Содержание

Об авторах 11

О колофоне 12

Вступление 13

Часть I. ЗАДАЧИ И ОПЕРАЦИИ 17

Примечание о текущем положении дел в сфере разработки мобильных приложений 17

Глава 1. Контроллеры пользовательского интерфейса 19

Задачи 19
Android 20
Как создать начальный контроллер пользовательского интерфейса приложения 20
Как изменить активный контроллер пользовательского интерфейса 22
Основные этапы жизненного цикла контроллера пользовательского интерфейса 27
iOS 30
Как создать начальный контроллер пользовательского интерфейса приложения 30
Как изменить активный контроллер пользовательского интерфейса 34
Основные этапы жизненного цикла контроллера пользовательского интерфейса 38
Что мы узнали 42

Глава 2. Представления 43

Задачи 43
Android 43
Создание нового представления 44
Вложение представлений друг в друга 49
Изменение состояния представлений 50
iOS 50
Создание нового представления 51
Вложение представлений друг в друга 53
С помощью Interface Builder 56
Изменение состояния представлений 57
Изменение позиции 58
Что мы узнали 59

Глава 3. Пользовательские компоненты 60
Задачи 60
Android 60
Как создать свое представление 61
Как использовать свое представление 66
iOS 68
Как создать свое представление 68
Как использовать свое представление 70
Что мы узнали 72

Глава 4. Пользовательский ввод 73
Задачи 73
Android 73
Получение события касания и реакция на него 74
Получение события ввода с клавиатуры и реакция на него 78
Обработка сложных жестов 81
iOS 84
Получение события касания и реакция на него 84
Получение события ввода с клавиатуры и реакция на него 85
Обработка сложных жестов 87
Что мы узнали 89

Глава 5. Передача сообщений 90
Задачи 90
Android 90
Использование обратных вызовов для реакции на действия 91
Передача сообщений подписчикам, заинтересованным в их получении 95
Получение и обработка сообщений 96
iOS 98
Использование обратных вызовов для реакции на действия 98
Передача сообщений подписчикам, заинтересованным в их получении 104
Получение и обработка сообщений 106
Замыкания вместо селекторов 107
Отмена подписки на уведомления 108
Что мы узнали 109

Глава 6. Файлы 111
Задачи 111
Android 111
Определение характеристик файла, таких как размер или дата последнего изменения 112
Чтение и запись данных в файлы 113

Копирование данных из одного файла в другой ... 118
iOS ... 119
Определение характеристик файла, таких как размер или дата последнего изменения ... 119
Чтение и запись данных в файлы ... 122
Копирование данных из одного файла в другой ... 123
Что мы узнали ... 125

Глава 7. Хранение данных ... 126

Задачи ... 126
Android ... 126
Соединение с базой данных ... 127
Создание таблицы или хранимого объекта ... 128
Запись данных в таблицу или хранимый объект ... 129
Чтение данных из таблицы или хранимого объекта ... 130

iOS ... 133
Настройка соединения со слоем хранения данных ... 133
Определение и создание таблицы или хранимого объекта ... 135
Запись хранимых данных в SQLite ... 136
Чтение данных из SQLite ... 137
Что мы узнали ... 138

Глава 8. Конкурентное (многопоточное) выполнение ... 140

Задачи ... 140
Android ... 140
Запуск задачи в фоновом потоке ... 141
Передача результатов из фонового потока в главный ... 144
Завершение потока выполнения ... 145
iOS ... 150
Запуск задачи в фоновом потоке ... 151
Передача результатов из фонового потока в главный ... 152
Что мы узнали ... 153

Глава 9. Сетевые взаимодействия ... 155

Задачи ... 155
Android ... 156
Загрузка текстового файла с удаленного сервера и его вывод ... 156
Создание запроса HTTP POST ... 157
Загрузка двоичного файла ... 159
iOS ... 160
Загрузка текстового файла с удаленного сервера и его вывод ... 161
Создание запроса HTTP POST ... 162
Загрузка двоичного файла ... 166
Что мы узнали ... 171

Глава 10. Обратная связь с пользователем 172
Задачи 172
Android 172
Отображение обратной связи с использованием системных инструментов 172
Snackbar 173
Изменение строки состояния 175
iOS 177
Отображение обратной связи с использованием системных инструментов 177
Изменение строки состояния 179
Что мы узнали 180

Глава 11. Предпочтения пользователя 182
Задачи 182
Android 182
Сохранение предпочтений пользователя 183
Чтение предпочтений пользователя 184
Работа с предпочтениями в многопользовательских приложениях 184
iOS 185
Сохранение предпочтений пользователя 185
Чтение предпочтений пользователя 188
Работа с предпочтениями в многопользовательских приложениях 189
Что мы узнали 190

Глава 12. Сериализация и транспорты 192
Задачи 192
Android 192
Сериализация и десериализация экземпляров объектов 192
iOS 200
Сериализация и десериализация экземпляров объектов 200
Дополнительные замечания для iOS 205
Что мы узнали 206

Глава 13. Расширения 207
Задачи 207
Android 207
Добавление новых возможностей в существующие API 207
iOS 209
Добавление новых возможностей в существующие API 209
Что мы узнали 212

Глава 14. Тестирование 213
Задачи 213
Android 213

Как писать и запускать модульные тесты ... 217
Как писать и запускать интеграционные тесты ... 220
iOS ... 222
Как писать и запускать модульные тесты ... 222
Что мы узнали ... 225

Часть II. ПРИМЕР ПРИЛОЖЕНИЯ ... 226

Глава 15. Добро пожаловать и настройка окружения ... 227

Сравнение нативных и кросс-платформенных инструментов разработки мобильных приложений ... 227
Веб-разработка ... 228
Другие подходы ... 228
Настройка окружения ... 229
Настройка окружения разработки для Android ... 229
Настройка окружения разработки для iOS ... 235
Что мы узнали ... 236

Глава 16. Создание приложения ... 237

Создание нового проекта ... 237
Android Studio ... 238
Xcode ... 242
Архитектура приложения ... 245
Создание первого экрана ... 246
Android ... 247
iOS ... 249
Что мы узнали ... 254

Глава 17. Вывод списка с данными ... 255

Оформление представлений ... 255
Android ... 255
iOS ... 265
Добавление кнопки ... 270
iOS ... 271
Списки, списки и еще раз списки! ... 271
Добавление нового представления каталога ... 272
Подключение кнопки ... 273
Книги ... 274
Заполнение представления списка ... 278
Android ... 278
iOS ... 285
Что мы узнали ... 288

Глава 18. Моделирование каталога библиотеки ... 290

Динамические данные в представлениях списков ... 290
Android ... 291

iOS 294
Пришло время вернуть объекты модели в реальность 298
JSON для одного, JSON для всего 298
Переключение слоя данных на использование JSON 300
Android 300
iOS 306
Что мы узнали 308

Глава 19. Сохранность данных 309

Детализация информации о книгах 309
Android 309
iOS 314
Сохранение книг для последующего использования 318
Android 319
iOS 320
Запись книг в хранилище 321
Android 322
iOS 331
Сохранение книг в закладках 339
Android 340
Что мы узнали 341

Глава 20. Сетевые операции в приложении 342

Поиск в сети 342
Android 345
iOS 347
Создание службы поиска 350
Установка Node и Express 350
Файл JSON с местоположениями библиотек 351
Вызов службы 352
Android 352
iOS 356
Что мы узнали 365

Предметный указатель 366

Об авторах

Шон Льюис (Shaun Lewis) – бывший ведущий разработчик программного обеспечения для iOS, а ныне руководитель подразделения Mobile Engineering в издательстве O'Reilly Media. Первая книга «How to Build a Website in a Weekend», которую он прочитал, коренным образом изменила устремления 15-летнего юноши. Теперь он имеет за плечами 12-летний опыт профессиональной разработки приложений для iPhone, начав заниматься этим, еще когда iOS еще называлась iPhone OS. Сотрудничал с рядом компаний из списка Fortune 500 и иногда выступает на встречах разработчиков продуктов Apple. Шон живет в Огайо со своей женой, двумя детьми и полным ящиком старых смартфонов.

Майк Данн (Mike Dunn) – ведущий инженер по мобильным технологиям и технологиям Android в издательстве O'Reilly Media. Майк является признанным членом сообщества AOSP и активно участвует в развитии экосистемы открытого исходного кода Android, занимаясь в том числе развитием и обслуживанием популярной библиотеки TileView. Внес свой вклад в разработку библиотеки Google Closure и в поддержку ExoPlayer – мультимедийного проигрывателя Google следующего поколения. Майк профессионально занимается программированием на протяжении многих лет и продолжает обучаться информатике в рамках магистратуры в технологическом институте штата Джорджия.

О колофоне

На обложке книги «Нативная разработка мобильных приложений» изображена норикийская лошадь (*Equus ferus caballus*). Название «норикийская» происходит от названия австрийской провинции *Норикум* в Римской империи. Порода выведена в Австрии, где она также известна как пинцгауская лошадь.

Норикийских лошадей разводили в австрийских Альпах для перевозки товаров по всей Европе. Это сильные, терпеливые и послушные животные. Мускулистые и крепкие, они весят почти тонну (в среднем 1550 фунтов, или 700 кг). Их короткие ноги надежно удерживают такой вес на пересеченной местности. Норикийские лошади могут иметь черный, лавровый или каштановый окрас.

В настоящее время, когда благодаря индустриализации спрос на ломовых лошадей снизился, норикийские лошади используются в основном для верховой езды. Они прекрасно себя чувствуют в горных условиях на высотах до двух тысяч метров.

Многие животные, изображаемые на обложках книг издательства O'Reilly, находятся под угрозой вымирания; все они очень важны для нашего мира. В наши дни норикийская лошадь считается «породой с ограниченным распространением». Чтобы узнать, чем вы можете помочь, посетите сайт animals.oreilly.com.

Иллюстрацию для обложки нарисовала Карен Монтгомери (Karen Montgomery) на основе старой черно-белой гравюры (источник неизвестен).

Вступление

Почему мы написали эту книгу

Эта книга является практическим перекрестным справочником разработчика приложений для iOS и Android. Под «нативной разработкой» мы подразумеваем использование оригинальных инструментов, предлагаемых каждой платформой – Swift и Cocoa для iOS и Java или Kotlin с набором инструментов для разработки программного обеспечения (Software Development Kit, SDK) от Android Open Source Project (AOSP) для Android.

Написать эту книгу нас побудила банальная потребность в таком справочнике. Оба автора имеют опыт работы с обеими платформами, но специализируются на одной. Нередко члены нашей команды (включая нас самих), занимаясь какой-то проблемой на одной платформе, находили ее решение на этой платформе, а затем делились этим решением с членами команды, работающими на другой платформе.

Часто это были задачи, общие для обеих платформ, такие как чтение или запись в базу данных либо в файлы, создание сетевого подключения или отображение обратной связи в привычном для пользователей виде. Начав писать код и документировать эти задачи на обеих платформах, мы быстро заметили, что подавляющее большинство решений относится к этой категории и перекрестный справочник по этим решениям мог бы очень пригодиться командам со смешанным составом, начинающим переход на другую платформу и даже разработчикам, желающим начать изучение обеих платформ сразу.

Мы надеемся, что эта книга будет полезна читателям в таком качестве и поможет им в решении типичных задач, возникающих при разработке мобильных приложений.

Имея целую команду разработчиков мобильных приложений с богатым опытом разработки для двух платформ, мы относительно легко преодолевали затруднения. Тем не менее мы долго вынашивали идею перекрестного справочника, потому что до настоящего времени на рынке не было книг, охватывающих эту тему с необходимой широтой. Мы легко можем представить разочарование разработчика, работающего в одиночку и столкнувшегося с такой же ситуацией; эта книга поможет освоить базовые подходы к решению большинства типичных задач разработки приложений для обеих платформ. В каждой главе, посвященной определенной категории задач, описывается весь процесс их решения в Android и iOS с использованием простых и понятных примеров кода. По нашей оценке, эти примеры охватывают 80 % базовых знаний, необходимых, чтобы приступить к разработке приложений; конечно же, мы полагаем, что читатель не остановится на достигнутом и продолжит расширять свои знания и читать документацию, чтобы самостоятельно найти решение для задач, которые мы намеренно обошли стороной. Мы также включили в справочник пошаговый пример разработки приложения, демонстрирующий реализацию практически всего, что может потребоваться современному при-

ложению, с использованием информации из предшествующего перекрестного справочника.

Поскольку все примеры кода для Android представлены на двух языках – Java и Kotlin, – эта книга имеет приятный побочный эффект: она может служить перекрестным справочником не только между iOS и AOSP, но и между Java и Kotlin для разработчиков на Android.

Примеры кода не являются псевдокодом; они написаны на конкретных языках программирования и должны компилироваться и действовать, как описано.

Кому адресована эта книга

Эта книга предназначена для всех программистов, работающих только с какой-то одной платформой или с обеими, либо знакомых с одной, но желающих освоить другую. Мы предполагаем у читателя наличие хотя бы поверхностного знания некоторого языка программирования. Вам не обязательно быть экспертом в Java или Swift, но некоторый опыт программирования пользовательского интерфейса не будет лишним.

Возможно, вам придется обратиться к официальной документации с описанием Objective-C, Swift, Java или Kotlin, чтобы понять некоторые основные особенности языков, которые упоминаются в этой книге.

Программистам, имеющим опыт работы с одной из платформ (iOS или Android), будет легко усвоить предоставленную информацию, потому что почти каждому примеру кода для одной платформы соответствует функционально эквивалентный пример для другой платформы.

Организация книги

Эта книга разделена на две части. В первой части представлены решения распространенных задач, которые приходится решать на любой платформе, таких как запись файла в локальное хранилище или создание HTTP-запроса. Вторая часть описывает процесс создания приложения на каждой платформе с использованием приемов из первой части.

Соглашения

В этой книге используются следующие соглашения по оформлению:

Курсив
: Используется для обозначения новых терминов, адресов URL и электронной почты, имен файлов и расширений имен файлов.

`Моноширинный шрифт`
: Применяется для оформления листингов программ и программных элементов внутри обычного текста, таких как имена переменных и функций, баз данных, типов данных, переменных окружения, инструкций и ключевых слов.

Моноширинный жирный
Обозначает команды или другой текст, который должен вводиться пользователем.

Моноширинный курсив
Обозначает текст, который должен замещаться фактическими значениями, вводимыми пользователем или определяемыми из контекста.


Так выделяются советы и предложения.


Так обозначаются примечания общего характера.


Так обозначаются предупреждения и предостережения.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте www.dmkpress.com, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте http://dmkpress.com/authors/publish_book/ или напишите в издательство: dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии данной книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Скачивание исходного кода

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и O'Reilly очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу dmkpress@gmail.com со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Благодарности

Мы хотели бы поблагодарить своих рецензентов: Пэриса Баттфилд-Аддисона (Paris Buttfield-Addison), Яна Дарвина (Ian Darwyn), Дона Гриффитса (Dawn Griffiths), Бена Кригера (Ben Kreeger), Пьера-Оливье Лоуренса (Pierre-Olivier Laurence), Шейна Степлса (Shane Staples) и Субатра Танабалан (Subathra Thanabalan).

Часть I

ЗАДАЧИ И ОПЕРАЦИИ

В этой части мы предоставим основополагающий код для выполнения различных задач, типичных для мобильных приложений. К их числу мы относим отображение пользовательского интерфейса, передачу данных, отправку и получение событий, выполнение сетевых запросов и обработку ответов, доступ к файловой системе и управление ею, а также чтение или запись в постоянные хранилища, такие как базы данных или файлы с настройками.

Примечание о текущем положении дел в сфере разработки мобильных приложений

На момент написания этой книги сфера разработки мобильных приложений для Android и iOS находилась в крайне неустойчивом состоянии, была сильно фрагментирована и переполнена противоречиями. Добавлялись новые библиотеки, предназначенные для замены существующих API, но не все они являлись результатом обсуждения и согласования членами сообщества разработчиков. Кроме того, почти каждый новый API был значительно сложнее своего предшественника. Выбирая информацию для включения в будущую книгу, включая основополагающие библиотеки и API, мы решили, что должны постараться оказать максимальную помощь как можно большему числу людей и проектов. В подавляющем большинстве случаев разработчики реализуют одни и те же особенности и адаптируют имеющиеся, а не занимаются созданием чего-то принципиально нового (с чистого листа). Учитывая это, почти во всех случаях мы по умолчанию используем существующие и проверенные временем библиотеки и приемы и стараемся избегать применения новых, еще не устоявшихся библиотек, появившихся относительно недавно (мы произвольно выбрали порог около года).

Кроме того, мы обнаружили, что многие новые API, пришедшие на смену прежним, уводят инфраструктуру от шаблонов, применяемых в существующих технологиях. Например, новый набор компонентов `Navigation` в Android можно использовать для управления пользовательским интерфейсом с применением тех же базовых подходов, которые представлены в этой книге (экземпляры `Fragment`), но в совершенно иной манере. Учитывая объем необходимой инфраструктуры, мы решили, что включение описания этих дополнительных шаблонов и поясняющего кода, помимо проверенных решений, которые, как мы

считаем, представляют реальность разработки для Android на данный момент, принесет больше вреда, чем пользы. Другим примером является технология баз данных: совсем недавно компания Google рекомендовала разработчикам для Android использовать свою библиотеку Room, тогда как в стандартные библиотеки AOSP включены средства для работы с SQLite. Никто не утверждает, что SQLite не имеет ограничений, и мы допускаем, что Room предлагает более современный подход, но одна из наших основных целей состояла в том, чтобы предоставить сведения, опираясь на которые, любой, знакомый с программированием в целом, смог бы быстро приступить к продуктивной работе, к тому же SQL является, пожалуй, одной из самых распространенных и зрелых технологий. Room пока не может похвастаться зрелостью; поэтому в своих примерах хранения данных мы используем SQLite. Если вы решите использовать Room, в этом нет ничего плохого! Мы советуем не упускать из виду новые технологии и рекомендации, и мы сами постараемся напоминать о них, когда это будет уместно, но не удивляйтесь, что мы используем `FragmentManager` со списком компонентов, необходимых для применения `Navigation` API, – это обдуманное решение, которое мы считаем правильным *на данный момент*.

Точно так же, и снова в интересах использования любых существующих знаний о предметной области, которыми вы, уважаемый читатель, обладаете, мы можем использовать менее эффективные, но более понятные или распространенные шаблоны либо методы. Например, для чтения из потока данных в Android мы часто используем `InputStream.read` без буферизации. Даже притом что во многих случаях буферизация является уместной, отказ от ее применения позволил нам не только сократить и упростить примеры кода, но и избавил от необходимости объяснять, как работают буферы (с обеих сторон, на входе и на выходе), какой размер буфера является наиболее подходящим в тех или иных обстоятельствах и почему эффективнее предварительно выделить один буфер, чем создавать новый для каждой операции чтения или записи. Потоковый буфер – сам по себе довольно простое понятие, но *правильно и полностью* объяснить его работу – нетривиальная задача. По аналогичной причине современные версии Java предлагают конструкцию try-with-resources для операций с экземплярами `Closable`. В этом конкретном случае кто-то, знакомый с конструкцией try-catch в другом языке (например, JavaScript), сразу распознает стандартный синтаксис и сможет сосредоточить свое внимание на описываемой нами задаче, и ему не придется приостанавливаться, чтобы познакомиться с альтернативным синтаксисом try-with-resources и с трудом продираться через еще один или два абзаца, которые никак не способствуют его главной цели. Конечно, в некоторых случаях try-with-resources – это более чем уместный прием, и мы всем советуем стремиться узнать как можно больше о каждом языке и фреймворке, но эта книга призвана помочь программистам начать *программирование*, а не освоить каждую технологию, которую мы будем затрагивать здесь.

Спасибо всем, кто терпеливо выслушивал нас, и за все высказанные мнения, которые привели к появлению этого примечания, мы искренне ценим вашу вдумчивость.

Глава 1

Контроллеры пользовательского интерфейса

Контроллеры пользовательского интерфейса (User Interface, UI) играют роль связующего звена между пользовательским интерфейсом и бизнес-логикой приложения, которая управляет этим пользовательским интерфейсом или получает информацию от него.

Если провести аналогию между приложением и пьесой Шекспира, поставленной в каком-нибудь театре Старого Света, тогда контроллер пользовательского интерфейса можно сравнить с помощником режиссера. Он выводит актеров на сцену, получает команды от режиссера и помогает в смене декораций.

Каждый раз, когда в приложении потребуется отобразить изображение, список или фрагмент текста, вам потребуется пользовательский интерфейс. Представление пользовательского интерфейса – отображение на экране – обычно определяется макетом (часто оформленным в виде разметки XML или HTML); контроллер пользовательского интерфейса действует как мост между командами ввода, запросами к базе данных, запросами межпроцессных взаимодействий (IPC), сообщениями и многим другим. В каком-то смысле это сердце любого приложения.

Все это жонглирование требует невероятно сложной серии событий с применением множества технологий, основанных друг на друге и действующих согласованно. К счастью, обе платформы, Android и iOS, предлагают ряд общих инструментов и абстракций для управления этим тяжелым процессом. Давайте познакомимся с некоторыми основными задачами в данной области, которые являются центральными для обеих платформ.

Задачи

В этой главе вы узнаете:

1) как создать начальный контроллер пользовательского интерфейса приложения;
2) как изменить активный контроллер пользовательского интерфейса;

3) основные этапы жизненного цикла контроллера пользовательского интерфейса.

Android

Менее чем за год до того, как мы взялись за эту книгу, компания Google рекомендовала для навигации в приложении использовать один экземпляр `Activity` и экземпляры класса `Fragment` в нем для реализации операций и управления представлениями. Для управления взаимодействиями между фрагментами и историей отображения предлагалось использовать новый компонент `Navigation`, выпущенный в пакете Jetpack.

Обратите внимание, что эта рекомендация идет вразрез с практиками, предлагавшимися с момента появления Android более десяти лет назад, когда `Activity` рекомендовалось использовать для любой «деятельности» (примерным аналогом «деятельности», или «активности», является «экран» либо отдельная веб-страница), а использование вложенных экземпляров `Fragment` то приветствовалось, то не приветствовалось. Фактически даже в настоящее время разработчики для Android начинают главу об `Activity` со следующих слов:

> Деятельность – это узкоспециализированная экранная форма, позволяющая пользователю выполнить определенную операцию.

В пользу обеих сторон можно привести веские аргументы, но, поскольку разработчиком Android является Google, мы считаем, что в будущем должны принять ее рекомендацию. Тем не менее существует *множество* давнишних приложений, разработчики которых не использовали этот шаблон и не планируют менять код, написанный за несколько лет работы, чтобы внедрить его. Мы не будем принимать чью-либо сторону и покажем основы обоих подходов. В случае сомнений мы будем использовать преобладающий существующий подход – запускать новые экземпляры `Activity`, передавать данные в виде экземпляров `Bundle` и управлять модульным контентом с помощью экземпляров `Fragment` и методов контроллера `Activity`, стараясь избегать использования более нового компонента `Navigation` архитектуры навигации и его родственников.

Как создать начальный контроллер пользовательского интерфейса приложения

Давайте сразу приступим к делу. Когда приложение запустится, оно выполнит некоторую логику инициализации и создаст «фон окна» (обычно сплошной цвет, в зависимости от вашего экрана, который можно заменить любым допустимым экземпляром `Drawable`). Эта работа выполняется в главном потоке и не может быть прервана или приостановлена – она просто будет выполнена. Обратите внимание, что если для приложения реализован свой класс `Application`, в этот момент будет вызван его метод `onCreate`. И снова, важно помнить об этом, вызов произойдет в главном потоке выполнения (его еще называют потоком пользовательского интерфейса), поэтому все остальные операции бу-

дут отложены до окончания его выполнения. Однако в настоящее время есть возможность организовать асинхронное выполнение операций в фоновых потоках.

По завершении инициализации приложение запустит один экземпляр класса `Activity`, который вы определили в манифесте приложения со значением `android.intent.category.LAUNCHER` в его элементе `category`. Описание этого экземпляра `Activity` должно также включать элемент `action` со значением `android.intent.action.MAIN`, определяющим любую из точек входа в ваше приложение (например, через значок запуска, глубокую ссылку, общесистемные широковещательные сообщения и т. д.).

Вы должны указать только каноническое имя класса, а создание экземпляров и их настройка выполняются автоматически (то есть этот процесс совершенно непрозрачен для нас как разработчиков или пользователей).

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

В полном манифесте этот фрагмент выглядит так:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="org.oreilly.nmd"
          xmlns:android="http://schemas.android.com/apk/res/android">

  <application
      android:allowBackup="false"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:roundIcon="@mipmap/ic_launcher_round"
      android:supportsRtl="true"
      android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".BrowseContentActivity" />
    <activity android:name=".BookDetailActivity" />
    <activity android:name=".SearchResultsActivity" />
  </application>

</manifest>
```

Обратите внимание, что любой экземпляр `Activity`, который предполагается использовать в приложении, должен быть зарегистрирован в *ApplicationManifest.xml* как дочерний узел узла `application` (`manifest` ⇒ `application` ⇒ все элементы `activity`). Убедитесь в этом, заглянув в блок кода выше, как прочтете это примечание.

```
<activity android:name=".MyActivity" />
```

Считается, что в процессе взаимодействия с приложением Android пользователь всегда находится в Activity (исключение составляют удаленные операции, такие как взаимодействие строки состояния с Service, но это очень незначительное исключение для данной главы). У вас никогда не будет пригодного для использования элемента пользовательского интерфейса, не входящего в состав какого-то экземпляра Activity (единственным исключением является класс RemoteViews – небольшое подмножество простых классов View, – доступный в окнах уведомлений).

Обратите внимание, что экземпляры Activity нельзя вкладывать друг в друга. Вообще говоря, один экземпляр Activity занимает сразу весь экран (или, по крайней мере, часть, делегированную приложению).

Как уже упоминалось, вы не должны создавать новые экземпляры Activity; от вас требуется только указать класс Activity, который следует запустить. За кулисами платформа Android создаст нужный экземпляр и выполнит все подготовительные операции, прежде чем отобразить его. Кроме того, эта операция выполняется *асинхронно*, и система сама решает, когда запустить новый экземпляр Activity.

Это особенно важно, потому что в файле манифеста разным экземплярам Activity могут быть назначены разные режимы запуска. Конкретный режим запуска может позволить одновременно существовать любому количеству экземпляров класса Activity. Например, вы можете разрешить пользователю иметь любое количество экземпляров ComposeEmailActivity в одном стеке задач, и при этом ограничить количество экземпляров других видов Activity, например разрешить только один экземпляр LoginActivity, что может привести к перемещению последнего использовавшегося LoginActivity на вершину стека задач или к уничтожению всего, что находится между текущим экземпляром Activity и последним использовавшимся LoginActivity, в зависимости от режима запуска. Мы не будем подробно останавливаться на режимах запуска, но вы обязательно загляните в документацию для разработчиков и познакомьтесь с этим вопросом поближе, если, конечно, вам это интересно.

Итак, мы благополучно запустили Activity, но почему на экране ничего не появляется? Потому что Activity – это класс уровня контроллера, а не видимое представление. Чтобы отобразить элементы на экране, нужен как минимум один экземпляр View или несколько (в виде дочерних элементов в экземпляре View, используемом в качестве корня в Activity). Обычно это делается с помощью метода setContentView и передачи ресурса макета в формате XML. Подробнее об этом мы расскажем в главе 2.

Как изменить активный контроллер пользовательского интерфейса

После того как начальный («запускающий») экземпляр Activity отобразится, появляется возможность запустить любой другой экземпляр Activity, вызвав метод startActivity(Intent intent) любого экземпляра Context (класс Activity наследует Context, поэтому он связан с ним отношением «является» – экземпляр Activity *является* экземпляром Context). Экземпляр Intent, в свою очередь, тре-

бует передать экземпляр Context в первом параметре и ссылку на запускаемый класс Activity:

Java

```
// предполагается, что запускающий Activity находится в области видимости
Intent intent = new Intent(this, AnotherActivity.class);
startActivity(intent);

// если он находится вне области видимости, но имеется доступ к объекту
// Context, можно использовать такой код...
// предполагается, что переменная "context" содержит ссылку на объект Context.
Intent intent = new Intent(context, AnotherActivity.class);
context.startActivity(intent);
```

Kotlin

```
// предполагается, что запускающий Activity находится в области видимости
val intent = Intent(this, AnotherActivity::class.java)
startActivity(intent)

// если он находится вне области видимости, но имеется доступ к объекту
// Context, можно использовать такой код...
// предполагается, что переменная "context" содержит ссылку на объект Context.
val intent = Intent(context, AnotherActivity::class.java)
context.startActivity(intent)
```

Важно понимать, что созданием, инициализацией и настройкой экземпляров Activity, которые вы покажете своему пользователю, будет заниматься система – их нельзя создать с помощью ключевого слова new, настроить или иным образом изменить при запуске. Мы передаем системе экземпляр Intent, который определяет, какой класс Activity мы хотим использовать, а система делает все остальное. По этой причине нет никакой возможности изменять свойства и вызывать методы экземпляра Activity непосредственно (с использованием стандартной библиотеки) во время запуска.

Но коль скоро нельзя изменять свойства и вызывать методы экземпляра Activity непосредственно в процессе запуска, как тогда передать ему информацию? Во многих фреймворках пользовательского интерфейса есть возможность создать новый экземпляр класса контроллера представления, записать в него некоторые данные и дать ему возможность отобразить их.

В Android ваши возможности весьма ограничены. Классический подход заключается в привязке простых значений к объекту Intent, например так:

Java

```
// предполагается, что запускающий Activity находится в области видимости
Intent intent = new Intent(this, AnotherActivity.class);
intent.putExtra("id", 10);
startActivity(intent);
```

Kotlin

```
// предполагается, что запускающий Activity находится в области видимости
Intent intent = Intent(this, AnotherActivity::class.java);
intent.putExtra("id", 10)
startActivity(intent)
```