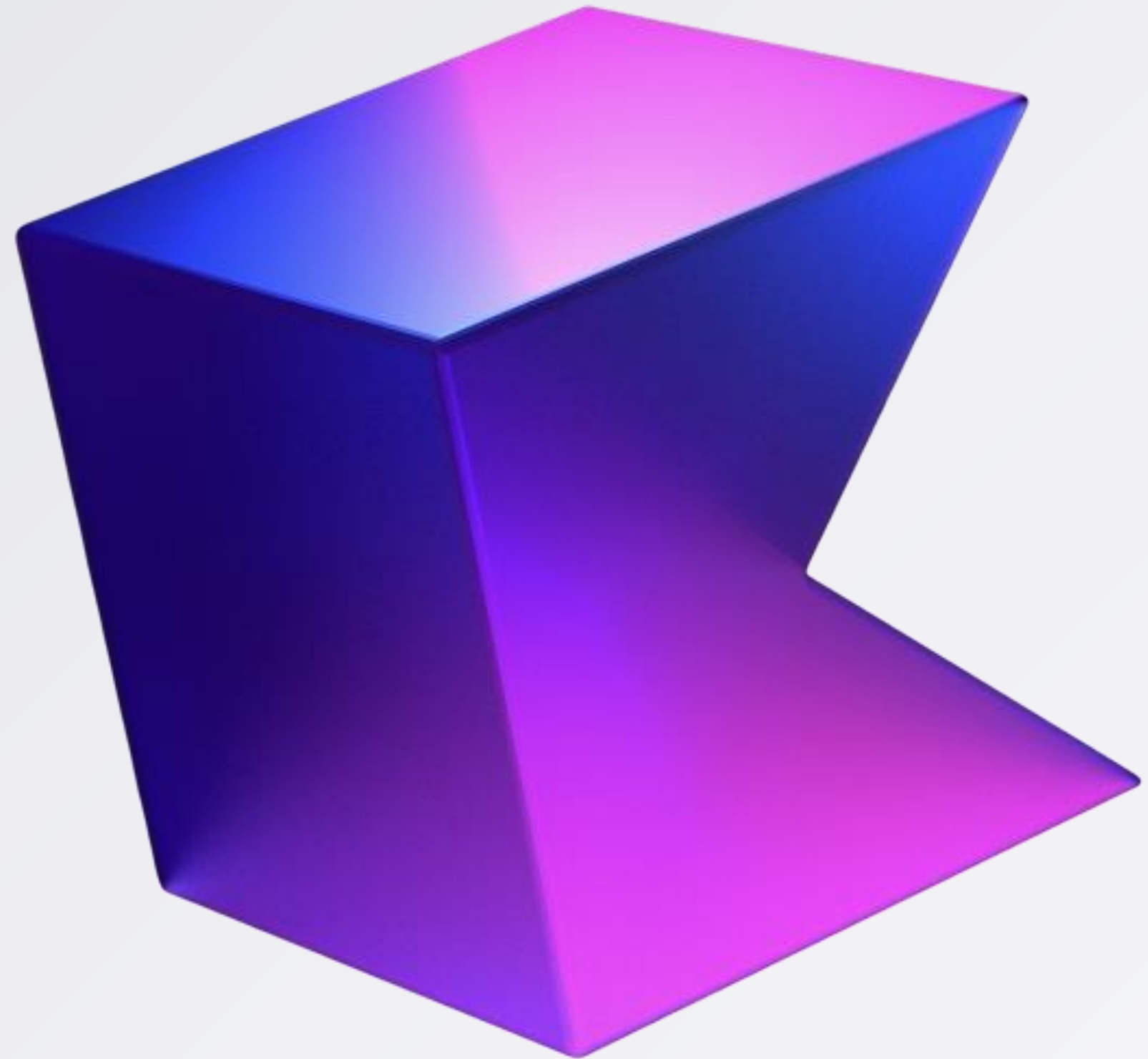
square.github.io/retrofit/

```
interface JokeApi {
    @GET("joke/programming?type=twopart")
    suspend fun getRandomProgrammingJoke(): Joke
}

fun createJokeApiService(): JokeApi {
    val baseUrl = "https://v2.jokeapi.dev/"
    val json = Json {
        ignoreUnknownKeys = true
    }
    val contentType = "application/json".toMediaType()

    return Retrofit.Builder()
        .baseUrl(baseUrl)
        .client(okHttpClient)
        .addConverterFactory(json.asConverterFactory(contentType))
        .build()
        .create(JokeApi::class.java)
}
```

Kotlinx Serialization



Kotlinx Serialization

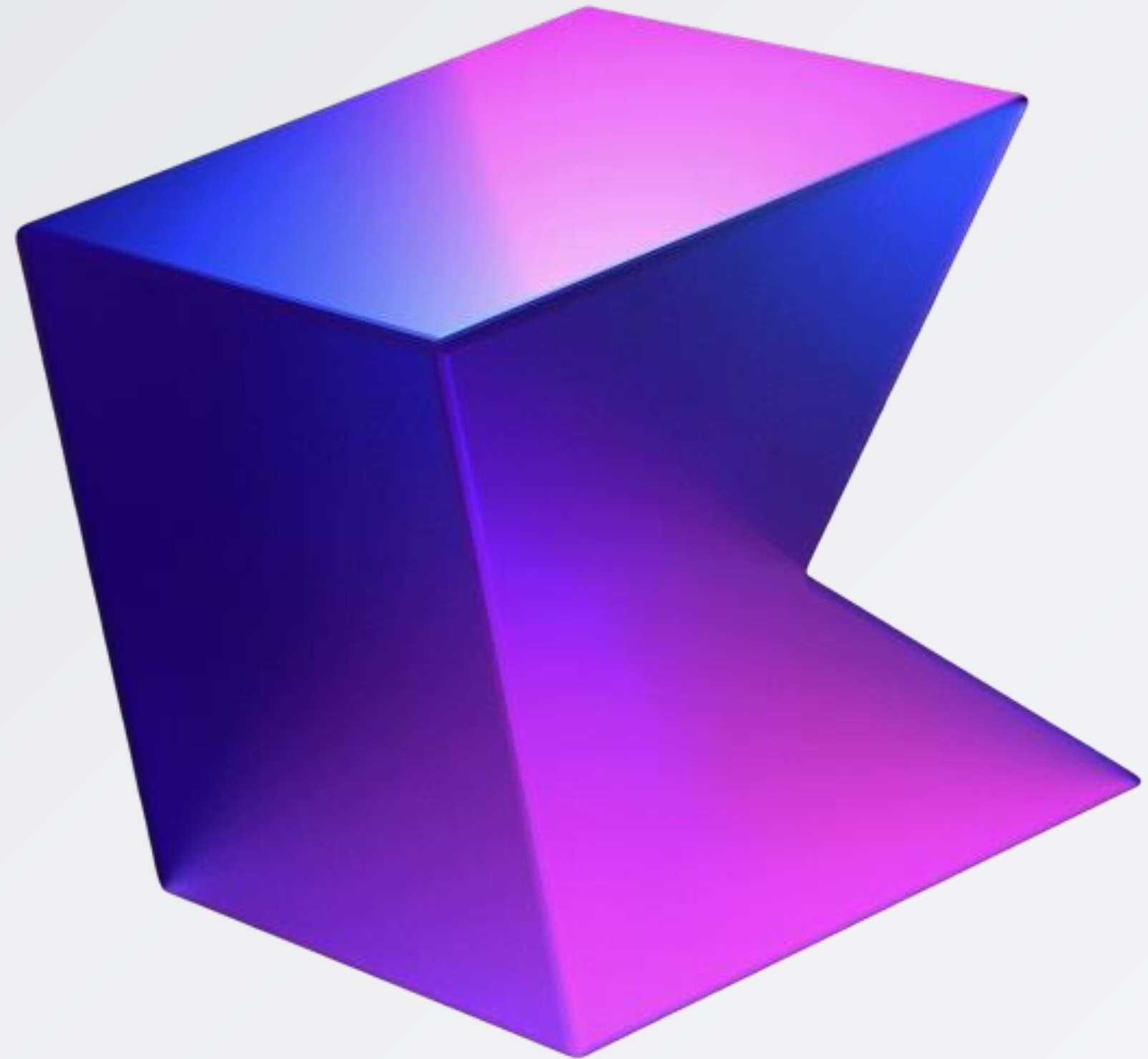
Kotlinx.serialization — это библиотека Kotlin, которая обеспечивает механизм сериализации и десериализации объектов в различные форматы, такие как JSON, ProtoBuf и другие. Она поддерживает аннотации для указания, как данные должны быть преобразованы.

kotlinlang.org/docs/serialization.html

@Serializable

```
data class Joke(  
    @SerializedName("error") val error: Boolean,  
    @SerializedName("category") val category: String,  
    @SerializedName("type") val type: String,  
    @SerializedName("setup") val setup: String,  
    @SerializedName("delivery") val delivery: String,  
    @SerializedName("id") val id: Int  
)
```

Manifest



Manifest

Файл AndroidManifest.xml в Android играет роль конфигурационного файла, который содержит важную информацию о структуре и возможностях вашего приложения. Он определяет компоненты приложения (такие как активности), необходимые разрешения, темы, а также основные метаданные приложения.

developer.android.com/guide/topics/manifest/manifest-intro

```
<manifest>
```

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

```
    <application>
```

```
        <activity android:name=".MainActivity">
```

```
            <intent-filter>
```

```
                <action android:name="android.intent.action.MAIN" />
```

```
            </intent-filter>
```

```
        </activity>
```

```
    </application>
```

```
</manifest>
```

Что почитать?

Документация по Kotlin: <https://kotlinlang.org>

Роман Елизаров, Дмитрий Жемеров, Светлана Исакова "Kotlin In Action", 2-е издание

Марчин Москала "Effective Kotlin"

Kotlin Koans – если вы знаете Java



Спасибо!

**Ваши
вопросы?**

