ГЛАВА 1

Создаем и настраиваем проект Kotlin Multiplatform

В этой главе рассмотрим:

- какие инструменты нужны для разработки;
- как их скачать и настроить;
- каково базовое устройство проекта KMP;
- какие существуют альтернативные IDE для работы;
- как превратить нативный проект в кросс-платформенный.

Для того чтобы начать разрабатывать мобильные приложения на Kotlin Multiplatform, нам прежде всего понадобится установить и настроить все инструменты и IDE. Вам потребуется SDK для всех таргетов, которые вы собираетесь поддерживать, то есть Android SDK для Android и iOS SDK, если вы будете писать под iOS.

> **ПРИМЕЧАНИЕ** Учтите, что если вы планируете разрабатывать мобильные приложения под iOS, то вам потребуется компьютер с Mac OS (Macbook Pro или Mac mini), так как на другую систему SDK установить не получится. Для скачивания инструментов вам будет нужен действующий аккаунт Apple Developer (для приобретения переходите на https://developer.apple.com).

Основной и официальной IDE для работы с Kotlin Multiplatform на данный момент является Android Studio от компании JetBrains. Для скачивания свежей версии необходимо перейти на сайт https://developer.android.com и выбрать Android Studio в разделе Develop Get (рис. 1.1).

Затем для скачивания Android Studio последней стабильной версии нажимаем Download Android Studio (рис. 1.2).

> **ВНИМАНИЕ!** Следующий шаг зависит от того, на какую ОС вы будете устанавливать IDE. Для Mac OS необходимо будет выбрать вариант Android Studio, соответствующий архитектуре вашего процессора (рис. 1.3).

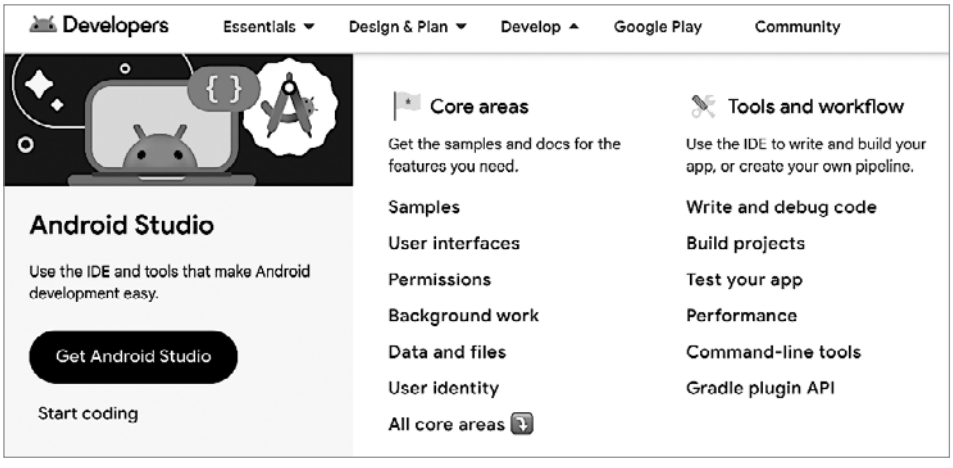

Рис. 1.1. Сайт developer.android.com

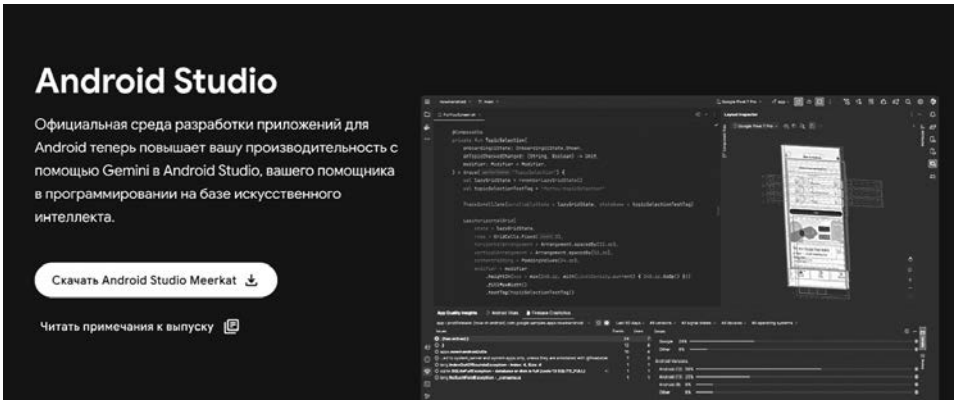

Рис. 1.2. Переходим к скачиванию

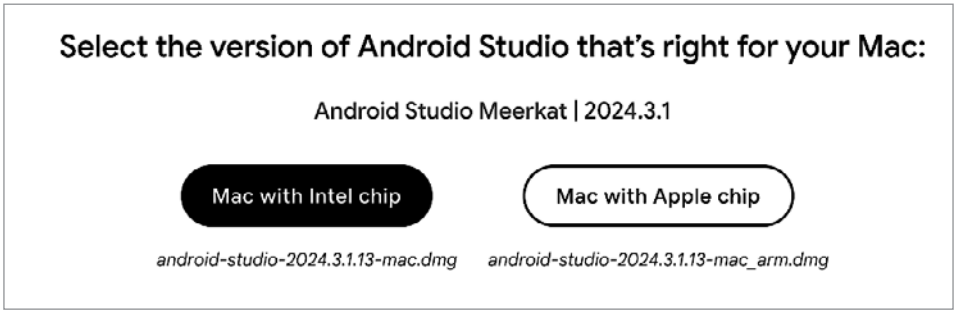

Рис. 1.3. Варианты дистрибутива

После того как вы скачали инсталлятор, запускайте его и следуйте инструкциям мастера установки — все стандартно. Если на вашем компьютере еще нет Android SDK, то вам предложат пройти шаг автоматической загрузки и установки компонентов SDK (рис. 1.4).

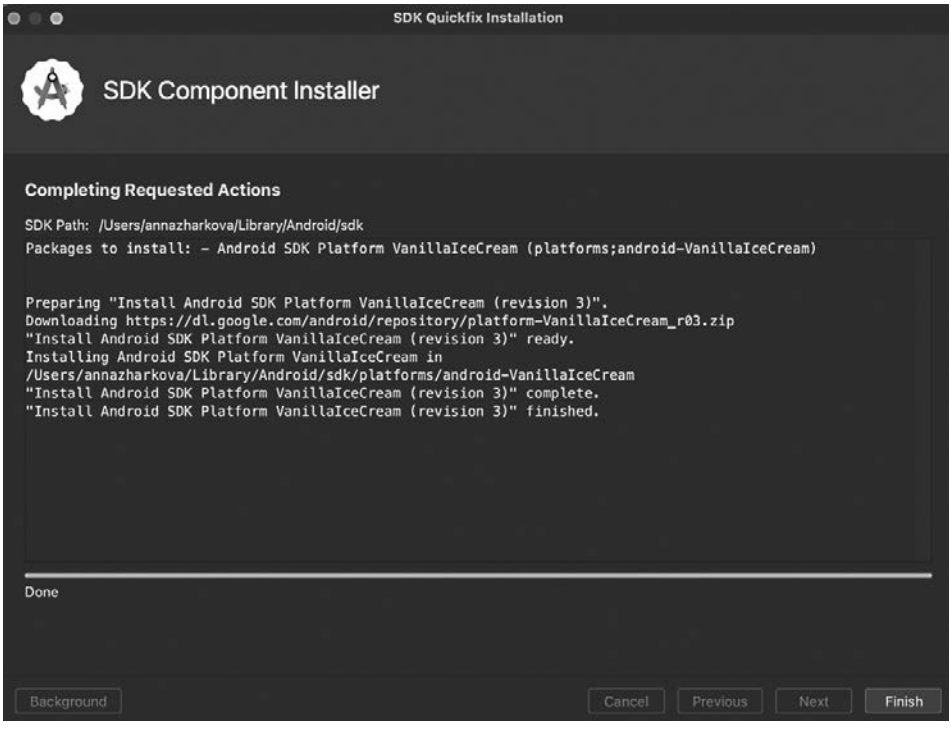

Рис. 1.4. Начало установки Android SDK

После нажатия кнопки Finish вы готовы приступить к работе. Остается одна маленькая деталь — установка плагина Kotlin Multiplatform Mobile. Для этого перейдем в меню Settings ▸ Plugins и на вкладке Marketplace наберем название нужного плагина в строке поиска (рис. 1.5).

Нажмем Apply ▸ OK.

> **ВНИМАНИЕ!** Если по каким-то причинам у вас проблемы с доступом к JetBrains Marketplace и самим библиотекам, попробуйте включить дополнительные настройки прокси сети вашего компьютера.

Если вы собираетесь разрабатывать и под iOS, вам потребуется установить Xcode со встроенным iOS SDK. Для загрузки перейдите на https://developer.apple.com/xcode/ (рис. 1.6).

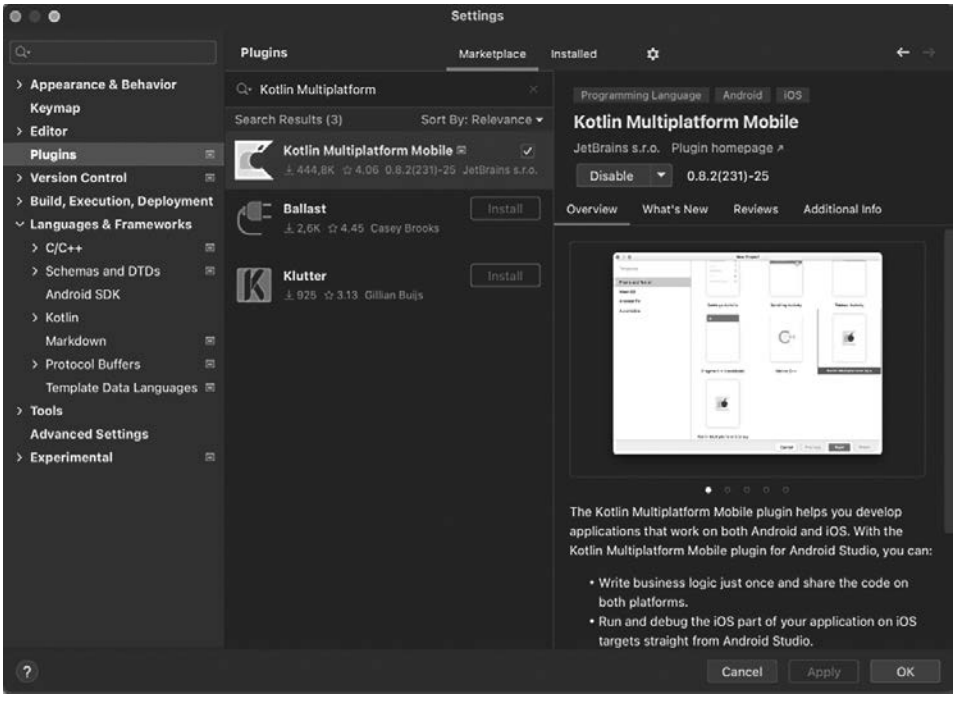

Рис. 1.5. Установка плагина

Рис. 1.6. Загружаем Xcode

Обратите внимание, что Kotlin Multiplatform гарантированно работает только со стабильными версиями Xcode.

Итак, все скачали, установили, и теперь вы точно готовы к работе.

1.1. Создание проекта с нуля

Начнем с создания приложения с нуля. Для этого переходим в меню File ▸ New ▸ New Project. Плагин KMM (Kotlin Multiplatform Mobile) позволяет нам создать базовое приложение с помощью базового шаблона (рис. 1.7).

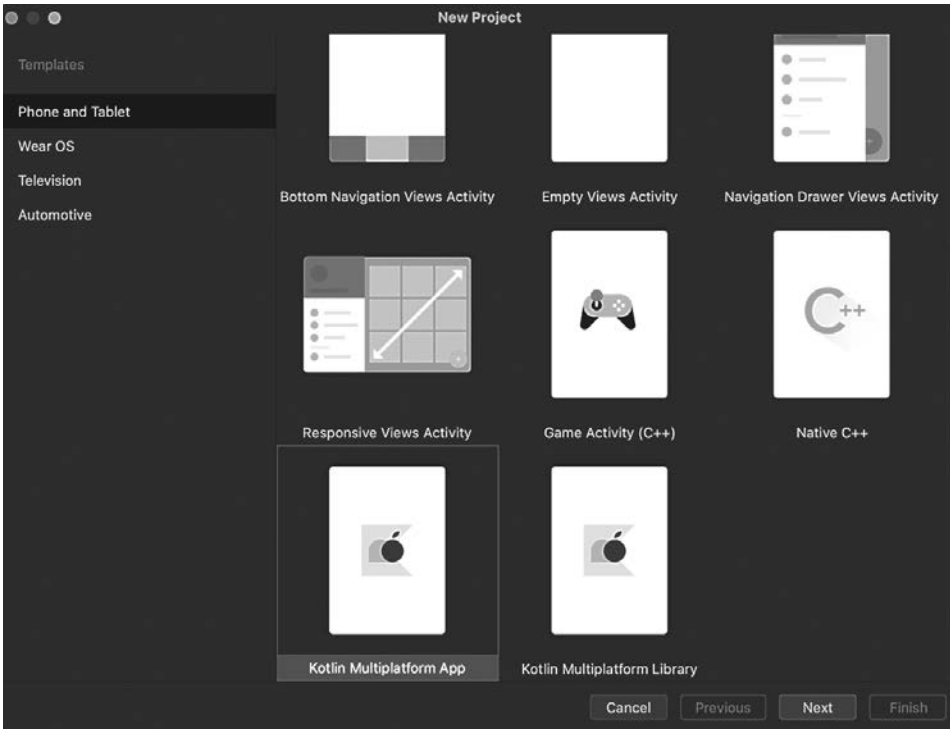

Рис. 1.7. Выбор шаблона приложения

Шаблон Koltin Multiplatform App создаст базовое приложение с несколькими мобильными таргетами. В мастере создания укажем наименование проекта, его расположение, а также целевую версию Android (рис. 1.8).

Далее настроим наименование общего модуля кода, название модулей нативных приложений и выберем способ подключения shared-модуля к iOS-приложению (рис. 1.9).

В настоящее время поддерживается подключение модуля как обычного Xcode-фреймворка или через CocoaPods. Ожидается, что через какое-то время будет возможно подключение и через Swift Package Manager. На момент написания книги соответствующая задача в YouTrack JetBrains еще в статусе Open, но большая часть связанных задач уже закрыта (рис. 1.10).

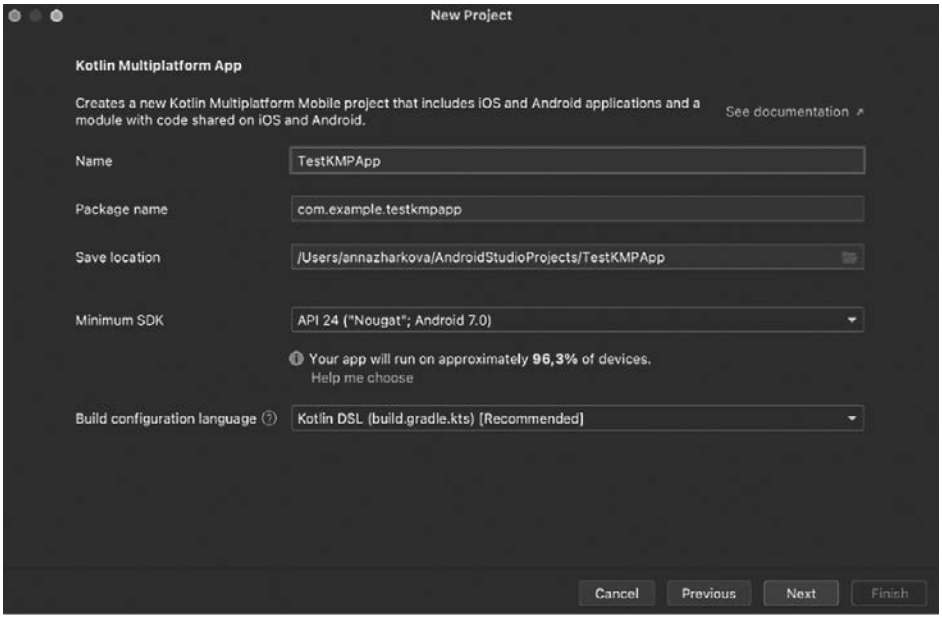

Рис. 1.8. Настройка имени проекта и расположения

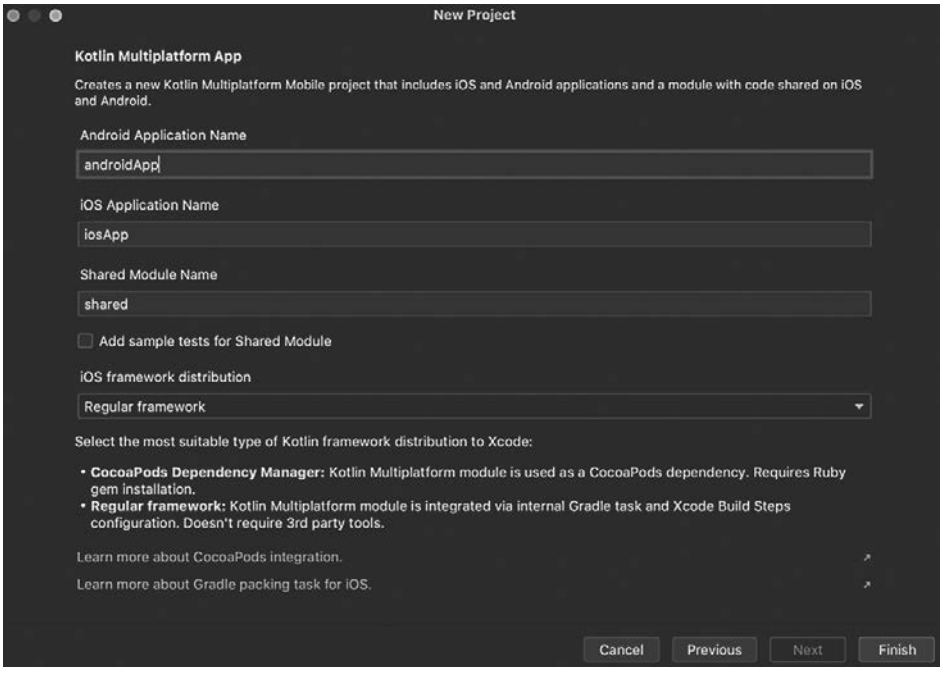

Рис. 1.9. Настройка параметров общего модуля при создании проекта

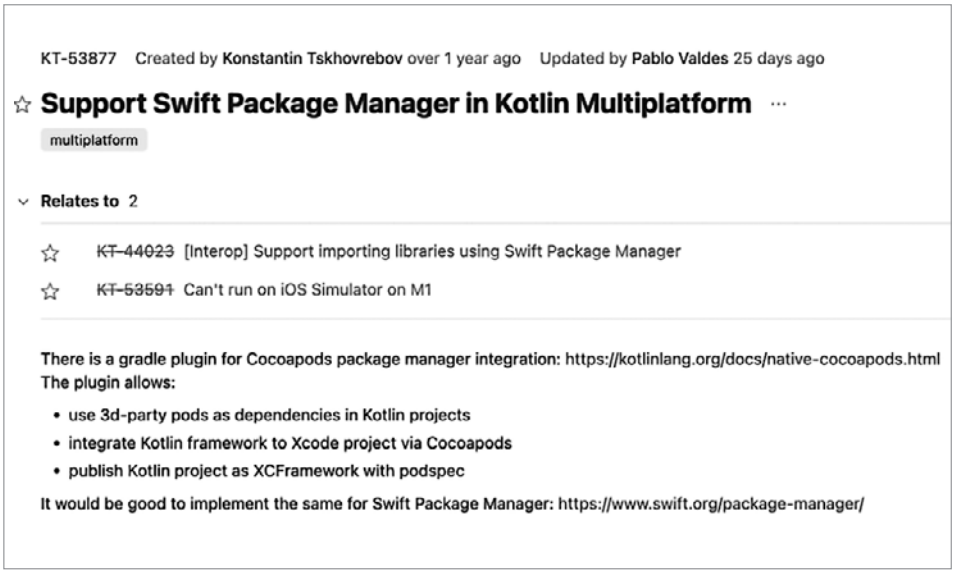

Рис. 1.10. Статус поддержки SPM

Нажмем Finish, и IDE приступит к синхронизации Gradle-зависимостей только что созданного проекта.

Пока происходит первичная настройка проекта, обратим внимание на его структуру. Базовый проект Kotlin Multiplatform состоит из следующих частей (рис. 1.11):

- shared-модуль;
- нативное приложение Android;
- нативное приложение iOS.

iOSApp и androidApp — это обычные нативные приложения iOS и Android, которые используют shared-модуль в качестве библиотеки с бизнес-логикой. Приложение iOS пишется на Swift и может быть реализовано как с помощью UIKit, так и с помощью SwiftUI. Приложение Android также может быть классическим или реализованным с помощью Jetpack Compose. В последних версиях Kotlin Multiplatform базовые нативные приложения реализуются сразу на Jetpack Compose и SwiftUI. Shared-модуль — это и есть, собственно, наш модуль общего кода. Подробнее о его анатомии мы поговорим чуть позже.

Для корректной работы кросс-платформенного приложения важно его правильно настроить. Поэтому сейчас обратим внимание на особенности настройки и управления зависимостями.

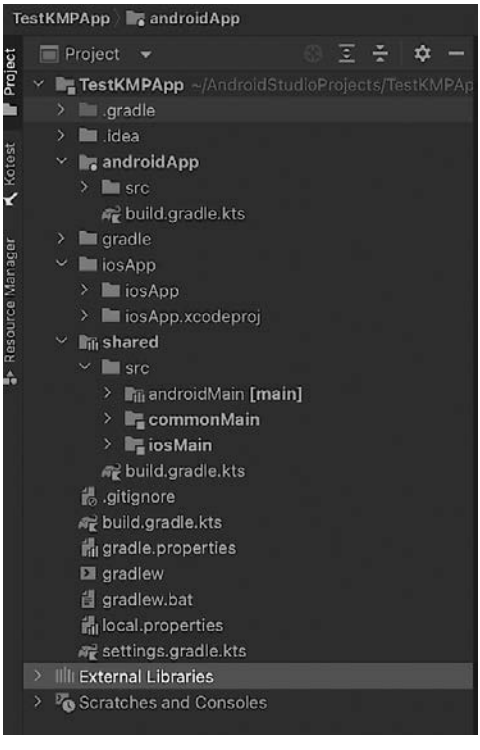

Рис. 1.11. Структура базового приложения KMP

1.2. Настройка проекта и управление зависимостями

В проекте KMP, как и в случае с Android, используется система автоматической сборки Gradle. Необходимые параметры прописываются в соответствующих скриптах `build.gradle.kts`. Такой скрипт есть в каждом модуле приложения. Дополнительно есть скрипт на все приложение для настройки и подключения плагинов (листинг 1.1).

Листинг 1.1. Подключение плагинов в build.gradle.kts

```
plugins {
    //trick: for the same plugin versions in all sub-modules
    alias(libs.plugins.androidApplication).apply(false)
    alias(libs.plugins.androidLibrary).apply(false)
    alias(libs.plugins.kotlinAndroid).apply(false)
    alias(libs.plugins.kotlinMultiplatform).apply(false)
}
```

Рассмотрим внимательнее `build.gradle.kts` shared-модуля. Начинается он с секции подключения плагинов в модуль (листинг 1.2).

Листинг 1.2. Подключение плагинов в shared-модуле

```
plugins {
    alias(libs.plugins.kotlinMultiplatform)
    alias(libs.plugins.androidLibrary)
}
```

По умолчанию подключен плагин Kotlin Multiplatform и плагин для работы Android-таргета как библиотека для Android.

Далее идет секция `kotlin`, в которой описываются все нужные таргеты и подключаются библиотеки для каждого из них (листинг 1.3).

Листинг 1.3. Подключение и настройка таргетов

```
kotlin {
    androidTarget {…}

    listOf(…).forEach {…}

    sourceSets {
        commonMain.dependencies {
            //put your multiplatform dependencies here
        }
        commonTest.dependencies {
            implementation(libs.kotlin.test)
        }
    }
}
```

По умолчанию для проекта указываются таргет Android и таргет(ы) iOS. Для подключения других таргетов (desktop, WASM) потребуется их прописать и настроить вручную.

Для таргета Android по умолчанию указывается поддерживаемая версия JVM (листинг 1.4).

Листинг 1.4. Android-таргет

```
androidTarget {
    compilations.all {
        kotlinOptions {
            jvmTarget = "1.8"
        }
    }
}
```

Таргет Android един для всех устройств и симуляторов. В случае iOS мы рассматриваем таргет под каждую аппаратную архитектуру по отдельности. Вот так выглядит автоматическое подключение библиотеки к iOS-таргетам (листинг 1.5).

Листинг 1.5. Настройка iOS-таргетов

```
listOf(
        iosX64(),
        iosArm64(),
        iosSimulatorArm64()
).forEach {
    it.binaries.framework {
        baseName = "shared"
        isStatic = true
    }
}
```

Чтобы было удобнее работать с кодом, Kotlin Multiplatform создает для нас единый таргет iOSMain:

```
sourceSets {
        iosMain.dependencies {/*...*/}
    }
```

В блоки `dependencies` мы можем добавлять необходимые зависимости для каждого таргета (листинг 1.6).

Листинг 1.6. Блок sourceSets

```
sourceSets {
    iosMain.dependencies {/*...*/}

    androidMain.dependencies { /*...*/ }

    commonMain.dependencies {
        //put your multiplatform dependencies here
    }
}
```

Мультиплатформенные версии библиотек прописываются в блоке для commonMain. Для установки зависимости строка подключения добавляется в нужный блок, после чего нужно нажать кнопку Gradle sync для синхронизации Gradle:

```
implementation("org.jetbrains.kotlinx:kotlinx-coroutines-core:*")
```

Вместо * выберите нужную версию библиотеки.

Также можно использовать специальный файл `gradle-lib.versions.iml`, в котором прописать нужные версии библиотек, разбив их на тематические секции (листинг 1.7).

Листинг 1.7. Регистрация версии библиотеки в каталоге Gradle-приложения

```
[versions]
coroutines = "1.10.1"

[libraries]
```

```
coroutines = {
    module = "org.jetbrains.kotlinx:kotlinx-coroutines-core",
    version.ref = "coroutines"
  }
```

Подключение становится более лаконичным (листинг 1.8). Такой подход позволяет вынести управление версиями из `build.gradle.kts`.

Листинг 1.8. Подключение библиотеки из каталога

```
sourceSets {
            commonMain.dependencies {
                implementation(libs.coroutines)
            }
        }
```

В самом низу `build.gradle.kts` — секция для настройки параметров Android-таргета и нативного приложения (листинг 1.9).

Листинг 1.9. Настройки нативного Android-приложения

```
android {
  namespace = "com.example.testkmpapp"
  compileSdk = 34
  defaultConfig {
    minSdk = 24
  }
  compileOptions {
    sourceCompatibility = JavaVersion.VERSION_1_8
    targetCompatibility = JavaVersion.VERSION_1_8
  }
}
```

Это настройки основного проекта и shared-модуля. Для подключения shared-модуля к нативному Android-приложению необходимо указать путь к нему в зависимостях `build.gradle.kts` основного модуля нативного приложения:

```
dependencies {
    implementation(projects.shared)
}
```

Для iOS-приложения необходимо сделать следующее.

На вкладке Build Settings в Search Path укажем путь для поиска нашего собранного фреймворка:

```
$(SRCROOT)/../shared/build/xcode-frameworks/$(CONFIGURATION)/$(SDK_NAME)
```

Далее в Linking ▸ General в Other linking flags добавим название модуля (в нашем случае shared) и флаги `$(inherited)` и `-frameworks` (рис. 1.12).