Оглавление

Об авторе

Фрэнк Заметти – автор ряда популярных технических книг, был программистом около 40 лет, 25 из которых занимался этим профессионально, надо же ему было что-то кушать. В те времена на его визитке значилось *архитектор*, хотя он продолжал писать тупой код и каждый день крутился как мог. Фрэнк – первоклассный гик: если он не заставляет свой компьютер выполнять приказы (скорее всего, дьявольские), то занят просмотром, чтением или написанием научной фантастики, моделированием рельсотрона, катушки Теслы или любой другой штуковины, способной прикончить его в любой момент; он любит без причины цитировать *«Вавилон 5»*, *«Властелин колец»*, *«Хроники Риддика»* или *«Настоящих гениев»*, а также играть в компьютерные игры. Еще Фрэнк – рок-музыкант (клавишник) и заядлый любитель пиццы и других углеводов. У него есть жена, собака и несколько детей. Если подвести итог его крутости, то он тот, кто всегда готов воскликнуть «С тобой мой меч!» (ну да, обычно гики цитируют *«Властелин колец»* без очевидной причины).

О техническом рецензенте (обозревателе)

Герман ван Росмален работает разработчиком и архитектором программного обеспечения для De Nederlandsche Bank NV, центрального банка в Нидерландах. За его плечами более 30 лет опыта разработки приложений на разных языках программирования. Герман был вовлечен в создание приложений для мейнфреймов, десктопов, серверов, веб-браузеров и смартфонов. Последние 4 года он занимается в основном разработкой на .NET/C# и Angular после 15 лет работы с Java.

Герман живет в маленьком городке Пейнаккер в Нидерландах со своей женой Лизбет и детьми Барбарой, Леони и Рамоном. Наравне с разработкой софта в свободное время Фрэнк тренирует женскую футбольную команду на протяжении последних 10 лет.

И конечно же, он болеет за Фейеноорд (футбольный клуб из города Роттердам в Нидерландах, который считается одним из ведущих клубов страны)!

О переводе

Данная книга была переведена специалистами компании Binwell, а также действующими преподавателями Binwell University. Надеемся, что наши книги и переводы позволят вам легко освоить новые технологии, а также построить успешную карьеру в сфере информационных технологий.

Над переводом книги «Flutter на практике» работали:

Артем Тищенко, переводчик – руководитель направления Mobile в компании Binwell, специалист в разработке нативных (iOS Swift/Objective C), а также кроссплатформенных (Xamarin и Flutter) мобильных приложений. Ментор Binwell University, соавтор ряда популярных статей для Microsoft Developer Blogs и Хабрахабры.

Вячеслав Черников, редактор перевода – руководитель разработки в компании Binwell, руководитель Binwell University. Работает в сфере Mobile и разработки ПО с 2005 года. Создавал приложения и игры для iOS, Android, Symbian, Windows Mobile, Meego, Linux и Windows UWP. Имеет богатый опыт разработки нативных (iOS Swift/Objective C) и кроссплатформенных (Xamarin, Qt, PhoneGap/HTML5, Unity) мобильных приложений. Автор книги «Разработка мобильных приложений на C# для iOS и Android» (ДМК Пресс, 2020), а также популярных статей для Хакера, Хабрахабры, Microsoft Developer Blogs, спикер, преподаватель и немного [безумный] ученый.

Также выражаем благодарность **Александру Рыжкову** и **Сергею Селютину** за помощь с корректировками.

Благодарности

Если вы никогда не занимались написанием книги, то я открою вам секрет: написание самой книги – это лишь малая часть большой работы над ней. Иногда, мне кажется, наименьшая часть!

Поэтому я хочу поблагодарить всех, кто усердно работал и помогал с этим проектом (не важно, лично или с редактурой), включая Нэнси Чен, Луиса Корригана, Джеймса Маркхэма, Германа ван Росмалена, Вэлмода Спара и Даниша Кумара. Если вашего имени нет в списке, хотя оно должно здесь быть, я приношу свои искренние извинения и благодарю вас.

Также я хотел бы поблагодарить Ларса Бака и Каспера Ланга за создание Dart, довольно элегантного и очень приятного в использовании языка программирования, лежащего в основе Flutter. Говорю как человек, который создал свой собственный язык и набор инструментов для него много лет назад, я очень-очень ценю то, что вы сделали, ребята. Честь вам и слава!

Работа над книгой по Flutter требует от меня благодарности почти всей команде разработчиков этого фреймворка. Я занимаюсь мобильной разработкой около 20 лет (посмотрите на etherient.com, страницу Products, а конкретно Eliminator – игра, которую я выпустил в 2001 году для платформы Pocket PC; я верю, что это было мое первое мобильное приложение, по крайней мере первая удачная попытка), и тогда, насколько я могу судить, я использовал достаточно много утилит, фреймворков и библиотек. Учитывая весь этот опыт, я могу с уверенностью сказать, что Flutter даже с первой версии был на голову выше всех.

Это поразительно, сколько команда Flutter смогла сделать за такой относительно короткий период времени, и без их тяжелой работы я бы, очевидно, не написал эту книгу! Я с нетерпением жду возможности все больше и больше использовать Flutter, а также мне очень интересно, что будет с Flutter дальше!

Введение

Создание кроссплатформенных приложений, которые выглядят и работают как нативные, – сложная задача даже после многих лет работы над ее решением. Вы можете писать отдельные программы для каждой платформы и пытаться сделать их дизайн максимально похожим. Но фактически это значит написать одно приложение несколько раз. Заказчики, как правило, не готовы за такое платить!

Может, взять HTML-страницу и использовать один и тот же код для всех платформ? Но тогда вы можете остаться в дураках с точки зрения возможностей самого устройства, не говоря уже о том, что производительность часто находится на низком уровне (существуют способы минимизирования проблем, но они никогда не исчезнут полностью).

Я занимаюсь этим уже второе десятилетие (серьезно!), поэтому я видел такое много раз. И если я замечу на горизонте образ единорога, то буду сомневаться до конца. Однако если, подойдя поближе, окажется, что единорог действительно реален?

Итак, я представляю вам единорога, который на самом деле существует, – Flutter!

Благодаря талантливым инженерам из Google Flutter – это платформа, позволяющая писать (более или менее) кроссплатформенный код, который одинаково работает на Android и iOS, при этом обеспечивая производительность, идентичную нативным приложениям. Flutter, созданный с использованием современных инструментов и методов разработки, открывает программистам мир мобильной разработки, в котором, даже не побоюсь сказать, *весело* работать!

В этой книге вы познакомитесь с Flutter, создав два реальных приложения вместо надуманных примеров, предназначенных лишь для демонстрации технологии. На этом пути вы узнаете много реальных тонкостей, включая проблемы, с которыми я столкнулся, и решения, которые я нашел. При этом вы получите практический опыт реального использования Flutter, который подготовит вас к созданию собственных приложений в будущем.

Вы также узнаете то, как создавать серверные приложения на Node.js и Web Sockets. Эти бонусы являются полезным дополнением к описанию самого фреймворка Flutter и языка Dart.

Кроме того, вы сможете создать дополнительное третье приложение, которое разительно отличается от первых двух, – игру! Да, мы вместе создадим игру на Flutter, чтобы рассмотреть такие возможности, которые редко встречаются в настоящих приложениях, но дают вам взглянуть на фреймворк с разных сторон и получить максимум опыта. Эта игра может быть не совсем «практичной», но игры всегда увлекательны, а немного веселья никому не повредит!

Дочитав до конца, вы получите представление о том, что такое Flutter, и у вас будет отличная возможность создать свое новое крутое приложение на его основе.

Если вы были компьютерным энтузиастом в 80-х годах, то наверняка помните, каково это было – набивать на своем компьютере машинный код из журнальной статьи в 20 страниц мелким шрифтом, чтобы сыграть в игру или запустить приложение для сверки ваших доходов и расходов (да, мы действительно это делали – были даже радиостанции, транслировавшие исходники, которые затем компилировались в готовое приложение с помощью специальной утилиты, подобно тому как сейчас кодируется звук для передачи по телефонной линии).

Итак, прежде чем начать, я предлагаю вам зайти на веб-сайт Apress, найти эту книгу и скачать себе исходные коды примеров. Это позволит обойтись без стирания пальцев в кровь, перепечатывая листинги из книги!

Не забывайте, лучший способ чему-то научиться – это делать, поэтому обязательно скачайте и измените примеры под себя, увидев своими глазами, на что эти изменения повлияют. После прочтения каждой главы, связанной с приложением, заходите в исходные коды и пробуйте добавить одну или две свои функции, и я даже дам вам пару советов. Думаю, вскоре вы поймете, что благодаря возможностям Flutter небольшие изменения могут существенно повлиять на то, что появляется на экране.

Итак, приготовьтесь к приятной и информативной поездке в мир Flutter!

Я надеюсь, что вам понравится эта книга и вы многому научитесь. Это моя главная цель! Так что перекусите, сядьте в кресле поудобнее, подготовьте ноутбук и приступайте. Вас ждут приключения! (Да, я прекрасно понимаю, как банально это звучит.)

Исходные коды примеров вы можете найти в репозитории на GitHub: `https://github.com/Apress/practical-flutter`.

ГЛАВА 1.

FLUTTER: ПЛАВНОЕ ПОГРУЖЕНИЕ

Поехали!

Если вы спросите десять разных разработчиков мобильных приложений, как они их разрабатывают для Android и iOS, то, скорее всего, получите десять разных ответов. Но это ненадолго, благодаря дебютанту на этой сцене – Flutter.

В первой главе мы рассмотрим разработку для мобильных устройств, то, как Flutter вписывается в эту картину и, возможно, полностью ее меняет. Мы начнем с ним работать, получим базовое представление о языке и фреймворке, а также подготовим почву для создания реальных приложений.

Итак, давайте сразу поговорим о том, что же такое мобильная разработка.

Медитации над бездной

Разработка софта – это непростая задача!

Я не хочу утомлять вас историей, но факт в том, что я начал, так или иначе, программировать с 7 лет, а это означает, что я занимаюсь этим почти 40 лет (около 25 из них профессионально). Я много повидал и много сделал, но главное, что я понял: разработка софта – это непростая задача. Конечно, некоторые отдельные задачи и проекты могут быть простыми, но в целом это довольно сложная работа, которой мы занимаемся!

И это мы еще не говорим о мобильной разработке, которая куда сложнее!

Я начал разработку мобильных приложений примерно два десятилетия назад, еще во времена Windows CE/Pocket PC и Palm Pilot (были и другие платформы, но именно эти две были единственными настоящими игроками на рынке). Тогда все было не так уж и плохо, несмотря на ограниченный набор устройств и возможностей инструментов разработки, которыми мы располагали. Безусловно, использование этих инструментов было не таким приятным, как сегодня, но был всего один способ разработки приложений для Pocket PC, один способ разработки приложений для Palm OS. Звучит не очень, но отсутствие выбора приводит к отсутствию путаницы, что является одной из самых больших трудностей в области разработки софта на сегодняшний день.

А еще, хотя сегодня это считается непопулярным, тогда не было понятия кроссплатформенной разработки. Раньше приходилось писать код дважды, чтобы приложение работало на обеих платформах. Учитывая различия между ними, это было не так-то просто.

С тех пор индустрия мобильных устройств и приложений претерпела немало эволюционных изменений, подъемов и падений. Долгое время у нас было много платформ для поддержки: Android, iOS, webOS, Tizen, Windows Mobile и несколько других, которые я даже не помню. Все это время перенос прило-

жений между платформами был нормой, поскольку не было хорошего кроссплатформенного подхода, по крайней мере без существенных компромиссов. Да, со временем стало проще, потому что улучшился инструментарий для нативной разработки. Apple выпустила свой SDK для iOS в 2008 году, а Google выпустил свой Android SDK год спустя – в 2009-м. Нам приходилось разрабатывать приложения для каждой платформы, поскольку разработка iOS основана на языке Objective-C (сегодня чаще на языке Swift), в то время как разработка Android основана преимущественно на языке Java (теперь чаще на Kotlin).

В конце концов, количество платформ стало сокращаться. На сегодняшний день это гонка Android и iOS, хотя есть и другие платформы, которые чаще всего используются *только* при решении специфических задач. В связи с этим применение кроссплатформенных инструментов становится более привлекательным.

Наша прогрессивная эпоха интернета предлагает создавать приложения с помощью технологий, лежащих в его основе, и как результат получить приложение, которое выглядит и работает примерно одинаково на обеих платформах (теоретически и на других тоже). Это сопровождается компромиссами, которые со временем минимизируются, но все еще существуют. Такие вещи, как производительность и прямой доступ к возможностям «железа», все еще сложно совместить с веб-технологиями.

Однако, помимо веб-технологий, в последние несколько лет мы наблюдали рождение и других кроссплатформенных инструментов, которые позволяют нам написать приложение один раз и работать с ним примерно одинаково в разных операционных системах. Популярные варианты – Corona SDK (в первую очередь для игр, но не обязательно), Xamarin, PhoneGap (просто веб-технологии, умно завернутые в собственный компонент WebView), Titanium и Sencha Touch (опять же, на основе веб-технологий, но с хорошим слоем абстракции над ним), может, еще несколько. Так что сейчас нам доступно множество различных вариантов, каждый со своими плюсами и минусами.

А теперь внимание! На арену выходит новый конкурент, жаждущий убить остальных и показать единственный верный путь написания кроссплатформенных мобильных приложений: Flutter.

Да, это немного глупое название… но знаете, мы можем закрыть на это глаза, потому что преимуществ у него выше крыши!

Что за (глупое) название?

Flutter – это продукт Google – ну, вы знаете, корпорации, которая основательно контролирует интернет, хорошо это или плохо (в случае с Flutter я думаю, что хорошо). Изначально этот фреймворк родился под именем Sky в 2015 году на саммите разработчиков Dart (не забудьте это слово, Dart, мы к нему скоро вернемся). Сначала он работал только на операционной системе Android от самого Google, но вскоре был портирован и на iOS, так что сегодня он поддерживает две ведущие мобильные операционные системы.

Первые версии Flutter были выпущены сразу после его анонса, а кульминацией стал выпуск стабильной версии «Flutter 1.0» 4 декабря 2018 года. После этого Flutter был готов к прайм-тайму, и пришло время для разработчиков запрыгивать на борт! Популярность Flutter можно было бы охарактеризовать как метеорическую, и на это есть веские причины.

Одна из них заключается в следующем: первоначально заявленная цель Flutter или, по крайней мере, одна из основных, заключалась в отрисовке пользовательского интерфейса со скоростью 120 кадров в секунду. Google осознавал, что отзывчивый интерфейс приведет пользователей в восторг, поэтому эта функциональность и легла в основу Flutter. Это благородная цель, которой достигают лишь немногие кроссплатформенные фреймворки (даже нативные инструменты – и те часто испытывают трудности со скоростью отрисовки сложного интерфейса).

Flutter также предоставляет свои готовые компоненты пользовательского интерфейса – в отличие, например, от Xamarin и ReactNative, он не использует нативные контролы. Другими словами, когда вы хотите, чтобы Flutter отобразил кнопку, он рисует ее сам, а не просит об этом операционную систему, как делают другие фреймворки. Именно это и отличает Flutter от остальных и позволяет приложениям быть одинаковыми на разных платформах. Важно то, что новые компоненты пользовательского интерфейса, или *виджеты* (это слово тоже запомните, потому что, как и Dart, оно тоже скоро встретится), могут быть добавлены во Flutter быстро и легко, не беспокоясь о том, поддерживает ли их сама операционная система.

Это также позволяет Flutter предоставлять специфические наборы виджетов в стилистиках Material и Cupertino. Первый реализует Material Design от самой Google – стиль Android по умолчанию. Последний реализует стиль iOS от Apple.

Flutter можно разделить на четыре основные части, включая Dart. Я собираюсь оставить это до следующего раздела, так что давайте перейдем ко второму компоненту – основному движку Flutter. Этот движок в основном написан на C++ и использует библиотеку Skia, так что производительность отрисовки сравнима с нативной. Skia – это небольшая графическая библиотека с открытым исходным кодом, которая также написана на C++ и имеет очень высокую производительность на всех поддерживаемых платформах.

В качестве третьего основного компонента Flutter предоставляет унифицированный доступ к возможностям поддерживаемых операционных систем. Другими словами, код для запуска камеры на iOS и Android будет единым, а для этого можно использовать готовые методы Flutter.

Последний компонент – это виджеты, но, как и Dart, они тоже заслуживают собственного раздела, так что вернемся к ним позднее.

Если коротко, то Flutter состоит из этих четырех модулей, поэтому не так уж и много нужно знать, чтобы начать на нем разрабатывать. Тем не менее я думаю, что немного углубленное изучение инструментов никогда не будет лишним. Надеюсь, вы согласны!

Теперь давайте более детально разберем Dart и виджеты, о которых говорили ранее.

Dart: Язык богов?

Когда Google начал работать над Flutter, им предстояло ответить на главный вопрос: какой язык программирования выбрать? Может быть, язык веб-разработки, такой как JavaScript? Или же Java, язык Android? Или ради поддержки iOS выбрать Swift (в конце концов, Swift является языком с открытым исходным кодом)? Возможно, что-то вроде Go или Ruby было бы хорошим вариантом. Как насчет «старой школы», C/C++? А может, попробовать C # от Microsoft (у него тоже открытый исходный код)?

Я уверен, что было много вариантов, но в конце концов Google решил (не без причины!) использовать язык, который они создали несколько лет назад: Dart.

Вся следующая глава посвящена Dart, поэтому сейчас я воздержусь от деталей, но приведу небольшой пример:

```
import "dart:math" as math;

class Point {
   final num x, y;
   Point(this.x, this.y);
   Point.origin() : x = 0, y = 0;
   num distanceTo(Point other) {
      var dx = x - other.x;
      var dy = y - other.y;
      return math.sqrt(dx * dx + dy * dy);
   }

   Point operator +(Point other) => Point(x + other.x, y + other.y);
}

void main() {
   var p1 = Point(10, 10);
   var p2 = Point.origin();
   var distance = p1.distanceTo(p2);
   print(distance);
}
```

Не обязательно сразу детально понимать всё, что вы здесь видите. Тем не менее если вы работали раньше с любым C-подобным языком, например Java или JavaScript, то готов поспорить, что вы без проблем во всем разберетесь. В этом и заключается главное преимущество Dart: большинство современных разработчиков смогут написать и понять подобный код довольно легко.

ПРИМЕЧАНИЕ. Интересно, что мы называем языки C-подобными, но сам C – потомок гораздо более старого языка ALGOL. Я думаю, что ALGOL никогда не получит заслуженного уважения, так что этой ремаркой я выражаю всю любовь к нему!

Не вдаваясь во все мельчайшие подробности (для этого предназначена следующая глава), я думаю, что даже небольшой справки по Dart будет достаточно. Google создал Dart еще в 2011 году, и изначально он был представлен на конференции GOTO в Орхусе, Дания. Первый релиз 1.0 состоялся в ноябре 2013 года, примерно за два года до выпуска Flutter. За Dart стоит благодарить Ларса Бака (который разработал ещё и JavaScript-движок V8, используемый в Chrome и Node.js) и Каспера Лунда.

Dart – это лаконичный язык, который быстро набирает обороты, в основном благодаря Flutter. Поскольку он считается языком общего назначения, его широко используют для создания веб-приложений, серверного кода и приложений IoT (Internet of Things, «интернет вещей»). Пока я писал эту главу, вышел опрос о том, какие языки программирования вызывают наибольший интерес разработчиков в 2019 году, опубликованный JAXenter: `https://jaxenter.com/poll-results-dart-word-2019-154779.html`. В результате два языка заметно опередили остальные: Dart и Python, Dart вырвался вперед. Dart испытал наибольший рост в 2018 году. И хотя Flutter – почти наверняка один из самых популярных вариантов его использования, но, несмотря на это, Dart развивается во всех направлениях. Так что будьте уверены, Dart не обделен вниманием.

Так что же такое Dart? Предыдущий пример кода демонстрирует главные ключевые моменты, о которых я хотел сказать:

- Dart полностью объектно-ориентирован;
- инфраструктура языка включает в себя сборщик мусора, поэтому нет необходимости следить за памятью;
- его синтаксис основан на C, который подойдет большинству разработчиков (тем не менее, как и у любого другого языка с похожим синтаксисом, у него есть ряд особенностей, которые поначалу могут сбить вас с толку);
- он поддерживает общие языковые конструкции, такие как интерфейсы, наследование, абстрактные классы, шаблонные классы (generics, или «дженерики») и статическую типизацию;
- Dart включает проверку соответствия типов. Это позволяет использовать алгоритм для контроля правильности вашего кода;
- он поддерживает многопоточность, так что одновременно может исполняться несколько отдельных процессов, обеспечивая высокую производительность;
- Dart может компилироваться в нативный код для повышения производительности. Он не только компилируется в код для процессоров ARM и x86 в режиме Ahead Of Time (при сборке приложения), но и может транслиро-

ваться в JavaScript, а также поддерживает динамическую компиляцию во время исполнения (Just In Time);

- Dart позволяет использовать большой набор репозиториев с готовыми библиотеками, которые обеспечивают дополнительную функциональность для всего, что может понадобиться разработчикам;
- поддержка популярных сред разработки, включая Visual Studio Code и IntelliJ IDEA;
- ядро Dart поддерживает создание «слепков» (snapshots), которые позволяют упаковать весь исполняемый код (не только ваш код, но и библиотеки) в единый двоичный файл, что ускоряет запуск приложения.

Dart также зарегистрирован в качестве международного стандарта ECMA-408, а его актуальную спецификацию всегда можно получить на сайте www.dartlang.org/guides/language/spec.

Как я уже отмечал ранее, вся вторая глава будет посвящена Dart, а пока мы перейдем к следующей важной теме.

Виджеты окружают!

Давайте вернемся к разговору о главном госте нашего шоу, Flutter, и концепции, которая лежит в его основе, – виджетах.

Flutter – это и есть виджет. Когда я говорю, что *он и есть* виджет, я имею в виду... ну... я имею в виду, что *почти* все в нем является виджетом (гораздо сложнее найти во Flutter то, что виджетом *не является*!).

Что же такое виджет, спросите вы? Это части вашего пользовательского интерфейса (хотя и не все виджеты явно отображаются на экране). Виджет также представляет собой фрагмент кода, например:

```
Text("Hello!")
```

... и это также виджет...

```
RaisedButton(
   onPress : function() {
      // Сделай что-нибудь.
   },
   child : Text("Click me!")
)
```

... и это тоже виджет...

```
ListView.builder(
   itemCount : cars.length,
   itemBuilder : (inContext, inNum) {
      return new CarDescriptionCard(card[inNum]);
   }
)
```

... и наконец, это виджет:

```
Center(
   child : Container(
      child : Row(
         Text("Child 1"),
         Text("Child 2"),
         RaisedButton(
            onPress : function() {
               // Do something.
            },
            child : Text("Click me")
         )
      )
   )
)
```

Последний пример интересен тем, что на самом деле это иерархия виджетов: виджет Center, а в нем виджет Container, содержащий виджет Row, который, в свою очередь, содержит дочерние виджеты Text и кнопку RaisedButton.

Не важно, что это за виджеты (хотя названия их отлично характеризуют), *главное*, что вся иерархия виджетов, которую вы видите, *сама по себе* также считается виджетом Flutter.

Да, во Flutter повсюду виджеты! Виджеты окружают! Flutter – это Опра [Популярная ведущая ТВ-шоу в США. – *Прим. перев.*] в мире фреймворков пользовательского интерфейса: вам нужен виджет – вы получаете виджет! Да, вы получаете виджет! Вы ВСЕ получаете виииииджеты!

Как я сказал ранее, во Flutter практически все – это виджет. Есть очевидные вещи, о которых люди думают, употребляя слово виджет в контексте пользовательского интерфейса: кнопки, списки, изображения, поля текстовых форм и все такое прочее. Это все виджеты. Но во Flutter-то, что вы виджетами не считаете, это все еще они. Например, рамка вокруг изображения, состояние поля текстовой формы, текст, отображаемый на экране, даже тема, которую использует приложение.

В результате мы видим, что код во Flutter – это гигантская иерархия виджетов (и эта иерархия имеет конкретное имя во Flutter: Widgets Tree, «дерево виджетов»). Видите ли, большинство виджетов являются контейнерами, это означает, что они могут иметь дочерние элементы. Некоторые виджеты могут иметь только один такой элемент, в то время как другие могут иметь много. И тогда у каждого из них может быть один или несколько дочерних элементов, и так далее, и так далее!

Все виджеты являются классами Dart, и у них есть обязательное требование: предоставить метод `build()`. Этот метод должен возвращать... подождите, подождите... *другие виджеты*! Есть очень мало исключений из этого, например низкоуровневые виджеты, такие как виджет Text, который возвращает прими-

тивный тип (в нашем случае – String), но большинство возвращают один или несколько виджетов. Помимо этого требования, виджет – это старый добрый класс Dart, который не отличается от класса в любом другом объектно-ориентированном языке (за исключением синтаксиса).

Виджет во Flutter расширяет (extends) один из стандартных классов, которые он же и предоставляет, что характерно для объектно-ориентированной парадигмы. Расширенный класс (extended class) определяет, с каким виджетом мы имеем дело на фундаментальном уровне. Есть два самых базовых класса, которые вы будете использовать, вероятно, 99 % времени: `StatelessWidget` и `StatefulWidget`.

Виджет, унаследованный от `StatelessWidget`, не изменяется после отображения и называется виджетом без состояния, потому что он не имеет состояния (логично). Такие виджеты, как `Icon` (отображает небольшие изображения) и `Text` (отображает строки текста), тоже называют виджетом без состояния. Примером подобного класса может быть следующее:

```
class MyTextWidget extends StatelessWidget {
   Widget build(inContext) {
      return new Text("Hello!");
   }
}
```

Да, здесь нет ничего особенного!

В отличие от `StatelessWidget`, наследники базового класса `StatefulWidget` изменяются, когда пользователь взаимодействует с ним. `CheckBox`, `Slider`, `TextField` – это все известные примеры виджетов с состоянием (и, кстати, когда вы видите, что они написаны с большой буквы, то это означает, что я имею в виду фактические имена классов Flutter, а не общие термины). Когда вы кодируете такой виджет, вам нужно создать *два* класса: сам класс виджета с состоянием (`StatefulWidget`) и класс состояния (`State`), связанный с ним. Вот пример виджета StatefulWidget и связанного с ним класса State:

```
class LikesWidget extends StatefulWidget {
   @override
   LikesWidgetState createState() => LikesWidgetState();
}

class LikesWidgetState extends State<LikesWidget> {
   int likeCount = 0;

   void like() {
      setState(() {
         likeCount += 1;
      });
   }

   @override
```

```
   Widget build(BuildContext inContext) {
      return Row(
         children : [
            RaisedButton(
               onPressed : like,
               child : Text('$likeCount')
            )
         ]
      );
   }
}
```

Опять же, я не жду, что вы полностью поймете этот код, так как расширять знания по Dart мы начнем позже. Но я всё равно считаю, что вы приблизительно понимаете, что здесь происходит. По крайней мере, то, как код виджета и его объект состояния взаимодействуют и связаны. Может, это не так уж очевидно, но не беспокойтесь, это ненадолго!

Возвращаясь к виджетам без состояния, следует отметить, что термин «виджет без состояния» не совсем точен, потому что, будучи классом Dart, который имеет свойства и инкапсулированные данные, виджеты без состояния в некотором смысле имеют состояние. Основное различие между StatelessWidget и StatefulWidget заключается в том, что виджет без состояния (stateless) не умеет *автоматически* перерисовываться при изменении его «состояния», тогда как виджет с состоянием может. Когда виджет с состоянием изменяется, независимо от того, что вызывает изменение, возникают определенные события жизненного цикла. Когда состояние виджета изменяется, то происходит вызов определенных методов, и он перерисовывается (если необходимо, то Flutter делает это автоматически)

Представьте: оба типа виджетов могут иметь состояние, но Flutter распознает и управляет только stateful-виджетом. Таким образом, только StatefulWidget может быть автоматически перерисован в ответ на внешнее событие, и это контролируется самим Flutter, а вам не нужно прописывать код вручную.

Возможно, теперь вы захотите пользоваться только виджетами с состояниями, так как это сократит вашу работу, но вскоре вы поймете, что это не совсем так. В результате вы предпочтете виджет без состояния, даже если сейчас это кажется вам нелогичным. Но давайте опустим это ненадолго.

Есть два важных аспекта, на которые вы уже наверняка обратили внимания. Во-первых, пользовательский интерфейс построен путем создания виджетов. Это приводит к дереву виджетов, о котором я упоминал ранее. В то время как код самих виджетов является обычным классом, важно учитывать вложенность и расположение виджетов на экране. Это важно потому, что большинство виджетов Flutter сами по себе довольно просты, и только через композицию вы можете создать сложный пользовательский интерфейс. Даже относительно тривиальный пользовательский интерфейс содержит в себе целую группу виджетов.

Во-вторых, пользовательский интерфейс во Flutter описывается напрямую в коде. Я знаю, что это кажется очевидным, но задумайтесь вот над чем: для пользовательского интерфейса Flutter нет отдельного языка разметки, как HTML в веб-разработке. Преимущество заключается в том, что есть только один язык для изучения, единая парадигма для восприятия. Может, сначала это и незаметно, но это важное преимущество Flutter над конкурирующими вариантами.

Пока это все, что нужно знать о виджетах. Мы изучим каталог виджетов более подробно начиная с главы 3, и, конечно, мы рассмотрим использование каждого из них в главе 4, когда будем делать приложения с ними. В конце концов, вы получите хорошие знания о наиболее распространенных виджетах Flutter, а также другие, не менее полезные базовые навыки использования и создания виджетов в целом.

Ближе к делу: плюсы и Минусы Flutter

Как и с любым фреймворком, нам как хорошим разработчикам нужно оценить преимущества и подводные камни Flutter. Во Flutter есть и то, и другое, я же не буду бросаться в крайности и говорить, что это «панацея от всех бед» или же полный провал. И если кто-то говорит вам, что он идеален, то вам просто пускают пыль в глаза. Во Flutter есть свои недостатки, но я скромно предположу, что есть довольно много проектов и разработчиков, для которых он может стать отличным вариантом, если не *лучшим*.

Давайте уже обсудим все плюсы и минусы Flutter, а также сравним его с конкурентами.

- За: горячая перезагрузка (hot reload) – к ней я вернусь после того, как мы изучим настройку окружения и взглянем на первый пример приложения – вы сами убедитесь, что это большое преимущество Flutter. ReactNative также включает возможность горячей перезагрузки, особенно если вы используете сторонний компонент Expo. Однако во Flutter эта функциональность реализована более качественно и стабильно. Немногие фреймворки, даже нативные, могут похвастаться подобными возможностями.
- Против: только для мобильных устройств. На момент написания этой книги можно было использовать Flutter только для разработки мобильных приложений iOS и Android. Если вы полюбите Flutter, то будете разочарованы тем, что не сможете использовать его для разработки всех ваших приложений. Тем не менее обратите внимание, что я начал со слов «на момент написания». Это потому, что с высокой степенью вероятности Flutter также будет доработан для поддержки приложений для веб-браузеров, Windows, macOS, Linux и других платформ. [И да! Flutter портировали на веб-браузеры и десктопные платформы, хотя процесс этот еще только начался. – *Прим. перев.*]

- За: да, он действительно кроссплатформенный – ваши приложения Flutter будут корректно работать на iOS и Android (и в конечном итоге преемнике Android – Fuchsia). Flutter поставляет два набора виджетов из коробки, один для iOS и один для Android, поэтому ваши приложения могут как выглядеть одинаково на обеих платформах, так и учитывать стилистику целевой операционной системы.
- Против: веб-разработчики, не привыкшие видеть описание логики поведения и разметки пользовательского интерфейса в одном классе, как правило, очень эмоционально реагируют на примеры Flutter. Для сравнения, ReactNative, у которого также в одном классе описывается и логика и разметка, первое время страдал от подобных жалоб, но потом разработчики привыкли.
- За: Dart – простой и мощный, объектно-ориентированный и строго типизированный, что позволяет разработчикам быть очень продуктивными, быстрыми и делать меньше ошибок. Как только вы пройдете тернистый начальный путь обучения, вам понравится Dart, особенно в сравнении с JavaScript, Objective-C или Java.
- Против: Google – я причисляю это к недостаткам, и вы определенно можете со мной не согласиться (раньше я часто спорил об этом сам с собой). Некоторые люди чувствуют себя некомфортно от такого контроля Google над интернетом, даже если Google этим активно не пользуется. Когда вы доминируете, то, как правило, многое контролируете. Тем не менее некоторые люди опасаются, что Google наращивает обороты и на еще одном направлении – для них это может быть уже слишком. Другие, конечно, посмотрят на это и скажут, как здорово, что гигант поддерживает такую замечательную технологию. Так что этот «недостаток» можно назвать спорным. И выбрать сторону можете только вы.
- За: виджеты – Flutter предоставляет разработчикам богатый набор виджетов, и этого может быть вполне достаточно для построения любого приложения. Вы также можете создавать и свой собственный виджет (на самом деле вы *всегда* будете так делать, не важно, на каком уровне), и даже использовать многие сторонние виджеты, дабы расширить возможности приложения. Эти виджеты так же просты в использовании, как и те, что нам предлагает сам Flutter.
- Против: дерево виджетов (widgets tree) может стать недостатком, ведь иногда вы будете сталкиваться с очень глубоко вложенной иерархией, и разобраться со структурой кода станет не так-то просто. С развитием интернета мы к этому уже привыкли, потому что HTML сам по себе является древовидным, однако поскольку практически все во Flutter является виджетами, иерархия иногда может быть даже глубже HTML, а стиль кода Dart выглядит сложнее. Конечно, есть методы, позволяющие это упростить. Но о них я расскажу позже на примерах реального кода, и да, это

все еще недостаток, потому что вы должны осознавать его и уметь работать с ним самостоятельно. Ни Flutter, ни Dart не предложат вам подсказок в решении.

- За: инструменты – как вы увидите в следующем разделе, настроить среду разработки для Flutter очень легко. Тем не менее вы можете выйти за пределы этой базовой среды и использовать многие уже привычные вам инструменты.
- Против: реактивное программирование и управление состоянием – Flutter обычно считается реактивным (reactive). Метод `build()`, который вы видели ранее, принимает в качестве аргумента текущее состояние, а то, что он возвращает, – это визуальное представление этого виджета, на основе обновленного состояния. Когда обновляется состояние, виджет «реагирует» на это и обновляет себя с помощью повторного вызова метода `build()`. Всё это – стандартный механизм Flutter и типовой жизненный цикл виджета. Сравните это с «нереактивными» подходами, когда вы создаете виджет, а затем самостоятельно вызываете нужные методы для его модификации или обновления. В целом реактивная парадигма довольно удобна, хотя для Flutter она может быть и недостатком, потому что иногда это усложняет простые вещи (вы сами увидите подобные проблемы в последующих главах, а также научитесь с ними справляться). С этим связана и сложность управления состояниями, которая является недостатком Flutter в том смысле, что нет канонически правильного и неправильного способа это делать. Существует множество подходов, и у каждого есть свои плюсы и минусы, а вам нужно будет решить, что лучше соответствует вашим потребностям (и да, я буду предлагать то, что считаю хорошим подходом). Google работает над таким каноничным подходом прямо сейчас, но пока он не готов, я буду рассматривать отсутствие определенного пути как недостаток (хотя некоторые считают гибкость преимуществом!).
- За: специфические для платформы виджеты – поскольку интерфейсы Flutter пишутся с помощью кода, у вас может быть одна кодовая база, которая поддерживает как iOS, так и Android, но даже в ней есть различия, которые вам нужно учитывать. Например, вы всегда можете узнать в коде значения `Platform.isAndroid` и `Platform.isIOS`, чтобы определить, на каком устройстве работает ваше приложение, а затем добавить условие для создания разных виджетов для разных платформ. Возможно, вам нужен RaisedButton на Android и Button на iOS.
- Против: размер приложения – приложения Flutter, как правило, немного больше своих нативных аналогов, потому что они включают основной движок Flutter, библиотеки и другие ресурсы фреймворка. Размер приложения элементарного «Hello, world!» на Flutter может превышать 7 МБ. Поэтому если вам решительно важен размер приложения, то Flutter может стать не лучшим выбором.

Надеюсь, что к этому моменту у вас появилось понимание сильных и слабых сторон Flutter, поэтому предлагаю перейти к практике.

Хватит болтать, начинаем практику с Flutter!

Прежде чем мы доберемся до кода, нам следует установить Flutter и некоторые инструменты, которые могут понадобиться. Надеюсь, вы понимаете, что не так уж просто начать писать код на Flutter, не установив его!

К счастью, настроить рабочее окружение довольно легко.

Flutter SDK

Первый шаг, который вы должны сделать, – это загрузка, установка и настройка Flutter SDK. Это очень важно! Второй шаг, который технически не обязателен, но необходим для целей этой книги, – это загрузка, установка и настройка Android Studio, включая Android SDK и эмулятор.

Во-первых, зайдите на `https://flutter.io` для загрузки установочных пакетов и получения документации Flutter. Нажмите кнопку **Get Started** в верхней части. Найдите **Install** и выберите свою операционную систему (Windows, MacOS или Linux).

Обратите внимание, что мне совсем не стыдно признать, что я в первую очередь пользователь Windows. Это то, в чем я разбираюсь и что предпочитаю! Так что эта книга будет ориентирована на Windows, и если вы используете другую ОС, то вы в какой-то степени будете сами по себе. С учетом вышесказанного я буду обращать ваше внимание на особенности инструментов для разных ОС, если будут иметься существенные различия. На самом деле их не должно быть вне зависимости от того, используете вы Windows или нет. При этом Flutter сам проинструктирует вас, если возникнут проблемы.

Выберите соответствующую ссылку, и Flutter предоставит вам информацию о загрузке и установке SDK. SDK не отличается от любого другого софта, поэтому трудностей быть не должно. Хочу отметить, что в инструкции вас просят указать путь для SDK. Но вы можете пропустить этот шаг. Просто обратите внимание, что если вы пропустите его, все команды должны быть выполнены из каталога SDK, или с указанием полных путей до приложений в этом каталоге. Как только мы дойдем до шага с Android Studio, вы обнаружите, что добавление SDK к пути действительно не имеет значения, Android Studio сделает это за вас. Но если вы собираетесь работать с командной строкой, то не забудьте прописать полный путь.

Первая команда, которую вы *будете* выполнять из командной строки, и первая, которую вы будете делать сразу после установки SDK в соответствии с инструкциями на сайте, – это «flutter doctor». Большинство команд, которые вы будете вводить при работе с SDK, если не все из них, начинаются с flutter, который фактически является исполняемым файлом, например doctor – это одна

из команд, которую вы можете выполнить с его помощью. Это важно, потому что команда doctor проверит, сможете ли вы начать работать, и если нет, то скажет вам, что не так.

Если вы запустите ее сейчас, то, скорее всего, обнаружите проблемы, так и должно быть на данном этапе, не волнуйтесь, следующий шаг это исправит: установка Android Studio.

Android Studio

Еще раз, инструкции на сайте Flutter сопровождают вас и имеют незначительные различия для каждой ОС, но как только вы установили Android Studio, запустите её и воспользуйтесь мастером настройки. Это загрузит Android SDK, образы эмулятора и все необходимое для его работы. Затем установите специальные плагины Dart и Flutter, в документации это подробно описано.

Если вы продолжите следовать инструкциям, запустится процесс подключения вашего Android телефона или планшета к компьютеру, но убедитесь, что `flutter doctor` его видит. Однако вы можете пропустить этот шаг! Конечно, если у вас есть устройство Android, то необходимость в эмуляторе пропадает.

Однако если вы предпочитаете iOS или если вам не нравится использовать реальное устройство при разработке кода Flutter – я не подключаю свой телефон, – тогда мое предложение состоит в том, чтобы войти в Android Studio, запустить менеджер AVD (Android Virtual Machine), который вы можете найти в меню конфигурации на экране запуска и создать себе виртуальное устройство Android. Я предлагаю создать виртуальное устройство Pixel_2, используя уровень API 28 (убедитесь, что вы установили данную версию API), и задать ему разрешение 1080 × 1920 (420 dpi) с операционной системой Android 9. Затем выберите образ x86 (x86_64). Если говорить о производительности, виртуальные устройства Android долгое время имели плохую репутацию, но сейчас этот тип виртуальных устройств работает исключительно хорошо, достигая почти нативной производительности в большинстве случаев. Хотя это не имеет значения, идем дальше и настроим его SD-карту на 512 МБ. Значения по умолчанию должны соответствовать вашим желаниям, но уровень API и тип процессора – это ключевые аспекты.

Когда всё будет готово – запускаем код Flutter на эмуляторе. Или вы можете сделать все это из командной строки с помощью SDK. Мы же в данной книге будем делать это в Android Studio.

Обратите внимание, что если вы повторно запустите `flutter doctor`, он все равно будет сообщать о проблеме, а именно что он не может найти устройство Android, предполагая, что виртуальное устройство, которое вы создали, не работает. Но если flutter doctor обнаружит его, он выдаст вам справку, что все хорошо. Наконец, если вы видите сообщение о том, что эмулятор *не* запущен, при условии что реальное устройство Android подключено и это единственная проблема, о которой сообщает flutter doctor, то вы можете не обращать на нее внимания.

Если вас сейчас интересует iOS, пожалуйста, успокойтесь! Хоть мы используем Android Studio, это никоим образом не означает, что все это не применимо для iOS. Об iOS необходимо знать в первую очередь то, что если вы хотите протестировать реальное устройство iOS или создать свое приложение для продажи, вам понадобится компьютер Mac и Apple Xcode IDE. Приложения для продажи не рассматриваются в этой книге, хотя, будь то для iOS или Android, эмулятор отлично соответствует нашим задачам.

Типичное приложение «Hello, World!»

Если вы продолжите следовать инструкции на веб-сайте, то последним шагом станет создание небольшого приложения Flutter. Документация там отличная, но я предлагаю пропустить ее и позволить мне провести вас по этому пути самому.

Первым шагом необходимо позволить Android Studio (в сочетании с Flutter SDK) создать приложение для нас. Процесс довольно прост, и как только мы запустим это базовое приложение на нашем виртуальном устройстве, мы немного изменим его, чтобы вы могли видеть горячую перезагрузку наглядно.

Но сначала давайте создадим проект! При первом запуске Android Studio вы увидите окно, как показано на рис. 1-1.

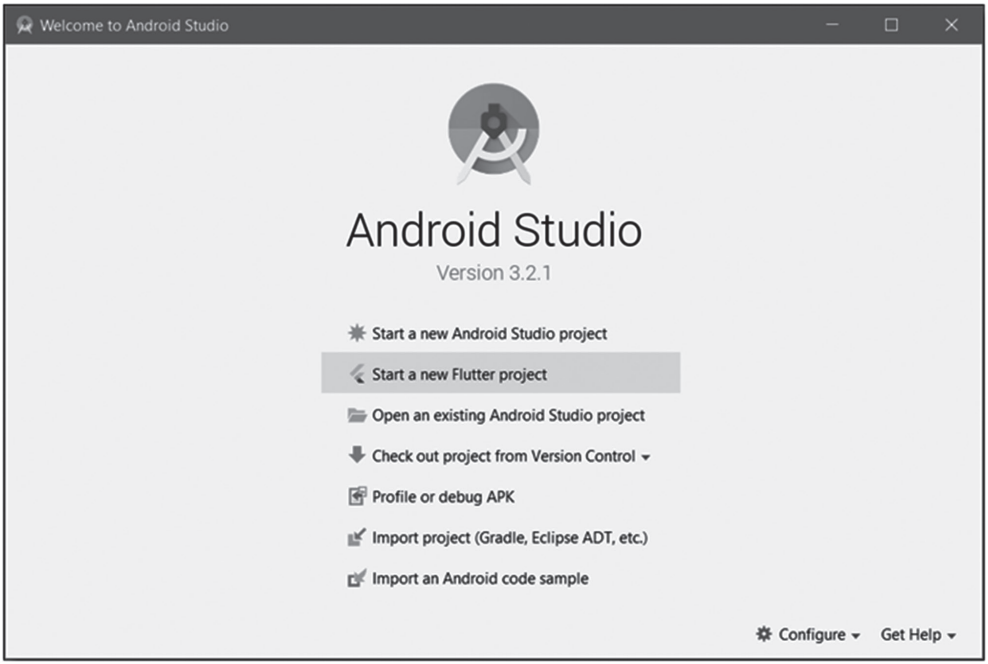

Рисунок 1-1. *Первый шаг в Android Studio*

Видите строчку *Start a new Flutter Project*? Это то, что нужно, кликайте! Вы увидите начальный экран мастера нового приложения, как показано на рис. 1-2.

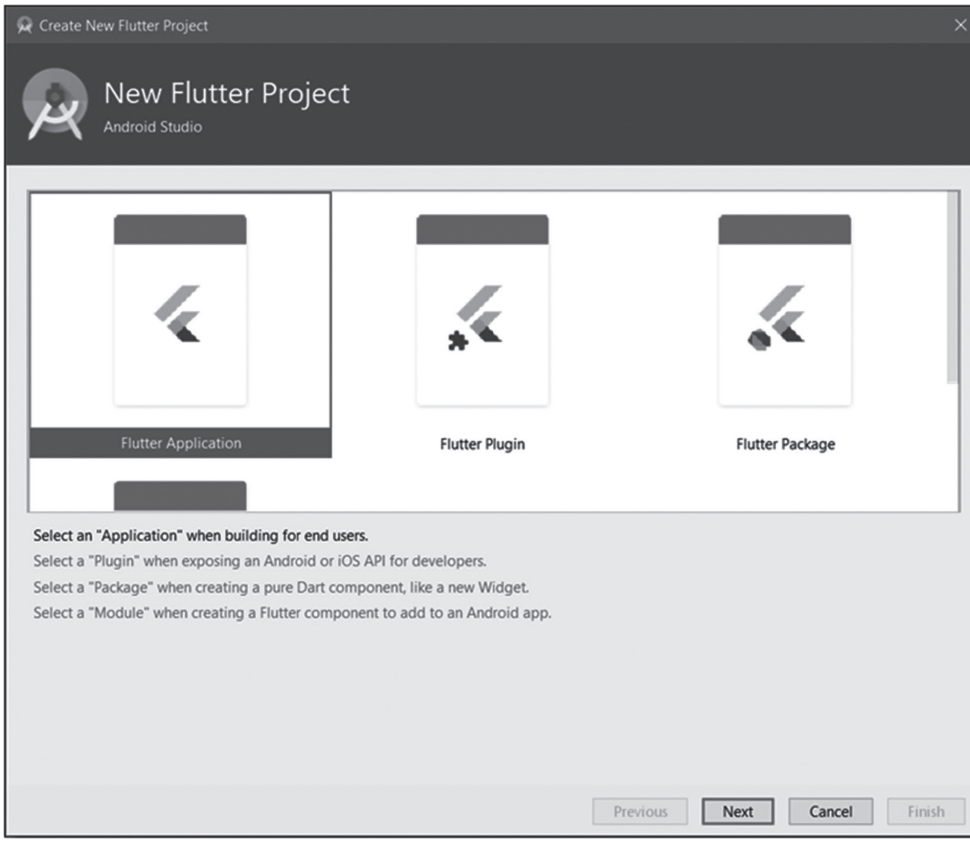

Рисунок 1-2. *Выберите тип проекта Flutter, который хотите создать*

Существует четыре типа проектов Flutter, которые вы можете создать:

- Flutter Application (его мы будем использовать в этой книге);
- Flutter Plugin (плагин позволяет использовать нативную функциональность Android или iOS для ваших приложений Flutter на основе Dart);
- Flutter Package (необходим только в том случае, если вы хотите распространять пользовательский виджет независимо от приложения);
- Flutter Module (позволяет встраивать приложение Flutter в нативное приложение Android).

Выберите **Flutter Application** и нажмите кнопку **Next** (Далее). Откроется окно, как показано на рис. 1-3.

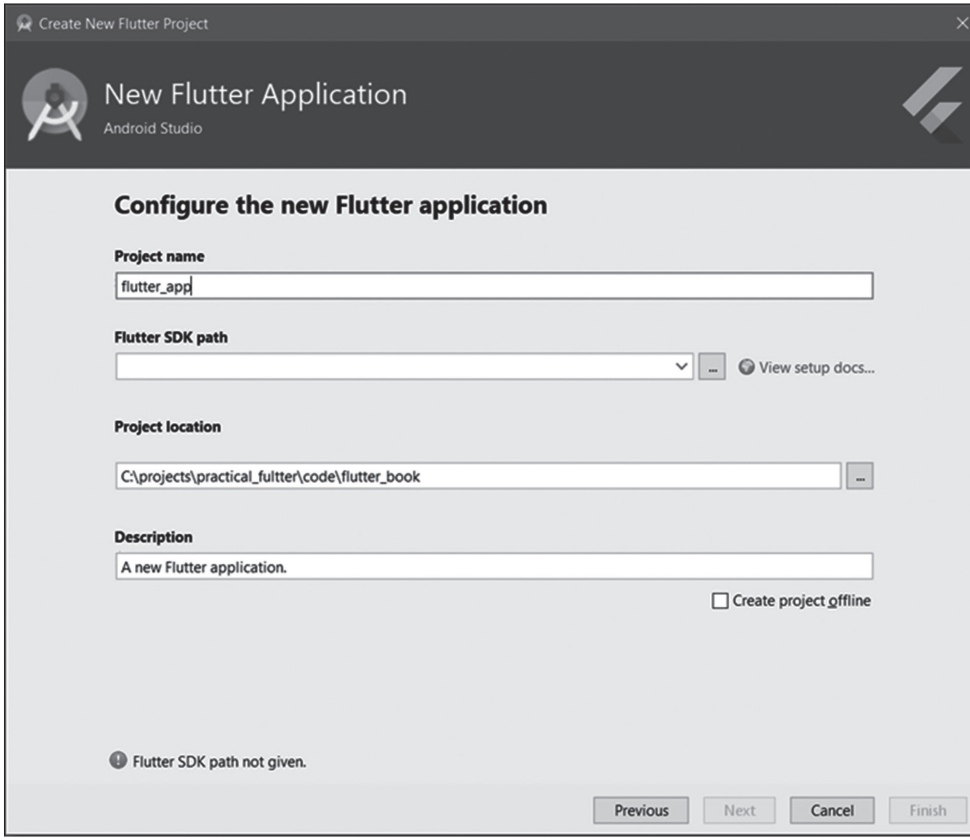

Рисунок 1-3. *Ввод необходимой информации о вашем приложении*

Здесь вы введете информацию о создаваемом приложении. Вы можете дать проекту любое название или описание, а также оставить значения по умолчанию. При необходимости обновите поле **Project location** (или просто используйте значение по умолчанию). Вы видите эту ошибку внизу? Не пугайтесь, вы уже выбрали нужный путь. Но если вы видите это, то Android Studio еще не знает, где находится Flutter SDK, и вам необходимо его указать. Просто перейдите к SDK, который вы должны были установить ранее, и убедитесь, что Android Studio этим доволен (ошибка исчезнет), и снова нажмите кнопку **Next**, чтобы перейти к экрану с рис. 1-4.

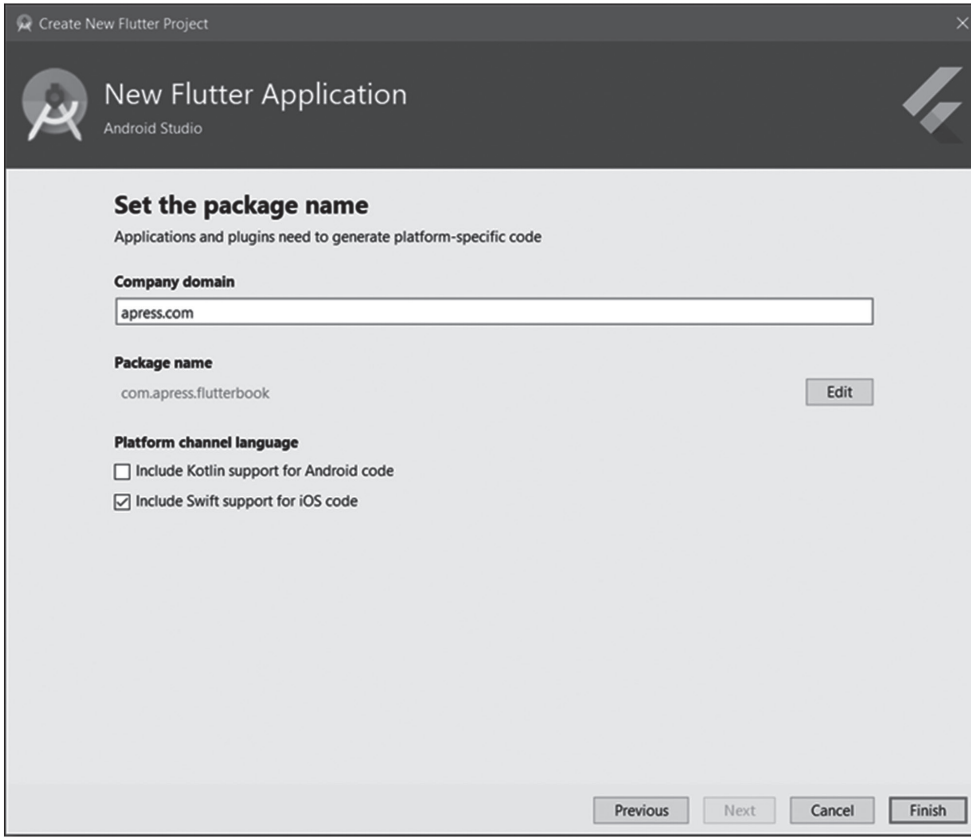

Рисунок 1-4. *Окончательная информация о проекте*

Этот экран требует немного больше информации, в первую очередь домен компании. Если у вас нет домена, то вы можете поместить любое значение, которое вам нравится. Тебя зовут Jim? Ты *можешь* ввести «Jim»! Ты можешь написать «Jim», даже если это не твое имя, хотя это было бы немного странно. Обратите внимание, что полное имя пакета состоит из названия приложения и домена вашей компании, введенного на последнем экране. Это имя пакета должно быть уникальным, если вы хотите опубликовать приложение в магазине приложений, хотя для нашего тестирования здесь это не имеет значения.

Примечание. Вы также можете увидеть поле `Sample Application`, в зависимости от версии Android Studio и установленного плагина Flutter. Это для того, чтобы мастер сгенерировал образец кода для вас, если хотите, но нам это не нужно.

Наконец, выберите язык платформы, оставляя Kotlin неотмеченным, так как Swift будет наиболее подходящим. Это относится к базовому языку платформы, используемому под обертками Flutter, и если вы не собираетесь взаи-

модействовать с нативным кодом приложения, вам, по идее, должно быть без разницы. В конечном счете это не имеет значения для целей книги.

Так что нажмите кнопку **Finish**, и Android Studio покажет вам простое приложение Flutter. Это может занять несколько минут, поэтому проверьте строку состояния внизу, чтобы убедиться, что все задачи выполнены. Когда всё будет готово, посмотрите в верхнюю часть Android Studio, на панель инструментов, и найдите выпадающий элемент, в котором перечислены подключенные устройства, как показано на рис. 1-5.

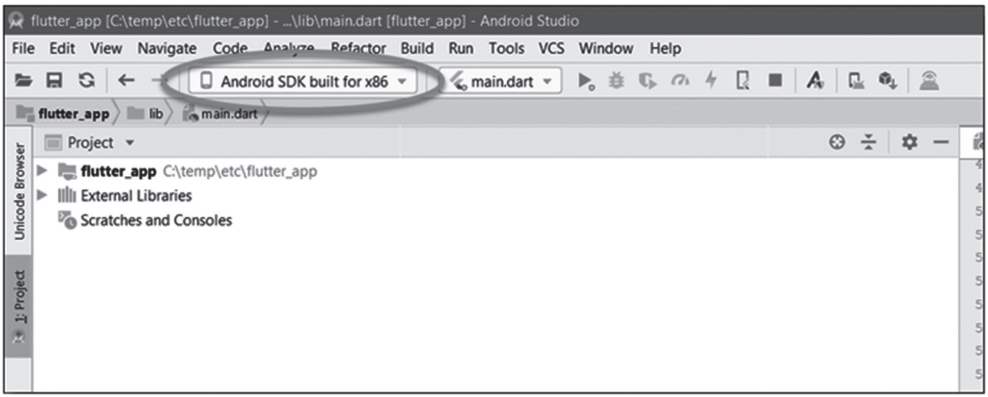

Рисунок 1-5. *Выпадающий список устройств в Android Studio*

Вы должны увидеть созданный ранее эмулятор. Выберите его, и, если он еще не запущен, он должен запуститься в ближайшее время. Как только это произойдет, нажмите на значок **Run** (зеленую стрелку рядом с выпадающим списком `main.dart`) – это точка запуска приложения. Подождите, пока приложение будет собрано, развернуто и запущено на эмуляторе (в зависимости от вашей машины это может занять до минуты, поэтому будьте терпеливы – процесс ускорится после первой сборки). Вы должны увидеть в эмуляторе что-то вроде рис. 1-6.

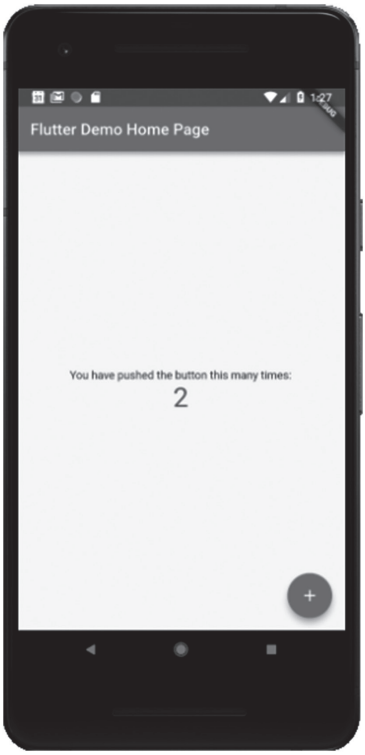

Рисунок 1-6. *Мое первое приложение Flutter!*

Это простое приложение, но оно многое показывает. Нажмите на круглую кнопку со знаком «плюс» (она называется «плавающей кнопкой действия», или Floating Action Button, FAB) и обратите внимание, что счетчик увеличивается с каждым щелчком мыши.

Получившийся код должен автоматически открываться в Android Studio (если файл `main.dart` найден в директории верхнего уровня) и быть следующим (я, конечно, удалил комментарии и отформатировал его, чтобы он лучше выглядел в печатном варианте):

```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {

   @override
   Widget build(BuildContext context) {
      return MaterialApp(
         title: 'Flutter Demo',
         theme: ThemeData(
            primarySwatch: Colors.blue,
```

```
            ),
            home: MyHomePage(title: 'Flutter Demo Home Page'),
        );
    }
}

class MyHomePage extends StatefulWidget {

    MyHomePage({Key key, this.title}) : super(key: key);

    final String title;

    @override
    _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {

    int _counter = 0;

    void _incrementCounter() {
        setState(() {
        _counter++;
    });
}

@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
            title: Text(widget.title),
        ),
        body: Center(
            child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: <Widget>[
                    Text(
                        'You have pushed the button this many times:',
                    ),
                    Text(
                        '$_counter',
                        style: Theme.of(context).textTheme.display1,
                    ),
                ],
            ),
        ),
        floatingActionButton: FloatingActionButton(
            onPressed: _incrementCounter,
            tooltip: 'Increment',
```

```
            child: Icon(Icons.add),
          ),
        );
    }
}
```

Хотя здесь не так много кода, зато он много делает. Думаю, пока вам недостаточно знаний, чтобы полностью разобраться в происходящем, ведь мы еще не говорили о Dart настолько подробно. Но я не хочу оставлять вас абсолютно в неведении, так что есть несколько ключевых моментов, которые я разъясню.

Во-первых, обратите внимание, что основная точка входа каждого приложения Flutter является методом `main()`. В `main()` вызывается метод `runApp()`, который возвращает виджет верхнего уровня. В начале иерархии всегда есть виджет, который содержит все остальные, здесь это экземпляр класса `MyApp`. Этот класс является виджетом без состояния, поэтому единственная его задача – предоставить метод `build()`. Виджет, возвращаемый из него (помните: `build()` – всегда возвращающий виджет, в котором могут быть дочерние элементы, а могут и не быть), – это экземпляр `MaterialApp`, который является виджетом, предоставленным Flutter (это можно увидеть в самом начале нашего кода). Мы поговорим об этом виджете в главе 3, когда будем подробнее рассматривать виджеты Flutter, но главное, что он обеспечивает базовую инфраструктуру для приложения Material. Как видите, мы задали название в одном из аргументов конструктора `MaterialApp`, это название вы увидите в строке состояния (Status Bar) вашего приложения. Также вы можете установить тему для приложения Flutter и предоставить подробную информацию о ней, например основной цвет, который она использует, у нас он синий.

Наш виджет `MaterialApp` содержит один дочерний элемент, который представлен экземпляром класса `MyHomePage`.

Класс `MyHomePage` определяет виджет с состоянием, так что нам понадобится два класса, класс «core», который наследуется от `StatefulWidget`, и класс состояния, связанный с ним, наследуется от `State`.

В любом случае, метод `build()` этого виджета снова возвращает единственный виджет, на этот раз `Scaffold`. Но не зацикливайтесь на этом, потому что в главе 3 мы разберем каждый из них. Если в двух словах, то `Scaffold` обеспечивает фундаментальный визуальный макет для приложения, включая такие элементы, как строка состояния (`AppBar` – это фактически виджет), где находится название. `Scaffold` также обеспечивает механизмы, позволяющие «зацепить» FAB, – экземпляр виджета `FloatingActionButton` передается в конструктор `Scaffold`.

Другой аргумент, переданный в конструктор `Scaffold`, – это тело, в которое мы добавляем другие виджеты в качестве дочерних. Здесь вы можете наблюдать мантру «все это виджет» в действии, потому что у нас есть центральный виджет, который является контейнерным виджетом, который – как вы уже догадались – центрирует свой дочерний элемент. В этом случае дочерним является виджет `Column`, который размещает уже своих детей в одну колонну.

Column содержит два дочерних виджета Text – один для статического текста «You have pushed the button this many times:», а другой для отображения количества нажатий кнопки.

Все станет понятнее, когда мы в течение следующих двух глав углубимся в Dart, а затем во Flutter. И хотя я опустил много деталей, мне кажется, этого объяснения вполне достаточно для достойного представления о том, что происходит в коде.

Горячая перезагрузка: вот что я люблю!

Здесь все становится невероятно крутым! Убедитесь, что у вас есть приложение, работающее в эмуляторе, а затем перейдите к Android Studio и найдите эту строку кода:

```
Text(
   'You have pushed the button this many times:',
),
```

Итак, вы можете изменить данный текст, поменяв «button» на «FAB» и зажав **Ctrl+S**, или выберите **Save All** в меню File. Теперь наблюдайте за эмулятором, и почти сразу вы увидите, что ваше изменение отражается на экране (это может занять несколько секунд, но всё же быстрее, чем при первом запуске).

Очень круто, не правда ли?

Горячая перезагрузка работает только в режиме отладки, в котором вы находитесь; это можно понять благодаря отладочному баннеру в правом верхнем углу приложения. В этом режиме ваше приложение фактически работает в виртуальной машине Dart (VM), а не компилируется в собственный код процессора, что происходит, когда вы создаете реальное приложение (да, ваше приложение будет работать медленнее в режиме отладки). Горячая перезагрузка работает путем обновления измененных файлов исходного кода в уже работающую виртуальную машину Dart, на которой размещается ваше приложение. Когда это происходит, VM обновляет классы, которые изменились, обновляя любые измененные поля и методы. Затем структура Flutter инициирует перестроение дерева виджетов, и ваши изменения отражаются автоматически. Вам не нужно пересобирать или перезапускать что-либо; все происходит автоматически по мере необходимости, чтобы ваши изменения были отображены на экране как можно быстрее.

Время от времени вы можете обнаружить, что изменение не приводит к перезагрузке, как ожидалось. Если это произойдет, первое, что нужно попробовать, – это нажать на значок горячей перезагрузки на панели инструментов, который на рис. 1-7 выглядит как молния (вы также можете найти опцию горячей перезагрузки в меню **Run** с соответствующей горячей клавишей **Ctrl+/**).

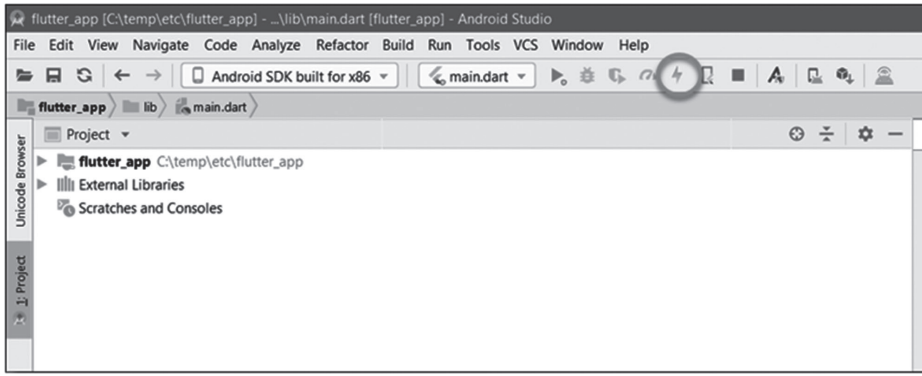

Рисунок 1-7. *Значок горячей перезагрузки в Android Studio*

Это должно помочь вам. Также обратите внимание на консоль, которая должна находиться в нижней части Android Studio, там вы увидите следующее сообщение:

```
Performing hot reload...
Reloaded 1 of 448 libraries in 2,777ms.
```

Кроме того, небольшая подсказка должна появиться рядом с консолью во время перезагрузки.

Обратите внимание, если вы нажали на **FAB** несколько раз, а затем изменили текст, текущее состояние приложения продолжит существование. Другими словами, количество повторных нажатий на кнопку сохранится после горячей перезагрузки (hot reload). Это позволяет легко изменять пользовательский интерфейс и мгновенно отображать текущее состояние, так что вы можете быстро и просто сверять вашу верстку с дизайном. Но что, если вы хотите, чтобы состояние не сохранялось? Тогда вы, вероятно, захотите выполнить горячий перезапуск (restart). Поэтому вам следует сделать это вручную (в отличие от горячей перезагрузки, которая происходит автоматически, когда вы вносите изменения в код и сохраняете его), выбрав опцию **Hot restart** (горячий перезапуск – не перезагрузка) в меню **Run** или нажав соответствующую горячую клавишу (**Ctrl+Shift+/**).

Интересно, что значка для горячего перезапуска на панели инструментов нет, но это всё же перезапустит ваше приложение, хоть и не выполнит новую сборку и очистку состояния.

Вы, естественно, можете перезапускать сборку в любое время (это происходит по команде **Run**), но каждый раз вы будете дожидаться компиляции, что ни капельки не экономит ваше время. Горячий перезапуск (restart) сработает почти так же быстро, как и горячая перезагрузка (reload), потому что работы он делает намного меньше, но достигает примерно того же эффекта.

Надеюсь, вы видите, насколько вы можете быть эффективны, используя горячую перезагрузку. Мне кажется, вы оцените это, когда познакомитесь с Flutter поближе!

Базовая структура приложения Flutter

Одна из последних тем, которой я коснусь во вступительной главе, – это общая структура приложения, которое было создано для вас. На рис. 1-8 вы видите первичную структуру каталога.

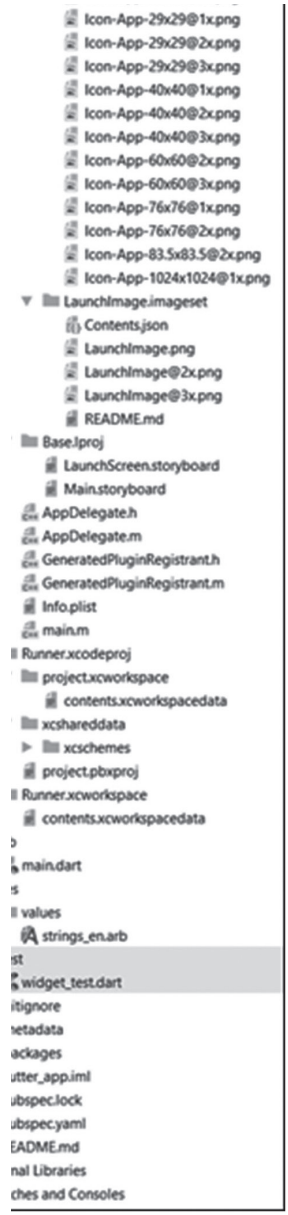

Рисунок 1-8. *Первичная структтура каталога проекта*

Как видите, существует пять каталогов верхнего уровня. А теперь подробнее о каждом из них:

- `android` – содержит специфический для Android код и ресурсы, такие как значки приложений, код Java, а также конфигурацию Gradle и ресурсы (Gradle является системой сборки Android). На самом деле это фактически целый проект Android, который вы можете построить с использованием стандартных инструментов Android. По большому счету, вам нужно изменять только значки (которые находятся в каталогах `android/app/src/main/res`, где каждый подкаталог имеет разное разрешение) и, в зависимости от того, что делает ваше приложение, файл `AndroidManifest.xml` в `android/app/src/main`, где вы можете установить специальные свойства приложения для Android;

- `ios` – как и android, этот каталог содержит код проекта, специфичный для iOS. Критическим содержимым здесь является каталог `ios/Runner/Assets.xcassets`, в котором находятся значки для вашего приложения, и файл `Info.plist` в `ios/Runner`, который служит той же цели, что и файл `AndroidManifest.xml` для приложений Android;

- `lib` – хотя сначала это может показаться странным, это место, где будет жить ваш код! Вы можете относительно свободно организовывать свой код, создавая любую структуру каталогов, хотя вам понадобится один файл, который служит точкой входа, и большую часть времени им будет main.dart, созданный автоматически;

- `res` – этот каталог содержит такие ресурсы, как строки для перевода вашего приложения на иностранные языки. Но в этой книге мы не будем иметь с ними дело;

- `test` – здесь вы найдете Dart-файлы для тестирования вашего приложения. Flutter предоставляет утилиту Widget Tester, которая использует автоматические тесты, чтобы подтвердить функциональность ваших виджетов. Мы не будем с ним работать, так же как и с res, потому что это необязательная часть разработки Flutter, о которой отдельно можно написать целую книгу! Тестирование – это, конечно, важно, но пока вы не научитесь писать приложения Flutter, вам будет нечего тестировать, а эта книга фокусируется на первой части вашего пути.

Хоть он и скрыт по умолчанию в Android Studio, есть также каталог .idea, в котором хранится информация о конфигурации Android Studio, поэтому вы можете его игнорировать (обратите внимание, что Android Studio основана на IDE IntelliJ IDEA, отсюда и название). Существует также скрытый каталог сборки, содержащий информацию, которую используют Android Studio и Flutter SDK для создания вашего приложения. Но мы проигнорируем и это.

Помимо каталогов, вы также найдете некоторые файлы в корне проекта. Это, как правило, единственные файлы, о которых вам нужно беспокоиться за

пределами каталога lib (все остальное на скриншоте вам не нужно знать вообще), и это:

- `.gitignore` – файл управления версиями Git используется, чтобы знать, какие файлы игнорировать из управления версиями. Использование Git совершенно необязательно при написании приложений Flutter, но этот файл генерируется в любом случае. Управление версиями – это то, что не сможет охватить даже целая книга, поэтому вы можете игнорировать данный файл;
- `.metadata` – данные, которые Android Studio отслеживает в вашем проекте. Вы можете игнорировать и это, так как вы никогда не будете редактировать их самостоятельно;
- `.packages` – у Flutter есть свой собственный менеджер пакетов для управления зависимостями в вашем проекте. Этот менеджер пакетов называется Pub, и он используется для отслеживания зависимостей в вашем проекте. Вы не будете взаимодействовать с ними напрямую или даже напрямую с Pub, поэтому его тоже можно оставить без внимания;
- `*.iml` – этот файл должен быть назван в честь вашего проекта и является файлом конфигурации проекта Android Studio. Вы никогда не будете редактировать его напрямую, так что игнорируем;
- `pubspec.lock` и `pubspec.yaml` – вы когда-нибудь работали с NPM? Знакомы с package.json и файлами package-lock.json, которые он использует? Ну, это те же вещи, но для Pub! Если вы незнакомы с NPM, `pubspec.yaml` – это то, как вы описываете свой проект для Pub, включая его зависимости. Файл `pubspec.lock` – это внутренний файл Pub. Вы определенно можете редактировать `pubspec.yaml`, но не `pubspec.lock`, а pubspec.yaml мы позже подробно рассмотрим;
- `README.md` – файл readme, который вы можете использовать, как хотите. Как правило, это файл Markdown – сайты, такие как GitHub, используют для отображения информации о вашем проекте при переходе к репозиторию, где этот файл находится в корневом каталоге.

Самым важным файлом здесь является pubspec.yaml, и он один из немногих, которые вам нужно будет редактировать, поэтом, если вы все забыли, его лучше запомните! Мы доберемся до него позже, когда нам нужно будет добавить зависимости в наш проект, но на данный момент сгенерированного файла вполне достаточно для наших нужд.

Еще парочка моментов «под прикрытием»

Если вы посмотрите на некоторые из файлов в каталоге *ios*, то заметите слово «Runner». Это подсказка о том, как работают приложения Flutter при сборке и запуске установочного пакета. Как отмечалось ранее, горячая перезагруз-

ка работает, потому что в режиме отладки ваш код запускается в виртуальной машине. Однако при сборке установочного пакета это так не работает. Ваш код компилируется в собственный код ARM. Фактически компилируется в библиотеку для процессора ARM, которую в дальнейшем использует нативное приложение, это объясняет, почему ваш код находится в каталоге lib.

Библиотеки Flutter наряду с вашим кодом приложения компилируются «до запуска» (Ahead-Of-Time, AOT) с помощью LLVM (Low-Level Virtual Machine, инфраструктура компилятора, написанная на C++, которая предназначена для компиляции и оптимизации программ, написанных на различных языках программирования) на iOS. На Android используется Native Development Kit (NDK) для сборки ARM-библиотеки. Эта библиотека включена в так называемый «runner», который является просто нативным приложением, которое... подождите, подождите... запускает ваше приложение. Представьте, что это тонкая обертка вокруг вашего приложения, которая знает, как запустить приложение, и предоставляет необходимые условия. В некотором смысле runner по-прежнему представлен виртуальной машиной, хотя и очень тонкой (почти как контейнер Docker, если вы с ним знакомы).

Наконец, runner вместе со скомпилированной библиотекой упаковывается в файл .ipa для iOS или файл .apk для Android, и у вас есть полный, готовый к установке или публикации пакет! Когда приложение запускается, runner загружает библиотеку Flutter и ваш код приложения, и с этого момента вся визуализация, ввод/вывод и обработка событий делегируется скомпилированному приложению Flutter.

ПРИМЕЧАНИЕ. Это очень похоже на то, как работает большинство кроссплатформенных мобильных игровых движков. Ранее я написал книгу о Corona SDK, библиотеке, которую я очень люблю, она работает очень похожим образом, хотя там используется язык Lua вместо Dart (который, могу поспорить, команда Flutter тоже рассматривала!). Любопытно, что Google, по сути, черпал вдохновение из игровых движков, чтобы создать Flutter, потому что это доказывает то, что я всегда говорил: если вы хотите быть лучшим программистом, единственный вид проекта, в котором вам следует отточить свои навыки, – это игровой. На этот раз мир получил целую платформу приложений! И если вы еще не заглянули вперед, последние две главы этой книги посвящены созданию игры Flutter, потому что я всегда советую создавать игры!

Итого

С этой главой вы начали свое путешествие в мир Flutter! Вы узнали о том, что такое Flutter, что он предлагает и почему вам стоит его использовать (и даже некоторые причины, по которым вы не захотите его использовать). Вы узнали о важных концепциях, таких как Dart и виджеты; узнали, как настроить свою среду разработки, чтобы работать с кодом Flutter; создали свое первое очень простое приложение Flutter и запустили его в эмуляторе.

В следующей главе вы поближе познакомитесь с Dart и получите хорошую базу, чтобы приступить к созданию реальных приложений Flutter!