

Содержание

Предисловие	14
Благодарности	15
Об этой книге	16
Почему эта книга?	16
Подходит ли вам эта книга?	17
Чем эта книга не является	18
Особый акцент на практических сценариях	18
Дорожная карта	18
О коде	24
Книжный форум	25
Об авторе	25
Об иллюстрации на обложке	25
Предисловие от издательства	26
Отзывы и пожелания	26
Список опечаток	26
Нарушение авторских прав	26
Глава 1. Введение	28
1.1. «Золотая середина» SWIFT	28
1.2. Под поверхность	30
1.3. Минусы Swift	30
1.3.1. Стабильность ABI	30
1.3.2. Строгость	31
1.3.3. Сложность протоколов	31
1.3.4. Параллелизм	33
1.3.5. Отход от платформ Apple	34
1.3.6. Время компиляции	34
1.4. Чему вы научитесь	35
1.5. Как извлечь максимум из этой книги	35
1.6. Минимальная квалификация	36
1.7. Версия Swift	36
Глава 2. Моделирование данных с помощью перечислений	37
2.1. OR в сравнении с AND	38
2.1.1. Моделирование данных с помощью структуры	38
2.1.2. Превращаем структуру в перечисление	41
2.1.3. Выбор между структурами и перечислениями	43
2.2. Перечисления для полиморфизма	44
2.2.1. Полиморфизм на этапе компиляции	44
2.3. Перечисления вместо создания подклассов	46
2.3.1. Формирование модели для приложения Workout	47
2.3.2. Создание суперкласса	48
2.3.3. Недостатки подклассов	48
2.3.4. Рефакторинг модели данных с помощью перечислений	49

2.3.5. Подклассы или перечисления – что выбрать	51
2.3.6. Упражнения	51
2.4. Алгебраические типы данных	51
2.4.1. Типы-суммы	52
2.4.2. Типы-произведения	53
2.4.3. Распределение суммы в перечислении	53
2.4.4. Упражнение	55
2.5. Более безопасное использование строк	56
2.5.1. Опасность необработанных значений	57
2.5.2. Сопоставление для строк	59
2.5.3. Упражнения	62
2.6. В заключение	62
Глава 3. Написание более чистых свойств	66
3.1. Вычисляемые свойства	66
3.1.1. Моделирование упражнения	67
3.1.2. Преобразование функций в вычисляемые свойства	68
3.1.3. Завершение	70
3.2. Ленивые свойства	70
3.2.1. Создание учебного плана	70
3.2.2. Когда вычисляемые свойства не помогают	71
3.2.3. Использование ленивых свойств	73
3.2.4. Делаем ленивое свойство устойчивым	74
3.2.5. Изменяемые и ленивые свойства	75
3.2.6. Упражнения	77
3.3. Наблюдатели свойств	79
3.3.1. Обрезка пробелов	79
3.3.2. Запуск наблюдателей свойств из инициализаторов	81
3.3.3. Упражнения	82
3.4. В заключение	83
Глава 4. Делаем опционалы второй натурой	87
4.1. Назначение опционалов	88
4.2. Чистое извлечение значений	89
4.2.1. Сопоставление для опционалов	90
4.2.2. Методы извлечения	91
4.2.3. Когда значение вас не интересует	92
4.3. Соккрытие переменной	93
4.3.1. Реализация протокола CustomStringConvertible	93
4.4. Когда опционалы запрещены	94
4.4.1. Добавление вычисляемого свойства	95
4.5. Возврат опциональных строк	96
4.6. Детальный контроль над опционалами	98
4.6.1. Упражнения	99
4.7. Откат назад, если опционал равен nil	99
4.8. Упрощение опциональных перечислений	99
4.8.1. Упражнение	101

4.9. Цепочки опционалов	102
4.10. Ограничение опциональных логических типов	103
4.10.1. Сокращение логического типа до двух состояний	104
4.10.2. Откат к значению true	104
4.10.3. Логический тип данных с тремя состояниями	105
4.10.4. Реализация протокола RawRepresentable	106
4.10.5. Упражнение	107
4.11. Рекомендации по принудительному извлечению значения	108
4.11.1. Когда принудительное извлечение значения является «приемлемым»	109
4.11.2. Аварийный сбой со стилем	109
4.12. Приручаем неявно извлекаемые опционалы	110
4.12.1. Как распознать неявно извлекаемый опционал	111
4.12.2. Неявно извлекаемые опционалы на практике	111
4.12.3. Упражнение	114
4.13. В заключение	114
Глава 5. Разбираемся с инициализаторами	117
5.1. Правила инициализаторов структуры	117
5.1.1. Пользовательские инициализаторы	118
5.1.2. Странность инициализатора структуры	120
5.1.3. Упражнения	121
5.2. Инициализаторы и подклассы	121
5.2.1. Создание суперкласса настольной игры	122
5.2.2. Инициализаторы BoardGame	123
5.2.3. Создание подкласса	125
5.2.4. Потеря вспомогательных инициализаторов	126
5.2.5. Возвращение инициализаторов суперкласса	127
5.2.6. Упражнение	129
5.3. Минимизация инициализаторов класса	130
5.3.1. Реализация назначенного инициализатора в качестве вспомогательного с использованием ключевого слова override	130
5.3.2. Деление подкласса на подклассы	132
5.3.3. Упражнение	133
5.4. Требуемые инициализаторы	134
5.4.1. Фабричные методы	134
5.4.2. Протоколы	136
5.4.3. Когда классы являются финальными	137
5.4.4. Упражнения	138
5.5. В заключение	138
Глава 6. Непринужденная обработка ошибок	142
6.1. Ошибки в Swift	143
6.1.1. Протокол Error	144
6.1.2. Генерация ошибок	144
6.1.3. Swift не показывает ошибки	145
6.1.4. Сохранение среды в предсказуемом состоянии	147

6.1.5. Упражнения	150
6.2. Распространение ошибок и перехват	151
6.2.1. Распространение ошибок	151
6.2.2. Добавление технических деталей для устранения неполадок	154
6.2.3. Централизация обработки ошибок	158
6.2.4. Упражнения	160
6.3. Создание симпатичных API	161
6.3.1. Сбор достоверных данных в типе	161
6.3.2. Ключевое слово <code>try?</code>	163
6.3.3. Ключевое слово <code>try!</code>	164
6.3.4. Возвращение опционалов	164
6.3.5. Упражнение	165
6.4. В заключение	165
Глава 7. Обобщения	168
7.1. Преимущества обобщений	169
7.1.1. Создание обобщенной функции	170
7.1.2. Рассмотрение обобщений	172
7.1.3. Упражнение	174
7.2. Ограничение обобщений	174
7.2.1. Нужна функция ограничения	175
7.2.2. Протоколы <code>Equatable</code> и <code>Comparable</code>	176
7.2.3. Ограничивать значит специализировать	177
7.2.4. Реализация протокола <code>Comparable</code>	178
7.2.5. Ограничение в сравнении с гибкостью	179
7.3. Ряд ограничений	179
7.3.1. Протокол <code>Hashable</code>	180
7.3.2. Комбинируем ограничения	181
7.3.3. Упражнения	182
7.4. Создание обобщенного типа	182
7.4.1. Желание совместить два типа, соответствующих протоколу <code>Hashable</code>	183
7.4.2. Создание типа <code>Pair</code>	183
7.4.3. Несколько обобщений	184
7.4.4. Соответствие протоколу <code>Hashable</code>	185
7.4.5. Упражнение	187
7.5. Обобщения и подтипы	187
7.5.1. Подтипы и инвариантность	188
7.5.2. Инвариантность в Swift	189
7.5.3. Универсальные типы Swift получают особые привилегии	190
7.6. В заключение	191
Глава 8. Становимся профессионалами в протольно-ориентированном программировании	194
8.1. Время выполнения в сравнении со временем компиляции	195
8.1.1. Создание протокола	195
8.1.2. Обобщения в сравнении с протоколами	196

8.1.3. Находим компромисс.	197
8.1.4. Переход ко времени выполнения.	198
8.1.5. Выбор между временем компиляции и временем выполнения.	199
8.1.6. Когда обобщение – лучший вариант	200
8.1.7. Упражнения	201
8.2. Зачем нужны ассоциированные типы	202
8.2.1. Недостатки протоколов	203
8.2.2. Попытка превратить все в протокол.	204
8.2.3. Разработка обобщенного протокола	205
8.2.4. Моделирование протокола с ассоциированными типами.	206
8.2.5. Реализация протокола с ассоциированными типами.	207
8.2.6. Протоколы с ассоциированными типами в стандартной библиотеке.	209
8.2.7. Другие случаи использования ассоциированных типов	209
8.2.8. Упражнение	210
8.3. Передача протокола с ассоциированными типами	211
8.3.1. Использование оператора where с ассоциированными типами	212
8.3.2. Типы, ограничивающие ассоциированные типы	213
8.3.3. Очистка API и наследование протокола.	215
8.3.4. Упражнения	216
8.4. В заключение.	217
Глава 9. Итераторы, последовательности и коллекции	221
9.1. Итерация.	222
9.1.1. Циклы и метод makeIterator	222
9.1.2. IteratorProtocol	223
9.1.3. Протокол Sequence.	224
9.1.4. Посмотрим поближе.	224
9.2. Сила Sequence.	226
9.2.1. Метод filter	226
9.2.2. Метод forEach.	226
9.2.3. Метод enumerated	227
9.2.4. Ленивая итерация	228
9.2.5. Метод reduce	229
9.2.6. Метод reduce into	230
9.2.7. Метод zip.	232
9.2.8. Упражнения	233
9.3. Создание обобщенной структуры данных с помощью Sequence	233
9.3.1. Bag в действии.	233
9.3.2. Создаем BagIterator	236
9.3.3. Реализация AnyIterator	238
9.3.4. Реализация ExpressibleByArrayLiteral.	239
9.3.5. Упражнение	240
9.4. Протокол Collection	241
9.4.1. Ландшафт Collection.	242
9.4.2. MutableCollection	242

9.4.3. RangeReplaceableCollection	244
9.4.4. BidirectionalCollection	245
9.4.5. RandomAccessCollection	245
9.5. Создание коллекции	246
9.5.1. Создание плана поездки	247
9.5.2. Реализация Collection	248
9.5.3. Пользовательские сабскрипты	249
9.5.4. ExpressibleByDictionaryLiteral	250
9.5.5. Упражнение	251
9.6. В заключение	252
Глава 10. map, flatMap и compactMap	255
10.1. Знакомство с map	256
10.1.1. Создание конвейера с помощью метода map	258
10.1.2. Использование метода map для словарей	260
10.1.3. Упражнения	261
10.2. Последовательности	262
10.2.1. Упражнение	263
10.3. Использование метода map для опционалов	263
10.3.1. Когда использовать метод map с опционалами	264
10.3.2. Создание обложки	265
10.3.3. Более короткий вариант нотации	267
10.3.4. Упражнение	269
10.4. map – это абстракция	269
10.5. Овладеваем методом flatMap	270
10.5.1. В чем преимущества flatMap?	270
10.5.2. Когда метод map не подходит	271
10.5.3. Борьба с пирамидой гибели	273
10.5.4. Использование метода flatMap с опционалом	275
10.6. flatMap и коллекции	279
10.6.1. flatMap и строки	280
10.6.2. Сочетание flatMap и map	281
10.6.3. compactMap	282
10.6.4. Вложенность или цепочки	283
10.6.5. Упражнения	285
10.7. В заключение	285
Глава 11. Асинхронная обработка ошибок с помощью типа Result	290
11.1. Зачем использовать тип Result?	291
11.1.1. Как раздобыть Result	292
11.1.2. Result похож на Optional, но с изюминкой	293
11.1.3. Преимущества Result	294
11.1.4. Создание API с использованием типа Result	295
11.1.5. Из Cocoa Touch в Result	297
11.2. Распространение Result	299
11.2.1. Создание псевдонимов типов	299

11.2.2. Функция search	300
11.3. Преобразование значений внутри Result	302
11.3.1. Упражнение	304
11.3.2. Использование метода flatMap для типа Result	304
11.3.3. Упражнения	306
11.4. Смешивание Result с функциями, генерирующими ошибку	307
11.4.1. От генерации ошибки к типу Result	307
11.4.2. Преобразование генерирующей функции внутри flatMap	309
11.4.3. Пропускаем ошибки через конвейер	310
11.4.4. Подходя к концу	312
11.4.5. Упражнение	312
11.5. Несколько ошибок внутри Result	313
11.5.1. Знакомство с AnyError	313
11.6. Невообразимый провал и Result	316
11.6.1. Когда протокол определяет Result	316
11.7. В заключение	319
Глава 12. Расширения протоколов	325
12.1. Наследование классов в сравнении с наследованием протоколов	326
12.1.1. Моделирование данных по горизонтали, а не по вертикали	327
12.1.2. Создание расширения протокола	328
12.1.3. Несколько расширений	329
12.2. Наследование в сравнении с композицией	330
12.2.1. Протокол Mailer	330
12.2.2. Наследование протокола	331
12.2.3. Композиционный подход	332
12.2.4. Высвобождаем энергию пересечения	334
12.2.5. Упражнение	336
12.3. Переопределение приоритетов	336
12.3.1. Переопределение реализации по умолчанию	336
12.3.2. Переопределение и наследование протоколов	337
12.3.3. Упражнение	338
12.4. Расширение в двух направлениях	339
12.4.1. Выбор расширений	339
12.4.2. Упражнение	341
12.5. Расширение с использованием ассоциированных типов	341
12.5.1. Специализированное расширение	343
12.5.2. Недостаток расширения	344
12.6. Расширение с конкретными ограничениями	344
12.7. Расширение протокола Sequence	346
12.7.1. Заглянем внутрь метода filter	346
12.7.2. Создание метода take (while ;)	348
12.7.3. Создание метода Inspect	350
12.7.4. Упражнение	351
12.8. В заключение	352

Глава 13. Шаблоны Swift	354
13.1. Внедрение зависимости	355
13.1.1. Замена реализации	355
13.1.2. Передача пользовательской версии Session	356
13.1.3. Ограничение ассоциированного типа	357
13.1.4. Замена реализации	358
13.1.5. Модульное тестирование и мокирование с использованием ассоциированных типов	360
13.1.6. Использование типа Result	362
13.1.7. Упражнение	363
13.2. Условное соответствие	364
13.2.1. Бесплатная функциональность	364
13.2.2. Условное соответствие для ассоциированных типов	365
13.2.3. Делаем так, чтобы Array условно соответствовал пользовательскому протоколу	366
13.2.4. Условное соответствие и обобщения	367
13.2.5. Условное соответствие для типов	368
13.2.6. Упражнение	372
13.3. Что делать с недостатками	372
13.3.1. Как избежать использования перечисления	374
13.3.2. Стирание типов	375
13.3.3. Упражнение	379
13.4. Альтернатива протоколам	380
13.4.1. Чем мощнее, тем хуже код	380
13.4.2. Создание обобщенной структуры	382
13.4.3. Эмпирические правила полиморфизма	383
13.5. В заключение	384
Глава 14. Написание качественного кода на языке Swift	387
14.1. Документация по API	388
14.1.1. Как работает Quick Help	388
14.1.2. Добавление выносок в Quick Help	389
14.1.3. Документация в качестве HTML с использованием Jazzy	391
14.2. Комментарии	392
14.2.1. Объясняем причину	392
14.2.2. Объясняем только непонятные элементы	393
14.2.3. Код несет истину	393
14.2.4. Комментарии – не повязка для неудачных имен	393
14.2.5. Зомби-код	394
14.3. Правила стиля	394
14.3.1. Согласованность – это ключ	395
14.3.2. Обеспечение соблюдения правил с помощью линтера	395
14.3.3. Установка SwiftLint	396
14.3.4. Настройка SwiftLint	397
14.3.5. Временное отключение правил SwiftLint	398
14.3.6. Автозамена правил SwiftLint	399

14.3.7. Синхронизация SwiftLint.	399
14.4. Избавляемся от менеджеров.	400
14.4.1. Ценность менеджеров.	400
14.4.2. Атака на менеджеров.	401
14.4.3. Прокладываем дорогу для обобщений.	401
14.5. Именованые абстракции.	402
14.5.1. Обобщенное или конкретное.	403
14.5.2. Хорошие имена не меняются.	403
14.5.3. Именованые обобщений.	404
14.6. Контрольный список.	404
14.7. В заключение.	405
Глава 15. Что дальше?	407
15.1. Создавайте фреймворки, предназначенные для Linux.	407
15.2. Изучите диспетчер пакетов Swift.	407
15.3. Изучайте фреймворки.	408
15.4. Бросьте себе вызов.	408
15.4.1. Присоединяйтесь к эволюции Swift.	409
15.4.2. Заключительные слова.	409
Предметный указатель.	410

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 облике, 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 пытаюсь 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'т 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 Райтера, 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 протоколно-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 «Bag». 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 протоколно-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 подготавливает 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 протоколно-ориентированном программировании

Протоколы – напоминающие классы типов в Haskell или типажи в Rust – святой Грааль Swift. Поскольку Swift можно считать протоколно-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 Сосоа 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 сопровождаются многими списками, выделяя важные понятия.

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

e-Univers.ru