Содержание

Предисловие

Я начинал как разработчик iOS в 2011 году. Мне нравилось создавать приложения для iPhone, и я до сих пор занимаюсь этим. Помимо разработки мобильных приложений, я также занимался веб-разработкой, изучая Ruby. Мне нравился этот короткий мощный язык, и я хотел использовать компилируемый язык, такой как Objective-C, но с элегантностью и выразительностью Ruby.

Затем компания Apple представила Swift, и похоже, они услышали меня. Для меня Swift был новым подходом к программированию, сочетая элегантность динамического языка со скоростью и безопасностью языка статического. Мне никогда не нравился синтаксис Objective-C. Раньше я говорил что-то вроде: «Да, Objective-C многословен, но он делает свою работу». Однако при работе со Swift я снова нахожу чтение и написание кода очень приятным занятием, как в случае с Ruby. Наконец-то я мог использовать статический язык и продолжать работать, при этом любя язык, с которым я работаю. Для меня это была хорошая комбинация.

Однако это не была любовь с первого взгляда. До того, как я действительно стал наслаждаться Swift, я много с ним боролся. Swift выглядит очень дружелюбно, но, боже, иногда было тяжело. На этапе компиляции все должно быть безопасно, и я больше не мог смешивать и сопоставлять типы в массивах. Между тем Swift был только ранней версией и постоянно менялся; было трудно идти в ногу с ним. «Что такое генераторы? О, их теперь называют итераторами? А зачем использовать оператор guard? Разве нельзя вместо этого использовать оператор if? Пфф, опционалы переоценены; можно использовать простую проверку на nil?» и т. д. Я даже не и думал о работе с обобщениями.

Тем не менее я выстоял и начал принимать эти концепции Swift. Я понял, что это были более старые концепции из других языков программирования, но в новом обличье, что действительно помогло мне лучше программировать и качественнее выполнять свою работу. Со временем я начал любить язык Swift и его милый синтаксис.

Начиная с Swift 2 я имел возможность поработать в большой компании, где мы производили код Swift в больших масштабах, начиная с команды, состоящей из примерно 20 разработчиков и заканчивая более чем 40. После работы со Swift при таком количестве разработчиков и после участия в сотнях запросов на принятие изменений я заметил, что у других разработчиков была такая же борьба, как и у меня. Либо мои коллеги писали просто отличный код, но не понимали, что от них скрыта более элегантная или надежная альтернатива, ждущая своего часа. Несмотря на то что наш код был верным, иногда он мог бы быть несколько чище, лаконичнее или чуть безопаснее. Я также заметил, что мы все держались подальше от мощных методов, таких как обобщения или метод `flatMap`, потому что их было трудно понять. Или же нам нравилась идея обобщений, но мы сами не знали, зачем и когда применять ее.

После этих реализаций я начал вести записи. Поначалу эти каракули были для меня заметками о том, как правильно извлекать опционалы, как работают ленивые свойства, как обращаться с обобщениями и т. д. Затем эти заметки созрели,

и, прежде чем я это понял, у меня было достаточно содержания для некоторых глав. Пришло время превратить эти заметки во что-то более сложное: книгу по программированию, которая может помочь другим сократить их путешествие по Swift. Имея на руках несколько сырых глав, я задавался вопросом, стоит ли мне выложить электронную книгу онлайн. Тем не менее при наличии «впечатляющих» 200 человек, подписавшихся на меня в Twitter, и не имея популярного блога, я решил, что не найду нужную аудиторию. Более того, я подумал, что мне нужно узнать много неизвестного о написании книги.

Я решил обратиться к издателю, чтобы он помог мне превратить эти грубые главы в большую книгу. Я обратился в издательство Manning, и с тех пор мы вместе работаем над книгой. Полагаю, что эти «маленькие заметки» превратились в нечто особенное. С помощью друзей Manning и Swift больше года я проводил большую часть своего свободного времени, сочиняя, полируя и пытаясь сделать сложные концепции Swift более простыми для понимания.

Я надеюсь, что когда вы будете читать эту книгу, она поможет вам стать мастером Swift. Надеюсь, что эта книга сделает ваше путешествие по Swift легким и увлекательным.

Чейрд ин't Вейн

Благодарности

Спасибо издательству Manning за то, что помогло мне опубликовать мою первую книгу.

Я хочу выразить особую благодарность Майку Стивенсу за то, что он воспользовался шансом, взяв меня на борт. Спасибо Хелен Стергиус за то, что работала со мной на протяжении всего этого процесса. Спасибо Алену Куньо за великолепные технические обзоры моих глав; нелегко постоянно указывать на ошибки и возможные улучшения; тем не менее я очень ценю это. И я также хочу поблагодарить Александра Павлицкого за его креативные мультипликационные иллюстрации, использованные на протяжении всей книги.

Мне очень помогли остальные члены команды Manning: Александар Драгосавлевич, Кэндис Гилхулли, Ана Ромак, Шерил Вейсман, Дейрдре Хиам, Дотти Марсико, Николь Борода, Мэри Пирджи, Кэрол Шилдс, Даррен Мейсс, Мелоди Долаб и Мария Тудор (Aleksandar Dragosavljević, Candace Gillhoolley, Ana Romac, Cheryl Weisman, Deirdre Hiam, Dottie Marsico, Nichole Beard, Mary Piergies, Carol Shields, Darren Meiss, Melody Dolab, Marija Tudor). Я хочу выразить особую благодарность друзьям, коллегам и тем, кто были моими подопытными свинками и рецензировали (частично) эту книгу, в частности это: Барт ден Холландер, Димитар Гюров, Дарио де Роза, Рейн Спейккерман, Янина Кутын, Сидни де Конинг, Торбен Шульц и Эдвин Чун Винг Квок (Bart den Hollander, Dimitar Gyurov, Dario de Rosa, Rein Spijkerman, Janina Kutyn, Sidney de Koning, Torben Schulz, and Edwin Chun Wing Kwok). Также я бы хотел поблагодарить других рецензентов: Алессандро Кампейса, Али Накви, Акселя Реста, Дэвида Джейкобса, Густаво Гомеса, Хельмута Райтерера, Джейсона Пайка, Джона Монтгомери, Кента Р. Шпилнера, Ларса Петерсена,

Марсело Пиреса, Марко Джузеппе Салафию, Мартин Филипп, Моника Гимарайнш, Патрик Риган, Тайлер Слэйтер и Вейерт де Бур (Alessandro Campeis, Ali Naqvi, Axel Roest, David Jacobs, Gustavo Gomes, Helmut Reiterer, Jason Pike, John Montgomery, Kent R. Spillner, Lars Petersen, Marcelo Pires, Marco Giuseppe Salafia, Martin Philp, Monica Guimaraes, Patrick Regan, Tyler Slater, and Weyert de Boer).

Я знал, что написание книги – дело непростое, но это было гораздо сложнее, чем я себе представлял. У нас с моей невестой Дженикой только что родилась дочь, и было довольно сложно создавать новую семью, проводить бессонные ночи, сохранить работу на полную ставку и писать книгу по программированию на втором языке. Я бы не смог этого сделать, если бы мне не нравилось писать эту книгу при поддержке своей невесты. Спасибо, Дженика, за то, что ты так терпелива ко мне.

Об этой книге

Swift – это молодой язык. На момент написания данных строк вышла четвертая версия, и она все еще не является ABI-стабильной, а это означает, что после выхода Swift 5 будут существенные изменения. Так почему же эта книга может рассказать вам, как написать свой код?

Ваш скептицизм был бы оправдан, но, пожалуйста, не торопитесь. Несмотря на то что Swift – относительно новый язык, я думаю, было бы справедливо сказать, что некоторые решения работают лучше, чем другие. Это еще важнее понимать, если вы используете Swift для реальных производственных приложений.

Swift заимствует множество важных концепций из других языков программирования, таких как Haskell, Ruby, Rust, Python, C # и др. Следовательно, было бы разумно следить за этими концепциями.

Сочетая парадигмы программирования с реальным опытом, эта книга делится очень забавными и полезными рекомендациями, которые вы можете незамедлительно применить в своей работе.

Программируя более десяти лет на разных языках и в разных командах, я хотел бы поделиться советами, хитростями и рекомендациями, которые очень помогли моей карьере в Swift, чего и вам желаю.

Почему эта книга?

Честно говоря, многие программы в этом мире работают на «некрасивом» коде, и это совершенно нормально. Если ваш продукт делает то, что нужно, это – нравится вам это или нет – вполне подойдет для бизнеса.

Будучи разработчиком, вы должны убедиться, что ваш продукт работает, и работает хорошо. Но ваши пользователи не будут заглядывать под капот и указывать на страшных операторов if. Перфекционизм вреден для разработки программного обеспечения и является причиной большого количества незавершенных проектов.

Тем не менее существует большой разрыв между «Он делает то, что нужно» и проектом, в котором были приняты отличные решения, которые окупаются в долгосрочной перспективе.

Работая над многочисленными проектами, я высоко ценю написание кода, который ваши коллеги и ваши будущие сотрудники будут *четко* понимать, потому что элегантный код означает меньше шансов на ошибки, повышенное удобство в обслуживании, лучшее понимание для разработчиков, которые наследуют код, море счастья для программистов и много других преимуществ.

Другой аспект, который я ценю, – это надежность кода, то есть то, насколько стойкими к рефакторингу являются некоторые элементы. Сломается, если чихнуть на него? Или можно ли изменить код без хлопот?

В этой книге я поделюсь своими советами, хитростями и рекомендациями, которые отлично подошли мне и компаниям, в которых я работал. Кроме того, она заполняет значительные пробелы в знаниях, которые могут возникнуть при работе со Swift.

Хотя это книга о Swift, многие принципы, о которых здесь говорится, не ориентированы на Swift и также распространяются на другие языки программирования, потому что Swift заимствует множество идей и парадигм из других языков. После того как вы закончите читать книгу, вам, возможно, будет легко применять эти концепции в других языках. Например, вы узнаете много нового об опционалах или о том, как использовать метод `reduce` при работе с массивами. Позже вы можете решить изучать Kotlin, где можно сразу же применять опционалы и вышеуказанный метод, где он носит название `fold`. Вы также можете обнаружить, что Rust – и его аналогичные реализации обобщений – проще освоить.

Из-за мультипарадигмальной природы Swift эта книга без предпочтения переключается между парадигмами объектно-ориентированного, функционального и протокольно-ориентированного программирования – хотя, по общему признанию, я все же предпочитаю другие методы, нежели создание подклассов. Переключение между этими парадигмами предлагает множество инструментов и решений проблемы, а также понимание того, почему определенное решение работает хорошо или нет. Если вы застряли в колее или открыты для множества новых идей программирования, эта книга ставит перед вами задачу решать проблемы различными способами.

Подходит ли вам эта книга?

Предполагается, что вы создали одно или несколько приложений на Swift. Вы работаете в команде? Еще лучше. Эта книга покажет вам, как написать хороший, понятный код, который будет оценен в командах, и поможет вам улучшить запросы на принятие изменений от других разработчиков. Ваш код будет более надежным и потребует меньше обслуживания от вас и вашей команды.

Эта книга заполняет пробелы в знаниях как для новичков, так и для опытных разработчиков Swift. Возможно, вы освоили протоколы, но все еще боретесь с использованием метода `flatMap` при работе с типами или асинхронной обработкой ошибок. Или, может быть, вы создаете красивые приложения, но держитесь подальше от обобщений, потому что их трудно интерпретировать. Или, может быть, вы знаете, когда использовать структуру вместо класса, но не знаете, что перечисления иногда являются лучшей альтернативой. В любом случае, эта книга по-

может вам с этими темами. К концу работа с обобщениями должна быть такой же естественной, как и в случае с циклами. Вы будете уверены в том, что будете вызывать метод `flatMap` для опционалов, будете знать, как работать с ассоциированными типами, и с радостью будете использовать метод `reduce` в своем повседневном труде при работе с итераторами.

Если вы планируете в будущем пройти собеседование на должность программиста, чтобы получить новое место, вам понравится. Вы сможете ответить на множество важных вопросов, касающихся компромиссов и решений относительно разработки на языке Swift. Эта книга может даже помочь вам написать элегантный код в заданиях.

Если вам просто нужно приложение из магазина приложений, продолжайте делать то, что делаете; нет необходимости читать эту книгу! Но если вы хотите написать более надежный, понятный код, который увеличивает ваши шансы получить место, преуспеть на работе или дать качественные комментарии по запросам на принятие изменений, вы попали в нужное место.

Чем эта книга не является

Эта книга ориентирована на Swift. В основном в ней используются примеры без фреймворков, потому что речь идет не об обучении Cocoa, iOS, Kitura или другим платформам и фреймворкам.

В этой книге я часто пользуюсь Apple Foundation, чего трудно избежать, если вам нужны примеры из реальной жизни. Если вы работаете в Linux, то можете использовать swift.org, чтобы получить аналогичные результаты.

Особый акцент на практических сценариях

Эта книга очень практична и демонстрирует советы и приемы, которые вы можете сразу же применять при повседневном программировании.

Не волнуйтесь: это не теоретическая книга. Вы узнаете много теории, но только благодаря использованию реальных проблем, с которыми рано или поздно сталкивается любой разработчик Swift. Тем не менее она не дотягивает до академического уровня, где обсуждается представление LLVM или машинный код.

Кроме того, я позаботился о том, чтобы избежать своей личной любимой мозоли: я не делю подкласс «Animal» с «Dog» и не добавляю протокол «Flyable» в «Bird». Я также не добавляю «Foo» в «Bar». Вы будете иметь дело с реальными сценариями, такими как общение с API, загрузка локальных данных, рефакторинг и создание функций, и увидите полезные фрагменты кода, которые можно реализовать в своих проектах.

Дорожная карта

В следующих разделах представлен обзор книги, разделенный на главы. Книга довольно модульная, и вы можете начать с любой главы, которая вас интересует.

Некоторые главы я считаю важными. Глава 4 «Делаем опционалы второй натурой» является ключевой, потому что опционы очень распространены в Swift, и мы будем возвращаться к ним снова и снова в других главах.

Чтобы понять абстрактную сторону Swift, я настоятельно рекомендую прочитать главу 7 «Обобщения», главу 8 «Становимся профессионалами в протокольно-ориентированном программировании» и главу 12 «Расширения протоколов». Эти главы закладывают прочную основу для ключевых навыков Swift. Обязательно прочтите их!

В качестве бонуса, если вы заинтересованы в изучении методов функционального программирования, обратите внимание на главу 2 «Моделирование данных с помощью перечислений», главу 10 «map, flatMap и compactMap» и главу 11 «Асинхронная обработка ошибок с помощью типа Result».

Глава 1: Введение

Эта глава – своего рода разминка. В ней рассказывается о текущем состоянии Swift, о его положительных сторонах и недостатках и о том, чем мы будем заниматься в ходе прочтения данной книги. Она не очень техническая, однако подготовит вас к тому, о чем пойдет речь далее.

Глава 2: Моделирование данных с помощью перечислений

Эта глава отлично подойдет, если вы хотите напрячь свой мозг, по-другому взглянуть на моделирование данных и увидеть, насколько далеко могут зайти перечисления, чтобы помочь вам.

Вы узнаете, как моделировать данные с помощью структур и перечислений и преобразовывать структуры в перечисления, и наоборот.

Вам будет предложено отойти от обычного подхода к классам, подклассам и структурам и посмотреть, как вместо этого моделировать данные с помощью перечислений и зачем это нужно.

Вы также познакомитесь с другими интересными способами использования перечислений и узнаете, как использовать их для написания более безопасного кода.

К концу этой главы вы можете поймать себя на мысли, что пишете гораздо больше перечислений.

Глава 3: Написание более чистых свойств

Swift обладает богатой системой свойств со множеством опций на выбор. Вы научитесь выбирать правильный тип свойств для правильных типов ситуаций. Вы также создадите чистые вычисляемые свойства и хранимые свойства с поведением.

Потом вы узнаете, когда использовать ленивые свойства, которые могут привести к незначительным ошибкам, если с ними обращаться неаккуратно.

Глава 4: Делаем опционалы второй натурой

Эта глава не оставит камня на камне касательно опциональных типов.

Опциональные типы настолько распространены, что в этой главе они рассматриваются очень подробно. Данная глава изобилует передовыми методами, советами и хитростями, которые улучшат ваш обычный код. Она будет полезна как новичкам, так и опытным разработчикам Swift.

Глава охватывает множество случаев использования опциональных типов, например при обработке опциональных логических типов, опциональных строк и перечислений, неявно извлекаемых опционалов и принудительном извлечении.

Глава 5: Разбираемся с инициализаторами

Жизнь в мире программирования начинается с инициализаторов. Спрятаться от них в Swift невозможно, и, конечно, вы уже работаете с ними. Тем не менее в Swift есть множество странных правил и ловушек, касающихся структур и классов и того, как инициализируются их свойства. Эта глава раскрывает эти странные правила, чтобы помочь вам избежать поединка с компилятором.

Это не просто теория; вы увидите, как можно написать меньше кода для инициализации, чтобы поддерживать чистоту своей кодовой базы, и получите понимание того, как создавать подклассы и как там применяются правила инициализаторов.

Глава 6: Непринужденная обработка ошибок

В этой книге есть две главы, посвященные обработке ошибок, где идет речь о двух разных идиомах: одна для синхронной обработки ошибок, а другая для асинхронной.

Данная глава посвящена синхронной обработке ошибок. Вы откроете для себя передовые методы, связанные с генерацией ошибок, их обработкой и поддержанием рабочего состояния ваших программ. В ней также рассказывается о распространении, добавлении технической информации и информации о пользователях, о преобразовании в NSError.

Помимо этого, вы узнаете, как сделать свои API-интерфейсы более приятными, заставляя их генерировать меньше ошибок, соблюдая при этом целостность приложения.

Глава 7: Обобщения

Обобщения – своего рода обряд посвящения для разработчиков Swift. Поначалу их трудно понять или работать с ними. Однако как только вы освоитесь, у вас появится искушение часто их использовать. В этой главе вы узнаете, когда и как применять их, создавая обобщенные функции и типы.

Вы увидите, как с помощью обобщений можно сделать код полиморфным, чтобы иметь возможность писать многократно используемые компоненты и в то же время сокращать кодовую базу.

Обобщения становятся еще более интересными, когда вы ограничиваете их протоколами для специализированной функциональности. Вы познакомитесь

с основными протоколами `Equatable`, `Comparable` и `Hashable` и узнаете, как смешивать и сопоставлять с ними обобщения.

После прочтения этой главы обобщения не будут чем-то пугающим, я обещаю.

Глава 8: Становимся профессионалами в протокольно-ориентированном программировании

Протоколы – напоминающие классы типов в Haskell или типажи в Rust – святой Грааль Swift. Поскольку Swift можно считать протокольно-ориентированным языком, в этой главе мы рассмотрим, как с пользой применять протоколы.

В ней рассказывается об обобщениях и показано, как они справляются с использованием протоколов в качестве типов. Вы сможете четко делать выбор (или переключаться) между ними. Протоколы с ассоциированными типами можно считать расширенными протоколами. В этой главе вы поймете, почему и как они работают, чтобы вам не пришлось воздерживаться от их использования. Здесь будет смоделирована часть программы с протоколами и будут приведены недостатки, которые в конечном итоге решаются с помощью ассоциированных типов.

Затем вы узнаете, как передавать протоколы с ассоциированными типами в функции и типы, чтобы иметь возможность создавать чрезвычайно гибкий, но абстрактный код.

В этой главе много внимания уделено тому, как использовать протоколы на этапе компиляции (статическая отправка) и во время выполнения (динамическая отправка), а также связанным с ними компромиссам. Эта глава призвана обеспечить надежную основу для протоколов, чтобы вы могли использовать более сложные шаблоны в последующих главах.

Глава 9: Итераторы, последовательности и коллекции

Нередко в Swift создается структура данных, которая использует не только основные типы, такие как наборы, массивы и словари. Возможно, вам потребуется создать специальное хранилище для кеширования или систему разбивки на страницы при загрузке канала Twitter.

Структуры данных часто приводятся в действие протоколами `Collection` и `Sequence`. Вы увидите, что протокол Sequence, в свою очередь, использует протокол `IteratorProtocol`. Применяя сочетание этих протоколов, можно расширять и реализовывать основные функции в своих типах данных.

Вначале вы увидите, как работает итерация с протоколами `IteratorProtocol` и `Sequence`. Вы познакомитесь с полезными шаблонами итераторов, такими как `reduce()`, `reduce(into :)` и `zip`, а также увидите, как работают ленивые последовательности (`lazy sequences`).

Вы создадите структуру данных под названием сумка, также известную как мультимножество, используя протокол `Sequence`.

Затем вы познакомитесь с протоколом `Collection` и описанием всех протоколов коллекций, предлагаемых Swift.

В завершение вы создадите еще одну структуру данных и узнаете, как привести ее в соответствие с протоколом `Collection`. В этой части много практики, и вы сможете сразу же использовать эти же методы в своем коде.

Глава 10: map, flatMap и compactMap

В этой главе освещаются ключевые понятия, обычно встречающиеся не только в Swift, но и в других фреймворках и языках программирования.

Рано или поздно вы столкнетесь с методами `map`, `flatMap` и `compactMap` в массивах, опционалах, типах ошибок и, возможно, даже функциональном реактивном программировании, например RxSwift.

Вы получите правильное представление о том, как очистить код, применив методы `map` и `flatMap` к опциональным типам, а также узнаете, как использовать метод `map` при работе со словарями, массивами и другими типами коллекции, и познакомитесь с преимуществами применения метода `flatMap` при работе со строками.

Наконец, вы сможете ознакомиться с методом `compactMap` и его элегантной обработкой опциональных типов в коллекциях.

Знание методов `map`, `flatMap` и `compactMap` на более глубоком уровне является хорошей основой для понимания того, как читать и писать более краткий, но элегантный код, и для работы с типом `Result` в главе 11.

Глава 11: Асинхронная обработка ошибок с помощью типа Result

Обработка ошибок в Swift немного отстает от асинхронной обработки ошибок. Вы познакомитесь с этим поближе и узнаете, как обеспечить безопасность на этапе компиляции для асинхронного программирования, воспользовавшись так называемым типом `Result`, который неофициально предлагается компанией Apple через менеджер пакетов Swift.

Возможно, вы уже используете какую-то версию типа `Result`, найденную во фреймворках. Но даже если вы знакомы с ним, я готов поспорить, что в этой главе вы увидите новые и полезные методы.

Вы начнете с изучения недостатков традиционной обработки ошибок в стиле Cocoa и того, почему `Result` может помочь вам в этом. Затем увидите, как преобразовать традиционный вызов в тот, что использует `Result`.

Кроме того, вы увидите преобразование генерирующих функций в `Result` и обратно. Вы будете применять специальный тип `AnyError` для создания большей гибкости, избегая NSError и гарантируя повышенную безопасность на этапе компиляции.

Вы познакомитесь с типом `Never`, который является уникальным способом сообщить компилятору Swift, что у `Result` ничего не получается или он потерпел неудачу.

Наконец, вы будете использовать то, чему научились при применении методов `map` и `flatMap` с опционалами, чтобы понять, как применять метод map при работе со значениями и ошибками и даже как использовать метод `flatMap` с типом

`Result`. В результате вы получите так называемый монадический стиль обработки ошибок, который дает возможность очень аккуратно и элегантно распространять ошибки в стеке вызовов с очень небольшим количеством кода, сохраняя при этом повышенную безопасность.

Глава 12: Расширения протоколов

Эта глава полностью посвящена моделированию данных с помощью уменьшения связанности (decoupling). В ней предлагаются варианты реализации по умолчанию посредством протоколов. Мы будем использовать умные переопределения и узнаем, как расширить типы интересными способами.

Для начала вы узнаете о моделировании данных с использованием протоколов по сравнению с подклассами.

Затем вы будете моделировать данные, используя два подхода: один подход предполагает наследование протокола, а другой использует композицию протокола. У обоих есть свои плюсы и минусы, которые вы обнаружите, когда перейдете к рассмотрению связанных с ними компромиссов.

Кроме того, вы увидите, как работают расширения протокола, если они переопределены наследованием протокола и конкретными типами. Это несколько гипотетично, но полезно, чтобы понимать протоколы на более глубоком уровне.

Вы также увидите, как осуществлять расширение в двух направлениях. Одно направление – это расширение класса, чтобы соответствовать протоколу, а второе – расширение протокола и ограничение его классом. Это тонкое, но важное отличие.

В конце главы вы расширите протокол `Collection`, а затем пойдете дальше и расширите протокол `Sequence` для создания расширений с возможностью многократного использования. Вы познакомитесь с `ContiguousArray` и функциями с ключевым словом `rethrows`, а также создадите полезные методы, которые сможете напрямую применять в своих проектах.

Глава 13: Шаблоны Swift

Это, возможно, самая трудная глава в книге, но это вершина, которую стоит покорить.

Цель данной главы – справиться с распространенными препятствиями, с которыми вы можете столкнуться. Шаблоны, описанные здесь, не являются перефразировкой принципов SOLID – об этом написано множество книг! Вместо этого она фокусируется на современных подходах к современному языку.

Вы узнаете, как мокировать API с помощью протоколов и ассоциированных типов, – что часто бывает удобно, – чтобы иметь возможность создавать автономную и тестовую версии API.

Затем вы увидите, как работает условное соответствие касательно обобщенных типов и протоколов с ассоциированными типами. После этого создадите обобщенный тип и запустите его, используя мощную технику условного соответствия, которая является еще одним способом написания очень гибкого кода.

Далее вы будете иметь дело с проблемой, с которой можно столкнуться при попытке использовать протокол в качестве конкретного типа. Для борьбы с этим вы будете использовать два метода: один включает в себя перечисления, а второй – продвинутую технику под названием стирание типов.

Наконец, вы также проверите, можно ли считать протоколы хорошим выбором. Вопреки распространенному мнению протоколы – не всегда то, что нужно. Вы найдете альтернативный способ создания гибкого типа, включающий в себя структуру и функции высшего порядка.

Глава 14: Написание качественного кода на языке Swift

Это наименее ориентированная на код глава, но, возможно, одна из самых важных.

Здесь рассказывается о написании чистого, понятного кода, который создает меньше головной боли для всех, кто работает в вашей команде (если вы являетесь ее членом). Она ставит перед вами задачу создания соглашений об именах, добавления документации и комментариев, а также разделения больших классов на небольшие обобщенные компоненты. Вы также настроите SwiftLint, инструмент, который добавляет согласованность стилей и помогает избежать ошибок в проектах, а также познакомитесь с архитектурой и тем, как преобразовать большие классы со слишком большим количеством обязанностей в более мелкие обобщенные типы.

Эта глава – неплохая проверка, чтобы увидеть, соответствует ли ваш код стандартам и стилям, что поможет при создании запросов на принятие изменений или завершении задания при собеседовании на новой работе.

Глава 15: Что дальше?

На этом этапе ваши навыки в Swift будут достаточно сильными. Я поделюсь с вами несколькими советами о том, куда двигаться дальше, чтобы вы могли продолжить свое путешествие.

О коде

Эта книга содержит много примеров исходного кода, как в пронумерованных листингах, так и в обычном тексте. В обоих случаях исходный код форматируется `шрифтом фиксированной ширины, как этот`, чтобы отделить его от обычного текста. Иногда код также выделяется **жирным шрифтом**, чтобы выделить то, что изменилось по сравнению с предыдущими шагами в этой главе, например когда в существующую строку кода добавляется новая функция.

Во многих случаях оригинальный исходный код был переформатирован. Мы добавили разрывы строк и переработали отступы, чтобы разместить доступное пространство страницы в книге. В редких случаях даже этого было недостаточно, и списки содержат маркеры продолжения строки (➥). Кроме того, комментарии в исходном коде часто удалялись из списков при описании кода в тексте. Аннотации к кодам сопровождают многие списки, выделяя важные понятия.

Исходный код для всех листингов, приведенных в этой книге, можно загрузить с сайта издательства Manning по адресу https://www.manning.com/books/swift-in-depth и с GitHub на странице https://github.com/tjeerdintveen/manning-swift-in-depth. В каждую главу включен исходный код, за исключением глав 14 и 15.

Книжный форум

Приобретение данной книги включает в себя бесплатный доступ к частному веб-форуму, организованному Manning Publications, где можно оставить комментарии о книге, задать технические вопросы и получить помощь от авторов и других пользователей. Чтобы получить доступ к форуму, перейдите по ссылке https://forums.manning.com/forums/swift-in-depth. Вы также можете узнать больше о форумах Manning и правилах поведения на странице https://forums.manning.com/forums/about. Обязательство издательства по отношению к нашим читателям состоит в том, чтобы обеспечить пространство, где может иметь место содержательный диалог между отдельными читателями и между читателями и авторами. Это не обязательство какого-либо конкретного участия со стороны авторов, чей вклад в форум остается добровольным (и не оплачивается). Мы предлагаем вам попробовать задать авторам несколько сложных вопросов, чтобы их интерес не угас! Форум и архив предыдущих обсуждений будут доступны на сайте издателя, пока книга находится в печати.

Об авторе

Чейрд ин’т Вейн – заядлый фанат Swift и внештатный разработчик для iOS с опытом работы в агентствах. Он является соучредителем небольшого стартапа и на момент написания этих строк помогает ING расширять их мобильные разработки. Начав свою карьеру в качестве разработчика Flash в 2001 году, он разрабатывал для iOS с помощью Objective-C, занимался веб-разработкой на Ruby и немного освоил другие языки программирования.

Когда он не работает, то проводит время с двумя своими дочерьми, отпуская шутки и балуясь с акустической гитарой.

Вы можете найти его в Twitter (@tjeerdintveen).

Об иллюстрации на обложке

Изображение на обложке книги озаглавлено «Человек из Омишаля, остров Крк, Хорватия». Эта иллюстрация взята из репродукции, опубликованной в 2006 году, из коллекции костюмов и этнографических описаний XIX века под названием *Далмация* профессора Франэ Каррара (1812–1854), археолога и историка, первого директора Музея древности в Сплите, Хорватия. Иллюстрации были получены от услужливого библиотекаря из Этнографического музея (бывший Музей античности), который расположен в римском ядре средневекового центра Сплита: руинах

дворца императора Диоклетиана, датируемого примерно 304 г. н. э. Книга содержит прекрасные цветные иллюстрации жителей разных регионов Далмации, сопровождаемые описаниями костюмов и быта.

С XIX века дресс-код изменился, и разнообразие регионов, столь богатое в то время, исчезло. Сейчас трудно отличить жителей разных континентов, не говоря уже о разных городах или регионах. Возможно, мы обменяли культурное разнообразие на более разнообразную личную жизнь – конечно, на более разнообразную и быстро развивающуюся технологическую жизнь.

В то время когда трудно отличить одну компьютерную книгу от другой, издательство Manning празднует изобретательность и инициативу компьютерного бизнеса, используя обложки, основанные на богатом разнообразии жизни регионов двухвековой давности, оживленной иллюстрациями из коллекций, подобной этой.

Предисловие от издательства

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 по адресу `http://dmkpress.com/authors/publish_book/` или напишите в издательство по адресу `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 этой 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.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications 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 электронной почты dmkpress@gmail.com со ссылкой на подозрительные материалы. Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Глава 1. Введение

В этой главе:

- краткий обзор популярности Swift и поддерживаемых платформ;
- преимущества Swift;
- более внимательный взгляд на тонкие недостатки Swift;
- о чем мы узнаем в этой книге.

Не секрет, что Swift поддерживается на многих платформах, таких как Apple iOS, macOS, watchOS и tvOS. Swift – это язык с открытым исходным кодом. Он также работает в Linux и набирает популярность на стороне сервера в таких веб-фреймворках, как Vapor, Perfect, Zewo и Kitura от IBM.

Кроме того, Swift медленно не только охватывает прикладное программирование (программное обеспечение для пользователей), но и начинает вводить системное программирование (программное обеспечение для систем), такое как SwiftNIO или инструменты командной строки. Swift превращается в мультиплатформенный язык. При изучении Swift перед вами открывается множество дверей.

Swift был самым популярным языком на сайте Stack overflow в 2015 году и остается в пятерке самых любимых языков в 2017 году. В 2018 году он поднялся на 6-е место (https://insights.stackoverflow.com/survey/2018).

Swift, безусловно, готов остаться, и если вы любите создавать приложения, веб-сервисы или другие программы, моя цель состоит в том, чтобы вы извлекли как можно больше пользы из этой книги как для себя, так и для своей команды.

Что мне нравится в Swift, так это то, что его легко изучать, но сложно овладеть. Всегда есть чему поучиться! Одна из причин заключается в том, что Swift включает в себя множество парадигм программирования, позволяя вам выбрать подходящее решение проблемы программирования, что вы и будете исследовать в этой книге.

1.1. «Золотая середина» SWIFT

Что мне нравится в динамическом языке, таком как Ruby, так это его выразительность. Ваш код говорит вам, чего вы хотите достичь, не слишком занимаясь управлением памятью и низкоуровневыми технологиями. Написание кода на Ruby один день и на Objective-C в другой убедили, что мне нужно было выбирать между скомпилированным языком, который работал хорошо, или экспрессивным, динамическим языком за счет более низкой производительности.

Тогда появился Swift и разрушил это заблуждение. Swift находит правильный баланс, когда он совместно использует множество преимуществ динамических языков, таких как Ruby или Python, предлагая дружественный синтаксис и сильный полиморфизм. Тем не менее Swift избегает некоторых недостатков динами-

ческих языков, в частности производительности, потому что Swift компилируется в машинный код с помощью компилятора LLVM. Поскольку Swift компилируется с помощью LLVM, вы получаете не только высокую производительность, но и тонны проверок безопасности, оптимизаций и гарантии того, что ваш код в порядке, прежде чем запускать его. В то же время Swift выполняет чтение как выразительный динамический язык, с которым приятно работать и легко выражать свои намерения.

Swift не может помешать вам делать ошибки или писать плохой код, но он помогает уменьшить программные ошибки на этапе компиляции, используя различные методы, включая статическую типизацию и сильную поддержку алгебраических типов данных (перечисления, структуры и кортежи), но не ограничиваясь ими. Swift также предотвращает ошибки null благодаря опциональным типам.

Недостатком некоторых статических языков является то, что вам всегда нужно определять типы. Swift делает этот процесс проще с помощью вывода типов и может выводить конкретные типы, когда это имеет смысл. Таким образом, вам не нужно явно указывать каждую переменную, константу и обобщение.

Swift представляет собой смесь различных парадигм программирования, поскольку независимо от того, используете ли вы объектно-ориентированный подход, функциональное программирование или привыкли работать с абстрактным кодом, Swift предлагает все это. В качестве основного убедительного аргумента Swift обладает надежной системой, когда речь заходит о полиморфизме, в форме обобщений и *протокольно-ориентированного программирования*, которое активно используется в качестве маркетингового инструмента как компанией Apple, так и разработчиками (см. рис. 1.1).

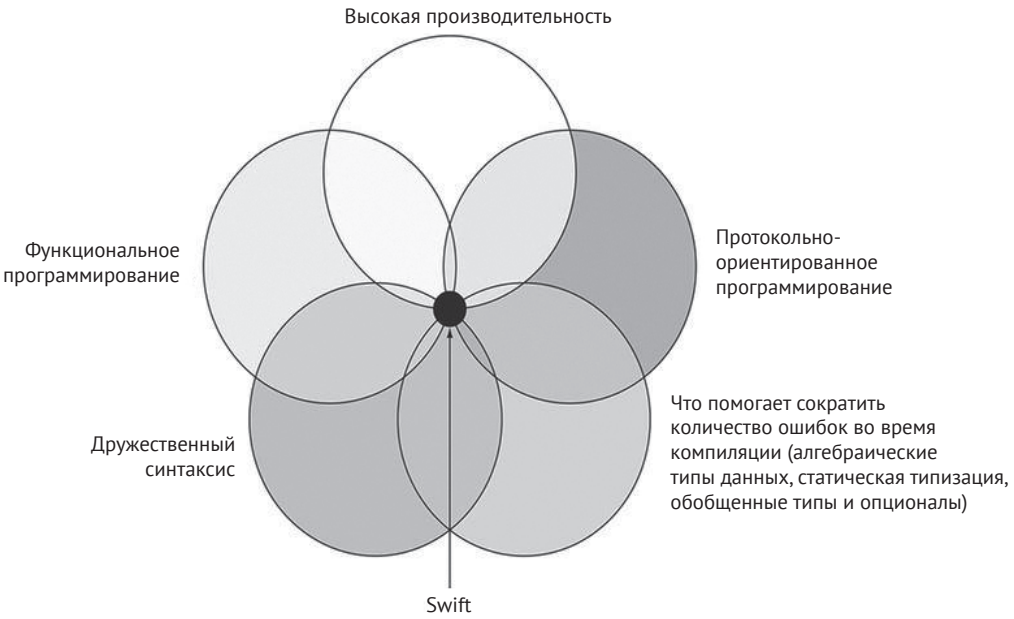

Рис. 1.1. «Золотая середина» Swift

1.2. Под поверхностью

Несмотря на то что Swift впечатляет своим дружественным синтаксисом и обещает создавать потрясающие приложения, это лишь верхушка айсберга. Соблазнительный вход в изучение Swift – это начать с разработки для iOS, поэтому вы узнаете, как создавать красивые приложения, которые состоят из важнейших компонентов из фреймворков Apple.

Но как только вам нужно будет самостоятельно доставить компоненты и приступить к созданию более сложных систем и структур, вы поймете, что Swift усердно работает над тем, чтобы скрыть многие сложности, и делает это успешно. Когда вам понадобится изучить эти сложности, и вы это сделаете, кривая сложности Swift возрастет в геометрической прогрессии. Даже самые опытные разработчики ежедневно обучаются новым трюкам и хитростям Swift!

Как только Swift вас зацепит, вы, скорее всего, столкнетесь со скачками скорости в форме обобщений и ассоциированных типов, и нечто такое «простое», как обработка строк, может вызвать больше проблем, чем вы могли ожидать (см. рис. 1.2).

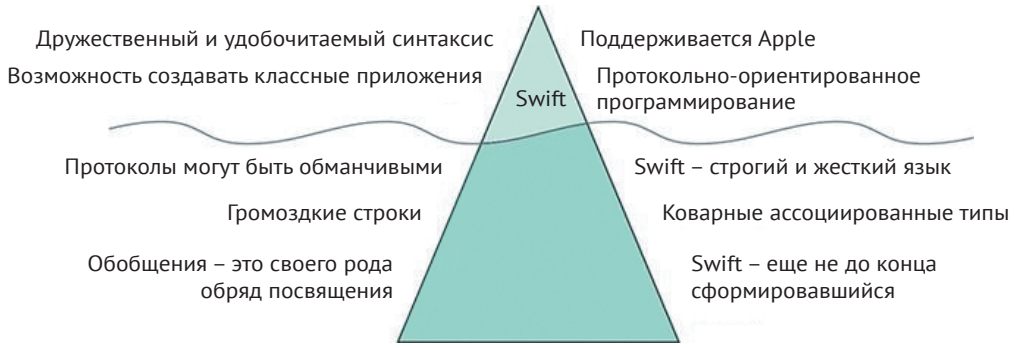

Рис. 1.2. Верхушка айсберга Swift

Эта книга помогает справиться с самыми распространенными сложностями. Вы будете освещать и решать любые проблемы и недостатки с помощью Swift и сможете использовать силы, которые привносят эти сложности, весело проводя время.

1.3. Минусы Swift

Swift – мой любимый язык программирования, но давайте не будем смотреть на него сквозь розовые очки. Как только вы признаете сильные и слабые стороны Swift, сможете адекватно решить, когда и как вы бы хотели его использовать.

1.3.1. Стабильность ABI

Swift все еще движется быстро и не является ABI-стабильным, а это означает, что код, написанный на Swift 4, не будет совместим с Swift 5, и наоборот. Представь-

те, что вы пишете фреймворк для своего приложения. Как только выйдет Swift 5, приложение, написанное на Swift 5, не сможет использовать ваш фреймворк, пока вы не обновите свой фреймворк до Swift 5. К счастью, Xcode сильно помогает в случае с миграцией, поэтому я ожидаю, что эта миграция не будет такой болезненной.

1.3.2. Строгость

Swift – строгий и жесткий язык, что является распространенной критикой статических языков, тем более при работе со Swift. Прежде чем освоиться, у вас может создаться впечатление, словно вы печатаете в наручниках. На практике вам нужно разрешить многие типы на этапе компиляции, например с помощью опциональных типов, смешивания значений в коллекциях или обработки перечислений.

Я бы сказал, что строгий характер Swift является одной из его сильных сторон. Как только вы пытаетесь компилировать, вы узнаете о коде, который не работает, а не о том, что клиент столкнулся с ошибкой во время выполнения. После того как вы сделали первоначальные инвестиции, чтобы освоиться со Swift, они станут естественными, и их ограниченность меньше будет вас останавливать. Эта книга поможет вам преодолеть себя, и Swift станет вашей второй натурой.

1.3.3. Сложность протоколов

Протоколы – это ключевой момент Swift. Но как только вы начнете работать с протоколами, будете задевать острые края и время от времени наносить себе порезы. То, что «должно просто работать», почему-то не работает. Протоколы хороши в своем текущем состоянии и уже достаточно хороши для создания качественного программного обеспечения, но иногда вы сталкиваетесь с трудностями, и вам приходится использовать обходные пути, о которых много говорится в этой книге.

Вот распространенный источник разочарования: если вам нужно передать типы, соответствующие протоколу `Equatable`, в функцию, чтобы увидеть, равны ли они, вас остановят. Например, можно попытаться проверить, равно ли значение всему массиву, как показано в приведенном ниже листинге. Вы увидите, что в Swift это не сработает.

Листинг 1.1. Попытка проверки равенства

```
areAllEqual(value: 2, values: [3,3,3,3])

func areAllEqual(value: Equatable, values: [Equatable]) -> Bool {
   guard !values.isEmpty else { return false }

   for element in values {
      if element != value {
         return false
      }
```

```
    }

    return true
}
```

Swift возвращает загадочную ошибку с неопределенным предложением относительно дальнейших действий:

```
error: protocol 'Equatable' can only be used as a generic constraint
because it has Self or associated type requirements
```

Вы поймете, почему это происходит и как избежать этих проблем, в главах 7 и 8.

Расширения протокола Swift – еще одно основное преимущество и одна из самых мощных функций, которые он может предложить. Протоколы могут действовать как интерфейсы. Расширения протокола предлагают реализации по умолчанию для типов, помогая вам избежать жесткого деления на подклассы.

Протоколы, однако, хитрее, чем может показаться, и могут удивить даже опытного разработчика. Например, допустим, у вас есть протокол под названием `FlavorType`, обозначающий продукт или напиток, который вы можете улучшить с помощью аромата, например кофе. Если вы расширите этот протокол с помощью реализации по умолчанию, которая *не* найдена в объявлении протокола, то можете получить удивительные результаты! Обратите внимание, что в следующем листинге у вас есть два типа `Coffee`, но они оба дают разные результаты при вызове для них `addFlavor` . Это небольшая, но важная деталь.

Листинг 1.2. Протоколы могут вас удивить

```
protocol FlavorType{
    // func addFlavor()
    // Мы получим разные результаты, если этот метод не существует.
}

extension FlavorType {
    func addFlavor() { // Создаем реализацию по умолчанию.
        print(«Adding salt!»)
    }
}

struct Coffee: FlavorType {
    func addFlavor() { // Структура Coffee предоставляет свою реализацию.
        print(«Adding cocoa powder»)
    }
}

let tastyCoffee: Coffee = Coffee() // tastyCoffee имеет тип 'Coffee'
tastyCoffee.addFlavor() // Добавляем какао-порошок

let grossCoffee: FlavorType = tastyCoffee // grossCoffee имеет тип FlavorType
grossCoffee.addFlavor() // Добавляем соль
```

Даже если вы имеете дело с кофе того же типа, сначала вы добавляете какао-порошок, а затем случайно добавляете соль, которая не помогает никому вставать по утрам. Какими бы эффективными ни были протоколы, иногда они могут вносить незначительные ошибки.

1.3.4. Параллелизм

Наши компьютеры и устройства – это параллельные машины, использующие несколько процессоров одновременно. Работая в Swift, вы уже можете выражать параллельный код через Grand Central Dispatch (GCD). Но языковая особенность параллелизма в Swift не существует.

Поскольку вы уже можете использовать GCD, не такая уж большая проблема, чтобы немного подождать подходящую модель параллелизма. Тем не менее GCD в сочетании со Swift не безупречен.

Во-первых, работа с GCD может создать так называемую пирамиду гибели, также известную как глубоко вложенный код, о чем свидетельствует кусок незавершенного кода в листинге 1.3.

Листинг 1.3. Пирамида гибели

```
func loadMessages(completion: (result: [Message], error: Error?) -> Void) {
   loadResource(«/user») { user, error in
      guard let data = data else {
         completion(nil, error)
         return
      }
      loadResource(«/messages/», user.id) { messages, error in
         guard let messages = messages else {
            completion(nil, error)
            return
         }
         storeMessages(messages) { didSucceed, error in
            guard let error != nil else
            {
               completion(nil, error)
               return
            }
            DispatchQueue.main.async { // При удалении элементов вы
            // уменьшаете их число
               completion(messages)
            }
         }
      }
   }
}
```

Во-вторых, вы не знаете, какая очередь вызывается асинхронным кодом. Если вам нужно вызвать `loadMessages`, у вас могут быть проблемы, если небольшое изменение перемещает блок завершения в фоновую очередь. Вы можете быть очень осторожны и завершить обратный вызов в главной очереди в месте вызова, но в любом случае есть компромисс.

В-третьих, обработка ошибок является неоптимальной и не соответствует модели Swift. И возвращаемые данные, и ошибка теоретически могут быть заполнены или равны нулю. Из кода не ясно, что это может быть только одно или другое. Мы рассмотрим это в главе 11. Вы можете ожидать модель асинхронного ожидания в Swift более поздних версий, возможно, версии 7 или 8, а это означает ожидание, пока эти проблемы не будут решены. К счастью, GCD более чем достаточно для большинства ваших потребностей, и вы можете обратиться к RxSwift в качестве альтернативы для реактивного программирования.

1.3.5. Отход от платформ Apple

Swift выходит на новую территорию для интернета и в качестве системного языка, особенно благодаря поддержке IBM в форме веб-сервера Kitura и переводу Swift в облако. Отказ от платформ Apple открывает захватывающие возможности, но имейте в виду, что вы можете столкнуться с нехваткой пакетов, которые могли бы помочь вам в этом. Для таких xOS-фреймворков, как iOS, watchOS, tvOS и macOS, можно использовать CocoaPods или Carthage, чтобы обрабатывать зависимости. За пределами семейства xOS вы можете использовать менеджер пакетов Swift, предлагаемый компанией Apple. Однако многие разработчики Swift сосредоточены на iOS, и вы можете столкнуться с заманчивым пакетом без поддержки диспетчера пакетов Swift.

Хотя нелегко соответствовать существующим экосистемам, таким как тысячи пакетов Python, npm из Node.js и Ruby gems, это также зависит от вашей перспективы. Отсутствие пакетов может быть сигналом того, что вы можете внести свой вклад в сообщество и экосистему, изучая Swift.

Хотя Swift – это язык с открытым исходным кодом, Apple держит руль. Вам не нужно беспокоиться о том, что Apple прекратит поддержку Swift, но не всегда можете соглашаться с направлением или скоростью обновлений Swift. Вы все еще должны зависеть от сторонних инструментов, чтобы получить зависимости, работающие для iOS и macOS, а Xcode пока плохо интегрируется с Swift Package Manager. К сожалению, обе эти проблемы, по-видимому, имеют низкий приоритет для Apple.

1.3.6. Время компиляции

Swift – высокопроизводительный язык. Но процесс компиляции может быть довольно медленным и отнимать много времени у разработчиков. Swift компилируется в несколько этапов с помощью компилятора LLVM. Хотя это дает оптимизацию при запуске кода, в повседневном программировании вы можете столкнуться с медленной сборкой, что может быть немного утомительно, если вы пытаетесь быстро протестировать работающий фрагмент кода. Также не каждый проект

Swift представляет собой отдельный проект. Как только вы включите несколько фреймворков, вы будете компилировать много кода для создания сборки, замедляя свой процесс.

В конце концов, у каждого языка программирования есть свои плюсы и минусы – это вопрос выбора правильного инструмента для работы. Я верю, что у Swift большое будущее, которое поможет вам создать красивое программное обеспечение с чистым, сжатым и высокопроизводительным кодом!

1.4. Чему вы научитесь

Цель этой книги – показать вам, как решать проблемы элегантными способами, применяя передовые методы.

Несмотря на то что на обложке этой книги написано Swift, я думаю, что одним из ее сильных сторон является то, что вы изучите концепции, которые легко перенести на другие языки. Вы узнаете о:

- концепциях функционального программирования, таких как `Reduce`, `FlatMap`, `Optional` и `Result`, и о том, как мыслить алгебраическими типами данных, используя структуры, кортежи и перечисления;
- многих реальных сценариях, к которым вы подходите с разных сторон, рассматривая плюсы и минусы каждого из подходов;
- обобщениях, охватывая полиморфизм на этапе компиляции;
- том, как написать более надежный, лаконичный и легко читаемый код;
- протокольно-ориентированном программировании, включая полное понимание ассоциированных типов, которые считаются самой сложной частью протоколов;
- работе Swift во время выполнения и на этапе компиляции с использованием обобщений, перечислений и протоколов;
- способе найти компромисс между функциональным, объектно-ориентированным и протокольно-ориентированным программированием.

В конце этой книги ваш пояс с инструментами будет тяжелым из-за всех этих вариантов, которые вам придется решать при программировании. После того как вы усвоите эти концепции, вы, возможно, обнаружите, что изучать другие языки, такие как Kotlin, Rust и подобные, которые разделяют схожие идеи и функциональность, намного проще.

1.5. Как извлечь максимум из этой книги

Отличный способ выучить язык программирования – выполнять упражнения перед чтением главы. Будьте честны и посмотрите, сможете ли вы *по-настоящему* закончить их, вместо того чтобы смотреть на них и думать, что вы уже знаете ответ. В иных упражнениях могут быть скрыты некоторые сложные ситуации, и вы увидите их, как только начнете над ними работать.

Выполнив упражнения, решите, хотите ли вы прочитать главу, чтобы узнать новый материал.

В этой книге материал идет потоком, и по мере прочтения его сложность растет. Тем не менее книга построена по модульному принципу. Можно читать главы не по порядку.

1.6. Минимальная квалификация

Эта книга не рассчитана на абсолютного новичка. Предполагается, что вы немного работали с Swift раньше.

Если вы считаете себя продвинутым новичком или на среднем уровне, большинство глав будет вам полезно. Если вы считаете себя опытным разработчиком, я все же полагаю, что многие главы будут полезны, дабы заполнить пробелы в ваших знаниях. Благодаря модульности этой книги вы можете выбирать главы, которые хотели бы прочитать, не читая предыдущих.

1.7. Версия Swift

Эта книга написана для Swift версии 4.2. Все примеры будут работать на этой версии либо в командной строке, либо в сочетании с Xcode 10.

Резюме

- Swift поддерживается на многих платформах.
- Swift часто используется для разработки под iOS, watchOS, tvOS и macOS и с каждым днем все больше и больше для веб-разработки, системного программирования, инструментов командной строки и даже машинного обучения.
- Swift проводит тонкую грань между высокой производительностью, удобочитаемостью и безопасностью на этапе компиляции.
- Swift легок в изучении, но его трудно освоить.
- Swift – безопасный и высокопроизводительный язык.
- В Swift нет встроенной поддержки параллелизма.
- Диспетчер пакетов Swift пока еще не работает для iOS, watchOS, tvOS или macOS.
- Swift включает в себя несколько стилей программирования, таких как функциональное программирование, объектно-ориентированное программирование и программирование протоколов.